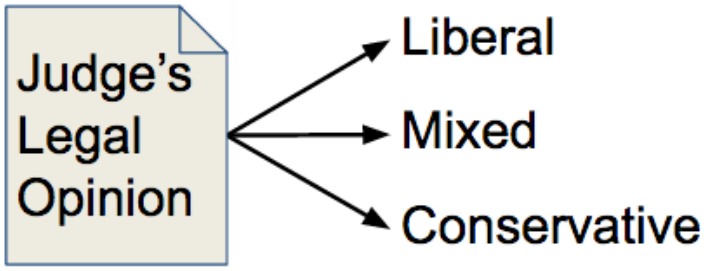
Inferring Political Valence from Written Judicial Decisions

Charlie Guthrie and Alex Pine
May 5, 2014



Abstract

Given n-grams from the transcript of an appellate court judge's legal opinion, our model classifies the opinion's political "valence". The exact definition of the valence depends on the case type, but typically falls into the category of "liberal" or "conservative". The model is trained on a set of appellate court cases with hand-coded valence values. Our goal is to predict a judge's valence from his written opinion with enough accuracy to eliminate the need for manual hand coding. Our model was able to outperform a simple majority classifier by first predicting the case's general category, stratifying the data on case category, and then building separate models on each stratum.

Introduction and Background

The inspiration and data from this project comes from the Judicial Research Initiative (JuRI) at the University of South Carolina. They created the "U.S. Courts of Appeals Database,"[1] which consists of 17,000 U.S. Appeals Court cases that have had their details coded into different variables describing their subject matter; this includes political valence codes for judges' decisions. In addition, Daniel L. Chen[2] has taken 300,000 written judicial opinions of Appeals Court judges from the JuRI project and converted them into n-grams, including the opinions coded by the U.S. Court of Appeals database.

The goal of this project is to use these n-grams to predict the political valence of the judge's opinion by using a machine learning model. Once the model has been built, it could be used to predict the political valences of the other 283,000 judges' opinions, as well as the opinions for cases held in the future. If the model was sufficiently accurate, it could assist or even

[1] http://artsandsciences.sc.edu/poli/juri/cta96_codebook.pdf
[2] https://scholar.google.com/citations?user=MlFf2igAAAAJ&hl=en

replace the tedious work of hand-coding the valences for these cases. It could also be generalized to predict other hand-coded features, and save a tremendous amount of manual coding.

Data

All data for this project comes from the "U.S. Courts of Appeals Database" created under the Judicial Research Initiative (JuRI) at the University of South Carolina. It is derived from historical documents detailing the thousands of cases presented to the U.S. Appeals Courts throughout the 20th century. The data is divided into three parts:

1. Coded metadata about individual cases
2. Files that map n-gram identifier numbers to stemmed n-grams
3. Files that contain the n-grams identifiers used in each judge's opinion document, along with the number of times each n-gram occured in the corresponding opinion document

Coded Metadata

The coded metadata data set consists of over 17,200 U.S. Appeals Court cases from the 20th century, summarized into 412 different variables meticulously hand-coded by legal scholars. It was presented to us as a 22 megabyte comma-delimited file. This data set includes subject matter categories for each case, their time and location, information about their judges, and so on. This data set contains the "political valence" of a judge's decision that we have tried to predict.

The political valence variable was coded on a scale from 0 to 3, loosely adhering to the following description:

3 = liberal
2 = mixed
1 = conservative
0 = apolitical/not defined

The exact meaning of these numbers varies depending on the type of case under consideration. For example, in prisoner petition cases, a "3" takes a position in favor of the prisoner, a "1" takes the opposite position, and a "2" represents a mixed opinion. In commercial or securities disputes, "3" represents the economic underdog, "1" represents the opposite, and "0" implies there is no clear economic underdog.

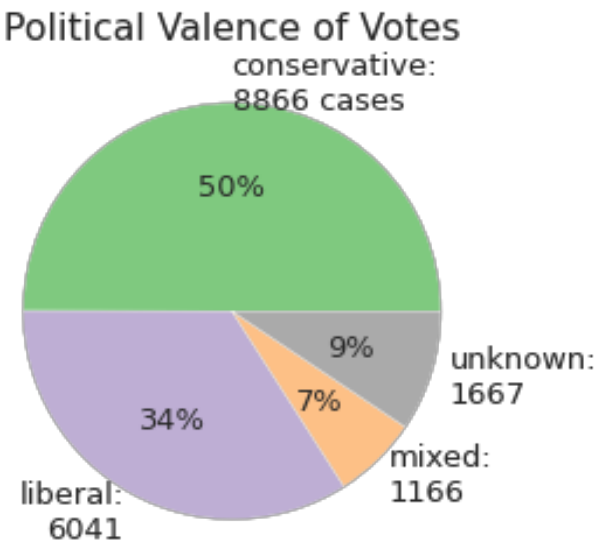


Fig. 1: Half of cases had a conservative valence

N-Gram Data

At the conclusion of an U.S. Appeals case, the presiding judges vote on the decision, and the majority decision determines the outcome of the case. Typically, one judge in the majority group writes his or her opinion for the group as a whole. Some of the other judges, however, will write their own opinions as well.

The opinions of judges from over 300,000 different U.S. Appeals Court cases were given to us as n-grams of sizes 1 through 8, paired with the number of times each one appeared in each text. Since there is sometimes more than one opinion per case, this resulted in over 400,000 sets of n-gram-count pairs, totaling over 40 gigabytes. This data was presented to us in over 1,000 pipe-delimited text files. In order to keep these files as small as possible, the files contained numerical identifiers for each n-gram in place of the n-gram text itself. The mapping between the n-gram identifier number and the n-gram text itself was presented to us in over 500 comma-delimited text files, requiring over 100 gigabytes. After filtering them down to the 17,000 cases with coded variables, there were still nearly 92 million unique n-grams in the dictionary.

Data Preparation

In order to infer the political valence of a case from the n-grams of a judge's opinion about the case, it was necessary to extract the n-grams corresponding to the cases for which political valences had been coded. Of the 17,200 cases whose metadata had been coded, roughly 1,700 of them had more than one judge's opinion written for them. However, there was only one political valence code for each case. Since the correspondence between the political

valence code and the opinion n-grams was unclear, we decided to only model cases where a single opinion had been written. This was roughly 15,500, or 90%, of all the coded cases.

Additionally, there were 1,667 cases that had been given the "apolitical/not defined" valence variable. We decided to treat these cases as though they had been given the "mixed" political valence, working under the assumption that the difference between an apolitical and a mixed political decision would be too subtle to be modeled accurately using n-gram based models.

After filtering the n-gram data down to the 15,000 cases for which we had political valence codes, we were still left with nearly 92 million different n-grams. We decided that it was necessary to filter these n-grams further before we did any kind of statistical modeling, because such a large ratio of features to examples would undoubtedly lead to overfitting. To do this, we wrote a program that counted the number of times each n-gram occurred across all cases, and discarded those that occurred less than a given threshold. We varied this threshold across different modeling runs in order to examine the effects of using different numbers of features. We discuss these effects later in the paper.

Once this preliminary round of feature reduction had been completed, the n-grams, and their corresponding political valence codes, were reordered chronologically. We did this so that the holdout set on which we evaluated our models would consist exclusively of cases that took place *after* those on which the model had been trained. This way, the evaluation would more closely reflect the situation in which our model would be used: to predict the political valence codes for new court cases that have not yet taken place.

Next, the matrix containing the n-gram counts for each of the 15,000 judge opinions, underwent the term frequency–inverse document frequency (TF-IDF) transformation[3]. This transformation simultaneously increases the weight of n-grams that appear several times within one document, while decreasing the weight of those that appear in many documents. This transformation was chosen so that "stop" words, such as articles, conjunctions, prepositions, and pronouns, could be automatically down-weighted without needing to decode the n-gram identifiers or compile a set of stop-words.

Lastly, the TF-IDF-transformed n-gram counts was stored as a sparse matrix using Scipy's Compressed Sparse Row ("CSR") format[4], and then saved on disk using the svmlight[5] format, which is designed to efficiently store CSR matrices. Using a sparse matrix allowed us to easily store the n-gram matrix entirely within memory, and saving it to disk allowed us to skip this data-modeling step on subsequent modeling experiments. The case identifiers and political valence codes corresponding to each case were also serialized and stored on disk in a separate file.

[3] http://i.stanford.edu/~ullman/mmds/ch1.pdf
[4] http://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.sparse.csr_matrix.html
[5] http://scikit-learn.org/stable/datasets/#datasets-in-svmlight-libsvm-format

The n-gram identifiers that correspond to each column of the feature matrix were stored on disk as well. These were used to extract the text of each of the n-grams from the files that contain the mapping between the n-gram identifiers to their text. This allowed us to look up the text of the n-grams that each of the trained models found to be the most useful in classifying the political valence.

Feature Selection and Modeling

Once the data had been prepared, our modeling process consisted of a two step pipeline:

1. Perform univariate feature reduction (on top of the n-gram filtering process done during data preparation).
2. Fit one of chosen classification models: Logistic Regression, Support Vector Machine (SVM), Multinomial Naive Bayes, and Bernoulli Naive Bayes.

This pipeline was run using Scikit Learn's "GridSearchCV"[6] process, which performs cross validation on all combinations of the hyperparameters that we specify. This way, all we had to do was specify the range of hyperparameters we wanted to test, and the pipeline process automatically chose the ones that resulted in the the greatest classification accuracy.

Feature Selection

During the data preparation step, we removed n-grams that did not have at least a minimum number of occurrences throughout the dataset. The thresholds for this process were purposefully picked to be low in comparison to the total number of n-grams, so that the majority of n-grams could be filtered using a more principled technique. The two techniques we chose to use were $\chi 2$ -based[7], and L1-regularized SVM[8] based feature reduction.

In the $\chi 2$ -based feature reduction runs Pearson's $\chi 2$ test between each column of n-grams and the target variable. The test indicates which of the n-grams are independent of the target category, and are therefore not informative in predicting it. Features that are determined to be sufficiently independent to the target category, judged by a given confidence level, are eliminated. The confidence level was included as a parameter in the grid search, so the value resulting in the highest accuracy was used. We let the confidence interval range from 0.4 (a high level of feature elimination) to 1.0 (no feature elimination).

Feature reduction using L1-regularized SVM simply trains a support vector machine using L1 regularization, and then returns the trained matrix, minus the features that were regularized to zero instead of predicting the results. The amount of regularization is controlled by a parameter that is included in the grid search, which we allowed to range from 0.01 (high level of regularization to 10 (less regularization).

[6] http://scikit-learn.org/stable/modules/generated/sklearn.grid_search.GridSearchCV.html
[7] http://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.chi2.html
[8] http://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html

There were more feature elimination options to choose from, but many of them weren't applicable to our project. For example, principal component analysis (PCA) can't be used on sparse matrices, and recursive feature elimination would have been too slow to run over thousands of features. These limitations, in addition to the limited time we had to run experiments, led us to only use $\chi 2$ -based, and L1-regularization SVM based feature reduction.

Models

We choose to do a three-way categorical classification, categorizing an opinion as either "liberal", "conservative", or "mixed". For this reason, we only chose chose models that were designed to handle multi-class classification. We used 75% of the n-gram data to train the models, and 25% of it to evaluate them, making sure that the latter data set occurred after the training data chronologically. Each model underwent its own round of cross-validated grid-search to find the optimal hyperparameters, with the classification accuracy being used to rank the different sets of hyperparameters.

Logistic regression with L2 regularization was the first classifier the we chose, and the only hyperparameter we included in the grid search was its regularization parameter, which we let vary between 0.01 and 10. To do the multi-class classification, we used the classifier's default "one-vs-all" behavior provided in the Scikit Learn package. We also used SVM with L2 regularization, varying it regularization parameter in the same range that we did with logistic regression.

We also used two variants of Naive Bayes: "Multinomial" Naive Bayes, and "Bernouilli" Naive Bayes. Both of these are designed to be used to do multi-class classification. The only difference between the two is that the Bernoulli variant treats each feature as though it we binary: any TF-IDF value greater than a small number (we used 0.01) is treated as a one, and everything else is considered a zero. Both of these classifiers had their priors fit to the training data. No hyperparameters were adjusted for these two classifiers during the grid search.

Initial Results

In order to evaluate the quality of our classifiers, we created a simple majority classifier that always predicted the most frequently occurring political valence in the training set. The accuracy of this classifier served as a baseline for our other classifiers.

The initial set of models did not perform noticeably better than the baseline majority classifier. The majority classifier classified 56% of the cases in the test data correctly, and the other models performed within 2% of that.

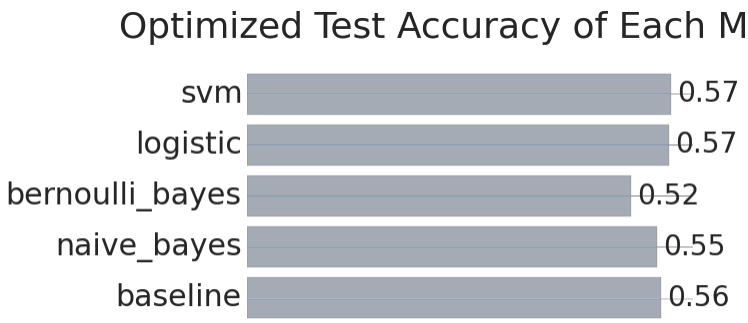


Fig. 2: SVM and logistic models, tuned with optimal regularization and feature selection parameters, performed the best, but did not do much better than baseline.

The model with the greatest accuracy on the test data was the Support Vector Machine model, which had test accuracy of 57.0%. Its average accuracy across the 3 folds of cross validation was 55.8%. This accuracy was achieved with a regularization parameter of 0.1, and, in the case of the $\chi 2$ -based feature reduction, a confidence level of 1.0. This means that the feature reduction performed in the data preprocessing step was sufficient for SVM.

The logistic regression model had the second best accuracy on the test data, 56.7%, and the highest average cross-validation accuracy, coming in at 55.9%. It used a regularization parameter of 1.0, and, surprisingly, it also used a confidence level of 1.0 for the the $\chi 2$ -based feature reduction.

These classifiers were run on data that had been pre-processed so that only n-grams that had occurred at least 100 times throughout the 15,000 cases would be considered. This had led to a data set with 23,534 different n-grams. The fact that the best two classifiers did not use any feature reduction beyond that which had been done in preprocessing made us believe that we may have thrown out too many features initially.

We reduced the threshold so that an n-gram need only appear at least 50 times to be considered, expanding the data set to include 52,923 different n-grams. Unfortunately, the results were nearly exactly the same for all four classifiers, plus or minus a few tenths of a percentage point. Logistic Regression and SVM were the best again, but this time they both used a confidence level of 0.8 for the $\chi 2$ -based feature reduction instead of 1.0. This indicates that the $\chi 2$ -based feature reduction was effective in improving accuracy--just not enough to improve the overall accuracy.

We tweaked other parameters as well, to no avail. For example, we tried using L1-regularized SVM to do feature-reduction, but it was no better than the $\chi 2$ -based feature reduction. We even tried ignoring the “mixed” labels during training, so that the classifiers only learned the “liberal” or “conservative” valances. This had no appreciable effect on accuracy, either.

In order to figure out what the classifier was doing, we took a look at the n-grams that had the largest coefficients in each of the models. Here they are for Logistic Regression and SVM:

Logistic Regression N-Grams

Conservative	Mixed	Liberal
sentenc, dividend, custod, appeal, invent, affirm, indict, convict	respect, bank, fault, remand, appeal dismiss, count, damag, contract	decis tax court, decis tax, custodian, deduct, new trial, remedi, revers remand, remand

Support Vector Machines N-Grams

Conservative	Mixed	Liberal
fault, lodg, save, know, fact suffici, new, conclud, clear	ct ed ann cas, ct ed, answer, assign error, favor, confess, requir, ct ed said	award damag, reach, consequ, consumm, recommend, proper, error instruct, assign error

Looking at these n-grams, it's clear that they are not random, but they are not clearly political terms either. There are many words that relate to the decision of the judge, such as "sentence", "remand", "indict", "fault", "convict", consequence", and "recommend". However, none of the n-grams clearly indicate a particular political valence. We expected the decision texts would be more polemical, so that the political valence could be distinguished by phrases common in political rhetoric. For example, we expected liberal decision to be identifiable from phrases like "equality", "discrimination", and "free speech", and conservative ones from "liberty", "religion", and "taxes."

The fact that the significant n-grams are not clearly political, coupled with the fact that the accuracy for all classifiers is always close to the majority classifier, implies that the classifiers merely learned the frequency of the three classes in the training data. There is no clear evidence that the classifiers were able gain meaningful insight from the n-gram occurrences.

The guide to the coded data[9] provided by the University of South Carolina describes in detail the situations under which each political valence should be assigned. This indicates that political valence is very context dependent: n-grams that are conservative in one context may be considered liberal in another.

[9] http://artsandsciences.sc.edu/poli/juri/cta96_codebook.pdf

For example, in “First Amendment” cases, a “3” is “for assertion of broadest interpretation of First Amendment protection”, and “1” is against such an assertion. On the other hand, in privacy cases, “3” is “for interest of person asserting privacy rights violated” and “1” is for the interest of the counterparty. See the table below for other examples.

Sample Decision Valances

Case Type	Sub-Type	Liberal	Conservative	Unknown
Criminal	Criminal	for the defendant	opposite	n/a
Economic Activity and Regulation	Tax	for gov. tax claim	for taxpayer	n/a
	Conflict over securities	for economic underdog	opposite	no clear underdog
Labor	suit against mgmt.	for union or individual	for mgmt.	n/a
	worker vs. union	for union	for individual	n/a

Incorporating Case Categories

The fact that the definition of political valence was so dependent on the case type suggested that we should take a different approach. Perhaps if we could use the n-grams from the text to predict coded variables other than political valence, we could use those variables in a second classifier that could predict the political valence.

We decided that the case category variables would most likely provide more context about political valence than any of the other variables in the coded data. There are nine different high-level categories that can be assigned to each case: “Civil Rights”, “Privacy”, “Labor Relations”, “First Amendment”, “Economic Activity”, “Due Process”, “Criminal”, “Miscellaneous”, and “Unknown”. Their frequencies in the data are listed in figure 3, below.

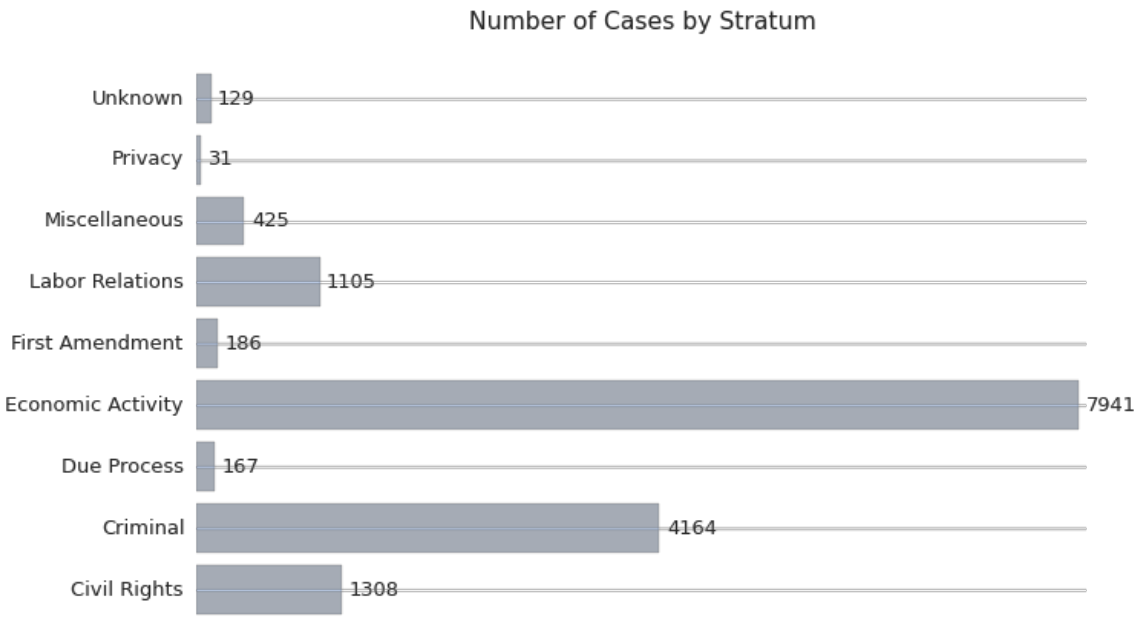


Fig. 3: Most cases concerned 'Economic Activity'.

Our models turned out to be much better at predicting case category than they were at predicting valence. All we did here was switch out the target variable, leaving the features and process the same. The chart below illustrates performance of the tuned models for predicting case category.

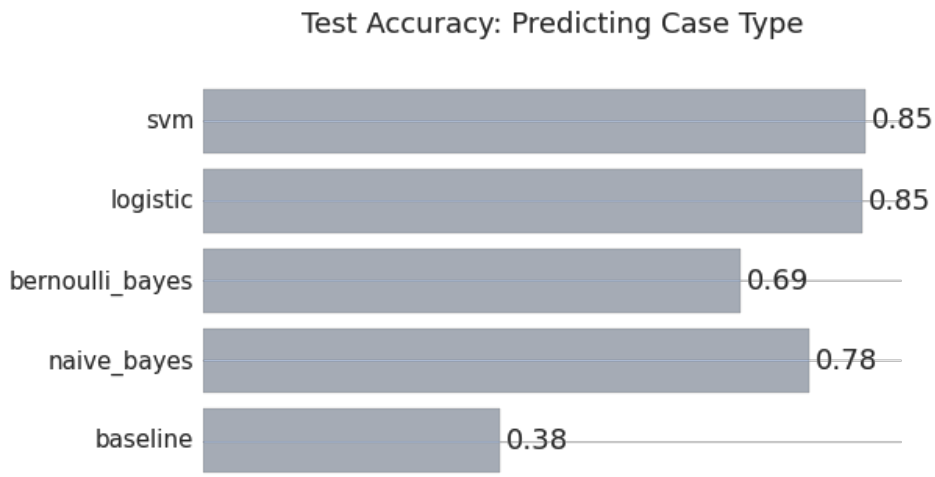


Fig. 4: A Support Vector Machine achieved the highest performance, at 85% accuracy.

Knowing that we could predict case type with reliable accuracy, we moved on to using case type to predict valence.

Case category as a feature

Initially, we simply included the case category of the judge's opinion as an extra feature, but this did not lead to any appreciable increase in accuracy. This is likely because all of our models were linear, so simply including the category would not cause an n-gram to take on different weights between categories. In order to do that, we would have to include features that represented the interaction between each feature and each category. However, we did not attempt this because it would have increased the number of features by a factor of ten, making the problem of a having too many features even worse.

Stratifying the data on case category

Next, we decided to stratify the n-grams by category, and train separate models on each stratum. This would allow an n-gram to imply a different political valence depending on the category.

This approach was much more successful than simply adding the case category as a feature. The categories that had a substantial number of cases, namely "Economic Activity", "Criminal", "Civil Rights", and "Labor Relations", had significantly higher accuracy than their respective baseline classifiers.

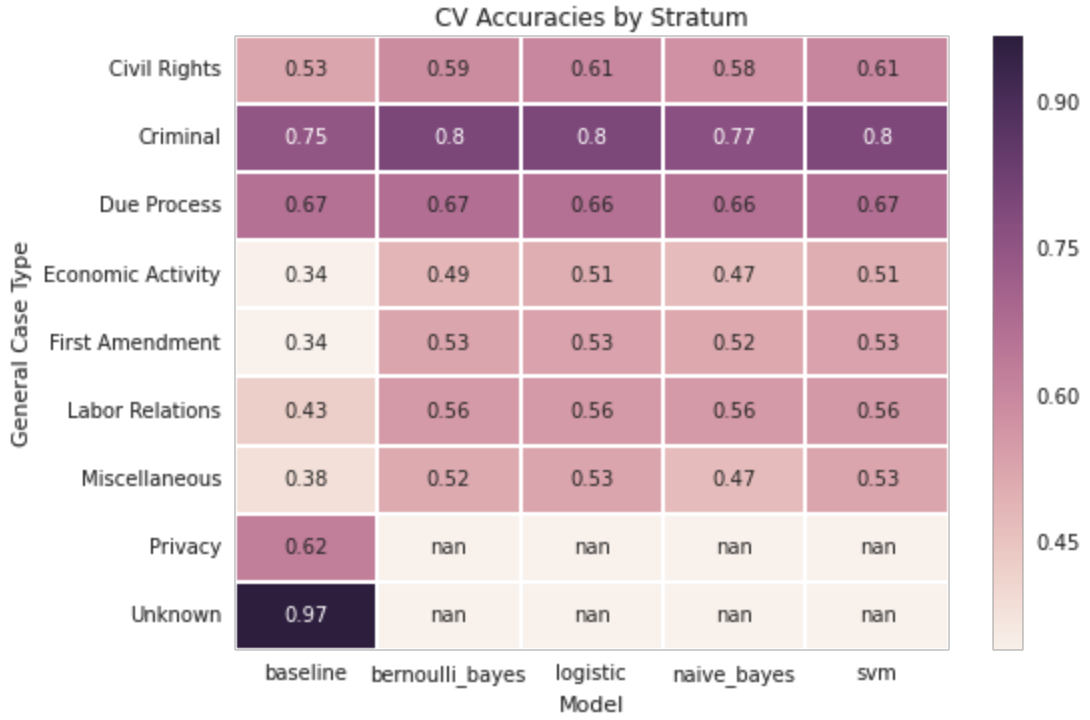


Fig. 5: Our models outperformed baseline in several of the strata that had the most data. In the 'Economic Activity' stratum, most notably, the logistic regression model achieved an accuracy of 51% - 17 percentage points better than baseline's 34%.

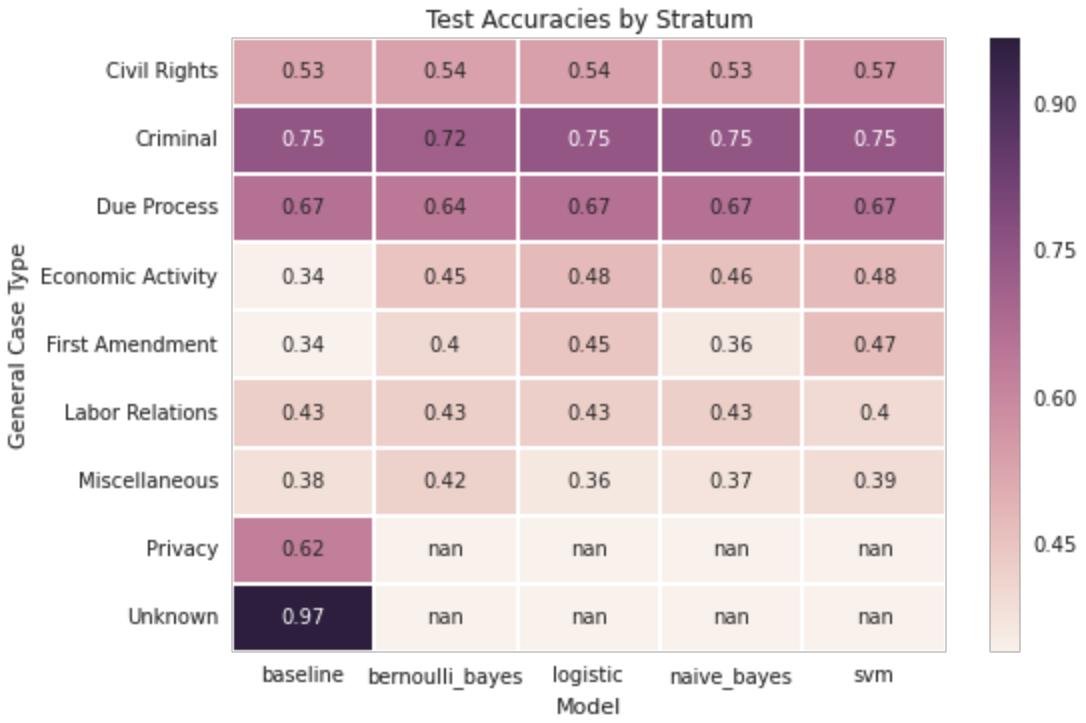


Fig. 6: Accuracy on the test data.

For example, the best accuracy for the “Economic Activity” category, achieved by the Logistic Regression model, was 17% better than its baseline classifier (51% vs. 34%). Similarly, the best classifier (again, Logistic Regression), did 5% better than its respective baseline.

Below are some of the n-grams with the largest coefficients in the Logistic Regression model from the “Economic Activity” stratum:

Conservative	Mixed	Liberal
lowest, govern, refere, case, month, gravamen, plaintiff, dividend, infring patent	lower court, arg, pray, curiam, bank, citi new, commit, compani, commission intern	recurr, defend, claim titl, dismiss prejudic, ag, injur parti, matter law, juri, petition

Although many of these words have no clear political meaning, many of them do. For example, in economic disputes, the codebook indicates that a “liberal” political valence be applied if the “economic underdog” won the case. It is typical for a company to sue an individual over patent infringement, so it makes sense that the “infring patent” n-gram is an indicator of “conservative” political valence. A similar argument could be made for n-grams like “dividend”, and the same argument could be made for the liberal valence for the “injur parti”, “ag” (most likely referring to the word “age”), and “petition” n-grams.

However, many categories had a very small number of cases associated with them, making the positive difference in their accuracy less trustworthy, as they are likely to be overfit. Two categories, “Privacy” and “Unknown”, had so few categories, and were so biased towards one political valence, that the classifiers could not be successfully trained on them. For this reason, it may only be possible to accurately predict valence in certain case categories.

Conclusions and Future Work

Despite extensive tuning and experimentation, our models are not yet able to predict valence from unstratified n-grams with reliable accuracy. As Thomas Edison famously stated, “I have not failed. I've just found 10,000 ways that won't work.” However, we were able to achieve better performance by stratifying the data on case category, suggesting that performance could be improved using a more sophisticated, multi-part model. Furthermore, the groundwork laid here can be used as a platform for future experimentation and exploration of the data. The techniques employed here, though extensive, are by no means exhaustive.

The accuracy we were able to achieve in predicting case type was very promising. The n-gram data may be well suited to predicting many of the other 412 coded variables. This may be a useful suggestion tool to aid in coding more documents, or even a replacement for the extensive human labor. The model could provide its best guess for a given coded variable. In addition, it could provide the words and phrases it used to arrive at that guess along the context of the original document.

It might also be possible to predict political valence from information about the judges that wrote the opinions, such as their age, their political party, etc. These variables are included in the coded data, but they are sparsely populated. If we could find this data from another source, or predict it from the n-grams, it could be very useful in predicting the political valence of their future opinions.

All the source code used to compute our results can be found at: https://github.com/pinesol/appeals.