Affirm or Reverse:
U.S. Circuit Appellate Courts Decision Prediction, and Language Feature Generations From Textual Judicial Opinions

Yurui Mu Li Lin Qin Xue Yang
Advisor: Daniel L. Chen Elliott Ash

May 15, 2017

1 Introduction

In United States, there are 13 appellate courts that sit below the U.S. Supreme Court. All 94 federal judicial districts are organized into 12 regional circuits, each of which has a court of appeals. The appellate court's task is to determine whether or not the law was applied correctly in the trial court. Appeals courts consist of three judges and do not use a jury.

In this machine learning project, our major goal is using data from the past 134 years of circuit court appeals hearings to predict whether Courts of Appeals would affirm or reverse the trial court decisions, along with analyzing the causal factors influencing the judges' decisions. Potential factors include political parties the judges belong to, which indirectly affect personal sentiments recorded in the texts. Further and deeper conclusion can be made from our results, for example, whether this case could help legislation toward a certain direction and influences from the past cases.

2 Datasets

Due to the diversity of features we will be analyzing, the following datasets are used:

- Main texts from year 1880 to 2013 that documented the judges' majority opinions, concurring opinions and dissenting opinions.
- Case-level case information: contains 387898 entries with features related to judges' personal backgrounds like the political inclinations, birthplaces, education levels, etc.
- Facts descriptions from year 1980 to 1985, including 164 text files.
- Policy dataset from manifesto project, in which included political campaign speeches and political statements with hand-labeled topics. Some example topics are: culture, immigration, nationalization, democracy. The dataset also includes a column indicating positive or negative valence of each topic.
- LIWC dictionary dataset, which included 73 dictionaries that span a variety of topics: male, female, certainty and tentative words.

3 Data Cleaning

For the majority cases, we performed the following: first, we split the case level data into paragraph level data, omit page number, and put them into several folders: majority opinions, concurring, dissenting, and concurring in part / dissenting in part.

Next, we preformed other minor data cleaning according to different needs. In the facts dataset, we again split each text file into paragraph level data, and omitted two files which cannot be opened due to formatting problem. We also cleaned the string format in metadata which includes case ID, name of dissenting and concurring judge, and their respective political parties. For the policy dataset, we filtered out non-english files, as well as 'NaN' missing values appeared in multiple columns of dataset and create dummy variables for missing values.

4 Baseline Model

Although with a large amount of cases over the years, there are only 273592 cases that have a clear affirm or reverse labeling. Specifically, there are 201048 affirmed cases, and 72544 reversed cases. Then we built a baseline model for the court decision prediction.

To predict circuit courts' affirming or reversing judicial decisions from the texts of majority opinions, concurring opinions and dissenting opinions, we applied several binary classifiers as our baseline models, such as AdaBoost, Naive Bayes, and Random Forest.

First, we used count vectors as text representation method. The count vectors are sparse vectors retrieved from sklearnnfeature extraction package. The vocabulary used to build the count vectors are a dictionary of word to $3-$gram phrases, which was built previously using NLTK Sparse Tree technique with SnowBall stemming tools. And then we match each count vector with each court decision. Using count vectors alone, we reached the final prediction of court decisions.

4.1 Naive Bayes

Naive Bayes Classifier is a classifier that is most commonly used in text mining problems. "Naive" in Naive Bayes means a strong assumption of independence among all the features used in the text. With such prior assumption, we applied Bayes Theorem to predict the probability of our final label depending on each word occurrences. It has been a major baseline model for multiple text categorization problems since 1950s. Problems including Email spam filters, Text topic identifications, etc, have proved the effectiveness of Naive Bayes model.
In our problem, we reached an AUC score of 0.69 with Naive Bayes model.

4.2 Random Forest

As is indicated by the name of Random Forest classifier, we get to see that it is a "forest" made up by decision "trees". It is an ensemble learning method applicable as both regressors and classifiers. The "stumps"(decision trees) are randomly selected during the traing process, which reduce the probability of overfitting in decision trees methods. Without given an previous or latter context of features, random forest bags up the decision stumps together and reaches a final decision.
In our Random Forest baseline model using only the count vectors of each document, we reached a final prediction AUC score of 0.79 on our test set.

4.3 AdaBoost Model

AdaBoost is short for Adaptive Boosting, a machine learning algorithm first brought out by Yoav Freund and Robert Schapire. Based on multiple "weak learners", each of which is a decision function, AdaBoost is used in conjunction with many other types of learning algorithms to improve their performance. The output of "wak learners" is combined into a weighted sum that represents the final output of the boosted classifier. AdaBoost is adaptive in the sense that subsequent weak learners are tweaked in favor of those instances misclassified by previous classifiers.
During our AdaBoost parameter tuning, we tried $n\_estimator = 100$ and $n\_estimator = 200$, both gave us surprisingly high AUC scores 0.96 and 0.95.

4.4 Model Evaluation Metric

There are several evaluation metrics commonly used in Machine Learning models, such as recall, accuracy, precision, confusion matrix, and ROC (Receiver operating characteristic) and AUC (Area Under ROC Curve). The evaluation metric we are using in our baseline models is AUC score. AUC score is the area under ROC curve. We are using AUC as evaluation metric because it can give us a graphical representation of our model performance. Among all the models we used, we reached an AUC score of 96% with AdaBoost model, an AUC score of 79% for Random Forest model and an AUC score of 63% with Naive Bayes model. From the differences in performances of our models, we could see how different model mechanisms leading to different model performances.

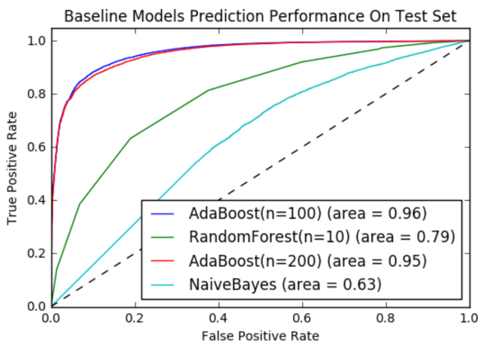

Figure 1: Baseline Model Performances Using AUC Metric

4.5 Model Performance Analysis

Now we analyze to find the reasons leading to the different performances of each baseline models. The poor performance of Naive Bayes should be related to the strong prior assumption of independence among features. However, given that our documents are all texts used in U.S. Circuit Courts, there are a certain ration of law and political terminologies massively used in the texts, which have a high correlation, rather than independence we assumed.

As for Random Forest, which we give high expectation on performances, gives the relatively low AUC score 0.79 on test set. And then we applied the same model on our training set, only to find an extremely high AUC score on our training set as high as 0.99. Such phenomenon tells us that our Random Forest has overfitted on training set. It might have remembered the exact paths of a decision was made in previous cases. Thus it gives us a perfect prediction accuracy on training set, and a low prediction accuracy on test set, which is not applicable in real-life.

At last, we get to AUC score of 0.96 with two of our AdaBoost models, with parameter n set to 100 and 200. Although the AUC score is very high. Our analysis is that we might have overlooked the decision key words in the court opinion documents. For example, in the case where judge decided to affirm the appeals, there are actually words like 'affirm' or 'affirmed', which have got very high weights in predicting process. Thus with such a combination of decision trees into Adaptive Boosting methods, we received a super high prediction accuracy.

However, we do not want such visible key words to affect our predicted results. So instead of directly using all the words and phrases appeared in the texts, We want to know how different the texts are as in different language features, such as sentiments, topics, topic valence, functional words used in the texts. Thus we spend the rest of the project focusing on feature engineering, as shown in the following section.

5 Feature Engineering

We devoted most of our time for feature engineering, since the generated features can be used for many purposes, including but not limited to predicting circuit court decisions. Furthermore, the number and the quality of features directly affects our model performance.

Due to the language sentiment nature of plaintiffs and defendants, we performed most of our analysis on paragraph levels, rather than by case.

1. **Paragraph Level LIWC Word Counts**

   Linguistic Inquiry and Word Count Dictionary, in short LIWC, is a text analysis method developed by American social psychologist James W. Pennebaker. As mentioned before, we were provided a total number of 73 dictionaries that covered a wide range of categories: from linguistics like pronoun, adverb, verb, to psychological words that directly express emotions, for example, joy, sad, anger, as well as topical categories like leisure, money. We calculate the percentage of words in a given case that fall into one or more of the categories.

   By performing LIWC analysis, our goal is to learn how words the judges chose to use in writing the cases reveal their thoughts, feelings, and motivations. We have counted the LIWC words for each paragraph in every case over the years, and later use them as an important feature for model training.

2. **Paragraph Level Sentiment Metrics Using VADER Model**

   Besides LIWC word counts, we incorporated Vader in Python NLTK package to help with our sentiment analysis. Vader is a lexicon and rule-based sentiment analysis tool that is specifically attuned to sentiments expressed in different texts. The output provides sentiment intensity and will be in the range of -1 (extremely negative) to 1 (extremely positive).

   Since the function runs in sentences, we tokenized our previously cleaned paragraphs into sentences and then assign each sentence 4 separate scores between -1 and 1. These scores denote the probability of being negative, positive, neutral and the final compound score, respectively. In particular, the compound score is the most representative, being the weighted average of the other three scores. By taking the average of all sentences in a paragraph, we obtained the paragraph level sentiment metrics. We also took dot product of sentiment and n-gram frequency across paragraphs for each case.

3. **Paragraph Level Word2Vec Vector Using Word Embedding Model**

   Word2vec model takes a large corpus of text as input and produces a vector space. Each unique word in the corpus was assigned a corresponding vector in the space. Words that share common contexts in the corpus are located in close proximity to one another in the space. We use the Gensim Word2Vec toolkit in Python to generate word embedding matrices for paragraphs and calculate an average word vector for each paragraph. Our output is an averaged vector of all the words that appeared in each paragraph.

4. **Paragraph Level Policy Topic Using Multiclass Linear SVC**

   For creating this feature, we built a multi-class classifier from the policy dataset, in which included political campaign speeches and political statements with hand-labeled topics such as culture, immigration, nationalization, democracy... By running the policy topic model, we can predict the political topic a case is referring to, and also incorporate the results into our final language features. There is a total of 47 different unique topics.

   The original policies are written in different langauges, so we filter out non-English texts using the Python package PyEnchant. By setting the threshold to 0.8, we keep documents with more than 80 percent words being English.

After careful consideration, we used three methods to generate features from texts: Tf-Idf, token counts and Doc2vec. For Tf-Idf and token count features, We used Sklearn package TfidfVectorizer and CountVectorizer to generate two matrices representing document features. And Doc2vec modifies the previously mentioned word2vec algorithm as an unsupervised learning method for larger blocks of text, such as sentences, paragraphs or entire documents.

Since this is a multi-class classification problem, machine learning models Linear SVC and Random Forest seemed suitable. Unfortunately, both models performed poorly using Doc2vec generated features. But the other two feature generating methods yield satisfying results. After tuning the model parameters using grid search and cross validation, the final prediction score for random forest and Linear Support Vector Classification reach 0.32 and 0.507 respectively. Thus we chose the better predictive Linear SVC model to assign policy topics for each paragraph.

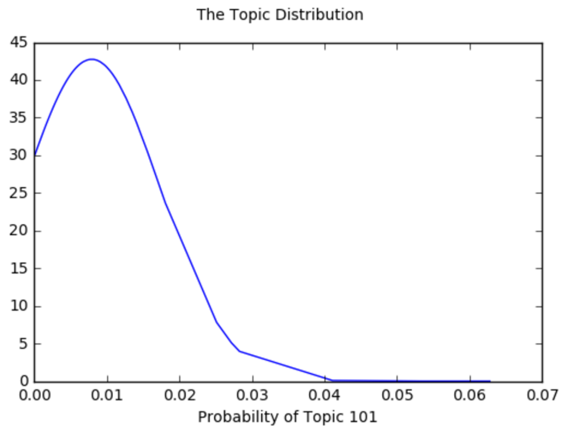

Figure 2: The Distribution of Topic 101

5. **Paragraph Level Valence Score Using Multiclass Linear SVC**

The policy dataset also includes a column indicating positive or negative valence for each topic. In the dataset, the valence column has three unique values: +, -, null. To simplify our labels, we convert the three labels to +1, -1 and 0, which means positive, negative and neutral. Similar to our policy topic feature, we built a separate classifier that predicts each paragraph's valence score with respect to its policy topics.

We split the combined text dataset into training data and test data with the ratio of 4:1, then generated matrices for Tf-Idf features and token counts from the texts. Then we again used grid search and cross validation to tune the hyper parameters of different models to optimize their performance. The model with the best performance is linear SVC with token counts as input, with an AUC score of approximately 0.74. With this model we were able to predict the distributions of valence score for each paragraph, which is an array of 3 values, i.e. the probability of being positive, neutral and negative.

We further investigated whether judges from different political parties show different distributions with respect to valence scores in their written cases. Hence we run our model in all cases and aggregate them by judges. In our dataset, each judge is labeled with party 0 or party 1. We graphed 6 different plots regarding positive, negative, and neutral valence score distributions in the two parties. While the distributions of positive and neutral valence look roughly the same, negative valence distributions show some divergence between the parties. From below graphs, since x-axis represents the absolute value of valence scores, we can infer that judges from party 1 tend to write more negatively.

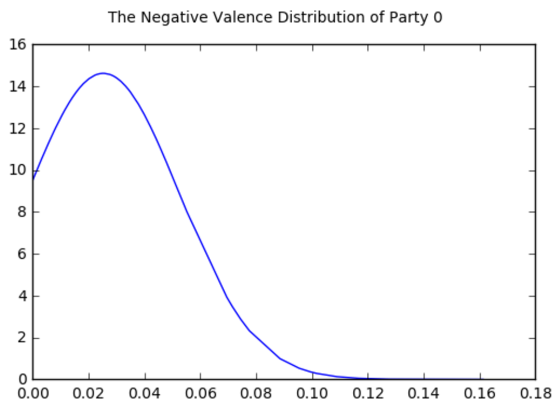

Figure 3: The Distribution of Negative Valence of Party 0

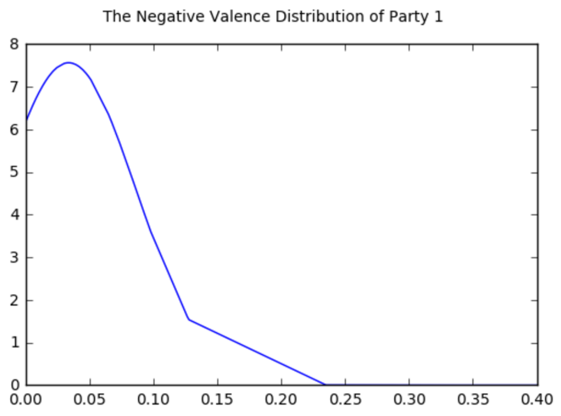

Figure 4: The Distribution of Negative Valence of Party 1

6. **Paragraph Level Fact Analysis Using Logistic Regression Model**

We want to find out a way to distinguish whether a paragraph is more related to law or facts in terms of writing style. Therefore we incorporated another dataset that contained facts descriptions from year 1980 to 1985. There are only 164 text files describing facts, which is much fewer than law cases in year 1980 to 1985. Therefore in order to balance the two labels in our dataset, we randomly selected a same amount of law cases in this time period.

To give a rough estimate of what are the differences in wording between law cases and facts, we first extract the most frequent n-grams in both datasets. The below graphs show some of the most common words that appeared in law cases and facts:

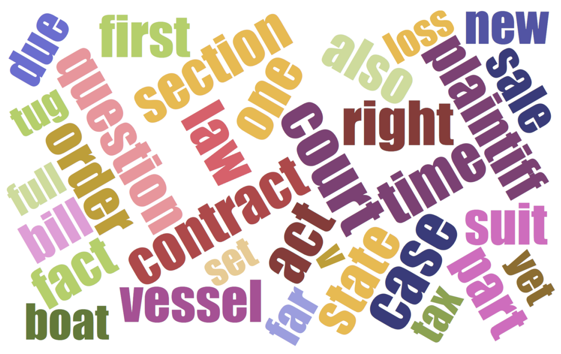

Figure 5: Most Common Words in Law Cases

From the graphs, we are able to point out some remarkable differences: law cases use words

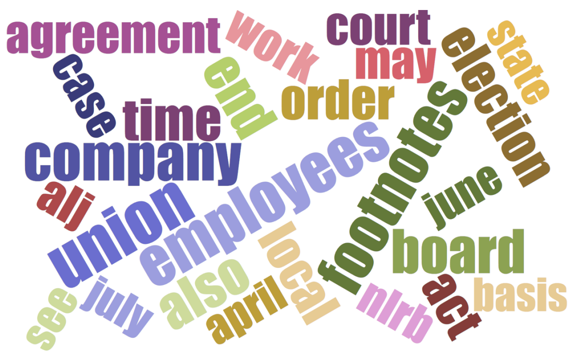

Figure 6: Most Common Words in Facts Descriptions

like law, contract, court, plaintiff much more often than facts. While on the other hand, facts have word choices like July, April, footnotes.

The next step is to build a classifier that separates law cases from facts. After several model parameter tunings, our final choice is logistic regression model with n-gram counts (up to trigrams) as features. With an accuracy of 0.91, the model is performing unusually well. Our guess is that a majority of the law cases contain distinct words that seldom appear in facts. Examples include "affirmed", "reversed", and "circuit court".

7. **Case Level N-gram Frequencies**

   We used NLTK package to tokenize and stem each word, and grammar sparse tree to capture meaningful expressions up to 3-grams. After computing n-gram frequencies for each case, words and phrases that appear in at least two cases a year in consecutively ten years are preserved, under the assumption that they are more meaningful compared to other less frequent words. The final n-gram dictionary consists of 25215 words. And the n-gram feature is later created by counting the number of occurrences of these words in every case.

8. **Document Level Citations**

   Since we are also interested in predicting future citations of a case by other judges, we create another important language feature, listing all citations that occurred in each case. Using regular expressions in Python, we defined a search pattern that is in the format "123 F.3d 10". After extracting the citations per paragraph, we summed them up to get citations per case.

6 Issues and Difficulties

Due to time constraints, we were only able to perform the above feature engineering. With more time, we will continue explore other measures that can be used in our predictive analysis. For example, Tf-Idf similarities between every two cases.

Another issue we encountered is the storage issue and time-consuming files transferring on HPC cluster server we used, NYU PRINCE cluster. When we tried to combine the language features we generated separately, we can fail the task because of insufficient disk quota on the server. Thus we spent a lot of time downloading, backup, and uploading files between our local computers, servers, and cloud storage, which is very time-consuming and low-efficient. Thus we need to find a way to solve such disk quota issue for our tens of gigabytes of data. Only if we stack them up together, we would be able to analyze more relations between language features and judges.

7 Further Analysis Using Language Features and Future Works

Now that we have gained massive language features from the textual judicial opinions. We are able to analyze how language features differ from judges, or from political parties the judges belong to. We are also interested to find out:

1. How political standings of judges influence the choices of words using LIWC words counts.
2. How judges differ in sentiments towards plaintiffs or defendants, generally and specifically on different topics, using VADER sentiment scores.
3. How do judges differ in inclinations toward different policy topics due to political parties of judges, by joining the policy topic data, valence data and vote-level judge info data.
4. We are also interested in finding the cross-citations of cases, to see if judges with similar political standings have similar citations. If so, can we use it to predict future judge decisions?
5. We are also interested in analyzing whether judges tend to manipulate languages and "twist" facts in language to push legislation on a topic towards a certain direction, by joining our generated fact vs law probability, political topic classified by machine, and valence score of each document.

8 Appendix

All of our results are produced by Python code under this link:
https://github.com/ym1495/dsga_1003_term_project

References

"Court Role and Structure." United States Courts. N.p., n.d. Web. 15 May 2017.
https://rare-technologies.com/doc2vec-tutorial/
http://scikit-learn.org/stable/
https://rare-technologies.com/word2vec-tutorial/
http://www.nltk.org/_modules/nltk/sentiment/vader.html
http://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html