Predicting Outcomes of US Supreme Court Oral Arguments

Team: Fei Wang, Reza R Pratama, Shradha Ganapathy, Xiomara Salazar

Description

The Supreme Court of the US (SCOTUS) is the highest court in the United States. It holds the power of judicial review – determining whether a statute violates the Constitution. The court consists of nine Justices (a chief justice and eight associate justices). Justices can serve for life (i.e., until they die, retire, resign, get impeached, etc.), so one Justice can influence decisions for many decades. Clearly, the decisions made by the SCOTUS have great public policy impact in the US.

In general, the cases that the SCOTUS review have been given a decision by a lower court, and are brought to the Court for appeal of said decision. For these cases, the Court will take briefs and conduct an oral argument with the attorneys representing the (usually 2) parties in this case. Following the oral argument, each Justice gets one vote, and the majority vote determines the case outcome. Example SCOTUS case: https://www.oyez.org/cases/2018/17-204

For this project, we are interested in predicting the outcome of the case (i.e., which party gets the majority vote) from the oral argument transcripts. These transcripts (see Dataset) are dialogs of English natural language text. This can be thought of as a text classification task, where the transcripts are the input documents. The labels we want to predict are the outcomes (e.g., Y/N the petitioner “wins”).

Dataset

Supreme Court Oral Arguments Corpus from Convokit. The subset of data used in our project encompasses the Supreme Court's transcripts across eight years of the George W. Bush and two terms (8 years) of the Barack Obama administration (2001 - 2018).

For further explanation on why data was pared down to this subset, please take a look at the section labeled 'Why did we select this subset of data?' in Checkpoint 1

Note that data was downloaded and saved as a csv using this script: download_data.py

Baseline

Our dataset contains 400 records of 1.0 and 201 records of 0.0.

Accuracy: 400/601 = 0.6655574043261231

Our models

In order to test which model would have the highest accuracy over our dataset, we implemented a variety of models and measured accuracy. A summary of these models with the highest accuracies and the hyperparameters/features that yielded these accuracies is given below:

Model	Best Accuracy	Other Information
K-Nearest Neighbor (averaged across 10 random splits)	• 0.6952 (TFIDF) • 0.676 (CountVectorizer)	• min_df = 7; max_df = 0.70; K = 9 • ngram_range = (2, 2), K = 7 • Train-Test split: test_size=0.2 (for TFIDF and CountVectorizer)
Random Forest (average of 50 epochs)	• 0.639776 (Ruling party) • 0.637908 (CountVectorizer) • 0.633053 (TFIDF) • 0.632680 (Repblic_judge_pc + Ruling party) • 0.631559 (Repblic_judge_pc)	• min_df = 5; max_df = 0.70 • Train test split: test_size=0.2
Logistic Regression	• 0.66412 (Bag of words) • 0.66412 (CountVectorizer)	• Used default - min_df = 1; max_df = 1 • Train test split: test_size=0.25
Multinomial NB	• 0.723810 (TFIDF) • 0.571429 (CountVectorizer) • 0.542857 (TFIDF with Oversampling) • 0.561905 (TFIDF with Undersampling)	• min_df = 5; max_df = 0.70 • Train test split: test_size=0.2
Linear SVC	• 0.6285 (TFIDF) • 0.571429 (CountVectorizer) • 0.542857 (TFIDF with Oversampling) • 0.561905 (TFIDF with Undersampling)	• min_df = 5; max_df = 0.70 • Train test split: test_size=0.2
Perceptron	• 0.628571 (TFIDF) • 0.571429 (CountVectorizer)	• min_df = 5; max_df = 0.70 • Train test split: test_size=0.2

Model	Best Accuracy	Other Information
	• 0.542857 (TFIDF with Oversampling) • 0.561905 (TFIDF with Undersampling)	

We also tried using pretrained model from transformers (https://huggingface.co/models?pipeline_tag=text-classification&sort=downloads), however we were unable to reach the baseline.

Conclusion / Next Steps

To improve accuracy we may need to refine our pre-processing / cleaning of the text as our models seem to be performing - on average - only marginally above the baseline.

checkpoint2_knn

May 8, 2023

[ ]:
```
import pandas as pd
from ast import literal_eval
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

[ ]:
```
import nltk
import re
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('wordnet')
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from nltk.stem import PorterStemmer
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data…
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data…
[nltk_data]   Package wordnet is already up-to-date!
```

[ ]:
```
path = '/content/drive/MyDrive/Supreme Project/dataset'
```

[ ]:
```
# load downloaded data
df_convos = pd.read_csv(path+'/conversations.csv')
df_speakers = pd.read_csv(path+'/speakers.csv')
df_utts = pd.read_csv(path+'/utterances.csv')
df_cases = pd.read_json(path_or_buf='https://zissou.infosci.cornell.edu/
 ↪convokit/datasets/supreme-corpus/cases.jsonl', lines=True)
df_cases = df_cases[(df_cases['year'] >= 2011) & (df_cases['year'] <= 2018) &␣
 ↪(df_cases['win_side'].isin([0,1]))]
```

[ ]:
```
# combine text from all utterances in a conversation back into one string based␣
 ↪on the conversation_id, coount how many utterances per conversation
```

```
utt_per_conv = df_utts.groupby('conversation_id')['text'].apply(lambda x: ' '.
 ↪join(x)).reset_index()
utt_per_conv['num_utterances'] = df_utts.groupby('conversation_id')['text'].
 ↪count().reset_index()['text']

# add the combined text to the conversations dataframe, merge on␣
 ↪conversation_id in utt_per_conv and id in df_convo
df_convos_utt = df_convos.merge(utt_per_conv, left_on='id',␣
 ↪right_on='conversation_id', how='left')
```

[ ]:
```
# combine text from all conversation in a cases into one string based on the␣
 ↪meta.case_id
conv_per_case = df_convos_utt.groupby('meta.case_id')['text'].apply(lambda x: '␣
 ↪'.join(x)).reset_index()
conv_per_case['num_conversations'] = df_convos_utt.groupby('meta.
 ↪case_id')['text'].count().reset_index()['text']
conv_per_case['num_utterances'] = df_convos_utt.groupby('meta.
 ↪case_id')['num_utterances'].sum().reset_index()['num_utterances']

# add the combined text case dataframe, merge on meta.case_id and id
df_cases_convo = df_cases.merge(conv_per_case, left_on='id', right_on='meta.
 ↪case_id', how='left')
df_cases_convo.head(3)
```

[ ]:
```
             id  year  citation  \
0  2011_11-1179  2011  567 US _
1   2011_11-182  2011  567 US _
2   2011_11-161  2011  566 US _

                                          title  \
0  American Tradition Partnership, Inc. v. Bullock
1                      Arizona v. United States
2                Armour v. City of Indianapolis

                            petitioner  \
0  American Tradition Partnership, Inc.
1                      Arizona et al.
2                   Christine Armour

                                       respondent docket_no          court  \
0  Steve Bullock, Attorney General of Montana, et…   11-1179  Roberts Court
1                                    United States    11-182  Roberts Court
2                              City of Indianapolis    11-161  Roberts Court

   decided_date                                        url  … win_side_detail  \
0  Jun 25, 2012  https://www.oyez.org/cases/2011/11-1179  …              3.0
```

```
1  Jun 25, 2012   https://www.oyez.org/cases/2011/11-182  …             7.0
2   Jun 4, 2012   https://www.oyez.org/cases/2011/11-161  …             2.0

   scdb_docket_id                                              votes  \
0     2011-073-01  {'j__john_g_roberts_jr': 2.0, 'j__antonin_scal…
1     2011-075-01  {'j__john_g_roberts_jr': 2.0, 'j__antonin_scal…
2     2011-062-01  {'j__john_g_roberts_jr': 1.0, 'j__antonin_scal…

                                        votes_detail  is_eq_divided  \
0  {'j__john_g_roberts_jr': 1.0, 'j__antonin_scal…            0.0
1  {'j__john_g_roberts_jr': 1.0, 'j__antonin_scal…            0.0
2  {'j__john_g_roberts_jr': 2.0, 'j__antonin_scal…            0.0

                                         votes_side meta.case_id  \
0  {'j__john_g_roberts_jr': 1.0, 'j__antonin_scal…          NaN
1  {'j__john_g_roberts_jr': 0.0, 'j__antonin_scal…  2011_11-182
2  {'j__john_g_roberts_jr': 1.0, 'j__antonin_scal…  2011_11-161

                                              text num_conversations  \
0                                              NaN               NaN
1  We'll hear argument this morning in Case 11-18…               1.0
2  We will hear argument this morning in case 11-…               1.0

   num_utterances
0             NaN
1           295.0
2           239.0

[3 rows x 25 columns]
```

[ ]:
```
df_cases_convo.dropna(subset=['text'], inplace=True)
```

[ ]:
```
# transform to pd.to_datetime
df_cases_convo.decided_date = pd.to_datetime(df_cases_convo.decided_date)
```

[ ]:
```
df_cases_convo.to_csv('df_cases_convo.csv', index=False)
```

[ ]:
```
# Cleaning the text
def preprocess_text(text):
    text = text.lower() # Lowercase the text
    text = re.sub('[^a-z]+', ' ', text)  # Remove special characters and numbers
    text = re.sub(r'\b\w{1,3}\b', '', text) # Remove words with length less␣
 ↪than 3
    words = nltk.word_tokenize(text) # Tokenize the text
    stop_words = set(stopwords.words('english')) # Remove stopwords
    words = [word for word in words if word not in stop_words]
    #lemmatizer = WordNetLemmatizer() # Lemmatize the words comment because slow
```

```
    #words = [lemmatizer.lemmatize(word) for word in words]
    stemmer = PorterStemmer() # Stem the words
    words = [stemmer.stem(word) for word in words]
    text = ' '.join(words) # Reconstruct the text

    return text
```

[ ]:
```
df = df_cases_convo.copy()
```

[ ]:
```
text = df_cases_convo.loc[:,['text','win_side']]
```

[ ]:
```
text.to_csv('text.csv', index=False)
```

[ ]:
```
# preprocess text
df.loc[:, 'text_pre'] = df['text'].apply(preprocess_text)
text.to_csv('text_clean.csv', index=False)
```

[ ]:
```
text = pd.read_csv('text_clean.csv')
```

[ ]:
```
text.head()
```

[ ]:

	text	win_side
0	We'll hear argument this morning in Case 11-18…	0.0
1	We will hear argument this morning in case 11-…	0.0
2	We will hear argument first this morning in Ca…	1.0
3	We'll hear argument next in Case 10-1320, Blue…	0.0
4	We'll hear argument first this morning in Case…	1.0

[ ]:
```
text.loc[:, 'text'] = text['text'].apply(preprocess_text)
```

[ ]:
```
text.head()
```

[ ]:

	text	win_side
0	hear argument morn case arizona unit state cle…	0.0
1	hear argument morn case armour citi indianapol…	0.0
2	hear argument first morn case astru capato mil…	1.0
3	hear argument next case blueford arkansa sloan…	0.0
4	hear argument first morn case caraco pharmaceu…	1.0

[ ]:
```
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
```

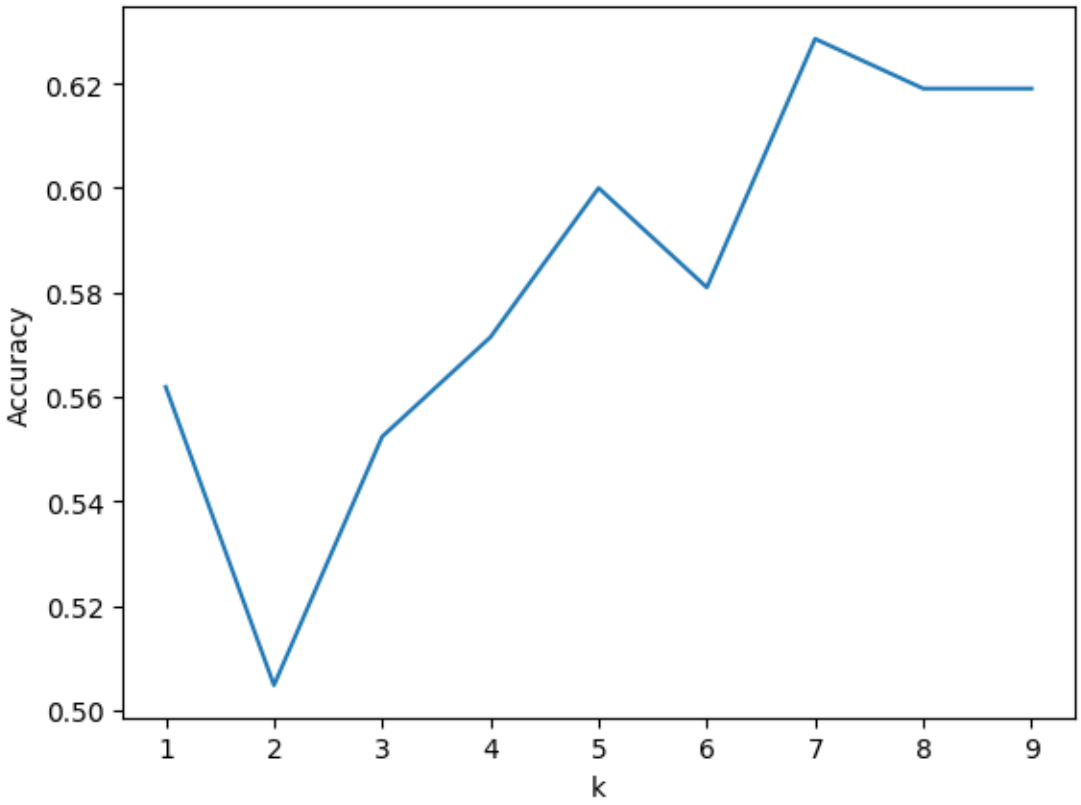
[ ]:
```
def knn_classification_plot(dataset, text_column, label_column, method='bow',␣
 ↪k_range=range(1, 10)):
  X = dataset[text_column]
```

```
  y = dataset[label_column]
  if method == 'bow':
    vectorizer = CountVectorizer()
  elif method == 'tfidf':
    vectorizer = TfidfVectorizer(min_df=5, max_df=0.7)
  X = vectorizer.fit_transform(X)
  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
  accuracies = []
  for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    accuracies.append(accuracy)

  best_k = k_range[accuracies.index(max(accuracies))]
  print(f"Best value of k: {best_k} with accuracy: {max(accuracies)}")

  plt.plot(k_range, accuracies)
  plt.xlabel('k')
  plt.ylabel('Accuracy')
  plt.show()
```

[ ]:
```
dataset = text
text_column = 'text'
label_column = 'win_side'
```

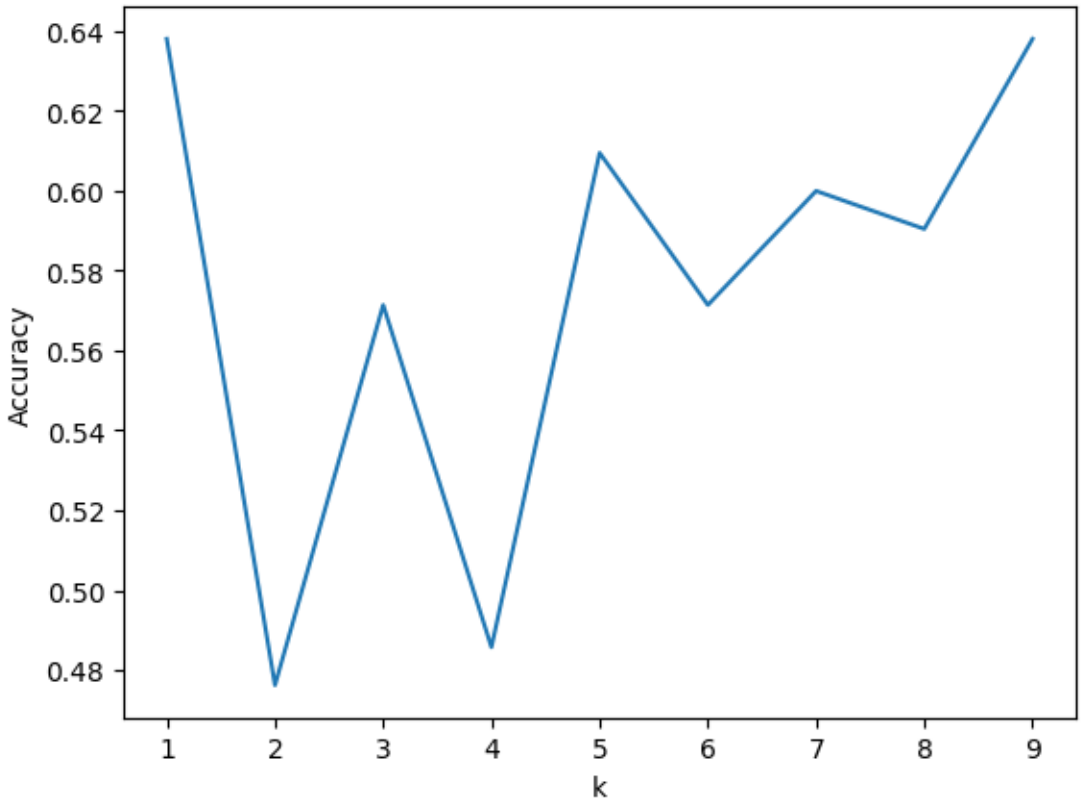
[ ]:
```
knn_classification_plot(dataset, text_column, label_column, method='bow')
```

```
Best value of k: 7 with accuracy: 0.6285714285714286
```



[ ]:
```
knn_classification_plot(dataset, text_column, label_column, method='tfidf')
```

```
Best value of k: 1 with accuracy: 0.638095238095238
```

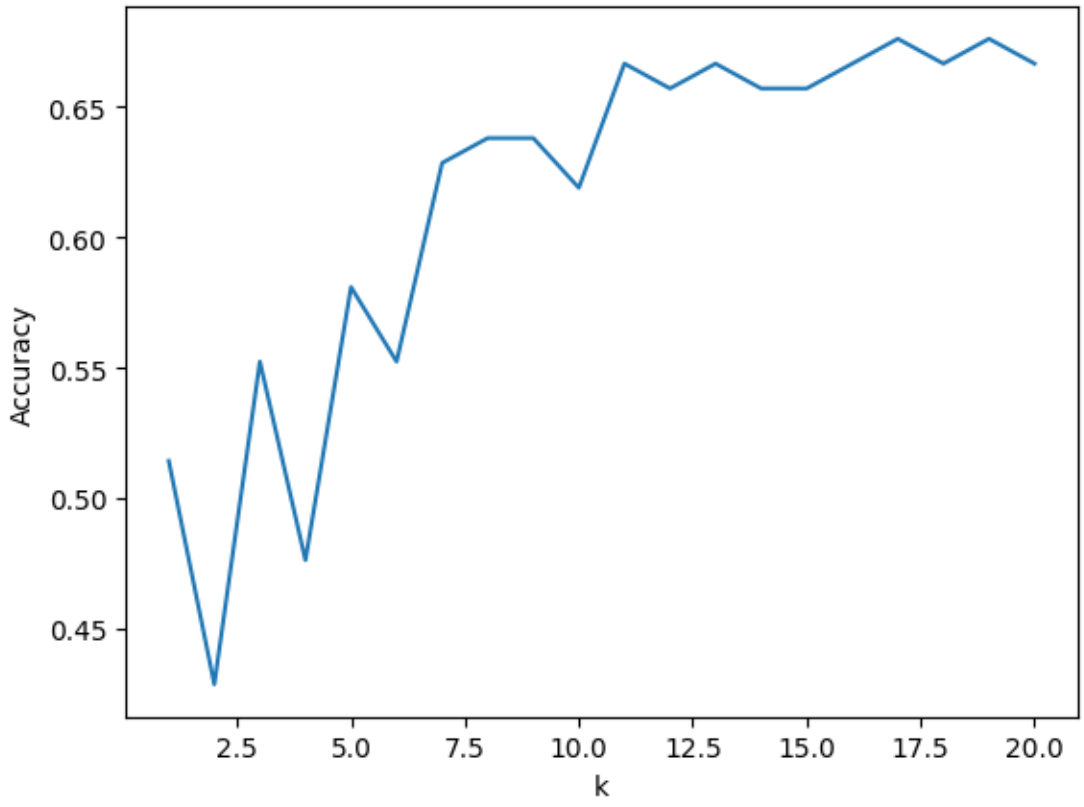


[ ]:
```
import numpy as np

def knn_avg_plot(dataset, text_column, label_column, ngram_range, min_df,␣
 ↪max_df, method='bow', k_range=range(1, 21), n_splits=10):
  split_accuracies = []
  for i in range(n_splits):
    X_train, X_test, y_train, y_test = train_test_split(dataset[text_column],␣
 ↪dataset[label_column], test_size=0.2)
    if method == 'bow':
      vectorizer = CountVectorizer(ngram_range=ngram_range)
    elif method == 'tfidf':
      vectorizer = TfidfVectorizer(min_df=min_df, max_df=max_df)
    X_train = vectorizer.fit_transform(X_train)
    X_test = vectorizer.transform(X_test)
    accuracies = []
  for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    accuracies.append(accuracy)
```

```
    split_accuracies.append(accuracies)
    #Take mean accuracy across random splits
    avg_accuracies = np.mean(split_accuracies, axis=0)
    #Find best value of k vis-a-vis accuracy
    best_k = k_range[np.argmax(avg_accuracies)]
    print(f"Best value of k: {best_k} with accuracy: {max(avg_accuracies)}")
    #Plot accuracies
    plt.plot(k_range, avg_accuracies)
    plt.xlabel('k')
    plt.ylabel('Accuracy')
    plt.show()
```

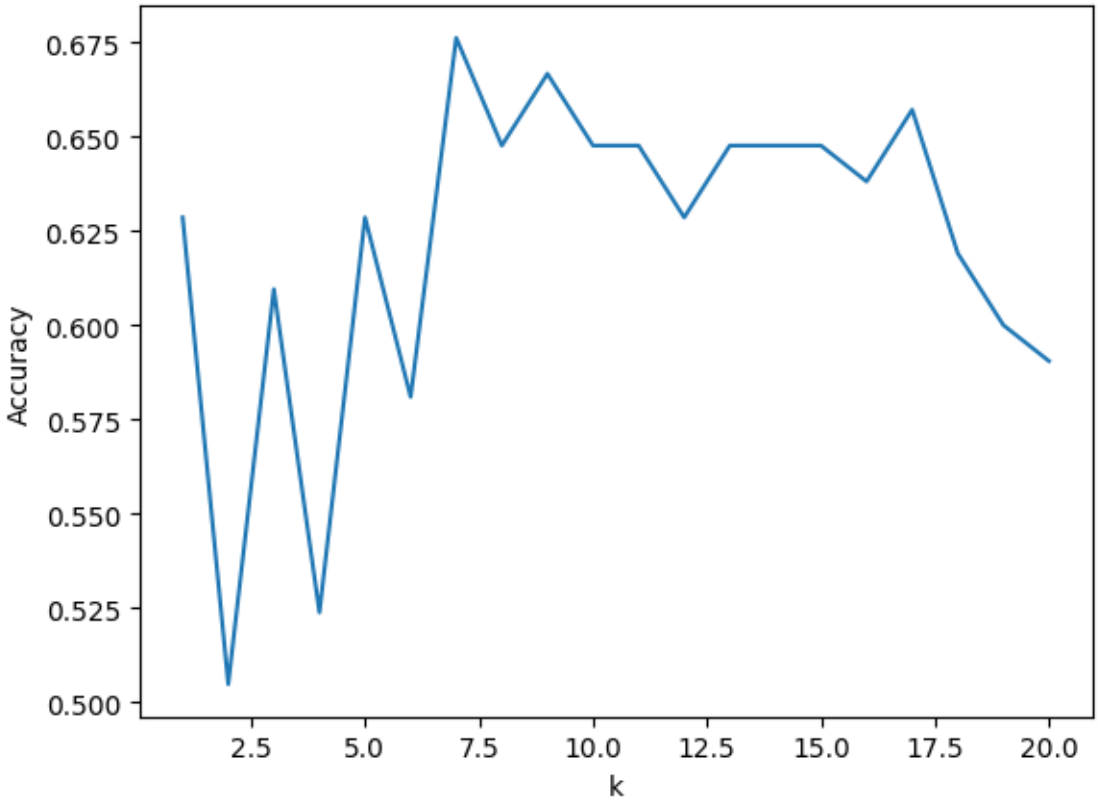
[ ]:
```
ngram_range = (1,2)
min_df = 7
max_df = 0.7
knn_avg_plot(dataset, text_column, label_column, ngram_range, min_df, max_df,␣
 ↪method='bow')
```

```
Best value of k: 17 with accuracy: 0.6761904761904762
```



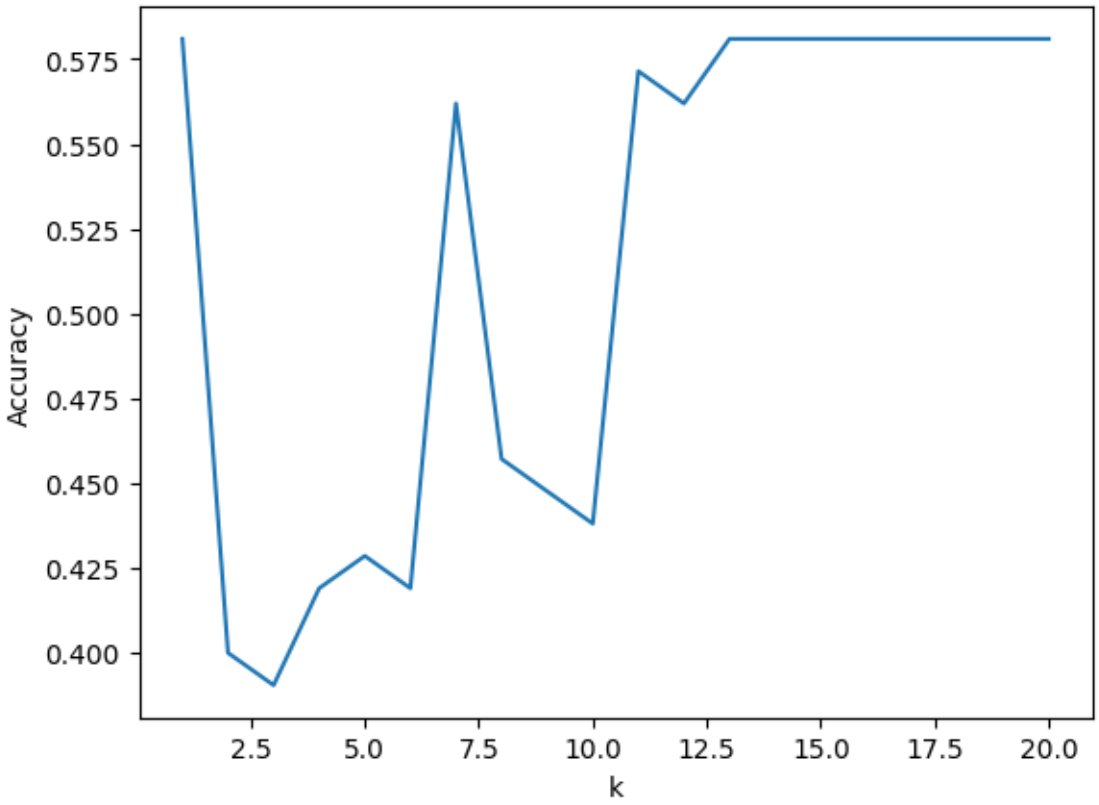
[ ]:
```
ngram_range = (2,2)
min_df = 5
max_df = 0.7
knn_avg_plot(dataset, text_column, label_column, ngram_range, min_df, max_df,␣
 ↪method='bow')
```

```
Best value of k: 7 with accuracy: 0.6761904761904762
```



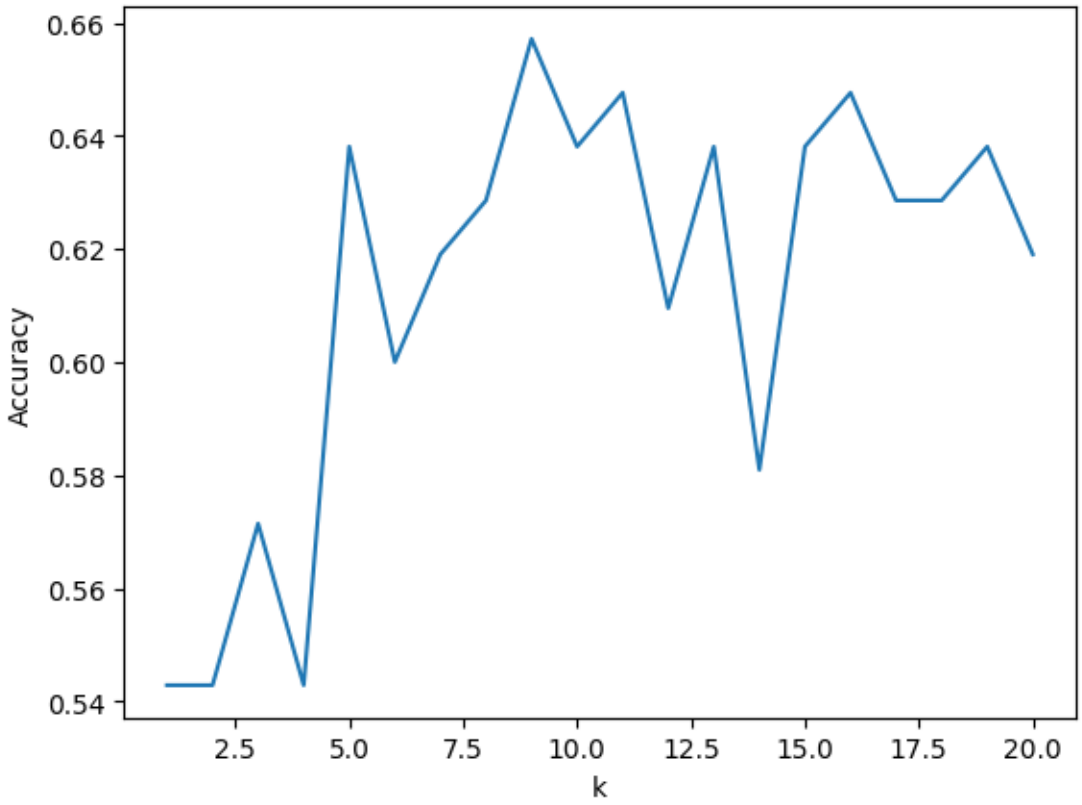
[ ]:
```
ngram_range = (3,3)
min_df = 5
max_df = 0.7
knn_avg_plot(dataset, text_column, label_column, ngram_range, min_df, max_df,␣
 ↪method='bow')
```

```
Best value of k: 1 with accuracy: 0.580952380952381
```



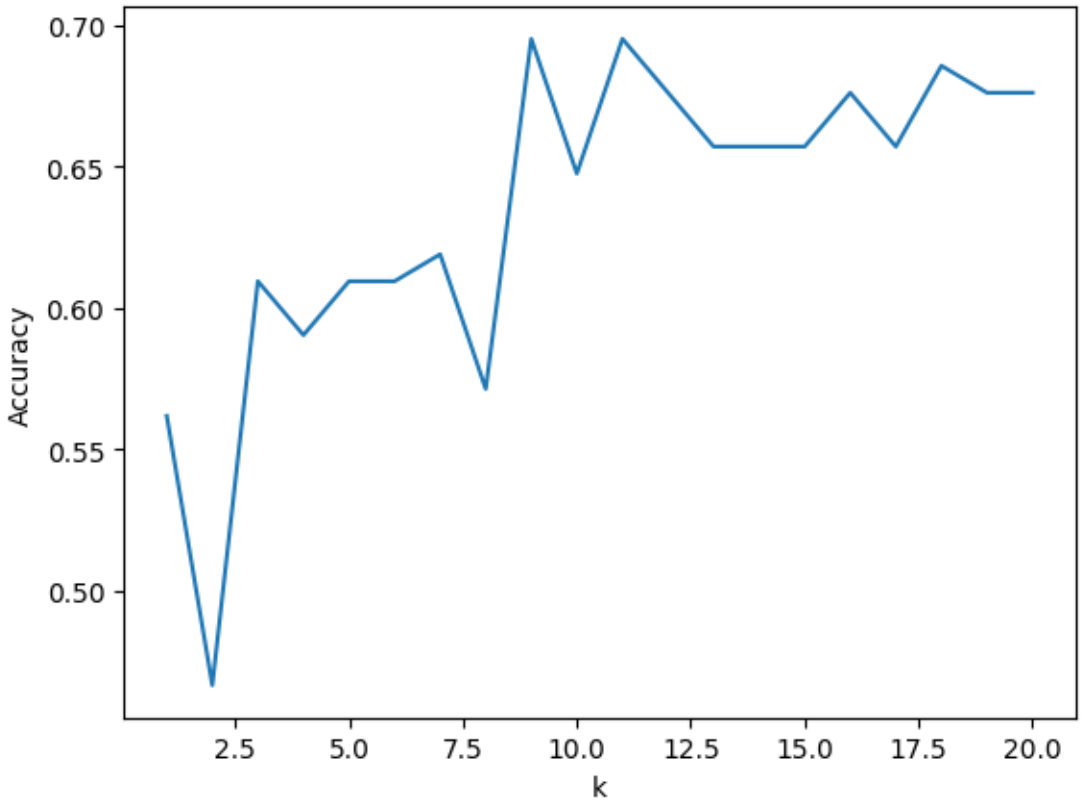
[ ]:
```
ngram_range = (2,2)
min_df = 6
max_df = 0.7
knn_avg_plot(dataset, text_column, label_column, ngram_range, min_df, max_df,␣
 ↪method='tfidf')
```

```
Best value of k: 9 with accuracy: 0.6571428571428571
```



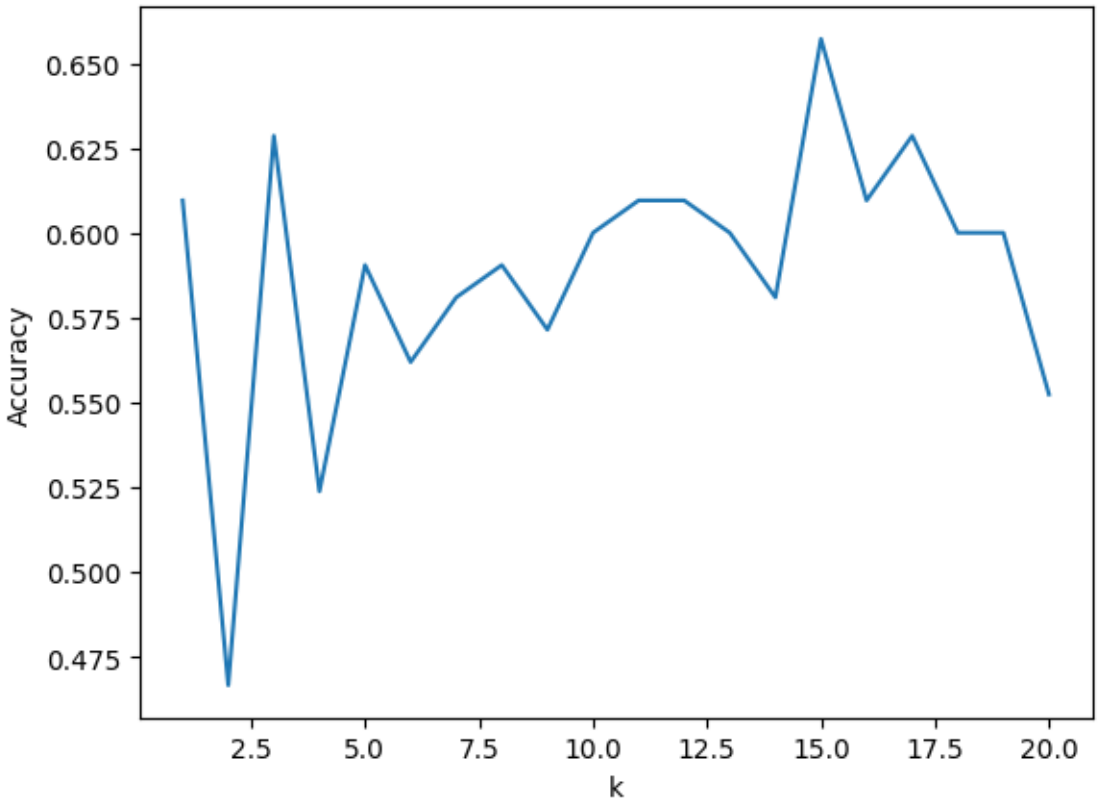
[ ]:
```
ngram_range = (2,2)
min_df = 7
max_df = 0.7
knn_avg_plot(dataset, text_column, label_column, ngram_range, min_df, max_df,␣
  ↪method='tfidf')
```

```
Best value of k: 9 with accuracy: 0.6952380952380952
```



[ ]:
```
ngram_range = (2,2)
min_df = 8
max_df = 0.7
knn_avg_plot(dataset, text_column, label_column, ngram_range, min_df, max_df,␣
 ↪method='tfidf')
```

```
Best value of k: 15 with accuracy: 0.6571428571428571
```


0.650
0.625
0.600
0.575
0.550
0.525
0.500
0.475
Accuracy
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
k

Checkpoint 2_Random Forest

May 8, 2023

Data processing

In [1]:
```
import pandas as pd
from ast import literal_eval
import seaborn as sns
import matplotlib.pyplot as plt
```

In [2]:
```
path = '/Users/wangchenhui/UChicago/ML/Final_Project/dataset/'
```

In [3]:
```
# load downloaded data
df_convos = pd.read_csv(path+'/conversations.csv')
df_speakers = pd.read_csv(path+'/speakers.csv')
df_utts = pd.read_csv(path+'/utterances.csv')
df_cases = pd.read_json(path_or_buf='https://zissou.infosci.cornell.edu/convokit/datasets/supreme-corpu
df_cases = df_cases[(df_cases['year'] >= 2011) & (df_cases['year'] <= 2018) & (df_cases['win_side'].is
```

In [4]:
```
# combine text from all utterances in a conversation back into one string based on the conversation_id
utt_per_conv = df_utts.groupby('conversation_id')['text'].apply(lambda x: ' '.join(x)).reset_index()
utt_per_conv['num_utterances'] = df_utts.groupby('conversation_id')['text'].count().reset_index()['tex

# add the combined text to the conversations dataframe, merge on conversation_id in utt_per_conv and i
df_convos_utt = df_convos.merge(utt_per_conv, left_on='id', right_on='conversation_id', how='left')
```

In [5]:
```
# combine text from all conversation in a cases into one string based on the meta.case_id
conv_per_case = df_convos_utt.groupby('meta.case_id')['text'].apply(lambda x: ' '.join(x)).reset_index
conv_per_case['num_conversations'] = df_convos_utt.groupby('meta.case_id')['text'].count().reset_index
conv_per_case['num_utterances'] = df_convos_utt.groupby('meta.case_id')['num_utterances'].sum().reset_

# add the combined text case dataframe, merge on meta.case_id and id
df_cases_convo = df_cases.merge(conv_per_case, left_on='id', right_on='meta.case_id', how='left')
```

In [6]:
```
df_cases_convo.dropna(subset=['text'], inplace=True)
```

In [7]:
```
# transform to pd.to_datetime
df_cases_convo.decided_date = pd.to_datetime(df_cases_convo.decided_date)
```

In [8]:
```
df_cases_convo.to_csv('df_cases_convo.csv', index=False)
```

Clean Data

In [9]:
```
import nltk
import re
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('wordnet')
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/wangchenhui/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to
[nltk_data]     /Users/wangchenhui/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     /Users/wangchenhui/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

In [10]:
```
# Cleaning the text
def preprocess_text(text):
    text = text.lower() # Lowercase the text
    text = re.sub('[^a-z]+', ' ', text)  # Remove special characters and numbers
    text = re.sub(r'\b\w{1,3}\b', '', text) # Remove words with length less than 3
    words = nltk.word_tokenize(text) # Tokenize the text
    stop_words = set(stopwords.words('english')) # Remove stopwords
    words = [word for word in words if word not in stop_words]
    #lemmatizer = WordNetLemmatizer() # Lemmatize the words comment because slow
    #words = [lemmatizer.lemmatize(word) for word in words]
    text = ' '.join(words) # Reconstruct the text

    return text
```

In [11]:
```
df = df_cases_convo.copy()
```

In [12]:
```
# preprocess text
df.loc[:, 'text_pre'] = df['text'].apply(preprocess_text)
```

In [13]:
```
# preprocess develop time
df.loc[:, 'start_date'] = df['transcripts'].apply(lambda x : re.findall(r'[A-Z][a-z]+ \d{2}, \d{4}', x
df.start_date = pd.to_datetime(df.start_date)
df.loc[:, 'develop_time'] = df.loc[:, 'decided_date'] - df.loc[:, 'start_date']
# df['develop_time'] = df['develop_time'].apply(lambda x : x.days)
```

In [14]:
```
# get party of the judges

def check_party_pc(x):
    rep_judge = ['j__clarence_thomas', 'j__anthony_m_kennedy', 'j__antonin_scalia', 'j__john_g_roberts_
            'j__john_paul_stevens', 'j__david_h_souter', 'j__william_h_rehnquist', 'j__neil_gorsuch',
    dem_judge = ['j__ruth_bader_ginsburg', 'j__stephen_g_breyer','j__sonia_sotomayor','j__elena_kagan'

    rep_ct = 0

    for judge in x:
        if judge in rep_judge:
            rep_ct += 1

    return rep_ct/len(x)
```

In [15]:
```
df['votes_side'][1]['j__john_g_roberts_jr']
```

Out[15]: 0.0

In [16]:
```
# get rep_judge yes

def check_rep_j_y_pc(x):
    rep_judge = ['j__clarence_thomas', 'j__anthony_m_kennedy', 'j__antonin_scalia', 'j__john_g_roberts_
             'j__john_paul_stevens', 'j__david_h_souter', 'j__william_h_rehnquist', 'j__neil_gorsuch',
    dem_judge = ['j__ruth_bader_ginsburg', 'j__stephen_g_breyer','j__sonia_sotomayor','j__elena_kagan'

    rep_y_ct = 0

    for judge in x:
        if judge in rep_judge:
            if x[judge] > 0:
                rep_y_ct += 1

    return rep_y_ct/len(x)
```

In [17]:
```
# get dem_judge yes

def check_dem_j_y_pc(x):
    rep_judge = ['j__clarence_thomas', 'j__anthony_m_kennedy', 'j__antonin_scalia', 'j__john_g_roberts_
             'j__john_paul_stevens', 'j__david_h_souter', 'j__william_h_rehnquist', 'j__neil_gorsuch',
    dem_judge = ['j__ruth_bader_ginsburg', 'j__stephen_g_breyer','j__sonia_sotomayor','j__elena_kagan'

    dem_y_ct = 0

    for judge in x:
        if judge in dem_judge:
            if x[judge] > 0:
                dem_y_ct += 1

    return dem_y_ct/len(x)
```

In [18]:
```
def check_party(x):
    if x > 2009:
        return 0
    else:
        return 1
```

In [19]:
```
# only numbers can apply to Random Forest Model
df_rf = pd.DataFrame()
df_rf.loc[:, 'text_len'] = df['text'].apply(lambda x : len(x))
df_rf.loc[:, 'text_pre_len'] = df['text_pre'].apply(lambda x : len(x))
df_rf.loc[:, 'num_utterances'] = df['num_utterances']
df_rf.loc[:, 'win_side'] = df['win_side']
df_rf.loc[:, 'develop_time'] = df['develop_time'].apply(lambda x : x.days)
df_rf.loc[:, 'rep_jpc'] = df['votes_side'].apply(check_party_pc)
df_rf.loc[:, 'rep_j_y_pc'] = df['votes_side'].apply(check_rep_j_y_pc)
df_rf.loc[:, 'dem_j_y_pc'] = df['votes_side'].apply(check_dem_j_y_pc)
df_rf.loc[:, 'party'] = df['year'].apply(check_party) # 1:  rep, 0: dem

df_rf
```

Out[19]:

	text_len	text_pre_len	num_utterances	win_side	develop_time	rep_jpc	rep_j_y_pc	dem_j_y_pc	party
1	81913	44829	295.0	0.0	61	0.625000	0.375000	0.000000	0
2	66589	34303	239.0	0.0	96	0.555556	0.333333	0.000000	0
3	55436	29849	201.0	1.0	63	0.555556	0.555556	0.444444	0
4	55012	29892	191.0	0.0	92	0.555556	0.000000	0.333333	0
6	59768	31534	210.0	1.0	134	0.555556	0.555556	0.444444	0
...	...	...	...	...	...	...	...	...	...
595	65067	35907	167.0	1.0	62	0.555556	0.555556	0.444444	0
596	61137	32779	179.0	0.0	91	0.555556	0.333333	0.111111	0
597	58012	32112	220.0	0.0	224	0.555556	0.222222	0.111111	0
598	67120	34254	319.0	0.0	140	0.555556	0.444444	0.000000	0
599	56642	29786	250.0	1.0	57	0.500000	0.500000	0.500000	0

521 rows × 9 columns

Random Forest

In [20]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, f1_score, precisi
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split

import itertools
```

In [60]:
```
def get_accuracy(feature_lst, X, df):

    #set y dataset
    y = df['win_side']
    # Train test split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

    # create the model
    model = RandomForestClassifier()
    # train the model
    model.fit(X_train, y_train)
    # Test the model
    predictions = model.predict(X_train)
    # Make the predictions
    y_pred = model.predict(X_test)

    dict1 = {'features': tuple(feature_lst),
             'features_num': len(feature_lst),
             'accuracy': accuracy_score(y_test, y_pred),
             'f1': f1_score(y_test, y_pred),
             'precision': precision_score(y_test, y_pred),
             'recall': recall_score(y_test, y_pred)}
    return dict1
```

In [22]:
```
# Define the list
#'rep_jpc', 'dem_j_y_pc', 'rep_j_y_pc'
features_list = ['text_len', 'text_pre_len', 'num_utterances', 'develop_time',
                 'rep_jpc', 'party']

# Get all possible combinations of the list
combinations = []
for i in range(1, len(features_list) + 1):
    combinations += list(itertools.combinations(features_list, i))

for i in range(len(combinations)):
    combinations[i] = list(combinations[i] )
combinations
```

Out[22]:
```
[['text_len'],
 ['text_pre_len'],
 ['num_utterances'],
 ['develop_time'],
 ['rep_jpc'],
 ['party'],
 ['text_len', 'text_pre_len'],
 ['text_len', 'num_utterances'],
 ['text_len', 'develop_time'],
 ['text_len', 'rep_jpc'],
 ['text_len', 'party'],
 ['text_pre_len', 'num_utterances'],
 ['text_pre_len', 'develop_time'],
 ['text_pre_len', 'rep_jpc'],
 ['text_pre_len', 'party'],
 ['num_utterances', 'develop_time'],
 ['num_utterances', 'rep_jpc'],
 ['num_utterances', 'party'],
 ['develop_time', 'rep_jpc'],
 ['develop_time', 'party'],
 ['rep_jpc', 'party'],
 ['text_len', 'text_pre_len', 'num_utterances'],
 ['text_len', 'text_pre_len', 'develop_time'],
 ['text_len', 'text_pre_len', 'rep_jpc'],
 ['text_len', 'text_pre_len', 'party'],
 ['text_len', 'num_utterances', 'develop_time'],
 ['text_len', 'num_utterances', 'rep_jpc'],
 ['text_len', 'num_utterances', 'party'],
 ['text_len', 'develop_time', 'rep_jpc'],
 ['text_len', 'develop_time', 'party'],
 ['text_len', 'rep_jpc', 'party'],
 ['text_pre_len', 'num_utterances', 'develop_time'],
 ['text_pre_len', 'num_utterances', 'rep_jpc'],
 ['text_pre_len', 'num_utterances', 'party'],
 ['text_pre_len', 'develop_time', 'rep_jpc'],
 ['text_pre_len', 'develop_time', 'party'],
 ['text_pre_len', 'rep_jpc', 'party'],
 ['num_utterances', 'develop_time', 'rep_jpc'],
 ['num_utterances', 'develop_time', 'party'],
 ['num_utterances', 'rep_jpc', 'party'],
 ['develop_time', 'rep_jpc', 'party'],
 ['text_len', 'text_pre_len', 'num_utterances', 'develop_time'],
 ['text_len', 'text_pre_len', 'num_utterances', 'rep_jpc'],
 ['text_len', 'text_pre_len', 'num_utterances', 'party'],
 ['text_len', 'text_pre_len', 'develop_time', 'rep_jpc'],
 ['text_len', 'text_pre_len', 'develop_time', 'party'],
 ['text_len', 'text_pre_len', 'rep_jpc', 'party'],
 ['text_len', 'num_utterances', 'develop_time', 'rep_jpc'],
 ['text_len', 'num_utterances', 'develop_time', 'party'],
 ['text_len', 'num_utterances', 'rep_jpc', 'party'],
 ['text_len', 'develop_time', 'rep_jpc', 'party'],
 ['text_pre_len', 'num_utterances', 'develop_time', 'rep_jpc'],
 ['text_pre_len', 'num_utterances', 'develop_time', 'party'],
 ['text_pre_len', 'num_utterances', 'rep_jpc', 'party'],
 ['text_pre_len', 'develop_time', 'rep_jpc', 'party'],
 ['num_utterances', 'develop_time', 'rep_jpc', 'party'],
 ['text_len', 'text_pre_len', 'num_utterances', 'develop_time', 'rep_jpc'],
 ['text_len', 'text_pre_len', 'num_utterances', 'develop_time', 'party'],
 ['text_len', 'text_pre_len', 'num_utterances', 'rep_jpc', 'party'],
 ['text_len', 'text_pre_len', 'develop_time', 'rep_jpc', 'party'],
 ['text_len', 'num_utterances', 'develop_time', 'rep_jpc', 'party'],
 ['text_pre_len', 'num_utterances', 'develop_time', 'rep_jpc', 'party'],
 ['text_len',
  'text_pre_len',
  'num_utterances',
  'develop_time',
  'rep_jpc',
  'party']]
```

In [53]:
```
len(combinations)
```

Out[53]:
```
63
```

For one epoch

In [73]:
```
results = []

# get accu from each diff features combinations
for feature_lst in combinations:
    results.append(get_accuracy(feature_lst, df_rf.loc[:,feature_lst], df_rf))

# get accu from ngram, bigram
results.append(get_accuracy(['ngram_text'], CountVectorizer().fit_transform(df['text_pre']), df))
results.append(get_accuracy(['bigram_text'], TfidfVectorizer().fit_transform(df['text_pre']), df))
```

In [74]:
```
# Create a DataFrame from the results list
results_df = pd.DataFrame(results)
results_df
```

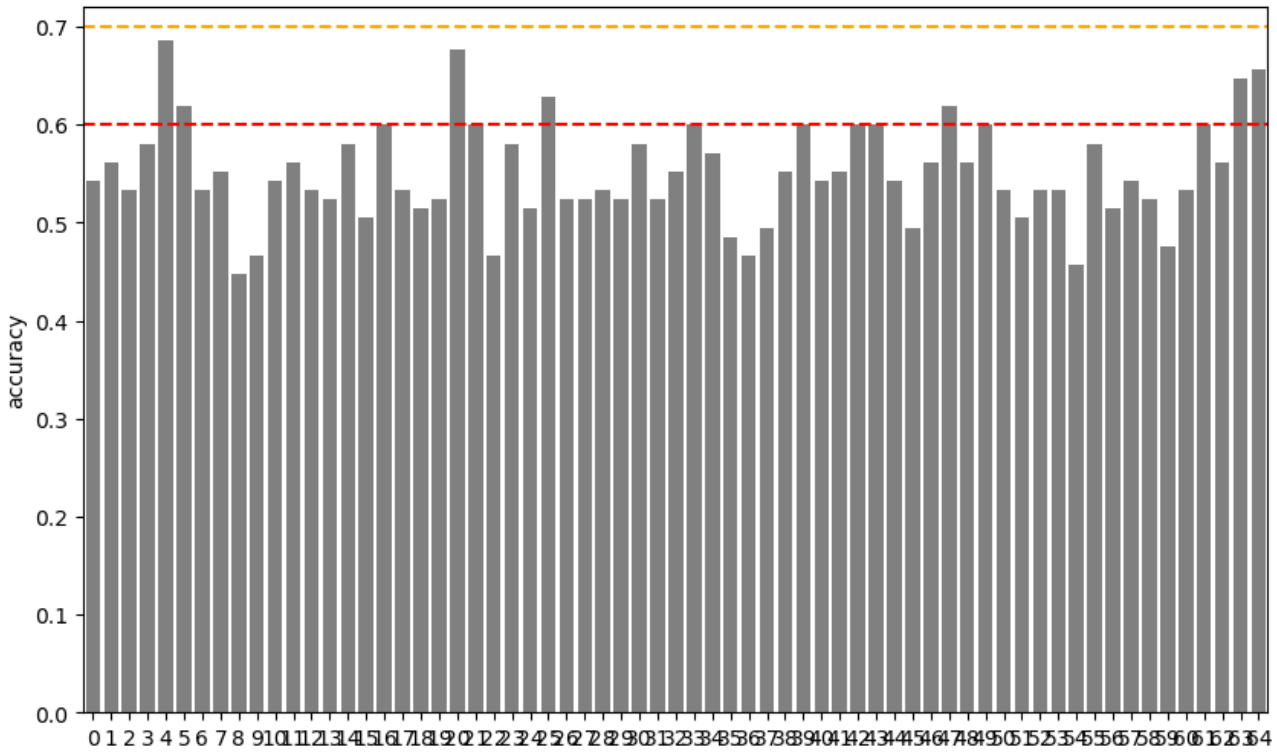
Out[74]:

	features	features_num	accuracy	f1	precision	recall
0	(text_len,)	1	0.542857	0.641791	0.614286	0.671875
1	(text_pre_len,)	1	0.561905	0.616667	0.649123	0.587302
2	(num_utterances,)	1	0.533333	0.642336	0.594595	0.698413
3	(develop_time,)	1	0.580952	0.685714	0.695652	0.676056
4	(rep_jpc,)	1	0.685714	0.811429	0.689320	0.986111
...	...	...	...	...	...	...
60	(text_len, num_utterances, develop_time, rep_j...	5	0.533333	0.652482	0.582278	0.741935
61	(text_pre_len, num_utterances, develop_time, r...	5	0.600000	0.704225	0.746269	0.666667
62	(text_len, text_pre_len, num_utterances, devel...	6	0.561905	0.705128	0.670732	0.743243
63	(ngram_text,)	1	0.647619	0.775758	0.640000	0.984615
64	(bigram_text,)	1	0.657143	0.793103	0.657143	1.000000

65 rows × 6 columns

In [75]:
```
plt.figure(figsize=(10, 6))
sns.barplot(x = results_df.index, y='accuracy', color='grey', data=results_df)
plt.axhline(y=0.6, color='red', linestyle='--')
plt.axhline(y=0.7, color='orange', linestyle='--')
```

Out[75]: <matplotlib.lines.Line2D at 0x7fe8b30a6670>



In [76]:
```
results_df.loc[(results_df.loc[:, 'accuracy'] > 0.6), :]
```

Out[76]:

	features	features_num	accuracy	f1	precision	recall
4	(rep_jpc,)	1	0.685714	0.811429	0.689320	0.986111
5	(party,)	1	0.619048	0.764706	0.619048	1.000000
20	(rep_jpc, party)	2	0.676190	0.806818	0.682692	0.986111
25	(text_len, num_utterances, develop_time)	3	0.628571	0.731034	0.706667	0.757143
47	(text_len, num_utterances, develop_time, rep_jpc)	4	0.619048	0.740260	0.686747	0.802817
63	(ngram_text,)	1	0.647619	0.775758	0.640000	0.984615
64	(bigram_text,)	1	0.657143	0.793103	0.657143	1.000000

In [77]:
```
results_df.loc[(results_df.loc[:, 'accuracy'] > 0.7), :]
```

Out[77]:

	features	features_num	accuracy	f1	precision	recall

For 50 epoch

In [78]:
```
results = []
count = 0
while(count <= 50):
    count += 1
    # get accu from each diff features combinations
    for feature_lst in combinations:
        results.append(get_accuracy(feature_lst, df_rf.loc[:,feature_lst], df_rf))

    # get accu from ngram, bigram
    results.append(get_accuracy(['ngram_text'], CountVectorizer().fit_transform(df['text_pre']), df))
    results.append(get_accuracy(['bigram_text'], TfidfVectorizer().fit_transform(df['text_pre']), df))
```

In [79]:
```
# Create a DataFrame from the results list
results_df = pd.DataFrame(results)
results_df
```

Out[79]:

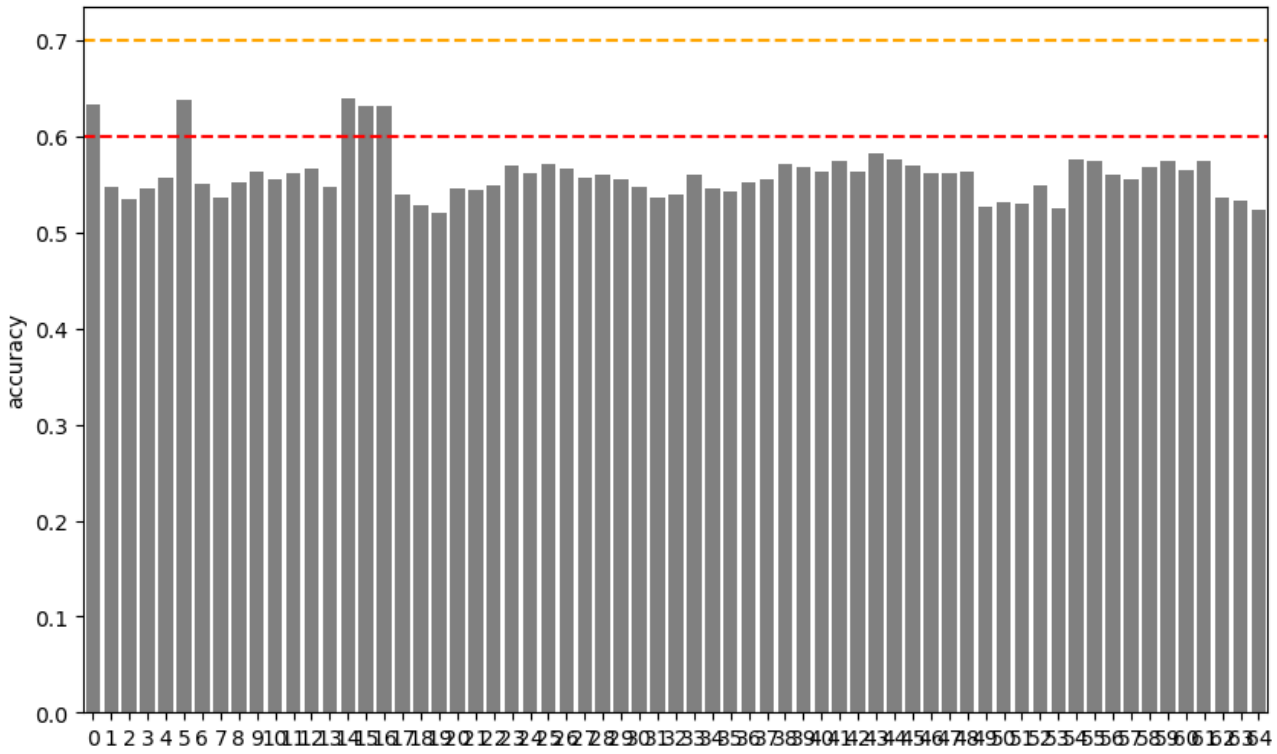
	features	features_num	accuracy	f1	precision	recall
0	(text_len,)	1	0.561905	0.676056	0.685714	0.666667
1	(text_pre_len,)	1	0.542857	0.647059	0.619718	0.676923
2	(num_utterances,)	1	0.523810	0.642857	0.642857	0.642857
3	(develop_time,)	1	0.533333	0.679739	0.584270	0.812500
4	(rep_jpc,)	1	0.580952	0.731707	0.588235	0.967742
...	...	...	...	...	...	...
3310	(text_len, num_utterances, develop_time, rep_j...	5	0.542857	0.657143	0.676471	0.638889
3311	(text_pre_len, num_utterances, develop_time, r...	5	0.485714	0.608696	0.608696	0.608696
3312	(text_len, text_pre_len, num_utterances, devel...	6	0.561905	0.697368	0.616279	0.803030
3313	(ngram_text,)	1	0.676190	0.804598	0.673077	1.000000
3314	(bigram_text,)	1	0.676190	0.806818	0.682692	0.986111

3315 rows × 6 columns

In [80]:
```
results_df = results_df.groupby('features', as_index = False).agg({'features_num': 'mean',
                                                                    'accuracy': 'mean',
                                                                    'f1': 'mean',
                                                                    'precision': 'mean',
                                                                    'recall': 'mean',})
```

In [81]:
```
plt.figure(figsize=(10, 6))
sns.barplot(x = results_df.index, y='accuracy', color='grey', data=results_df)
plt.axhline(y=0.6, color='red', linestyle='--')
plt.axhline(y=0.7, color='orange', linestyle='--')
```

Out[81]: <matplotlib.lines.Line2D at 0x7fe907ae06a0>



In [84]:
```
results_df.loc[(results_df.loc[:, 'accuracy'] > 0.6), :].sort_values(by = 'accuracy', ascending = Fals
```

Out[84]:

	features	features_num	accuracy	f1	precision	recall
14	(party,)	1.0	0.639776	0.779434	0.639776	1.000000
5	(ngram_text,)	1.0	0.637908	0.776424	0.641069	0.986679
0	(bigram_text,)	1.0	0.633053	0.773451	0.636580	0.987719
16	(rep_jpc, party)	2.0	0.632680	0.773840	0.635639	0.992047
15	(rep_jpc,)	1.0	0.631559	0.772821	0.635128	0.989987

In [83]:
```
results_df.loc[(results_df.loc[:, 'accuracy'] > 0.7), :]
```

Out[83]:

	features	features_num	accuracy	f1	precision	recall

```
import pandas as pd
from ast import literal_eval
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

```
import nltk
import re
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('wordnet')
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

```
# path = 'dataset/' #change to your data location
# # load downloaded data
# df_convos = pd.read_csv(path+'/conversations.csv')
# df_speakers = pd.read_csv(path+'/speakers.csv')
# df_utts = pd.read_csv(path+'/utterances.csv')
# df_cases = pd.read_json(path_or_buf=path+'/cases.jsonl', lines=True)
# df_cases = df_cases[(df_cases['year'] >= 2011) & (df_cases['year'] <= 2018) & (df_cases['win_side'].isin([0,1]
```

```
path = 'http://rezarzky.my.id/dataset/' #change to your data location
# load downloaded data
df_convos = pd.read_csv(path+'/conversations.csv')
df_speakers = pd.read_csv(path+'/speakers.csv')
df_utts = pd.read_csv(path+'/utterances.csv')
df_cases = pd.read_json(path_or_buf='https://zissou.infosci.cornell.edu/convokit/datasets/supreme-corpus/cases.j
df_cases = df_cases[(df_cases['year'] >= 2011) & (df_cases['year'] <= 2018) & (df_cases['win_side'].isin([0,1]))
```

```
# count number win/lose cases
df_cases['win_side'].value_counts()
```

```
1.0    400
0.0    201
Name: win_side, dtype: int64
```

```
# combine text from all utterances in a conversation back into one string based on the conversation_id, coount h
utt_per_conv = df_utts.groupby('conversation_id')['text'].apply(lambda x: ' '.join(x)).reset_index()
utt_per_conv['num_utterances'] = df_utts.groupby('conversation_id')['text'].count().reset_index()['text']

# add the combined text to the conversations dataframe, merge on conversation_id in utt_per_conv and id in df_co
df_convos_utt = df_convos.merge(utt_per_conv, left_on='id', right_on='conversation_id', how='left')
```

```
# combine text from all conversation in a cases into one string based on the meta.case_id
conv_per_case = df_convos_utt.groupby('meta.case_id')['text'].apply(lambda x: ' '.join(x)).reset_index()
conv_per_case['num_conversations'] = df_convos_utt.groupby('meta.case_id')['text'].count().reset_index()['text']
conv_per_case['num_utterances'] = df_convos_utt.groupby('meta.case_id')['num_utterances'].sum().reset_index()['n

# add the combined text case dataframe, merge on meta.case_id and id
df_cases_convo = df_cases.merge(conv_per_case, left_on='id', right_on='meta.case_id', how='left')
```

```
df_cases_convo.head(3)
```

	id	year	citation	title	petitioner	respondent	docket_no	court	decided_date	
0	2011_11-1179	2011	567 US _	American Tradition Partnership, Inc. v. Bullock	American Tradition Partnership, Inc.	Steve Bullock, Attorney General of Montana, et...	11-1179	Roberts Court	Jun 25, 2012	https://www.oyez.(
1	2011_11-182	2011	567 US _	Arizona v. United States	Arizona et al.	United States	11-182	Roberts Court	Jun 25, 2012	https://www.oyez.(
2	2011_11-161	2011	566 US _	Armour v. City of Indianapolis	Christine Armour	City of Indianapolis	11-161	Roberts Court	Jun 4, 2012	https://www.oyez.(

3 rows × 25 columns

```
df_cases_convo.dropna(subset=['text'], inplace=True)
df_cases_convo.shape[0]
```

```
521
```

```
# Cleaning the text
def preprocess_text(text):
    text = text.lower() # Lowercase the text
    text = re.sub('[^a-z]+', ' ', text)  # Remove special characters and numbers
    text = re.sub(r'\b\w{1,3}\b', '', text) # Remove words with length less than 3
    words = nltk.word_tokenize(text) # Tokenize the text
    stop_words = set(stopwords.words('english')) # Remove stopwords
    words = [word for word in words if word not in stop_words]
    #lemmatizer = WordNetLemmatizer() # Lemmatize the words comment because slow
    #words = [lemmatizer.lemmatize(word) for word in words]
    text = ' '.join(words) # Reconstruct the text

    return text
```

```
text = df_cases_convo.loc[:,['text','win_side']]
text.head(3)
```

	text	win_side
1	We'll hear argument this morning in Case 11-18...	0.0
2	We will hear argument this morning in case 11-...	0.0
3	We will hear argument first this morning in Ca...	1.0

```
text['text'] = text['text'].apply(preprocess_text) #apply preprocess
text.head(10)
```

	text	win_side
1	hear argument morning case arizona united stat...	0.0
2	hear argument morning case armour city indiana...	0.0
3	hear argument first morning case astrue capato...	1.0
4	hear argument next case blueford arkansas sloa...	0.0
6	hear argument first morning case caraco pharma...	1.0
8	hear argument morning case christopher smithkl...	0.0
10	hear argument first morning case coleman court...	0.0
11	hear argument next case compucredit corporatio...	1.0
12	hear argument next case number credit suisse s...	1.0
13	hear argument morning case dorsey united state...	1.0

```
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import LinearSVC
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, f1_score, precision_score,
from sklearn.feature_selection import chi2
from sklearn.feature_extraction.text import CountVectorizer
import pickle
import numpy as np

#bag of words
bow_converter = CountVectorizer()
x = bow_converter.fit_transform(text['text'])
words = bow_converter.vocabulary_.keys()
len(words)
```

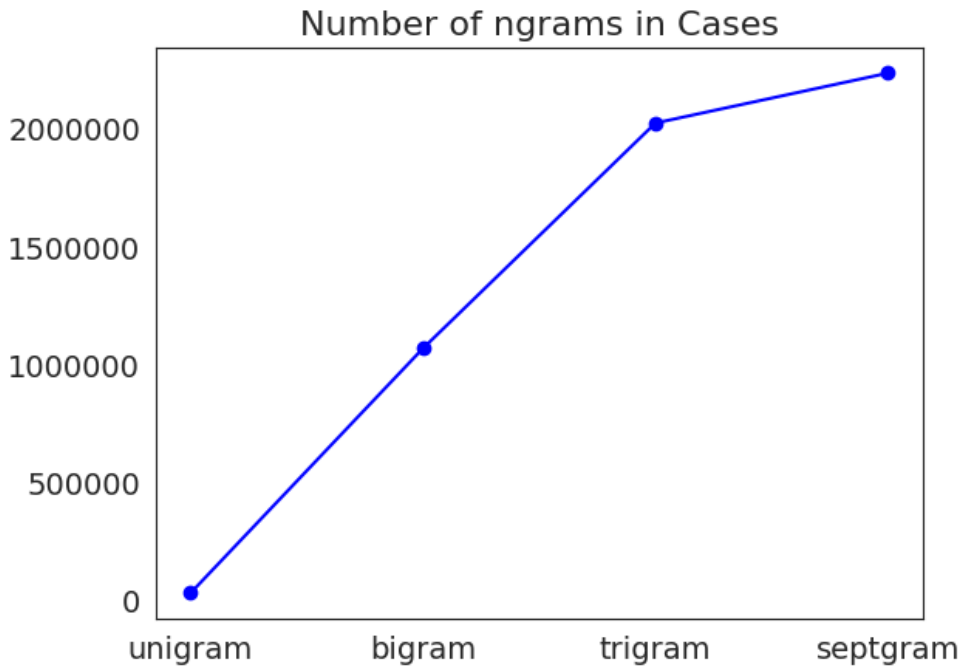
```
28920
```

```
#bag of n-grams
bigram_converter = CountVectorizer(ngram_range=(2,2))
x2 = bigram_converter.fit_transform(text['text'])
bigrams = bigram_converter.get_feature_names_out()
```

```
trigram_converter = CountVectorizer(ngram_range=(3,3))
x3 = trigram_converter.fit_transform(text['text'])
trigram = trigram_converter.get_feature_names_out()
```

```


quadgram_converter= CountVectorizer(ngram_range=(4,4))
x4 = quadgram_converter.fit_transform(text['text'])
quadgram = quadgram_converter.get_feature_names_out()

cinqgram_converter= CountVectorizer(ngram_range=(5,5))
x5 = quadgram_converter.fit_transform(text['text'])
quadgram = quadgram_converter.get_feature_names_out()

sixgram_converter= CountVectorizer(ngram_range=(6,6))
x6 = sixgram_converter.fit_transform(text['text'])
sixgram = sixgram_converter.get_feature_names_out()

septgram_converter= CountVectorizer(ngram_range=(7,7))
x7 = septgram_converter.fit_transform(text['text'])
septgram = septgram_converter.get_feature_names_out()
```

```
sns.set_style("white")
counts = [len(words), len(bigrams), len(trigram), len(septgram) ]
plt.plot(counts, color='blue')
plt.plot(counts, 'bo')
#plt.margins(0.1)
plt.ticklabel_format(style = 'plain')
plt.xticks(range(4), ['unigram', 'bigram', 'trigram', 'septgram'])
plt.tick_params(labelsize=14)
plt.title('Number of ngrams in Cases', {'fontsize':16})
plt.show()
```




```
from sklearn.feature_extraction import text
```

Bag of Words Transformation

```
X = text['text']
y =  text['win_side']

training_data, test_data = train_test_split(text, random_state=42)
bow_transform = CountVectorizer(ngram_range=(3,3), lowercase=False)
```

```
X_tr_bow = bow_transform.fit_transform(training_data['text'])
```

```
len(bow_transform.vocabulary_)
```

```
2214078
```

```
X_tr_bow.shape
```

```
(390, 2214078)
```

```
X_te_bow = bow_transform.transform(test_data['text'])
```

```
y_tr = training_data['win_side']
y_te = test_data['win_side']
```

```
# Create the tf-idf representation using the bag-of-words matrix
from sklearn.feature_extraction import text


tfidf_transform = text.TfidfTransformer()
X_tr_tfidf = tfidf_transform.fit_transform(X_tr_bow)
```

```
X_te_tfidf = tfidf_transform.transform(X_te_bow)
```

Classification with Logistic Regression

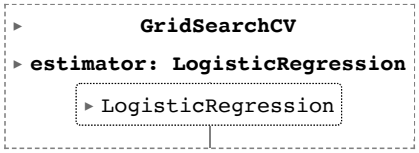
```
def simple_logistic_classify(X_tr, y_tr, X_test, y_test, description, _C=1.0):
    model = LogisticRegression(C=_C).fit(X_tr, y_tr)
    score = model.score(X_test, y_test)
    print('Test Score with', description, 'features', score)
    return model
```

```
model_bow = simple_logistic_classify(X_tr_bow, y_tr, X_te_bow, y_te, 'bow')
model_tfidf = simple_logistic_classify(X_tr_tfidf, y_tr, X_te_tfidf, y_te, 'tf-idf')
```

```
Test Score with bow features 0.6564885496183206
Test Score with tf-idf features 0.6641221374045801
```

```
import sklearn.model_selection

param_grid_ = {'C': [1e-5, 1e-3, 1e-1, 1e0, 1e1, 1e2]}
bow_search = sklearn.model_selection.GridSearchCV(LogisticRegression(), cv=5, param_grid=param_grid_)
tfidf_search =sklearn.model_selection.GridSearchCV(LogisticRegression(), cv=5,
                                   param_grid=param_grid_)
```

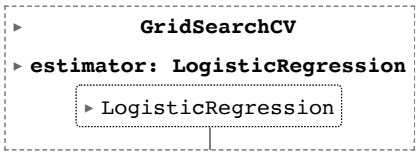
```
bow_search.fit(X_tr_bow, y_tr)
```




```
bow_search.best_score_
```

```
0.6256410256410256
```

```
tfidf_search.fit(X_tr_tfidf, y_tr)
```




```
tfidf_search.best_score_
```

```
0.6256410256410256
```

```
bow_search.best_params_
```

```
{'C': 1e-05}
```

```
tfidf_search.best_params_
```

```
{'C': 1e-05}
```

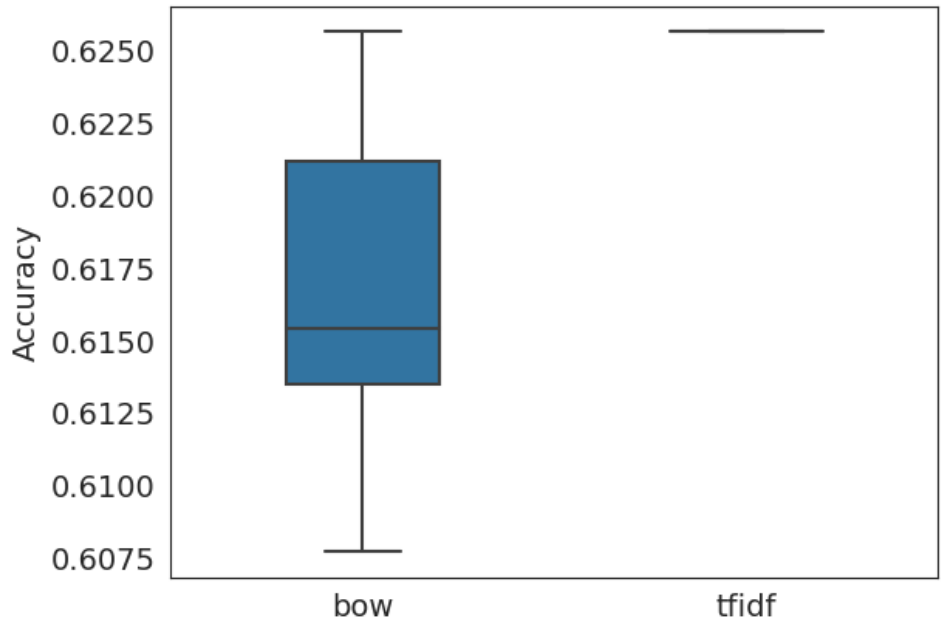
```
bow_search.cv_results_
```

```
{'mean_fit_time': array([13.27562661, 19.39422917, 27.39296756, 28.68174825, 30.77088923,
        20.50505261]),
 'std_fit_time': array([1.69049347, 0.76807263, 0.93387687, 1.48087482, 1.05447008,
        1.94858288]),
 'mean_score_time': array([0.00862556, 0.00819273, 0.00906992, 0.00925088, 0.00922804,
        0.01067486]),
 'std_score_time': array([0.00146609, 0.00073957, 0.00164712, 0.00196291, 0.00139663,
        0.00182396]),
 'param_C': masked_array(data=[1e-05, 0.001, 0.1, 1.0, 10.0, 100.0],
              mask=[False, False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'params': [{'C': 1e-05},
  {'C': 0.001},
  {'C': 0.1},
  {'C': 1.0},
  {'C': 10.0},
  {'C': 100.0}],
 'split0_test_score': array([0.62820513, 0.61538462, 0.61538462, 0.61538462, 0.61538462,
        0.61538462]),
 'split1_test_score': array([0.62820513, 0.62820513, 0.61538462, 0.61538462, 0.61538462,
        0.6025641 ]),
 'split2_test_score': array([0.62820513, 0.61538462, 0.6025641 , 0.6025641 , 0.6025641 ,
        0.6025641 ]),
 'split3_test_score': array([0.62820513, 0.62820513, 0.61538462, 0.61538462, 0.61538462,
        0.6025641 ]),
 'split4_test_score': array([0.61538462, 0.62820513, 0.61538462, 0.62820513, 0.62820513,
        0.61538462]),
 'mean_test_score': array([0.62564103, 0.62307692, 0.61282051, 0.61538462, 0.61538462,
        0.60769231]),
 'std_test_score': array([0.00512821, 0.00628074, 0.00512821, 0.0081084 , 0.0081084 ,
        0.00628074]),
 'rank_test_score': array([1, 2, 5, 3, 3, 6], dtype=int32)}
```

```
search_results = pd.DataFrame.from_dict({'bow': bow_search.cv_results_['mean_test_score'],
                                         'tfidf': tfidf_search.cv_results_['mean_test_score']})
search_results
```

	bow	tfidf
0	0.625641	0.625641
1	0.623077	0.625641
2	0.612821	0.625641
3	0.615385	0.625641
4	0.615385	0.625641

```
%matplotlib inline
ax = sns.boxplot(data=search_results, width=0.4)
ax.set_ylabel('Accuracy', size=14)
ax.tick_params(labelsize=14)
```




```
model_bow = simple_logistic_classify(X_tr_bow, y_tr, X_te_bow, y_te, 'bow',
                                     _C=bow_search.best_params_['C'])
model_tfidf = simple_logistic_classify(X_tr_tfidf, y_tr, X_te_tfidf, y_te, 'tf-idf',
                                     _C=tfidf_search.best_params_['C'])
```

```
Test Score with bow features 0.6641221374045801
Test Score with tf-idf features 0.6641221374045801
```

✓ 21s completed at 4:05 PM

In [3]:
```
import pandas as pd
from ast import literal_eval
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

In [4]:
```
import nltk
import re
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('wordnet')
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from nltk.stem import PorterStemmer
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\rezar\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\rezar\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\rezar\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

Load Data

In [10]:
```
path = 'dataset/' #change to your data location
# load downloaded data
df_convos = pd.read_csv(path+'/conversations.csv')
df_speakers = pd.read_csv(path+'/speakers.csv')
df_utts = pd.read_csv(path+'/utterances.csv')
df_cases = pd.read_json(path_or_buf=path+'/cases.jsonl', lines=True)
df_cases = df_cases[(df_cases['year'] >= 2011) & (df_cases['year'] <= 2018) & (df_cases['win_side'].isin([0,1]))]
```

In [11]:
```
# count number win/lose cases
df_cases['win_side'].value_counts()
```

Out[11]:
```
1.0    400
0.0    201
Name: win_side, dtype: int64
```

In [12]:
```
# combine text from all utterances in a conversation back into one string based on the conversation_id, coount how many utterances
utt_per_conv = df_utts.groupby('conversation_id')['text'].apply(lambda x: ' '.join(x)).reset_index()
utt_per_conv['num_utterances'] = df_utts.groupby('conversation_id')['text'].count().reset_index()['text']

# add the combined text to the conversations dataframe, merge on conversation_id in utt_per_conv and id in df_convo
df_convos_utt = df_convos.merge(utt_per_conv, left_on='id', right_on='conversation_id', how='left')
```

In [13]:
```
# combine text from all conversation in a cases into one string based on the meta.case_id
conv_per_case = df_convos_utt.groupby('meta.case_id')['text'].apply(lambda x: ' '.join(x)).reset_index()
conv_per_case['num_conversations'] = df_convos_utt.groupby('meta.case_id')['text'].count().reset_index()['text']
conv_per_case['num_utterances'] = df_convos_utt.groupby('meta.case_id')['num_utterances'].sum().reset_index()['num_utterances']

# add the combined text case dataframe, merge on meta.case_id and id
df_cases_convo = df_cases.merge(conv_per_case, left_on='id', right_on='meta.case_id', how='left')
df_cases_convo.head(3)
```

Out[13]:

	id	year	citation	title	petitioner	respondent	docket_no	court	decided_date	url	...	win_side_detail
0	2011_11-1179	2011	567 US _	American Tradition Partnership, Inc. v. Bullock	American Tradition Partnership, Inc.	Steve Bullock, Attorney General of Montana, et...	11-1179	Roberts Court	Jun 25, 2012	https://www.oyez.org/cases/2011/11-1179	...	3.0
1	2011_11-182	2011	567 US _	Arizona v. United States	Arizona et al.	United States	11-182	Roberts Court	Jun 25, 2012	https://www.oyez.org/cases/2011/11-182	...	7.0
2	2011_11-161	2011	566 US _	Armour v. City of Indianapolis	Christine Armour	City of Indianapolis	11-161	Roberts Court	Jun 4, 2012	https://www.oyez.org/cases/2011/11-161	...	2.0

3 rows × 25 columns

In [14]:
```
df_cases_convo.dropna(subset=['text'], inplace=True)
df_cases_convo.shape[0]
```

Out[14]:
```
521
```

In [15]:
```
df_cases_convo.head()
```

Out[15]:

	id	year	citation	title	petitioner	respondent	docket_no	court	decided_date	url	...	win_side_
1	2011_11-182	2011	567 US _	Arizona v. United States	Arizona et al.	United States	11-182	Roberts Court	Jun 25, 2012	https://www.oyez.org/cases/2011/11-182	...	
2	2011_11-161	2011	566 US _	Armour v. City of Indianapolis	Christine Armour	City of Indianapolis	11-161	Roberts Court	Jun 4, 2012	https://www.oyez.org/cases/2011/11-161	...	
3	2011_11-159	2011	566 US _	Astrue v. Capato	Michael J. Astrue, Commissioner of Social Secu...	Karen K. Capato	11-159	Roberts Court	May 21, 2012	https://www.oyez.org/cases/2011/11-159	...	
4	2011_10-1320	2011	566 US _	Blueford v. Arkansas	Alex Blueford	Arkansas	10-1320	Roberts Court	May 24, 2012	https://www.oyez.org/cases/2011/10-1320	...	
6	2011_10-844	2011	566 US _	Caraco Pharmaceutical Laboratories, Ltd. v. No...	Caraco Pharmaceutical Laboratories, Ltd., et al.	Novo Nordisk A/S, et al.	10-844	Roberts Court	Apr 17, 2012	https://www.oyez.org/cases/2011/10-844	...	

5 rows × 25 columns

Data Preprocessing

In [16]:
```
# Cleaning the text
def preprocess_text(text):
    text = text.lower() # Lowercase the text
    text = re.sub('[^a-z]+', ' ', text)  # Remove special characters and numbers
    text = re.sub(r'\b\w{1,3}\b', '', text) # Remove words with length less than 3
    words = nltk.word_tokenize(text) # Tokenize the text
    stop_words = set(stopwords.words('english')) # Remove stopwords
    words = [word for word in words if word not in stop_words]
    #lemmatizer = WordNetLemmatizer() # Lemmatize the words comment because slow
    #words = [lemmatizer.lemmatize(word) for word in words]
    stemmer = PorterStemmer() # Stem the words
    words = [stemmer.stem(word) for word in words]
    text = ' '.join(words) # Reconstruct the text

    return text
```

In [17]:
```
text = df_cases_convo.loc[:,['text','win_side']]
text.head(3)
```

Out[17]:

	text	win_side
1	We'll hear argument this morning in Case 11-18...	0.0
2	We will hear argument this morning in case 11-...	0.0
3	We will hear argument first this morning in Ca...	1.0

In [189…
```
text.to_csv('text.csv', index=False)
```

In [18]:
```
text['text'] = text['text'].apply(preprocess_text) #apply preprocess
text.head(1)
#text.to_csv('text_clean.csv', index=False)
```

Out[18]:

	text	win_side
1	hear argument morn case arizona unit state cle...	0.0

In [6]:
```
text = pd.read_csv('text_clean.csv')
text.head(1)
```

Out[6]:

	text	win_side
0	hear argument morn case arizona unit state cle...	0.0

Baseline

In [19]:
```
# Calculate The Baseline for Accuracy, Precision, Recall, F1
accuracy = df_cases['win_side'].value_counts()[1]/df_cases['win_side'].shape[0]
print('Accuracy: ', accuracy)
```

```
Accuracy:  0.6655574043261231
```

Model Selection and Vectorize

In [21]:
```
from sklearn.linear_model import LogisticRegression, Perceptron, SGDClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import LinearSVC
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, f1_score, precision_score, recall_score


def Classifier(X_train, X_test, y_train, y_test):

    # Train and evaluate the classifiers
    classifiers = {
        "Logistic Regression": LogisticRegression(max_iter=1000),
        "Naive Bayes": MultinomialNB(),
        "Linear SVC": LinearSVC(),
        "Random Forest": RandomForestClassifier(),
        "Perceptron": Perceptron(),
    }

    results = []

    for classifier_name, classifier in classifiers.items():

        # Train the classifier
        classifier.fit(X_train, y_train)
```

```
        # Make predictions on the test set
        y_pred = classifier.predict(X_test)

        # Add the scores to the results dictionary
        results.append({
            'classifier': classifier_name,
            'accuracy': accuracy_score(y_test, y_pred),
            'f1': f1_score(y_test, y_pred),
            'precision': precision_score(y_test, y_pred),
            'recall': recall_score(y_test, y_pred)
        })
    return pd.DataFrame(results)
```

In [24]:
```
def Vectorize(vectorizer, X, y):
    X = vectorizer.fit_transform(X)
    y = y
    return X, y
```

Run Model

In [169…
```
# Vectorize the text using TF-IDF
vectorizer = TfidfVectorizer(min_df=5, max_df=0.7)
X, y = Vectorize(vectorizer, text['text'], text['win_side'])
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
Classifier(X_train, X_test, y_train, y_test)
```

Out[169]:

	classifier	accuracy	f1	precision	recall
0	Logistic Regression	0.704762	0.822857	0.727273	0.947368
1	Naive Bayes	0.723810	0.839779	0.723810	1.000000
2	Linear SVC	0.628571	0.745098	0.740260	0.750000
3	Random Forest	0.714286	0.831461	0.725490	0.973684
4	Perceptron	0.628571	0.731034	0.768116	0.697368

In [161…
```
# Vectorize the text using CountVectorizer
vectorizer = CountVectorizer(min_df=5, max_df=0.8)
X, y = Vectorize(vectorizer, text['text'], text['win_side'])
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
Classifier(X_train, X_test, y_train, y_test)
```

Out[161]:

	classifier	accuracy	f1	precision	recall
0	Logistic Regression	0.580952	0.671642	0.633803	0.714286
1	Naive Bayes	0.571429	0.634146	0.650000	0.619048
2	Linear SVC	0.590476	0.661417	0.656250	0.666667
3	Random Forest	0.590476	0.742515	0.596154	0.984127
4	Perceptron	0.571429	0.651163	0.636364	0.666667

In [28]:
```
# USING IMBLEARN
from imblearn.over_sampling import RandomOverSampler
from imblearn.under_sampling import RandomUnderSampler

vectorizer = TfidfVectorizer(min_df=5, max_df=0.8)
X, y = Vectorize(vectorizer, text['text'], text['win_side'])
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)


# Resample the training data
print('--OVERSAMPLING--')
ros = RandomOverSampler(random_state=0)
X_train_resampled, y_train_resampled = ros.fit_resample(X_train, y_train)
classifier = Classifier(X_train_resampled, X_test, y_train_resampled, y_test)
print(classifier)

print('--UNDERSAMPLING--')
ros = RandomUnderSampler(random_state=0)
X_train_resampled, y_train_resampled = ros.fit_resample(X_train, y_train)
classifier = Classifier(X_train_resampled, X_test, y_train_resampled, y_test)
print(classifier)
```

```
--OVERSAMPLING--
            classifier  accuracy        f1  precision    recall
0  Logistic Regression  0.580952  0.661538   0.641791  0.682540
1          Naive Bayes  0.542857  0.606557   0.627119  0.587302
2           Linear SVC  0.533333  0.637037   0.597222  0.682540
3        Random Forest  0.580952  0.721519   0.600000  0.904762
4           Perceptron  0.580952  0.666667   0.637681  0.698413
--UNDERSAMPLING--
            classifier  accuracy        f1  precision    recall
0  Logistic Regression  0.609524  0.649573   0.703704  0.603175
1          Naive Bayes  0.561905  0.616667   0.649123  0.587302
2           Linear SVC  0.619048  0.636364   0.744681  0.555556
3        Random Forest  0.514286  0.495050   0.657895  0.396825
4           Perceptron  0.600000  0.625000   0.714286  0.555556
```

Using Over/Undersampling not help much :-(

NEXT! Try using pretrained model from transformers.

https://huggingface.co/models?pipeline_tag=text-classification&sort=downloads

In [2]:

```python
import torch
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
# from transformers import BertTokenizer, BertForSequenceClassification, Trainer, TrainingArguments
from transformers import DistilBertTokenizer, DistilBertForSequenceClassification, Trainer, TrainingArguments


# Load your data
data = pd.read_csv('text_clean.csv')

# Split your data into training and testing sets
train_texts, test_texts, train_labels, test_labels = train_test_split(data['text'], data['win_side'], test_size=0.2, stratify=data

# Initialize the BERT tokenizer
# tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
tokenizer = DistilBertTokenizer.from_pretrained("distilbert-base-uncased")


# Tokenize the text data
train_encodings = tokenizer(train_texts.tolist(), truncation=True, padding=True)
test_encodings = tokenizer(test_texts.tolist(), truncation=True, padding=True)

# Create PyTorch Class from dataset
class SCOTUSDataset(torch.utils.data.Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = labels

    def __getitem__(self, idx):
        item = {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}
        item['labels'] = torch.tensor(self.labels[idx]).long()
        return item

    def __len__(self):
        return len(self.labels)

train_dataset = SCOTUSDataset(train_encodings, train_labels.tolist())
test_dataset = SCOTUSDataset(test_encodings, test_labels.tolist())

# Initialize the BERT model for sequence classification
# model = BertForSequenceClassification.from_pretrained('bert-base-uncased', num_labels=2)
# https://huggingface.co/distilbert-base-uncased-finetuned-sst-2-english
model = DistilBertForSequenceClassification.from_pretrained("distilbert-base-uncased")

# Set up training arguments
training_args = TrainingArguments(
    output_dir='./results',
    num_train_epochs=3,
    per_device_train_batch_size=2, #changed to 2 because the GPU
    per_device_eval_batch_size=4,
    warmup_steps=500,
    weight_decay=0.01,
    logging_dir='./logs',
    logging_steps=100,
    evaluation_strategy="steps",
    save_strategy="steps",
    save_steps=1000,
    load_best_model_at_end=True,
)

# Create the Trainer
trainer = Trainer(
    model=model,
```

```
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=test_dataset
)

# Train the model
trainer.train()

# Evaluate the model
predictions = trainer.predict(test_dataset)
predicted_labels = predictions.label_ids
y_pred = (predictions.predictions.argmax(-1)).tolist()

# Calculate performance metrics
accuracy = accuracy_score(test_labels, y_pred)
precision = precision_score(test_labels, y_pred)
recall = recall_score(test_labels, y_pred)
f1 = f1_score(test_labels, y_pred)

print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1-Score: {f1:.4f}")
```

```
Downloading (…)solve/main/vocab.txt:   0%|          | 0.00/232k [00:00<?, ?B/s]
Downloading (…)okenizer_config.json:   0%|          | 0.00/28.0 [00:00<?, ?B/s]
Downloading (…)lve/main/config.json:   0%|          | 0.00/483 [00:00<?, ?B/s]
Downloading pytorch_model.bin:   0%|          | 0.00/268M [00:00<?, ?B/s]
Some weights of the model checkpoint at distilbert-base-uncased were not used when initializing DistilBertForSequenceClassificatio
n: ['vocab_projector.bias', 'vocab_layer_norm.bias', 'vocab_projector.weight', 'vocab_transform.bias', 'vocab_transform.weight',
'vocab_layer_norm.weight']
- This IS expected if you are initializing DistilBertForSequenceClassification from the checkpoint of a model trained on another t
ask or with another architecture (e.g. initializing a BertForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing DistilBertForSequenceClassification from the checkpoint of a model that you expect
to be exactly identical (initializing a BertForSequenceClassification model from a BertForSequenceClassification model).
Some weights of DistilBertForSequenceClassification were not initialized from the model checkpoint at distilbert-base-uncased and
are newly initialized: ['pre_classifier.bias', 'classifier.weight', 'pre_classifier.weight', 'classifier.bias']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
c:\Users\rezar\anaconda3\envs\torch_cuda\lib\site-packages\transformers\optimization.py:391: FutureWarning: This implementation of
AdamW is deprecated and will be removed in a future version. Use the PyTorch implementation torch.optim.AdamW instead, or set `no_
deprecation_warning=True` to disable this warning
  warnings.warn(
  0%|          | 0/624 [00:00<?, ?it/s]
{'loss': 0.6994, 'learning_rate': 1e-05, 'epoch': 0.48}
  0%|          | 0/27 [00:00<?, ?it/s]
{'eval_loss': 0.6757726669311523, 'eval_runtime': 2.6986, 'eval_samples_per_second': 38.909, 'eval_steps_per_second': 10.005, 'epo
ch': 0.48}
{'loss': 0.6696, 'learning_rate': 2e-05, 'epoch': 0.96}
  0%|          | 0/27 [00:00<?, ?it/s]
{'eval_loss': 0.6764814853668213, 'eval_runtime': 2.7597, 'eval_samples_per_second': 38.048, 'eval_steps_per_second': 9.784, 'epoc
h': 0.96}
{'loss': 0.7036, 'learning_rate': 3e-05, 'epoch': 1.44}
  0%|          | 0/27 [00:00<?, ?it/s]
{'eval_loss': 0.6685644388198853, 'eval_runtime': 2.7916, 'eval_samples_per_second': 37.613, 'eval_steps_per_second': 9.672, 'epoc
h': 1.44}
{'loss': 0.7099, 'learning_rate': 4e-05, 'epoch': 1.92}
  0%|          | 0/27 [00:00<?, ?it/s]
{'eval_loss': 0.7087978720664978, 'eval_runtime': 2.7839, 'eval_samples_per_second': 37.716, 'eval_steps_per_second': 9.698, 'epoc
h': 1.92}
{'loss': 0.6922, 'learning_rate': 5e-05, 'epoch': 2.4}
  0%|          | 0/27 [00:00<?, ?it/s]
{'eval_loss': 0.7302305698394775, 'eval_runtime': 2.7669, 'eval_samples_per_second': 37.949, 'eval_steps_per_second': 9.758, 'epoc
h': 2.4}
{'loss': 0.6753, 'learning_rate': 9.67741935483871e-06, 'epoch': 2.88}
  0%|          | 0/27 [00:00<?, ?it/s]
{'eval_loss': 0.6546196341514587, 'eval_runtime': 2.7631, 'eval_samples_per_second': 38.001, 'eval_steps_per_second': 9.772, 'epoc
h': 2.88}
{'train_runtime': 136.1508, 'train_samples_per_second': 9.166, 'train_steps_per_second': 4.583, 'train_loss': 0.6893020134705764,
'epoch': 3.0}
  0%|          | 0/27 [00:00<?, ?it/s]
Accuracy: 0.6381
Precision: 0.6381
Recall: 1.0000
F1-Score: 0.7791
```

Still below the baseline. Should redo the data preparation/or using other method.

In [ ]: