Issue2vec: Legal Issue Embeddings Using Citegrams

Issue2vec: Legal Issue Embeddings Using Citegrams

Owen Murphy

A Thesis in the Field of Software Engineering

for the Degree of Master of Liberal Arts in Extension Studies

Harvard University

March 2022

Abstract

Most case law decisions are divided into discrete sections that address specific legal issues. But even though those sections are generally independent from one another, unrefined machine learning and natural language processing techniques treat those sections as a single document. Moreover, caselaw decisions contain citations to precedential caselaw decisions. But the tokens comprising those citations provide minimal value to the machine learning process.

This project explores these observations by creating corpus of documents where each document is a specific section from a case law decision, and where each citation is replaced with a unique n-gram or, "citegram." The results demonstrate that isolating specific caselaw sections facilitates document similarity operations and that citegrams ably capture semantic information.

Author's Biographical Sketch

Owen Murphy is an attorney licensed in both Massachusetts in Rhode Island. He focuses on technology and privacy issues. He began studying software engineering in 2013. In 2016, he began his master's studies at the Harvard Extension School, earning a graduate certificate in cybersecurity.

Dedication

This project is dedicated to my parents, Owen and Mary Beth Murphy. I love you and am forever grateful for your support.

Table of Contents

Chapter I

Introduction

Caselaw decisions and their features. In the case of *State v. Rivera*, 221 A.3d 359 (R.I. 2019), a defendant was convicted of several crimes including burglary and using a firearm. He appealed his convictions to the Rhode Island Supreme Court. There, he argued that (1) a recorded phone call and its associated transcript were improperly introduced at the trial and (2) the trial justice should have declared a mistrial because, during the pretrial discovery process, his attorney did not receive all the necessary preparation materials. The Court issued a caselaw decision denying both claimed issues. For each issue, the Court expressly set forth the issue, recited the governing law, and applied that law to the facts of the dispute.

In other words, *Rivera* was a single caselaw decision that addressed several discrete legal issues. The question of whether the recording should have been introduced was an evidentiary issue, grounded in the Rules of Evidence. The discovery issue was governed by the Rules of Criminal Procedure. And while both issues were subject to the Rhode Island Supreme Court's previous decisions, the particular precedent applicable to each issue was itself discretely related to each issue. For example, when the Court analyzed the transcript issue, it relied on *State v. Ahmadjian*, 428 A.2d. 1070 (R.I. 1982). The *Ahmadjian* case involved the admission of transcripts and thus shed light on the first issue, but it was completely irrelevant to the discovery issue.

Like *Rivera*, many caselaw decisions address multiple discrete legal issues, which are totally unrelated besides the fact that they arose during the same case. To an attorney, the presence of multiple unrelated issues within the same caselaw document is unremarkable. It is routine for an attorney to find a case and then home in on the pertinent issues. But the fact that multiple discrete issues are frequently contained in one caselaw document is an impediment for machine learning.

Artificial Intelligence and machine learning. Machine learning falls within the field of Artificial Intelligence and is a fundamental component of the burgeoning data science industry. Machine learning attempts to imitate the way humans learn. It involves applying statistical algorithms to data, in order to make classifications, predictions, and to unearth nonobvious patterns and features. Machine learning itself contains several subsets: Supervised Machine Learning, Unsupervised Machine Learning, Semi-supervised Machine Learning, Reinforcement Learning, and Deep Learning. (IBM, Machine Learning).

In Supervised Machine Learning, a model is trained using labeled data. When the training is complete, the model can be used to classify new, never-before-seen data. In Unsupervised Machine Learning, there are no predetermined labels. The algorithms analyze and cluster data by extracting patterns and features, with limited need for human assistance. Given that capability, Unsupervised Learning lends itself to tasks such as exploratory data analysis and pattern recognition. Semi-supervised Learning offers a happy medium between Supervised and Unsupervised learning. It uses a small set of labeled data to guide classification and feature extraction form a larger set of unlabeled data. Reinforcement Learning is a behavioral machine learning model, which learns over

time by a process of trial and error, instead of by sample training data. (IBM, Machine Learning).

In Deep Learning, which is the approach used in this paper, a model is trained using a complex artificial neural network. It is possible for one of the previously discussed machine learning techniques to leverage a neural network. (IBM, Supervised Learning). Deep learning differs because it uses a deep neural network, that contains at least three layers. By adding layers, accuracy is generally improved. (IBM, Deep Learning).

Neural networks are comprised of several layers, including an input layer, one or more hidden layers, and an output layer. Each layer has a collection of nodes, or neurons, that interact with all the neurons in all the other layers. The input layer is where data enters the network. The output layer emits the output of data that has passed through the network. The hidden layer(s) lie between the input and output layers. They are responsible for extracting features from the data and deriving the relationships between the input and output. (Ravichandiran, 2019).

Similar to the human brain, the actual learning occurs through the union of forward and backpropagation. (Ravichandiran, 2019). During the process of forward propagation, data enters the input layer and is propagated through the hidden layers to the output layer. At each layer, the data is adjusted by randomly set weights and biases. It is then passed to the nodes within the next layer, where it is input to the activation function of a particular node. If a node's activation function is satisfied, the node passes the data to the next layer of the network where it is once again adjusted by a weight and bias and subjected to an activation function. (IBM, Machine Learning). The output generated by

the network is assessed by a loss function. Depending on the results of the loss function, the randomly initialized weights are adjusted through the process of backpropagation and gradient descent. Working together through many iterations, forward and backpropagation are used to optimize the weights within the network. (Ravichandiran, 2019).

How machine learning applies to caselaw in the scope of this paper. This machine learning discussion is basic and general, but it helps illuminate how the format of caselaw decisions can frustrate the machine learning process. When treating the *Rivera* case as a single document, text from multiple discrete issues is blended in a single instance of data. As a result, the data includes multiple discrete issues, which are unrelated to each other and unlikely to appear together in other caselaw documents. Given that the goal of machine learning is to extract features and patterns, that arrangement is suboptimal.

The presence of citations poses similar impediments. To an attorney, citations are informative features that add semantic value and (hopefully) legitimize the propositions that they support. For example, if two caselaw decisions both cite to the same precedential case, there is a greater probability that both cases are addressing the same issue. But to the learning machine, citations are merely a series of word tokens. Because those tokens are ubiquitous over a collection of caselaw documents, they do not necessarily form a data feature that can be used to optimize a model.

The outline of this paper. This project ventures to probe these barriers by transforming caselaw documents into section documents and replacing citations with n-grams that are globally unique to a given caselaw decision. The next chapter presents

background information about caselaw decisions, with particular focus on sections and citations. It then presents a history of the machine learning advancements that inspired this project. The chapter concludes with a discussion of the motivations and goals of this project. In chapter three, discussion turns to the design of the system that was developed to achieve those goals. Chapter four details how the system was implemented and used to produce three different models. Those models include the Base Model, which was trained using entire caselaw documents; the Section Model, which was trained using specific sections from caselaw documents; and the Citegram Model, which was trained using the same sections, but with specific n-grams (citegrams) in place of citations.

In chapter five, I use these models to conduct several experiments and present the results and evaluations. I conclude that models trained using specific caselaw sections generally outperforms those trained on entire caselaw decisions. I also conclude that once transformed into citegrams, citations are able capture the semantic meaning of the cases that they represent. In the concluding remarks, I suggest potential avenues of future research.

Chapter II

Background

This section provides background information on the caselaw features and machine learning techniques that inspired this project. The first section describes the relevant caselaw features. The second section catalogs prior work in the fields of natural language processing and machine learning. On the bases of those discussions, this section concludes with the motivations and goals of this paper.

2.1 Relevant caselaw features

Caselaw decisions are frequently divided into specific sections. They also contain citations to other relevant caselaw decisions. These two features are key to this project.

2.1.1 Sections and section headings

By rule and in practice, legal writings are frequently divided into sections. Sections can be thought of as a caselaw document's constituent objects. There are frequently recurring major sections, which typically include a procedural history section, a facts section, an analysis section, and a conclusion. At least some courts require parties to submit briefs and memoranda including most or all these sections. For example, Rule 16 of the Rhode Island Supreme Court's Rules of Appellate Procedure requires a brief to contain procedural history, facts, legal analysis, and a conclusion.

Sections are often denoted by descriptive headings and/or identifying symbols. For example, the sections of a legal brief or caselaw decision are likely to be structured along the lines of:

I. Procedural History
II. Facts
III. Analysis
IV. Conclusion

The analysis section is the meat and potatoes of a caselaw document. It sets forth the applicable law and applies it to the specific instance of facts presented in a case. The analysis section itself follows certain conventions. Typically, the analysis section articulates the legal issue, discusses the law applicable to that issue, compares that law to an instant set of facts, and ends with a conclusion on that issue.

If there are multiple issues in a case, the analysis section is likely to contain distinct subsections which address those issues. Those subsections are themselves often identified by headings and symbols. Opposed to the major sections, these subsections frequently include a uniquely descriptive title. For example, the *Rivera* case discussed in the introduction contains the following structure:

I. Facts and Travel
   A. The Evidence
   B. The Motion for a Mistrial

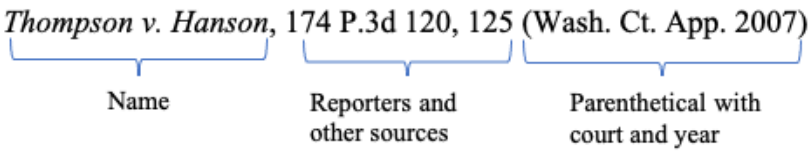
II. Discussion
   A. Admission of the ATF Call Recording and Transcript
      1. Standard of Review
      2. Analysis
   B. Motion for a Mistrial
      1. Standard of Review
      2. Analysis
   C. Motion for a New Trial

III. Conclusion

In this example, the major sections are represented by roman numerals and descriptive headings. At the next hierarchy level, capital letters and descriptions are used to identify the sections. The descriptions in the second level, particularly the description attending to section "A.", are unique descriptions of the events in the *Rivera* case. At the third level, numbers are used to identify sections. In this example, the descriptions of those sections – Standard of Review and Analysis – are non-unique frequently recurring descriptions. That is not true of every case, however. Depending on how the sections are structured, lower-level sections may contain unique descriptions.

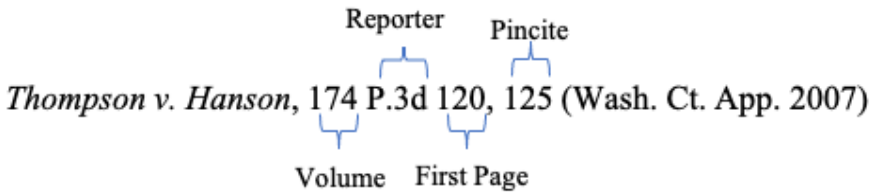
2.1.2 Citations

A citation is a reference to another legal document. Citations reference different types of legal documents including statutes, court rules, law review articles, and other types of secondary sources. This project focuses on the type of citation that most frequently adorns the body of caselaw decisions, case citations. A case citation includes three elements: (1) the name of the case, (2) reporters and other sources, and (3) a parenthetical indicating the court and year of the decision. (The Blue Book, 2020, p. 11):

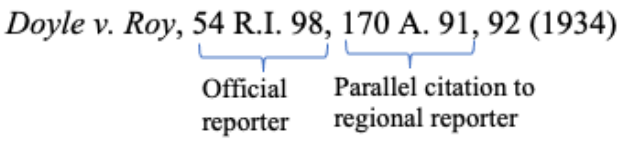


The name refers is the title of the case, which is usually the parties to the dispute. (The Blue Book, 2020, p. 97). In some instances, the name might consist of a different description, for example, in rem jurisdiction cases begin with the phrase, "*In re*." [The Blue Book, 2020, p. 98). Reporters are series of books that contain caselaw decisions. In the United States, there are commercially published regional reporters and official state

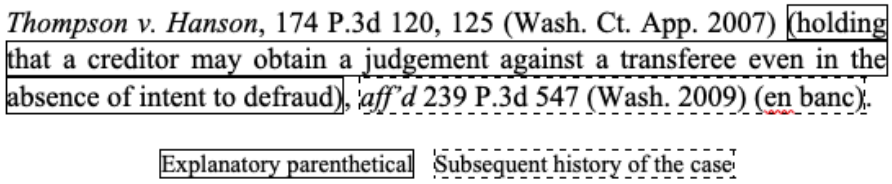
reporters. The reporters and other sources field indicates the published or unpublished source where a case can be found, including the identity of a specific reporter and the volume and page numbers. (The Blue Book, 2020, pp. 11, 103). The first number in this field is the volume of the reporter that contains the case. The second number refers to the first page of the case within the reporter. The first page number may be followed by a pincite, which refers to the specific page that the citation is referencing:



The rules of a particular jurisdiction dictate what reporter should appear in a citation. Some states require that citations reference both regional reporters and state reporters. For example, a state's rules might require a citation to first list the official state reporter, followed by a "parallel citation" to a regional reporter. (The Blue Book, 2020, p. 103). As a result, some citations include references to multiple reporters:



A case citation might also include two additional elements: (4) explanatory parenthetical information, and/or (5) the prior history or subsequent history of a case. (The Blue Book, 2020, pp. 11, 95):



To this point, the discussion has exclusively focused on full citations. There are also short citations. Once a full citation to authority has been provided, subsequent references to that authority can take the form of short citations. (The Blue Book, 2020, p. 16). A short citation typically abbreviates the name of the case and the reporter, and includes the word "at" followed by a pincite to the specifically referenced page(s):

Thompson, 174 P.3d at 125.

Short citations can also include parallel citations:

Doyle, 54 R.I. at 70, 170 A. at 91.

When referring to an immediately preceding citation, a citation can be abbreviated with the token "*Id.*" (The Blue Book, 2020, p. 17). On its lonesome, "*Id.*" refers to the same pincite as the prior citation. To refer to a different page within the immediately preceding authority, the word "at" and the particular pincite are added:

> "The rights of the targets of the investigation are deserving of consideration and cannot be overlooked." *Mora,* 821 F.2d at 869. That is why "any doubts about the integrity of the evidence should be laid at law enforcement's doorstep." *Id.* at 868. "It would be illogical (and unfair) to ask an accused to prove affirmatively that tampering has occurred." *Id.*

A citation can be inserted into the text of a document in one of two ways. (The Blue Book, 2020, p. 4). Most typically, citations appear in citation sentences that follow a substantive sentence. One citation sentence may contain multiple citations separated by semicolons:

> The U.S. Supreme Court has the power to invalidate statutes that are repugnant to the U.S. Constitution. *Marbury v. Madison*, 5 U.S. 137, 177-79 (1803); *Fletcher v. Peck*, 10

> U.S. 87, 139 (1810); *Dred Scott v. Sanford*, 60 U.S. 393, 449 (1856).

Where a citation only relates to part of a sentence, the citation can appear as a citation clause. Citation clauses are set off from the text by commas and immediately follow the proposition to which they relate:

> The Supreme Court adopted a broad reading of the Commerce Clause during the New Deal, *Wickard v. Filburn*, 317 U.S. 111, 128-29 (1942), though in recent years the Supreme Court has reigned in its broad reading somewhat, *United States v. Lopez*, 514 U.S. 549, 624 (1995).

A citation can also be qualified by a signal, which is a shorthand message that alerts the reader to the relationship between a proposition and the source of authority for that proposition. For example, the signal "*See*" alerts the reader that the cited authority supports, but does not directly state, a proposition. Other signals include *See, See also, Cf., Compare, E.g., Accord, Contra, But see, But cf.,* and *See generally.* (The Blue Book, 2020, p. 5)

2.2 Prior work in natural language processing and machine learning

Several milestone achievements in the field of natural language processing and machine learning are germane to this project.

2.2.1 Bag-of-words and bag-of-n-grams

To apply neural network methods to text-based data, the text must be transformed into a fixed-length vector representation. At the genesis, vector transformation was achieved by applying the bag-of-words and bag-of-n-grams approaches. (Harris, 1954).

The bag-of-words method produces a vector that captures how frequently specific words occur within a document. It is based on the intuition that documents containing similar proportions of the same words are themselves likely to be similar. The same intuition underlies the bag-of-n-gram approach. An n-gram is a sequence of *n* tokens, which is processed so that it appears as a single word within the vocabulary. By converting a frequently occurring sequence of tokens into a single token, a new word is included within the bag-of-words.

To enhance computational efficiency, the words within the vocabulary can be subjected to preprocessing techniques, such as removing stop words, fixing misspelled words, and stemming. By thus decreasing the vocabulary, vector sparsity and computational overhead are reduced. (Le & Mikolov, 2014). Either method can also be enhanced by term frequency – inverse document frequency ("tf-idf"). In tf-idf, words and n-grams are weighted based on the number of times they appear in a document. That weight is offset by the number of documents in the corpus that contain the word. (Le & Mikolov, 2014).

Both bag-of approaches are limited by their inability to capture semantic meaning. The approaches simply tally the words within a document, with no consideration of word order. As a result, semantic meaning is discarded. (Le & Mikolov, 2014). Take for example, the sentences "The old man drove the car." and "The man drove the old car." Despite their distinct meanings, each sentence produces the same vector.

By adding n-sequences of words into the mix, the bag-of-n-grams approach preserves some semblance of word order. But it only does so in short spurts and leaves a lot to be desired. Once again, documents composed of the same words and/or n-grams

produce the same vector representations, even where those sentences have entirely different meanings.

Like bag-of-words then, bag-of-n-grams fails to capture the "distances between the words." (Le & Mikolov, 2014). "This means that words 'powerful', 'strong' and 'Paris' are equally distant, despite the fact that semantically, 'powerful' should be closer to 'strong' than 'Paris.'" (Le & Mikolov, 2014). Bag-of-n-grams also comes at a computational cost. When n-grams are added to the vocabulary, the vocabulary and vocabulary vector expands. (Le & Mikolov, 2014).

2.2.2 Latent Dirichlet allocation

Whereas the bag-of techniques treat documents as a collection of words, Latent Dirichlet Allocation ("LDA") views documents as a collection of topics, which are in turn composed of a collection of words. (Blei, et al, 2003). Given a collection of documents and specified number of topics, LDA determines which words frequently appear together. Based on that determination, LDA establishes which words fall within the same topic. When a new document is analyzed, the topics within that document can be inferred. (Blei, et al, 2003).

2.2.3 Word embeddings

In 2013, (Mikolov, et al, 2013a) introduced word2vec, an unsupervised machine learning model that transforms words into vectors. Each word is represented by a vector, wherein each point captures part of that word's meaning. To populate each point, word2vec marshals the distributional hypothesis. Namely, word2vec leverages the observation that words which appear in similar locations are more likely to share the

same meaning. Unlike bag-of-words and bag-of-n-grams, it follows, word2vec preserves differences in syntax and semantics. (Le, et al, 2014).

(Mikolov, et al, 2013a) established two log-linear models to generate word vectors: continuous bag-of-words (WV-CBOW) and skip-gram. In WV-CBOW, vectors are generated by predicting a single word from a window of context words surrounding it. Using eight context words, for example, WV-CBOW selects a target word, the four preceding (history) words, and the four proceeding (future) words. The history and future words are passed as input into a neural network. The model is trained by correctly classifying the middle word.

The skip-gram model works somewhat in reverse. In the skip-gram model, a central word is passed as input and the model is trained by predicting words within a specified range.

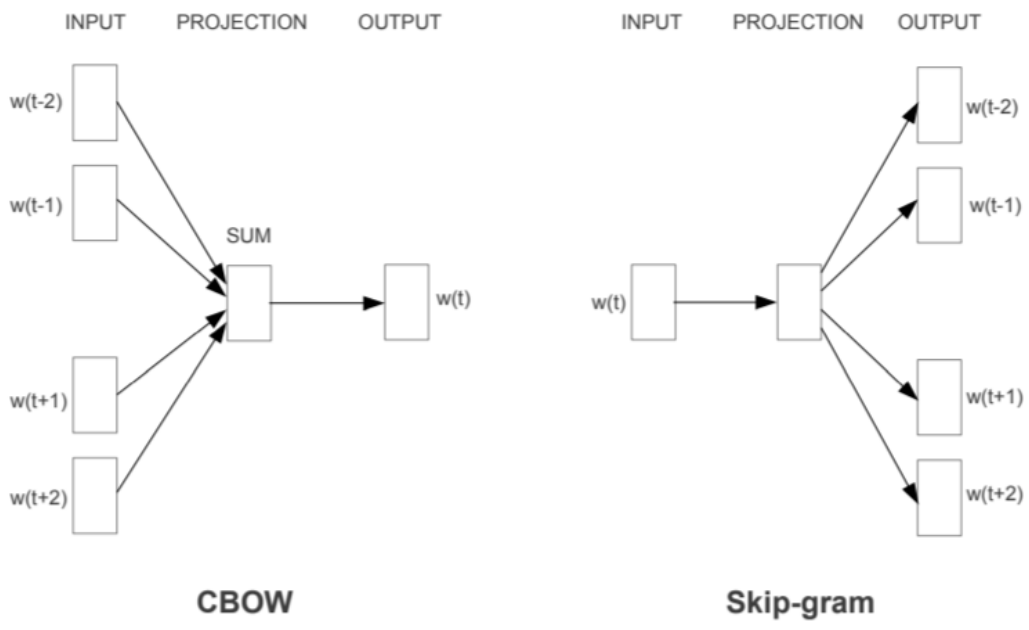


Figure 1: Illustration of word2vec WV-CBOW and Skip-gram methods (Mikolov, et al, 2013a).

Under either approach, the result is a collection of word vectors. Semantically similar words have similar vector representations and therefore appear in similar locations within the vector space. For example, the vector representation for the word "strong" is close to that of the word "powerful." The word vectors can also be subjected to vector operations, to reveal subtle semantic relationships between words. For example, vector operations and reveal a city and the country it belongs to: "France is to Paris as Germany is to Berlin." (Mikolov, et al, 2013a).

2.2.4 Sentence and document embedding

After the success of word embeddings, researchers began developing way to map larger units of text into a vector space. Several document embedding techniques were developed in the following years.

The seminal document embedding research is set forth in (Le & Mikolov, 2014). That work, widely referred to as doc2vec, introduced the Paragraph Vector. The Paragraph Vector is "an unsupervised algorithm that learns fixed-length feature representations from variable-length pieces of text such as sentences, paragraphs, and documents." (Le & Mikolov, 2014). Despite its name then, Paragraph Vector can generate a vector from any piece of text. Indeed, (Le & Mikolov, 2014) chose the name "Paragraph Vector" to emphasize that the method can be applied to variable-length pieces of text.

Marshalling the word2vec approaches, (Le & Mikolov, 2014) developed two Paragraph Vector models: Paragraph Vector Distributed Memory Model (PV-DM) and Paragraph Vector Distributed Bag of Words model (PV-DBOW).

In PV-DM, every document is mapped to a unique vector which is inserted into matrix *D*. For their part, every word is mapped to a unique vector represented by a column in Matrix *W*. With *N* paragraphs each mapped to *p* dimensions, and *M* words each mapped to *q* dimensions, the model has a total of $N * p + M * q$ parameters. As in WV-CBOW, a paragraph vector is produced by training a neural network on the task of predicting a center word within a window of context words. But in PV-DM, the document's id is also used in the prediction task. The paragraph vector is shared across all the context windows generated from that same document, but not across all the documents. The word vector matrix, in contrast, is shared among all the documents.

As illustrated below, the paragraph vector and word vectors are averaged or concatenated to predict the missing word.

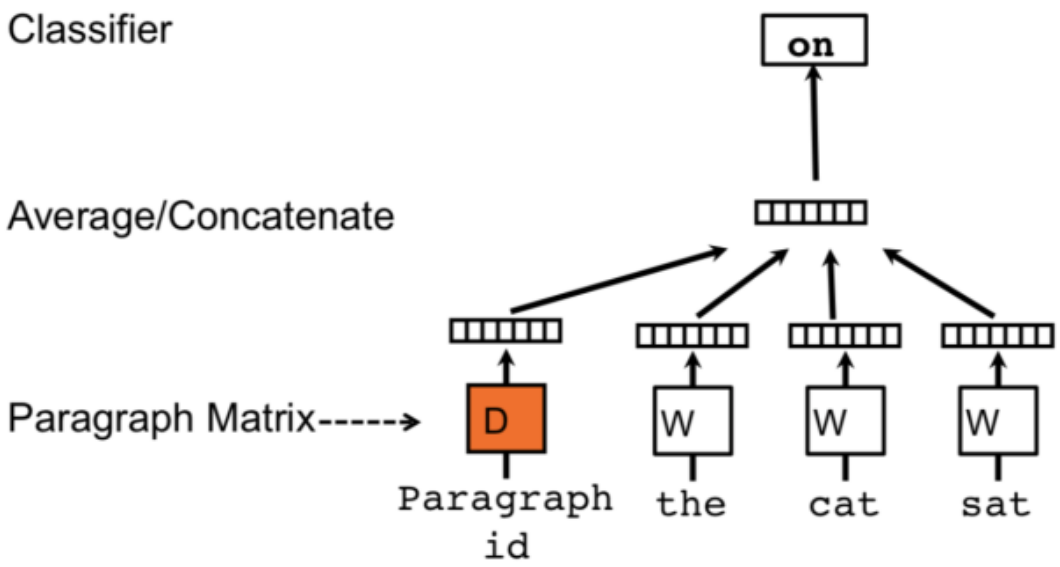


Figure 2: PV-DM model (Le & Mikolov, 2014).

Both paragraph vectors and word vectors are trained using stochastic gradient descent and backpropagation. At every step of the gradient descent, a context is sampled from a random document and used to compute an error gradient and update the model

parameters. At prediction time, a paragraph vector is inferred for a new document through gradient descent. In that procedure, more columns are added to *D*, while *W*, *U*, and *b* are held fixed.

In the PV-DBOW approach, paragraph vectors are trained by predicting a target word from a randomly sampled text window. At each iteration of stochastic gradient descent, a text window is sampled and a random word is taken from that window. A classification task is then formed, using the specific document vector. PV-DBOW is thus similar to the word2vec skip-gram approach.

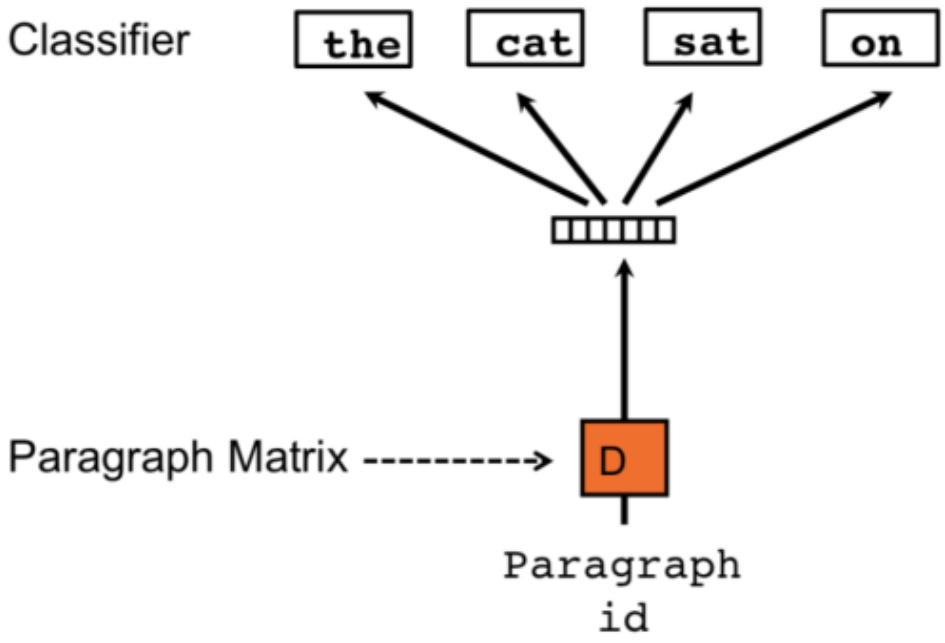


Figure 3: PV-DBOW (Le & Mikolov, 2014).

(Le & Mikolov, 2014) tested the Paragraph Vector approaches by conducting classification tasks. They took the top ten snippets that resulted from 1 million popular search queries. For each query, they created a set of three documents: two paragraphs extracted from results produced by the same query, and one paragraph that resulted from

a different query. Upon that triplet fixation, the researchers sought to identify which paragraphs were produced from the same query. They concluded that Paragraph Vector was proficient in understanding the meaning of documents and that it demonstrated a 32% relative improvement in error rate compared to traditional approaches such as bag-of-words, bag-of-n-grams, and weighted bag-of-bigrams.

One year later, (Dai et al., 2015) applied PV-DBOW to document similarity tasks. In their first set of experiments, (Dai et al, 2015) examined 4,490,000 Wikipedia articles, processed them, and ultimately obtained a vocabulary of 915,715 words. They trained Paragraph Vectors on the articles and, as demonstrated in Figure 4, confirmed that similar categories were grouped together.

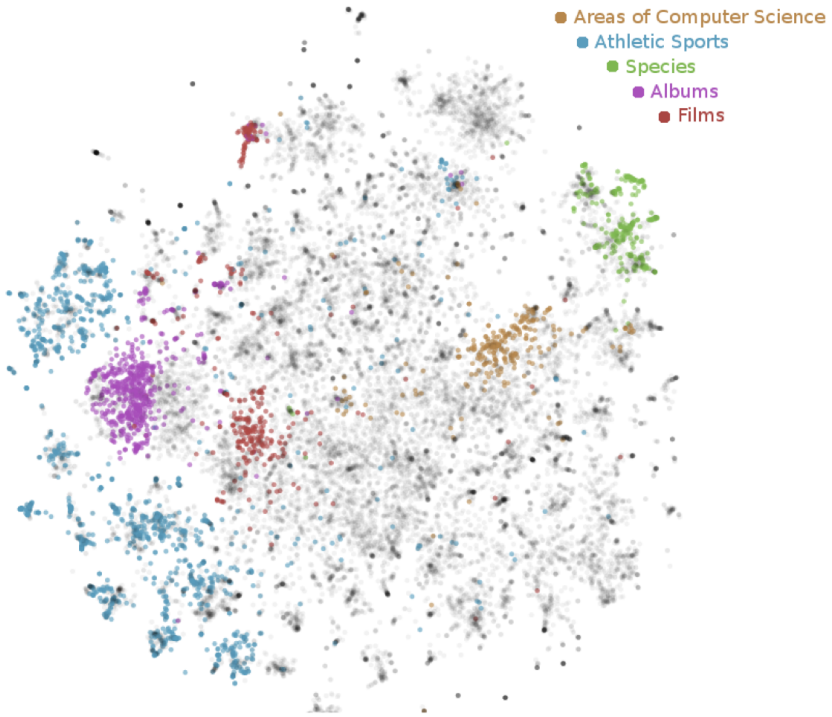


Figure 4: Visualization of Wikipedia paragraph vectors

(Dai et al, 2015) then examined nearest neighbors of the Wikipedia articles and compared them to the nearest neighbors that were produced by LDA. The analysis

demonstrated that the Paragraph Vector produced more accurate nearest neighbors. For example, when comparing the neighbors nearest to the Wikipedia "Machine learning" article, Paragraph Vector did not yield any false positives, whereas LDA produced four.

LDA	Paragraph Vectors
Artificial neural network	Artificial neural network
Predictive analytics	Types of artificial neural networks
Structured prediction	Unsupervised learning
Mathematical geophysics	Feature learning
Supervised learning	Predictive analytics
Constrained conditional model	Pattern recognition
Sensitivity analysis	Statistical classification
SXML	Structured prediction
Feature scaling	Training set
Boosting (machine learning)	Meta learning (computer science)
Prior probability	Kernel method
Curse of dimensionality	Supervised learning
Scientific evidence	Generalization error
Online machine learning	Overfitting
N-gram	Multi-task learning
Cluster analysis	Generative model
Dimensionality reduction	Computational learning theory
Functional decomposition	Inductive bias
Bayesian network	Semi-supervised learning

Figure 5: The nearest neighbor results to the Wikipedia article on "Machine Learning." Bold face text indicates articles that (Dai et al, 2015) found to be unrelated.

(Dai et al, 2015) next demonstrated that, like word2vec, vector operations can be applied to Paragraph Vectors to detect and manipulate semantic meaning. First, (Dai et al, 2015) found articles relating to "Lady Gaga" by measuring the cosine similarity between the documents. They then demonstrated that vector operations can be used to find equivalent documents. More specifically, they found the Japanese equivalent of "Lady Gaga" by taking the "Lady Gaga" paragraph vector, subtracting the "American"

word vector, and adding the "Japanese" word vector:, i.e, *pv*("Lady Gaga") - *wv*("American") + *wv*("Japanese"). Such operations could be useful, the researchers suggested, to applications relating to corpus navigation, dataset exploration, or book recommendations.

In their next experiment, (Dai et al, 2015) constructed two datasets composed of Wikipedia article triplets. Using those datasets, they compared Paragraph Vector to benchmark document embedding methods including LDA and bag-of-words. Where appropriate, the researchers varied the number of embedding dimensions. They found that Paragraph Vector was more accurate than the benchmark methods and that Paragraph Vector was most accurate when using 10,000 dimensions/topics. They further concluded that Paragraph Vector's accuracy improved where model training included the joint training of word vectors.

The first dataset was handpicked. It contained 172 triplets of articles that the researchers knew were related, based on their domain knowledge. For example, "'Deep learning' is closer to 'Machine learning' than 'Computer network.'" (Dai et al, 2015). In analyzing that dataset, Paragraph Vector achieved 93% accuracy with a dimensionality of 10,000 embeddings. The second dataset included 19,786 triplets. Two articles within each set were taken from the same Wikipedia category and were thus close in meaning, whereas the third article was randomly selected. Against that dataset, Paragraph Vector achieved 78.8% accuracy on a dimensionality of 10,000.

In addition to Wikipedia articles, (Dai et al, 2015) performed experiments wherein they used Paragraph Vector to find related scholarly articles. In those experiments, the researchers extracted text from 886,000 scholarly articles and applied a

minimum frequency cutoff to obtain a vocabulary of 969,894 words. They then obtained the nearest neighbors to both (Le et al, 2013) and (Dai et al, 2015). In accordance with their projections, the nearest neighbor of (Dai et al, 2015) was in fact (Le et al, 2013).

To gauge the performance of the different models in comparing the academic papers, (Dai et al, 2015) divided the papers into 20,000 triplets, where each triplet contained two papers sharing at least one subject, and the third paper was chosen at random from papers that contained no shared subjects. They determined that Paragraph Vector performed on par with LDA's best performing number of topics, and that Paragraph Vector was less sensitive to differences in embedding size. Using the research papers as data, both LDA and Paragraph Vector performed best with a parameter of 100 dimensions.

2.2.5 Unsupervised models that leverage sentence ordering

(Kiros et al, 2015) introduced the Skip-Thought model which, unlike the Paragraph Vector, is trained on the *order* of sentences within a corpus. The Skip-Thought model is like the previously discussed skip-gram models. But instead of using a word to predict the surrounding context, a sentence is used to predict the sentences around it. The Skip-Thought model has three components: The Encoder Network; Previous Decoder Network; and the Next Decoder Network. The Encoder takes a given sentence, *x(i)*, and generates the fixed-length vector representation, *z(i)*. The Previous Decoder Network takes *z(i)* and attempts to generate the sentence preceding *x(i)*; whereas the Next Decoder Network attempts to generate the sentence following *x(i)*. When training is completed, the Encoder can be used to generate fixed length sentence representations, which can be used for various tasks including classification. In the

reported results, (Kiros et al, 2015) demonstrated that skip-thought vectors ably captured both semantics and syntax of encoded sentences.

FastSent (Hill et al, 2016) is similar to Skip-Thought, in that it uses adjacent sentences as a prediction target and word sequences to help capture semantics. Similarly, (Pagliardini et al, 2018) introduced sent2vec, which extended the (Mikolov et al, 2013a, 2013b) C-BOW method to the sentence level. Like doc2vec, sent2vec produces embeddings of collections of words by averaging the word embeddings for unigrams and n-grams present within a sentence. But unlike doc2vec, sent2vec accounts for the sequence of the groups of words.

2.3 Motivation and Goals

It comes as no surprise that embedding approaches have been applied within the legal domain. But while significant attention has been directed to machine learning within the legal industry, the available literature suggests that some opportunities have been overlooked.

From all that appears, no research project has been constructed using the same document types from the same legal jurisdiction. For example, (Chalkidis, 2019) used various types of legal documents (a mixture of statutes, caselaw decisions, and executive orders) from various international jurisdictions. While (Nay, 2016) strictly used United States federal government documents, the approach comingled documents from the three government branches. At least one project appears to have focused exclusively on United States caselaw, the document type that this study focuses on. (Sugathadasa et al, 2018). But it is unclear if that caselaw came from a single legal jurisdiction.

This is a notable oversight given the conventions and idiosyncrasies of specific courts. In Rhode Island, for example, the Supreme Court systematically uses specific terms such as "trial justice," and abbreviations such as "PCR" (for postconviction-relief application). In addition, Rhode Island cases contain citations to specific Rhode Island statutes, which are unlikely to arise with any frequency in other jurisdictions. By using a mixture of caselaw from multiple jurisdictions then, a model is likely to be trained on words that have no meaning across a great number of documents within the corpus.

Beyond that, these approaches yield vectors that are overly inclusive in terms of topics and/or legal issues. As discussed in section 2.1.1, a given caselaw decision is likely to address various issues, which should each be viewed as a discrete document. But in the aforementioned projects, document vectors were created using the entire collection of words over each document, with no regard to the structure of caselaw. By categorizing multiple unrelated issues within the same document, the prior approaches mash together various legal topics which have no meaningful correlation to each other.

The prior approaches also fail to account for citations. Because specific legal issues rely on the same authoritative caselaw decisions, a collection of similar issues is likely to contain at least some of the same citations. That said, the raw format of citations isn't helpful to the machine learning process. A citation is composed of individual tokens. In and of themselves, those tokens do not provide context to the machine learning process. Take for example the citation *State v. Rivera*, 221 A.3d 359 (R.I. 2019). During the training process, each word in the citation is tokenized into a list of words:

["State", "v.", "Rivera," "221", "A.3d", "359", "(R.I.", "2019)"]

Thus tokenized, the citation becomes a list of words that are both ubiquitous and infrequent. The word "State" will appear in almost every citation to a Rhode Island criminal case; the word "v." will appear in virtually every citation; the word "A.3d" will appear in every case that cites to the third version of the Atlantic Reporter; and the words "(R.I." and "2019)" will appear in every citation to a Rhode Island case in the year 2019. The remaining words, "221" and "359" are non-descriptive and are unlikely to appear unless another case cites to *State v. Rivera*.

To be sure, (Sugathadasa et al, 2018) observed that citations reference other legal decisions which may be relevant to the given case. (Sugathadasa et al, 2018) also accounted for citations by recording them within an index object. But unlike this project, (Sugathadasa et al, 2018) did not leverage citations to inject additional semantic meaning within the text of the documents. Nor did (Sugathadasa et al, 2018) attempt to process away the tokens composing the actual citation. And because (Sugathadasa et al, 2018) did not treat cases as individual sections, the citations that it associated with each case likely contained citations to different legal issues that were unrelated to each other.

The goal of this project is to explore those oversights and train a model that can more accurately infer related legal issues.

Chapter III

Design

In this section, I detail the requirements necessary to achieve my stated goals. I then provide an overview of a system that addresses the requirements, with discussion of the three major components. In the final section, I provide descriptions of the packages that my components depended on.

3.1 Requirements

Certain requirements were necessary to achieve my stated goals:

1. Collect caselaw documents from one specific legal jurisdiction. Due to personal familiarity, I used caselaw decisions from the Rhode Island Supreme Court.
2. Create a service that parses individual sections from the caselaw documents.
3. Create a service to locate citations within caselaw documents and replace them with citegrams that are unique to the specific case.
4. Create three corpora to train three specific models. Namely, create a corpus of caselaw documents; a corpus of individual sections from the caselaw documents; and a corpus of individual sections from the caselaw documents with the citations replaced by citegrams.
5. Train three models on each respective corpus and evaluate the results.

3.2 Systems overview

The following component diagram shows the overall design of the system:

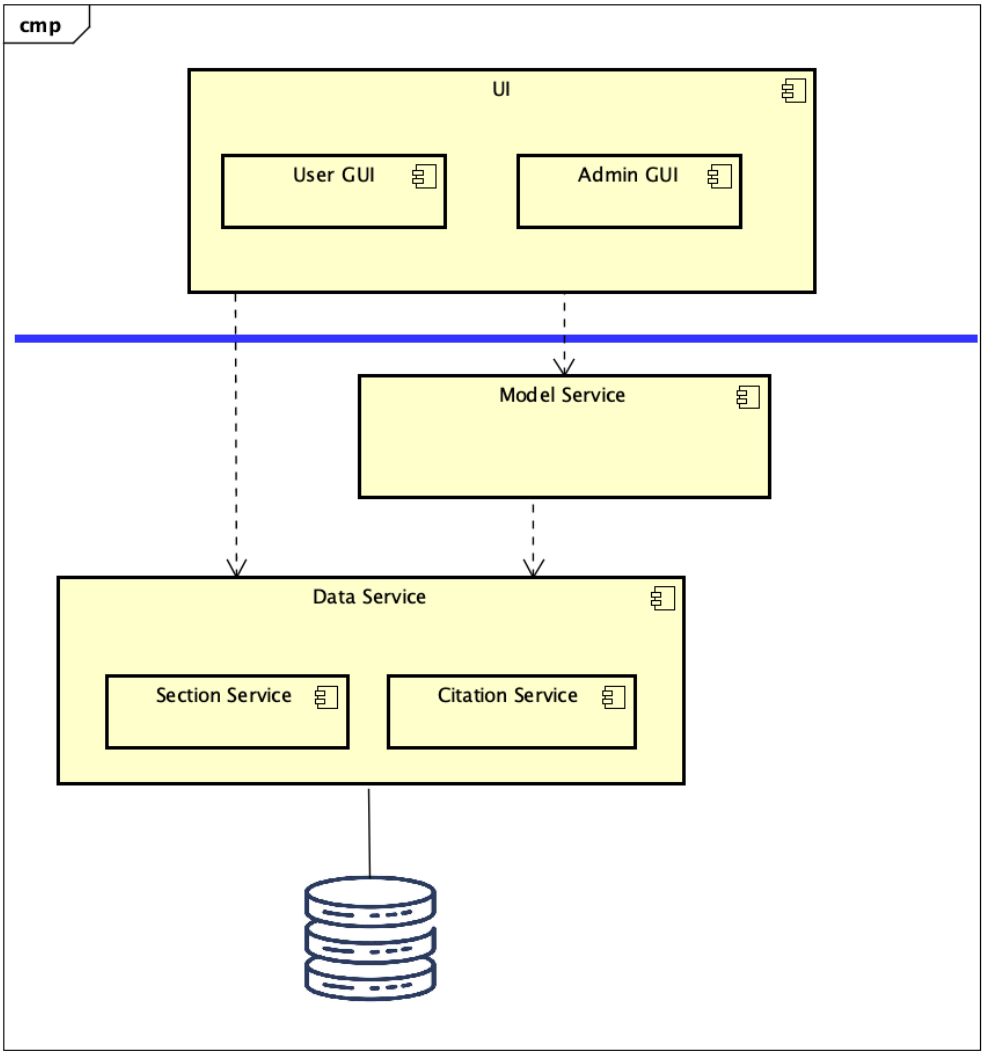


Figure 6: Component diagram of issue2vec system.

There are two principal components, the Data Service and Model Service. The Data Service is responsible for transposing text documents into Law objects, which are subsequently used to train the models. The Data Service itself contains two components. The Section Service is used to locate sections within a case and populate those sections with the underlying text. It is complemented by the Citation Service, which is used to locate citations and create corresponding Cite objects. The Citation Service can also be

called to replace citations with citegrams. To ensure that the citegrams for a given citation are universally consistent, the Cite Service has access to a database of metadata.

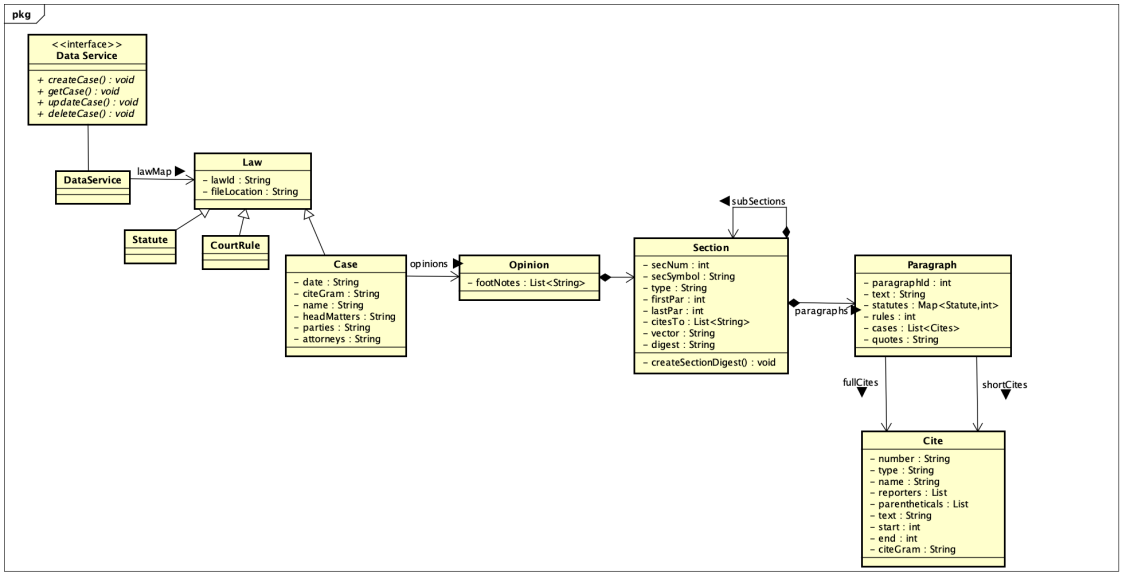


Figure 7: Data Service class diagram.

As depicted in Figure 7, the Data Service is used to create and access Law objects. The Case class, which inherits from the Law class, is the focus of this project. A Case object is composed of Opinions, which are in turn composed of Sections. Most caselaw documents have a single opinion. However, when a justice or justices publish a concurrence or dissent, the caselaw document has multiple opinions. Sections are composed of Paragraph objects, which represent a textual paragraph. In turn, Paragraph objects contain lists of Cite objects. Those Cite objects contain data about a particular citation, including the citegram of that citation.

The Model Service is responsible for training the models that form the basis of evaluation for this project. By calling the Data Service, the Model Service can compose the corpora necessary to train each respective model. For example, the Model Service can call the Data Service to return the entire text of a Case object. It can alternatively call the

Data Service to return the specific sections of that Case object. It can also call the Data Service to return the text with citegrams appearing in place of the citations within the text.

3.3 Dependencies

The program was written in python. It relied on several dependency packages, most notably including:

- **Gensim** (v. 3.8.3): Gensim is an open-source library used for unsupervised topic modeling and natural language processing. It is particularly suited for this project. It includes implementations of word2vec, doc2vec, latent semantic analysis, non-negative matrix factorization, LDA, and td-idf. According to Wikipedia, Gensim, as of 2018, has been used and cited in over 1400 commercial and academic applications. *See* https://pypi.org/project/gensim.
- **Natural Language Toolkit** (v. 3.5)**:** NLTK is a suite of libraries used for natural language processing. NLTK supports integral NLP tasks, such as classification, tokenization, stemming, tagging, parsing, and semantic reasoning. *See* https://www.nltk.org.

Chapter IV

Implementation

In this section, I describe how I implemented the project. The discussion includes details about the main algorithm of each service, including challenges and design choices that arose during implementation.

4.1 Collecting the caselaw documents

I created an account at Harvard Law School's Case Access Project (CAP), www.case.law, a service which publicizes United States caselaw decisions in JSON format. I created a script to download CAP cases. Due to personal familiarity, I used Rhode Island Supreme Court decisions as the data. In total, I downloaded 18,691 Rhode Island caselaw decisions. To create the corpora to train the respective models, I implemented the Section Service and Citation Service, as described below.

4.2 Algorithm descriptions and challenges

This project involves three major modules, each of which was integral to the project. Developing these modules was a non-trivial task, with considerable challenges and design choices.

4.2.1 Section Service

The Section Service is responsible for dividing a caselaw decision into specific sections. As discussed in chapter 2.1.1, individual sections within cases are frequently signified by symbols, keywords, or a combination of thereof. Leveraging those features,

I created an algorithm to identify and extract individual sections from caselaw decisions. In pseudocode, my approach was to:

```
Create sections:

      Create sections using headings within the case:

            Create sections by identifying symbols and key
            phrases

            If there are no symbols or key phrases, save entire
            case opinion as a single section

      Optionally organize sections into subsections
```

Developing a functional Section Service was complicated by the fact that the Rhode Island Supreme Court is inconsistent with its use of symbols. Some caselaw documents are not explicitly divided into sections. Even when the caselaw documents are divided into sections, the hierarchical use of symbols and keywords is inconsistent. For example, one case might organize symbols in a hierarchy of: Roman Numeral, Lower-Case Letter, Number. But in another caselaw document, the symbol hierarchy might appear as Upper-Case letter, Number, Lower-Case letter. This inconsistent formatting complicated the organization of subsections. To arrange sections into nested subsections, it was necessary to determine the hierarchy of the symbols in each caselaw document.

In yet other instances, sections are only identified by keywords and headings; they are not signified by symbols at all. In those instances, there was no reliable programmatic way to determine if the sections were organized into a hierarchy of subsections. Note that the Rhode Island Supreme Court often denotes sections by centering the text and using bold font. Unfortunately, that metadata was is not available in the CAP documents.

Another challenge was that individual sections may or may not pertain to the same legal issue. The case of *Tempest v. State*, 141 A.3d 677, 691 (R.I. 2016), is a helpful example. In that case, the State raised the defense of laches. To establish the defense, the State is required to prove that (1) the opposing party unreasonably delayed seeking relief and (2) the State was prejudiced by the delay. In its opinion, the Rhode Island Supreme Court set forth the issue in one section, and then addressed each requirement in independent subsections. As a result, the *Tempest* case has three different sections addressing the laches issue. But in other instances where the Court addressed the issue of laches, such as *Raso v. Wall*, 884 A.2d 391, 395 (R.I. 2005), all the legal discussion was set forth in a single section.

The inconsistent formatting necessitated a design choice: Whether to train the model by passing an entire section and its nested subsections or to pass only individual subsections. I implemented a parameter that allows the Section Service to be run with or without the use of subsections. In my initial tests, each training document was composed of a section and its nested subsections. That approach resulted in a corpus of 27,131 documents. After evaluating models trained using that corpus, I decided to treat each individual section and subsection as its own document. That approach was more in-line with my motivation and goal; namely, parsing cases into discrete sections. Using that approach, I created a corpus of 37,954 documents.

4.2.2 Citation Service

Using my knowledge of legal citation format, I developed a dictionary of regular expressions to locate citations within a text. The dictionary contains 103 patterns, capturing almost all relevant reporter publications. Composing it was a non-trivial task.

I also developed a suite of functions to pinpoint the beginning and end of the citations; that is, the names and parentheticals. My ultimate approach, in pseudocode, was to:

```
Get the full cites:

      Locate the full citations in each paragraph by using
      regular expressions to search for reporter patterns

      Eliminate duplicate full cites

      Instantiate a new citation object for each non-duplicate
      match:

            Find the end of the citation

            Extract the reporters (including parallel citations,
            if present)

            Extract the case name/locate start of citation:

                  Create SQL queries using reporter cite and
                  select case name from database

                  Using the case name, find the beginning of the
                  citation

            Locate and record explanatory parentheticals

            Get the citegram:

                  Create SQL query using reporters and select
                  citegram from database


Get short cites:

      For each full cite, extract volume and source from each
      reporter and create regex search patterns using the volume
      and source information

            For each regex match, extract the text of the short
            cite:

                  In case of short cite with parallel citations,
                  find the beginning of the first reporter within
                  the text

                  Using name of the full cite, find the name of
                  the short cite within the text

                  Find end of the short cite within the text

                  Instantiate the short cite:
```

```
                        Copy relevant information from the full
                        cite, including the reporters and
                        citegram

                        Record the indexes of where the text of
                        the short cite falls within the text

Get abbreviations:

      Use regular expressions to match "Id." within the text

      For each match, instantiate a Cite object:

            Record the indices of where the text of the
            abbreviated cite falls within the document

      Determine the citation to which the abbreviation refers:

            Combine all the citations into an ordered list

            Update each abbreviation with the appropriate
            citegram
```

Producing the algorithm was a non-trivial task that required significant development and debugging. The different types of citations – full citations, short citations, and abbreviations – each presented unique challenges that required finetuning of the search and parsing techniques. The algorithm also had to be flexible enough to handle both citation sentences and citation clauses. The task was further complicated by the fact that legal citation styles have changed over the years. Now, for the most part, the Rhode Island Supreme Court's caselaw decisions include citations that adhere to the format delineated by (The Blue Book, 2020). Locating and parsing those citations was relatively straightforward. However, citations from prior generations were more difficult to handle. Locating those citations using regular expressions was rather straightforward. But parsing the citations to extract the captions and parentheticals required significant testing and reworking.

The presence of duplicate full citations and distinct citations to the same reporter and volume posed further difficulties. In a small number of cases, documents included

multiple full citations to the same case. It was necessary to record the presence of duplicate full cites to avoid misidentifying them as short cites. Similarly, a small number of cases included citations to cases that were published in the same volume of the same reporter. It was necessary to note those occurrences, to ensure that the appropriate short citations were located.

In developing the Citation Service, I realized that the citegram of the case being processed would never appear in the actual text of that case. This is because a caselaw decision would never cite to itself. However, for the purposes of creating citegrams that provide additional semantic meaning to the text of a caselaw decision, it was desirable to have the citegram of the given case within the text of itself. For that reason, I appended the citegram of the case being processed to any paragraph that contained other citegrams.

The system that I ultimately implemented was a considerable improvement upon my initial version. In the initial version, I created catch-all regex patterns for full cites and short cites. The patterns included capture groups for all the portions of a full cite, including the name, reporters, and parenthetical information. That approach proved overly rigid, and it also allowed a significant number of citations to slip through. Upon further reflection, I realized that this process could be improved. It was overkill to capture all the components of a cite in a single regular expression, as locating a full reporter citation is sufficient to locate full cites. I therefore abandoned my dictionary of regex expressions in favor of a new one that only included reporter patterns. Using (The Bluebook, 2020), I created dictionary entries for nearly all regional and state reporters. That said, there were benefits to my initial approach. Namely, matching a whole citation alleviated the need for further parsing of the text. When I abandoned that approach in

favor of locating the reporters, I had to develop a suite of functions to parse the text and retrieve the name, end of the cite, and parentheticals.

My initial version involved many calls to a metadata database, which proved costly in terms of time. In my initial approach, I made calls to the database for every full cite and short cite. I then iterated through the resulting match objects, passing them to a series of functions designed to extract the citations and ensure that I found correct results. To streamline this process, I created a slimmer database table, using only the required columns. I also restricted database calls to full cites. I then stored the reporter information as a property within the full cite objects and used that information to locate short cites. More specifically, I used the reporter information within each full cite to populate regex patterns that I used to locate short cite patterns within the text.

Using my initial version, it took about five days to execute the program over the copra, resulting with 72,875 citations being located and replace them with citegrams. Using the modified approach, it took about two days to insert 180,930 citegrams.

4.2.3 Model Service

After the documents were transformed into objects, I used those objects and the Model Service to create three different models:

- **Base Model**: In this model no sections were used. Instead, the training documents contained the full text of each case. Similarly, no citegrams were inserted, so the citations within the documents appear as they normally would in a caselaw decision.

- **Section Model**: This model was trained using the same collection of cases. However, each section of each case was passed to the model as its own document. No citegrams were inserted into the text, so the citations within the documents appear as they normally would.

- **Citegram Model**: This model was trained using the same section documents as the Section Model. In this model however, citegrams were inserted in place of the citations.

Trialing revealed that the best results were produced by training the model with a vector size of 100 dimensions, 30 epochs, in DBOW mode.

In each case, all words were lower-cased and stemmed. In addition, I created my own collections of stopwords, "law stopwords." It contains terms that frequently appear in caselaw decisions. Using my legal knowledge and familiarity with Rhode Island Supreme Court decisions, I also created a dictionary object of law-bigrams, which was used to replace frequently occurring bigrams with a single token. I also created a list of Rhode Island Supreme Court Justice's and removed those strings from the training text.

Chapter V

Results and Evaluation

I conducted several evaluations to compare the Base model, Section Model, and Citegram Model. In the first set of evaluations, I selected a case that discussed multiple unrelated legal issues and used it to evaluate each model. I appraised the models by creating visualizations of the vector spaces that each generated. Those visualizations reveal that Section Model and Citegram Model outperform the Base Model in the task of clustering relevant texts. I confirmed the results by retrieving the nearest neighbor documents and manually inspecting them. To further evaluate how each model performed in identifying documents, I hand chose a specific section from a recently published decision and used it as test data. Once again, the Section Model and Citegram Model outperformed the Base Model in the task of identifying similar legal texts. Finally, I evaluated triplets of legal documents. For each of five arbitrarily selected legal issues, I created a triplet of documents. Each triplet contained two documents that discuss the issue, as well as a randomly selected document. The results once again demonstrate that the Citegram and Section Models do a superior job in identifying similar legal text.

In the next set of evaluations, I assessed the semantic information captured by citegrams. I chose citegrams that represented well-known cases and, from the vocabulary of the Citegram Model, obtained the nearest neighbor citegrams. I manually reviewed the results to determine if the nearest neighbor citegrams discussed the same issues that were discussed in the well-known case citegrams. With varying success, the nearest neighbor citegrams were from cases that discussed the same legal issues. In a second

experiment, I created triplets of citegrams where two citegrams represented cases discussing the same legal issue. I measured the distance between those citegrams and a randomly selected citegram. The results demonstrated that citegrams were able to capture the issues of the cases that they discussed.

In my final evaluation, I applied the services to another collection of caselaw decisions, CAP's collection of United States Supreme Court decisions. My evaluations revealed that the Section and Citegram Services were able to handle the United States Supreme Court caselaw decisions. It also reinforced the finding that dividing cases into sections bolsters document similarity operations. Namely, United States Supreme Court decisions tend to address single legal issues. Dividing those cases into sections did not demonstrate the improvements that were observed using the Rhode Island Supreme Court cases. While the Section Service did not bolster document similarity operations, the citegrams once again captured semantic information.

5.1 Evaluating the models using visualizations and nearest neighbors

To determine if dividing cases into sections produced more accurate document groupings, I used the case of *State v. von Bulow*, 475 A.2d 995 (R.I. 1984). The case addresses several distinct legal issues, and the analysis section is composed of four primary sections. The first section is itself divided into two subsections, each of which addresses attorney-client privilege issues. (*Id.* at 1003-12). The second and third sections relate to Fourth Amendment search warrant issues. (*Id.* at 1012-21). The second section is also divided into subsections.

5.1.1 Vector space visualizations

I first created visualizations of the vector space for each model, specifically highlighting cases that contained choice phrases. I also specifically highlighted *von Bulow*.

The visualization of the Citegram Model demonstrates that the specific *von Bulow* sections are appropriately clustered with similar sections. More specifically, the *von Bulow* sections pertaining to attorney-client privilege appear near other cases discussing attorney-client privilege; and the *von Bulow* sections discussing Fourth Amendment issues are proximate to other sections discussing Fourth Amendment issues.

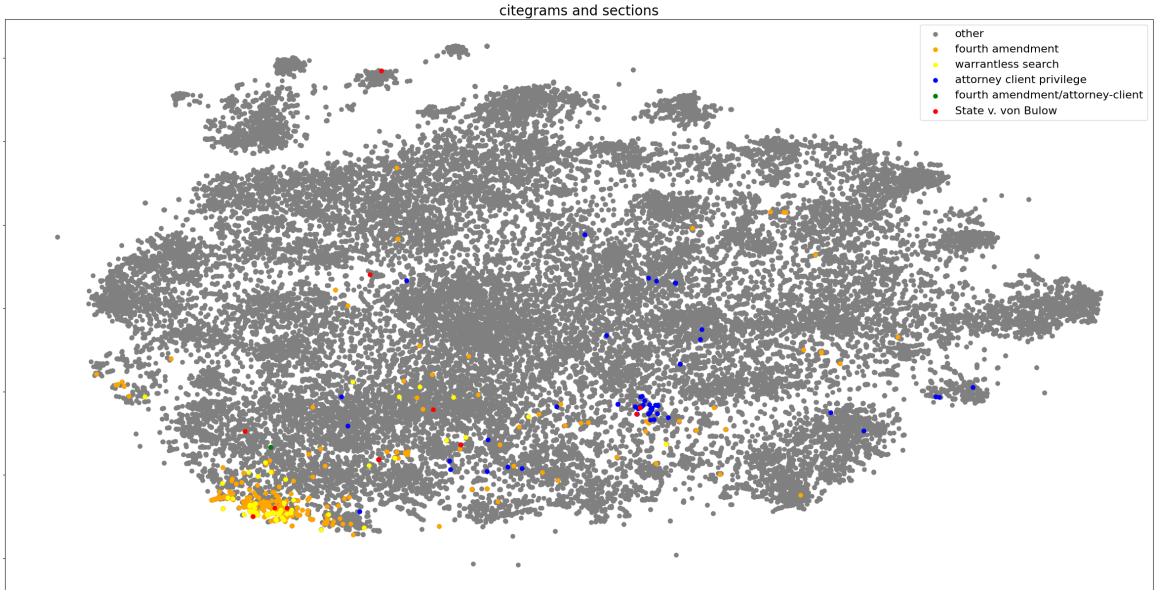


Figure 8: Visualization of the Citegram Model.

The same can be said of the Section Model:

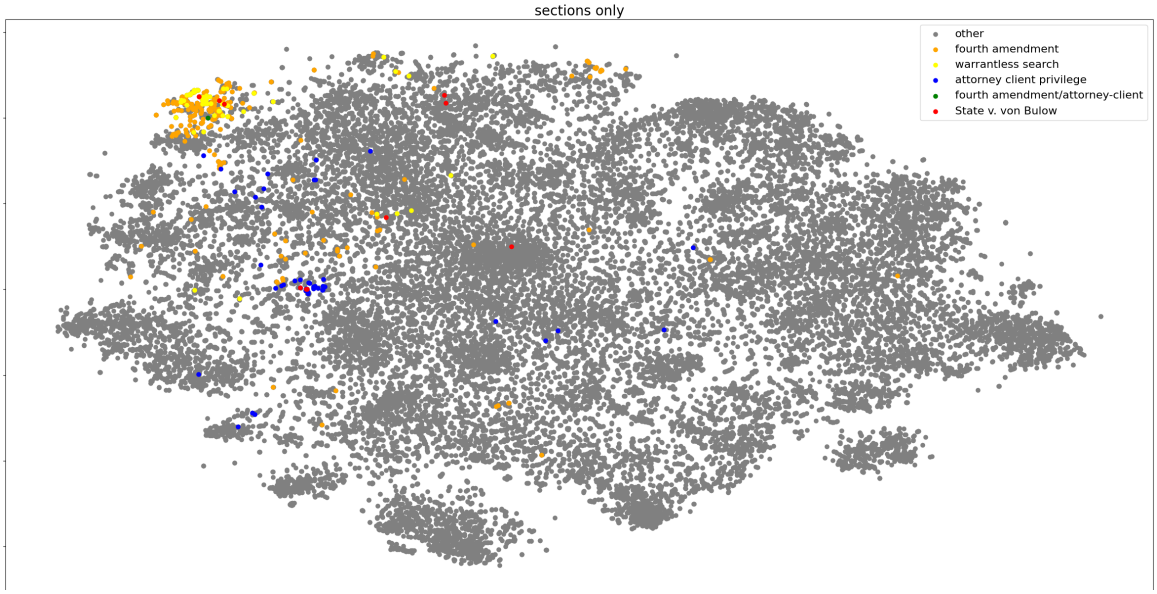


Figure 9: Visualization of the Section Model.

In contrast, when using the Base Model and passing the entire case as a document, *von Bulow* is plotted in an inexact area. It appears on the fringe of Fourth Amendment and warrantless search cases. The *von Bulow* case is not noticeably close to attorney-client privilege cases, even though that issue makes up a significant portion of the case.

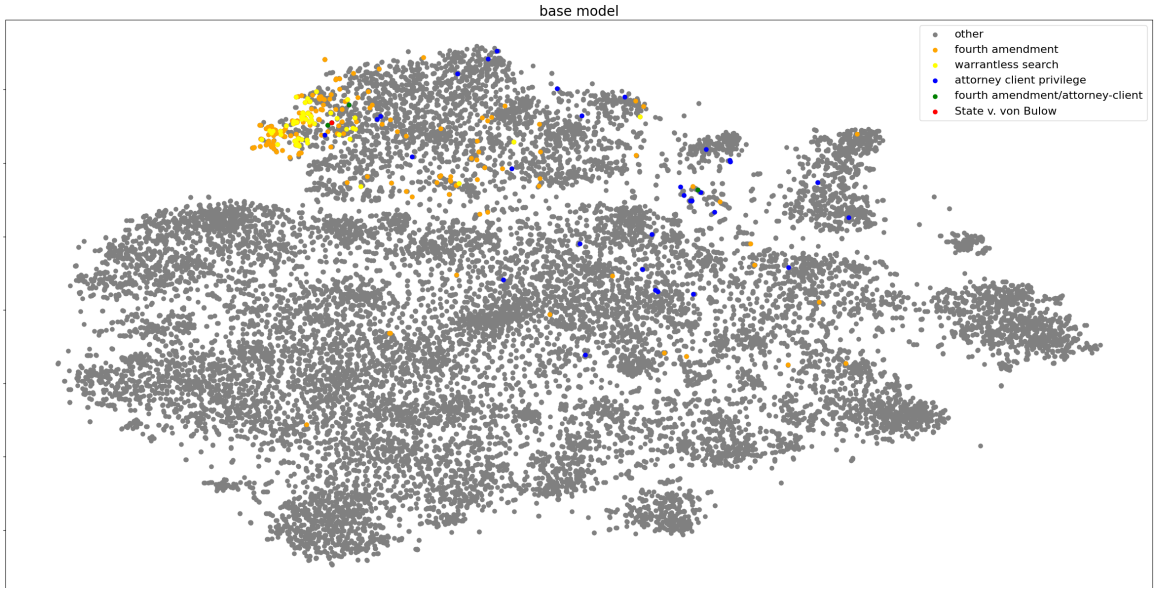


Figure 10: Visualization of the Base Model.

In some sense, Figure 10 reveals that the Base Model does a commendable job in clustering the caselaw documents. But that wouldn't ring true to a researcher attempting to find similar attorney-client privilege cases. In practice, attorneys address issues one-by-one. It is unlikely that an attorney would be interested in finding a collection of cases that discuss two unrelated issues, such as Fourth Amendment searches and attorney-client privilege.

5.1.2 Nearest neighbors

A qualitative inspection of the nearest neighbors reinforces the tale told by the visualizations. Using the *von Bulow* attorney-client privilege section as input, I retrieved the ten nearest neighbors for each of the three models. I manually inspected each resulting case to determine if it contained relevant attorney-client privilege discussion.

Both the Citegram Model and Section Model outperformed the Base model. Of the top ten nearest neighbors that were produced by the Base model, only two contained relevant documents.

Table 1: The Base Model's nearest neighbors to von Bulow attorney-client privilege section. Relevant results appear in bold.

Base Model	
Case Citation	**Cosine Similarity**
Rosati v. Kuzman, 660 A.2d 263 (R.I. 1995)	**0.813**
State v. Juarez, 570 A.2d 1118 (R.I. 1990)	**0.764**
R.I. Grand Jury v. Doe, 641 A.2d 1295 (R.I. 1994)	0.7084
State v. Almonte, 644 A.2d 295 (R.I. 1994)	0.689
In re Doe, 717 A.2d 1129, 1134 (R.I. 1998)	0.667
State v. Brouillard, 745 A.2d 759 (R.I. 2000)	0.657
State v. Yarborough, 636 A.2d 1333, 1334 (R.I. 1994)	0.655
State v. Barkmeyer, 949 A.2d 984, 989 (R.I. 2008)	0.652
In re Grand Jury Subpoena, 748 A.2d 821 (R.I. 2000)	0.646
Tona, Inc. v. Evans, 590 A.2d 873 (R.I. 1991)	0.645

On the other hand, both the Section Model and Citegram Model returned six relevant results. Between those two contestants, the Citegram Model generally outperformed the Section Model, yielding four relevant documents within the top five results.

Table 2: The Section Model's nearest neighbors to von Bulow attorney-client privilege section. Relevant results appear in bold.

Section Model	
Case Citation	**Cosine Similarity**
Rosati v. Kuzman, 660 A.2d 263 (R.I. 1995)	**0.793**
State v. Marrapese, 583 A.2d 537 (R.I. 1990)	**0.768**
State v. Juarez, 570 A.2d 1118 (R.I. 1990)	**0.733**
R.I. Grand Jury v. Doe, 641 A.2d 1295 (R.I. 1994)	0.692
State v. Patino, 93 A.3d 40, 54 (R.I. 2014)	0.687
DeCurtis v. Visconti, et al., 152 A.3d 413 (R.I. 2017)	**0.663**
Capuano v. Outlet Co., 579 A.2d 469 (R.I. 1990)	**0.663**
State v. Fuentes, 433 A.2d 184 (R.I. 1981)	**0.662**
In re Doe, 717 A.2d 1129, 1134 (R.I. 1998)	0.662
State v. Guido, 698 A.2d 729 (R.I. 1997)	0.658

Table 3: The Citegram Model's nearest neighbors to von Bulow attorney-client privilege section. Relevant results appear in bold.

Citegram Model	
Case Citation	**Cosine Similarity**
Rosati v. Kuzman, 660 A.2d 263 (R.I. 1995)	**0.788**
State v. Marrapese, 583 A.2d 537 (R.I. 1990)	**0.78**
State v. Juarez, 570 A.2d 1118 (R.I. 1990)	**0.729**
State v. Patino, 93 A.3d 40, 54 (R.I. 2014)	0.674
DeCurtis v. Visconti, et al., 152 A.3d 413 (R.I. 2017)	**0.669**
State v. Fuentes, 433 A.2d 184 (R.I. 1981)	**0.641**
State v. Leuthavone, 640 A.2d 515 (R.I. 1994)	0.639
State v. Tassone, 749 A.2d 1112 (R.I. 2000)	0.634
Mortg. & Title Co. v. Cunha, 745 A.2d 156 (R.I. 2000)	**0.633**
State v. Guido, 698 A.2d 729, 734 (R.I. 1997)	0.632

In the case of the Base Model, it is important to note that a section – opposed to an entire document – was used to infer the nearest neighbors. As it turns out then, the Base Model's performance was aided by the Section Service. As noted above, the Base Model returned two relevant documents when it was passed the specific *von Bulow* attorney-client privilege section. When it was passed the entire *von Bulow* case, it returned the same documents. However, the documents appeared lower in the results and had lesser cosine similarities.

Table 4: Base Model's nearest neighbors when passed the entire von Bulow document. Relevant results appear in bold.

Base Model	
Case Citation	**Cosine Similarity**
State v. Eiseman, 461 A.2d 369 (R.I. 1983)	0.725
State v. Wyche, 518 A.2d 907 (R.I. 1986)	0.719
State v. Barkmeyer, 949 A.2d 984 (R.I. 2008)	0.717
State v. Juarez, 570 A.2d 1118 (R.I. 1990)	**0.712**
State v. Smith, 512 A.2d 818 (R.I. 1986)	0.709
State v. Brouillard, 745 A.2d 759 (R.I. 2000)	0.702
State v. Almonte, 644 A.2d 295 (R.I. 1994)	0.699
State v. Dufour, 99 R.I. 120, 206 A.2d 82 (1965)	0.697
Rosati v. Kuzman, 660 A.2d 263 (R.I. 1995)	**0.684**
State v. Guido, 698 A.2d 729 (R.I. 1997)	0.681

5.2 Searching using unseen test data

To further assess the models, I selected issues from a recently published case, *State v. Segrain*, 252 A.3d 1255 (R.I. 2021), and used them as test data. The first issue involves a defendant's motion to suppress eyewitness identification. I isolated the issue in its own document, transformed it into a vector, and passed it to each model to obtain the nearest neighbors. For each result, I manually inspected the caselaw document and

determined if there was on-point legal analysis. The Citegram Model and Section Model easily outperformed the Base Model in this task. As shown in Table 5, each of the top ten results in the Citegram model contained on-point legal analysis. The Section Model contained returned eight on-point analysis sections. The Base Model only returned four.

Table 5: Citegram Model Top ten nearest neighbors for State v. Segrain eyewitness suppression issue. Relevant results appear in bold.

Citegram Model	
Case Citation	**Cosine Similarity**
State v. Imbruglia, 913 A.2d 1022 (R.I. 2007)	**0.813**
State v. Lynch, 770 A.2d 840 (R.I. 2001)	**0.776**
State v. Wray, 38 A.3d 1102 (R.I. 2012)	**0.765**
State v. Courteau, 461 A.2d 1358 (R.I. 1983)	**0.758**
State v. Addison, 748 A.2d 814 (R.I. 2000)	**0.755**
State v. Luciano, 739 A.2d 222 (R.I. 1999)	**0.733**
State v. Rodriquez, 478 A.2d 171 (R.I. 1984)	**0.723**
State v. Cline, 122 R.I. 297, 405 A.2d 1192 (1979)	**0.716**
State v. Ivy, 558 A.2d 209 (R.I. 1989)	**0.714**
State v. Texter, 923 A.2d 568 (R.I. 2007)	**0.714**

Table 6: Section Model Top ten nearest neighbors for Segrain eyewitness suppression issue. Relevant results in bold.

Section Model	
Case Citation	**Cosine Similarity**
State v. Wray, 38 A.3d 1102 (R.I. 2012)	**0.805**
State v. Luciano, 739 A.2d 222 (R.I. 1999)	**0.775**
State v. Imbruglia, 913 A.2d 1022, 1028 (R.I. 2007)	**0.77**
State v. Washington, 42 A.3d 1265, 1271 (R.I. 2012)	**0.766**
State v. Addison, 748 A.2d 814 (R.I. 2000)	0.757
State v. Nabe, 92 A.3d 205 (R.I. 2014)	0.756
State v. Lynch, 770 A.2d 840 (R.I. 2001)	**0.752**
State v. Washington, 655 A.2d 701 (R.I. 1995)	**0.746**
State v. Grant, 840 A.2d 541 (R.I. 2004)	0.747
State v. Davis, 131 A.3d 679 (R.I. 2016)	0.741

Table 7: Base Model top ten nearest neighbors for State v. Segrain eyewitness suppression issue. Relevant results appear in bold.

Base Model	
Case Citation	**Cosine Similarity**
State v. Wray, 38 A.3d 1102 (R.I. 2012)	**0.812**
State v. Imbruglia, 913 A.2d 1022 (R.I. 2007)	**0.785**
State v. Roldan, 131 A.3d 711 (R.I. 2016)	0.734
State v. Ivy, 558 A.2d 209 (R.I. 1989)	**0.733**
State v. Rivera, 839 A.2d 497 (R.I. 2003)	0.732
State v. Nabe, 92 A.3d 205 (R.I. 2014)	0.731
State v. Gallop, 89 A.3d 795 (R.I. 2014)	**0.723**
State v. Silva, 84 A.3d 411 (R.I. 2014)	0.723
State v. Pona, 66 A.3d 454 (R.I. 2013)	0.723
State v. Pona, 926 A.2d 592 (R.I. 2007)	0.721

5.3 Triplet evaluation of legal issues

In this experiment, I arbitrarily chose five legal issues. For each legal issue, I created a triplet of caselaw documents. In each triplet, the first two documents were the results of manual legal research. More specifically, I found two cases with representative discussions of the particular issue. The third document was randomly selected from the CAP collection of Rhode Island Supreme Court decisions.

Similar to (Le & Mikolov, 2014) the goal of the experiment is to identify which model best identifies the same legal issue. A better model is one that achieves a small distance for the same-issue cases, and a larger distance between the same-issue cases and random case. To achieve this, I passed each document to each model and inferred a new vector. In the case of the Base Model, the new vector was inferred from the entire caselaw document. In the case of the Section Model, the new vector was inferred from the specifically relevant section. The vector inferred from the Citegram Model was also

based on the relevant section, but citegrams were inserted in place of citations prior to inference.

To score the performances, I computed a ratio of same-issue distance : random-issue distance. The random-issue distance measure is the average distances between each same-issue case and the randomly selected case. A lower ratio reflects that the model performed better at achieving a small distance between the same-issues cases and a larger distance between the randomly selected cases.

As the results in Table 8 reflect, the Citegram and Section Models outperformed the Base Model on average. The Citegram Model demonstrated the best performance in three of the five instances.

Table 8: Triplet evaluations for five legal issues. The best scores appear in bold.

model	average score	custodial interrogation	expert testimony /malpractice	adverse possession	libel	speedy trial
citegram	**0.7232**	**0.5828**	**0.6679**	0.7042	0.8204	**0.8407**
section	0.7352	0.5882	0.6787	0.6850	**0.7947**	0.9296
base	0.7406	0.5832	0.6798	**0.6714**	0.8167	0.9517

5.4 Citegram semantics

To evaluate the use of citegrams, I assessed whether citegrams, as word vectors, were able to capture semantic information of the cases that they represent. I conducted two evaluations to this end. First, I manually selected citegrams that represented well-known and oft cited caselaw decisions. I obtained the nearest neighbor citegrams and read each case to determine if the citegram represented a case with a relevant legal discussion. I then created triplets of citegrams, where two represented cases discussing the same legal issue and the third was randomly selected. I concluded that citegrams captured semantic information and are thereby a useful feature.

5.4.1 Citegram nearest neighbors

To gauge the semantic information captured by citegrams, I selected well-known cases and used the corresponding citegram to find the nearest neighbors of the citegram within the vocabulary of word vectors. I then inspected the nearest neighbor citegrams, to determine if they were relevant to the issue discussed in the well-known case. The results largely demonstrate that the most similar citegrams to a given citegram are other citegrams representing cases that discuss similar issues.

In *Miranda v. Arizona*, 383 U.S. 436 (1966), the United States Supreme Court held that the Fifth Amendment requires police to advise defendants of their constitutional rights when questioning them in police-dominated atmospheres. The citegram representing *Miranda* is cg_miranda_v_arizona_384_us_426. As demonstrated in Table 9, the most similar words to the *Miranda* citegram were mostly other citegrams relating to the custodial interrogation issue.

Table 9: Nearest citegrams to Miranda v. Arizona. Citegrams representing cases with relevant legal discussions appear in bold.

Citegram	Cosine Similarity
cg_edwards_v_arizona_451_us_477	**0.747**
cg_state_v_espinosa_283_a2d_465	**0.728**
cg_state_v_lachapelle_308_a2d_467	**0.714**
cg_johnson_v_zerbst_304_us_458	**0.712**
cg_brewer_v_williams_430_us_387	**0.687**
cg_fare_v_michael_c_442_us_707	**0.669**
cg_massiah_v_united_states_377_us_201	**0.654**
cg_state_v_brown_399_a2d_1222	0.651
cg_state_v_ferola_518_a2d_1339	**0.638**
cg_johnson_v_new_jersey_384_us_719	**0.628**

In *Daubert v. Merrell Dow Pharms., Inc.*, 509 U.S. 579 (1993), the United States Supreme Court held that: (1) general acceptance is not a necessary precondition to

admissibility of scientific evidence under Federal Rules of Evidence, and (2) a trial judge is obligated to ensure that an expert's testimony rests on a reliable foundation and is relevant to task at hand. The citegram representing *Daubert* is cg_daubert_v_merrell _dow_pharmaceuticals_inc_509_us_579. The most similar citegrams to the *Daubert* citegram included a mixed bag. Only five of the citegrams represented cases that dealt with the admissibility of expert testimony. It is notable that the five relevant citegrams were all Rhode Island Supreme Court cases, on which the vocabulary was trained.

Table 10: Nearest citegrams to Daubert. Citegrams representing cases with relevant legal discussions appear in bold.

Citegram	Cosine Similarity
cg_metro_properties_inc_v_national_union_fire_ insurance_co_of_pittsburgh_pa_934_a2d_204	0.687
cg_state_v_correia_600_a2d_279	**0.684**
cg_gallucci_v_humbyrd_709_a2d_1059	**0.673**
cg_barcon_associates_inc_v_tricounty_asphalt_corp_86_nj_179	0.661
cg_state_v_gardner_616_a2d_1124	**0.643**
cg_state_v_traficante_636_a2d_692	0.637
cg_blockburger_v_united_states_284_us_299	0.636
cg_state_v_morales_621_a2d_1247	**0.633**
cg_parcell_v_state_228_kan_794	0.630
cg_aguilar_v_texas_378_us_108	0.626

5.4.2 Citegram triplet evaluation

To further evaluate how well citegrams capture the cases that they represent, I conducted triplet evaluation. I took the same arbitrary legal issues used in section 4.3 and manually selected citegrams of two cases discussing each issue. I then created a triplet of citegrams for each issue. In each triplet, the first two citegrams were the results of manual legal research. The third citegram was randomly selected from the Citegram Model's vocabulary. The distances between the manually selected citegrams was smaller

than the distances between the manually selected citegrams and randomly selected citegram. The results thereby indicate that the semantic information captured by each citegram produces a useful representation of the given case.

Table 11: Results for citegram triplets, where a and b represent the manually selected citegrams and c represents a randomly selected citegram. Smallest distances appear in bold.

distance between citegrams	custodial interrogation	expert testimony /malpractice	adverse possession	libel	speedy trial
a to b	**0.424**	**0.525**	**0.328**	**0.740**	**0.594**
a to c	0.736	0.628	0.825	0.804	0.809
b to c	0.879	0.649	0.854	0.799	0.629

5.5 United States Supreme Court caselaw vectors

As mentioned throughout this paper, this project was tailored to Rhode Island Supreme Court decisions. Hence, the functions that constitute the Section Service and Citation Service were developed and debugged using Rhode Island Supreme Court cases. That said, an exploration into United States Supreme Court cases is a fitting end. It also provides a means to evaluate how the services perform on cases from another jurisdiction.

Using CAP's collection of United States Supreme Court decisions, I created 43,797 caselaw objects, with 87,543 distinct sections. I used those objects to train three additional models that mirror those from the previous evaluations. A visualization of the models demonstrates that the Section Service was flexible enough to treat United States Supreme Court decisions. I also evaluated the models by comparing triplets of cases. The triplet evaluation demonstrates that the unique nature of United States Supreme Court cases diminishes the value of parsing caselaw into sections. Because United States

Supreme Court decisions are generally limited to single legal issues, dividing them into specific sections did not result in the same improvements observed in the Rhode Island Supreme Court cases.

5.5.1 Bill of Rights visualizations

The United States Bill of Rights guarantees citizens certain freedoms by placing limits on the government's power. It is composed by the first ten amendments to the United States Constitution. The Bill of Rights remains a celebrated step forward in the field of human rights. The following figures include visualizations of three models – a base model, section model and citegram model – which were trained using United States Supreme Court decisions. Each case and/or section that discusses one of the amendments within the Bill of Rights is highlighted.

The visualizations demonstrate that the Section Service was flexible enough to handle the collection of United States Supreme Court decisions. Indeed, 87,543 sections were extracted from the 43,797 case objects. And when one compares Figure 10 to Figures 11 and 12, it is apparent that the Section Service was able to parse individual sections that pertained to each amendment. Furthermore, the sections that were parsed into documents appear in close, dense groupings.

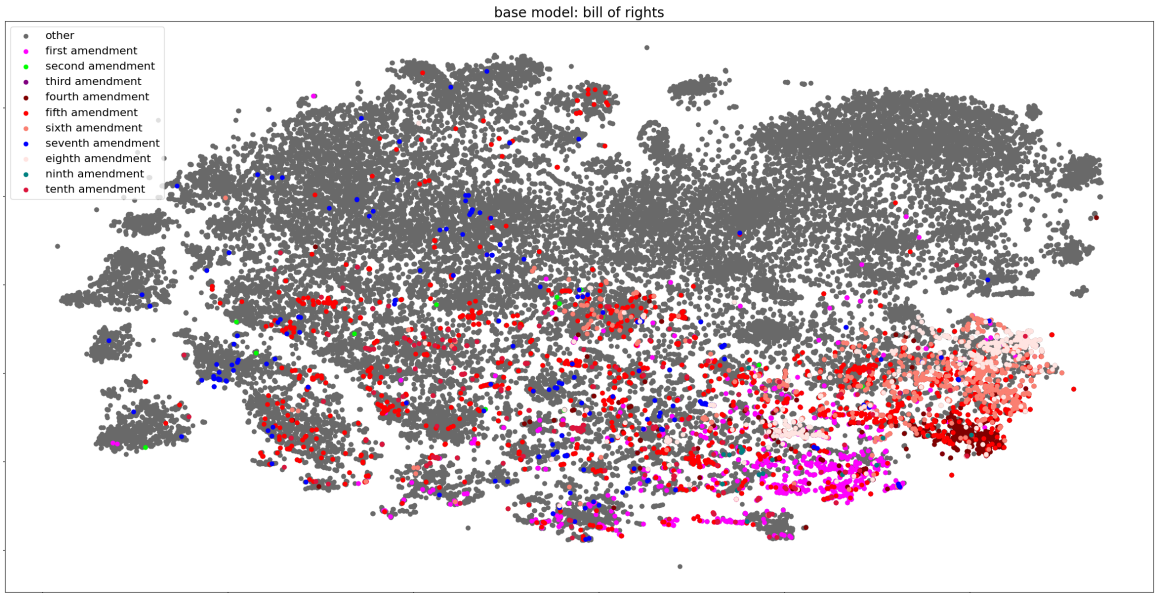


Figure 10: Visualization of Bill of Rights Base Model.

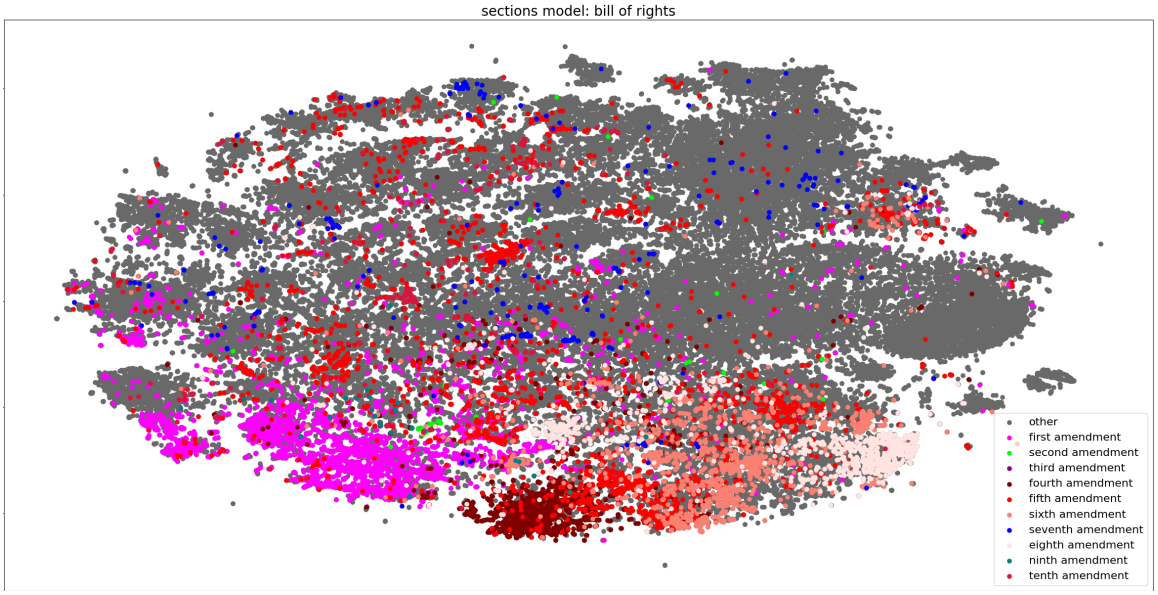


Figure 11: Visualization of Bill of Rights Section Model

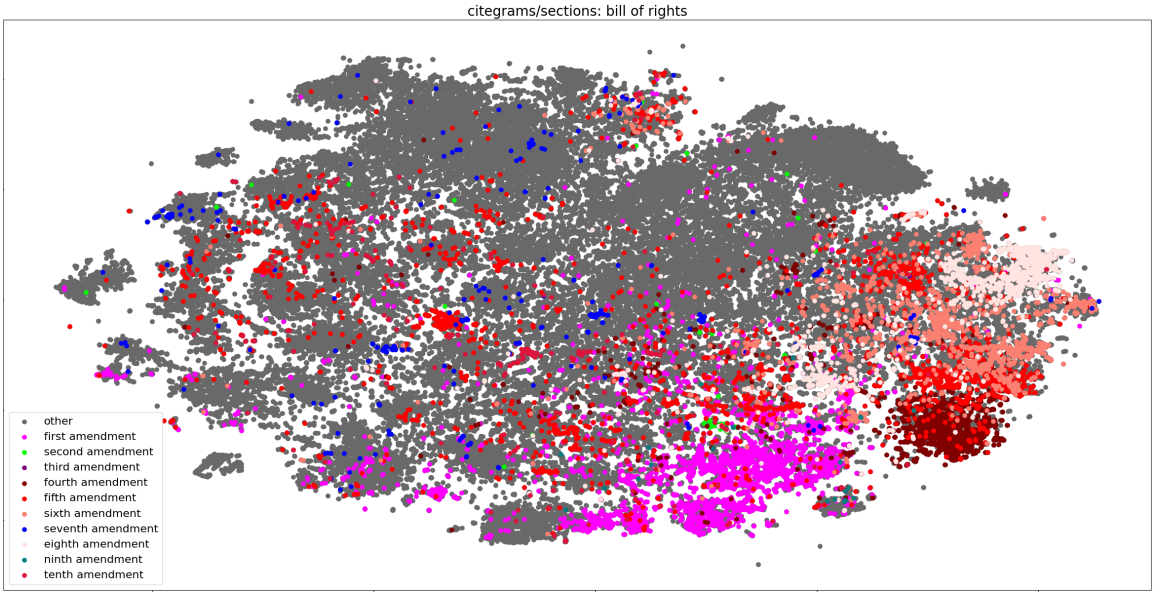


Figure 12: Visualization of Bill of Rights Citegram Model.

5.5.2 Bill of Rights Triplet evaluation

That said, the Section Service and Citegram Service provide minimal value in the task of clustering United States Supreme Court decisions. Given the nature of United States Supreme Court decisions, this outcome is relatively unsurprising. A considerable number of United States Supreme Court decisions are single issue cases. That being the case, the value of distilling the caselaw decisions into discrete sections is minimal.

This observation was produced by triplet evaluation. I selected five landmark decisions from the United States Supreme Court. For each case, I conducted legal research and found another case that addresses the landmark case. I then created five triplets of cases, where each triplet contained the landmark case, the related case, and a randomly selected case. Applying the same scoring process as in section 5.3, I measured the cosine distances between the related cases and random case and assigned a score. In contrast to the results presented in 5.3, the Base Model outperformed both the Section and Citegram Models.

Table 12: Triplet evaluation of United States Supreme Court vectors for landmark cases. The best performance appears in bold.

model	average score	tinker v. des moines (first amendment)	mcdonald v chicago (second amendment)	terry v. ohio (fourth amendment)	gideon v. wainwright (sixth amendment)	ropper v. simmons (eighth amendment)
citegram	0.641	0.635	0.650	0.713	0.671	0.534
section	0.706	0.687	0.724	0.758	0.758	0.601
base	**0.612**	**0.594**	**0.605**	**0.675**	**0.664**	**0.521**

5.5.3 Bill of Rights citegram triplets

The results in Table 12 demonstrate that there was minimal value in parsing United States Supreme Court decisions into sections. However, the results also indicate that citegrams provide value to the semantic calculus. The Citegram Model placed second in each of the five triplet evaluations, outperforming the Section Model by a significant margin.

A triplet evaluation of citegrams used in the preceding case triplet evaluation confirms that citegrams provide semantic value within the realm United States Supreme Court decisions. In *Roper v. Simmons*, 543 U.S. 551 (2005), the United States Supreme Court found that the Eighth Amendment precluded the government from sentencing a juvenile to death. The Court therefore communicated the punishment against the juvenile defendant, Christopher Simmons. As show in Table 13, the top ten citegram neighbors to the *Simmons* citegram, cg_roper_v_simmons_543_us_551, include nine Eight Amendment cases. More pointedly, several of the top results are cases dealing with the issue of cruel and unusual punishment applied to children or mentally incapacitated adults.

Table 13: Top ten closest citegrams to cg_roper_v_simmons_543_us_551. Relevant cases appear in bold. Cases addressing juveniles or mental incapacity appear in bold/italics.

Citation	Cosine Similarity
Graham v. Fla., 560 U.S. 48 (2010)	***0.74394202***
Stanford v. Kentucky, 492 U.S. 361 (1989)	***0.67567396***
Kennedy v. Louisiana, 554 U.S. 407 (2008)	**0.67567396**
Atkins v. Virginia, 536 U.S. 304 (2002)	***0.66732603***
Tison v. Arizona, 481 U.S. 137 (1987)	**0.63798869**
Eutzy v. Fla., 471 U.S. 1045 (1985)	**0.62468052**
Powell v. State of Tex., 392 U.S. 514 (1968)	**0.61934489**
H. L. v. Matheson, 450 U.S. 398 (1981)	0.61463583
Penry v. Lynaugh, 492 U.S. 302 (1990)	***0.61215293***
Roach v. Aiken, 474 U.S. 1039 (1986)	***0.59880459***

Chapter VI

Conclusions and Future Work

We observed that machine learning can be used to detect patterns within in a collection of data and that it can be used in combination with natural language processing. Standing on the shoulders of machine learning and natural language approaches, this project has demonstrated that unique features of caselaw decisions can be leveraged to improve the machine learning process. By parsing caselaw decisions into individual sections, a single caselaw decision can be transformed into several documents, which address discrete legal issues. Working with Rhode Island Supreme Court caselaw decisions, it was observed that the streamlined section documents improve document similarity operations. That observation was reinforced by evaluating United States Supreme Court decisions, which generally address a single legal issue. In those evaluations, dividing caselaw decisions into discrete sections did not improve similarity operations. We also saw that by transforming citations into unique n-grams, the citations within the text of a caselaw decision can be used as additional vocabulary words that provide semantic value.

This project opens the door to several avenues of future work. The methods and results of this work are plainly applicable to legal research and writing and can be further developed towards those ends. The Section Service and Citation Service were developed using Rhode Island Supreme Court cases. While they were versatile enough to treat United States Supreme Court decisions, it stands to reason that the services would need to be tailored to handle cases from other jurisdictions. Similarly, the caselaw documents used in this project were all in plaintext; any bold, italicized, underlined, or centered font

used in the published decisions were not present. Because specialized fonts and spacing are used to identify both sections and citations, those features could be leveraged to improve the Section Service. The Paragraph Vector techniques used to develop the models were based on the research of (Le & Mikolov, 2014). Unlike the Skip-Thought (Kiros et al, 2015) and FastSent (Hill et al, 2016), these models do not account for sentence order. The services developed in this project could be extended to those models. Finally, we observed that the identifying and symbols and descriptions of sections are useful to the process of natural language processing and machine learning. We also observed that, from case to case, courts are inconsistent with their use of those symbols and headings. To assist machine learning, it would be helpful for Courts to consistently employ symbols in the same hierarchy in each one of their caselaw decisions.

References

Blei, D. M., Ng A. Y., & Jordan, M. I., (2003). Latent Dirichlet Allocation, *Journal of Machine Learning Research*, 3, 993-1022. http://www.jmlr.org/papers/volume3/blei03a/blei03a.pdf

Dai, A. M., Olah, C., & Le, Q. V. (2015). Document embedding with paragraph vectors. *NIPS Deep Learning Workshop*. arXiv, Article 1507.07998v1. https://arxiv.org/pdf/1507.07998.pdf

Chalkidis, I. (2019). Deep learning in law: early adaptation and legal word embeddings trained on large corpora, *Artificial Intelligence and Law*, 27, 171-198. https://doi.org/10.1007/s10506-018-9238-9

Harris, Z. S. (1954). Distributional structure. WORD, 10(2-3), 146-162. https://www.tandfonline.com/doi/abs/10.1080/00437956.1954.11659520

Hill, F., Cho K., & Korhonen, A. (2016). Learning Distributed Representations of Sentences from Unlabelled Data, *Proceedings of NAACL-HLT*. arXiv, Article 1602.03483v1. https://arxiv.org/pdf/1602.03483.pdf

IBM Cloud Education (2020, May 1). *Deep Learning.* https://www.ibm.com/cloud/learn/deep-learning

IBM Cloud Education (2020, July 15). *Machine Learning.* https://www.ibm.com/cloud/learn/machine-learning

IBM Cloud Education (2020, August 19). *Supervised Learning,* https://www.ibm.com/cloud/learn/supervised-learning

Kiros, R., Yukun, Z., Salakhutdinov, R.R., Zemel, R., Urtasun, R., Torralba, A., & Fidler, S. (2015). Skip-Thought Vectors, *NIPS 2015 - Advances in Neural Information Processing Systems*, 28, 3294-3302. https://dl.acm.org/doi/10.5555/2969442.2969607

Le, Q.V. & Mikolov, T. (2014). Distributed Representations of Sentences and Documents, *ICML 2014 - Proceedings of the 31st International Conference on Machine Learning*. 14, 1188-1196. https://arxiv.org/pdf/1405.4053.pdf

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013a). Efficient estimation of word representations in vector space. arXiv:, Article 1301.3781. https://arxiv.org/pdf/1301.3781.pdf

Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., and Dean, J. (2013b). Distributed Representations of Words and Phrases and their Compositionality, *NIPS - Advances in Neural Information Processing Systems*. 26, 3111-3119.

Nay. J. (2016) Gov2Vec: Learning distributed representations of institutions and their legal text, *Proceedings of the first workshop on NLP and computational social science*, Association for Computational Linguistics, 49-54. https://ssrn.com/abstract=3087278.

Pagliardini, M., Gupta P., & Jaggi, M. (2018). Unsupervised Learning of Sentence Embeddings Using Compositional n-Gram Features, *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 1, 528-540. https://arxiv.org/pdf/1703.02507.pdf

Ravichandiran, Sudharsan. (2019). *Hands-On Deep Learning Algorithms with Python*. Packt Publishing.

Sugathadasa, K., Ayesha, B., de Silva, N., Perera, A.S., Jayawardana, V., Lakmal, D., & Perera, M. (2019). Legal document retrieval using document vector embeddings and deep learning. arXiv, Article 1805.10685. https://arxiv.org/pdf/1805.10685.pdf

The Blue Book: A Uniform System of Citation (21st ed.). (2020). Cambridge, MA: The Harvard Law Review Association.