CS230

Measuring Political Polarization of Federal Regulations

Cindy Chung
SUNet ID: cchung13

Jake Jares
SUNet ID: jjares

Mary Zhu
SUNet ID: maryzhu

1 Description of Overall Task and Background

We have trained a LSTM recurrent neural network on text sequences from the Code of Federal Regulations to investigate the degree of political polarization in federal rulemaking. Our baseline objective is to solve a supervised learning problem: predict the partisan composition of the Congressional coalition that enacted a given regulation from the text of said regulation. The performance of this prediction in different areas (e.g. different agencies or different years) can then be interpreted as a measure of relative polarization. For example, if one can more easily discriminate "Republican" rules from "Democratic" rules at the EPA than at the Department of Transportation (DOT), then one might infer that rulemaking at the EPA is more "polarized" than at the DOT. Thus, the outcome we predict is a value reflecting the partisan leaning of the combined House and Senate coalition that initially passed the authorizing statute for a regulation. We have defined two measures of partisan lean for this task:

$$y_i = \frac{D_{A_i}}{D_{A_i} + R_{A_i}} \tag{1}$$

$$y_i = \frac{D_{A_i}}{D_{T_i}} - \frac{R_{A_i}}{R_{T_i}} \tag{2}$$

where, for piece of legislation i, D_{A_i} and R_{A_i} reflect the number of Democratic and Republican "Aye" votes, respectively, and D_{T_i} and R_{T_i} represent the total number of Democratic and Republican votes, respectively. Measure 1 reflects the percentage of the enacting coalition comprised by Democratic members, and thus ranges between 0 and 1. A value of 0.5 reflects perfect bipartisanship (in this particular sense), whereas $y > 0.5$ indicates a Democratic lean to the enacting coalition. Measure 2 reflects the difference in the proportion of Democrats voting in support and the proportion of Republicans voting in support. This second measure therefore has a range of -1 to 1, with a value of 0 indicating bipartisanship.

A key measurement issue arises in the fact that many public laws are enacted without a recorded roll call vote, and instead lie on other more opaque passage procedures. To deal with sporadically missing data in one of the two chambers needed for passage (House and Senate), we create two different measures (within each of Measure 1 and 2) either "interpolating" missing votes from the problematic chamber or "dropping" them. As described in the next section, this choice is ultimately not very consequential to the measure.

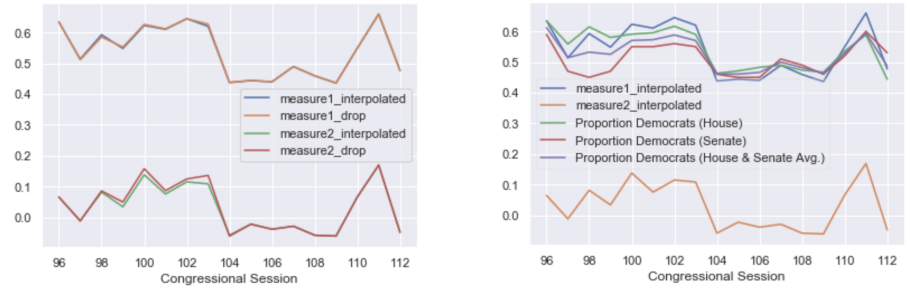
Several studies have researched similar topics, and serve as guidelines to reference as we develop our own unique approach. Iyyer et. al (2014) applies a recursive neural network to identify the political position evinced by a sentence. By annotating their dataset with crowdsourced political annotations at the phrase and sentence level rather than relying on a bag-of-words model or hand-designed lexica, their model detects bias more accurately than existing models. Du et. al (2017) addresses the critical problem with traditional aspect-level sentiment classification when attempting to perform stance classification - that is, that the identification of stance is dependent on the target, which might not always be explicitly mentioned text sequences used for training. They construct a neural network-based model that uniquely incorporates target-specific information into stance classification using an attention mechanism locating the critical parts of text which are related to target.

2 Dataset

2.1 Description

Our training corpus involves the text of the Code of Federal Regulations (CFR), broken up into each year-title-part level of organization. Specifically, each yearly edition of the CFR is divided into 50 titles representing broad areas under Federal

regulation (i.e. Food and Drugs, Wildlife and Fisheries, Energy, etc.), which are further divided into parts covering specific regulatory areas. We obtained cleaned text files for all 338,832 year-title-parts of the CFR between 1970 - 2019. We then scraped the U.S. Code database and used a public crosswalk to link each year-title-part to the Congressional bills authorizing each rule. Finally, we linked our set of bills to Congressional voting records. Our final linked dataset contains 158,123 samples.



(a) Average Outcome Measures 1 and 2 by Congressional Session

(b) Average Outcome Measures 1 and 2 vs Congressional Composition

Figure 1: Dataset composition and distributions.

Figure 1(a) depicts the annual average of our four prospective outcome measures. Figure 1(b) captures the average outcomes between the measures calculated by equation (1) vs equation (2) as well as the Congressional composition between Democrats in the U.S. Senate and the U.S. House of Representatives. These suggest that, despite a different center between Measures 1 and 2, the four different metrics described above capture very similar changes. We thus focus mainly on the "interpolated" version of Measure 1.

2.2 Preprocessing

Each text sequence in our dataset is associated with a year-title-part in the CFR. These textfiles are usually quite long, with most in the thousands of words and some in the millions. This has required substantial preprocessing to make running our models computationally feasible. For data reliability reasons, we have limited our focus to the 20 years 1997-2016 in the CFR. Our preprocessing approach from this point focused on retaining as many observations as possible. We first remove all stop words (meaningless words like "a," "the," "of," and "to") as determined by the Python NLTK library. Afterwards, we trimmed all text samples to just their first 1,000 words, and then filtered for text files between 200-1,000 words. This left us with over 100,000 files to work with. Given that most of these trimmed text files contain 1,000 (unpadded) words, training time remained a very binding constraint through the model development process, and our ability to try out a very wide array of hyperparameter specifications was somewhat limited.

3 Recurrent Neural Network: Hyperparameter and Architecture Choices

As the outcome we desire to predict is continuous, we found it natural to fit a regression model by minimizing a mean-squared error loss function.[1] Our regression model applies up to three long short-term memory (LSTM) RNN layers and up to two linear layers to an input sequence. Using this Torchtext framework guide, we implement 620-820 features in our input embedding layer (which represents the individual words of our text data as real-valued vectors in a pre-defined space). We also added options for recurrent dropout, and ran our model for at least 5 epochs per run using the Stochastic Gradient Descent optimizer. Tuning our hyperparameter choices unloving starting out with smaller sample sizes (as seen in previous milestones) for computational ease. Hyperparameter choices from these earlier (small-sample) iterations guided out choice for the full sample. Hyperparameters were selected across 6 dimensions (more on this to follow), of which the learning rate proved especially salient.

The full dataset of over 100,000 observations is divided into train, dev, and test sets with a .7-.15-.15 split. This particularly conservative split is motivated by our need to create a large enough test set to ensure a more accurate analysis and application of our politicization measure. Specifically, to measure the performance of the algorithm to predict the partisan origin of rules across different agencies, there must be a large enough set of test set rules for each agency we'd like to evaluate. The current split leaves us with over 15,000 rules in the test set, which is enough to provide an accurate error estimate within large executive departments. The dataset is randomly split into the train/dev/test split according to two different random sampling methods: "unclustered" and "clustered."

[1] In our milestone submissions, we instead framed our problem as classification, as it allowed us to more easily use off-the-shelf code for our early model iterations. However, our final goal is fundamentally a regression problem, and is therefore the focus of our final writeup.

For our "unclustered" split, we simply divide the data into train, dev, and test sets and randomize based on year-title-part, as we had done for our previous two Milestones.

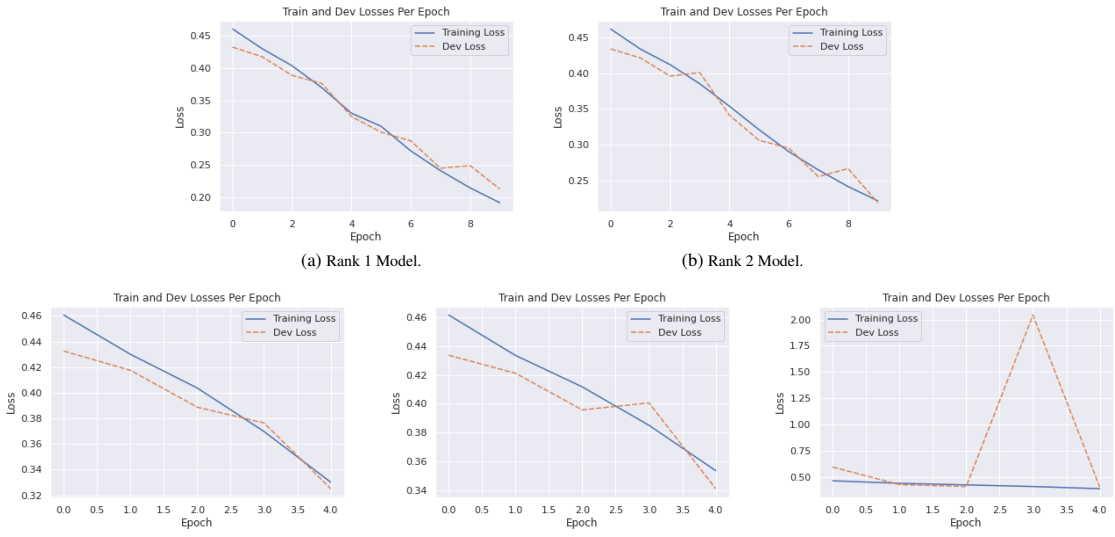
Our "clustered" split randomizes by clustering at the title-part level instead of the usual year-title-part level. The reasoning for this method is that in our data, many of the year-title-parts do not change significantly across time meaning our text sample observations at the year-title-part level are very similar, with few major modifications. On one hand, this is helpful for our model to learn and train better since the train set is representative of our test set. On the other hand, for the purposes of our post-model analysis, we want to see if we can predict the political origins of text from of some title-parts using text from other title-parts. Therefore, good performance on a "clustered" split would be more useful for our goal of measuring polarization from the text of rules.

While in principle we might obtain a different split other than the specified .7-.15-.15 split in terms of observation count, our dataset includes so many title-parts (nearly 7,000) with similar amounts of year-title parts that we actually achieve roughly the same split on a year-title-part level too.

4 Results

4.1 Unclustered Regression Model

Table 1 and Figure 2 display the train and dev losses for the best performing model parameter combinations found in our iterations. Although we had experimented with multiple combinations of a wide range of hyperparameter values, we observe some general behavior among the top 5 highest performing models revealed below for better performing results. First, all top performing models are run using a Stochastic Gradient Descent optimizer, as we found through many iterations of testing that an Adam optimizer was quite unstable to use for regression models. We also found that the most accurate models all had a batch size of 8, momentum of 0, embedding size of 820, 256 hidden dimensions, 2 LSTM layers, and a learning rate of 0.1.



(a) Rank 1 Model. (b) Rank 2 Model.

(c) Rank 3 Model. (d) Rank 4 Model. (e) Rank 5 Model.

Figure 2: Train and dev MSE losses per epoch on unclustered dataset.

Table 1: Accuracy results and hyperparameter values of unclustered models outputting Figures 2(a) - (e), sorted from most accurate to least accurate by MSE score.

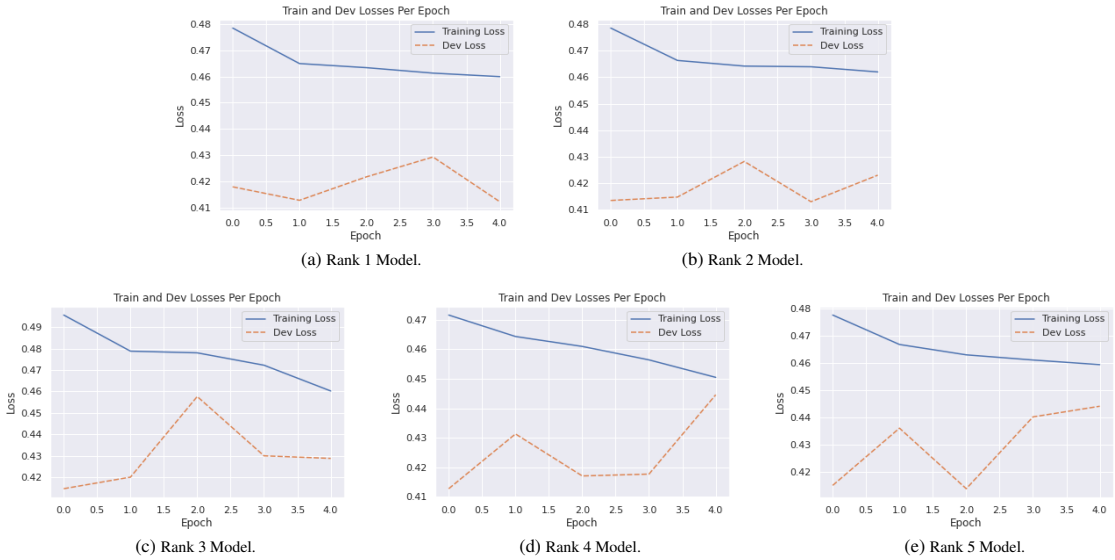
Rank	Dev MSE Loss	Test MSE Loss	Recurrent Dropout	Total Epochs
1	0.002009	0.002399	0.0	10
2	0.002082	0.002406	0.1	10
3	0.003037	0.003101	0.0	5
4	0.003174	0.003204	0.1	5
5	0.003699	0.003699	0.3	5

Given time and computational constraints, we trained each model for only 5 epochs, and then examined the train and dev losses to determine the model specifications with the lowest dev MSE losses. We singled out two models (Ranked 1 and 2 in Table 1) that performed with relatively low dev MSE losses, and trained each one for an additional 5 epochs (for a total of

10 epochs each). We found that additional training further lowered the dev losses. Out of all of our iterations, the model depicted in Figure 2(a) has the lowest MSE score. It uses 2 LSTM layers followed by 2 linear layers to produce a Dev MSE Loss of 0.002009 and Test MSE loss of 0.002399.

4.2 Clustered Regression Model

Figure 3 and Table 2 display the train and dev losses for the best performing model parameter combinations found in our iterations over the clustered sample. As with our unclustered model, although we had experimented with multiple combinations of a wide range of parameter values, we observe common features among the top 5 highest performing models revealed below; the following models are all are run over 5 epochs with a batch size of 8, momentum of 0, embedding size of 620, and 64 hidden dimensions.



(a) Rank 1 Model. (b) Rank 2 Model.

(c) Rank 3 Model. (d) Rank 4 Model. (e) Rank 5 Model.

Figure 3: Train and dev MSE losses per epoch on clustered dataset.

Table 2: Accuracy results and hyperparameter values of clustered models outputting Figures 3(a) - (e), sorted from most accurate to least accurate by MSE score.

Rank	Dev MSE Loss	Test MSE Loss	Number of LSTM Layers	Learning Rate	Recurrent Dropout
1	0.003733	0.003994	2	0.1	0.3
2	0.003835	0.004046	3	0.1	0.5
3	0.003853	0.004245	2	0.3	0.4
4	0.003988	0.004264	2	0.1	0.1
5	0.003988	0.004360	2	0.1	0.5

Given time and computational constraints, we trained each model for only 5 epochs to determine the model specifications with the lowest MSE loss, and then trained those models over additional epochs in an attempt to further drive dev loss down. However, training more on these models led to no significant improvements. Nevertheless, the model depicted in Figure 3(a) minimizes on MSE loss the most; it uses 2 LSTM layers and a linear layer at the end to produce a Dev MSE loss of 0.003733 and a Test MSE Loss of 0.003994.

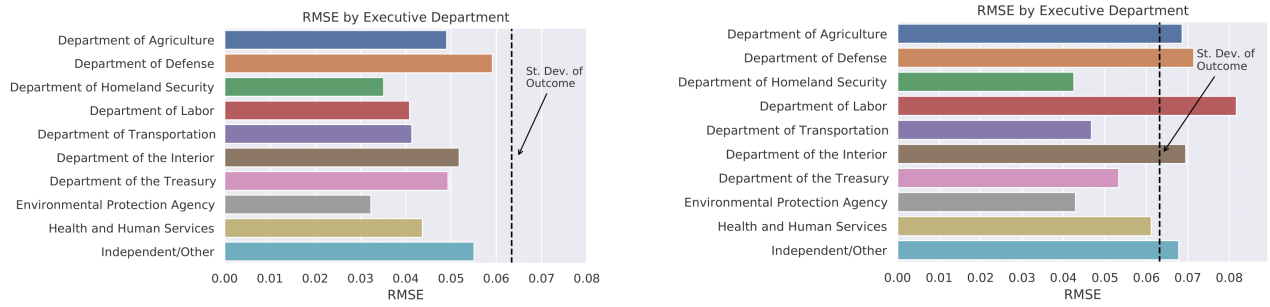
5 Analysis

5.1 Comparison Between Unclustered and Clustered Model Results

We observe a considerable difference between the performance of the regression models when run on the unclustered data in comparison to the clustered data. All models fed with unclustered data featured in Figure 2 and Table 1 beat the null error rate threshold of 0.004, and perform with significantly higher accuracy. In contrast, we observe that none of the models fed with clustered data displayed in Figure 3 and Table 2 beat the null error rate (except for the Rank 1 model by an extremely small fraction), indicating that they cannot be used to make predictions. This is due to the fact that there are more representative samples present in the train and test set when using unclustered splits, as explained in Section 3.

5.2 Measuring Polarization of Specific Agencies

The original goal of this project was to measure the relative political polarization of in federal rulemaking. To relate back to our initial objective and to further analyze our results, we select the 10 agencies associated with the most text files, and calculate the mean squared error according to the regression model for each agency. The following bar charts reveal the ability to predict the partisanship of each agency based on its regulatory text.



(a) Unclustered model results.

(b) Clustered model results.

Figure 4: RMSE by U.S. executive department.

Figure 4 displays the RMSE of regulatory text fed into both our unclustered and clustered models compared to the standard deviation of the outcome variable. Evidently, the results align with our theoretical expectations. In Figure 4(a), which displays the results generated by the model run on unclustered samples, we see that the standard deviation of the outcome is 0.063. An often percieved "liberal" agency, the Environmental Protection Agency, has the lowest RMSE of around 0.032, while the Department of Homeland Security (the department housing ICE), has the second lowest RMSE of around 0.035. Indeed, expert perceptions of agency ideology are consistent with these two agencies being among the most ideologically extreme (Richardson et al. 2017). The accuracy of model predictions for these agencies indicate that their political leanings are the most predictable from their rulemaking text. We observe the same pattern in Figure 4(b), which displays the results generated by the model run on clustered samples. Here, we see that the two agencies with the lowest RMSE values are again the Environmental Protection Agency and the Department of Homeland Security, both with slightly higher scores relative to the unclustered version of approximately 0.042.

6 Discussions and Conclusions

We ultimately find that, when using unclustered splits for our train, dev, and test sets, our LSTM RNN is capable of predicting the political leanings of U.S. executive departments through their regulatory text with reasonably high accuracy, as measured by Dev and Test MSE Loss. However, our models using clustered splits displayed significantly poorer performance. The most probable reason for this is that the clustered models we present in Section 4.2 are ill-fitted with the parameter combinations we tested them on. As mentioned in Section 2.2, due to the vast size of our dataset, the models' computational limitations, and the time constraints this posed, we were unable to test our model's performance on the full dataset as thoroughly as we would have liked to. Given how much our clustered results in Figure 4(b) resemble our expectations, there is a reasonable chance that, with further finetuning, our models' performances could have been increased.

Another possibility we've considered is that perhaps the regulatory texts of U.S. federal agencies are not overtly political. If so, then regardless of how optimized or how suitable the parameter combinations of our model, it could be that U.S. federal regulatory texts are not distinctly political, and it is thus inherently impossible to predict their political leanings.

Finally, we considered that training a high-performing RNN may be difficult without feeding representative samples to the training set. Indeed, this consideration aligns with what we learned in our course curriculum.

References

[1] Du, Jiachen, et al. "Stance classification with target-specific neural attention networks." International Joint Conferences on Artificial Intelligence, 2017.

Iyyer, Mohit, et al. "Political ideology detection using recursive neural networks." Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). 2014.

Richardson, Mark D., et al. "Elite Perceptions of Agency Ideology and Workforce Skill." Journal of Politics (Volume 80, No. 1), 2017.

Contributions

Cindy led research on model architecture selection, created the structure for our main regression and classification programs, created results visualizations, tuned model hyperparameters, and contributed to all report submissions and the presentation. Jake collected and built the data used in the project, created data visualizations, helped with hyperparameter tuning, and edited report submissions and presentations. Mary spearheaded each report writeup and the final presentation, worked on hyperparameter tuning, wrote code for data preprocessing, and helped build the initial dataset.