Supreme Court Case Outcome Prediction Using Bayesian Networks

Feolu Kolawole

March 11, 2025

Abstract

The Supreme Court is one of the most powerful bodies in the United States government. Each court hearing affects millions of people - whether directly or not. In many cases, knowing which way a court would lean on a particular case could prove valuable. This project explores a Bayesian network approach to predict Supreme Court case dispositions based on observable case attributes, creating a probabilistic framework that models the relationships between case characteristics and judicial decisions.

Bayesian Network Construction

We constructed our network through dependency tests and domain knowledge. We tested whether the variables were independent by examining if $P(A, B) = P(A)P(B)$. For each variable pair, we computed:

$$D(A, B) = |P(A, B) - P(A)P(B)| \tag{1}$$

Where $D(A, B)$ represents the distance from perfect independence (0 would indicate perfect independence). Setting a threshold for $D(A, B)$ values above said threshold were considered dependent, I formed the following network structure:

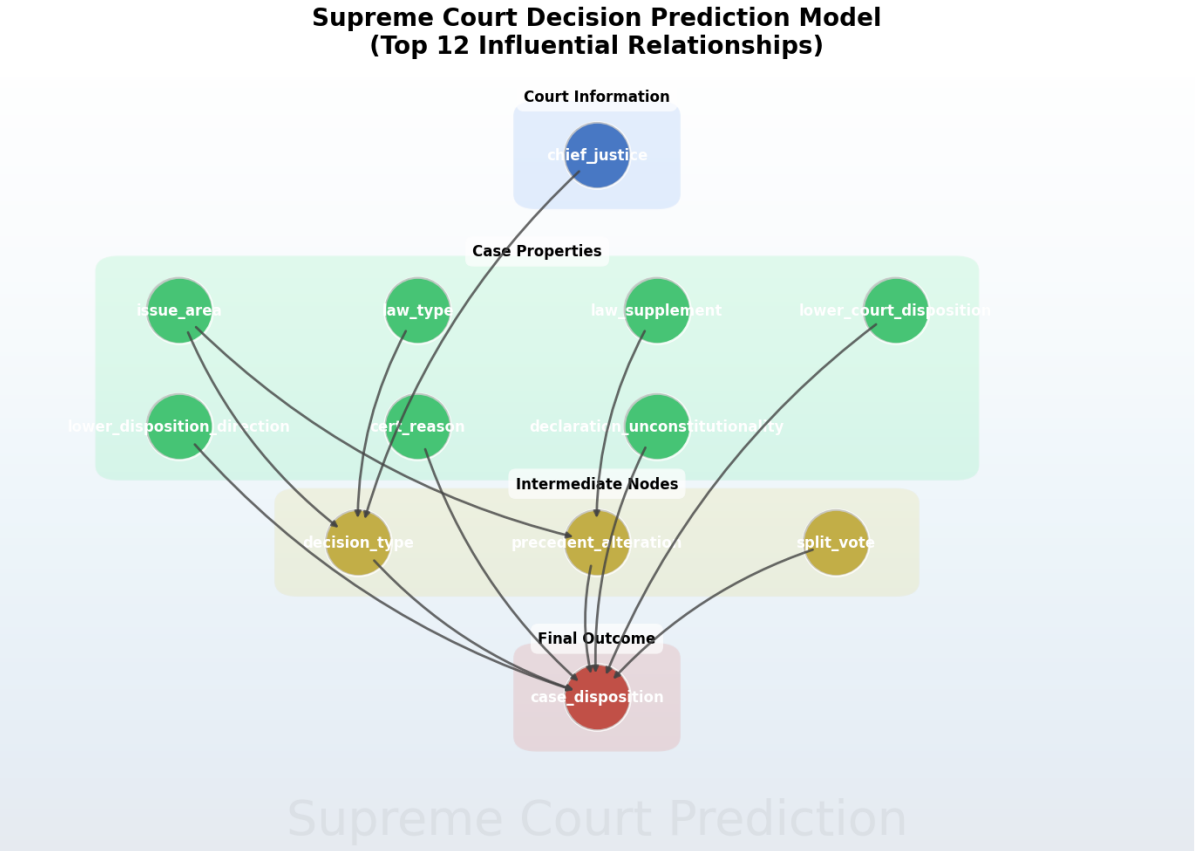


Figure 1: Structure of the Bayesian network showing variables and dependencies.

Conditional Probability Table Construction
After establishing the structure, I built conditional probability tables (CPTs), represented by a large dictionary, by iterating through data on previous historical Supreme Court cases. For each variable, we defined and calculated the following:
For root nodes with no parents:

$$P(X = x) = \frac{\text{Count}(X = x)}{\text{Total cases}} \tag{2}$$

For child nodes with parents:

$$P(X = x|\text{Parents}(X) =?) = \frac{\text{Count}(X = x, \text{Parents}(X) =?)}{\text{Count}(\text{Parents}(X) =?)} \tag{3}$$

This was implemented in Python, creating a nested dictionary structure:

- For root variables: {`value: probability`}
- For child variables: {`(parent1_val, parent2_val, ..., child_val): probability`}

Inference Using Rejection Sampling
We implemented rejection sampling to estimate $P(\text{case\_disposition} = y|\text{observations})$:

Algorithm 1 Rejection Sampling

Input: CPTs, observations, target_val, N
Output: Probability of target_val given observations

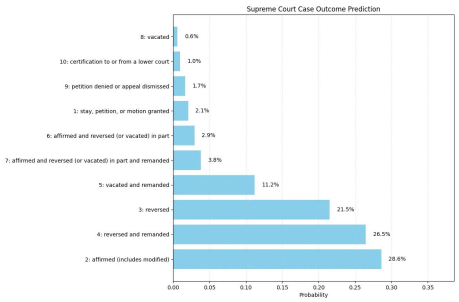
1. target_event ← 0
2. total_accepted ← 0
3. For $i = 1$ to N:
   (a) sample ← GenerateSample(CPTs)
   (b) If Validate(sample, observations):
      i. total_accepted ← total_accepted + 1
      ii. If sample[case_disposition] = target_val:
         A. target_event ← target_event + 1
4. If total_accepted = 0:
   (a) Return 0
5. Else:
   (a) Return target_event / total_accepted

There are a few caveats caused by rejection sampling to keep in mind when utilizing this network, specifically in Step 4...

Main Challenges of Rejection Sampling
Rejection sampling has two important limitations:
1. **Inefficiency with many observations**: Each variable added to the network added inefficiencies because it required generating increasingly large amounts of samples in order to provide enough samples to make a decent prediction.

2. **Zero accepted samples**: If a user is too strict with their observations, it can cause a lack of accepted samples, or worse, no accepted samples. To caveat this I considered only taking a random combination of no more than observation variables that the user inputs, but decided to leave it as is in the case the specific variables were common enough to generate substantial sample amounts.

Before we get into the evaluation,, it is important to note that in the dataset, there are 11 recorded possibilities for each court case disposition. A test run for example, yielded the following predictions shown in the figure.



Evaluation and Results

I evaluated the model by performing rejection sampling several times each several observations and a sample size of 30. Taking percentage of was correct predictions yielded which the follow statistic stood out:

- **Top-3 Accuracy**: Percentage of cases where the actual disposition appears among the three highest probability predictions

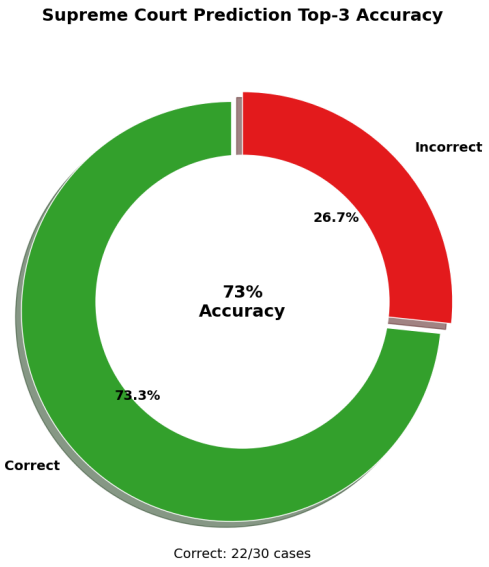


Figure 2: Prediction accuracy showing Top-3 performance.

Given that there are 11 total possibilities for each supreme court disposition, it is clear that the network provides a clear increase in accuracy than one could give on their own intuition.

Implementation Details

The only libraries used for non-visualization purposes were numpy and pandas, in addition, GPT was used to layout the visualizations and clean data.

References

Washington University Law. (2024). *The Supreme Court Database*. Retrieved March 10, 2025, from `http://scdb.wustl.edu/documentation.php`