In [39]:

```

```

Hoori Javadnia , Assignment Module 4

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 [ ]:

```
from collections import Counter, defaultdict
import nltk
from nltk.corpus import stopwords
import numpy as np
import pandas as pd
import random
import re
import sqlite3
```

In [40]:

```
stop_words = stopwords.words("english") # Stop words
```

In [41]:

```
# Database read
convention_2020_db = ("/Users/javadniahoori/Downloads/509/4/assignment/2020_Conventions.db")

congressional_db = ("/Users/javadniahoori/Downloads/509/4/assignment/congressional_data.db")
```

In [ ]:

```

```

In [42]:

```
convention_db = sqlite3.connect(convention_2020_db)
convention_cur = convention_db.cursor()
```

In [63]:

```
#see what in the table
sql_query = """SELECT name FROM sqlite_master
  WHERE type='table';"""
convention_cur.execute(sql_query)
```

```
print("List of tables\n")
print(convention_cur.fetchall())
```

```
List of tables

[('conventions',)]
```

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 [60]:

```
# cleaan the data below
def preprocess_text(text):
    text = str(text)
    # Lowercase all text
    text = str.lower(text)


    re_whitespaces = re.compile(r"\s+") # Remove white spaces
    text = re_whitespaces.sub(" ", text)


    re_misc = re.compile(r"[&#:()<>{}\[\]\\,\.]")# Remove noise
    text = re_misc.sub(" ", text)
    # Remove extra additional noise
    re_noise = re.compile(r"\s[\W]\s")
    text = re_noise.sub("", text)


    text = text.strip().replace("  ", " ")


    tokens = text.split(" ") # Split on white spaces
    # Drop empty tokens
    tokens = [x for x in tokens if x != ""]


    tokens = [x for x in tokens if not x in stop_words] # Remove stop word

    # Join tokens into a single string
    tokens = " ".join(tokens)
```

```
    return tokens
```

In [61]:

```
convention_data = []
# Result is a list of tuples (raw_text, party)
query_results = convention_cur.execute(
    '''
    SELECT text, party
    FROM conventions
    '''
)
for row in query_results:
    raw_text = row[0]
    party = row[1]

    # Pre-process tokens
    tokens = preprocess_text(raw_text)

    # Create sublist of tokens and political party
    tokens_and_party = [tokens, party]
    convention_data.append(tokens_and_party)
```

In [45]:

```
random.choices(convention_data, k = 10)
```

Out[45]:

```
[['foreign prince', 'Republican'],
 ['reproductive justice', 'Democratic'],
 ['mission fight future equal ideals founders hopes children sacrifices veterans brave men women uniform famili
es',
  'Democratic'],
 ['black americans standing native land probably represent oregon dual viruses covid-19 racism laid bare equal
healthcare access deaths communities color',
  'Democratic'],
 ['joe’s purpose always driven forward strength unstoppable faith unshakable it’s politicians political parties
even it’s providence god faith us yes many classrooms quiet right playgrounds still listen closely hear sparks
change air across country educators parents first responders americans walks life putting shoulders back fighti
ng haven’t given need leadership worthy nation worthy honest leadership bring us back together recover pandemic
prepare whatever else next dr',
  'Democratic'],
 ['he’ll love heart', 'Democratic'],
 ['rhode island ocean state restaurant fishing industry decimated pandemic lucky governor gina raimondo whose p
rogram lets fishermen sell catches directly public state appetizer calamari available 50 states calamari comeba
ck state rhode island casts 1 vote bernie sanders 34 votes next president joe biden',
  'Democratic'],
```

```
 ['knows it’s like send child war', 'Democratic'],
 ['america', 'Democratic'],
 ['trillions dollars repatriated back united states sitting foreign lands far long america became envy world re
newed strength came leverage president demanded allies pay fair share defense western world father rebuilt migh
ty american military adding new jets aircraft carriers increased wages incredible men women uniform expanded mi
litary defense budget $721 billion per year america longer weak eye enemy moment president trump ordered specia
l forces kill deadliest terrorists planet day mighty moab dropped insurgent camps day america took stance never
defeated enemy al-baghdadi soleimani dead issue issue economy wall military trade deals tax cuts supreme court
justices va hospitals prescription drugs school choice right try moving embassy jerusalem peace middle east nev
er-ending wars finally ended promises made promises first time kept',
  'Republican']]
```

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 [46]:

```python
word_cutoff = 5

# List of all tokens
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.")
```

```
With a word cutoff of 5, we have 2400 as features in the model.
```

In [62]:

```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}

    """
    clean_text = text
    words = clean_text.split(" ")

    unique_words = set(words)


    words_in_fw = [w for w in unique_words if w in fw]

    ret_dict = dict.fromkeys(words_in_fw, True)# Key = word, value = True

    return(ret_dict)
```

In [48]:

```
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 [49]:

```
featuresets = [(
    conv_features(text, feature_words),
        party) for (text, party) in convention_data]
```

In [50]:

```
random.seed(20220507)
random.shuffle(featuresets)

test_size = 500
```

In [51]:

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

```
0.502
```

In [52]:

```
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
                    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
                 abraham = True           Republ : Democr =     11.9 : 1.0
                  defund = True           Republ : Democr =     11.9 : 1.0
               greatness = True           Republ : Democr =     11.5 : 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?

The accuracy is 50%. between top 25 features, only 2 indicate Democrats and the rest suggest The Republican party had a higher word presence compared to the Democratic party. This suggests that Republicans tend to use more unique words, contributing to their greater prominence in text.

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 [53]:

```
cong_db = sqlite3.connect(congressional_db)
cong_cur = cong_db.cursor()
```

In [54]:

```
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 [55]:

```
tweet_data = []
for row in results:
    raw_text = row[2].decode("utf-8")
    party = row[1]

    # Pre-process tokens
    tokens = preprocess_text(raw_text)

    # Create sublist of tokens and political party
    tokens_and_party = [tokens, party]

    # store the results in convention_data
    tweet_data.append(tokens_and_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 [56]:

```
random.seed(20201014)

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

In [57]:

```
for tweet, party in tweet_data_sample:
    # Convert tweet to list
    tweet = "".join(tweet)
    # Convert to input dictionary
    tweet_dict = conv_features(tweet, feature_words)
    # Predict party
    estimated_party = classifier.classify(tweet_dict)

    # Fill in the right-hand side above with code that estimates the actual party
    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 today spoke house floor abt protecting health care women praised @ppmarmont
e work central coast https //t co/wqgtrzt7vv
Actual party is Democratic and our classifer says Republican.

Here's our (cleaned) tweet: go tribe! rallytogether https //t co/0nxutfl9l5
Actual party is Democratic and our classifer says Democratic.

Here's our (cleaned) tweet: apparently trump thinks easy students overwhelmed crushing burden debt pay student
loans trumpbudget https //t co/ckyqo5t0qh
Actual party is Democratic and our classifer says Republican.

Here's our (cleaned) tweet: we’re grateful first responders rescue personnel firefighters police volunteers wor
king tirelessly keep people safe provide much-needed help putting lives line https //t co/ezpv0vmiz3
Actual party is Republican and our classifer says Republican.

Here's our (cleaned) tweet: let’s make even greater !! kag 🇺🇸 https //t co/y9qozd5l2z
Actual party is Republican and our classifer says Republican.

Here's our (cleaned) tweet: 1hr @cavs tie series 2-2 i'm allin216 @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 //t co/fkvmw3c
```

```
qdi
Actual party is Democratic and our classifer says Republican.

Here's our (cleaned) tweet: really close $3500 raised toward match right whoot!! that’s $7000 non-math majors r
oom help us get https //t co/tu34c472sd https //t co/qsdqkypsmc
Actual party is Democratic and our classifer says Republican.

Here's our (cleaned) tweet: today comment period @potus’s plan expand offshore drilling opened public 60 days m
arch 9 share oppose proposed program directly trump administration comments made email mail https //t co/baayme
jxqn
Actual party is Democratic and our classifer says Republican.

Here's our (cleaned) tweet: celebrated @icseastla’s 22 years eastside commitment amp; saluted community leaders
last night’s awards dinner! https //t co/7v7gh8givb
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 [58]:

```
# 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.
    tweet = "".join(tweet)
    # Convert to input dictionary
    tweet_dict = conv_features(tweet, feature_words)
    # Predicted party
    estimated_party = classifier.classify(tweet_dict)

    results[party][estimated_party] += 1
```

```python
    if idx > num_to_score:
        break
```

In [59]:

```python
results
```

Out[59]:

```
defaultdict(<function __main__.<lambda>()>,
            {'Republican': defaultdict(int,
                         {'Republican': 3654, 'Democratic': 624}),
             'Democratic': defaultdict(int,
                         {'Republican': 4760, 'Democratic': 964})})
```

Reflections

Write a little about what you see in the results

Out of 10,002 predicted candidates, 4,618 were correct.Seems like we have a bias towards republican. .

In [ ]:

In [ ]: