Anndi Russell
Data Science Capstone
Fall Quarter 2020

Predicting the Vote of a Supreme Court Justice Using Natural Language Processing

Introduction

The purpose of my project is to predict the direction of the vote of a Supreme Court justice based on the transcripts of the oral arguments. The Supreme Court often waits many months after hearing an argument before they release a decision. A model that can accurately predict which direction a justice will vote is useful because it allows for people and organizations whom the ruling will impact to prepare for a most likely outcome.

The program has two major components: text extraction to retrieve the relevant portions of the PDFs of the transcripts, and model creation where data is merged with the relevant data for the outcome of interest and the models are built (one per current justice). The code was split into five separate notebooks; descriptions of the scope of each notebook can be found in the appendix.

Supreme Court Background and Context

Most of the cases heard by the Supreme Court come by way of certiorari, in which the Supreme Court is hearing a case that has already been ruled on by a lower court. These are the types of cases I will be analyzing in this project (there are other types of cases that I do not include in my research, thought those are not as common). In the cases I'm examining, there is a petitioner and a respondent. The petitioner is arguing to overturn the lower court's ruling, whereas the respondent is arguing to uphold it. The case is split into three timed sections: petitioner, respondent, and rebuttal. In the rebuttal, the petitioner gets to answer to what the respondent brought forth. In each section, justices may ask questions to the representing lawyer.

The outcome of a ruling is referred to as the "opinion". The majority opinion is the ruling that will stand, and that majority can rule in favor of or in opposition of the petitioner. The minority group of justices issue what is known as a "dissent", indicating that they disagree with the majority ruling. My research does not focus on whether a justice was in the majority or whether they dissented; rather, it focuses on whether a justice's vote was in support of the petitioner or not. This is equivalent to whether the justice voted to overturn the lower court's ruling or not. My project will analyze the words a justice spoke to the petitioner and to the respondent and predict the direction of their vote.

Source and Description of Data

My program uses two major data sources. The first source of data is a directory of the PDFs of the oral argument transcripts, publicly available on the Supreme Court website. Prior to 2004, the transcripts did not indicate which justice was speaking, thus for my program I am only using transcripts from 2004 and later. This does eliminate some data from the two justices that started prior to 2004 (Justices Thomas and Breyer); since it is older data that is discarded and there is still plenty of more recent data left, I do not anticipate that this would have a major impact on my model performance. From these PDFs, my program extracts and records all the words spoken but a

particular justice. Two different sets of text are recorded: one during the petitioner phase (including rebuttal) and one during the respondent phase.

My second data source was from a database maintained by Washington University Law. Their up-to-date files include a wide variety of information about each case. While there are many analyses that could be done with this data, I am interested in only a small subset of features of this data, discussed later.

Program Implementation

Text extraction

In the first portion of my program (notebook 1_create_dataframe), I read in text from all of the PDF transcripts and created a DataFrame for each justice that recorded the case number, the names of the petitioner and respondent, and the words spoken by a justice (split by petitioner and respondent sections of the transcript). This was a labor-intensive process that involved looking for common patterns across the transcripts, slicing at appropriate locations, removing page headers and footers, etc. Here I will outline the general approach I took for one justice along with some snapshots of the code and PDFs; the same process was used for all justices. For specific descriptions of all steps see the notebook.

To get Justice Breyer's spoken text, I first pulled a list of all case numbers heard by him (from the Washington University Law data). As I iterated through the files in my directory of PDFs, I extracted the case number from the name of the file. If Breyer heard the case, I moved forward with the text extraction, beginning with pulling all the text out into string form using the pdfplumber library. I removed header and footer text that was on each page and not part of the transcript, as well as removing all numbers (including page and line numbers). I then looked for the pattern "ON BEHALF OF THE PETITIONER" and the same for the respondent. The petitioner pattern will show up twice in most cases, as that denotes the rebuttal portion as well. This is a snapshot of this pattern in a PDF:

```
 7              ORAL ARGUMENT OF ALLISON M. ZIEVE
 8                 ON BEHALF OF THE PETITIONER
 9               MS. ZIEVE:  Mr. Chief Justice, and may it
10   please the Court:
```

Figure 1: Sample of the PDF

Using this pattern, I was able to split the PDF into petitioner, rebuttal, and respondent portions, as well as record the name of the petitioning and responding lawyer using pattern recognition on this section of the PDF. Each case was ended with the text "Whereupon, at [time], the case was submitted". Using all of these pieces, I now had three sections: petitioner, responded, and rebuttal. If these splits could not be achieved (if the case did not have a petitioner and respondent because of the case type), the case was skipped. Around 4/5 of cases were included for each justice.

The next step was to find all of the times Breyer's name appeared in the form "JUSTICE BREYER", as this indicated he was speaking. After finding the index of this pattern, I looked for the next occurrence of someone's name (another justice or one of the lawyers present) in the same all-capitals form; this indicated someone else was speaking and Breyer was done. Using this technique,

I was able to find all the text spoken by Breyer in each section. His name was removed from the string, and the text was recorded in the DataFrame. Petitioner and rebuttal section texts were combined; both involved the justice speaking to the same party. The final DataFrame looked like this:

	case_num	pet_name	res_name	pet_text	res_text
0	15-513	SULLIVAN	SINGH	but the government starts again the judgm...	so well lets get to that okay because do we...
1	15-1191	KNEEDLER	BROOME	the problem is with the exception thats been...	im sorry if we leveled up how would that aff...

Figure 2: Final DataFrame recording text spoken by a certain justice

This same type of DataFrame was created for each of the eight justices and stored in a .pkl file to be imported to the next notebook. These DataFrames each took between 30 minutes and 2+ hours to create, depending on the length of term served by the justice.

Data merging

In the next notebook (notebook 2_merge_data), I began by importing the .pkl DataFrame for each justice. For clarity, I will talk through the steps I took here for Justice Breyer; this was performed for all eight justices. After importing Breyer's text DataFrame, I imported the .csv file downloaded from the Washington University Law database. From this file, I kept the columns recording the case number, the date of the argument, which party won (petitioner or respondent), the name of the justice, and whether that justice voted in the majority or the dissent group. From this data I created my outcome of interest, final_vote, that indicated how the justice voted (1 for petitioner and 0 for respondent). I filtered this DataFrame to only rows where justiceName was Breyer and then merged with the text DataFrame on the case number feature in both DataFrames. Rows for which there was no final vote recorded were dropped. There were a very small number of missing votes overall (only 2 for most justices). This DataFrame for each justice was saved to .pkl for use in the next notebook. This final saved DataFrame looked like this for each justice:

	case_num	pet_name	res_name	pet_text	res_text	dateArgument	partyWinning	justiceName	majority	final_vote
0	15-513	SULLIVAN	SINGH	well thats not this question i had the same ...	do you have just on the top of your head som...	11/1/2016	0.0	SGBreyer	2.0	0.0
1	15-1191	KNEEDLER	BROOME	theres a lot of complicated things but the q...	on this but i would say at some point the p...	11/9/2016	1.0	SGBreyer	2.0	1.0
2	15-423	STETSON	CARROLL	so how does it work with diversity we have a...	i dont know that thats a different issue i ...	11/2/2016	1.0	SGBreyer	2.0	1.0

Figure 3: The final merged DataFrame

Data Exploration and Visualization

Now that I have the spoken text combined with some other interesting variables, I did a few visualizations to further motivate and contextualize my research (notebook 3_data_exploration_and_visualization). There are many approaches a person could take to predicting the vote of a justice. For example, each justice is generally agreed upon to have an ideological leaning (conservative or liberal), usually aligning with the president who appointed him or her. This figure from Axios demonstrates one take on their ideological leanings (farther left is more liberal, farther right is more conservative):

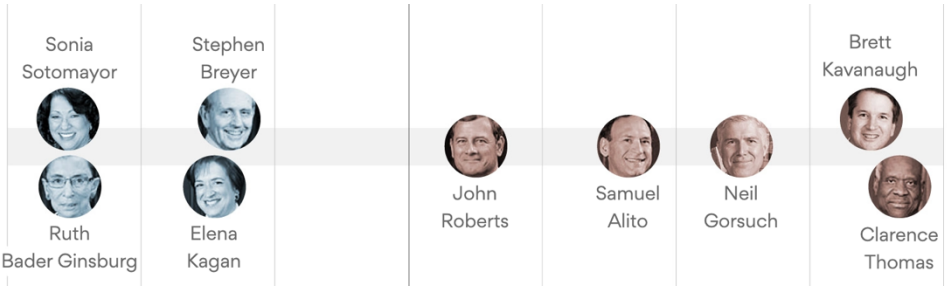


Figure 4: Ideological leanings of Supreme Court Justices as reported by Axios (citation in bibliography)

I found that justices vote 'across party lines' more often than we might initially think. The Washington University Law database recorded whether a vote of a justice on a case leaned conservative or liberal. I plotted the frequency of a justice's voting direction. Here, I show three justices, where blue indicates a liberal vote and red indicates conservative:

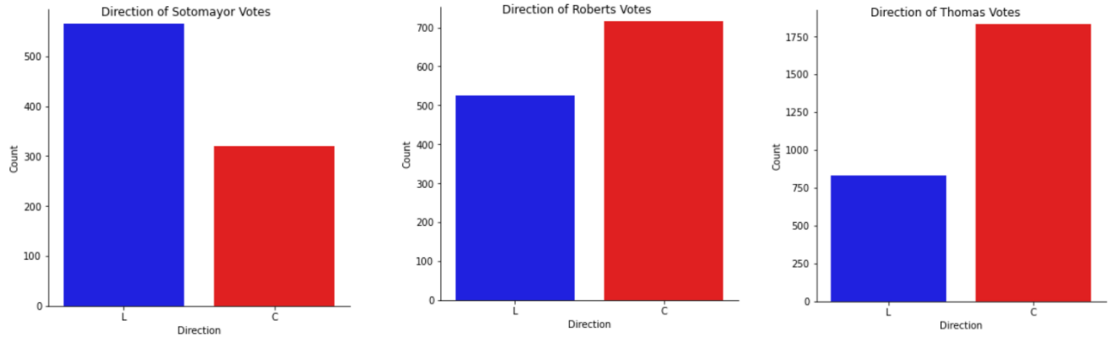


Figure 5: Barplot showing the count of direction of votes for Justices Sotomayor, Roberts, and Thomas

This shows that while the majority of the time a justice votes in the direction of their ideological leaning, they do vote the other direction with a surprising frequency. Ideological leaning alone is not enough to predict how a justice would vote. To further explore this, I used the information recorded in the database that coded each case as one of fourteen "issue areas". I took the four most common of these areas for exploration: Criminal Procedure, Civil Rights, Economic Activity, and Judicial Power. I wanted to see if a justice would obviously vote conservative or liberal on a specific issue. Here is what I found for Justices Thomas and Kagan:

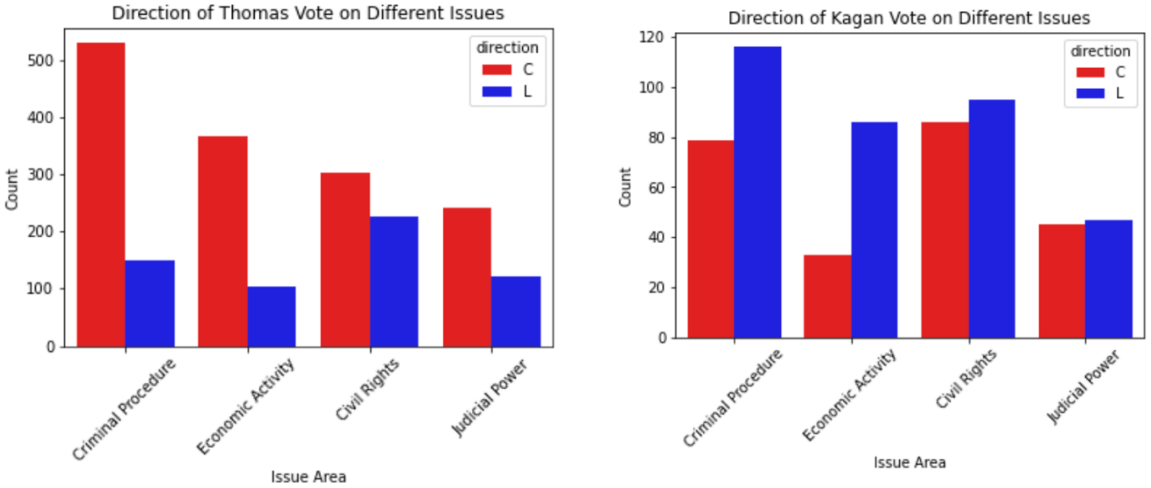


Figure 6: The direction of votes (conservative or liberal) for Justices Thomas and Kagan on four issue areas

Through these plots, we can see that Thomas is more likely to vote conservative on issues relating to criminal procedure, and he has a more balanced voting history for Civil Rights rulings. Kagan generally votes liberal in Economic Activity cases, and her history of voting on Judicial Power rulings is closer to balanced. This illustrates that there are additional avenues to explore for how we can predict the ruling of a justice. It also illustrates that there likely are no sure-fire ways to predict a vote, so building a model based on the text of the oral argument could be a valuable contribution to the body of research on this subject. Additional plots of this nature can be seen in the notebook. The final figure I plotted was the average number of words a justice speaks in a case.

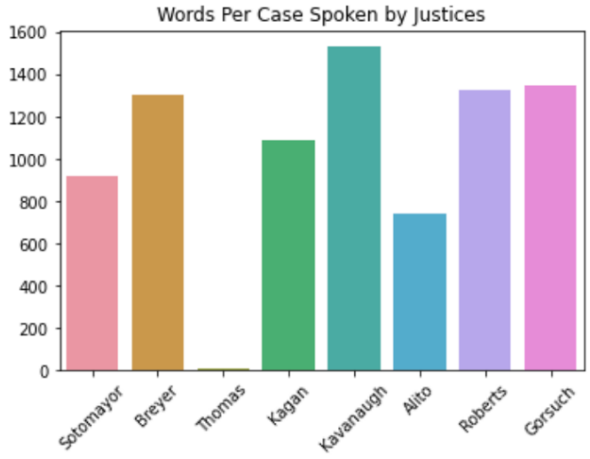


Figure 7: Word count per case spoken by each justice

Kavanaugh tends to speak the most, and Thomas rarely speaks at all (this is commonly known about him; he has only spoken in a small handful of cases in recent years). This is helpful context as Thomas's model is likely to perform poorly with very little data to use.

Building the Models and Optimizing Hyperparameters

For a sentiment analysis problem, Stochastic Gradient Descent algorithm with a logistic loss function is a good choice. This algorithm is successful at large scales and with sparse data, which is what we often encounter with text classification. As the first step in building my models (one per justice), I wanted to see how an SGD algorithm stacked up against three other classification algorithms: standard Logistic Regression, K-Nearest Neighbors, and Support Vector Machines.

To start with building my models (notebook 4_build_model), I imported each of the merged DataFrames for their stored .pkl files. I created a function, all_models, that takes in a DataFrame for a justice, splits into X (the text) and y (the vote outcome) then into training and testing sets (note: for these models, I only used the petitioner text for uniformity of sentiment), transforms the text with a TfidfVectorizer (HashingVectorizer is another option for vectorizing text, but I chose the tfidf vectorizer because it penalizes words that appear frequently across all the documents; this puts a greater emphasis on words that are more unique and more likely to be predictive of a particular sentiment), and build the four models mentioned above. I printed off the accuracy score on the training set as well as the test set, along with the confusion matrix for the test set and the actual proportion of 1s from the test set.

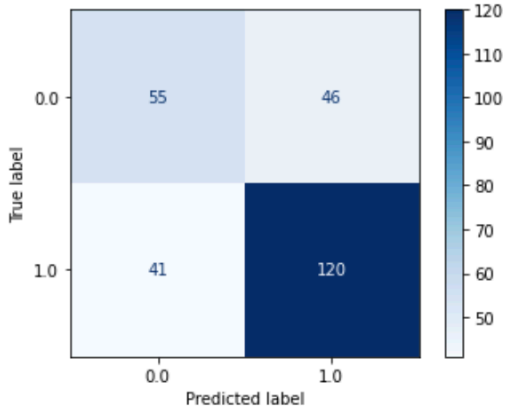
Stochastic Gradient Descent performed with the highest accuracy for all models. This is a snapshot of the code used for the SGD model (for the full code see the notebook):

```
#Stochasic gradient descent
print("SGD")
clf = SGDClassifier(loss='log', random_state=1, penalty='l1', alpha=0.0001)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
print(confusion_matrix(y_pred,y_test))
plot_confusion_matrix(clf, X_test, y_test,cmap=plt.cm.Blues)
plt.show()
print(f'Train score: {clf.score(X_train, y_train)}')
print(f'Test score: {clf.score(X_test, y_test)}')
print()
```

Figure 8: Stochastic gradient descent model construction

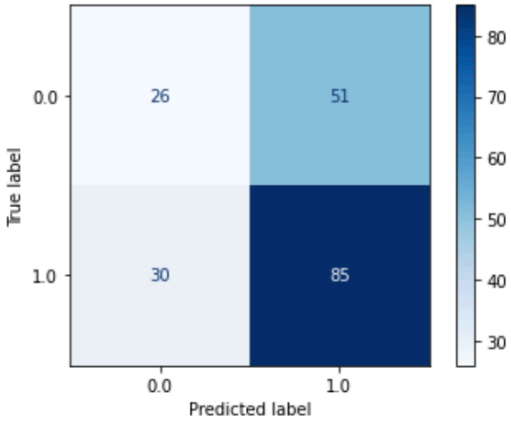
Here are the SGD results for Justices Breyer and Sotomayor (to see all, refer to notebook):

```
SGD
[[ 55  41]
 [ 46 120]]
```




```
Train score: 0.9679245283018868
Test score: 0.6679389312977099
```

```
SGD
[[26 30]
 [51 85]]
```




```
Train score: 0.9922680412371134
Test score: 0.578125
```

Figure 9: Breyer on the left, Sotomayor on the right. Training and testing accuracies and the confusion matrix for the test data for the SGD model.

For Breyer, Logistic Regression and K-Nearest Neighbors had accuracy scores of 0.64 and 0.61, respectively. Support Vector Machines only predicted 1 for everything, and thus had the same accuracy as the proportion of 1s in the overall data (0.61). Stochastic Gradient Descent had by far the highest training accuracy, which suggests that it was learning more than Logistic Regression and K-Nearest Neighbors, though the low test accuracy indicates it isn't generalizing well to new data. Similar observations were made for each justice model.

To try and optimize the performance of SGD, I ran a grid search on a set of hyperparameters for both the vectorizer and the SGD model. These are the hyperparameters I tuned, where "vect" is the TfidfVectorizer and "clf" is the SGD classifier:

```
param_grid = [{'vect__ngram_range': [(1, 1), (1, 2)],# unigrams or bigrams
               'vect__stop_words': [stop, None], #use stop words or not
               'vect__tokenizer': [tokenizer, tokenizerporter], #use porter stemmer or not
               'vect__use_idf':[True,False], #use inverse document frequency or not
               'clf__penalty': ['l1', 'l2'], #l1 or l2 penalty
               'clf__n_iter_no_change': [3,5,7,9], #stopping criterion
               'clf__alpha': [1e-4, 1e-3, 1e-2, 1e-1, 1e0, 1e1, 1e2, 1e3]}, #regularization term
              ]
```

Figure 10: Grid search cross-validation code for SGD logistic model

The code from my grid search printed out the best parameters selected through the cross-validation process. I recorded the best set of parameters of each justice's model, and then began in my final notebook, notebook 5_models_with_tuned_params. This is where I created my final model with the tuned hyperparameters and reported the final accuracies of my models.

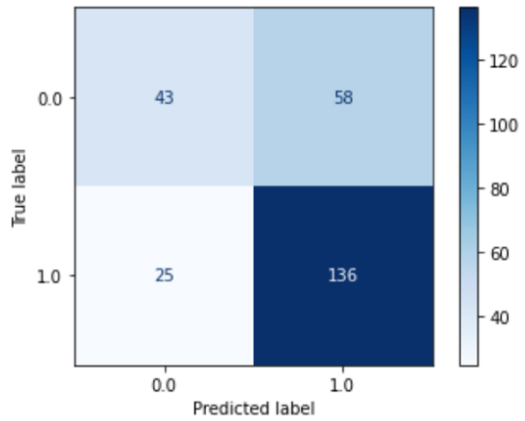
Fitting and Testing Final Models

In the final notebook (5), I used the optimized hyperparameters to fit a model on each of the petitioner text and respondent text for all justices, resulting in 16 total models (though I found that model performance was not better with one set of text over the other, so my final 8 models only use the petitioner text). This is the code for fitting the final model with the petitioner text:

```
breyer_df = pd.read_pickle("df_merged_breyer.pkl")
breyer_df=breyer_df.reset_index()

X=np.array(breyer_df.pet_text)
y=np.array(breyer_df.final_vote)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,stratify=y,random_state=42)

vect = TfidfVectorizer(tokenizer=tokenizer, use_idf=False, ngram_range=(1,2), stop_words=None)
X_train = vect.fit_transform(X_train)
X_test = vect.transform(X_test)

print("Ratio")
print(np.bincount(y_train.astype(int))[1]/len(y_train))
print()

print("SGD")
clf = SGDClassifier(loss='log', random_state=1, penalty='12', alpha=0.0001, n_iter_no_change=7)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
print(confusion_matrix(y_pred,y_test))
plot_confusion_matrix(clf, X_test, y_test,cmap=plt.cm.Blues)
plt.show()

print(f'Train score: {clf.score(X_train, y_train)}')
print(f'Test score: {clf.score(X_test, y_test)}')
```

Figure 11: Code for fitting the final model with tuned hyperparameters

I printed out the resulting train and test accuracies, confusion matrix, and precision and recall scores. Here, I display the final results for Justice Breyer as a sample:




```
Train score: 0.9679245283018868
Test score: 0.683206106870229
Precision: 0.7010309278350515
Recall: 0.84472049689441
```

Figure 12: Final model performance for Justice Breyer

As you can see the test accuracy was 0.68, which is quite low. The model is doing some work, as it is not just predicting for the majority class every time (verified through the confusion matrix and the precision and recall scores). This was not true for every justice though; for one of the justice models created (Thomas's respondent text model), tuning the hyperparameters actually ended up having a negative effect on the models, as it "optimized" to just predicting the majority class (since that accuracy was the highest it could achieve). This is a good demonstration of finding the balance of accuracy, precision, and recall, and of how a grid search is only optimizing for accuracy which isn't always going to give a better model overall. Here I display the final accuracies for the model for each justice (to see detailed evaluations of each model see the notebook):

Thomas:	0.585
Breyer:	0.683
Sotomayor:	0.562
Kagan:	0.633
Alito:	0.680
Gorsuch:	0.627
Roberts:	0.691
Kavanaugh:	0.613

Conclusion:

The final result of my project is a model for each justice, predicting whether a justice will vote in favor of the petitioner or the respondent based on the words they said during the petitioner portions of the trial. I found during my research some interesting facts about how justice vote and how they speak; certain justices tend to speak more in a case, while other barely speak at all. Justices considered more ideologically moderate tend to vote conservative or liberal with more similar frequencies, although all justices vote both directions more than I initially hypothesized that they would.

My final models have low performance. The highest accuracy I achieved was with Justice Roberts at 0.691, Justice Breyer at 0.683, and Justice Alito at 0.680; Justice Thomas was the lowest with an accuracy of just 0.585 (he rarely speaks, so there was not much data to use).

Overall, the models should not be considered good predictors of a vote on their own, but they might be able to contribute to research that also incorporates other factors such as the topic of the case, or the ideological leaning of the justice.

Citations:

- Supreme Court: https://www.supremecourt.gov/oral_arguments/argument_transcript/2020
- Washington University Law Database: http://scdb.wustl.edu/index.php
- Axios figure: https://www.axios.com/supreme-court-justices-ideology-52ed3cad-fcff-4467-a336-8bec2e6e36d4.html

Appendix:

Notebook names and descriptions:

- 1_create_dataframe: This notebook processes all of the PDFs, extracting the text spoken by each justice. The DataFrames storing the text are saved to .pkl.
- 2_merge_data: This notebook merges the text data from notebook 1 with the data from the Washington University Law database to create a DataFrame that includes both the text and the outcome of interest. The DataFrames are saved to .pkl.
- 3_data_exploration_and_visualization: This notebook shows visualizations regarding how often a justice votes a certain way in cases that they hear. This notebook also explores the number of words justices speak on average.
- 4_build_model: In this notebook, initial models are build and grid search cross-validation is run with the SGD model to tune hyperparameters. Optimal hyperparameters were recorded for use in the next notebook.
- 5_models_with_tuned_params: This final notebook builds the final model for each justice using tuned hyperparameters, as well as reporting final accuracy, recall, precision, and a confusion matrix for the performance of each model.