Naive Bayes on Political Text

In this notebook we use Naive Bayes to explore and classify political data. See the `README.md` for full details. You can download the required DB from the shared dropbox or from blackboard

In [85]:
```
import sqlite3
import nltk
import random
import numpy as np
from collections import Counter, defaultdict

# Feel free to include your text patterns functions
#from text_functions_solutions import clean_tokenize, get_patterns
import re
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
nltk.download('punkt')
```

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

Out[85]: `True`

In [86]:
```
convention_db = sqlite3.connect("2020_Conventions.db")
convention_cur = convention_db.cursor()
```

Part 1: Exploratory Naive Bayes

We'll first build a NB model on the convention data itself, as a way to understand what words distinguish between the two parties. This is analogous to what we did in the "Comparing Groups" class work. First, pull in the text for each party and prepare it for use in Naive Bayes.

NOTE FROM KB: First I gather up what the data tables are called in the data. I find that the database contains just one table that is called "conventions." I then produce what the schema is called. To get the text, we can query "text" and "party" for this part of the assignment.

In [87]:
```
# Query to get the list of all tables
tables_query = "SELECT name FROM sqlite_master WHERE type='table';"
convention_cur.execute(tables_query)

# Fetch the tables
tables = convention_cur.fetchall()

print("Tables in the database:")
for table in tables:
    print(table[0])
```

```
# Get the schema of the conventions table
for table in tables:
    print(f"\nSchema of '{table[0]}' table:")
    pragma_query = f"PRAGMA table_info({table[0]});"
    convention_cur.execute(pragma_query)
    columns = convention_cur.fetchall()
    for column in columns:
        print(column)
```

```
Tables in the database:
conventions

Schema of 'conventions' table:
(0, 'party', 'TEXT', 0, None, 0)
(1, 'night', 'INTEGER', 0, None, 0)
(2, 'speaker', 'TEXT', 0, None, 0)
(3, 'speaker_count', 'INTEGER', 0, None, 0)
(4, 'time', 'TEXT', 0, None, 0)
(5, 'text', 'TEXT', 0, None, 0)
(6, 'text_len', 'TEXT', 0, None, 0)
(7, 'file', 'TEXT', 0, None, 0)
```

NOTE FROM KB: Below I write a function that first cleans the text, tokenizes it, and then finally removes stopwords.

In [88]:

```
convention_data = []

# fill this list up with items that are themselves lists. The
# first element in the sublist should be the cleaned and tokenized
# text in a single string. The second element should be the party.

def clean_tokenize(text):
    # Clean the text
    # Lowercase the text
    text = text.lower()
    # Remove non-alphabetic characters and punctuation
    text = re.sub(r'[^a-zA-Z\s]', '', text)

    # Tokenize
    tokens = word_tokenize(text)

    # Remove stopwords
    tokens = [word for word in tokens if word not in stopwords.words('englis

    return ' '.join(tokens)

# Getting text then party from the conventions table
query_results = convention_cur.execute(
                            '''
                            SELECT party, text FROM conventions
                            ''')

for row in query_results :
    cleaned_text = clean_tokenize(row[1])
```

```
    # Append the rows into the convention_data list
    convention_data.append([cleaned_text, row[0]])
```

Let's look at some random entries and see if they look right.

In [111…
```
random.choices(convention_data,k=10)
```

Out[111…

```
[['congratulations youre citizens united states america behalf department h
omeland security honor call fellow americans mr president want commend dedi
cation rule law restoring integrity immigration system thank hosting patrio
tic celebration white house today',
  'Republican'],
 ['im pleased announce vice president joe biden officially nominated democr
atic party candidate president united states',
  'Democratic'],
 ['every generation us fight believe turn jack g proud saw demonstrations g
oing across country',
  'Democratic'],
 ['joe biden selected kamala harris running mate', 'Democratic'],
 ['american history tells us thats darkest moments weve made greatest progr
ess found light dark moment believe poised make great progress find light',
  'Democratic'],
 ['im reelected best yet come thank much took office middle east total chao
s isis rampaging iran rise war afghanistan end sight',
  'Republican'],
 ['im rebecca friedrichs veteran california public school educator im give 
voice americas great teachers voices silenced decades unions claim represen
t us dedicated teachers served within unions spoke defense children parents
scientific fact american values trouble brutalized booed platform barred co
mmittees shouted even spit upon union leaders unions treat devoted teachers
whats even worse agenda control deceives americans children theyve intentio
nally rewritten american history perpetuate division pervert memories ameri
can founders disparage judeochristian virtues theyre lenient discipline pol
icies morphed schools war zones back defunding police abolishing ice unions
collect billions annually unsuspecting teachers push radical agenda classro
oms',
  'Republican'],
 ['gave energy students shes great teacher', 'Democratic'],
 ['team usa indeed take home gold', 'Republican'],
 ['skip content company careers press freelancers blog services transcripti
on captions foreign subtitles translation freelancers contact login return 
transcript library home transcript categories transcripts election transcri
pts classic speech transcripts congressional testimony hearing transcripts 
debate transcripts donald trump transcripts entertainment transcripts finan
cial transcripts interview transcripts political transcripts press conferen
ce transcripts speech transcripts sports transcripts technology transcripts
aug democratic national convention dnc night transcript speeches barack oba
ma kamala harris hillary clinton nancy pelosi rev blog transcripts election
transcripts democratic national convention dnc night transcript speeches ba
rack obama kamala harris hillary clinton nancy pelosi night democratic nati
onal convention dnc august speakers include sen vicepresidential nominee ka
mala harris former sec state democratic nominee hillary clinton house speak
er nancy pelosi sen elizabeth warren former president barack obama read ful
l transcript event transcribe content try rev free save time transcribing c
aptioning subtitling',
  'Democratic']]
```

If that looks good, we now need to make our function to turn these into features. In my solution, I wanted to keep the number of features reasonable, so I only used words that occur at least `word_cutoff` times. Here's the code to test that if you want it.

In [90]:
```
word_cutoff = 5

tokens = [w for t, p in convention_data for w in t.split()]

word_dist = nltk.FreqDist(tokens)

feature_words = set()

for word, count in word_dist.items() :
    if count > word_cutoff :
        feature_words.add(word)

print(f"With a word cutoff of {word_cutoff}, we have {len(feature_words)} as
```

```
With a word cutoff of 5, we have 2327 as features in the model.
```

In [91]:
```
def conv_features(text,fw) :
    """Given some text, this returns a dictionary holding the
       feature words.

       Args:
            * text: a piece of text in a continuous string. Assumes
            text has been cleaned and case folded.
            * fw: the *feature words* that we're considering. A word
            in `text` must be in fw in order to be returned. This
            prevents us from considering very rarely occurring words.

       Returns:
            A dictionary with the words in `text` that appear in `fw`.
            Words are only counted once.
            If `text` were "quick quick brown fox" and `fw` = {'quick','fox'
            then this would return a dictionary of
            {'quick' : True,
             'fox' :    True}

    """

    # Split the text into words
    words = text.split()

    # Create a dictionary for the features
    ret_dict = dict()

    # Check each word in the text
    for word in words:
        if word in fw:
            ret_dict[word] = True

    return ret_dict
```

In [92]:
```
assert(len(feature_words)>0)
assert(conv_features("donald is the president",feature_words)==
       {'donald':True,'president':True})
assert(conv_features("some people in america are citizens",feature_words)==
                    {'people':True,'america':True,"citizens":True})
```

Now we'll build our feature set. Out of curiosity I did a train/test split to see how accurate the classifier was, but we don't strictly need to since this analysis is exploratory.

In [93]:
```
featuresets = [(conv_features(text,feature_words), party) for (text, party)
```

In [94]:
```
random.seed(20220507)
random.shuffle(featuresets)

test_size = 500
```

In [95]:
```
test_set, train_set = featuresets[:test_size], featuresets[test_size:]
classifier = nltk.NaiveBayesClassifier.train(train_set)
print(nltk.classify.accuracy(classifier, test_set))
```

```
0.494
```

In [96]:
```
classifier.show_most_informative_features(25)
```

```
Most Informative Features
                   china = True           Republ : Democr =     25.8 : 1.0
                   votes = True           Democr : Republ =     23.8 : 1.0
             enforcement = True           Republ : Democr =     21.5 : 1.0
                 destroy = True           Republ : Democr =     19.2 : 1.0
                freedoms = True           Republ : Democr =     18.2 : 1.0
                 climate = True           Democr : Republ =     17.8 : 1.0
                supports = True           Republ : Democr =     17.1 : 1.0
                   crime = True           Republ : Democr =     16.1 : 1.0
                   media = True           Republ : Democr =     14.9 : 1.0
                 beliefs = True           Republ : Democr =     13.0 : 1.0
               countries = True           Republ : Democr =     13.0 : 1.0
                 defense = True           Republ : Democr =     13.0 : 1.0
                  defund = True           Republ : Democr =     13.0 : 1.0
                    isis = True           Republ : Democr =     13.0 : 1.0
                 liberal = True           Republ : Democr =     13.0 : 1.0
                religion = True           Republ : Democr =     13.0 : 1.0
                   trade = True           Republ : Democr =     12.7 : 1.0
                    flag = True           Republ : Democr =     12.1 : 1.0
               greatness = True           Republ : Democr =     12.1 : 1.0
                 abraham = True           Republ : Democr =     11.9 : 1.0
                    drug = True           Republ : Democr =     10.9 : 1.0
              department = True           Republ : Democr =     10.9 : 1.0
               destroyed = True           Republ : Democr =     10.9 : 1.0
                   enemy = True           Republ : Democr =     10.9 : 1.0
               amendment = True           Republ : Democr =     10.3 : 1.0
```

Write a little prose here about what you see in the classifier. Anything odd or interesting?

My Observations

KB: One of the most interesting ones is China. I actually think the China issue has become somewhat more bipartisan since the last election cycle. It always seemed to me that Republicans would imply or accuse Democrats of being sympathetic towards communist China and that Democrats would likewise imply or accuse Republicans of being sympathetic towards nationalist, church/state unified Russia. Trump definitely made it a point of his presidency to antagonize China more than previous presidents by pushing the trade war. Therefore, it is not a surprise that it was a major talking point of Republicans in 2020. However, I bet if we re-did this analysis in 2024, the proportion difference would be much narrower. That would be an interesting topic to test.

Something else that stands out is how many more Republican buzzwords there are than Democrat buzzwords. Of the 25 words, only two are Democrat words. This tells me that perhaps the dataset is very biased towards containing Republican content or Republican buzzwords. It could also mean that Republicans simply focus more intensely on certain subjects than Democrats do. However, I doubt this is the case. It would be worthwhile to explore more about the dataset to understand why the Republican words stick out like this.

Part 2: Classifying Congressional Tweets

In this part we apply the classifer we just built to a set of tweets by people running for congress in 2018. These tweets are stored in the database `congressional_data.db`. That DB is funky, so I'll give you the query I used to pull out the tweets. Note that this DB has some big tables and is unindexed, so the query takes a minute or two to run on my machine.

In [97]:
```
cong_db = sqlite3.connect("congressional_data.db")
cong_cur = cong_db.cursor()
```

In [98]:
```
results = cong_cur.execute(
        '''
           SELECT DISTINCT
                  cd.candidate,
                  cd.party,
                  tw.tweet_text
           FROM candidate_data cd
           INNER JOIN tweets tw ON cd.twitter_handle = tw.handle
               AND cd.candidate == tw.candidate
               AND cd.district == tw.district
           WHERE cd.party in ('Republican','Democratic')
               AND tw.tweet_text NOT LIKE '%RT%'
        ''')

results = list(results) # Just to store it, since the query is time consumir
```

In [100…
```
tweet_data = []

# Now fill up tweet_data with sublists like we did on the convention speeche
# Note that this may take a bit of time, since we have a lot of tweets.
for candidate, party, tweet_text in results:
    # Decode the tweet_text from bytes to string
    if isinstance(tweet_text, bytes):
```

```
        tweet_text = tweet_text.decode('utf-8')

    # Preprocess the tweet
    cleaned_text = clean_tokenize(tweet_text)

    # Create a feature set for the tweet
    feature_set = conv_features(cleaned_text, feature_words)

    # Append the tuple (feature set, party) to tweet_data
    tweet_data.append((feature_set, party))
```

There are a lot of tweets here. Let's take a random sample and see how our classifer does. I'm guessing it won't be too great given the performance on the convention speeches...

In [101…
```
random.seed(20201014)

tweet_data_sample = random.choices(tweet_data,k=10)
```

In [107…
```
for tweet, party in tweet_data_sample :

    # Use the classifier to estimate the party
    estimated_party = classifier.classify(tweet)

    print(f"Here's our (cleaned) tweet: {tweet}")
    print(f"Actual party is {party} and our classifer says {estimated_party}
    print("")
```

```
Here's our (cleaned) tweet: {'earlier': True, 'today': True, 'spoke': True,
'house': True, 'floor': True, 'protecting': True, 'health': True, 'care': Tr
ue, 'women': True, 'work': True, 'coast': True}
Actual party is Democratic and our classifer says Republican.

Here's our (cleaned) tweet: {'go': True}
Actual party is Democratic and our classifer says Democratic.

Here's our (cleaned) tweet: {'trump': True, 'thinks': True, 'easy': True, 's
tudents': True, 'burden': True, 'debt': True, 'pay': True, 'student': True}
Actual party is Democratic and our classifer says Republican.

Here's our (cleaned) tweet: {'grateful': True, 'first': True, 'responders':
True, 'rescue': True, 'police': True, 'working': True, 'tirelessly': True,
'keep': True, 'people': True, 'safe': True, 'provide': True, 'help': True,
'putting': True, 'lives': True, 'line': True}
Actual party is Republican and our classifer says Republican.

Here's our (cleaned) tweet: {'lets': True, 'make': True, 'even': True, 'grea
ter': True}
Actual party is Republican and our classifer says Republican.

Here's our (cleaned) tweet: {'im': True, 'scared': True}
Actual party is Democratic and our classifer says Republican.

Here's our (cleaned) tweet: {'new': True, 'city': True, 'glad': True, 'conti
nue': True, 'serve': True}
Actual party is Democratic and our classifer says Republican.

Here's our (cleaned) tweet: {'really': True, 'close': True, 'raised': True,
'toward': True, 'right': True, 'thats': True, 'room': True, 'help': True, 'u
s': True, 'get': True}
Actual party is Democratic and our classifer says Republican.

Here's our (cleaned) tweet: {'today': True, 'plan': True, 'expand': True, 'o
pened': True, 'public': True, 'days': True, 'march': True, 'share': True, 'p
rogram': True, 'trump': True, 'administration': True, 'made': True, 'email':
True, 'mail': True}
Actual party is Democratic and our classifer says Republican.

Here's our (cleaned) tweet: {'celebrated': True, 'years': True, 'commitmen
t': True, 'community': True, 'leaders': True, 'last': True, 'nights': True}
Actual party is Democratic and our classifer says Republican.
```

Now that we've looked at it some, let's score a bunch and see how we're doing.

In [108…

```python
# dictionary of counts by actual party and estimated party.
# first key is actual, second is estimated
parties = ['Republican','Democratic']
results = defaultdict(lambda: defaultdict(int))

for p in parties :
    for p1 in parties :
        results[p][p1] = 0
```

```
num_to_score = 10000
random.shuffle(tweet_data)

for idx, tp in enumerate(tweet_data) :
    tweet, party = tp
    # Now do the same thing as above, but we store the results rather
    # than printing them.

    # get the estimated party
    estimated_party = classifier.classify(tweet)

    results[party][estimated_party] += 1

    if idx > num_to_score :
        break
```

In [109…

```
results
```

Out[109…

```
defaultdict(<function __main__.<lambda>()>,
            {'Republican': defaultdict(int,
                         {'Republican': 3748, 'Democratic': 530}),
             'Democratic': defaultdict(int,
                         {'Republican': 4855, 'Democratic': 869})})
```

Reflections

Write a little about what you see in the results

KB: Like I talked about above, the dataset seems to have a heavy Republican bias, where most of the tweets are Republican. The model does a good job of classifying the Republican the Republican ones as Republican but a poor job of identifying the Democrat ones as Democrat. One way of improving the model would be to use a more balanced training set that skews towards containing a lower proportion of Republican tweets to Democrat tweets.