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 [1]:

```
import sqlite3
import random
import numpy as np
import pandas as pd
from collections import Counter, defaultdict
import nltk
nltk.download('stopwords')
stopwords = set(nltk.corpus.stopwords.words('english'))
from nltk.classify.scikitlearn import SklearnClassifier

from string import punctuation
import re
import os
import html



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

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

In [2]:

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

# Stopwords
sw = set(stopwords)

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

def descriptive_stats(tokens, num_tokens = 5, verbose=True) :
    counter = Counter()
    tokens.map(counter.update)
    freq_df = pd.DataFrame.from_dict(counter, orient='index', columns=['freq'])
    counter_df = pd.DataFrame.from_dict(counter, orient='index').reset_index()
    num_tokens = sum(freq_df['freq'])
    num_unique_tokens = freq_df.shape[0]
    lexical_diversity = num_unique_tokens / num_tokens
    num_characters = sum((counter_df['index'].str.len()) * counter_df[0])

    if verbose :
        print(f"There are {num_tokens} tokens in the data.")
        print(f"There are {num_unique_tokens} unique tokens in the data.")
        print(f"There are {num_characters} characters in the data.")
        print(f"The lexical diversity is {lexical_diversity:.3f} in the data.")
        print(f"The top 5 most common words are")
        print(counter.most_common(5))

    return(0)

def remove_stop(tokens) :
    return [t for t in tokens if t.lower() not in stopwords]
```

```
def remove_punctuation(text, punct_set=tw_punct) :
    return("".join([ch for ch in text if ch not in punct_set]))

def tokenize(text):
    return re.findall(r'\S+', text)

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

    for transform in pipeline :
        tokens = transform(tokens)

    return(tokens)
```

In [3]:

```
def clean(text):
    # convert html escapes like &amp; to characters.
    text = html.unescape(text)
    # tags like <tab>
    text = re.sub(r'<[^<>]*>', ' ', text)
    # markdown URLs like [Some text](https://....)
    text = re.sub(r'\[([^\[\]]*)\]\([^\(\)]*\)', r'\1', text)
    # text or code in brackets like [0]
    text = re.sub(r'\[[^\[\]]*\]', ' ', text)
    # standalone sequences of specials, matches &# but not #cool
    text = re.sub(r'(?:^|\s)[&#<>{}\[\]+|\\:-]{1,}(?:\s|$)', ' ', text)
    # standalone sequences of hyphens like --- or ==
    text = re.sub(r'(?:^|\s)[\-=\+]{2,}(?:\s|$)', ' ', text)
    # sequences of white spaces
    text = re.sub(r'\s+', ' ', text)
    return text.strip()
```

In [4]:

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

In [5]:

```
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 [6]:

```
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;")

convention_table = []

for row in query_results :
    # store the results in convention_data
    text, party = row
    convention_table.append((text, party))

convention_df = pd.DataFrame(convention_table, columns = ["text", "party"])

convention_df["tokens"] = convention_df['text'].apply(
    prepare, pipeline = my_pipeline)
```

```python
convention_df["cleantext"] = convention_df["text"].apply(clean)
convention_df["cleantext"] = convention_df["cleantext"].apply(str.lower)
convention_df["cleantext"] = convention_df["cleantext"].apply(
    remove_punctuation)

#convention_df['text'] = tokens

convention_data = convention_df[["cleantext", "party"]].values.tolist()
```

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

In [7]:
```python
random.choices(convention_data,k=10)
```

Out[7]:
```
[['mccain passed his vote with a thumbs down', 'Democratic'],
 ['we need to change the paradigm and that happens here with us',
  'Democratic'],
 ['at the end of the day i think we’re all very happy we had that meeting',
  'Democratic'],
 ['and if you give him your cell phone number… ashley biden  015143  he’s going to call it',
  'Democratic'],
 ['questions about money he made from foreign business dealings while his father was vice preside
nt',
  'Republican'],
 [' relatives as a first american and citizen of the standing rock sioux tribe i welcome you to t
he paha sapa the black hills the site of my creation story and home to the oceti sakowin the grea
t sioux nation we often say  we are all related our next president must lead by this philosophy f
or the betterment of our next seven generations we cast 3 votes for senator bernie sanders and 17
votes for our next president joe biden',
  'Democratic'],
 ['good evening i’m sally yates speaking at a political convention is something i never expected
to be doing but the future of our democracy is at stake i’m here in my hometown of atlanta where
as a young lawyer i joined our nation’s justice department for nearly 30 years through democratic
and republican administrations i worked alongside my doj colleagues to advance our nation’s promi
se of equal justice',
  'Democratic'],
 ['let’s give parents the peace of mind that their kids are safe and are being set up for succes
s',
  'Democratic'],
 ['focused on the wellbeing of children social media use and opioid abuse',
  'Republican'],
 ['the plan was working everybody had a job making money spending money boom bang boom we’re goo
d',
  '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 [8]:
```python
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}")
print(f"we have {len(feature_words)} as features in the model.")
```

```
With a word cutoff of 5
we have 2510 as features in the model.
```

In [9]:
```python
convention_df["tokens"] = convention_df["cleantext"].apply(
    prepare,pipeline=my_pipeline)
```

In [10]:
```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}
        
    """
    
    # Your code here
    
    ret_dict = dict()
    
    tokens = tokenize(text)
    for token in tokens:
        if token in fw :
            ret_dict[token] = True
    
    return(ret_dict)
```

In [11]:
```python
assert(len(feature_words)>0)
print(conv_features("donald is the president",feature_words))
#=={'donald':True,'president':True})
print(conv_features("some people in america are citizens",feature_words))
#=={'people':True,'america':True,"citizens":True})
#All result in true - the data is structured differently so ASSERT does not work
```

```
{'donald': True, 'is': True, 'the': True, 'president': True}
{'some': True, 'people': True, 'in': True, 'america': True, 'are': 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 [12]:
```python
featuresets = [(conv_features(text,feature_words), party)
               for (text, party) in convention_data]
```

In [13]:
```python
random.seed(20220507)
random.shuffle(featuresets)

test_size = 500
```

In [14]:
```python
test_set, train_set = featuresets[:test_size], featuresets[test_size:]
classifier = nltk.NaiveBayesClassifier.train(train_set)
print(nltk.classify.accuracy(classifier, test_set))
```

```
0.444
```

In [15]:
```python
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
               greatness = True           Republ : Democr =     12.1 : 1.0
                 abraham = True           Republ : Democr =     11.9 : 1.0
                  defund = 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

The analysis of the classifier reveals intriguing patterns. Republicans often emphasize patriotic buzzwords like "destroy," "freedoms," and "flag," aiming to evoke national pride and potentially nationalism among their supporters. The repeated mentions of "enemy," "isis," and "China" suggest a divisive tone. In contrast, Democrats focus on broader issues like climate and voting, possibly reflecting their emphasis on environmental concerns and mobilizing voters. The distinction in language use underscores the parties' different priorities and communication strategies during the critical period of 2020, particularly regarding the global pandemic and geopolitical tensions.

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]:
```
resultsdf = pd.DataFrame(results, columns=['author', 'party', 'text'])
```

In [19]:
```
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.


text = []

for row in resultsdf["text"]:
    try:
        text.append(row.decode())
    except:
        text.append(row.encode())
```

In [20]:
```
resultsdf["text"] = text
resultsdf["tokens"] = resultsdf["text"].apply(prepare,pipeline=my_pipeline)
resultsdf["cleantext"] = resultsdf["text"].apply(clean)
resultsdf["cleantext"] = resultsdf["cleantext"].apply(str.lower)
resultsdf["cleantext"] = resultsdf["cleantext"].apply(remove_punctuation)
resultsdf
```

Out[20]:

	author	party	text	tokens	cleantext
0	Mo Brooks	Republican	"Brooks Joins Alabama Delegation in Voting Aga...	[brooks, joins, alabama, delegation, voting, f...	brooks joins alabama delegation in voting agai...
1	Mo Brooks	Republican	"Brooks: Senate Democrats Allowing President t...	[brooks, senate, democrats, allowing, presiden...	brooks senate democrats allowing president to ...
2	Mo Brooks	Republican	"NASA on the Square" event this Sat. 11AM – 4P...	[nasa, square, event, sat, 11am, –, 4pm, stop,...	nasa on the square event this sat 11am – 4pm s...
3	Mo Brooks	Republican	"The trouble with Socialism is that eventually...	[trouble, socialism, eventually, run, peoples,...	the trouble with socialism is that eventually ...
4	Mo Brooks	Republican	"The trouble with socialism is eventually you ...	[trouble, socialism, eventually, run, peoples,...	the trouble with socialism is eventually you r...
...	...	...	...	...	...
664651	David McKinley	Republican	We had a great time at the WVU Homecoming para...	[great, time, wvu, homecoming, parade, yesterd...	we had a great time at the wvu homecoming para...
664652	David McKinley	Republican	We need more transparency in Washington #wvpol...	[need, transparency, washington, #wvpol, https...	we need more transparency in washington #wvpol...
664653	David McKinley	Republican	We saw there is a double standard in DC and th...	[saw, double, standard, dc, rules, simply, don...	we saw there is a double standard in dc and th...
664654	David McKinley	Republican	Wow! Huge crowd in Charleston at the @WVGOP vi...	[wow, huge, crowd, charleston, wvgop, victory,...	wow huge crowd in charleston at the wvgop vict...
664655	David McKinley	Republican	https://t.co/0QmZlRfEcD https://t.co/FY520NC2GB	[httpstco0qmzlrfecd, httpstcofy520nc2gb]	httpstco0qmzlrfecd httpstcofy520nc2gb

664656 rows × 5 columns

In [21]:
```
query_results = resultsdf[["cleantext", "party"]]
tweet_data = query_results.values.tolist()
#tweet_data
```

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 [22]:
```
random.seed(20201014)

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

In [23]:
```
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 mode
```

```
With a word cutoff of 5, we have 51762 as features in the model.
```

In [24]:
```python
featuresets = [(conv_features(text,feature_words), party)
               for (text, party) in tweet_data]
tweet_data = featuresets
```

In [25]:
```python
random.seed(20201014)

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

In [26]:
```python
classifier = nltk.NaiveBayesClassifier.train(tweet_data_sample)
```

In [27]:
```python
for tweet, party in tweet_data_sample :
    estimated_party = classifier.classify(tweet)
    # 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': True, 'today': True, 'i': True, 'spoke': True, 'on': Tru
e, 'the': True, 'house': True, 'floor': True, 'abt': True, 'protecting': True, 'health': True, 'c
are': True, 'for': True, 'women': True, 'and': True, 'praised': True, 'their': True, 'work': Tru
e, 'central': True, 'coast': True}
Actual party is Democratic and our classifer says Republican.

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

Here's our (cleaned) tweet: {'apparently': True, 'trump': True, 'thinks': True, 'its': True, 'jus
t': True, 'too': True, 'easy': True, 'for': True, 'students': True, 'overwhelmed': True, 'by': Tr
ue, 'the': True, 'crushing': True, 'burden': True, 'of': True, 'debt': True, 'to': True, 'pay': T
rue, 'off': True, 'student': True, 'loans': True, '#trumpbudget': True}
Actual party is Democratic and our classifer says Democratic.

Here's our (cleaned) tweet: {'we’re': True, 'grateful': True, 'for': True, 'our': True, 'first':
True, 'responders': True, 'rescue': True, 'personnel': True, 'firefighters': True, 'police': Tru
e, 'and': True, 'volunteers': True, 'who': True, 'have': True, 'been': True, 'working': True, 'ti
relessly': True, 'to': True, 'keep': True, 'people': True, 'safe': True, 'provide': True, 'muchne
eded': True, 'help': True, 'while': True, 'putting': True, 'their': True, 'own': True, 'lives': T
rue, 'on': True, 'the': True, 'line': True}
Actual party is Republican and our classifer says Republican.

Here's our (cleaned) tweet: {'let’s': True, 'make': True, 'it': True, 'even': True, 'greater': Tr
ue, '#kag': True, 'us': True}
Actual party is Republican and our classifer says Republican.

Here's our (cleaned) tweet: {'we': True, 'have': True, 'about': True, 'until': True, 'the': True,
'cavs': True, 'tie': True, 'up': True, 'series': True, '22': True, 'im': True, '#allin216': True,
'repbarbaralee': True, 'you': True, 'scared': True, '#roadtovictory': True}
Actual party is Democratic and our classifer says Democratic.

Here's our (cleaned) tweet: {'congrats': True, 'to': True, 'on': True, 'his': True, 'new': True,
'gig': True, 'at': True, 'sd': True, 'city': True, 'hall': True, 'we': True, 'are': True, 'glad':
True, 'you': True, 'will': True, 'continue': True}
Actual party is Democratic and our classifer says Democratic.

Here's our (cleaned) tweet: {'we': True, 'are': True, 'really': True, 'close': True, 'have': Tru
e, 'over': True, '3500': True, 'raised': True, 'toward': True, 'the': True, 'match': True, 'righ
t': True, 'now': True, 'that’s': True, '7000': True, 'for': True, 'majors': True, 'in': True, 'ro
om': True, '😂': True, 'help': True, 'us': True, 'get': True, 'there': True}
Actual party is Democratic and our classifer says Democratic.

Here's our (cleaned) tweet: {'today': True, 'the': True, 'comment': True, 'period': True, 'for':
True, 'potus’s': True, 'plan': True, 'to': True, 'expand': True, 'offshore': True, 'drilling': Tr
ue, 'opened': True, 'public': True, 'you': True, 'have': True, '60': True, 'days': True, 'until':
True, 'march': True, '9': True, 'share': True, 'why': True, 'oppose': True, 'proposed': True, 'pr
ogram': True, 'directly': True, 'with': True, 'trump': True, 'administration': True, 'comments':
True, 'can': True, 'be': True, 'made': True, 'by': True, 'email': True, 'or': True, 'mail': True}
Actual party is Democratic and our classifer says Democratic.

Here's our (cleaned) tweet: {'celebrated': True, '22': True, 'years': True, 'of': True, 'eastsid
e': True, 'commitment': True, 'saluted': True, 'community': True, 'leaders': True, 'at': True, 'l
ast': True, 'night’s': True, 'awards': True, 'dinner': True}
Actual party is Democratic and our classifer says Democratic.
```

In [28]:

```python
print(nltk.classify.accuracy(classifier, tweet_data))
```

```
0.48471991526443753
```

In [29]:

```python
classifier.show_most_informative_features(25)
```

```
Most Informative Features
                     and = True           Republ : Democr =      3.0 : 1.0
                    help = True           Republ : Democr =      3.0 : 1.0
                   their = True           Republ : Democr =      3.0 : 1.0
                    #kag = None           Democr : Republ =      1.9 : 1.0
                    been = None           Democr : Republ =      1.9 : 1.0
                    even = None           Democr : Republ =      1.9 : 1.0
            firefighters = None           Democr : Republ =      1.9 : 1.0
                   first = None           Democr : Republ =      1.9 : 1.0
                grateful = None           Democr : Republ =      1.9 : 1.0
                 greater = None           Democr : Republ =      1.9 : 1.0
                      it = None           Democr : Republ =      1.9 : 1.0
                    keep = None           Democr : Republ =      1.9 : 1.0
                   let’s = None           Democr : Republ =      1.9 : 1.0
                    line = None           Democr : Republ =      1.9 : 1.0
                   lives = None           Democr : Republ =      1.9 : 1.0
                    make = None           Democr : Republ =      1.9 : 1.0
              muchneeded = None           Democr : Republ =      1.9 : 1.0
                     our = None           Democr : Republ =      1.9 : 1.0
                     own = None           Democr : Republ =      1.9 : 1.0
                  people = None           Democr : Republ =      1.9 : 1.0
               personnel = None           Democr : Republ =      1.9 : 1.0
                  police = None           Democr : Republ =      1.9 : 1.0
                 provide = None           Democr : Republ =      1.9 : 1.0
                 putting = None           Democr : Republ =      1.9 : 1.0
                  rescue = None           Democr : Republ =      1.9 : 1.0
```

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

In [30]:
```
# 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 [31]:
```
results
```

Out[31]:
```
defaultdict(<function __main__.<lambda>()>,
            {'Republican': defaultdict(int,
                         {'Republican': 1408, 'Democratic': 2870}),
             'Democratic': defaultdict(int,
                         {'Republican': 2303, 'Democratic': 3421})})
```

Reflections

With a smaller sample size, the tweet dataset exhibits lower keyword indicativeness compared to congressional data. Although overall accuracy reaches 48.5%, establishing clear party-keyword connections proves challenging. The model's inclination towards Republicans suggests a disparity between political speeches and tweets. The abundance and diversity of tweets, including hashtags and errors, complicate classification. Naive Bayes' independence assumption and the dominance of Republican features may skew results. Refined feature engineering, accounting for the nuances of Twitter discourse, is crucial for improving accuracy in capturing the intricacies of political language on social media.