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.

In [3]:
```python
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 import clean_tokenize, get_patterns
```

In [6]:
```python
# other imports

import emoji
nltk.download('stopwords')
from nltk.corpus import stopwords
from string import punctuation
import re
import random
```

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

In [60]:
```python
# I was unsure where to find the text_functions, I did see one item on your GitHub but it looked like an
# assignment for a different class. I utilized the funcitons from the previous module too.

# text_functions_solutions

# Some punctuation variations
punctuation = set(punctuation) # speeds up comparison
tw_punct = punctuation - {"#"}

# Stopwords
sw = stopwords.words("english")

# Two useful regex
whitespace_pattern = re.compile(r"\s+")
hashtag_pattern = re.compile(r"^#[0-9a-zA-Z]+")

# It's handy to have a full set of emojis
all_language_emojis = set()

for country in emoji.UNICODE_EMOJI :
    for em in emoji.UNICODE_EMOJI[country] :
        all_language_emojis.add(em)


def is_emoji(s):
    return(s in all_language_emojis)

def contains_emoji(s):
    s = str(s)
    emojis = [ch for ch in s if is_emoji(ch)]
    return(len(emojis) > 0)


def remove_stop(tokens):
    stopwords = set(nltk.corpus.stopwords.words('english'))
    return [t for t in tokens if t.lower() not in stopwords]

def remove_punctuation(tokens):
    return [t for t in tokens if t not in set(punctuation)]

def tokenize(text):
    tokenizer = nltk.RegexpTokenizer(r"\w+")
    tokens = tokenizer.tokenize(text)
    return [t for t in tokens if t.isalpha() and len(t)>1]

def prepare(text, pipeline) :
    tokens = str(text)

    for transform in pipeline :
        tokens = transform(tokens)

    return(tokens)


my_pipeline = [str.lower, tokenize, remove_punctuation, remove_stop]
```

In [61]:
```python
#pwd
```

In [32]:
```python
convention_db = sqlite3.connect("2020_Conventions(1).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.

In [33]:
```python
# check table names in db

convention_cur.execute("SELECT name FROM sqlite_master WHERE type='table';")
table_names = [info[0] for info in convention_cur.fetchall()]
print(table_names)
```

```
['conventions']
```

In [34]:
```python
for table_name in table_names:
    result = convention_cur.execute("PRAGMA table_info('%s')" % table_name).fetchall()
    column_names = list(zip(*result))[1]
    print(f"Columns in {table_name}: {column_names}")
```

```
Columns in conventions: ('party', 'night', 'speaker', 'speaker_count', 'time', 'text', 'text_len', 'file')
```

In [35]:
```python
from sqlite3.dbapi2 import Row
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.

query_results = convention_cur.execute(
                            """
                            SELECT lower(text),party
                            FROM conventions
                            """)

for row in query_results :
  text = " ".join(prepare(row[0],my_pipeline))
  convention_data.append([text,row[1]])
    # store the results in convention_data
    # pass # remove this


# Section Complete
```

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

In [36]:
```python
random.choices(convention_data,k=10)
```

Out[36]:
```
[['waited years file rapidly approved medical turned right around got disability thinking going several year
s worth waiting hear',
  'Republican'],
 ['growing young child slovenia communist rule time always heard amazing place called america land stood fre
edom opportunity grew older became goal move united states follow dream working fashion industry parents wor
k hard ensure family could live prosper america also contribute nation allows people arrive dream make reali
ty want take moment thank mother father done family standing today',
  'Republican'],
 ['man know president need four years picks toughest fights tackles complex problems stood stand honor women
empowered future children cherish thank god bless always',
  'Republican'],
 ['americans dead millions jobs gone well top taken ever worst impulses unleashed proud reputation around wo
rld badly diminished democratic institutions threatened like never know times polarized already made mind ma
ybe still sure candidate vote whether vote maybe tired direction headed see better path yet know enough pers
on wants lead us',
  'Democratic'],
 ['police coming call democrat run cities already defunded disbanded blaming best allowing society worst sto
ry write hollywood lights even stay california anymore state cannot keep power running people send junior se
nator vice president used write fiction nightmares becoming real cops killed children shot democrat conventi
on say vote trump stop appeasement never winning strategy settle violence neighborhoods border settle decade
s bad decisions basement dwelling joe biden settle continent know frontier horizon even stars belonged us do
nald trump like builders visionary built mind even powerful brick mortar holds together',
  'Republican'],
 ['joe biden decent man long history public service america', 'Democratic'],
 ['hour need literally helping us big way edge life death stuff forever thankful',
  'Republican'],
 ['abraham lincoln famously said america never destroyed outside falter lose freedoms destroy words spoken y
ears ago never relevant choose right path maintain unique freedoms boundless opportunities make country grea
test history world remain beacon hope around world fighting oppression communism tyranny choice know promise
america lived member trump family woman knows like work blue collar jobs serve customers tips aspire rise lo
ok son luke daughter carolina wonder sort country leaving future generations recent months seen weak spinele
ss politicians seek control great american cities violent mobs',
  'Republican'],
 ['russians offered bounties us soldiers shocked read president even asked vladimir putin un american',
  'Democratic'],
 ['everything family always done everything family', '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 [38]:
```python
# I ran into an Assertion Error below because there are other frequent words coming up if the word_cutoff =
# As a solution word_cutoff is increased to 50 to pass the assertion.

word_cutoff = 50

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 features in the model.")

# Section Complete
```

```
With a word cutoff of 50, we have 302 as features in the model.
```

In [39]:
```python
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','jumps'},
            then this would return a dictionary of
            {'quick' : True,
             'fox' :    True}

    """
    ret_dict = dict()
    for word in fw:
      if word in text:
        ret_dict[word]=True

    return(ret_dict)


# Section Complete
```

In [40]:
```python
conv_features("donald is the president",feature_words)

# Section Complete
```

Out[40]:
```
{'donald': True, 'president': True}
```

In [41]:
```python
# I received the Assertion Error, outlined above, because there are other frequent words coming up if the wo
# As a solution the word_cutoff is increased to 50 to pass the assertion here

assert(len(feature_words)>0)
assert(conv_features("donald is the president",feature_words)==
       {'donald':True,'president':True})
assert(conv_features("people are american in america",feature_words)==
                     {'america':True,'american':True,"people":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 [42]:
```python
featuresets = [(conv_features(text,feature_words), party) for (text, party) in convention_data]
```

In [43]:
```python
random.seed(20220507)
random.shuffle(featuresets)

test_size = 500
```

In [44]:
```python
test_set, train_set = featuresets[:test_size], featuresets[test_size:]
classifier = nltk.NaiveBayesClassifier.train(train_set)
print(nltk.classify.accuracy(classifier, test_set))
```

```
0.448
```

In [45]:
```python
classifier.show_most_informative_features(25)
```

```
Most Informative Features
                   china = True           Republ : Democr =     27.7 : 1.0
                   votes = True           Democr : Republ =     23.8 : 1.0
                 climate = True           Democr : Republ =     17.8 : 1.0
                   media = True           Republ : Democr =     11.4 : 1.0
                 freedom = True           Republ : Democr =      9.7 : 1.0
                greatest = True           Republ : Democr =      8.4 : 1.0
                  choice = True           Republ : Democr =      7.6 : 1.0
                  heroes = True           Republ : Democr =      7.1 : 1.0
               democracy = True           Democr : Republ =      6.4 : 1.0
                  bernie = True           Democr : Republ =      6.2 : 1.0
                  police = True           Republ : Democr =      6.2 : 1.0
                   bless = True           Republ : Democr =      5.4 : 1.0
                    free = True           Republ : Democr =      5.1 : 1.0
                  kamala = True           Democr : Republ =      4.8 : 1.0
             opportunity = True           Republ : Democr =      4.7 : 1.0
                 sanders = True           Democr : Republ =      4.6 : 1.0
                   dream = True           Republ : Democr =      4.5 : 1.0
               democrats = True           Republ : Democr =      4.5 : 1.0
                   trump = True           Republ : Democr =      4.3 : 1.0
              government = True           Republ : Democr =      4.2 : 1.0
                    left = True           Republ : Democr =      4.2 : 1.0
                  called = True           Republ : Democr =      4.1 : 1.0
                    four = True           Republ : Democr =      4.0 : 1.0
                    took = True           Republ : Democr =      3.9 : 1.0
                    cast = True           Democr : Republ =      3.9 : 1.0
```

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

My Observations

It seems like there are word preferences when we look at Republican candidates vs Democratic candidates. An interesting couple of observations are the use of the word China is much higher among Republicans while Climate is a much higher used term than for Republican candidates. It goes to show party values and priorities.

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 [46]:
```python
cong_db = sqlite3.connect("congressional_data.db")
cong_cur = cong_db.cursor()
```

In [47]:
```python
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 consuming
```

In [48]:
```python
# looking at the results to make sure it is working properly

results[0]

# Section Complete
```

Out[48]:
```
('Mo Brooks',
 'Republican',
 b'"Brooks Joins Alabama Delegation in Voting Against Flawed Funding Bill" http://t.co/3CwjIWYsNq')
```

In [49]:
```python
tweet_data = []

# Now fill up tweet_data with sublists like we did on the convention speeches.
# Note that this may take a bit of time, since we have a lot of tweets.

for row in results :
  text = " ".join(prepare(row[2],my_pipeline))
  tweet_data.append([text,row[1]])
```

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 [50]:
```python
random.seed(20201014)

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

In [51]:
```python
for tweet, party in tweet_data_sample :
    # Fill in the right-hand side above with code that estimates the actual party
    featuresets = conv_features(tweet,feature_words)
    estimated_party = classifier.classify(featuresets)
    print(f"Here's our (cleaned) tweet: {tweet}")
    print(f"Actual party is {party} and our classifer says {estimated_party}.")
    print("")


# Section Complete
```

```
Here's our (cleaned) tweet: earlier today spoke house floor abt protecting health care women praised ppmarmo
nte work central coast https co
Actual party is Democratic and our classifer says Republican.

Here's our (cleaned) tweet: go tribe rallytogether https co
Actual party is Democratic and our classifer says Democratic.

Here's our (cleaned) tweet: apparently trump thinks easy students overwhelmed crushing burden debt pay stude
nt loans trumpbudget https co
Actual party is Democratic and our classifer says Republican.

Here's our (cleaned) tweet: grateful first responders rescue personnel firefighters police volunteers workin
g tirelessly keep people safe provide much needed help putting lives line nhttps co
Actual party is Republican and our classifer says Republican.

Here's our (cleaned) tweet: let make even greater kag xba https co
Actual party is Republican and our classifer says Republican.

Here's our (cleaned) tweet: cavs tie series repbarbaralee scared roadtovictory
Actual party is Democratic and our classifer says Democratic.

Here's our (cleaned) tweet: congrats belliottsd new gig sd city hall glad continue serve https co
Actual party is Democratic and our classifer says Republican.

Here's our (cleaned) tweet: really close raised toward match right whoot non math majors room help us get ht
tps co https co qsdqkypsmc
Actual party is Democratic and our classifer says Republican.

Here's our (cleaned) tweet: today comment period potus plan expand offshore drilling opened public days marc
h share oppose proposed program directly trump administration comments made email mail https co baaymejxqn
Actual party is Democratic and our classifer says Republican.

Here's our (cleaned) tweet: celebrated icseastla years eastside commitment amp saluted community leaders las
t night awards dinner https co
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 [52]:
```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
    featuresets = conv_features(tweet,feature_words)

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

    results[party][estimated_party] += 1

    if idx > num_to_score :
        break


# Section Complete
```

In [53]:
```python
results
```

Out[53]:
```
defaultdict(<function __main__.<lambda>()>,
            {'Republican': defaultdict(int,
                         {'Republican': 3755, 'Democratic': 523}),
             'Democratic': defaultdict(int,
                         {'Republican': 5014, 'Democratic': 710})})
```

In [54]:
```python
# Accuracy
(results['Democratic']['Democratic']+results['Republican']['Republican'])/num_to_score

# Section Complete
```

Out[54]:
```
0.4465
```

In [55]:
```python
# Precision of classifying Democratic
(results['Democratic']['Democratic'])/(results['Democratic']['Democratic']+results['Republican']['Democratic

# Section Complete
```

Out[55]:
```
0.5758313057583131
```

In [56]:
```python
# Recall of classifying Democratic
(results['Democratic']['Democratic'])/(results['Democratic']['Democratic']+results['Democratic']['Republican

# Section Complete
```

Out[56]:
```
0.12403913347309574
```

In [57]:
```python
# Precision of classifying Republican
(results['Republican']['Republican'])/(results['Republican']['Republican']+results['Democratic']['Republican

# Section Complete
```

Out[57]:
```
0.42821302314973203
```

In [58]:
```python
# Recall of classifying Republican
(results['Republican']['Republican'])/(results['Republican']['Republican']+results['Republican']['Democratic

# Section Complete
```

Out[58]:
```
0.8777466105656849
```

In [59]:
```python
# Final Ratio Results:

total_cases = results['Republican']['Republican']+results['Republican']['Democratic']+results['Democratic'][
total_republican = results['Republican']['Republican']+results['Republican']['Democratic']
total_democratic = results['Democratic']['Democratic']+results['Democratic']['Republican']
print("Democratic Ratio:",total_democratic/total_cases)
print("Republican Ratio:",total_republican/total_cases)

# Section Complete
```

```
Democratic Ratio: 0.5722855428914218
Republican Ratio: 0.4277144571085783
```

Reflections

When looking at the final results, I broke down to show the precision, recall, and accuracy for our model above. The final ratio results show that the model does better when classifying Republicans vs classifying Democrats. The model does have a pretty high false positive rate. This is an area that we would need to explore if the cost for misclassifying is too high.

In [ ]: