Comparison of Embedding Methods on SCOTUS Cases and Empirical Analysis of Phase Transitions in Node2Vec Hyperparameters

By
Ethan Koch

Senior Honors Thesis
Department of Statistics and Operations Research
University of North Carolina at Chapel Hill

April 2, 2018

Approved: 
Prof. Shankar Bhamidi, Thesis Advisor
Prof. Quoc Tran-Dinh, Reader
Abstract

The first goal of this paper is to analyze the information retained in different representations of the same data. In our case, the Supreme Court of the United States (SCOTUS) releases an opinion for cases which they hear. Each case contains the citations of previous cases as well as its opinion text. Thus, we have text data as well as citation data for each case, upon which we run different embedding models. Secondly, it is a comparison the performance of embedding models, specifically Doc2Vec and Node2Vec, that are extensions of the same group of models, Word2Vec. Since its inception, Word2Vec has become the basis for several other embedding models, including Doc2Vec and Node2Vec. Lastly, it is an analysis of the performance of Node2Vec as its parameters change. We find a rapid transition as the parameter increases, going from near-useless to performing near-perfectly within a short window of values. Instead of running this analysis on a real-world data set where most information is unknown, such as the SCOTUS cases, we run many simulations of networks using the Stochastic Block Model.

1 SCOTUS Corpus

The Supreme Court of the United States releases an opinion for cases that they decide to hear. Our data set from CourtListener\(^1\) has 27,885 of these. Thus, in the SCOTUS network, we have 27,885 nodes (corresponding to each case) and 234,209 edges, where an edge exists if one case cites another\(^2\). It is worth noting that we consider the SCOTUS network as being undirected. My hope is that this will remove the time series aspect of the data that is difficult to account for. Additionally, this simplifies the implementation of previously discussed algorithms. This citation graph is precisely the data set on which we ran Node2Vec to generate embeddings for each case.

Each case (node) has text associated with it as well. We did not attempt to account for differences in author of the case, though we do not believe this would affect the embeddings greatly,

\(^1\)www.courtlistener.com: a 501(c)(3) non-profit with a goal of gathering legal documents to offer free access to researchers

\(^2\)Data downloaded using: github.com/idc9/law-net/
since the cases are generally quite thorough and discuss the topic of the case at length. We did no further preprocessing, though perhaps very slight improvements may be made using methods such as tokenization due to language variations over time. In fact, attempting to account for language variations over time would be an interesting extension. However, it is outside the scope of this project. For the purposes of this project we used the raw text documents and it is on these that we ran Doc2Vec to generate embeddings for each case.

2 Word2Vec as the Basis for Doc2Vec and Node2Vec

Word2Vec, though referred to as a model, is actually a group of models with a goal of creating vector representations for words in a corpus of text. Its extension of creating vector representations for groups of words, or documents, is Doc2Vec. Alternatively, Node2Vec creates vector representations for nodes in a network. Because Word2Vec is the basis for Doc2Vec and Node2Vec, we will start with a thorough explanation of it. After this, Doc2Vec and Node2Vec can be explained by making straightforward alterations or extensions to Word2Vec.

2.1 Introduction to Word2Vec

The Word2Vec algorithm was originally introduced by Mikolov et al [1]. The algorithm has been a catalyst for a great deal of interest and research in embedding methods from researchers in Statistics, Applied Mathematics, and Natural Language Processing\(^3\) due to its improvement over the previous state-of-the-art methods in retaining the meaning of each word in a corpus. Benefits of the Word2Vec algorithm include: the dimensionality of embeddings is significantly less than the number of documents \(D\) for any reasonably sized corpus, the embeddings are dense as opposed to sparse in the case of TF-IDF, and the embeddings can be used in clustering or classification models\(^4\) [6], [7]. It is also important to note that while Word2Vec has many different architectural choices

\(^3\)A few embedding methods from recent years: a generative version of Word2Vec [2], Intelligent Word Embedding (IWE) [3], BioVec and ProtVec [4]

\(^4\)An example of Word2Vec used in a classification model can be found in [5]
and variations, the ones focused on in the theory portion of this paper will be those that offer the core intuition underlying the algorithm.

The goal of the Word2Vec algorithm is to generate an embedding for every word in a corpus that retains the meaning of that word in relation to every other word. It is based on the distributional hypothesis, which states that words appearing in similar contexts tend to have similar meanings [8]. The reason the meaning of a given word is only retained in relation to other words is that any given direction in a word’s embedding, $v_j \in \mathbb{R}^{s \times 1}$, the direction $s_k$ itself is most likely uninterpretable in and of itself [9]. Additionally, the norm of the vector is generally regarded as useless, so cosine similarity is used when comparing two vectors because the direction is most of interest in comparison. In fact, the norm of the vector has been shown to reflect the frequency of the corresponding word instead of its meaning [10].

2.2 Skip-gram (SG)

The skip-gram model creates word embeddings by having a goal of taking a word and predicting the words around it during training. The skip-gram model was first introduced by Mikolov et al [11]. In an effort to illustrate how this model works, let us use the following text as an example document:

*Happy families are all alike; every unhappy family is unhappy in its own way*

This is the first line of *Anna Karenina* by Leo Tolstoy. Given a document of text, we generate the algorithm’s input-output pairs by first specifying $c$, the size of the context or window. The set of observations in the skip-gram model is then:

$$
\{(w_i, w_{i-j}) \mid 0 \leq i \leq W, \ -c + i \leq j \leq c + i, \ j \neq 0\}
$$

where $W$ is the number of words in our corpus. Similarly, we define $V$ as the number of *unique* words in our corpus. Now, if we specify $c$ to be 2 in our example, then some of the input-output pairs would be:
(Happy, families), (unhappy, family),
(family, is), (families, are)

With this in mind, we can think about the model embedding words that appear in similar contexts near to each other. The third and fourth pairs should push the model toward embedding is and are near each other because they both appear within the context of some form of the word family. Mathematically, our goal is to maximize [12]:

\[
p(w_o|w_i; \theta) = \frac{e^{(w_o^T \theta)}}{\sum_{i=0}^{V} e^{(w_i^T \theta)}}
\]

for a given word, \(w_o\). However, in terms of the entirety of our corpus, we wish to find the \(\theta\) which maximizes:

\[
L(\theta) = \frac{1}{V} \sum_{i=0}^{V} \sum_{o=-c, o \neq 0}^{c} \log p(w_o|w_i; \theta)
\]

In this setting, \(w_o\) represents a vector of zeros with length \(V\), where the \(o^{th}\) entry is 1. For sake of simplicity, we will also refer to \(w_o\) as the word corresponding to the \(o^{th}\) entry. Each \(w_o\) is of the following form:

\[
w_o = \begin{bmatrix}
0 \\
\vdots \\
1 \\
\vdots \\
0
\end{bmatrix}
\]

This is called one-hot encoding and serves to isolate only the row in the matrix of learned parameters, \(\theta\), which correspond to that word, \(w_o\). This is the method by which input words are isolated to be used in the prediction step of the algorithm. However, in practice one typically uses key/value pairs for computational efficiency.

The actual embedding generated, \(v_o\), which corresponds to word \(w_o\), is exactly the row that is
isolated by multiplying our one-hot encoded input vector with the weight matrix:

\[ v_o^T = w_o^T \theta \]

remembering that \( w_o \in \mathbb{R}^{V \times 1}, \theta \in \mathbb{R}^{V \times s}, \) and \( v_o \in \mathbb{R}^{s \times 1}. \)

This is quite different than the traditional settings where the quantities of interest are the output of a model, either predicted values or probabilities. Here we are interested in the parameters of the model, which later may be used in a more traditional statistical model.

### 2.3 Continuous Bag of Words (CBOW)

The Continuous Bag-of-Words model (CBOW) can be thought of as the reverse of the skip-gram, though it achieves the same end goal of creating embeddings for the words in a corpus. The CBOW model uses a word’s context in order to predict that word [12]. In the CBOW model, the input-output pairs are generated as follows:

\[
\{ ((w_{o-c}, \ldots, w_{o+c}), w_o) \mid 0 \leq o \leq W \}
\]

In the context of our *Anna Karenina* example, with \( c = 2 \), this corresponds to a few input-output pairs being:

\[
((\text{Happy, are}), \text{families}), ((\text{all, every}), \text{alike}), ((\text{every, family}), \text{unhappy}), ((\text{its, way}), \text{own})
\]

One may notice that the shown input-output pairs are not of the same dimension in CBOW, though the matrix multiplications in training will require this to be true. So, we define a function \( g : \mathbb{R}^{V \times c-1} \rightarrow \mathbb{R}^{V \times 1} \) to be an element-wise averaging function (one can also define it as a concatenating function) so that on a word-level the goal becomes to maximizes [12]:

\[
p(w_o | w_{o-c}, \ldots, w_{o+c}; \theta) = \frac{e^{g(w_{o-c}^T \theta, \ldots, w_{o+c}^T \theta)}}{\sum_{i=0}^{V} e^{w_i^T \theta}}
\]
The CBOW model is the one we chose to run on the SCOTUS corpus for performance reasons. Which architecture of the many provides the best results is still an open question, though some research has been done [13].

In order to give a firm understanding of the model we have presented CBOW as using the words on either side of a given word, \( w_o \), to predict \( w_o \) and thereby generate embeddings. However, one can alter CBOW, as well as other word embeddings models, to define \( w_o \) as the word directly after the context. The input-output pairs are then generated in the following way:

\[
\{(w_{o-c}, w_{o-c+1}, \ldots, w_{o-1}), \ w_o) \mid 0 \leq o \leq W\}
\]

In fact, one can even choose \( w_o \) to be the word directly before the context. However, it is not known whether these variations provide better or worse results.

These variations notwithstanding, there are some architectural options to the basic SG and CBOW models we have not presented. Many of these options have the effect of making the model less computationally expensive to train and are therefore used in practice by software. Some common options include: negative sampling, hierarchical softmax, and stochastic gradient descent. Being one of the most important model architecture options, we will provide a brief overview of stochastic gradient descent in the following section.

### 2.4 Training the Model: Stochastic Gradient Descent

In Word2Vec, Doc2Vec, and Node2Vec, Stochastic Gradient Descent (SGD) is the optimization method we used to tune the parameters of the model. SGD is a variation of Cauchy’s Gradient Descent (GD) [14]. GD is defined by the following steps:

1. Choose initial parameters, typically randomly selected from a probability distribu-
tation:
\[
\theta = \begin{bmatrix}
\theta_{0,0} & \theta_{0,1} & \ldots & \theta_{0,s} \\
\theta_{1,0} & \theta_{1,1} & \ldots & \theta_{1,s} \\
\vdots & \vdots & \ddots & \vdots \\
\theta_{V,0} & \theta_{V,1} & \ldots & \theta_{V,s}
\end{bmatrix}, \; \theta_{i,j} \text{ chosen from } \Theta
\]

where \( \Theta \) is some probability distribution, often \( \mathcal{U}[0, 1] \) or \( \mathcal{N}(0, 1) \). In the context of Word2Vec, \( s \) is the embedding size of the word vectors, chosen beforehand, and \( V \) is the number of unique words in the corpus.

2. Calculate the gradient of the loss function over the entirety of the training data set. The parameters, \( \theta \), become itself minus the calculated gradient with a multiplied learning rate.

\[
\theta = \theta - \alpha \nabla_\theta L(\theta)
\]

3. Repeat step 2 until some convergence rule is achieved. This is typically when a set number of iterations is reached or the gradient becomes sufficiently small.

Stochastic Gradient Descent’s precursor is a stochastic approximation method, but is almost exactly Gradient Descent, with a small change for mostly computational purposes [15], [16]. It is very expensive to calculate \( \nabla_\theta L(\theta) \) on the entirety of the training data set, so instead we use Stochastic Gradient Descent. The difference is that step 2 is not done for every example in the training set. Instead, a subset of examples are randomly chosen from the training set and these examples are used to calculate an approximation of the gradient of the loss function. SGD will often converge to a global minimum, and almost always converges to a local minimum, depending on the conditions [17], [18]. SGD also requires more iterations than Gradient Descent for convergence, due to its use of a subset of examples.

Lastly, \( \alpha \) is set either to a fixed value or a decreasing range of values. This is effectively allowing large changes in the parameters toward the beginning of the iterations and decreasing the change of parameters as training continues. Due to the likelihood that the parameters must change a great deal from the initial setting to move near the global minimum in the parameter space, this
approach makes sense and works in practice [19].

3  Doc2Vec

Word2Vec generates embeddings for every unique word in a corpus. However, this is not directly useful if one wishes to compare, say, the abstracts of different academic articles. For this reason Doc2Vec\footnote{We used Gensim’s implementation of Doc2Vec: radimrehurek.com/gensim/models/doc2vec.html} was introduced by Mikolov et al [1]. Doc2Vec generates embeddings for each document, $d_i \in \mathbb{R}^{s \times 1}$. The definition of a document is left completely up to the researcher. One convenient aspect of Doc2Vec is that the documents can be of variable length. In our case, we consider each of the case opinions from SCOTUS to be a different document. Doc2Vec is almost identical to Word2Vec, with a few modifications. In fact, word embeddings are also generated as part of training one type of Doc2Vec model. Doc2Vec is also a group of models, with certain architectural choices left to the researcher. Two different Doc2Vec models will be discussed in this paper.

3.1  Paragraph Vector - Distributed Bag of Words (PV-DBOW)

PV-DBOW is most similar to the Word2Vec Skip-Gram model. The vectors that correspond to a given document here are learned by a process of predicting the words in $d_i$. The input-output pairs are generated as follows:

$$\{(d_i, w_{i,j}) \mid 0 \leq i \leq D, 0 \leq j \leq W_i\}$$

where $D$ is the number of documents, $W_i$ is the length of the sequence of words corresponding to document $d_i$, $w_{i,j}$ is the $j^{th}$ word in the sequence of words corresponding to document $d_i$, and $c$ is the size of the context or window.

To further illustrate this let us extend the Anna Karenina example we have begun. Consider the corpus to be the first line in Anna Karenina by Leo Tolstoy and the last line in The Great Gatsby.
by F. Scott Fitzgerald:

(\textit{So we beat on, boats against the current, borne back ceaselessly into the past, happy families are all alike; every unhappy family is unhappy in its own way})

where, giving the document IDs 0 and 1 to \textit{Anna Karenina} and \textit{The Great Gatsby}, respectively, would result in a few of the input-output pairs, with $c = 2$, being:

$$
(0, \text{boats}), (0, \text{ceaselessly}), (0, \text{borne}) \\
(1, \text{Happy}), (1, \text{unhappy}), (1, \text{families})
$$

while noting that the document IDs and words would be replaced with one-hot encoded vectors in the actual algorithm.

The PV-DBOW method, according the original paper, only needs to “store the softmax weights as opposed to both softmax weights and word vectors” as is necessary in the model we will discuss next, PV-DM, resulting in PV-DBOW to utilize less memory in training [9]. The final output of the model is a vector for each document in the corpus. Ideally, documents that contain similar sequences of words will be mapped near each other in the resulting vector space. In these algorithms, cosine similarity is used as the measure of how “similar” two vectors are. Henceforth, cosine similarity will be what we mean when referring two vectors being similar or near each other.

Because this is an unsupervised algorithm, one must do further analysis to identify how well the algorithm has performed on the corpus in question. Comparisons of well-understood documents can be a start. For example, if one’s corpus was Wikipedia articles, it would help to check that the embeddings for WWI and WWII were closer to each other than, say, WWI and epistemology. However, while some work has been done in how to evaluate embedding models like Word2Vec and Doc2Vec, there is not yet a widely agreed upon methodology or metric when pre-existing labels are not present [20]. However, it has been shown that PV-DBOW generally works better in practice, so this is the algorithm we chose to run on the SCOTUS corpus [21].

3.2 Paragraph Vector - Distributed Memory (DM)

PV-DM is most similar to the Word2Vec CBOW model. Recall that in CBOW the word vectors are learned by the process of predicting a specific word given its context. The extension required
to make this a document-embedding algorithm is quite straightforward. All that is required is to append the document ID to the input as if it were an additional context word. Thus, the input-output pairs are generated as follows:

$$\left\{ \left( \left( d_i, w_{i,o-c}, \ldots, w_{i,o+c} \right), w_o \right) \mid \frac{c}{2} \leq o \leq W_i - \frac{c}{2}, \ c \neq 0 \right\}$$

In the context of our Anna Karenina and The Great Gatsby example, this would result in a few of the input-output pairs being:

$$((0, \text{borne, ceaselessly}), \text{back}), ((0, \text{on, against}), \text{boats}), ((1, \text{Happy, are}), \text{families}), ((1, \text{every, family}), \text{unhappy})$$

with $c = 2$ and the words and IDs corresponding to one-hot encoded vectors, of course.

Although the extension to Doc2Vec is straightforward, it turns out to be a very powerful algorithm as evidenced by [21]. The PV-DM algorithm also results in word embeddings due to the words being used as inputs along with the document ID. This does cause the PV-DM algorithm to be more memory-intensive than its counterpart, but avoids training a Word2Vec model separately, if both are intended to be used.

4 Node2Vec

Another extension of Word2Vec is Node2Vec\(^6\), an embedding algorithm for graph or network data sets [22]. Whereas Word2Vec and Doc2Vec are algorithms that were inspired by creating embeddings given sequences of words, the actual algorithms are agnostic to the word itself. The only importance a word provides to the algorithm is the vector to which it is associated. Thus, the Word2Vec and Doc2Vec models could just as easily use sequences of numbers, letters, or any combination of the two to output embeddings. Node2Vec leverages this by generating a sequence of node IDs which will later be used in the Word2Vec algorithm in order to generate embeddings for the nodes in a graph. The key insight of this algorithm is Random Walk by which these sequences

\(^6\)We used a slightly modified version of the original author’s Node2Vec implementation found at: github.com/aditya-grover/node2vec
of node IDs are generated. At the end of the algorithm, similar nodes should be near each other.

4.1 Graph or Network Object

A graph or network, $G$, is defined as an ordered pair of its nodes, $V$, and edges which connect the nodes, $E$, so that $G = (V, E)$. We only consider undirected graphs in this paper for simplicity. To make this and the following ideas concrete, let us consider a simple graph of cities, where an undirected edge exists if a person can drive from one city to the other on a single highway. Our nodes and edges are:

$$V = \{ \text{Charlotte, Atlanta, Nashville, Birmingham} \}$$

$$E = \{ (\text{Charlotte, Atlanta}), (\text{Atlanta, Birmingham}), (\text{Nashville, Birmingham}) \}$$

The adjacency matrix, $A \in \mathbb{R}^{|V| \times |V|}$, is used to mathematically represent this graph structure. The entry, $A_{i,j}$, is 1 if there exists an edge between node $i$ and $j$; otherwise, the entry is 0. One can replace the 1 with the weight of the edge connecting the two nodes, though our SCOTUS citation network does not have weights, so here too we use 1 for the existence of an edge. Note that the adjacency matrix is symmetric about its diagonal entries. The adjacency matrix for our example looks like:

$$A = \begin{bmatrix}
0 & 1 & 0 & 0 \\
1 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 \\
0 & 1 & 1 & 0
\end{bmatrix}$$

The homophily hypothesis states that connected nodes are similar and therefore should be near each other after embeddings are generated [23]. In our small example, this means that after running an embedding algorithm we would hope to find that: 1) Charlotte is similar to Atlanta, but not to Birmingham and Nashville, 2) Atlanta is similar to Charlotte and Birmingham, but not to Nashville, 3) Nashville is similar to Birmingham, but not to Atlanta and Charlotte, and 4) Birmingham is similar to Atlanta and Nashville, but not to Charlotte.

The counterpart of homophily is structural equivalence. The hypothesis of structural
equivalence is that similar nodes play similar roles in the network, though may not connect to the same nodes at all [24]. In context, of our example, it would mean we would hope to see Atlanta and Birmingham similar to each other, because they are both the link connecting two different cities. Additionally, Charlotte and Nashville should be similar, because they are both connected only to the linking nodes and no others.

If attempting to generate a sequence of nodes in a graph, there are trade-offs that a researcher must make when choosing between these two sampling approaches. Breadth-First Search (BFS) approaches consider all nodes immediately connected to the current node when choosing which node to visit next. Alternatively, Depth-First Search (DFS) approaches consider nodes in increasing distance from the sources when choosing the next node. BFS approaches favor the homophily hypothesis, whereas DFS approaches favor the structural equivalence hypothesis. That is, when generating a sequence of nodes, if one believed the homophily hypothesis to better encapsulate the relationships between the graph’s nodes, one would employ the BFS approach to generate a random sequence. This would result in nodes that are connected often being immediately next or near each other in the sequence. In our example, a few random walks favoring this approach may look as follows:

\[(Charlotte, Atlanta, Charlotte, Atlanta, Birmingham)\]
\[(Atlanta, Birmingham, Atlanta, Birmingham, Nashville)\]

On the other hand, if one believed that the relationship between nodes more closely resembled the structural equivalence hypothesis, it would be better to favor the DFS approach, in which the structure of the graph is more easily apparent in the sequence. A few random walks favoring this approach may look as follows:

\[(Charlotte, Atlanta, Birmingham, Nashville, Birmingham)\]
\[(Nashville, Birmingham, Atlanta, Charlotte, Atlanta)\]

The reason that algorithms using graph structures must choose between these approaches is that there is no natural ordering of the data like in text data, where every sentence can be taken as a sequence, or time-series data, where the progression of time provides natural, sequential ordering to the data. Additionally, there is generally no start or end to the graphs like in the given examples of
other data types.

Most often in real-world applications, however, some nodes are better described by homophily while others by structural equivalence. Clearly an exclusively BFS or DFS approach will not accommodate the differences between nodes in such a case. The Random Walk introduced in Node2Vec algorithm addresses precisely this issue.

### 4.2 Random Walk

The random walk of the Node2Vec model, as was previously stated, is the key contribution of this paper. It is the method by which a sequence of nodes is generated. This algorithm creates a single walk by starting a given node, $V_i$, and using a probability distribution over the previously visited node, the current node, and the immediate neighbors of the current node, chooses a “next” node in the sequence. This process is repeated until the walk is of the desired length. In a directed graph setting one must consider in-edges as well as out-edges, but this is not required for our SCOTUS citation network, an undirected graph. A node is considered a neighbor of another node if they are connected by an edge. This is repeated a specific number of times for each node in the graph. Thus, the parameters specific to the random walk algorithm are as follows [22]:

- $p$: Return parameter, a higher value (larger than $\max(q, 1)$) favors a BF approach, $p \in \mathbb{R}_{\geq 0}$
- $q$: In-out parameter, a higher value (larger than 1) favors a DF approach, $q \in \mathbb{R}_{\geq 0}$
- $m$: Number of random walks to generate per node, $m \in \mathbb{N}^+$
- $l$: Number of nodes sampled in each random walk, $m \in \mathbb{N}^+$

When $p = 1$ and $q = 1$, this random walk is exactly the random walk from the DeepWalk Algorithm introduced by Perozzi, Al-Rfou, and Skiena [25]. Also, one may notice that this will result in a matrix, $W \in \mathbb{V}^{m|V| \times l}$, with the rows corresponding to random walks, each of length $l$. Continuing with our previous example, if we choose $m = 1$ and $l = 4$, we may obtain the following...
After the walks are generated, the examples are ready to be input to the Word2Vec algorithm in order to obtain node embeddings. However, let us first take a closer look at how the “next” node is chosen at each step in a given walk.

The probability distribution used to model the next possible nodes is [22]:

\[
p(c_i = x | c_{i-1} = v) = \begin{cases} 
\pi_{v,x}, & \text{if } (v, x) \in E \\
0, & \text{otherwise}
\end{cases}
\]

where the constant, \(Z\) is used to normalize the unnormalized probabilities, \(\pi_{v,x}\), that guide the walk, so that they sum to 1. Now, let \(\pi_{v,x} = \alpha_{p,q}(t,x) \cdot A_{v,x}\), so that these unnormalized probabilities are obtained as follows:

\[
\alpha_{p,q}(t,x) = \begin{cases} 
\frac{1}{p}, & \text{if } d_{t,x} = 0 \\
1, & \text{if } d_{t,x} = 1 \\
\frac{1}{q}, & \text{if } d_{t,x} = 2
\end{cases}
\]

where \(d_{t,x}\) is the shortest path between nodes \(t\) and \(x\) [22].

This value, \(d_{t,x}\), will be exactly 0 for only the node \(t\), exactly 1 if \(x\) is a neighbor of \(t\), and exactly 2 if \(x\) is a neighbor of \(v\) and not \(x\). Thus, it is clear why \(p\) is considered the return parameter, since it dictates the probability with which a node will return to node the walk has just visited. Likewise, \(q\) is considered the in-out parameter because it dictates the probability with which the walk will move further from previously visited nodes, thereby getting “out” of the current node neighborhood, roughly speaking, into new parts of the network [22].

\[\text{We used Gensim’s implementation of Word2Vec: radimrehurek.com/gensim/models/word2vec.html}\]
In practice, it can be too expensive to calculate these probability distributions while generating the random walks, so they are calculated prior to generating walks for each possible scenario. This is more memory-intensive, but significantly decreases the amount of time required to generate the random walks. Without this improvement, the same task would be repeated multiple times, given that each node begins a random walk \( m \) number of times.

Less work has been done with Node2Vec than Word2Vec or Doc2Vec, though it is a promising node embedding algorithm and has many practical applications ranging from Law, to Social Networks, to Medicine, and more. Some of the work that has been done on Word2Vec is how to go about choosing the hyperparameters of the algorithm [26]. This has yet to be done with Node2Vec and the following portion of my paper begins to deal with this issue.

5 Stochastic Block Model

While it is tricky to identify how an algorithm performs as a specific parameter varies on a real, messy data set, one may perform such a task on a synthetic data set where the ground truth is known. For example, identifying how well Node2Vec performs as the walk length increases on the SCOTUS citation network is not helpful because not only are we not aware of the upper bound of the results of the algorithm, but there is also variability in the existence of edges that may or may not be due to the node itself, which we cannot know. Thus, we use the Stochastic Block Model (SBM) to run simulations looking at the performance of the models under certain parameter settings [27].

The SBM is a random graph created by defining a probability matrix, \( B \in \mathbb{R}^{|C|\times|C|} \), where \(|C|\) is the number of communities decided beforehand and the entry, \( B_{i,j} \), is the probability that an edge will exist for two given nodes, one in community \( i \) and one in community \( j \) [27]. In the simplest setting, one defines the probability of connecting to nodes within the same community and the probability of connecting to nodes outside of a node’s community. All information is known about a graph generated from an SBM because one need only define the number of nodes in each
community, and then for every node in a given community the edges will be created according to the specified probability matrix. Note that neither the rows nor the columns in the probability matrix are required to sum to 1, as is required in a probability distribution. This allows for a great deal of flexibility when generating many SBM samples in a simulation.

Due to the existence of a few parameters in the SBM itself, we chose to hold a few constant (e.g. number of nodes in each community, probability of connecting to a node within of one’s community) while varying the parameter corresponding to the probability of connecting to a node outside of a given node’s community as well as a Node2Vec parameter, walk length. This is quite computationally expensive due to the number of simulations required, so while some progress has been made, it is not as complete a picture as we prefer, due to computational restrictions in accessible computer technologies\(^8\).

## 6 Results and Analysis

In this section I describe the results obtained by the implementation of the above methods\(^9\). We generated different embeddings on the SCOTUS data set, compared the clustering labels from the embeddings, tried to identify the difference in information retained by the different embeddings models, created a classification model to identify the embedding model that was most predictive for issue areas, and finally analyzed the change in performance of clustering methods on our Node2Vec embeddings as a Node2Vec parameter increased. The results found in the following sections are, broadly speaking, preliminary and encourage further research on these topics instead of providing a final, definite answer to each question posed.

---

\(^8\)We ran almost all of our models on UNC-Chapel Hill’s Longleaf cluster: help.unc.edu/subject/research-computing/longleaf/

\(^9\)All code from this project can be found at: github.com/ethankoch4/unc_honor_thesis
6.1 Doc2Vec Similarity Plots

After running the Doc2Vec algorithm on the SCOTUS corpus, we became curious about how similarity is affected by time. So, for each document we calculated its similarity to every other document and plotted the similarity over time. In the following plot the blue line corresponds to the date of the document held constant, each green dot corresponds to the similarity score between the document held constant and some other document in the corpus, and the red line is the median similarity score at each year.

As seen in the above plot, every case after about 2006 displays a marked jump in similarity scores between pre-2006 cases and post-2006. The reason for this is not yet clear and further investigation must be done to identify the cause of this jump, but one initial thought is that certain procedural precedents may have changed that are referred to in every post-2006 case. However, even this is not likely, since this has certainly happened in the court’s past and jumps like this are not seen in previous cases. As an example, the plot for this case from 1892 is much more stable over time:
Thus, while this jump seems to contain interesting differences and may merit further analysis, it will require the recruitment of legal experts and further investigation of the cases themselves instead of the algorithms performed on them, which is outside of the scope of this paper. Let’s move on to comparing the Doc2Vec and Node2Vec algorithms run on the SCOTUS corpus.

6.2 Doc2Vec vs. Node2Vec on SCOTUS: Varying Cluster Sizes

The goal in running clustering algorithms on both the Node2Vec and Doc2Vec embeddings is explore the difference in what information is stored in each. Because labels do not exist for every single number of clusters setting, we examined four different plots to obtain an idea of the differences and similarities between the embeddings. For each plot, four different widely-used clustering metrics are shown alongside the mean of those four metrics. The metrics are: V-Measure, Homogeneity, Normalized Mutual Information, and Completeness. One may notice that each of these require labels from clustering and true labels of the nodes. For computational reasons we was unable to run a metric that did not require true labels. However, this would be a next step in the investigation of Doc2Vec and Node2Vec on SCOTUS. So, instead of providing true labels we provided either labels from a different clustering method or from a different embedding method. The scores in the first graph were obtained by scoring the labels obtained from the hierarchical clustering method from each of Node2Vec and Doc2Vec:
The above plot essentially shows the extent to which the hierarchical clustering labels from Node2Vec and those from Doc2Vec agree with each other as the number of clusters increases. The approximately linear increase shows that Node2Vec’s labels and Doc2Vec’s labels agreed more as the number of clusters increases. Though further investigation is required to verify, we take this to mean that initial cluster splits represent very different ideas for Node2Vec and Doc2Vec, but they converge over time. We think this is intuitive, because if the number of labels is exactly 27,885, the number of cases, then the scores will be perfect since each cluster has exactly one element (assuming each cluster would have at least one).

The second plot in this section was obtained in exactly the same way as the first, with hierarchical clustering replaced by KMeans clustering. That is, the plot shows scores obtained by comparing KMeans labels on Node2Vec to KMeans labels on Doc2Vec as the number of clusters increases. The following plot further supports my hypothesis that initial splits at small numbers of clusters represent very different ideas which converge as the number of clusters increases. It also shows an upward linear trend, though it is a more sharp increase than its hierarchical clustering counterpart:
The scores in the following two plots were obtained by comparing the labeling from KMeans and hierarchical clustering. The first plot shows the clustering labels with regard to Node2Vec only. The scores in the plot have a slight downward, linear trend. This means that the KMeans clustering labels on Node2Vec and the hierarchical clustering labels on Node2Vec disagreed slightly more as the number of clusters increased. This means to me that almost all the information found by Node2Vec can be represented in a small number of clusters. However, we would expect this plot to increase again at some point if the number of clusters extended beyond 500. The metrics mostly mirrored each others’ behaviors, relatively speaking:
As was previously stated, the final plot in this section was obtained by comparing the KMeans clustering labels on Doc2Vec and the hierarchical clustering labels on Doc2Vec. This plot shows a trend that is similar to that of the positive portion of an inverse tangent curve. There is a sharp increase at first which levels off as the number of clusters increases. We hypothesize that this means Doc2Vec contains a great deal of nuanced information that require a larger number of clusters to display, relative to Node2Vec. Again, the metrics mostly mirrored each other:

![Doc2Vec Clustering Results](image)

Speaking in non-technical terms, we believe these plots as a whole show that Node2Vec contains a few very important pieces of information about the cases which override the importance of others, while Doc2Vec contains many different pieces of information, all of which are similarly important in distinguishing the cases. We believe that this is what caused the first two plots to be linear upward. Further investigation would be required to confirm or deny this hypothesis. However, let’s now take a look at these embeddings in a situation where labels do exist for each case.

### 6.3 Doc2Vec vs. Node2Vec on SCOTUS vs. Issue Areas

A group of researchers at Washington University Law\(^\text{10}\) took it upon themselves to categorize each SCOTUS case into 14 different “issue areas” including categories like Privacy, First

\(^{10}\)The group can be found at: scdb.wustl.edu/
Amendment, Due Process, and Economic Activity. This provides true labels for the SCOTUS cases against which we can compare our clustering labels. Let us look at how KMeans clustering on Node2Vec vs Doc2Vec performed:

The above plot shows that, surprisingly, clustering on Node2Vec embeddings did a better job of identifying the issue area of a case than clustering on Doc2Vec. Doc2Vec has an advantage over Node2Vec in terms of mass of information in that each case contains a great deal of text describing in detail the proceedings and decision. However, Node2Vec only has the citation network. We believe that this strength of Doc2Vec was also its downfall in this setting. Because legal text uses a lot of the same terminology to describe how the case unfolded, Doc2Vec may not have been able to ignore the similarities and extract the differences well enough to reflect the issue areas in its clusters. There are, of course, many other variables in this, such as cleaning steps, the clustering method, etc. Thus, it is too early to identify the origin of this difference. However, this gives me reason to believe the citation network offers information about the cases that is difficult to extract from the text.

We also performed multinomial logistic classification on Node2Vec, Doc2Vec, and a concatenation of the embeddings as well, to get more of an idea of how well the embeddings provide information about the issue areas. For training we used an 85/15 split for training/test sets, as well as 3-Fold Cross-Validation to identify the parameters of the best performing model. The
results are as follows:

<table>
<thead>
<tr>
<th>Embedding Algorithm</th>
<th>Score on Test Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Doc2Vec</td>
<td>0.676</td>
</tr>
<tr>
<td>Node2Vec</td>
<td>0.591</td>
</tr>
<tr>
<td>Both Concatenated</td>
<td>0.701</td>
</tr>
</tbody>
</table>

Both embeddings offer significant improvement over the baseline, 0.071, which is the score one would expect to obtain with random assignments. Interestingly, though, the logistic model on Doc2Vec performed better than that on Node2Vec. Additionally, using a concatenated version of the embeddings offered some improvement over either exclusively using the Doc2Vec or Node2Vec embeddings. This indicates that there is information in Doc2Vec embeddings not captured in Node2Vec embeddings, as well as vice versa. Thus, the difference in performance of the clustering as previously seen was most likely due to KMeans on Doc2Vec representing information other than the 14 issue areas in its 14 clusters. However, the logistic model was able to, for the most part, distinguish between the 14 issue areas.

Lastly, that the concatenated version performed slightly worse than the logistic classification model on Doc2Vec embeddings is most likely due to Doc2Vec containing most of the information offered by Node2Vec. However, this is only marginally so. The embedding dimension of both Doc2Vec and Node2Vec is 300. Now, only 161 of the largest 300 coefficients in the concatenated-embedding logistic classification model were from Doc2Vec. This may be because Doc2Vec’s features are only slightly more predictive toward issue area classification or because only a portion of Doc2Vec’s features are predictive toward issue area classification but they are strongly predictive. A next step in investigation would be to identify whether a better method of joining the two embedding spaces would provide different results, as concatenation is quite simple and does not take certain factors, such as potential multicollinearity, into account.
6.4 Phase Transitions using SBM

An ever-present question when running the Node2Vec and Doc2Vec algorithms is: what parameters are good enough? As previously stated, some research has been done on this with regard to Doc2Vec and Word2Vec, but not on Node2Vec [26], [21]. In order to answer this question we must begin with a data set wherein the ground truth is known. This is not the case for SCOTUS, so instead we use the Stochastic Block Model (SBM). All labels of nodes in this are set before the edges are generated, so we can score our cluster labels against the true labels in every case.

The change two parameters dictating the existence of edges, in-class connection probability and out-of-class connection probability, offer some insight into how well Node2Vec may perform on a given network. Thus, we hold in-class probability fixed at 0.8 and increase the out-of-class probability from 0.0 to 0.79 with a step size of 0.01. The following plot shows the change in performance of the Node2Vec algorithm on 20 different sampled SBM’s at every point, as the out-of-class probability increases:

![Graph showing performance change](image)

There exist certain theoretic thresholds beyond which the class labels in the 2-community setting should not be detectable\textsuperscript{11} or recoverable\textsuperscript{12}. It is interesting to note that we do not believe

\begin{equation}
\text{Undetectable point: } p_e = \frac{c + (n-1)\sqrt{\pi}}{n c}, \text{ where } c \text{ is average node degree, } n \text{ is number of communities, and } p_e \text{ is the epsilon value beyond which the communities are undetectable, when } c_{\text{in}} > c_{\text{out}} \text{ [28]}
\end{equation}

\begin{equation}
\text{Communities are impossible to recover if: } \frac{\alpha \beta}{2} - \sqrt{\alpha \beta} < 1, \text{ where } \alpha = p \cdot \frac{n}{\log(n)} \text{ and } \beta = q \cdot \frac{n}{\log(n)} \text{ where } p \text{ is the}
\end{equation}

\textsuperscript{11}Undetectable point: } p_e = \frac{c + (n-1)\sqrt{\pi}}{n c}, \text{ where } c \text{ is average node degree, } n \text{ is number of communities, and } p_e \text{ is the epsilon value beyond which the communities are undetectable, when } c_{\text{in}} > c_{\text{out}} \text{ [28]}

\textsuperscript{12}Communities are impossible to recover if: } \frac{\alpha \beta}{2} - \sqrt{\alpha \beta} < 1, \text{ where } \alpha = p \cdot \frac{n}{\log(n)} \text{ and } \beta = q \cdot \frac{n}{\log(n)} \text{ where } p \text{ is the}
these theoretic thresholds are always correct in practice, since we often were able to detect or recover the communities in settings the theoretic thresholds indicate should have been impossible. Thus, we also include an empirical threshold, beyond which the thresholds of empirical success have not been obtained. We say a given sample’s SBM clustering is successful if 75% of the labels were correct. At a given setting, the algorithm was considered to be empirically successful if 75% of the iterations at that setting were successful. The exact location of the empirical threshold in any given setting is only somewhat of interest. The question of interest is really how the location of the threshold changes as one of the Node2Vec parameters changes. So, the simulations that produced the above plot were repeated for increasing walk lengths, in order to identify how the Node2Vec parameters change the ability of the researcher to recover the communities, using the empirical thresholds:

As is shown in the above plots, it seems to be the case that phase transitions exist in the parameter settings. There are specific parameter values on one side of which the algorithm can do no better than random guessing and at which performs rapidly goes to almost exact recovery. Actual data sets may not experience such sharp changes due to data quality issues, computational complexity issues, and the reality that even for a real data set with 2-communities, the probability

\[ q \] is the probability to connecting to a node outside of one’s community, and \( n \) is the number of nodes, when \( \alpha > \beta \). [29]
that a node connects to a node outside of its class may not be constant as it is in the SBM. This would be another natural extension of this project, since such SBM variations have been introduced [30]. However, the above plots are the beginning of an interesting insight into how the parameters of the Node2Vec model, and likely other embedding algorithms, change the ability to recover the true labels of nodes.

7 Conclusion

The work of this paper focused on the SCOTUS data set, but we hope that similar projects will be undertaken on other data sources. While unique, there exist many data sets that contain both text and network structures. Because of this, we expect to see further research in areas pertaining to the Word2Vec, Doc2Vec, and Node2Vec models. Likewise, the embedding methods discussed have industry applications in next-word suggestion, machine translation, and sentiment classification, among others. We suggest the usefulness of Node2Vec and Doc2Vec embeddings in finding alike court documents, classifying a case by issue area, and comparing the similarity of cases over time. Furthermore, the analysis of Node2Vec’s parameters using the SBM is necessary groundwork in aiding researchers and practitioners wishing to understand how to efficiently decide upon hyperparameters for their models.

---

13 A proposal for a Word2Vec application in the retail industry: www.deeplearningtrack.com/single-post/2017/08/15/Neural-networks-for-retail-industry
References


