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 nltk.corpus import stopwords
from string import punctuation
from collections import Counter, defaultdict
# Some punctuation variations
punctuation = set(punctuation) # speeds up comparison

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

# Feel free to include your text patterns functions
#from text_functions_solutions import clean_tokenize, get_patterns

def remove_stop(tokens) :
    # modify this function to remove stopwords
    tokens = [t for t in tokens if t not in sw and 'https' not in t]

    tokens = [t for t in tokens if t.isalpha() or t.isnumeric()]
    return tokens

def remove_punctuation(text, punct_set=punctuation) :
    return("".join([ch for ch in text if ch==' ' or (ch not in punct_set and ch.isalnum())]))

def tokenize(text) :

    # modify this function to return tokens
    token = text.split()
    return token

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

    for transform in pipeline :
        tokens = transform(tokens)

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

In [2]:
```
convention_db = sqlite3.connect(r'C:\Users\gauth\Documents\Harini\San Diego University\applied text mining\2020
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 [3]:
```
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 text,party FROM conventions
                            ''')

for row in query_results :
    # store the results in convention_data

    text = row[0]

    text = ' '.join(prepare(text,my_pipeline))
    convention_data.append([text,row[1]])
```

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

In [4]:
```
random.choices(convention_data,k=10)
```

Out[4]:
```
[['military better equipped better resourced better manned military world president trump demolished terrorist isis caliphate middle east eliminated leader albaghdadi one worlds brutal terrorists president trump took decis ive action iranian terrorist mastermind qasem soleimani man responsible deaths hundreds american service men ir aq nato allies failed meet commitments upheld president trump demanded parody nato members increased contributi ons 100 billion year natos secretary general credits president donald j trump',
  'Republican'],
 ['must interject one thing end chat said could favor said sure said tell get grandsons hair good humor sensiti ve years later vivid mind',
  'Democratic'],
 ['please', 'Republican'],
 ['china would prefer joe biden', 'Republican'],
 ['born 15 heart condition known transposition greater arteries ended requiring transplant thankfully aca alrea dy taken effect thats something think time especially coverage potential coverage maxes initial hospital stay 3 million billed back insurance',
  'Democratic'],
 ['washington dc welcome 2020 republican national convention tonight celebrating america land promise',
  'Republican'],
 ['seen tragic shootings sides officers killing citizens citizens killing officers line duty police officers fe llow citizens live country families live communities unfortunately democrats turned backs brave police officers call defunding danger cities neighborhoods children isnt ironic democrat politicians personal security protect dont forego security replaced social workers especially since thats want police need funding less frequent psyc hological examinations nonlethal remote restraint technology deescalation use force training common sense solut ions president trump supports true sincere police reform thats right donald trump',
  'Republican'],
 ['everything family weve always done everything family', 'Democratic'],
 ['time next year want see joe biden oval office', 'Democratic'],
 ['say crime time however time fair weve made mistakes none us want defined forever based worst decision prison became playwright mentor certified hospice volunteer ordained minister received special olympics event coordina tor year award work disabled women thing worse unjustly imprisoning body trying imprison mind transformation de scribed extraordinary truth thousands people like deserve opportunity come home never stopped fighting freedom christian faith prayers many kept hope alive',
  '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 [5]:
```
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 features in the model.")
```

```
With a word cutoff of 5, we have 2373 as features in the model.
```

In [6]:
```
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}

    """

    # Your code here
    ret_dict = dict()
    for token in text.split():
        if token in fw:
            ret_dict[token] = True
    return(ret_dict)
```

In [7]:
```
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 [8]:
```
featuresets = [(conv_features(text,feature_words), party) for (text, party) in convention_data]
```

In [9]:
```
random.seed(20220507)
random.shuffle(featuresets)

test_size = 500
```

In [10]:
```
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 [11]:
```
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?

Observations

Most of the important features belongs to Republican party.The model accuracy is 50% only. China appears more in Republic vs. Democratic

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 [12]:
```
cong_db = sqlite3.connect(r'C:\Users\gauth\Documents\Harini\San Diego University\applied text mining\congressio
cong_cur = cong_db.cursor()
```

In [13]:
```
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 [14]:
```
from tqdm import tqdm
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 tqdm(results) :
    # store the results in convention_data

    text = row[2]

    text = ' '.join(prepare(text,my_pipeline))
    tweet_data.append([text,row[1]])
```

```
100%|████████████████████████████████████████| 664656/664656 [00:38<00:00, 17215.26it/s]
```

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 [15]:
```
random.seed(20201014)

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

In [16]:
```
word_cutoff = 0

tokens = [w for t, p in tweet_data_sample 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 0, we have 119 as features in the model.
```

In [17]:
```
for tweet, party in tweet_data_sample :

    estimated_party = classifier.classify(conv_features(tweet,feature_words))
    # 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: bearlier today spoke house floor abt protecting health care women praised ppmarmont e work central coast
Actual party is Democratic and our classifer says Republican.

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

Here's our (cleaned) tweet: bapparently trump thinks easy students overwhelmed crushing 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 firefighters police volunteers working t irelessly keep people safe provide muchneeded help putting lives
Actual party is Republican and our classifer says Republican.

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

Here's our (cleaned) tweet: bwe cavs tie series 22 im repbarbaralee scared roadtovictory
Actual party is Democratic and our classifer says Republican.

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

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

Here's our (cleaned) tweet: btoday comment period plan expand offshore drilling opened public 60 days march 9 s hare oppose proposed program directly trump administration comments made email mail
Actual party is Democratic and our classifer says Republican.

Here's our (cleaned) tweet: bcelebrated 22 years eastside commitment amp saluted community leaders last 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 [18]:
```
# 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.seed(20220507)
random.shuffle(tweet_data)


word_cutoff = 5

tokens = [w for t, p in tweet_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.")


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(conv_features(tweet,feature_words))

    results[party][estimated_party] += 1

    if idx > num_to_score :
        break
```

```
With a word cutoff of 5, we have 47065 as features in the model.
```

In [19]:
```
results
```

Out[19]:
```
defaultdict(<function __main__.<lambda>()>,
            {'Republican': defaultdict(int,
                         {'Republican': 3678, 'Democratic': 714}),
             'Democratic': defaultdict(int,
                         {'Republican': 4620, 'Democratic': 990})})
```

In [20]:
```
sum(results['Republican'].values())
```

Out[20]:
```
4392
```

In [21]:
```
sum(results['Democratic'].values())
```

Out[21]:
```
5610
```

Reflections

For the Republican party, there are 15% false predictions.For Democratic party, there are 80% false predictions. The accuracy is much lower here.

In [ ]: