CSC 396: Predicting Outcomes of Supreme Court Cases from Oral Arguments

Sedona O'Hearn, Viswa Karuturi, Keenan Fiedler

Introduction

Humans are fallible and easily influenced by both conscious and unconscious bias. In areas like the legal system where decisions must rely on evidence, this can be an especially problematic trait. With the rise of artificial intelligence in nearly every sector of society, including the judicial system, this also means that the algorithms themselves can be susceptible to biased data.

As the rulings for supreme court cases can have massive consequences, we chose to set our focus there. The aim was to construct a model that is trained on oral arguments, to then see how well it does on predicting which side of the party should be ruled in favor of. A key aspect to our project was deciding the best way to train the model, and more specifically how the dataset should be cleaned. Analysis was done on three different dataset samples and we evaluate their performance in this report.

Core Code Implementation

To construct our dataset, we used the two datasets in the references. We recommend that the user starts from the 01_preprocessing.ipynb file, but a full reproduction of our analysis can construct the cases.json base data file using the python script collect_data.py, if the two datasets are downloaded and placed in the data directory as described in the README instructions. This python script matches the cases between the two datasets and produces a single cases.json file with all the necessary information for our preprocessing. The datasets are sourced at the end of the report.

Preprocessing

For data preprocessing, see the 01_preprocessing.ipynb file. This notebook loads the cases.json already in the data directory, and condenses the data into a Pandas dataframe with the decisions converted from true/false into 1 or 0. For cases when the first party wins, the gold label is 1, otherwise the gold label is 0. The text is then cleaned of special characters like newlines, tabs, and extra spaces. Two Pandas dataframes are created with the full set of data in each, our raw and clean datasets. The clean dataset has a series of words that could be related to decision leakage and court rulings removed. The two datasets are then saved as csv files into the data

directory as cases_raw.csv and cases_clean.csv. After this, the notebook plots a series of diagnostics, including label counts, length of text, cases per year, presence of decision leakage words, and most frequent tokens.

For the 04 notebook (clean and balanced) notebook , the clean dataset is loaded from the cases_clean.csv. Data is split into training and testing as described previously. The training dataset then has a random selection of cases with the first party winning (the more dominant label) removed from it until the two labels are equal in number of cases. This new reduced dataset is then passed into the model and has error analysis information generated as described previously.

Model Architecture

For the neural network, all of the 02-04 notebooks implement the same model architecture and error analysis, so those will be summarized in the following paragraphs. Each notebook begins by loading the chosen preprocessed dataset, which contains the case text and binary labels whether the first party won. The text is converted into numerical format using tokenization pipeline: we lowercase all text, extract word tokens using a regex pattern ([a-zA-Z]+), and filter out english stopwords and any words that appear only once across all documents. The remaining vocabulary is capped at 45,000 unique tokens with special tokens <PAD> (index 0) and <UNK> (index 1) reserved for padding and unknown words. Each case is represented as a sequence of token indices or either padded with zeroes or truncated with a fixed length of 45,000 tokens. To manage this data efficiently, we define a custom PyTorch dataset class, SCOTUSDataset, which wraps the sequences and labels into tensors. These tensors are fed through a DataLoader with a batch size of 8; we chose this to balance memory constraints with stable training performance given the long sequence length, with shuffling enabled for training and a fixed random seed (42) for reproducibility.

The model itself is a simple feedforward neural network. We chose a feedforward architecture for two reasons: first, it's more interpretable than RNNs or transformers, which matters for legal applications where we want to understand what impacts the predictions; second, the feedforward model can handle full length of oral arguments through mean pooling without truncating the text. Our architecture consists of an embedding layer that maps each token to a 100-dimensional vector, followed by mean pooling across the sequence to produce a single fixed-size representation of the entire transcript. This mean pooled vector is passed through a hidden layer with 128 units and ReLU Activation. We picked 128 units as a balance between having enough capacity to learn complex patterns in legal language while avoiding overfitting on our relatively small dataset of 2,679 cases. Finally, a sigmoid output layer squashes the hidden layer output to a value between 0 and 1, which we interpret as the probability of first party winning. We train using the 80/20 stratified train-test split, meaning 80% of the data (2,143 cases) goes to training and 20% (536 cases) to testing, with the split maintaining the same proportion of winning/losing cases in both sets to avoid biased evaluation. We optimized using binary cross-entropy loss (nn.BCELoss), which penalizes the model more heavily when it's

confidently wrong. We use the Adam optimizer with an initial learning rate of 0.001. We went with Adam optimizer because it adapts the learning rate for each parameter during training, and we believe that the learning rate of 0.001 is fairly standard and prevents the model from making aggressive updates. We trained the model with 10 EPOCHS because the loss curve flattened around that point and training with more EPOCHS would not really improve the accuracy.

Diagnostic Outputs

All notebooks generate the same diagnostic outputs. We plot training loss curves to monitor convergence, construct confusion matrices to visualize classification errors, and generate prediction confidence histograms. For error analysis, we identify misclassified cases as either false positives or false negatives, then examine patterns in text length, rare word usage (words appearing fewer than 200 times), and the presence of specific word categories: negation words like "not" and "no," procedural terms like "jurisdiction" and "appeal," and statutory references like "section" and "statute." We also compute summary statistics comparing correctly classified cases to misclassified ones, and identify specific rare words that appear disproportionately in errors. These diagnostics let us compare how the model behaves across the raw, cleaned, and balanced datasets.

Results

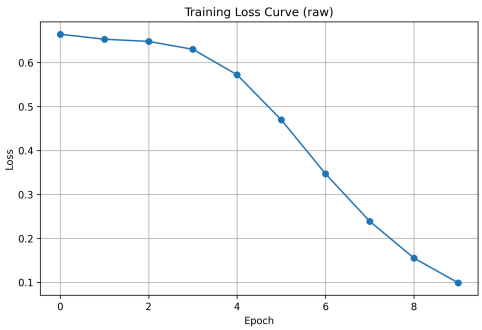
Performance

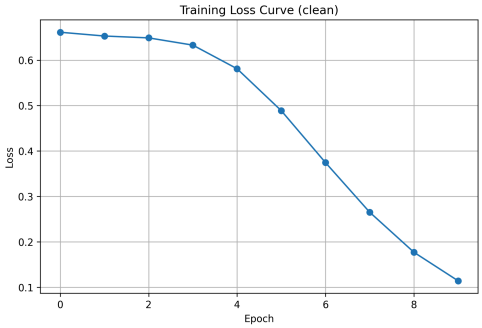
Metric	Raw	Clean and Unbalanced	Clean and Balanced
Accuracy	0.5764925373134330	0.5970149253731340	0.5410447761194030
Precision	0.6455696202531650	0.6575	0.692
Recall	0.7456140350877190	0.7690058479532160	0.5058479532163740
F1 Score	0.6919945725915880	0.7088948787062000	0.5844594594594590

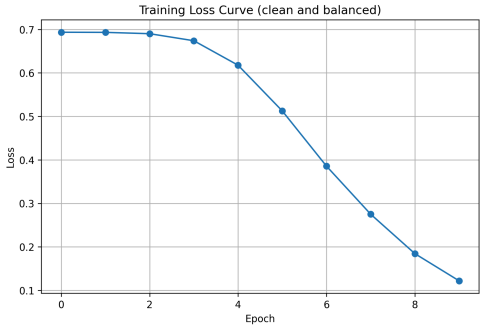
Cleaning the data by removing leakage words augmented performance across all metrics, with accuracy, precision, recall and F1-score all increasing compared to the raw model. Balancing the training set by ensuring the first party winnings and second party winning sets were equal did improve how well it actually predicted a first party winning verdict. This unfortunately came at the cost of the other three performance factors. Most likely, this is because the balanced training set ended up being much smaller and thus could not perform as well overall.

Training Loss Curve

After each of the 10 epochs, we graphed the amount of errors being made through a training loss curve for each data set.



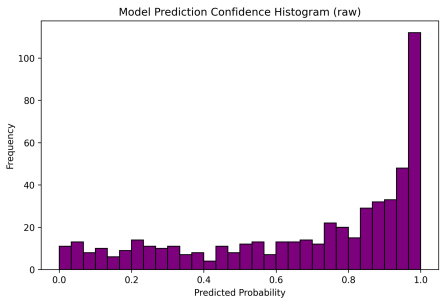


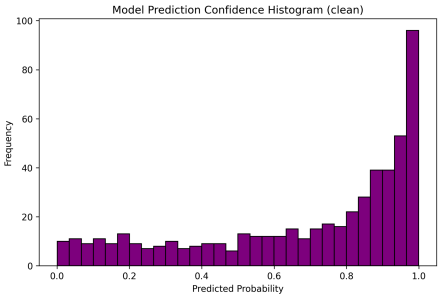


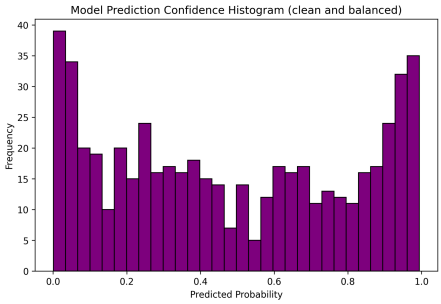
All three graphs showed less and less errors being made as more rounds happened. The clean and balanced one seems to have just a slightly more difficult training curve than the other two. This could be due to the fact that it can't rely on that first party winner bias to overfit, or it could perform slightly worse overall because it has less training data to go off of. Still, it's not a very significant difference.

Model Prediction Confidence

We also visualized how confident the model was in its predictions. In our data, 0 is the second party winning, and 1 is the first party winning.



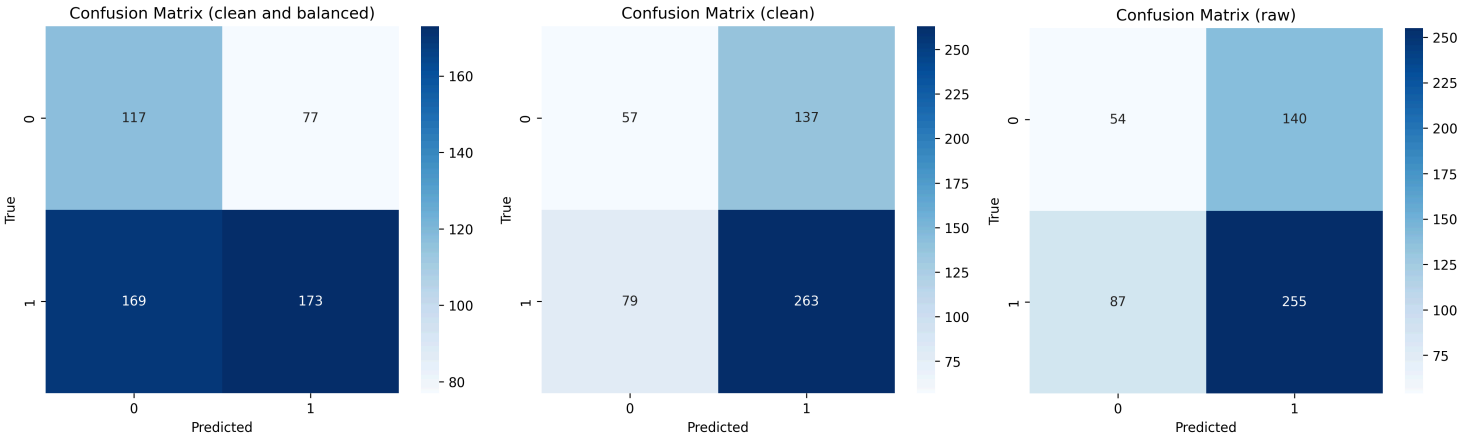




The first two graphs tell us how skewed the data was to train the model on the higher number of first party winning cases. When trained on this unbalanced data containing more first party winning verdicts, the model learned that this was always the right way to go and was therefore always confident when it gave the label of 1. When the training data was balanced though, the model became less biased in its confidence. There's more uncertainty if the label it gives is truly the correct one or not.

Error Analysis

Confusion Matrix Observations



Confusion matrices for each implementation (raw, cleaned, & cleaned and balanced) were constructed.

For the clean and raw implementations, the majority of errors were false positives, where the model predicted a verdict of 1 for cases that were actually not 1. For the clean and balanced model, it was the opposite meaning they had more false negatives.

This is most likely due to the fact that the two unbalanced versions leaned heavily towards predicting 1 because there's more cases with that verdict for training. Once the training was balanced, the model could no longer fall back on this dominant class and became more cautious about predicting 1.

Word Category Patterns

Since the models make decisions based on the text, we examined certain characteristics of it to see if there were any patterns. We created three categories with a few key words in each one. The negation category contained more negative words such as "no", the procedural category contained more words related to how cases are handled and included words like "appeal". Finally, the statute count contained words linked with actual specific laws, such as "subsection".

Clean and Balanced

Feature	**Avg Incorrect**	**Avg Correct**	**Avg False Positives**	**Avg False Negatives**
Negation words	9.60569	8.81724	9.46753	9.66863
Procedural words	39.32926	46.04137	44.41558	37.01183

Statute/citation words	118.30894	112.70344	107.76623	123.11242

Clean and Unbalanced

Feature	Avg Incorrect	Avg Correct	Avg False Positives	Avg False Negatives
Negation words	9.43055	9.00937	8.86131	10.41772
Procedural words	37.62037	46.56562	40.42335	32.75949
Statute/citation words	121.36574	111.16562	116.14598	130.41772

Raw

Feature	Avg Incorrect	Avg Correct	Avg False Positives	Avg False Negatives
Negation words	9.66079	8.94498	8.90714	10.87356
Procedural words	42.82378	48.21359	44.48571	40.14942
Statute/citation words	120.51541	112.25566	115.93571	127.88505

Across models, false negative cases tended to have slightly more negation, procedural and statute words than the false positive cases. These features may be introducing more complexity or ambiguity that leads the model to be overcaution and label cases as 0 incorrectly.

The gap between false positives and false negative amounts of rare words is much higher in the unbalanced models. The unbalanced models were trained more heavily on 1 case and typically leaned towards that unless something broke the pattern. So whenever a case had more rare words, the model would flip to not 1 as a way to break the pattern.

Text Length Affect

The length of the text was also considered. False negatives for the balanced results had slightly longer texts than false positive cases, once again suggesting that the more complex, the more overcaution the model becomes.

Rare Word Behavior

Finally, a comparison between rare words was done. Since the texts are extremely long, a rare word was labeled as such if it occurred less than 200 times. For the balanced model, false negatives had more rare words on average than the correct cases or the false negatives.

Clean and Balanced

Word	Error Count	Correct Count	False Positive	False Negative	Total Cases	Error Ratio
popular	14	2	2	12	16	0.875
particulars	12	2	3	9	14	0.85714
ninety	12	2	3	9	14	0.85714
taft	11	2	4	7	13	0.84615
lodged	15	3	3	12	18	0.83333

Clean and unbalanced

Word	Error Count	Correct Count	False Positive	False Negative	Total Cases	Error Ratio
integration	9	2	8	1	11	0.81818
maturity	9	2	6	3	11	0.81818
eggs	8	2	6	2	10	0.8
converting	8	2	4	4	10	0.8
discrepancy	8	2	6	2	10	0.8

Raw

Word	Error Count	Correct Count	False Positive	False Negative	Total Cases	Error Ratio
ambit	11	2	7	4	13	0.84615
fraudulently	10	2	6	4	12	0.83333
surveys	9	2	4	5	11	0.81818
maturity	9	2	6	3	11	0.81818
integration	9	2	8	1	11	0.81818

One thing we noted about the specific rare words found was that they had either more complex or more specific definitions which could contribute to the false negative label. The unbalanced models displayed something different, as the false positives ended up having more rare words. This is probably because of the bias towards the 1 verdict leading it to choose that whenever there's more complex words in the text.

Conclusion

Ultimately, we found that the unbalanced but clean dataset worked best for the data we had available. On a purely analytical level, the raw dataset probably didn't perform as well because many of the leakage words it contained were actually stopwords, such as phrases like "the court affirms". Our implementation of the cleaned dataset meanwhile involved removing cases from the party with more data so that it balanced with the second party. So there ended up being less information for the model to train on, thus leading to a worse performance as well.

In reality though, we'd actually want to aim for the cleaned and balanced dataset. A model that's only learned to guess the first party because it was trained on won't work as well when given real cases because it's already biased. But we'd want to either add more training cases or change our implementation for balancing, such as by duplicating the cases from the minority party.

We believe that in the future, it would be good to continue exploring the best way to set up the dataset so that the model can perform as best it can. We might also consider alternatives to the oral arguments, such as breaking down the case facts itself into different features for the model to train on. While it can be scary to think about an algorithm one day making big decisions like supreme court case rulings, it's only by taking the time to understand how it works behind the scenes that we can best set ourselves up for success.

Dataset Sources

The court transcripts are sourced from a repository by walkerdb that updates each week.

https://github.com/walkerdb/supreme_court_transcripts

The winner information and facts of the cases are sourced from a Kaggle dataset.

https://www.kaggle.com/datasets/deepcontractor/supreme-court-judgment-prediction