# 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 [1]:
```
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
from nltk.corpus import stopwords
import re
import pandas as pd
import matplotlib.pyplot as plt
```

In [2]:
```
sw = set(stopwords.words('english'))
```

In [3]:
```
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.

In [4]:
```
def clean_text(text, is_tweet=False):

    text = text.lower()

    if is_tweet:
        # Keep the hashtag and tagging so it's easier to drop later
        text = re.sub(r"[^a-zA-Z0-9 #@]", '', text)
    else:
        # Only keep alpha-numeric values + space
        text = re.sub(r"[^a-zA-Z0-9 ]", '', text)

    # 2 or more space into one
    text = re.sub('\s+', ' ', text)

    # tokenize
    text = text.split(' ')

    # remove stopwords from tokens
    text = [x for x in text if x not in sw]

    if is_tweet:
        # each tweet contains url/tagging at the end of the tweet
```

```
        # drop them if so
        text = [x for x in text if 'http' not in x and '@' not in x]

    return ' '.join(text)
```

In [5]:

```
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 * from conventions
                            ''')


cols = [x[0] for x in query_results.description]

print('Columns are {}'.format(cols))

for row in query_results:

    text = row[-3]
    text = clean_text(text)

    convention_data.append([text, row[0]])

print('Total length of {}'.format(len(convention_data)))
```

```
Columns are ['party', 'night', 'speaker', 'speaker_count', 'time', 'text', 'text
_len', 'file']
Total length of 2541
```

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

In [6]:

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

Out[6]:

```
[['vote america thats way get joe biden person ensure get lives back normal',
  'Democratic'],
 ['name stacia brightmon single mother two boys work houston texas sb engineers
constructors warehouse receiving clerk serving country marine corps worked vario
us jobs thought needed degree order career put significant debt earn bachelors d
egree finance despite degree efforts one point boys even found homeless laid yet
received email texas workforce commission sbs women construction earn learn appr
enticeship program thought id give try bit skeptical first said hey theyre willi
ng take chance willing take chance stacia brightmon 2223 start program making 16
00 hour trained become pipe fitter helper looking back wasnt easy boy worth felt
empowered learning new skills able support family worked hard got promoted even
got raises later learned president donald trump ivanka trump behind scenes make
sure people like chance rise succeed president trump started national council am
erican worker american workforce policy advisory board',
  'Republican'],
 ['heavy mean ive left emergency department tears lot colleagues people exhauste
d looking tsunami thats coming winter covid influenza think us wondering system
going collapse people come broken arms come heart attacks strokes appendicitis g
oing',
  'Democratic'],
 ['kayla mostly held 12 12 cell solitary confinement cold dirty isis terrorist s
hine bright lights face shaved head beat tortured leader isis abu bakr albaghdad
```

```
i raped repeatedly 18 months endured endure agonizing back forth us obama admini
stration isis put faith government government let us president obama refused mee
t us isis already beheaded americans day never heard joe biden instead obama adm
inistration hid behind policy much felt hopeless kept us negotiating save kaylas
life administration showed concern terrorists guantanamo american hostages syri
a',
  'Republican'],
 ['support defend', 'Republican'],
 ['speak truths act faith ask place us believe country us stand together better
future already see doctors nurses home healthcare workers frontline workers risk
ing lives save people theyve never met',
  'Democratic'],
 ['president trump continues place strong women significant positions throughout
administration campaign far president us history',
  'Republican'],
 ['finally rock getting hours weeks months unspeakable pain unending surgeries a
nchor relearned walk helping every step every stumble military spouses hold fami
lies together praying loved ones safety wherever theyre deployed serving caregiv
ers disabled service members picking pieces starting whenever next tour next war
arises joe biden understands sacrifices hes made son beau deployed iraq burden a
lso shouldered family joe knows fear military families live hes felt dread never
knowing deployed loved one safe understands bravery muster strength every hour e
very day beau overseas thats kind leader service members deserve',
  'Democratic'],
 ['national importance', 'Republican'],
 ['followed rules obeyed laws learned history embraced values proved men women h
ighest integrity easy went lot appreciate us today youve earned prized treasured
cherished priceless possession anywhere world called american citizenship theres
higher honor greater privilege honor president thank much time id like recognize
five new citizens join us today robert martin ramirez alcaser 000103 bolivia law
ful permanent resident united states since 2013',
  '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 [7]:
```
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 fea
```

```
With a word cutoff of 5, we have 2378 as features in the model.
```

In [8]:
```
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','ju
            then this would return a dictionary of
            {'quick' : True,
             'fox' :    True}

    """
    ret_dict = {token: True for token in set(text.split(' ')) if token in fw}

    return(ret_dict)
```

In [9]:
```
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 [10]:
```
featuresets = [(conv_features(text,feature_words), party) for (text, party) in c
```

In [11]:
```
random.seed(20220507)
random.shuffle(featuresets)

test_size = 500
```

In [12]:
```
test_set, train_set = featuresets[:test_size], featuresets[test_size:]
classifier = nltk.NaiveBayesClassifier.train(train_set)
print(nltk.classify.accuracy(classifier, test_set))
```

```
0.498
```

In [13]:
```
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
```

In [14]:
```
cd = pd.DataFrame(convention_data, columns=['Text', 'Party'])
cd.head()
```

Out[14]:

| | Text | Party |
|---|---|---|
| 0 | skip content company careers press freelancers... | Democratic |
| 1 | im calling full session 48th quadrennial natio... | Democratic |
| 2 | every four years come together reaffirm democr... | Democratic |
| 3 | fight perfect union fighting soul country live... | Democratic |
| 4 | must come together defeat donald trump elect j... | Democratic |

In [15]:
```
cd['Party'].value_counts()
```

Out[15]:
```
Democratic    1551
Republican     990
Name: Party, dtype: int64
```

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

## My Observations

I tried a few different word_cutoff sizes and seems that 5 is working the best for this task. One main thing looking odd from the list is how the classifier model is favoring much on Republican.

Converting the `convention_data` into a Pandas dataframe, we can see the counts of each party is quite not balanced. Even though the number of `Replublican` is lower than the other, the model decided to favor more on it. Not only that, the top 25 features (or words) that are appearing above do not particularly favor one party to another (or it should not). Like the word "media" for example, in a normal sense it is a word that should quite be neutral between both parties but the classifier decided to use this word as one top feature to distinguish the label. This may be due to it's only taking account the number of a word appearing across the texts and not the actual meaning (semantics) behind the word. I'm assuming that the word 'china' is appearing most in one party (based on the print above) and not any much in the other. Because of this, its accuracy may not be so high, in fact lower than 50%.

Another possible solution would be configuring some of words appearing in stopwords like your or my because "your freedom got destroyed" vs "my freedom got destryoed" have different

meanings but if those stopwords are dropped, they will end up "freedom got destroyed" having the same meaning. Maybe this could help distinguishing the label?

## 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 [16]:
```
cong_db = sqlite3.connect("congressional_data.db")
cong_cur = cong_db.cursor()
```

In [17]:
```
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 [18]:
```
# 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.

# Decode the tweets since it is in byte
tweet_data = [[clean_text(row[2].decode(), is_tweet=True), row[1]] for row in re
```

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 [19]:
```
random.seed(20201014)

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

In [20]:
```
featuresets = [(conv_features(text,feature_words), party) for (text, party) in c
```

In [21]:
```
for tweet, party in tweet_data_sample:
    fw = conv_features(tweet, feature_words)
    estimated_party = classifier.classify(fw)
    # Fill in the right-hand side above with code that estimates the actual part
```

```
    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 healt
h care women praised work central coast
Actual party is Democratic and our classifer says Republican.

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

Here's our (cleaned) tweet: apparently trump thinks easy students overwhelmed cr
ushing burden debt pay student loans #trumpbudget
Actual party is Democratic and our classifer says Republican.

Here's our (cleaned) tweet: grateful first responders rescue personnel firefight
ers police volunteers working tirelessly keep people safe provide muchneeded hel
p putting lives
Actual party is Republican and our classifer says Republican.

Here's our (cleaned) tweet: lets make even greater #kag
Actual party is Republican and our classifer says Republican.

Here's our (cleaned) tweet: 1hr tie series 22 im #allin216 scared #roadtovictory
Actual party is Democratic and our classifer says Republican.

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

Here's our (cleaned) tweet: really close 3500 raised toward match right whoot th
ats 7000 nonmath majors room help us get
Actual party is Democratic and our classifer says Republican.

Here's our (cleaned) tweet: today comment period plan expand offshore drilling o
pened public 60 days march 9 share oppose proposed program directly trump admini
stration comments made email mail
Actual party is Democratic and our classifer says Republican.

Here's our (cleaned) tweet: celebrated 22 years eastside commitment amp saluted
community leaders last nights awards dinner
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 [22]:

```
# 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)

y_pred = []
y_true = []
```

```
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.
    fw = conv_features(tweet, feature_words)

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

    results[party][estimated_party] += 1

    y_true.append(party)
    y_pred.append(estimated_party)

    if idx > num_to_score :
        break

y_true = pd.Series(y_true, name='Actual')
y_pred = pd.Series(y_pred, name='Pred')
```

In [23]:
```
results
```

Out[23]:
```
defaultdict(<function __main__.<lambda>()>,
            {'Republican': defaultdict(int,
                         {'Republican': 3714, 'Democratic': 564}),
             'Democratic': defaultdict(int,
                         {'Republican': 4829, 'Democratic': 895})})
```

In [24]:
```
pd.crosstab(y_true, y_pred)
```

Out[24]:

| Pred | Democratic | Republican |
|---|---|---|
| Actual | | |
| **Democratic** | 895 | 4829 |
| **Republican** | 564 | 3714 |

## Reflections

As mentioned in the first part, it seems that the classifier is favoring the `Repulican` party than the other which can easily be seen by the results or the confusion matrix right above. This could be due to imbalanced dataset or selection of keywords (shown above with 25 top features). To better or improve the performance of the model, choosing which set of words to use to distinguish between different labels is important. At this moment, regardless of the actual semantics of texts, it's naively making most of its prediction as one ( `Republican` ) hoping to achieve the best result.

A few ways to improve the model's accuracy exist as mentioned earlier in the part 1. Another possible way would be using different model (of course). Because there is no hyperparameter tuning for this model, there is limit to what we can do to improve the model. Also actually using the hashtags could improve as well but since there are a very few data we can utilize (not even 10k), this may not be viable.