Western Kentucky University

TopSCHOLAR®

Mahurin Honors College Capstone Experience/Thesis Projects

Mahurin Honors College

2020

A Description of a Humans Knowledge using Artificial Intelligence

DJ Price
Western Kentucky University, donald.price077@topper.wku.edu

Follow this and additional works at: https://digitalcommons.wku.edu/stu_hon_theses

Part of the Artificial Intelligence and Robotics Commons, Linguistics Commons, Mathematics Commons, and the Other Computer Sciences Commons

Recommended Citation
Price, DJ, "A Description of a Humans Knowledge using Artificial Intelligence" (2020). *Mahurin Honors College Capstone Experience/Thesis Projects.* Paper 886.
https://digitalcommons.wku.edu/stu_hon_theses/886

This Thesis is brought to you for free and open access by TopSCHOLAR®. It has been accepted for inclusion in Mahurin Honors College Capstone Experience/Thesis Projects by an authorized administrator of TopSCHOLAR®. For more information, please contact topscholar@wku.edu.

A DESCRIPTION OF A HUMANS KNOWLEDGE USING ARTIFICIAL INTELLIGENCE

A Capstone Experience/Thesis Project Presented in Partial Fulfillment

of the Requirements for the Degree Bachelor of Arts

with Mahurin Honors College Graduate Distinction

at Western Kentucky University

By

D.J. Price

August 2020

****

CE/T Committee:

Dr. Uta Ziegler, Chair

Dr. Claus Ernst

Dr. Jannai Shields

Copyright by

D.J. Price

2020

ABSTRACT

There currently does not exist a way to easily view the relationships between a collection of written items (e.g. sports articles, diary entries, research papers). In recent years, novel machine learning methods have been developed which are very good at extracting semantic relationships from large numbers of documents. One of them is the (unsupervised) machine learning model Doc2Vec which constructs vectors for documents. The research project detailed in this paper uses this and other already existing algorithms to analyze the relationship between pieces of text. We set forth a broader ambition for this project before discussing the use and need of Doc2Vec. We set and evaluate criteria in order to examine the feasibility of Doc2Vec for accomplishing this broader ambition.

I dedicate this thesis to my friends Trevor and Alexis for the friendship and encouragement they always gave me. I also dedicate this work to my little brother in Phi Mu Alpha, Jonah Hathaway, for always inspiring me to better today than I was yesterday.

ACKNOWLEDGEMENTS

I thank the Mahurin Honors College at WKU for helping to fund this research through the granting of Gatton and Craft RELA award in both the Fall of 2019 and the Spring of 2020. I also thank the Honors College for allowing the opportunity to me and other students completing an honors thesis to defer our defense to the summer in response to the radical changes brought on by the COVID-19 pandemic.

VITA

EDUCATION

Western Kentucky University, Bowling Green, KY

B.A. in Mathematics - Mahurin Honors College Graduate

Honors Capstone: *A Description of a Humans Knowledge using Artificial Intelligence* - July 2020

Carol Martin Gatton Academy of Mathematics and Science in Kentucky - May 2016

PROFESSIONAL EXPERIENCE

Department of Extended Learning and Outreach, WKU

Web Developer - Jan. 2017 - May 2019

Music Department, WKU

Band Manager - Aug. 2019 - Dec. 2019

Mathematics Department, WKU

Undergraduate Research Assistant - Jan. 2020 - May 2020

Wycliffe Associates, Orlando, FL

Web Development Intern - May 2020 - Present

AWARDS AND HONORS

Cum Laude, WKU, July 2020

Fruit of the Loom Undergraduate Award, February 2019

Hugh F. and Katherine A. Johnson Mahtematics Award, June 2020

Contents

List of Figures

1 Introduction

In recent years, machine learning has helped to create various new techniques for interpreting data and has even helped to introduce brand new fields of research. Through open source coding practices and the ease of access to increasingly large amounts of data for little or no cost, machine learning is something that can be studied and practice by a wide range of individuals regardless of their professional level in the field. This leads to many exciting ideas becoming more and more feasible to create and make into a reality. This paper addresses the core mechanisms of one machine learning technique and the feasibility of that technique for accomplishing a broader ambition of the author. We begin with an analogy to put this broader ambition into the mind of the reader before discussing the machine learning technique in question. We then begin discussing the relevant background for this paper.

There is an analogy that the author became aware of before beginning this project. It is an analogy to relate the concept of knowledge of a human to a ball in 3 dimensions. Imagine a 3 dimensional ball that represents the knowledge of a person. We say that the inside of the ball represents all of the things that person knows, the outside of the ball represents all of the things that person does not know, and the surface of the ball represents all of the questions that a person may have. Thus, to learn more about anything translates to expanding this ball outward into the region around it that represents the unknown. This is acheived through asking questions and finding answers.

This analogy isn't perfect or rigorous by any means, but it is an interesting concept. We may then copy this analogy to 2 dimensions, where now the sphere is a circle. Everything inside the circle are things that are known by a person, everything outside is unknown, and the border of the circle represents questions. The author then formulated the question: “How might we represent these areas corresponding to what a person knows and does not know?”. If asked about a particular topic, a person will generally respond with words. That is, a person may articulate a specific order of words to represent their knowledge of a certain topic. It seems reasonable then that we may write these words down and label them with a topic. Those words together with the topic identifier then represent what a person knows about a particular topic. There could 10 words, 100 words, or thousands. It all depends on the person.

It is interesting to consider how we might organize these groups of words inside of the circle from the analogy. Suppose someone studies English Literature for a living. The core of that persons knowledge would then presumably correlate with large amounts of English Literature. That person would most likely have a lot to say about Jane Austen, Charles Dickens, and Emily Brontë. However, that person may not be as knowledgeable about other topics such as rocket science, discrete calculus, or the reproduction patterns of rabbits. The core of their “knowledge circle” would then be filled with strings of words discussing those authors and the outer edges of the circle may correspond more closely to other, more obscure topics to that person.

The question then is: “How do we take these strings of words and organize them according to what they have to do with each other?”. With our example before, it may be relatively easy for another human to analyze those strings of words that represent a persons knowledge and then organize them according to topical information. For example, it would be reasonable to organize groups of words in the “knowledge circle” that describe books written by Jane Austen separately from books written by Emily Brontë (even though there may be some similarities between them). What that accomplishes is subtle. There would be documents of text organized in such a way that documents sharing topical information would be placed near each other and topically distinct documents would be far away from each other. We could then use this organization of documents to see what topics a person is very knowledgeable about what topics lie on the “edge” of their knowledge. We may then be able to see what questions could be asked to expand this core knowledge of a person and how a person might more easily learn about the things that they do not know. That may be a useful idea to some, but it is impractical for a human to do such a thing when the knowledge of another human can exceed thousands and thousands of words or even documents representing groups of those words.

The proposed solution to this problem could be machine learning. Why read through thousands of lines of text to understand what the topics of some documents are when a computer can do it faster and most likely more reliably? This brings us to overarching ambition behind this project and paper. We want to be able to take in a large set of documents that detail a persons knowledge on a wide variety of topics. We want to have a computer organize these documents on a 2D plane such that documents which are topically similar are near each other and topics which have relatively little to do with one another are far apart. These documents may be represented as “tweet length” pieces of text that are meant to describe a persons knowledge regarding a specific topic. As a person learns more and more about something, they may add more documents and the computer should be able to update its organizations of documents to reflect this newfound information that a person put into the system.

Our proposed solution for this is a machine learning algorithm known as Doc2Vec (or Paragraph Vector)[2]. Doc2Vec is a machine learning algorithm that will take a set of documents and convert them into vectors. These vectors are used as numerical representations of those documents. We may then use these vectors to analyze different things about the documents to which they correspond. This paper analyzes Doc2Vec in a setting relevant to the discussion of the overarching ambition that has just been described. We set forth criteria to evaluate Doc2Vec and use those evaluations to make conclusions about how feasible Doc2Vec is in accomplishing the broader goal we have set.

2 Background

Before beginnig our conversation about background that is relevant for this thesis, we define a few terms that are helpful for the discussion.

When we discuss textual data that a machine learning technique learns from, we typically refer to the data as the *corpus.* This word encompasses all of the words, sentences, or documents that are used for training a machine learning algorithm. When we discuss words in a corpus, it is helpful to have a term referring to only the set of words. The word we use is *vocabulary.* The vocabulary of a corpus is all of the words in that corpus. They are ordered alphabetically so that there is a first word in the vocabulary. Also, speaking about the words in the corpus is not the same as speaking about the words in the vocabulary. All of the words in the corpus are in the vocabulary and vice versa. However, each word in the vocabulary is unique. The corpus is the collection of all the data and the vocabulary is simply all of the distinct words in the corpus in order.

There are also two general types of machine learning techniques: *supervised* and *unsupervised* techniques. A supervised machine learning technique is a technique where the data that it is given to learn about is labelled in some way. The data it is given has some or all parts of it described or labelled by humans before the technique trains. A good example of this is the area of picture classification. A machine learning technique must be given examples of what certain things look like, such as dogs or cats, before it is able to classify new photos as being a dog or a cat. A human must "supervise" the algorithm in the sense that a human tells it what is a dog and what is a cat. An unsupervised machine learning technique is one that does not have any sort of labelled data from a human. A good example of this is finding clusters in two dimensional data. An unsupervised technique may be given only the data and use an algorithm to determine how many clusters of points there are or where clusters are if it is given a number of clusters to find. These two types of machine learning techniques represent one of the main divisions in machine learning techniques.

In the remainder of this section, we discuss neural networks and some different ways to represent words as vectors. We detail word representations before document representations because word representations have been studied more and many of the concepts carry over when looking at document representations.

2.1 Overview

Word meanings are a fluid concept. Given a particular word, it can be difficult to tell what exactly that word means without having a context. That is, word meanings and the semantics of surrounding words are often ambiguous. Therefore, architectures and models have been designed and studied in the field of computer science in order to provide rigorous approaches to determining the relationships between words. We start by understanding neural networks as they are the underpinning of many machine learning algorithms and many of the concepts transfer directly to Word2Vec and Doc2Vec. We then inspect very simple approaches to building the relationships between words before introducing a more sophisticated method.

2.2 Neural Networks

Neural networks (sometimes referred to as Neural Nets) have been essential in the recent boom in machine learning. The approaches used by Word2Vec and Doc2Vec to learn vector representations of words and documents are closely tied to a neural net structure. This section gives a brief overview of neural nets and some key ideas behind them.

Neural nets are designed to emulate the human brain to some degree, which the naming suggests. The most basic neural net is a single neuron. Figure 1 shows what this neuron would look like.

The arrow pointing into the neuron represents some value that the neuron is going to take as an input. The arrow pointing out of the neuron represents some value that the neuron is going to output. The input can be thought of as

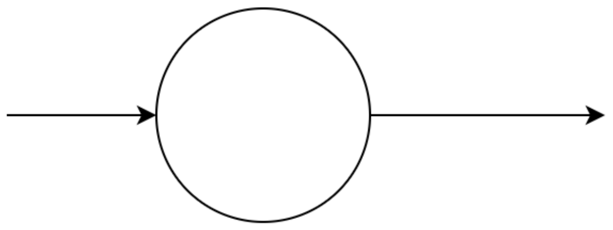


Figure 1: A Single Neural Net Neuron with One Input

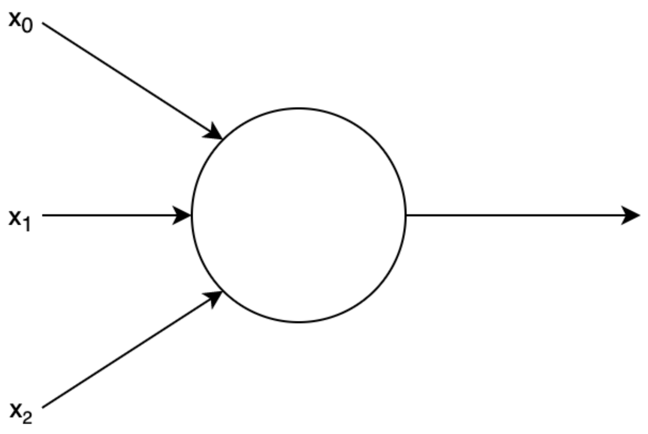


Figure 2: A Single Neural Net Neuron with Multiple Inputs

the electrical signal that a neuron in the brain receives. The output can then be though of as a measure of how much this neuron fires. That is, suppose we say that the output is exactly 1 when the neuron fires at full capacity (giving a strong electrical signal) and the output is exactly 0 when the neuron is not firing at all (giving no electrical signal). Often, this is precisely what is done in practice. Let us consider now what the math behind this looks like.

An actual neuron in a neural net (much like neurons in human brains) will generally have more than one input. See Figure 2.

Here, we consider the inputs of the neuron to be the numerical values x_1, x_2, and x_3. These values are usually outputs from other neurons which are then weighted. Suppose we give the weights w_1, w_2, and w_3 to x_1, x_2, and x_3, respectively. The weights can be thought of as the strength of the connection between individual neurons. The input for this single neuron will then be calculated as $X = w_1 * x_1 + w_2 * x_2 + w_3 * x_3$. Note that we do not yet call this the output of the neuron. We see later that the weights are what enables the network to learn from the data it is given.

Depending on the values of the weights, X may or may not be a value between 0 and 1. To handle this, X is passed through a function that "squishes" it into the the interval $[0, 1]$; this function is referred to as the *activation* function for a neuron. In practice, this function can vary from neural net to neural net, and in some cases, may even vary from neuron to neuron. For explanation purposes, we consider the *sigmoid function*, defined below, to be the activation function of every neuron.

$$\sigma(X) = \frac{1}{1 + exp(-X)}$$

where $exp(X)$ is the common exponential function. See Figure 3.

Using the sigmoid function, a single neuron can take in multiple inputs and return a value that is between 0 and 1 representing the firing, or lack thereof, of a human neuron. Suppose we are given inputs to a neuron with the associated weights. Let $\vec{x}$ be the vector containing all of the inputs and $\vec{w}$ be the vector containing all of the weights associated with those inputs. Following our example of a single neuron in Figure 2, the vectors would be $\vec{x} = [x_1, x_2, x_3]$ and $\vec{w} = [w_1, w_2, w_3]$. The output of this neuron is then $\sigma(\vec{x} \cdot \vec{w})$, where $\vec{x} \cdot \vec{w}$ is the dot product of $\vec{x}$ and $\vec{w}$.

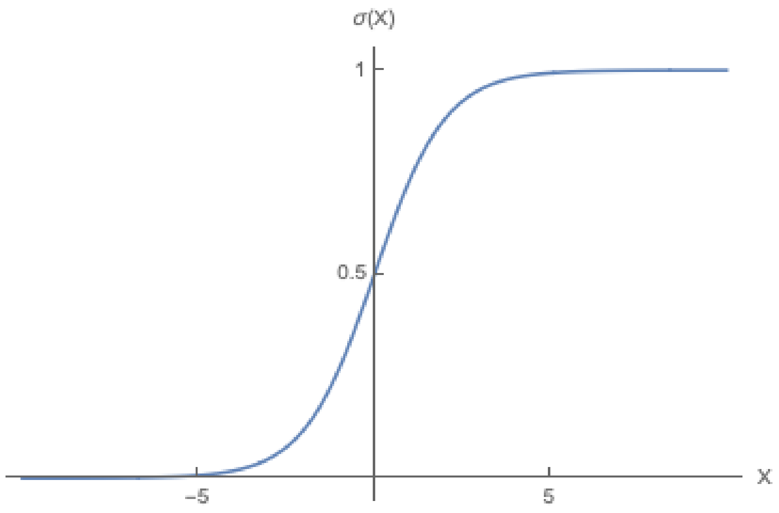


Figure 3: The Sigmoid Function

We build more sophisticated networks by putting many of these neurons together in sequence. For a very specific subset of neural nets, known as *feed forward networks*, neurons are connected in layers where each layer has a fixed number of neurons determined when the network is constructed. There are three general names given to the layers based on their location: *input* layer, *hidden* layer, and *output* layer. The input layer is the very first layer that accepts data as input. The output layer is the final layer of the network that gives the output of the network. A hidden layer is any layer of neurons between the input and output layer. These types of networks are known as feed forward networks because they feed data from the input layer to the output layer. There are other kinds of networks that can have data flow in the opposite direction or even in a circular fashion. However, we do not discuss those networks here. Figure 4 shows a simple example of a feed forward neural network.

Generally, the inputs to a network will have no weights associated with them. Every connection between neurons, however, will have a weight associated with

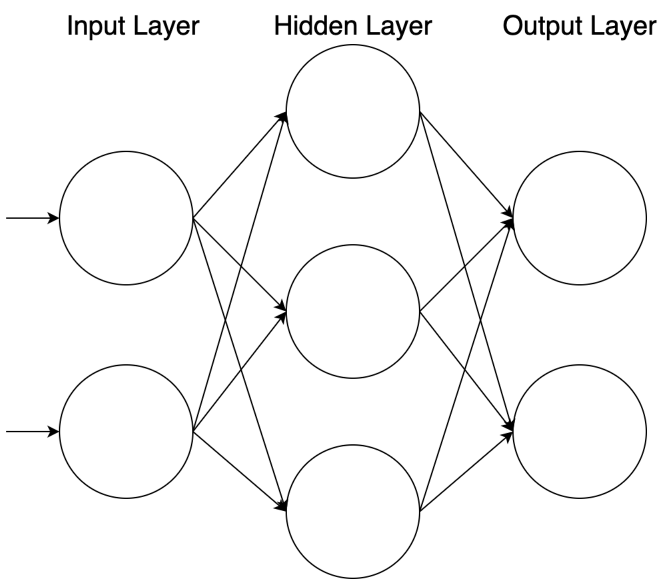


Figure 4: A Simple Neural Net

it. In practice, these neural networks can have thousands of neurons split between the input layer, output layer, and multiple hidden layers. Different neural network architectures perform better at different tasks. For this purpose, the weights in a neural network are defined using a slightly different notation that utilizes matrices.

For the example in Figure 4, let x_1 and x_2 be the inputs to the first and second neuron in the input layer, respectively. Let $\vec{x} = [x_1, x_2]$. Let $w^n_{i,j}$ be the weight associated with the connection between the i^{th} node in the n^{th} layer and the j^{th} node in the $n + 1^{st}$ layer where the top neuron in each layer is the first neuron, the input layer is the first layer, and we start counting from 1 in both cases. Then, we define the following two weight matrices:

$$W_1 = \begin{bmatrix} w^1_{1,1} & w^1_{1,2} & w^1_{1,3} \\ w^1_{2,1} & w^2_{2,2} & w^2_{2,3} \end{bmatrix}_{2\times 3}$$

$$W_2 = \begin{bmatrix} w^2_{1,1} & w^2_{1,2} \\ w^2_{2,1} & w^2_{2,2} \\ w^2_{3,1} & w^2_{3,2} \end{bmatrix}_{3\times 2}$$

where the subscripts on the matrices denote the dimensions of the matrix, simply for reference.

We may then write the output of the input layer as $\vec{x} \times W_1$, which is a 1×3 vector. Then, the values of the hidden layer neurons are equal to the outputs of the input layer with the sigmoid function applied to them. We denote this layer $\vec{h}$. To be precise, given an input vector $\vec{x}$, $\vec{h}$ is calculated as follows:

$$\vec{h} = \sigma(\vec{x} \times W_1)$$

$$= [\sigma(w^1_{1,1} * x_1 + w^1_{2,1} * x_2), \sigma(w^1_{1,2} * x_1 + w^1_{2,2} * x_2), \sigma(w^1_{1,3} * x_1 + w^1_{2,3} * x_2)]$$

$$= [h_1, h_2, h_3]$$

where inputting a vector to the sigmoid function just means to apply the sigmoid function to each value in the vector.

We denote the values of the neurons in the output layer as $\vec{o}$ and it is calculated similarly: $\vec{o} = \sigma(\vec{h} \times W_2)$. This results in a 1×2 vector. Thus, the entire calculation of the neural network can be written as $\vec{o} = \sigma(\sigma(\vec{x} \times W_1) \times W_2)$.

Now, for a neural network to "learn" something, it must generally have two things: data to train on and a method by which to learn. Neural nets are trained on any form of data ranging from images to population statistics. If the data can be transformed into some sort of numerical representation, then a neural network can most likely be used to interpret the data (with varying degrees of success depending on the specific nature of the data).

The methods that neural nets use to learn, however, are all somewhat similar. To train a neural network to perform a certain task, an *objective* function is defined (sometimes referred to as a *loss* function, or an *error* function, among many other names). This objective function measures the output of the network against what the expected output is. We generally want to minimize the value of this objective function. For example, the input to the network may be an image of a handwritten digit and the output is what the neural network believes the digit to be. That is, there are 10 neurons in the output layer where a value of 1 in first neuron means the network is very confident the image is a 0, a value of 1 in the second neuron means the network is very confident the image is a 1, and so on. The objective function would give a relatively high value if all of the output neurons gave a result of 1 (assuming that the input image is, in fact, supposed to be exactly one digit) and a relatively low value if the network correctly classified the input image. An example of what this objective function may look is as follows:

$$J(\theta) = \sum_{i=1}^{N} (o_i - y_i)^2$$

where N is the number of output neurons, o_i is the value of the i^{th} output neuron and y_i is the expected output of the i^{th} output neuron after the neural network has been given an input and each neuron has calculated its output based on the structure and formulas we have defined. θ is a placeholder for all of the parameters of the model that we will adjust to get a better output based on the objective function. For our purposes, the only parameters we consider adjusting for now are the weights associated with the connections between neurons. Stated more clearly, we have a function J that is dependent on multiple variables, and the task of learning is represented as finding the minimum of J in terms of the parameters being considered.

Given a set of inputs, we can minimize the value of the objective function by using a method known as *gradient descent*. In calculus, the gradient of a multi-variable function $f : \mathbb{R}^n \to \mathbb{R}$ represents the direction in which to take a step in the input space to generate the largest possible increase to the output. That is, if we take steps in the direction of the gradient of a function, we will be increasing the output of the function. If instead we take steps in the opposite direction of the gradient, then we will be decreasing the output of the function. This is precisely how objective functions are minimized. The gradient of a function is calculated using the partial derivatives of the function with respect to each input parameter. Since each weight of the network is a parameter, we consider the partial derivative of J with respect to each of those weights. Then, to update the network i.e. to *step in the right direction*, each weight is updated proportionally to the partial derivative of J with respect to it (where this proportion is usually small and reverses the sign of the numerical value of the derivative). This proportion is known as the *learning rate* and is a parameter that can be adjusted, but is generally not a parameter updated when training the model. By calculating the gradient for each training example, we update the network and thereby end up with a network that is better at minimizing the objective function. With a sufficient number of successive repetitions of this process, a network that minimizes the objective function J is yielded. In practice, the actual value of "a sufficient number of successive repetitions" is typically determined experimentally.

There are many, many ways at improving upon the described method. There are many problems that could arise: getting stuck in local minima of the objective function when updating the network, the number of neurons in a network may have to be adjusted, the computational complexity of large neural networks, etc. A detailed discussion of these topics is not relevant for the context of this thesis.

2.3 Word Representations

We now discuss ways to represent words numerically in order to formalize what words will look like when given to a neural net.

2.3.1 One Hot Vectors

In machine learning, a *one-hot vector* (sometimes called a *one-hot encoding*) is a $1 \times N$ matrix, or vector, that is used to distinguish unique words in a vocabulary. Consider the sentence "I like deep learning". If this were our entire corpus, we would have four words in our vocabulary: "I", "like", "deep", and "learning". The corresponding one-hot vectors could be as follows:

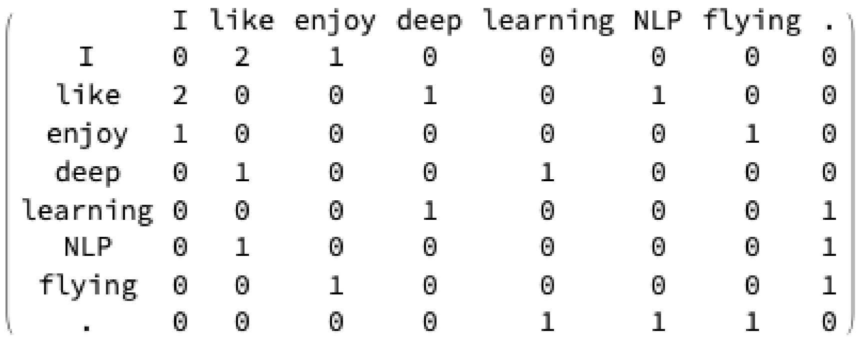
$$\text{I} \rightarrow [1, 0, 0, 0]$$

$$\text{like} \rightarrow [0, 1, 0, 0]$$

$$\text{deep} \rightarrow [0, 0, 1, 0]$$

$$\text{learning} \rightarrow [0, 0, 0, 1]$$

Each word has a corresponding vector such that there is exactly one element in the vector that has a value of 1 with the rest of the values being 0, hence the naming *one-hot*. The dimension of the vector is equal to the number of words in the vocabulary. Converting words in a corpus to one-hot vectors is a nice way of



$$\left(\begin{array}{ccccccccc}
 & \text{I} & \text{like} & \text{enjoy} & \text{deep} & \text{learning} & \text{NLP} & \text{flying} & . \\
\text{I} & 0 & 2 & 1 & 0 & 0 & 0 & 0 & 0 \\
\text{like} & 2 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\
\text{enjoy} & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
\text{deep} & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
\text{learning} & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
\text{NLP} & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\
\text{flying} & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
. & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0
\end{array}\right)$$

Figure 5: A co-occurrence matrix

organizing the vocabulary and assigning numerical representations to the words, which is an integral part of learning word representations for computers. However, there is virtually no semantic information about the words gained through this approach.

2.3.2 Co-occurrence Matrices

Another rudimentary way of representing words is by using a *co-occurrence matrix*. Consider the corpus consisting of three sentences: “I like deep learning. I like NLP. I enjoy flying.” The objective is to take this information and determine which words are most similar and which words are most dissimilar. Figure 5 shows a co-occurrence matrix for the corpus we consider.

A co-occurrence matrix is constructed by determining the atomic units of the corpus and counting how many times each unit appears next to another unit. Specifically, in this corpus, the atomic units are the words ”I”, ”like”, ”enjoy”, ”deep”, ”learning”, ”NLP”, ”flying”, and the end of sentence marker ”.”. Then, for example, ”I” appears next to ”like” twice, so the value at the first row and second column in the matrix is set to 2. Likewise for the second row and first column because the matrix is symmetric. Note that this is not *the* co-occurrence matrix for the corpus because we could reorder the rows and columns to get a different layout of 0's, 1's, and 2's in this case.

Using this representation of the corpus, it is possible to find relationships and similarities between words. It would be reasonable to deduce from this matrix that the words "like" and "enjoy" are similar because they both show up next to the word "I". It would then also be reasonable to say that "deep", "NLP", and "flying" are similar because they show up next to the words "like" and "enjoy". However, the number of relations that can be deduced from this representation of the corpus is limited and the relations that can be deduced from the numbers may or may not always make semantic sense. Suppose that the word "like" were replaced with "hate" in every instance. Then, the words "hate" and "enjoy" could be considered similar since they both would show up next to the word "I". In a sense, those words are similar because they make up a similar part of speech, though their meaning is quite different. In any case, this model is limited in the semantic information it can obtain. It is also very difficult for this sort of modeling of a corpus to scale as the dimension of the matrix is equal to the number of words and phrases we consider. For thousands of different words, this becomes difficult to manage.

Note that one-hot vectors are useful in this context. If we store the vocabulary as one-hot vectors, then we can simply perform a matrix multiplication with the co-occurrence matrix to obtain a vector corresponding to the word. That is, suppose we store the word "enjoy" as $[0, 0, 1, 0, 0, 0, 0, 0]$. Then, multiplying this vector by the co-occurrence matrix would yield the vector $[1, 0, 0, 0, 0, 0, 1, 0]$ which is the row (or column) of the matrix that corresponds to "enjoy".

2.3.3 Dense Word Embeddings

So far we have discussed what would be considered *sparse embeddings* of words. They are considered sparse specifically because of the abundance of 0's that show up in the representation of each word. In practice, sparse word embeddings have proven to be weak in terms of generating meaningful relationships for words from a corpus. In this section, we discuss what a *dense word embedding* is, what exactly a meaningful relationship is, and one machine learning algorithm that can be used for creating such dense embeddings.

Dense word embeddings have become very popular and useful for many reasons. These reasons include performance boosts for computer models, the ability to have a fixed vector size no matter the vocabulary size, etc. The most prevalent reason is the ability of dense word embeddings to satisfy analogical comparisons. That is, a machine learning model is considered good when it can generate embeddings for words that capture semantic and syntactic similarities between words. It is worth noting before proceeding that capturing syntactic information is not necessarily very hard. The co-occurrence matrix approach that was previously discussed captures the syntactic relationship between “like” and “enjoy”; it would also capture the syntactic relationship between “hate” and “enjoy” as laid out in the example in that section. However, that specific model would fail at capturing the semantic relationships between these words. Specifically, we want “like” and “enjoy” to be more similar than “hate” and “enjoy” because the former typically express the same sentiment.

A common way to test machine learning models that interpret text is to perform analogical comparisons of words. A few analogies that could be used as a test are as follows:

China	:	Beijing	::	Russia	:	Moscow
Japan	:	Tokyo	::	Germany	:	Berlin
Spain	:	Madrid	::	Portugal	:	Lisbon

These statements assert that, for example, the embedding for $China$ minus the embedding $Beijing$ should be roughly equal to the embedding for $Russia$ minus the embedding for $Moscow$. That is, the model would be considered good, in this example, if it captured the semantic relationship between countries and their capitals (assuming that the training data is sufficient to show these relationships). One model that has been the standard for producing and understanding how to produce these sorts of word embeddings is known as Word2Vec.

The Word2Vec model was originally proposed in [1]. Word2Vec is short for "Word to Vector". This is reminiscent of the fact that the model takes words in a large corpus and converts them to vectors. These vectors (word embeddings) capture the meaning of the corresponding word. Given two word vectors, we can calculate similarity between them where our similarity function is bounded above and below. That is, this similarity function takes two vectors as inputs and maps them to some interval of finite length. One function used quite often in machine learning for this task is the cosine similarity formula. It is known that, given two vectors $\vec{u}$ and $\vec{v}$, the following relation is true:

$$cos(\theta) = \frac{\vec{u} \cdot \vec{v}}{||\vec{u}|| * ||\vec{v}||}$$

where θ is the angle between the vectors and $|| \cdot ||$ is the Euclidean Norm. For this particular relation, the similarity measure is defined as $cos(\theta)$. That is, the similarity of two vectors is measured by the cosine of the angle between them. $cos(\theta)$ is bounded above by 1 and below by -1 where the vectors coincide when the value is 1, the vectors are exactly opposite when the value is -1, and the vectors are orthogonal when the value is 0. The vectors are said to be more similar the closer $cos(\theta)$ is to 1 and more dissimilar the closer $cos(\theta)$ is to -1. This function is extremely easy to implement. Let u_i and v_i be the i^{th} components of $\vec{u}$ and $\vec{v}$, respectively. Then $cos(\theta)$ can be written as:

$$cos(\theta) = \frac{\sum_{i=1}^{N} u_i * v_i}{\sqrt{\sum_{i=1}^{N} u_i^2}\sqrt{\sum_{i=1}^{N} v_i^2}}$$

where N is the dimension of both the vectors $\vec{u}$ and $\vec{v}$. This is computationally inexpensive and scales well as the dimension of the vectors increase. This method of calculating similarity between vectors has also been proven experimentally to work well and is thus a staple in machine learning papers and algorithms.

In the following section, we show some examples of the results of using Word2Vec.

2.3.4 Word2Vec Examples

Note that in this section, the figures were generated using mocked word vectors. The figures shown are consistent with results in Word2Vec models trained on a large corpus of data.

One of the most famous examples of using Word2Vec is the so-called $king-queen$ example. Figure 6 illustrates that taking the word vector for *king*, subtracting the vector for *man*, and adding the vector for *woman* results in a vector being very close to the vector for *queen* (where a dark orange strip represents a

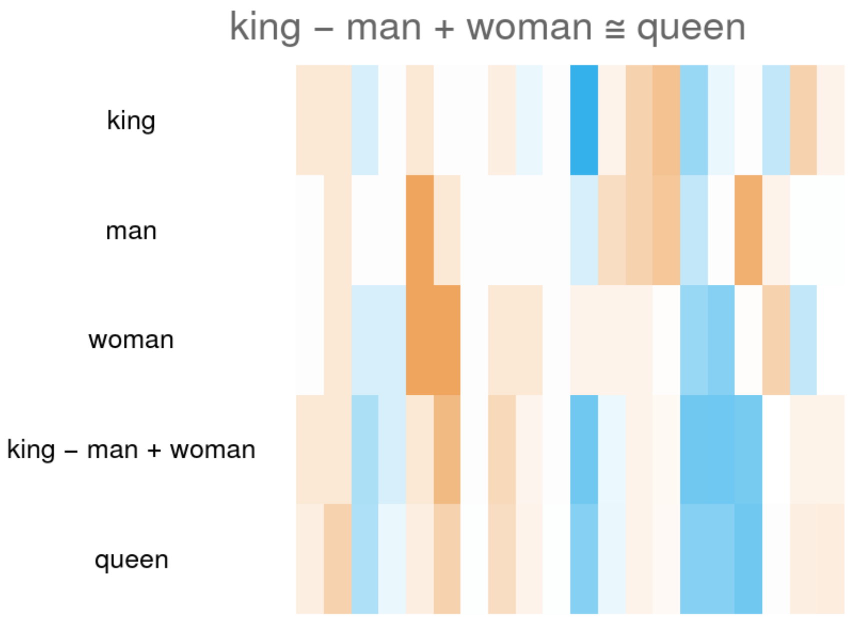


Figure 6: The Famous King-Queen Word2Vec Example

value close to 1 and a dark blue represents a value close to -1). That is, the vectors appear to have captured the semantic relationship and meaning between all of those words. This makes Word2Vec very good at completing analogical statements. In this example, king is to man as queen is to woman. Make one of those words an unknown that we want to predict (i.e. queen) and we can use Word2Vec to predict that unknown. It is important to note that the actual numbers in these vectors are not what is necessarily important; the initial values of the vectors before training is random. Rather the *relationships* the word vectors capture are what is important and consistent in this model.

Figure 7 shows a projection of high dimensional word embeddings into 2D. We see that the vectors between each country and its corresponding capital are very similar. That is, they are almost parallel. We can make a similar sort of statement as was described in Figure 6. Here, we could take the vector for $Germany$, subtract the vector for $France$, add the vector for $Paris$, and get a

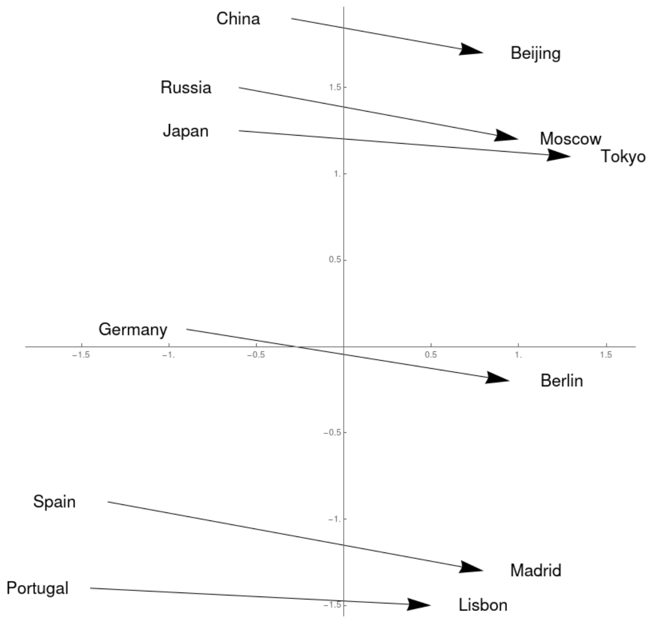


Figure 7: Relations Between Countries and Capitals (adapted from Figure 2 in [1])

vector that is very close to the vector for *Berlin*.

It is worth noting that, as with any other machine learning model, Word2Vec is entirely susceptible to bias in the training data. Figure 8 shows these biases. The vertical axis is only present in order to give more room to place words in the picture. The horizontal axis indicates semantic similarity for the words shown in relation to the words he and she. Words further to the left are more similar to "she" and words further to the right are more similar to "he". Many of these examples make sense. “Beard” is more closely associated with being masculine whereas “sisters” is more closely associated with being feminine. In contrast, “friend” is located directly in the middle. These results could be flawed due to poor training data, or they could indicate relations between words that are true but are not entirely apparent at first glance. In any case, the takeaway from this example is that quality of training data is integral to be able to get a model that

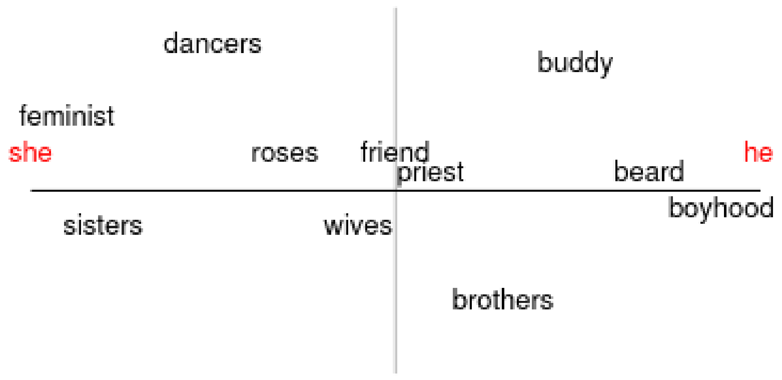


Figure 8: Masculine vs Feminine Traits (adapted from Figure 3 in [3])

represents language well.

3 Approach

The idea of this project was partially based on some general exposure of the author to machine learning ideas and particularly the basic ideas of Word2Vec and Doc2Vec. The goal of this project was and is to assess the feasibility of the Doc2Vec machine learning algorithm to accomplish the broader goal set forth in the introduction of this paper. Upon initial studies, Doc2Vec seemed to be very promising for this project. The paper in which it is presented details a few experiments where it excelled at the classification tasks it was given. These experiments include sentiment analysis (determining whether a piece of text means something "good" or "bad") and infomration retrieval (determining which two documents out of three belong together). More details of these experiments and their results can be found in [2]. For the broader goal that has been discussed in this paper, we have established three important criteria that describe how well Doc2Vec might work for our project.

First, given a large enough set of documents that are classified by a human into several distinct classes, to what extent does Doc2Vec learn document embeddings that mirror those (human) classification decisions? Doc2Vec is useful only if its learning algorithm aligns with the intuitive decisions humans make about which documents "belong together".

Second, for a large enough set of documents that Doc2Vec separates well into distinct classes, to what extent do the document embeddings learned by Doc2Vec capture intra-cluster relationships? While each document might have a main topic which is the basis for the classification, it also may touch on many other topics. To capture the relationship between a collection of written items, it is important for Doc2Vec to discover some relationships between documents based on these subtopics.

Third, given a Doc2Vec model that has clustered documents according to their topics determined by humans, is that model able to accept another brand new document with a topic that has been given to the model and cluster that document with the pre-existing documents of the same topic? Doc2Vec should be able to accept new documents and classify them based on the documents it has already seen. A new document with a similar topic as documents that have already been clustered should be close to the topics in that cluster.

In the following sections, we describe the Skip-Gram Model Architecture for the Word2Vec machine learning algorithm. One learning method for it, namely the Softmax classifier, is described as well. Through the intuition gained in the exposition of Word2Vec, we then discuss Doc2Vec. We then describe the dataset used for this project and how we justify the use of that dataset. Finally, we more precisely layout how we assess the criteria with which we evaluate the Doc2Vec algorithm. We do not yet evaluate the criteria in this section; we simply outline more clearly what it means to do well or poorly for each criteria. Through this order of explanation, we aim to give the reader an understanding of each major piece of this project so that the results and conclusion for this paper can be more naturally digested.

3.1 Skip-Gram Model

There are two distinct architectures used for training the Word2Vec model that were proposed in [1]. Here, we only discuss one of them: the Skip-Gram (SG) model. Although before discussing the architecture, we showcase the softmax classifier.

3.1.1 Softmax Classifier

The softmax classifier is a common operation used in machine learning. It takes a vector as an input and returns a vector as an output. The usefulness of the softmax classifier is that it takes the input vector and converts it to a vector of what can be thought of as a confidence rating. That is, it takes all of the values to another value between 0 and 1. If each value represents the likelihood of an outcome, a value closer to 1 can be interpreted as representing a much higher confidence that the corresponding outcome will occur. The operation also exaggerates the differences between the numbers in the vector and normalizes them so that all the output values will add up to be 1. To define the operation, let X be a vector with dimension N and let x_i be the i^{th} element of X. Then, the softmax classifier for x_i is defined as follows:

$$softmax(x_i) = \frac{exp(x_i)}{\sum_{j=1}^{N} exp(x_j)}$$

and then the softmax classifier for X is defined as follows:

$$softmax(X) = [softmax(x_1), softmax(x_2), ..., softmax(x_N)]$$

This operation is widely used because it is relatively inexpensive in terms of computation time. The value of the denominator for each element of $softmax(X)$ needs to be computed only once and it can be stored and reused in order to calculate the value of the rest of the output elements. Shown below are a few examples of the softmax classifier being applied to low dimensional vectors where the vector on the left is the input vector and the vector on the right is the output.

$$\begin{array}{ccc} [1, 1, 1] & \rightarrow & [0.333, 0.333, 0.333] \\ [-10, 20, 30] & \rightarrow & [4.248 * 10^{-18}, 4.540 * 10^{-5}, 0.999] \\ [6, 7, 3141] & \rightarrow & [3.068 * 10^{-1362}, 8.339 * 10^{-1362}, 1.000] \end{array}$$

The reader can confirm the elements of the computed vectors on the right add up to 1 (or close enough to 1 to be considered equal for computational purposes). These examples also illustrate that the softmax classifier computes relative levels of confidences of events happening. If the values in the input vectors were thought of as weights for the likelihoods of some events occurring, the corresponding values in the output vectors can be interpreted as the relative confidence that those events happen. Additionally, the softmax operation does not weight values based on position i.e. if all values in the output are the same, then all values in the output will be as well. The operation also preserves the relative magnitude of the input vector so that the smallest element of the output corresponds to the smallest element of the input and so on. Because of these properties of the softmax classifier, it is widely used for determining what a machine learning model is predicting based on the output it gives.

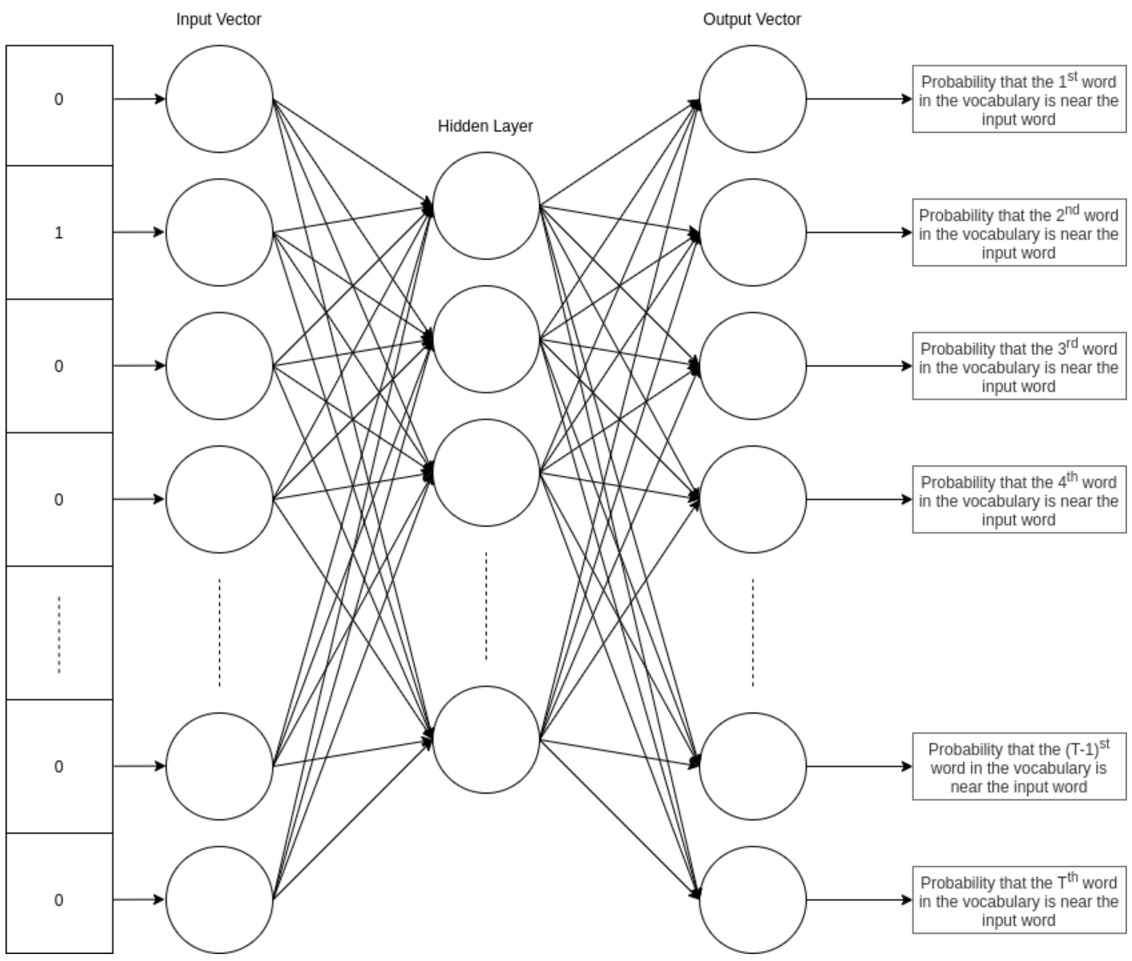
3.1.2 Skip-Gram Architecture Overview

In machine learning, it is common to make neural networks try to produce an output that is not exactly relevant to what the given task is actually trying accomplish. For example, consider the goal of dimensionality reduction. Many approaches to this problem that use neural networks structure the network such that there are the same number of input neurons as there are output neurons. The task given to the network is to simply produce an output that is as similar as possible to the input. To learn the dimensionality reduction, the network will have a hidden layer at some point between the input and output that has a number of neurons equal to the desired number of dimensions for the reduction. This means that the data is being compressed to a number of dimensions desired when it reaches this hidden layer of the network. The actual relevant information that the network produces are the set of weights from the input layer to the hidden layer containing the values after dimension reduction. This is a clever way of producing a good dimension reduction without actually having to decide ahead of time what a good reduction looks like. That is, it's possible to then save these weights and the activation functions of the neurons to easily repeat the dimensionality reduction on data given later.

The Skip-Gram (SG) architecture for Word2Vec takes a similar approach in the way that it tries to predict words. The task it is given is to predict which words will most likely be nearby a given input word. It will produce a relative confidence for each word in the vocabulary corresponding to the likelihood that that word is nearby the input word in the corpus. We will see that what is actually retrieved from the network in the end are the weights between the input layer and the hidden layer. The output is discarded once training has been completed.

In this architecture, the system is trained to predict nearby words given an input word. To clarify this concept, let w_c be an input word to the architecture and let w_o be a word that is close to w_c. Note that, at this point, the exact meaning of "close" has not yet been defined. Assume for simplicity (for now) that w_o is a word either directly to the left or right of w_c in the corpus. These two words are then used as a training example for the network. The input is w_c and the expected output is w_o. Since the same input word w_c is "close to" other words as well, this translates to the network being tasked with predicting what words will be near a given input word.

Words that are close to each other can change depending on the topical information discussed in the corpus that is considered. For example, consider the sentences "I love deep learning." and "I love deep diving." If both of these sentences are present in the corpus, this will incentivice the neural network to associate the word "deep" with both words "learning" and "diving". If only one of these sentences were in the corpus, the representation of the word "deep" would then change. What this means then is that the network with the SG architecture is tasked with capturing semantic information about words; it is tasked with understanding how words are related to one another based on their proximity to



Input Vector
Hidden Layer
Output Vector
0
1
0
0
0
0
Probability that the 1st word in the vocabulary is near the input word
Probability that the 2nd word in the vocabulary is near the input word
Probability that the 3rd word in the vocabulary is near the input word
Probability that the 4th word in the vocabulary is near the input word
Probability that the (T-1)st word in the vocabulary is near the input word
Probability that the Tth word in the vocabulary is near the input word

Figure 9: The SG Architecture Neural Network for Word2Vec

each other.

Given this task, a corresponding neural network structure needs to be designed. The input to the network is a one-hot vector that corresponds to the input word. There is one hidden layer consisting of a number of neurons equal to the dimension of the word embeddings that will be the final result of training. Finally there is the output layer which has a number of neurons equal to the number of words in the vocabulary (which is, in fact, the same as the dimension of the input vector). Figure 9 shows what this looks like as a neural network along with a sample input.

Once training has been completed, the weights going from the input layer to the hidden layer are saved. For each word in the vocabulary, there are a number of weights corresponding to that word where the number of weights is the size of the hidden layer. The vector of these weights now represents the embedding of the corresponding word. The vectors for each word are the relevant results of the trained network and they are what is used to represent the words as a dense word embedding.

3.1.3 Skip-Gram Architecture Network

We now introduce the neural network design for this architecture. The discussion in this section is designed to be much more technical than discussions previously in this thesis.

For this particular architecture, the activation function for each neuron in the hidden layer is the identity function. The activation function for the output layer is the softmax classifier. The softmax classifier is applied to the output layer as if it were a vector. Because the activation function for each neuron in the hidden layer is the identity function, the value of the output neurons before applying the softmax classifier is represented by a straightforward matrix multiplication. Before continuing, we explicitly define some of the parameters and properties of the architecture. Figure 9 can be used to visualize how each of these parameters and properties relate to the neural network structure.

Let T be the size of the vocabulary and $\vec{x}$ be a one-hot vector input for this network so that $\vec{x}$ is a $1 \times T$ dimensional vector. Let d be the desired dimension for the final word embeddings that the architecture will produce. Consequently, this will be the size of the hidden layer in the architecture. Let V be the weight matrix from the input layer of the network to the hidden layer. Let U be the weight matrix from the hidden layer to the output layer. V and U will have dimension $T \times d$ and $d \times T$, respectively.

The values in V and U before training are determined randomly and uniformly. These matrices should be thought of as representing two different word embeddings for each word in the vocabulary. V represents word embeddings for input words and U represents word embeddings for output words. To be precise, each row of the matrix V corresponds to a word embedding for a word in the vocabulary where the i^{th} row is the embedding for the i^{th} word in the vocabulary. The input one-hot vector $\vec{x}$ then acts as a "look-up" for this word embedding since, through the matrix multiplication, it selects the row of the matrix corresponding to the input word. The columns in the matrix U then correspond to separate word embeddings for words in the vocabulary where, as before, the i^{th} column is the embedding for the i^{th} word in the vocabulary. Thus, before the activation function is applied, the values of the neurons at the output layer can be calculated as $\vec{x} \times U \times V$ which is a vector with dimension $1 \times T$. Each value in this vector then represents the dot product between the word embedding for the input word in the matrix V and the other embedding of each word in the vocabulary which are stored in U. The softmax classifier is then applied to this vector. Each output neuron represents the confidence of the network that the corresponding word in the vocabulary is a word that is close to the input word.

It is now necessary to define exactly what it means for a word to be "near" another word. To do this, a *window size* is defined. Let $m \in \mathbb{N}$ be the window size for the architecture. Then, for a given word w_t in the corpus, we consider the word w_{t+j} to be near w_t where $j \in \{-m, -m+1, ..., m-1, m\} \setminus \{0\}$. All such words w_{t+j} are then said to be in the same window as w_t. As an aside, note that $j \neq 0$ means that w_t is not considered to be near itself (although it is entirely possible that a word shows up more than once in a window in which case the word *is* considered to be near itself).

We now define the objective function J:

$$J(\theta) = \frac{1}{W} \sum_{t=1}^{W} \sum_{-m \leq j \leq m, j \neq 0} -log(p(w_{t+j}|w_t))$$

where θ represents all of the parameters of the system, W is the size of the corpus, and w_i is the one-hot vector corresponding to the i^{th} word in the corpus. The inner summation symbol represents the process of looking at nearby words for a given word, and the outside summation represents repeating the process for all the words in the corpus. The value of $p(w_{t+j}|w_t)$ is determined using the result of the calculation described by Figure 9. That is, given an input word and a word that is expected to be near it, the model should predict a value where the closer to 1 the value is, the better the the model is judged to have performed. To be precise, $p(w_{t+j}|w_t)$ is defined as follows:

$$p(w_{t+j}|w_t) = p(o|c) = \frac{exp(u_o \cdot v_c)}{\sum_{w=1}^{V} exp(u_w \cdot v_c)}$$

where o is the outside (or output) word index in the vocabulary of the $j+t^{th}$ word in the corpus and c is the center word index in the vocabulary of the t^{th} word in the corpus, and u_o and v_c are the outside and center vectors of indices o and c. That is, u_o is the o^{th} column vector of the matrix U and v_c is the c^{th} row vector of the matrix V, both of which were defined earlier in this section.

The reader should be careful to notice that the subscripts of w_{t+j} and w_t in the function J refer to a position of a word in the corpus, whereas the subscripts of u_o and v_c refer to a position of a word embedding in the vocabulary. Because each of u_o and v_c can be obtained by multiplying the corresponding one-hot vector for a word in the corpus by the appropriate matrix, the implementation of this algorithm requires a separate data structure that allows for retrieving the one-hot vector needed given a word in the corpus. We do not discuss this separate data structure here, but one should keep in mind that there must be a translation done in order to calculate the value of the function p.

At this point, we may consider minimizing the objective function. Recall from our earlier discussion of neural networks that objective functions are typically minimized using gradient descent. For a number of reasons, though, gradient descent is impractical in practice. The most prevalent reason is that gradient descent requires the calculation of the gradient for every input and output pair followed by an update to the entire network. This can get very slow and very unwieldy quite quickly.

In practice, another method that is very similar to gradient descent is used that is called *stochastic gradient descent*. In stochastic gradient descent, instead of updating the network with respect to all of the training examples at once, the network is updated with respect to only one training example. Because of this, we may calculate the partial derivative with respect to each of the parameters of the network but only update those parameters once. This eliminates the need to consider summing in the objective function and we may focus directly on the summand, $-log(p(w_{t+j}|w_t))$. The partial derivative with respect to a center word vector is shown here:

$$\frac{\partial}{\partial v_c}(-log(p(w_{t+j}|w_t))) = -u_o + \sum_{x=1}^{T} p(x|c) * u_x$$

v_c is then updated by a value directly proportional and opposite to this partial derivative. The partial derivative with respect to u_o, an outside word vector, is similar. The update is made all at once for all u_o and v_c in the system.

Stochastic gradient descent is considered to be sufficient in many cases involving neural networks because it tends to generate neural networks that perform well even though not every parameter is being updated at each step of learning.

It should be noted that there are various ways to improve performance of training these word vectors. The original paper introducing Word2Vec describes methods to improve the learning efficiency [1]. These methods include a better approach for calculating the softmax classifier called *hierarchical softmax*. It is more computationally efficient than the standard softmax classifier and is typically used in practice, however because of the overall complexity of the method, we only explained the softmax classifier in this paper. Another learning method that is discussed in [1] is referred to as *negative sampling*. Instead of learning word representations by learning which words are "close", negative sampling chooses words that are far away in the corpus from the given input word. The model will then be updated by pushing word embeddings further apart for words that are far apart in the corpus.

After the discussion of the Word2Vec model and its particular architecture, we now present and discuss Doc2Vec, which invokes a very similar method of learning.

3.2 Doc2Vec

The approach for learning embeddings that Doc2Vec employs is inspired by the methods for learning word embeddings. In particular, the architectures that Doc2Vec utilizes for learning are similar to the architectures that Word2Vec uses for learning. As such, documents are also represented as embeddings. In this section, we detail the Distributed Bag of Words architecture, which is abbreviated as PV-DBOW (the "PV" comes from the fact that the original paper refers to Doc2Vec as "Paragraph Vector" [2]). This architecture is very similar to the SG architecture in that it is given a similar task in order to generate document embeddings.

In the PV-DBOW architecture, a neural network similar to the one in the SG architecture is formed. The task given to this network is to learn document embeddings that predict which word embeddings best represent the document. The word embeddings and the document embeddings are learned together. The word embeddings are shared by all words in all documents. That is, the word "deep" would have the same embedding regardless of which document it appears in.

In Doc2Vec, documents are treated similarly as to how center words are treated in Word2Vec. Each document is assigned a one-hot vector corresponding to its position in the corpus. Note that the order of the documents is irrelevant. They are only ordered so that they are easier to retrieve when training. One-hot vectors for documents are then used as inputs to the neural network. A matrix

D which is analogous to the matrix V from the Word2Vec discussion is used to retrieve a vector corresponding to the embedding for the input document. This document embedding is then used to predict words in a window.

The prediction can be thought of as essentially the same as the prediction in the SG architecture. A document is given as input to the network. The resulting document vector obtained via multiplication with matrix D gives the documents embedding and corresponds to the hidden layer of the network. The activation function on this layer is the identity. The resulting document vector is then multiplied with a word vector residing in a matrix that is analogous to the matrix U from before.

Learning happens by stochastic gradient descent. The objective function can be explained similarly to the one for Word2Vec. It also uses a window size and has an extra summation over all of the documents. More precisely:

$$J(\theta) = \frac{1}{D} \sum_{d=1}^{D} \sum_{t=1}^{d(w)} \sum_{-m \leq j \leq m} -log(p(w_{t+j}|D_d))$$

where D is number of documents, $d(w)$ is the number of words in the d^{th} document, w_{t+j} is the $t + j^{th}$ word of the document in question, and D_d is the document embedding for the d^{th} document in the corpus.

For simple stochastic gradient descent, a random document is selected as input and a random word and the gradient for that word and that document is computed and used to update the weights. Since the objective function calculates the summand based on the word and document, the location of the word in the window may be ignored in the update. A diagram of the network for Doc2Vec is identical to the one shown in Figure 9.

In essence, Doc2Vec attempts to generate document embeddings that are representative of and related to the meaning of the words (as captured by the word embedding) of the document. Because of this, if two documents share many of the same word meanings, they should be similar documents. By the same logic, if two documents do not share many word meanings at all, they should be dissimilar documents. Furthermore, since we expect Doc2Vec to learn the meaning of words, if a document contains words that are synonymous to words of another document, those documents should still be considered similar.

3.3 Dataset

Text was gathered from three distinct topics. Those topics, broadly, are card game articles, sports articles, and dance papers. These documents were chosen because the author is familiar with all three topics. The specifics of what the portions of text are, how they were processed, and the method used to obtain them is described in the remainder of this section.

3.3.1 Gathering Data

The card game articles in our dataset all pertain to a specific popular playing card game known as *Magic: The Gathering*. It is a card game with a rich history and with many content creators writing articles and producing videos about it every day. The articles for the game were obtained from *CoolStuffInc*[1]. Specifically, 50 articles split between 2 authors were obtained from the company website. The content in these articles consists of various keywords relating to the cards that have been printed in the game that we believed would set these documents apart

[1]https://www.coolstuffinc.com/

from other documents discussing other topics.

The sports articles were all obtained from the ESPN[2] website. Out of the 38 articles that were gathered, 10 were written about baseball (MLB), 10 about football (NFL), 8 about basketball (NBA), and 10 about hockey (NHL). These articles contain different keywords relating to the sports they discuss while also containing some commonalities between them. In a sense, there are 4 sub-topics within the sports articles corresponding to the 4 different sports discussed. We found this valuable because we could put all 38 articles under the topic of sports while also having a smaller dataset that we could use to try to understand the intra-clustering properties of Doc2Vec.

Our last topic that we chose was dance. Specifically, the documents that were selected for the topic of dance were articles and papers discussing the traiditional Indian form of dance known as Bharatanatyam. There were 26 articles gathered. Bharatanatyam is a particular form of dance that the author had studied briefly and was familiar enough with to justify including it as a topic. The content of these dance documents was certain to be somewhat distinct from the other documents gathered.

All documents gathered were cleaned and stripped of unnecessary characters. In machine learning, there are certain words that are referred to as *stopwords*. Stopwords are words or characters that have been generally agreed upon by researchers to not add any value to the learning process for natural language processing algorithms. For example, the single quote mark ' is removed from text as it is uneccessary to store. The effect that it has on differentiating the words

[2]https://www.espn.com/

"its" and "it's" (and other similar word pairs) is generally agreed upon to be irrelevant to the learning of algorithms. Words such as "a", any other single letters, and punctuations were also removed from the documents. These characters are considered to not have much weight on the semantic meaning of a document. All words in the document were also normalized to be in lower case. Because of this, the words "Jack" and "jack" would be treated as exactly the same.

3.3.2 Justification of Data

These documents share words, phrases, and terminology. However, if a human were given these documents, they would be able to sort them into three general categories. The card game articles are very clearly discussing happenings in the card game and are analyzing different strategies and cards. The sports articles deal with different players in the sports and news regarding what has happened in those sports. The dance articles and papers are full of different keywords and ideas that are sure not to show up in most other contexts due to their very specific topical context. However, we decided that human reasoning was not a stringent enough criteria for determining that these documents belonged to three separate classes.

In order to make the distinctions more rigorous, we evaluated these documents using two traditional supervised classification algorithms. Supervised machine learning algorithms require a human to pre-define what sort of the thing the algorithm should be learning. In the case of text classification, a human must split documents into separate classes and then tell the supervised algorithm which documents are in which class. The performance of the algorithm is evaluated by tasking it with classifying new documents that is has never seen. These two sets of documents (i.e. the set the algorithm learns on and the set it is tested on) are commonly referred to as the "training" data and the "test" data. Each set of data contains documents that are classified before training by a human. Our goal with these two classification algorithms was to see how well a supervised learning algorithm could learn the classes of the documents that we, as humans, have defined. If traditional supervised algorithms could not perform this task, then it would be difficult to test Doc2Vec on its abilities to do the same thing.

For both supervised algorithms that are discussed in the rest of this section, 70% of the data in the dataset was used for the training data and the remaining 30% was used for testing. Specifically, 70% of each predetermined class of documents was used for training data and the remaining 30% of each predetermined class was used for testing. For training, this results in 35 card game articles, 27 sports articles, and 18 dance articles. For testing, this results in 15 card game articles, 11 sports articles, and 8 dance articles. The accuracy of each of the supervised algorithms is reported as a percentage representing what percentage of the testing data the algorithm correctly classified.

The first algorithm that we used on the documents is the *Naïve Bayes* algorithm. There are multiple Naïve Bayes algorithms including Multinomial, Bernoulli, and Gaussian. For our purposes, the Multinomial Classifier is the most appropriate. Traditionally, this algorithm has been used for classifying documents into categories such as sports, politics, technology, etc. [4, 5]. It is also often used as a standard for determining which emails among many are spam and not spam.

At its core, the Multinomial Classifier determines the probability that a document is in a certain class based on the frequency of words in the document. That is, once the classifier has learned on a set of documents, it determines if a new document belongs to a class by calculating the similarity of documents based on the frequency of common words between documents. In that sense, we expect this classifier to be a good representation of whether or not the documents we chose are actually representative of three different classes.

After training the Multinomial Classifier, we obtained an 85% success rate on the classification of the testing data. It classified all card game articles and sports articles given to it correctly. Out of the 8 dance documents it was given, it classified 3 as belonging to the "dance" class and 5 as belonging to the "card game" class.

The second algorithm that we considered is referred to as *Support Vector Machines* or *SVMs*. SVMs were originally designed for binary classification of data. That is, it was originally intended to find the boundary, if one existed, between two sets of data that belonged to distinct classes. Much research has been done to adapt this method of supervised classification to work with more than just two classes [6, 7]. Figure 10 shows an example of how SVMs work to classify data in a binary setting.

In the figure, there are two classes in the data: the black dots and the white dots. The three lines H_1, H_2, and H_3 show three different ways of trying to separate this data. The line H_1 does not the separate the data and would be a poor attempt at trying to classify the data. The line H_2 technically does work; it separates the data correctly and would be good to use as a separator if the dataset

[3]Licensed Under Creative Commons: https://creativecommons.org/licenses/by-sa/3.0/deed.en

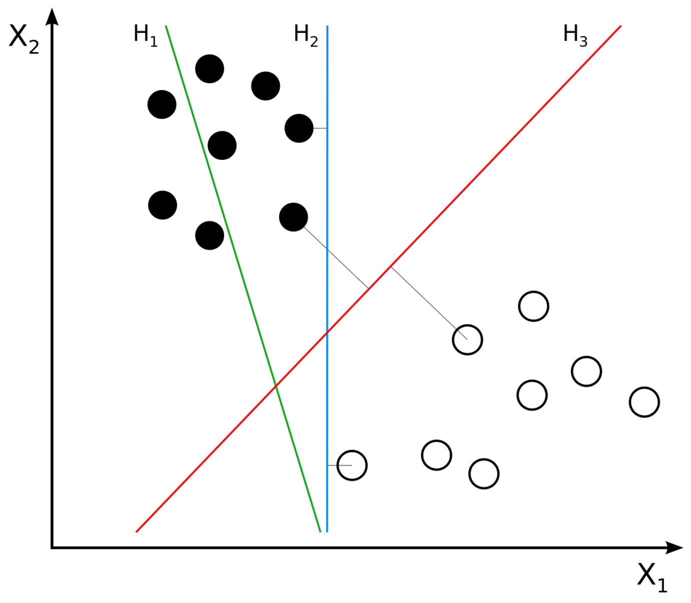


Figure 10: A Graphical Representation of an SVM Classifier

Used with permission from [8]^3. Last Accessed on July 26, 2020

were to never change. SVMs attempt to build a line of separation more closely resembling the line H_3. This line not only separates the data, but does so while also maximizing the margin between the line (see the gray lines in Figure 10) and every single data point. In this regard, one would consider the line H_3 to be the best line separating the data.

Of the methods devised for extending SVMs beyond binary classification, the method we used is known as "one-vs-the-rest". In this method of classification, each document in a certain class is treated as though it and other documents in that class are separate from everything else. In essence, it treats the problem as a binary classification for multiple subsets of the data. Because of the distinct topics we chose, we believed that this method would be appropriate for determining if those topics are, in fact, distinct from one another.

After training the SVMs, we obtained a 100% success rate on the classification of the testing data. In fact, we lowered the training data to only 25% of the total documents in the dataset (where the documents were split as before) and after running 1000 simulations with different random initial states, the classifier achieved an average of 99.8% accuracy.

Because of our own intuition of the classes for the documents and the results of the supervised classification algorithms, we determined that the dataset was sufficient to evaluate Doc2Vec on the criteria we had established for it.

3.4 Evaluation Criteria

In this section, we explain to a greater degree the criteria that were mentioned at the beginning of Chapter 3. We detail how we evaluated the criteria and explain why the criteria are reasonable to use to determine how "good" Doc2Vec is for our purposes. The next few sections breakdown each criteria and how we assessed them in the project. The results for assessing these criteria on the outcome generated by Doc2Vec learning will be given in Chapter 4 later in this paper.

Before transitioning into a discussion of the criteria, we explain the phrase "a large enough set of documents" that is mentioned in the first and second criteria. The definition of this is a bit arbitrary. Doc2Vec, along with most machine learning algorithms, can work on many different sizes of datasets. In general, more data is better. Theoretically, the more examples something has to learn from, the better it will learn (assuming the data is "good" data, however "good" is defined in the particular domain).

We considered the dataset we have gathered to be "large enough". This is because we wanted to test to see if Doc2Vec could handle a dataset of this size. For the broader goal of this project, datasets will be relatively small. In general, the dataset we have compiled would be considered extremely tiny. However, to ensure that we obtain results that are the most pertinent to our purpose, the dataset we have gathered is considered "large enough".

3.4.1 Criteria 1

The first criteria we stated was that "for a large enough set of documents that are classified by a human into several distinct classes, to what extent does Doc2Vec learn document embeddings that mirror those (human) classification decisions?". We approached this by looking for clusters in the document embeddings learned by Doc2Vec. There are various ways to do this. One popular way is by using the k-means clustering algorithm, which is the approach we use to assess the criterion.

The k-means clustering algorithm works by partitioning observations in a dataset into k different clusters. The algorithm can work for data in any finite number of dimensions. The input to the algorithm is the expected number of clusters. The output is a list of which of the entries in the dataset belong to which cluster that the algorithm identified.

When the algorithm receives the number of expected clusters, it randomly initializes that number of cluster centers. That is, given an input of 3 expected clusters, the algorithm generates 3 points in the input space m_1, m_2, and m_3 to be the center points of the 3 clusters by selecting 3 points at random from the data. The data is then partitioned by associating each data point with the closest cluster center among m_1, m_2, and m_3. The algorithm then iteratively moves each of these randomly chosen center points such that the sum of the means of the distances for points in each cluster from its center point is smaller than the previous step. One may specify the number of iterations for the algorithm, but this is not necessary as the algorithm may stop once there is a neglible change in the quantity that the algorithm minimizes.

The k-means algorithm will tell us how Doc2Vec clusters the documents. If Doc2Vec clusters most of the document embeddings such that the topically similar documents are in the same clusters, then we assert that Doc2Vec has demonstrated learning the classes among the documents.

3.4.2 Criteria 2

The second criteria we stated was that "for a large enough set of documents that Doc2Vec separates well into distinct classes, to what extent do the document embeddings learned by Doc2Vec capture inter- and intra-cluster relationships?". We approached this by examining the relative positions of documents in 2D projections of the document embeddings produced by Doc2Vec. We analyzed documents that consistently showed up on the "outside" of their cluster, consistently showed up "inside" their cluster, or consistently showed up "between" two clusters. What is meant by "inside", "outside", and "between" will be defined in Chapter 4.

The 2D projections of the document embeddings are created using the t-SNE projection algorithm[4]. This algorithm works by examining local relationships between points in the input space (the document embeddings) in order to ensure that the points in the output space (the 2D projection) are a good representation of the input space. That is, points that are close in the input space will be close in the output space and vice versa.

[4]https://lvdmaaten.github.io/tsne/

Using this projection algorithm, we can then analyze the documents which often show up next to each other. The documents that exhibit consistent behavior are of special interest as they indicate to an observer that there is a reason they show consistent behavior. We then analyzed these documents in order to identify what about them is causing them to exhibit this consistent behavior.

We analyzed different examples on a case by case basis in Chapter 4 in order to determine different relationships that Doc2Vec captures. Each pair (or set) of documents analyzed is given a rating which indicates the certainty of the analyst having noticed similarities explaining the closeness (or lack thereof) of the documents. The rating is a numerical value from between 0 and 5, inclusive. 0 represents that the analyst is not confident why the specific behavior is exhibited and a 5 indicates that the analyst is confident why the behavior is exhibited. In the case that Doc2Vec associates documents based on human noticeable similarities between documents, we assert that it has performed well.

3.4.3 Criteria 3

The third criteria we stated was that "given a Doc2Vec model that has clustered documents according to their topcis by humans, is that model able to accept another brand new document with a topic that has been given to the model and cluster that document with the pre-existing documents of the same topic?". If Doc2Vec is truly learning general relationships among the different classes of documents, it should be able to take a new document that is decided to clearly belong to one of the classes and produce a document embedding for it that is near other embeddings of documents of the same class. In each case, we describe where and how these new documents were obtained.

For this criteria, we compare document embeddings using cosine similarity, which was mentioned in section 2.3.3. Doc2Vec has the ability to infer an embedding for a new document. Once this embedding has been inferred, we then check all of the other embeddings in the document to see which ones are closest to it using the cosine similarity. Once these have been obtained, we analyze how many of these resulting vectors are of the same class as the document we inferred an embedding for.

We assert that, at the very least, the first closest embedding to a new document embedding should be in the same class as the new document embedding. If Doc2Vec consistently places new documents in the appropriate classes by giving embeddings for them that are close to embeddings of other documents of the same class, we say that it is "good".

4 Results

In this section, a general overview of commonalities in the experiments that were run is presented followed by a more detailed look at the experiments and the collected data. We show results for each of the three criteria we have discussed for Doc2Vec separately in sections 4.2, 4.3, and 4.4.

4.1 Overview of Experiments

In this thesis, all experiments utilized the python programming language with the gensim[5] library. This library provides implementations of Word2Vec,

[5]https://radimrehurek.com/gensim/

Doc2Vec, and various other natural language processing algorithms.

Some variables and properties of the Doc2Vec models remained consistent throughout all experiments. The window size was fixed to be 5. The architecture used for all experiments was the PV-DBOW architecture, which is the architecture for Doc2Vec that is discussed in section 3.2. The dataset was fixed i.e. all Doc2Vec models that were trained were analyzing the same dataset discussed in a previous section. After training, Doc2Vec models were saved and used for analysis. That is, new models were not trained each time we ran analysis on them, unless otherwise noted. To be precise, when the models were trained in python, the objects corresponding to them were saved through process known as *pickling*[6]. This process is common in python and simply serializes data so that it can be unserialized later.

We note that the learning rate is continuously updated to approach a relatively small value throughout the training process for the implementation of Doc2Vec in the gensim library. This is a common technique in machine learning. As the neural network "learns" more, it should approach a local minimum of the objective function. Reducing the learning rate as the neural network trains helps to ensure that the parameter updates do not make the objective function "jump" over a minimum. If the learning is too high of a value, it is possible to move too far in one direction when taking steps to minimize the objective function, which can result in a minimization process that skips over local minima of the objective function.

The variables that were manipulated for our experiments were the document embedding size and the number of epochs for training a model. The document

[6]https://docs.python.org/3/library/pickle.html

embedding size is the number of dimensions that the resulting embeddings from Doc2Vec have. We experimented, generally, with document embedding sizes from 5 to 50, in increments of 5. We experimented with a number of epochs from 10 to 100, in increments of 5. Some results in these ranges are not discussed here if there were no noticeable differences. For instance, some models trained for 60 epochs compared to models trained for 100 epochs (with the same document embedding size) show no difference when evaluated against our criteria due to the relatively tiny changes to the document embeddings that happen at that point of learning. For each different set of parameters, a new model was trained from scratch.

The rest of this section contains the individual results for each of the three criteria.

4.2 Criteria 1

Recall that the first criterion stated was as follows: "The first criteria we stated was that for a large enough set of documents that are classified by a human into several distinct classes, to what extent does Doc2Vec learn document embeddings that mirror those (human) classification decisions?". We analyzed this criterion with an implementation of the k-means algorithm given by the scikit-learn[7] library in python.

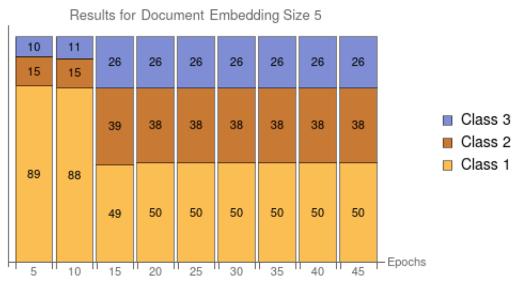
Figure 11 shows the results for running the k-means clustering algorithm on models with varying document embedding sizes and number of epochs for training. The important aspect to notice of the bar charts in the figures is that each document embedding size eventually reaches a state where the k-means clustering

[7]https://scikit-learn.org/

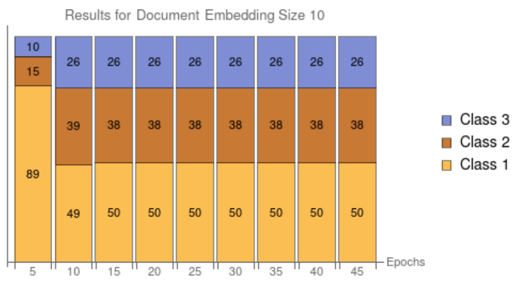
algorithms finds clusters of size 50, 38, and 26. Recall that there were 50 card game articles, 38 sports, and 26 dance documents. Note that it is not necessarily the case that the embeddings that the k-means algorithm identified as being in these clusters actually correspond to the classes of documents we have identified solely because the numbers line up. Upon further investigation of the clustering, however, we found that the k-means algorithm was asserting that the 50 card game articles, 38 sports articles, and 26 dance documents each belonged to their own cluster.

Note that in most models represented in Figure 11, the card articles are associated with cluster 1, the sports with cluster 2, and the dance articles with cluster 3. There is an exception in the model shown in Figure 11d. In the columns for epochs of 40 and 45, the sports belong to cluster 3 and the dance articles belong to cluster 2. This also happens in Figure 11e with epochs 35 and 40. Since the label for each cluster does not actually convey any meaning, this clustering is essentially equivalent to the other clusterings.

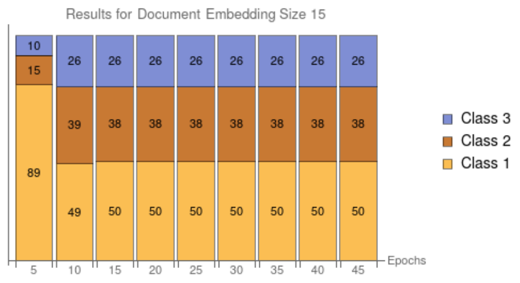
Before we assert that Doc2Vec has performed well at this criteria, we investigate this criteria a bit further. The data in Figure 11 show multiple examples of the k-means algorithm being evaluated on individual models. While this is promising, it is not yet conclusive. Recall from earlier discussions that the embeddings that are produced by Word2Vec and Doc2Vec are initialized to be random before training occurs. Thus, we investigate further by isolating examples from Figure 11 and running multiple trials of the k-means algorithm on Doc2Vec models with the same parameters but different random initializations. In particular, we choose a specific number of epochs from each of the sub-figures in Figure 11 and generate



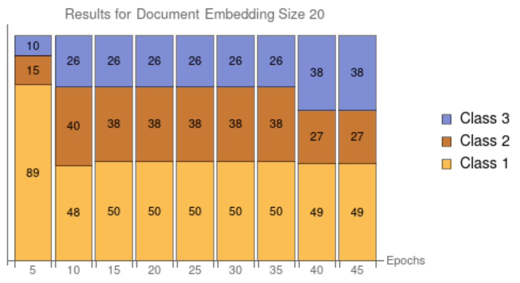
(a) Document Embedding Size 5



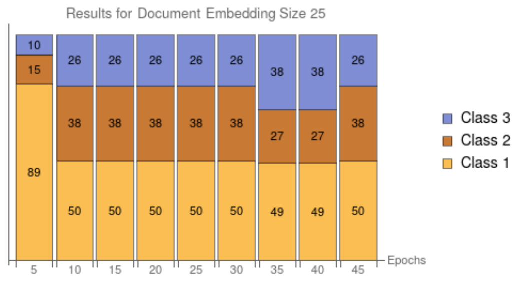
(b) Document Embedding Size 10



(c) Document Embedding Size 15



(d) Document Embedding Size 20



(e) Document Embedding Size 25

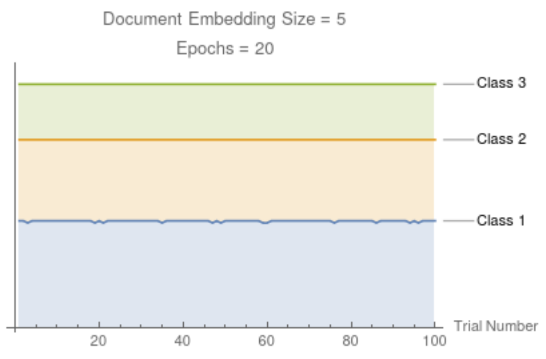
Figure 11: Results for Criteria 1 Experiments

multiple models trained for that number of epochs and a document embedding size determined by the sub-figure being considered.

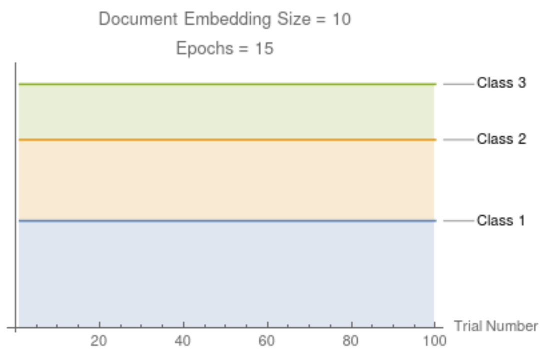
Figure 12 shows a graphical representation of the data obtained from running multiple trials with selected numbers of document embedding sizes and epochs to train a model. The selection of the number of epochs for each document embedding size was chosen such that they represent the largest number of epochs for which a model resulted in clusters with sizes equal to the number of documents in each predetermined class. 100 trials were run for each set of parameters with the models having different random initializations for each trial.

It can be deduced from Figure 12 that Doc2Vec generally clusters the documents according the number of documents we expect to be in each of the three classes. It is notable that class 1, which corresponds to the card game articles, is clustered fairly consistently into its own cluster as compared to class 2 and class 3 which correspond to sports and dance, respectively (except in a few cases). It is also notable the models with a document embedding size of 10, 15, and 25 performed much better on average than models with other parameters. Models with the other parameters have more discrepancies more often.

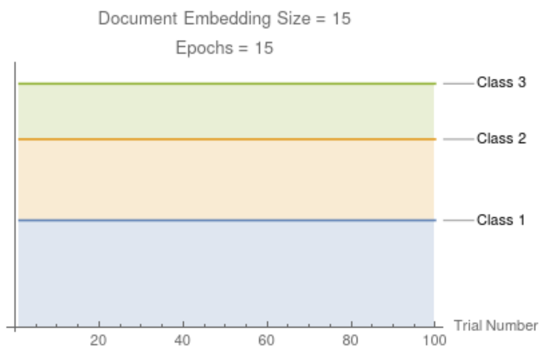
Most of the “mis-clusterings” occurs between the sports and dance documents. It is important to note that the “mis-clustering” of these documents is not necessarily a bad thing. Doc2Vec can and should be learning relationships between documents. It is not only meant to split documents according to their differences. It should also give us insight into which documents are similar. In this regard, it is an interesting phenomena that the sports and dance documents have a consistent “mis-clustering” across the board. This phenomena in particular will



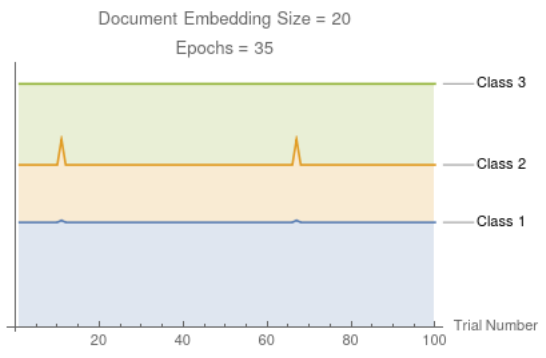
(a) Document Embedding Size 5 - Epochs 20



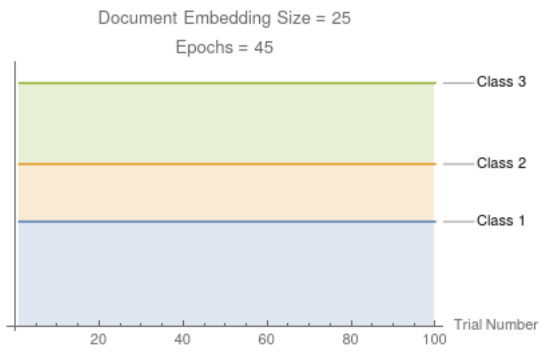
(b) Document Embedding Size 10 - Epochs 15



(c) Document Embedding Size 15 - Epochs 15



(d) Document Embedding Size 20 - Epochs 35



(e) Document Embedding Size 25 - Epochs 45

Figure 12: Results for Criteria 1 Experiments

be discussed more in Section 4.3.

Based on the results gathered, we conclude that Doc2Vec does perform very well on the dataset that we have. It clusters the document embeddings such that they are able to be recognized as three distinct clusters with a minimal amount of overlap. Based on our results from the supervised learning algorithms discussed earlier in this paper, this behavior is to be expected if Doc2Vec is learning topical information about the dataset. These results give evidence that Doc2Vec is recognizing and reflecting that there is a distinct difference between the three classes of documents.

4.3 Criteria 2

Recall that the second criterion stated was as follows: "For a large enough set of documents that Doc2Vec separates well into distinct classes, to what extent do the document embeddings learned by Doc2Vec capture inter-cluster relationships?". We analyzed this criterion by using an implementation of the t-SNE algorithm mentioned in the earlier section of this paper describing the evaluation criteria. We projected the document embeddings to 2D and then analyzed patterns in the projections.

Before discussing specific examples for this section, we first show what the projections look like in general and establish some common terminology that is useful for the discussion. Figure 13 shows an example of a 2D projection using the t-SNE algorithm. In this projection (and in all subsequent 2D projections), boxes with numbers in them represent 2D projections of the higher dimensional document embeddings. Original document embedding size along with the number

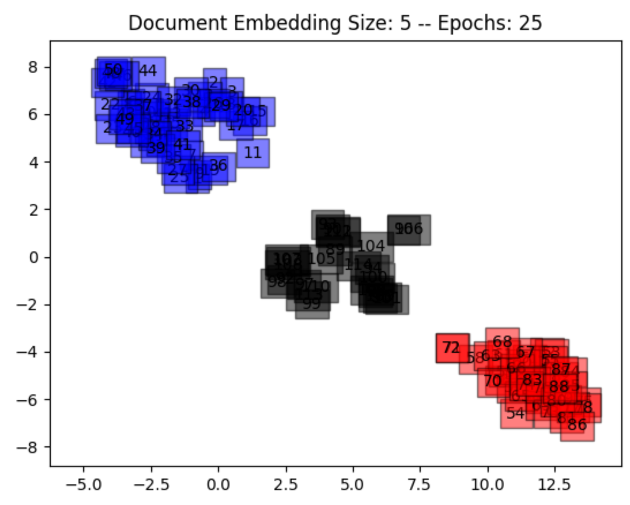


Figure 13: t-SNE Projection Example

of epochs that the corresponding was trained for is displayed at the top as a title for the plot. We use the convention that boxes colored blue represent card game articles, boxes colored red are sports articles, and boxes colored black are dance articles. The numbers inside of the boxes represent the index of the document in the corpus for the project.

Note that the bounds on the horizontal and vertical axes may not necessarily be consistent throughout all examples. The horizontal and vertical coordinates of points projected to 2D are relatively arbitrary. The t-SNE algorithm tries to keep the 2D projections somewhat close to the origin and therefore these bounds are not pertinent to the discussion.

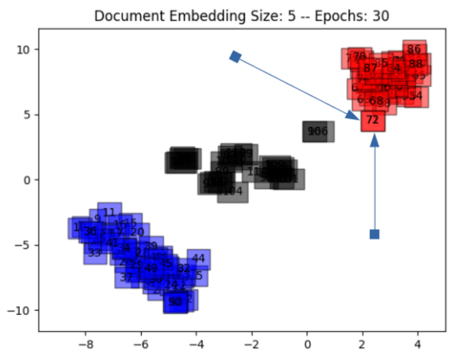
To discuss this criterion, we showcase specific examples in the subsections that follow. Pairs of two documents were chosen based on patterns that have been noticed in the 2D projections of the data and those documents were then analyzed by the author to determine what might have caused the pattern noticed. The author then assigned a score from 0 to 5 for each example where a higher score indicates that the author felt more confident in noticing what had caused the pattern.

4.3.1 Example 1

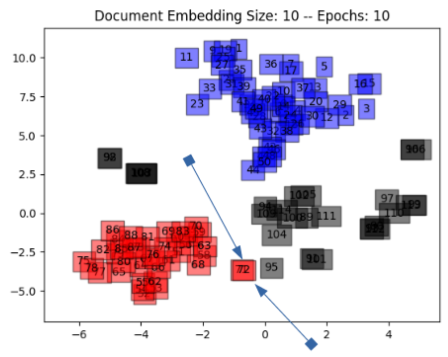
For the first example, we consider two sports articles. Figure 14 shows some projections for the data. Notice that in all these projections, there are two red boxes labeled with 71 and 72 that consistently overlap each other and are on the outside of the rest of the red boxes. This behavior is consistent throughout other projections.

The content of these two documents were analyzed. Upon reading the documents closely, the author found that out of the over 1000 words in each article, the first 462 were exactly the same. After loading the websites from where these articles were retrieved, it is true that this is how the articles were added to the website. That is, there was not an error in the processing of the articles or in the retrieval of them from the website. The articles do, in fact, share a significant portion of their text.

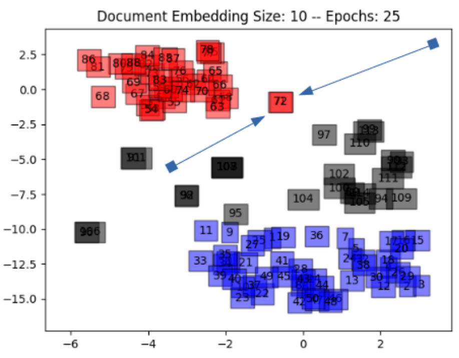
While this explains why those document embeddings would be projected to 2D in relatively the same position, it does not necessarily explain why these two document embeddings were consistently projected to be on the outside of the sports article clusters. Upon further examination of these documents, it was found that they both discuss a tweet made by a general manager of the Houston Rockets. This particular tweet happened to offend many people in China because it showed support for Hong Kong anti-government protesters. The articles discuss ramifications of this and generally speak about sports in China and the global



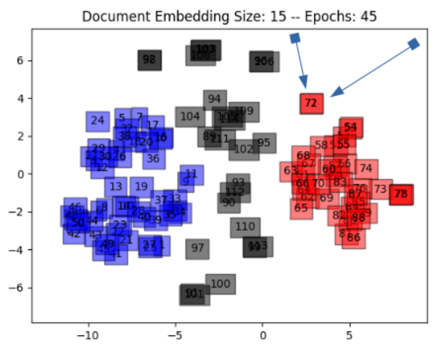
(a) Document Ebmedding Size 5 - Epochs 30



(b) Document Embedding Size 10 - Epochs 10



(c) Document Embedding Size 10 - Epochs 25



(d) Document Embedding Size 15 - Epochs 45

Figure 14: Results for Example 1

influence of the NBA. Knowing this, it makes sense why these documents would show up on the outside of clusters in these 2D projections. The other sports articles in the dataset discuss sports in general (anywhere from general sports news to specific players) and do not make mention of politics or Chinese sports.

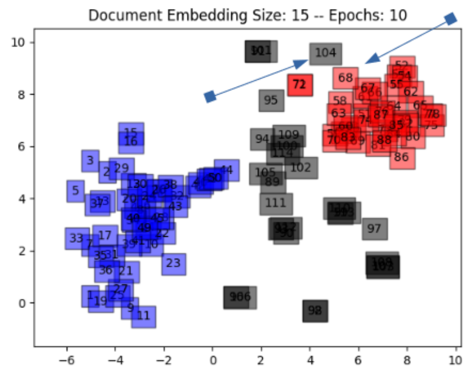
Because of the inherent difference between these two documents and the rest of the dataset along with their overlap in content, the author gives this example a rating of 5.

4.3.2 Example 2

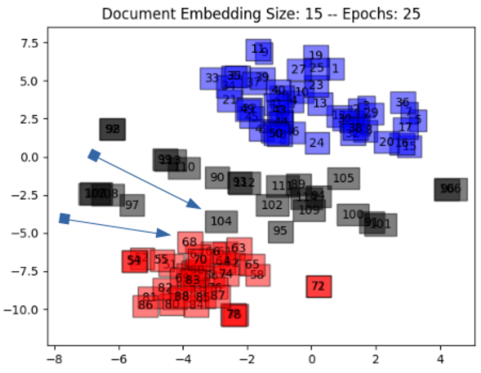
For the second example, we consider a sports article and a dance paper. Figure 15 shows projections for the data. In these projections, the red box with a label of 68 consistently shows up close to the black box labeled 104. This behavior is not as consistent throughout other projections as the behavior was for the previous example. However, it interesting to investigate the possibility of these documents being related in some form.

Upon analyzing these documents the author was unable to determine any key factors that may relate the two documents. The sports article with the label of 68 generally discusses some friends in the NFL and their relationship on and off the football field. The dance document with the label of 104 discusses personal and background stories of different dancers across India. The only somewhat feasible connection between these documents that the author noticed is that they both discuss personal stories and relationships of people in general. The author did not feel that this constituted a very deep or inherent connection.

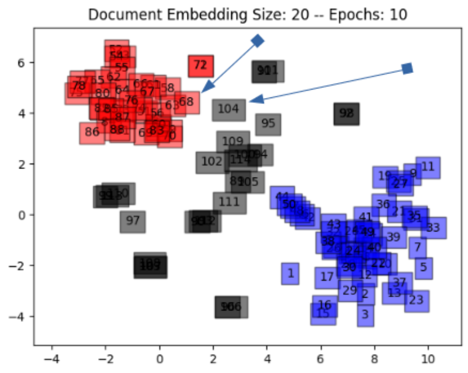
Because of the lack of obvious connection between these two documents, the



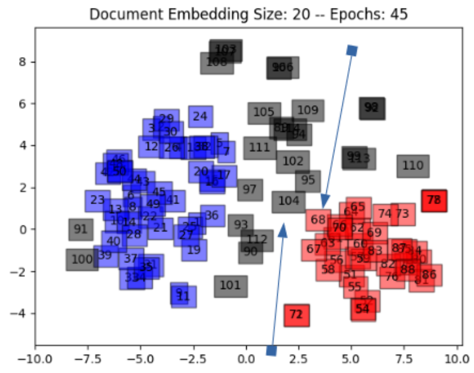
(a) Document Ebmedding Size 15 - Epochs 10



(b) Document Embedding Size 15 - Epochs 25

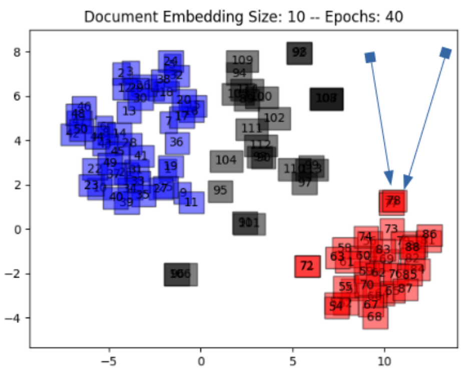


(c) Document Embedding Size 20 - Epochs 10

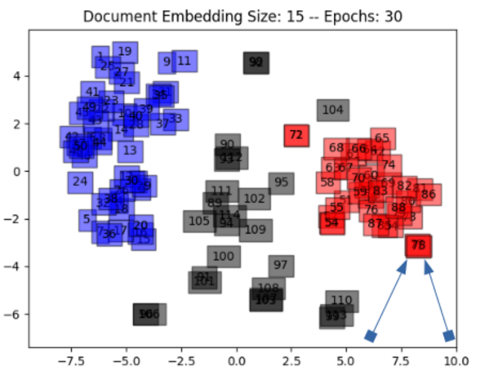


(d) Document Embedding Size 20 - Epochs 45

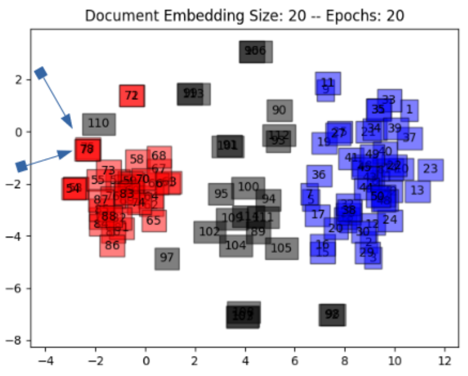
Figure 15: Results for Example 2



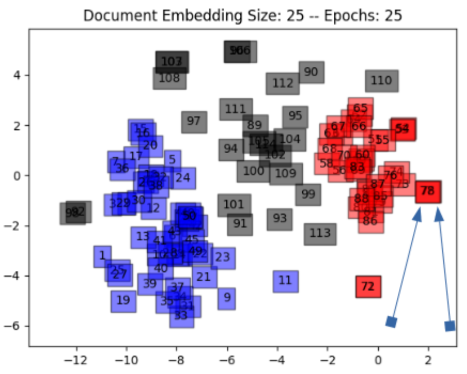
(a) Document Ebmedding Size 10 - Epochs 40



(b) Document Embedding Size 15 - Epochs 30



(c) Document Embedding Size 20 - Epochs 20



(d) Document Embedding Size 25 - Epochs 25

Figure 16: Results for Example 3

author gives this example a rating of 1. While it is possible to somewhat stretch the meaning of "similar" in this case to make these two documents seem close in topical information, they are not easily determined to be similar.

4.3.3 Example 3

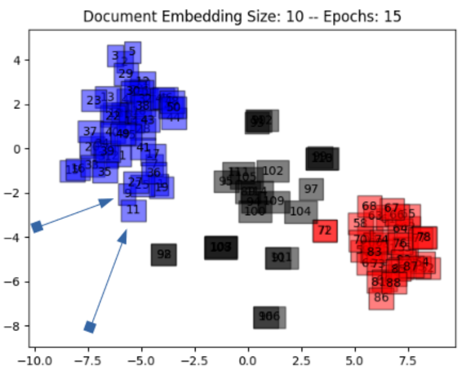
For the third example, we consider another pair of sports articles. Figure 16 shows projections for the data. In these projections, the two red boxes with labels 78 and 77 consistently show up on top of each other.

Upon reviewing these sports articles, it was found that they consist of exactly the same text. They are, in fact, the exact same article. The first thing to note is that these articles actually did come from different URLs. The fact that the articles are exactly the same is an interesting coincidence. Nevertheless, this explains why the projections always put the embeddings for these two documents relatively close in 2D.

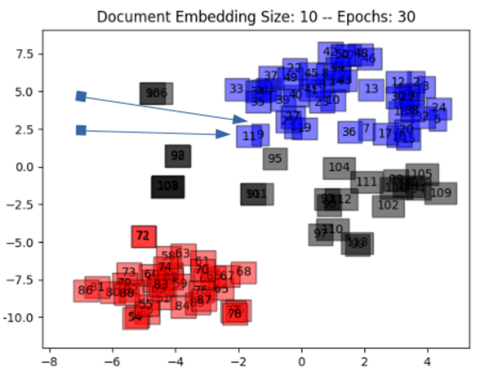
The more interesting thing to note here is that Doc2Vec actually does put things that are exactly alike in very similar places in the output space. That is, for documents that are extremely similar in content, Doc2Vec produces document embeddings that are extremely similar. This is not necessarily surprising; it certainly would make sense for a human to say that two documents that are actually the same document should be exactly similar in any other non-textual representation of those documents.

It is interesting, however, to note that Doc2Vec does perform exactly as we might expect it to in this situation. Recall from the Approach section that Doc2Vec does not actually compare documents. It "learns" document embeddings based on the words in the document. It never updates documents based on their relation to other documents. Thus, it is interesting that even though it is performing stochastic gradient descent and it is not necessarily comparing documents, it is still able to converge to similar document embeddings for documents that are exactly alike.

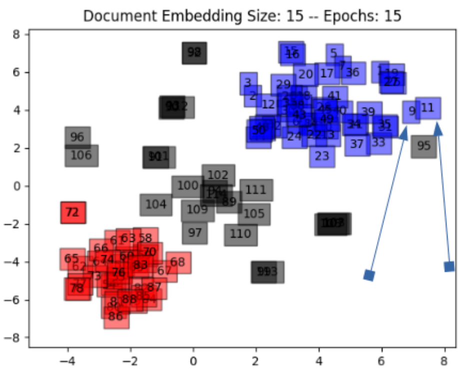
The author gives this example a rating of 5. It is quite evident why two documents that are exactly the same would have extremely similar document embeddings and projections into 2D.



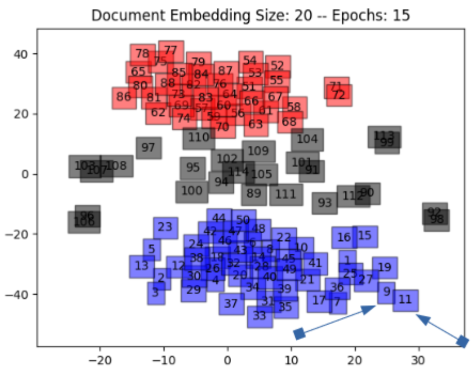
(a) Document Ebmedding Size 10 - Epochs 15



(b) Document Embedding Size 10 - Epochs 30



(c) Document Embedding Size 15 - Epochs 15



(d) Document Embedding Size 20 - Epochs 15

Figure 17: Results for Example 4

4.3.4 Example 4

For the fourth example, we consider two card game articles. Figure 17 shows projections for the data. In these projections, notice that the blue boxes labelled 9 and 11 consistently show up next to each other. This activity is consistent throughout other projections as well.

The documents were analyzed that corresponded to the boxes with labels 9 and 11. Upon examining the documents, it was not entirely apparent at first why they might be considered similar. However, when compared to the rest of the card game articles, the similarities to each other and the differences to other documents became apparent.

The card game that these articles were written about is casual, but it also has many tournaments associated with it. There are competitive players in the game who compete for prizes, money, etc. The documents with labels 9 and 11 actually discuss the tournament side of the game. They address mentality during tournaments, the atmosphere of tournamnets, and give tips on succeeding in them. That alone helps to justify why the documents are consistently similar; they are discussing the same sort of information. When comparing them to the rest of the card game articles it becomes clear why they tend to be on the outside of the clusters that are projected into 2D.

The other card game articles deal more with analysis of specific cards in the game and how to best understand the impact that a certain card or collection of cards has on the card game in general. They often show different decks built from these cards and discuss strategies involving the decks. The documents with labels 9 and 11 do not discuss this sort of information. While all the documents talk about the same card game, the two documents in question actually represent a different part of the game. Namely, they talk more about the aspect of competing in the game rather than specific strategies or cards. It is interesting that these may be the things that Doc2Vec is recognizing.

The author gives this example a rating of 4. While it is not as clear cut as previous examples why these documents would overlap or why they would be different from other documents in the same class, the author feels confident that the documents do show similarities with each other and clear differences from other card game documents.

4.4 Criteria 3

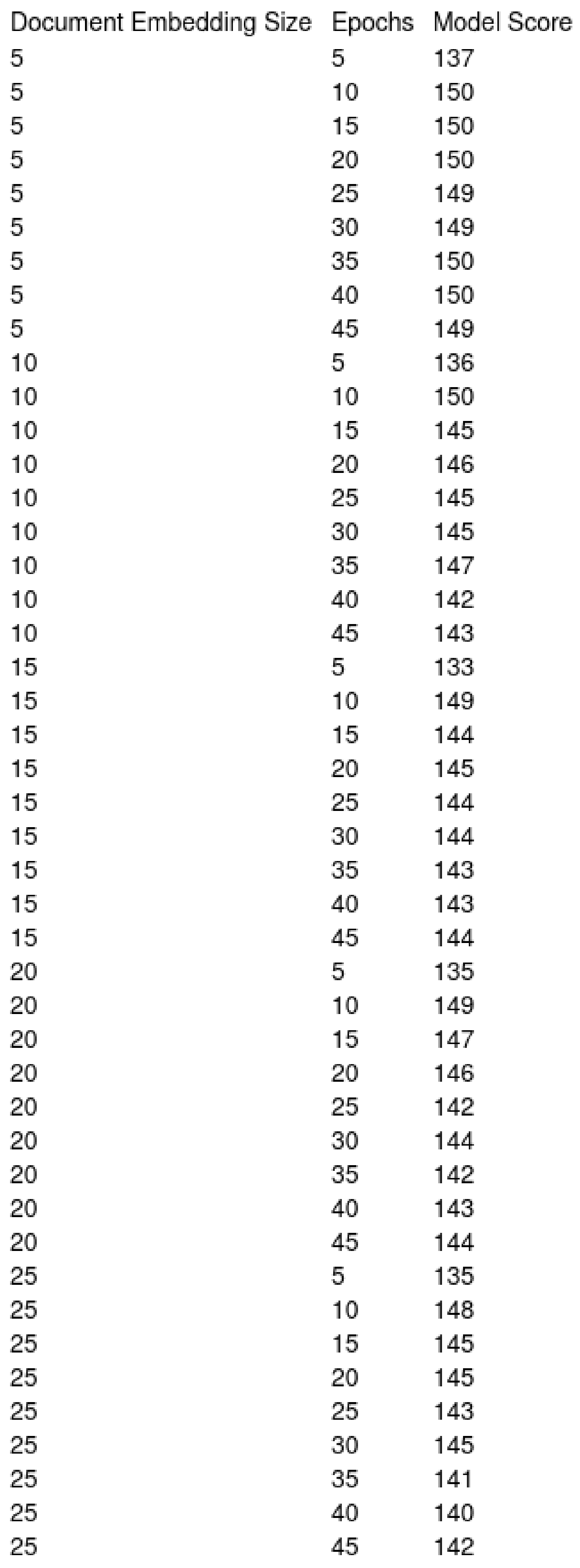
Recall that the second criterion was as follows: “Given a Doc2Vec model that has clustered documents according to their topics determined by humans, is that model able to accept another brand new document with a topic that has been given to the model and cluster that document with the pre-existing documents of the same topic?”. We analyzed this criterion by selecting 5 new documents from each predetermined topic in the dataset (for a total of 15 new documents) and then calculated the document embedding for each of these new documents based on the Doc2Vec models we had trained. We then analyzed the document embedding for each of these new documents by calculating the top 10 nearest other document embeddings in each Doc2Vec model. This calculation was done using the built in similarity functionality in gensim which implements the cosine similarity measure.

New documents were given a score between 0 and 10 representing how many of the 10 most similar documents to it were of the same predetermined class. This was done for each document and for each Doc2Vec model that we had trained to analyze. Each model was then given a score out of 150 representing how well it classified new document embeddings. This score was simply a sum of the rating for each document. Thus, since there are 15 documents each having a possible score between 0 and 10, a Doc2Vec model with a score 150 represents that it classified the new documents very well.

The analysis of this criteria is motivated by the fact that new documents that Doc2Vec has not seen should be represented with an embedding that shows up in a cluster that Doc2Vec has already defined. The ability of Doc2Vec to place new documents close to other documents of the same topic is important so that we may see if Doc2Vec is actually “learning” relationships between documents or if it is simply separating documents because of differences between them.

Figure 18 the peak of the scores tend to vary between different document embedding sizes. Models with a document embedding size of 5 tend to have the results that are closest to 150 out of all the models tested. In fact, models with the embedding size of 5 are the only models (aside from one model with embedding size 10) that achieve a score of 150. In this sense, these models perform the best. It is interesting to note that models rarely dropped below a score of 140. This generally only happens when the number of epochs that a model trained for was equal to 5. It is also notable that models that trained for 10 epochs always achieve the highest score out of all other models with the same document embedding size.

Based on the results gathered here, we conclude that Doc2Vec performs well on our dataset when appropriate training conditions are chosen. Recall from the analysis of the first criteria that most models were able to perform well in that task which exhibited the fact that Doc2Vec is able to cluster things according to how humans recognize content. This criteria is markedly different in that it demonstrates how, with the correct training configurations, Doc2Vec can not only make these clusters, but it can see new data and cluster it appropriately.



Document Embedding Size	Epochs	Model Score
5	5	137
5	10	150
5	15	150
5	20	150
5	25	149
5	30	149
5	35	150
5	40	150
5	45	149
10	5	136
10	10	150
10	15	145
10	20	146
10	25	145
10	30	145
10	35	147
10	40	142
10	45	143
15	5	133
15	10	149
15	15	144
15	20	145
15	25	144
15	30	144
15	35	143
15	40	143
15	45	144
20	5	135
20	10	149
20	15	147
20	20	146
20	25	142
20	30	144
20	35	142
20	40	143
20	45	144
25	5	135
25	10	148
25	15	145
25	20	145
25	25	143
25	30	145
25	35	141
25	40	140
25	45	142

Figure 18: Model Scores

5 Conclusion and Future Work

Based on the results shown in Section 4, we conclude that Doc2Vec is well suited for the tasks we want it to perform. We believe, though, that the fact that Doc2Vec did not consistently achieve everything to be considered good under each criteria is an encouraging thing. We do not necessarily want Doc2Vec to perfectly cluster documents such that there is no ambiguity in topics. Part of our interest in the way that humans classify topics is that it is not necessarily completely distinct; there are relationships between different topics of information even if they are somewhat impercetible at first.

The results of this project can be used as a grounds to explore deeper, more interconnected data in the future. We have established resonable criteria to evaluate Doc2Vec models and interpret the output of the model in terms of how well topics among the data are learned. Further analysis can be done on bigger datasets where the data has already been partitioned into topics by humans and our tests here can be replicated on those datasets.

In the future, the author would like to explore different ways of learning document embeddings. Doc2Vec is not the only technique to accomplish this and it would be very fruitful to analyze other unsupervised methods of doing this. Work must also be done on the issue of viewing these resulting embeddings in 2D. The success of the overarching ambition of this project hinges on the ability to meaningfully project document embeddings of high dimensionality into a 2D space. This is a separate project and the t-SNE algorithm was considered good enough for this thesis. It may be that this algorithm is good enough in the long run, but more analysis needs to be done on this front.

Analysis may also be done on modifying the Doc2Vec technique. It may be reasonable to tag data with specific topics and add this as a training parameter to the Doc2Vec learning algorithm. This would, in essence, for Doc2Vec to become a more supervised method of learning. It would be of much interest to see if the results shown in this paper can be replicated after making this modification to Doc2Vec.

Another interesting point of exploration would be to have Doc2Vec produce embeddings for a set of documents that have all been prescribed a single topic by a human. It would be worthwhile to see if Doc2Vec creates any sort of clusters in that data using some of the criteria we've laid out here to evaluate it. For instance, we could do the same sort of analysis with only the sports articles we have. It would most likely be necessary to train on more than just 38 articles, but it would be interesting nonetheless.

References

[1] Mikolov, Tomas; Sutskever, Ilya; Chen, Kai; Corrado, Greg; Dean, Jeffrey *Distributed Representations of Words and Phrases and their Compositionality*, NIPS Proceedings, (2013)

[2] Le, Quoc; Mikolov, Tomas *Distributed Representations of Sentences and Documents*, CoRR, (2014)

[3] Bolukbasi, Tolga; Chang, Kai-Wei; Zou, James; Sligrama, Venkatesh; Kalai, Adam *Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings*, NIPS Proceedings, (2016)

[4] Kibriya, Ashraf M.; Frank, Eibe; Pfahringer, Bernhard; Holmes, Geoffrey *Multinomial Naive Bayes for Text Categorization Revisited*, AI 2004: Advances in Artificial Intelligence, 2005, Springer Berlin Heidelberg, pp. 488-499

[5] Su, Jiang; Sayyad-Shirabad, Jelber; Marwin, Stan *Large Scale Text Classification using Semi-supervised Multinomial Naive Bayes*, 28th International Conference on Machine Learning Proceedings, (2011)

[6] Hsu, Chih-Wei; Lin, Chih-Jen *A Comparison of Methods for Multi-class Support Vector Machines*, IEEE Trans. Neural Netw. 13, 2, pp. 415–425 (2002)

[7] Fung, Glenn M.; Mangasarian, O. L. *Multicategory Proximal Support Vector Machine Classifiers*, Machine Learning, 59, 1, pp. 77–97 (2005)

[8] https://commons.wikimedia.org/wiki/File:Svm_separating_hyperplanes_(SVG).svg

Appendix

The articles discussed in the criteria 2 results section are available at the following links, organized by the document number that was specified. They are listed in the order they were discussed.

71	→	http://www.espn.com/espn/wire?section=nba&id=27788802
72	→	http://www.espn.com/espn/wire?section=nba&id=27780769
68	→	http://www.espn.com/espn/wire?section=nfl&id=28044115
104	→	https://ntnuopen.ntnu.no/ntnu-xmlui/bitstream/handle/11250/2561634/SaraAzzarelli-Choreomundus.pdf?sequence=1
77	→	http://www.espn.com/espn/wire?section=nba&id=28046151
78	→	http://www.espn.com/espn/wire?section=nba&id=28046097
9	→	https://www.coolstuffinc.com/a/jimdavis-08232019-exploring-different-ways-to-play-on-a-digital-playground
11	→	https://www.coolstuffinc.com/a/jimdavis-08162019-the-future-of-magic-esports-and-organized-play