Metallic Jurisprudence

Ryan Barron
UMBC Computer Science Department – CMSC 678 Machine Learning & CMSC 673 Natural Language Processing
ryanb4@umbc.edu

Abstract

United States Supreme court transcripts are dissected and analyzed in a neural network to discover if the selected featurization reveals any meaningful patterns to determine case outcomes. The vector embedding representations of the court case transcripts were not distinguishable and resulted in the equivalent of random guessing in the recurrent neural network as instantiated.

Introduction

The need for academia to be focused on a particular field, such as math or computer science allows for depth of study into topics directly relevant to those academic solutions. The specialization is necessary to develop a mastery over a topic yet leaves interdisciplinary studies in want of attention. Specifically, Political Science and Computer Science have a wide gap that traditionally is not given attention. However, the gap can be bridged through machine learning and natural language processing. Some in the academic field has begun to connect the two areas of study with a broader approach to the possibilities of computing the law. One example is MIT, which posted topics in their computational law category (MIT, 2020) about companies such as DONOTPAY (Browder, Greenwood, Wilson, 2020), which claims to be the world's first robot lawyer. Further investigation and contemplation allowed me to realize that I could use machine learning and natural language processing on supreme court case transcripts to make predictions about the case outcomes. I believe this would be a helpful endeavor for society, given that the law is to be predictable and stable. Understanding how the court thinks in a reproducible manner in machines is an extension of the reliability of the law.

When cases make their way through the united states legal system to the supreme court, they will result in a final decision on the court's position, even in the instances of dismissals. The final decisions are the final interpretation of the United States laws on the question brought to the court. For the court to determine the merits of the case, there must be a hearing and submitted documents to the court by which the court will hear from both parties, as the American legal system is adversarial. Since the law is ideally predictable, and going to court is expensive, it would be ideal to have a strong premonition of the court's jurisprudence to predict the ruling before ever attempting it in the courtroom. This is similar to investing in stocks—it is wise to know if the stock is predicted to increase in value or decrease in value, which is analogous to winning or losing a case.

The foundation of the constitution was to provide a stable means of rules for which society should abide by, and not change them unless the majority of the people under its jurisdiction agreed. The court system was an extension of this property to ensure fairness under the law. Since the time of our countries founding, computers have developed with extraordinary power. Fields such as machine learning and natural language processing, even computer science itself did not exist at the writing of the country's first documents. Yet political science may find its roots in ancient times, the first formations of an organized society. Despite the strong contrasts, technical studies can further promote the purpose and aim of the countries founding documents.

In court, the mind of the judge takes in the input of the courtroom communications, such as arguments and affidavits by the parties, then contemplates the writings of relevant laws, and how those laws should be interpreted regarding the facts and circumstances of the cases before them. This is like the process that a neural network built to process the same documents predicts the outcome of the case. A difference is that the neural network in this build is greatly simplified and will not consider the laws. Rather the networks will consider the pattern of words from previous case transcripts concerning the results of the given transcript to produce a prediction as to what the justices would rule.

Related Work

My project fits in with previous work in other computational projects based on legal documents or laws that have been done in several respects. First, several studies reach into the realm of computational law by studying aspects of the wording in legal documents, such as frequency, or sentiment. Second, several studies have developed recurrent neural networks to classify sentiments.

For instance, the study, "Plain English Summarization of Contracts" (Manor, Junyi, 2019), describes the process of taking legal corpora that are filled with complex law language as input to a model, then turning the input into a more plain language summary, with the focus on capturing the semantics. The model used n-grams of sizes between one and four. The study states the normal summarization techniques are less effective on legal documents due to a difference in the domain and further makes a call for the community to develop the area.

The work in my project contains similar aspects to the summarization, in that it uses the innate semantics of the communication to capture the sentiment of the court. This is done through the glove vector embeddings in the current project, whereas the previous study used n-grams.

Further, "The Extent of Repetition in Contract Language" (Simonson, Broderick, Herr, 2019) has made abstractions about the legal documents and legal corpora, such that they could analyze the usage of words within the documents. The study examined types of documents such as Non-disclosure agreements, Prime contracts, Purchase orders, Service Agreements, and subcontracts. K-nearest neighbors were used to discovering that non-disclosure agreements had the highest amount of repetition than the other documents. Logically this makes sense because they are binding the signee to silence on matters contained within the document, whereas the others are binding the signee to specific actions, which can be more nuanced and require extra language.

The work done in my project similarly makes abstractions about the legal corpora of the specific supreme court case transcript context through linguistic annotations such as tokenization, glove embeddings, and Parts of Speech tags. These abstractions are then used to discover how they influence the final decisions of the court in the neural network. However, my project differs from these studies in that they were not performed on case outcomes but rather on legal documents that have a standard form. My project will be on the court case transcripts.

More specific to using the patterns in previous studies, my project is like "Justice Blocks and Predictability of U.S. Supreme Court Votes" (Guimerà, Sales-Pardo, 2011), where the sitting court justice history on topic issues was used to predict the vote of the court. My project will also predict the outcome of the cases but using the wording of the transcripts rather than who is on the court, or what their previous positions are.

Approach

To have an idea of how the court will make its decisions, I needed to collect the supreme court case transcripts, which is how the justices asked questions, how the parties to present their cases, and the general interactions to determine which points are important to the court. Similarly, the communications in the courtroom influence and help to determine the resulting decisions of the court. To do this, the transcripts will be broken into their tokens, converted into their word embeddings, and then processed through a recurrent neural network for sentiment classification patterns. The case results are the sentiments, and the similarities of the cases will determine the sentimental patterns of the court's thinking patterns shown in their words.

I expected the given method to work because schools of thought have certain word markers and patterns of speech that delineate their schools of thought from others. Given that many previous analyses of the supreme court have been successfully made on the ideological affiliation of the sitting justices, it follows that sifting through the words of the given case transcripts for these justices will reveal clues about how their ideological affiliations are evident in the cases themselves, and when found, will point the neural network toward a sentiment—either to agree with the plaintiff or not, which works out into the gold labels of the transcripts.

For a successful machine learning project, from a pure implementation perspective, there is a minimum series of steps that must be made to form the project. These include collecting the data, formatting the data, choosing the correct model, building the model, training, and tuning the model, and testing the model.

Collecting Data - Sources

There are many records of the supreme court cases, but most of them are in PDF format, which does not have the most reliable extraction methods due to encodings. I first attempted to get the supreme court case transcripts by downloading them from the supreme court database, then used a pdf parser to extract the information. This had many errors and would have taken a significant effort to extract, even with the errors, which included unknown character encodings, subscripts in the middle of the text, and spelling mistakes. On further search, I found Eric Wiener's git repository, found at https://github.com/EricWiener/supreme-court-cases, that has cleaned cases from 1956 to 2018, tokenized them, and formatted them into structured data, in JSON format. Similarly, Walker Boyle has a repository at https://github.com/walkerdb/supreme_court_transcripts, of cases in JSON format. Both come from an undocumented API at https://api.oyez.org that is available for the public. I will consult both repositories, as well as any additional information I may seek out from the API for the data I will use in the project.

Collecting Data - Extraction, and Labeling

The repository of cases contained JSON files of the cases, with multiple files for one case, given they were extracted from an API to be served to a webpage. This meant that they had much more data than I was expecting to work within the project. The most relevant fields were the transcript section,

the number of justices joining the decision, and the link found in the case to locate the Justia resulting decision of the court in writing. While this project was not concerned with the written decision of the court, I used the label from the bottom of the case decision as to the golden label for the model.

To get the gold labels, I wrote a selenium (Software Freedom Conservancy (SFC), 2020) program, which is a web browser automation tool. I selenium used to extract all of the text from the court's decision and then search the last few lines for one of the following gold labels, which I called actions of the court:

- reversed and remanded,
- reversed,
- remanded,
- affirm,
- vacate,
- dismissed,
- approved,
- and denied.

If the text did not contain one of the labels in the final lines of the court's decision, then the case was labeled as "not found in the case" could be found. Going through all of the case URLs was possible because I placed the URLs extracted from the Walker Boyles OYEZ cases into a single text file, and then iterated them in selenium. The Justia website had the text for all cases in only two types of classes, "headertext" and "tab-opinion" embedded in the HTML, so I only needed to check for the first one, and if not found check for the second type. I could then extract the text, then store it into a text file locally to be further processed in the next step of the pipeline.

To ensure the accuracy of the extracted values, I took a sample of 20 cases from the extracted values and manually navigated to the Justia webpage to ensure the labeled actions was the actual outcome of the case. Of the cases I sampled, all of them were labeled according to the actual case outcome.

Formatting Data

After extracting all of the information, or at least knowing the fields to access in each file locally, I iterated through all of the cases, and formatted only the ones containing all three fields, the number of justices join the decision, the transcript, and the golden label. These were also formatted into JSON files, with each case being a separate instance of a JSON object containing the three fields. This formatting was necessary for the later step of inputting these formatted cases into the torchtext data loader of Pytorch.

The number of remaining cases after the filtering for the three fields, transcript, number of justices, and action type, left roughly five thousand eight hundred cases.

While the data was collected with eight types of actions, to simplify the results of the projects, I created a function to distinguish the eight classes into a binary result, with five of the classes in the zero encoding and three in the one encoding. Specifically, the following actions were encoded as one:

- reversed and remanded,
- reversed,
- approved.

Similarly, the following five were given the zero encoding:

- remanded,
- affirm,
- vacate,
- dismissed,
- and denied.

These labels were partitioned this was because the one-labeled actions are the results when the supreme court accepts the appellant's request or claim, and the zero-encoded actions are broader than rejecting the appellants' arguments or agreeing with the appellee, but essentially serve as a rejection of the case at the supreme court level, if not rejected.

Selecting the Correct Model

Given the sequential nature of case transcripts, such that ideas flow logically from the spoken words of the courtroom, which are subsequently recorded into transcript records, a recurrent neural network was selected. Recurrent neural networks output a current state and a hidden state, which can be used to select predictions about the states. Given that I was seeking to use the network for classifying the text into one kind of result, rather than labeling each word as is the manner in part of speech tagging, the intermittent output states could be disregarded, and the hidden states were passed to the next recurrent neural network cells until the final cell is reached. This last cell then passes the final hidden state to a linear layer to predict one of the eight possible sentiments, or classes for the case result. Selecting a single label as the results from a recurrent neural network is called one to many. This contrasts with many, called many to one, which is derived from there being many input words in the sequence flowing out to one single label.

Recurrent Neural Network

The network was defined in terms of Pytorch, so naturally, the *torch.nn.RNN* class was used. On examination of the Pytorch documentation, the defined class for recurrent neural network cells builds the hidden state by computing the product of the current input with its weight, then adds its bias, and then computes the product of the previous hidden state with its weight and adds its bias, and finally adds all of them up. The sum is passed into a hyperbolic tangent function, which results in the current cell's hidden state. This works out to, according to Pytorch documentation:

$$h_t = \tanh\left(W_{ih}x_t + b_{ih} + W_{hh}h0_{(t-1)} + b_{hh}\right)$$

(Pytorch, 2020)

The transcripts were tokenized using spaCy's tokenizer, (spaCy, 2020) and then embedded using glove embeddings (Pytorch, 2020). These embeddings were passed through the recurrent neural network, such that each word produced a hidden state computed through the above equation, which is then passed to the next words to continue the computation until the words have been exhausted. The Field class from Pytorch's *torchtext.data* library (Pytorch, 2020) handled the sizing of the batches, which was a convenience given some case transcripts were only a few thousand words, while others were more than fifty thousand words.

The results of the recurrent layer in the neural network are passed to a single linear layer. The layer takes in the size of the hidden state, which was three hundred. which is expected to output a single dimension, which is the label the model is guessing for the case transcript. The guess is between one and zero but is processed with the sigmoid function, so the tensors are between zero and one, and then rounded to the nearest integer so it is aligned with one of the two class options.

The objective function selected was the *torch.optim.Adam* (Pytorch, 2020) from Pytorch since it requires little tuning of the hyperparameters (Kingma, 2014), and slows the learning rate over iterations.

The loss function selected for the project was *nn.L1Loss*, since the options are only between 2 options, so if the selection is wrong, the error will calculate the mean of all of the differences in the predictions. Specifically, Pytorch documentation lists this as:

$$l(x, y) = \{l_1, \dots, l_N\},\ \mathrm{l}_n = |x_n - y_n|$$

(Pytorch, 2020)

Experiments

In any endeavor of discovery, there is a certain amount of permuting the possibilities until favorable results begin to become apparent. This can be in the form of hyperparameters, selecting different models, changing optimizer functions, changing hardware, etc. Before diving into the definition of the neural network, I had the decision type of the supreme court cases from the Walker Boyle repository derived from Oyez. While deciding which of the labels to use, the decision type regarding the number of justices joining the case decision or the action type regarding how the merits of the case are ruled, I created a graph of the decision types (figure 1), which shows the volume of decisions made and how the number of justices joining in them, and who was sitting chief justice at the time of the decisions. The years for the repository start with lumped years and shows that these lumped years do not have a high volume of cases. This is likely due to a lack of digitization, or lack of transcripts of the cases during these earlier periods, as the years do not become individual until 1955, which is one year after justice Warren became chief justice. It seemed like the actions of the court were a more valuable measurement of the case transcript than the number of justices joining the decisions, as the actions are less about the justices and more about the legal result.

Similarly, I built a graph to examine the paring of decisions of the court's actions with the number of justices joining the decision (figure 2). For both graphs, the color and size are measurements of the same attribute—the number of cases. That is the bigger circle, and the more vibrant the color, the more cases there is that fall into the category. The color scheme is called "magma".

Experimental Setup

To set up the experiment, the programs were written in Jupyter Notebook files. Next Pytorch's torchtext library was utilized to load the case text with distinguished fields, specifically the three fields of decisions, actions, and case text. Further, the fields for the cases were used in torchtext's Bucket Iterator class to create train and test iterators for the given batches. These iterators for each of the manual batch files were passed into the recurrent neural network for training.

Pytorch has the option to compute using the GPU or CPU, however, all parts of the computation must be in the same computation unit to be able to processes the functions. The GPU is faster, but I used the default CPU.

Initially, I attempted to run all 5843 cases through the recurrent neural network in batches created by the bucket iterator. During the tokenization and vector embedding stage, the memory was around 12.2 GB, which was doable, but then it attempted to allocate another 2 GB during the recurrent network stage, which put the memory requirements beyond the memory resources of my computer. As a result, I broke the cases into a series of randomized batch files, such that the cases were from random years and with random results.

Before encoding the gold labels as one or zero, I attempted to try a multi-class classification for the sentiment but found that the guesses were all defaulting to the same guesses, and so decided simplifying the results to binary options would make distinguishing differences easier for the model. The multiclass classification also caused me to change the loss function form *nn.MultiLabelMarginLoss* to the *nn.L1Loss*.

I encountered some difficulty with the Pytorch save functionality. I attempted to save my network with the following: *torch.save(model.state_dict(), PATH)* (Pytorch, 2020). However, I encountered and multiple attribute errors stating that tuples cannot be called on certain functions in save. As a result, I could not process my manual batches in sequence, and could not process all of the cases at once due to the limitation in local computer resources. Therefore, the results

could only be done on single batches of the whole. This is at most 1575 cases at a time. Even with the reduced number of cases in the batches, it would take up to eleven GB of RAM, which was compressed. The total committed amount of RAM with the batches on average was 18 GB of RAM.

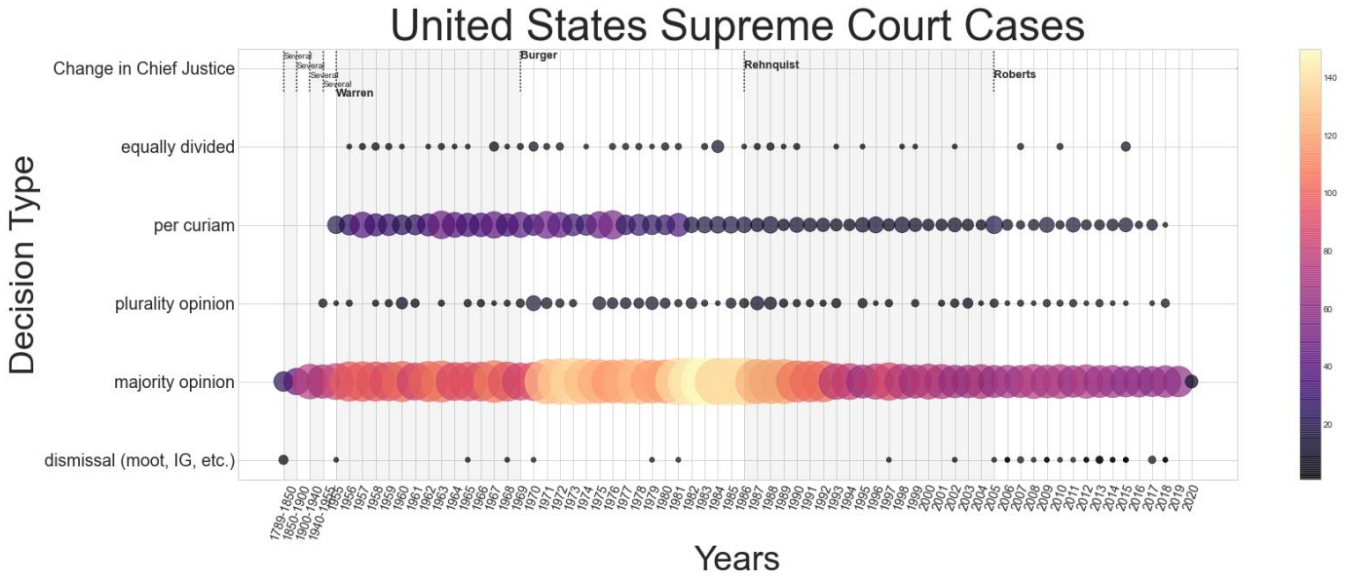

Figure 1.Shows the most decisions happened in the Burger court and are majority opinions. Ryan Barron, 2020.

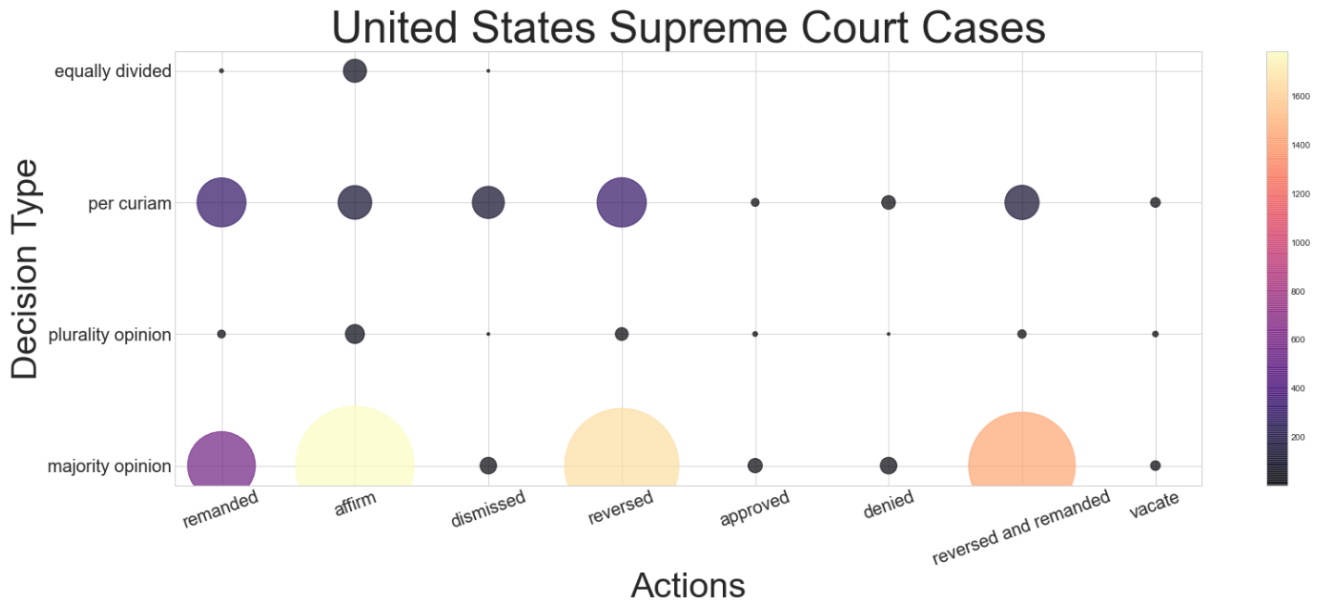

Figure 2. Shows types of decisions with their paired actions. Ryan Barron, 2020.

Experimental Results

After running the recurrent neural network through 10 iterations, with embedding dimensions of 200, the hidden state dimension size of 300, and batch sizes of 5, the number of correct was 43.11% on the training data and 41.07% on the test data, given that most of the predictions just stated a flat coverage of all of the word inputs, despite running for more than ten hours.

Discussion of Results

The case gold labels were encoded as one and zero. Given that the results were predicting around half as correct, the current results are essentially a random guess on the case transcripts. However, an examination of the guess label tensors for the batches showed that nearly all values in the guesses were identical except for a few, and therefore all guesses made about the case were essentially the same guess. This shows that the model as implemented cannot distinguish differences in the case transcript texts to be able to predict the outcomes of the court.

Conclusion

Given the case transcripts could not distinguish the case transcript better than a random guess means the other factors determining the outcomes. In the introduction, it was mentioned that the judges consider the laws themselves when considering the cases. It may be that the court follows a standard pattern of discovery for all cases, and the distinguishing marks of the justices rational come in the hidden chambers. Similarly, as mentioned in the () study, the party affiliation and installing party have high predictability of the outcomes of the cases, which means there may be more of an underlying bias in the justices to come to the conclusions rather than the outspoken evidence that was expected when the project was started.

As mentioned in "The Extent of Repetition in Contract Language" (Simon-son, Broderick, Herr, 2019), there is repetition in the law, and patterns are the nature of machine learning. As such machine learning and Natural language processing pair well with processing legal text. However, if this repetition is too similar then it may not be possible to distinguish sentiments.

The study "Plain English Summarization of Contracts" (Manor, Junyi, 2019), called for further development of the natural language processing tools and resources for the legal domain. It may be that the development in the legal domain will allow for the embeddings to distinguish legal language with a higher degree of significance placed on the nuances of structure and diction.

In the same way that the gold labels were extracted from the supreme court cases, there are other federal jurisdictions, and state jurisdictions that post their transcripts and rulings online. These records can likewise be extracted, processed, and trained in networks to find their patterned preference for a legal ruling. Such processing of the legal system could very well develop into a check on abuses of power from within the legal system, such as discrepancies of rulings between races, sexes, or other unacceptable distinguishable attributes in the parties bringing suit. Overall, analysis of the legal system with machine learning will improve both the availability, predictability, and equality of the law.

In terms of fairness under the law, in further studies, it may be a better approach to make a neural network to tokenize and vector embed the decisions in addition to the case transcripts and have the model predict the output decision rather than just the zero and one label.

References

Barron, R. 2020. Figure 1. Using matplotlib.

Barron, R. 2020. Figure 2. Using matplotlib.

Boyle, W. US Supreme Court Annotated Transcripts. October 2020. In *Github*. https://github.com/walkerdb/supreme_court_transcripts

Dan Simonson, D.; Broderick D.; Herr, J. 2019. The Extent of Repetition in Contract Language. *ACL Anthology*. https://www.aclweb.org/anthology/W19-2203.pdf

Guimerà, R.; Sales-Pardo, M. 2011. Justice Blocks and Predictability of U.S. Supreme Court Votes. In *PLOS*. https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0027188

Kingma, Diederik, P.; Ba, J. 2014. Adam: A Method for Stochastic Optimization. Cornel University. https://arxiv.org/abs/1412.6980

Manor, L.; Li, J. J. 2019. Plain English Summarization of Contracts. *ACL Anthology*. https://www.aclweb.org/anthology/W19-2201.pdf

Massachusetts Institute of Technology. 2020. Computational Law. https://law.mit.edu/

Oyez API. 2020. Oyez. https://api.oyez.org/user/login?destination=front

Facebook. 2020. Pytorch. https://pytorch.org/docs/stable/index.html

Software Freedom Conservancy (SFC). 2020. The Selenium Browser Automation Project. https://www.selenium.dev/documentation/en/

US Supreme Court Center. 2020. Justia. https://supreme.justia.com/

Subramanian, V. 2018. Deep Learning with PyTorch: A practical approach to building neural network models using PyTorch. Birmingham, Mumbai: Kindle.

Acknowledgments

The work done for this report was through UMBC's CMSC 678 – Machine Learning instructed by Dr. Cynthia Matuszek and also in conjunction with UMBC's CMSC 673- Natural Language Processing instructed by Dr. Frank Ferraro. Thanks for all of the help!