



Department of Computer Science and Engineering (Data Science)

Report on

“Predictive Analytics in Judicial Decision-Making: A Focus on Supreme Court Rulings”

Project Report for Computational Linguistic Course

By

Farin Khan 60009210140

Nidhi Pabari 60009210173

Guided by

Prof. Pooja Vartak

Date

04.05.2024

Table of Content

1. Introduction	1
2. Data Overview	2
3. Data Preprocessing	3
4. Model Overview	7
5. Results	8
6. Conclusion	9

Useful Links

Introduction

In recent years, artificial intelligence has made significant strides across various domains, revolutionising traditional practices and introducing predictive analytics into previously unexplored territories. One such area is the legal system, where the utilisation of predictive analytics has the potential to reshape judicial decision-making processes. Our project, titled "Predictive Analytics in Judicial Decision-Making: A Focus on Supreme Court Rulings," delves into this intersection of law and technology, aiming to leverage natural language processing (NLP) techniques to enhance the understanding and prediction of Supreme Court judgments.

Our primary aim is twofold: first, to develop a robust NLP application showcasing a spectrum of text preprocessing techniques, and second, to harness the power of predictive modelling to discern underlying patterns that influence court decisions. By transforming raw textual data into actionable features using NLP methodologies, we aspire to emulate the nuanced decision-making processes of human juries, thereby paving the way for a more informed and data-driven approach to judicial analysis.

Data Overview

The foundation of our project lies in a meticulously curated dataset sourced from the Supreme Court of the United States (SCOTUS), available on Kaggle. This dataset encapsulates 3304 cases spanning from 1955 to 2021, offering a comprehensive glimpse into the historical evolution of Supreme Court rulings. Each case within the dataset is richly annotated with essential details, including case identifiers, the factual background of the case, and most crucially, the decision outcome.

What sets this dataset apart is its inclusion of the **facts** of each case, a rarity in related datasets. This inclusion is pivotal for our natural language processing (NLP) endeavours as it provides the raw material necessary to extract meaningful insights and patterns. Our target variable, the **first party winner**, serves as a binary indicator, distinguishing whether the first party or the second party emerged victorious in the legal dispute.

Unnamed: 0	ID	name	href	docket	term	first_party	second_party	facts	facts_len	majority_vote	minority_vote	first_party_winner	decision_type	disposition	issue_area
1	1	50613 Stanley v. Illinois	https://api.oyez.org/cases/1971/70-5014	70-5014	1971	Peter Stanley, Sr.	Illinois	<p>Joan Stanley had three children with Peter ...	757	5	2	True	majority opinion	reversed/remanded	Civil Rights
2	2	50623 Giglio v. United States	https://api.oyez.org/cases/1971/70-29	70-29	1971	John Giglio	United States	<p>John Giglio was convicted of passing forged...	495	7	0	True	majority opinion	reversed/remanded	Due Process
3	3	50632 Reed v. Reed	https://api.oyez.org/cases/1971/70-4	70-4	1971	Sally Reed	Cecil Reed	<p>The Idaho Probate Code specified that "male...	378	7	0	True	majority opinion	reversed/remanded	Civil Rights
4	4	50643 Miller v. California	https://api.oyez.org/cases/1971/70-73	70-73	1971	Marvin Miller	California	<p>Miller, after conducting a mass mailing cam...	305	5	4	True	majority opinion	vacated/remanded	First Amendment
5	5	50644 Kleindienst v. Mandel	https://api.oyez.org/cases/1971/71-16	71-16	1971	Richard G. Kleindienst, Attorney General of th...	Ernest E. Mandel, et al.	<p>Ernest E. Mandel was a Belgian professional...	2282	6	3	True	majority opinion	reversed	First Amendment

Fig 1. Snapshot of dataframe head of *justice.csv* loaded in python notebook

The above figure provides a snapshot of the data, showcasing the structure of our dataset. It includes essential columns such as *facts*, *first_party_winner*, *decision_type*, *disposition* and *issue_area*. Our focus lies primarily on the '*facts*' column, where we will apply a range of natural language processing (NLP) techniques to extract meaningful insights. Additionally, the '*first_party_winner*' column serves as our target variable for prediction tasks, a key aspect that will be explored in detail in subsequent sections of this report.

Data Preprocessing

Data preprocessing is a crucial step in any data science project, especially in Natural Language Processing (NLP) tasks like predicting Supreme Court rulings. In this chapter, we will discuss the steps involved in preparing our dataset for training the predictive model.

Our preprocessing steps were divided into six key phases:

Step 1: Data Inspection and Selection

We started by inspecting the data frame to check for data types and null values. Fortunately, there were no null values, and we selected the relevant columns for prediction: 'facts,' 'first_party_winner,' 'decision_type,' and 'disposition'. Irrelevant columns were dropped to focus our analysis.

Step 2: Text Preprocessing

The 'facts' column, containing textual data, underwent extensive preprocessing. We performed the following tasks:

- Initial Cleaning: Converted text to lowercase, removed punctuations and special characters, and stripped unnecessary spaces.
[`^\w\s`] matches any non-alphanumeric and non-whitespace character.
- Tokenization: Split the text into tokens (words).
- Stopword Removal: Eliminated common stopwords to focus on meaningful words.
- Lemmatization: Convert words to their base/root form using lemmatization, enhancing the text's semantic understanding.

These tasks are implemented using a custom preprocessing function, `utils_preprocess_text`, which streamlined the text cleaning process.

Python code:

```
def utils_preprocess_text(text, flg_lemm=True,
lst_stopwords=None):
    ## clean
    text = re.sub(r'^\w\s', '', str(text).lower().strip())

    ## Tokenize
    lst_text = text.split()

    ## remove Stopwords
    if lst_stopwords is not None:
        lst_text = [word for word in lst_text if word not in
lst_stopwords]
```

```

## Lemmatization
if flg_lemm == True:
    lem = nltk.stem.wordnet.WordNetLemmatizer()
    lst_text = [lem.lemmatize(word) for word in lst_text]

## back to string from list
text = " ".join(lst_text)
return text

```

	facts	facts_clean
0	Joan Stanley had three children with Peter Sta...	joan stanley three child peter stanley stanley...
1	John Giglio was convicted of passing forged mo...	john giglio convicted passing forged money ord...
2	The Idaho Probate Code specified that "males m...	idaho probate code specified male must preferr...
3	Miller, after conducting a mass mailing campai...	millier conducting mass mailing campaign advert...
4	Ernest E. Mandel was a Belgian professional jo...	ernest e mandel belgian professional journalis...
...
3093	For over a century after the Alaska Purchase i...	century alaska purchase 1867 federal governmen...
3094	Refugio Palomar-Santiago, a Mexican national, ...	refugio palomarsantiago mexican national grant...
3095	Tarahrick Terry pleaded guilty to one count of...	tarahrick terry pleaded guilty one count posse...
3096	Joshua James Cooley was parked in his pickup t...	joshua james cooley parked pickup truck side r...
3097	The Natural Gas Act (NGA), 15 U.S.C. §§ 717–71...	natural gas act nga 15 usc 717717z permit priv...

3098 rows × 2 columns

Step 3: Data Concatenation

After preprocessing the 'facts' column, we concatenated it with the target variable 'first_party_winner' to create a combined text-feature dataset.

Step 4: One-hot Encoding

One-hot encoding is a technique used to convert categorical variables into a numerical format that machine learning algorithms can understand. It works by creating binary columns for each category in the categorical variable, where each column represents a category and has a value of 0 or 1 indicating the absence or presence of that category in a particular observation.

One-hot encoding is done for 'decision_type' and 'disposition'. They become binary columns for each category (1 if present, 0 otherwise). This encoding helps models interpret categorical data, like identifying winning parties or case dispositions, improving predictive accuracy.

Step 5: Word2Vec Embedding

We applied the Word2Vec technique to the 'facts' column to create word embeddings. Word2Vec captures semantic relationships between words by representing them in a high-dimensional vector space, enhancing our model's understanding of textual data.

Python code:

```
from gensim.models import Word2Vec

data_final = pd.DataFrame()
sentences = [text.split() for text in df_n11['facts_clean']]
word2vec_model = Word2Vec(sentences, vector_size=100, window=5,
min_count=1, sg=1)

def document_vector(word2vec_model, doc):
    doc = [word for word in doc.split() if word in
word2vec_model.wv]
    if len(doc) > 0:
        return np.mean(word2vec_model.wv[doc], axis=0)
    else:
        return np.zeros(word2vec_model.vector_size)

data_final[['word2vec_dim{}'.format(i) for i in
range(word2vec_model.vector_size)]] =
df_n11['facts_clean'].apply(lambda x:
pd.Series(document_vector(word2vec_model, x)))

data_final = pd.concat([data_final, df_cat3], axis=1,
join='inner')
data_final
```

	word2vec_dim0	word2vec_dim1	word2vec_dim2	word2vec_dim3	word2vec_dim4	word2vec_dim5	word2vec_dim6	word2vec_dim7	word2vec_dim8	word2vec_dim9
0	-0.176163	0.168063	0.061386	0.137642	0.254622	-0.559096	-0.071851	0.298727	-0.384770	-0.263107
1	-0.461444	0.161543	0.144218	0.151577	0.212473	-0.681313	-0.017510	0.324238	-0.369767	-0.353069
2	-0.219511	0.138281	0.088050	0.100666	0.149089	-0.495141	-0.042705	0.236552	-0.334964	-0.114066
3	-0.320539	0.098317	0.051376	0.113272	0.305084	-0.478945	-0.077205	0.194728	-0.242807	-0.094521
4	-0.225577	0.112759	0.111801	0.146623	0.169906	-0.523601	-0.047811	0.259689	-0.294468	-0.167838
...
3093	-0.190596	0.118589	0.027530	0.057230	0.291049	-0.525937	0.047945	0.352890	-0.305178	-0.216220
3094	-0.378760	0.120414	0.081209	0.229008	0.302677	-0.576575	-0.074977	0.340789	-0.282412	-0.288515
3095	-0.404034	0.047486	0.042076	0.220939	0.252665	-0.581484	-0.154843	0.314057	-0.287403	-0.234160
3096	-0.286288	0.060765	0.053491	0.045147	0.321404	-0.470375	-0.063698	0.267536	-0.200364	-0.187382
3097	-0.323483	0.187925	0.008353	0.121777	0.266572	-0.666867	0.004755	0.406506	-0.348328	-0.232637

3098 rows × 118 columns

Step 6: Final Dataset Preparation

To create the final dataset for training, we concatenated the target+facts dataset with the remaining relevant columns ('decision_type' and 'disposition').

As part of the analysis, unigrams and bigrams are visualised separately for 'first_party_winner' and 'second_party_winner.' This visualisation provided insights into the most frequent single words and word pairs associated with each party in the Supreme Court rulings.

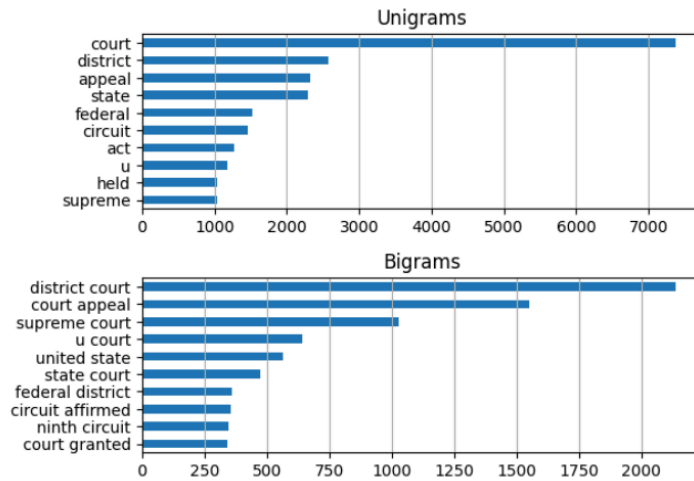


Fig 2. Unigrams and Bigrams of facts from cases with first party winner

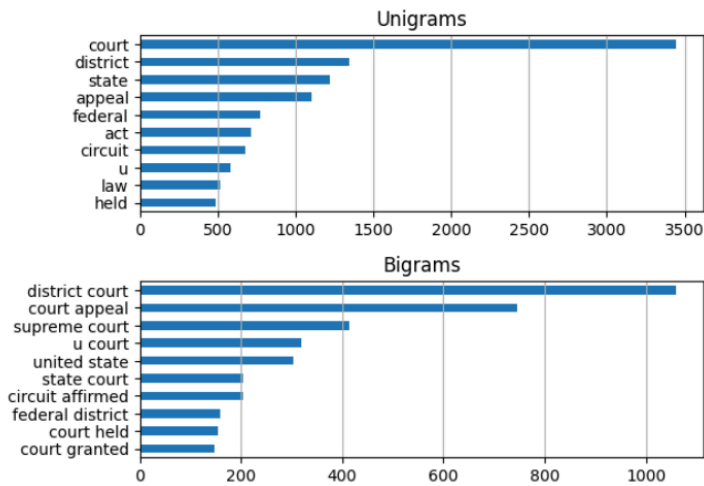


Fig 3. Unigrams and Bigrams of facts from cases with second party winner

The data preprocessing phase is critical for ensuring the quality and relevance of our dataset for training the predictive model. With the text cleaned, tokenized, lemmatized, and transformed into word embeddings, and relevant features selected and encoded, our data is now ready for the next phase: model training and evaluation.

Model Overview

Before training data on any machine learning model, it is split into training and testing sets. The train-test split is a crucial step in evaluating and validating machine learning models. It involves dividing the dataset into training and testing sets to assess the model's performance on unseen data. The data is split into training and testing sets using the 'train_test_split' function from Scikit-Learn.

The hyperparameters used in our code are:

- `data_final.drop(columns=['first_party_winner'])`: Features used for training, excluding the target variable.
- `data_final['first_party_winner']`: Target variable to predict.
- `test_size=0.3`: 30% of the data is reserved for testing.
- `random_state=10`: Ensures reproducibility by fixing the random seed.

Random Forest is a powerful ensemble learning technique that combines multiple decision trees to make predictions. Each tree in the forest contributes to the final prediction, and the ensemble approach improves accuracy and generalisation.

The Random Forest model is trained with the following hyperparameters:

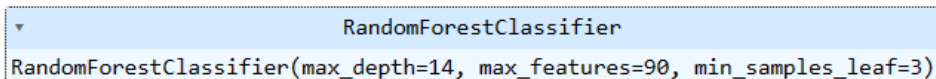
- Max Depth: 14
- Max Features: 90
- Min Samples Leaf: 3
- Number of Estimators: 100

These hyperparameters control the complexity and behaviour of the individual decision trees within the Random Forest.

The Random Forest Classifier is trained using the 'fit' method on the training data.

Python code:

```
from sklearn.ensemble import RandomForestClassifier
rand=RandomForestClassifier(max_depth=14, max_features=90,
min_samples_leaf=3, n_estimators=100)
rand.fit(X_train,y_train)
```



This overview covers the Random Forest model's hyperparameters, training process, and the importance of the train-test split in evaluating model performance. Adjustments to hyperparameters and evaluation metrics can further refine the model's effectiveness in predicting Supreme Court rulings.

Results

The Random Forest model achieved the following accuracy scores:

- Training Accuracy: 97.97%
- Testing Accuracy: 96.88%

The high training accuracy indicates that the model learned well from the training data, capturing patterns and relationships effectively. The slightly lower testing accuracy suggests good generalisation performance, as the model maintains high accuracy on unseen data.

The F1 score is a metric that combines precision and recall, providing a balanced measure of a model's performance, especially in binary classification tasks like predicting winning parties. It is chosen as an evaluation metric due to its suitability for binary classification tasks and its ability to handle class imbalance. In predicting Supreme Court rulings, where outcomes may be skewed towards one party winning, the F1 score provides a comprehensive assessment of the model's predictive capabilities.

The F1 score obtained for the Random Forest model is approximately 97.70%. This score reflects the model's ability to achieve a balance between precision (correctly identifying positive cases) and recall (capturing all positive cases).

Conclusion

The Random Forest model has demonstrated strong performance in predicting Supreme Court ruling outcomes based on textual data. The high accuracy scores on both training and testing data, coupled with a robust F1 score, highlight the model's effectiveness and reliability.

A key factor contributing to these high scores is the utilisation of word embeddings through techniques like Word2Vec. Word embeddings capture semantic relationships between words, enhancing the model's understanding of textual data. This NLP aspect plays a pivotal role in improving the model's predictive power and generalisation capability.

Further analysis, such as examining confusion matrices or exploring feature importance, can provide additional insights into the model's behaviour and potential areas for refinement. The model's results showcase its potential for real-world applications in legal analytics and decision-making processes.

In conclusion, the integration of NLP techniques, particularly word embeddings, has been instrumental in achieving the model's accuracy and F1 score, underscoring its reliability and predictive power in predicting Supreme Court rulings.

Useful Links

Project:  CL_Experiment10

Dataset: [Supreme Court Judgment Prediction \(kaggle.com\)](https://www.kaggle.com/datasets/ashp/uscj-rulings)