# Political Naive Bayes

June 6, 2022

## 0.1 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.

[218]:
```
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
from string import punctuation
import re as re
```

[2]:
```
convention_db = sqlite3.connect("2020_Conventions.db")
convention_cur = convention_db.cursor()
```

### 0.1.1 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.

[16]:
```
#In order to understand what to query, we're going to need the name of the␣
↪table and the name of the columns.
#1. get a lit of all available tables
tables = convention_cur.execute("SELECT name FROM sqlite_master WHERE␣
↪type='table';")
for table in tables:
    print(table)
```

```
('conventions',)
```

[17]:
```
#get a list of all available columns from that table
convention_db.row_factory = sqlite3.Row
cursor = convention_db.execute('select * from conventions')
```

```python
# instead of cursor.description:
row = cursor.fetchone()
names = row.keys()
print(names)
```

['party', 'night', 'speaker', 'speaker_count', 'time', 'text', 'text_len',
'file']

[279]:
```python
sw = stopwords.words("english")
punctuation = re.sub('#','',punctuation)

def remove_stop_words(tokens):
    return [t for t in tokens if t not in sw]

def clean_text(text):
    #remove stop words - because of the nature of our stopword dictionary,
    #we have to split by space OR apostrophe OR comma
    this_tokens = re.split("[\,|\'|\s+]",text.lower())
    this_tokens = remove_stop_words(this_tokens)
    #remove URLs
    this_tokens = [t for t in this_tokens if re.search('http',t) is None ]
    #untokenize
    clean_text = " ".join(this_tokens)
    #remove any duplicate spaces
    clean_text = re.sub(' +', ' ',clean_text)
    #remove punctuation
    clean_text = re.sub(f'[{punctuation}]','',clean_text)
    #remove trailing spaces
    clean_text = clean_text.strip()
    return clean_text
```

[280]:
```python
convention_data = []
dirty = []
# 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.

# we can set the text to lowercase with SQLite's lower() function.
query_results = convention_cur.execute(
                            '''
                            SELECT lower(text), party FROM conventions
                            ''')

for row in query_results :

    #debugging list for clean text that ends up being too short.
    dirty.append(row[0])
```

```
    cleaned_text = clean_text(row[0])

    # store the results in convention_data
    clean_row = [cleaned_text,row[1]]
    convention_data.append(clean_row)
```

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

[281]:
```
random.choices(convention_data,k=10)
```

[281]:
```
[['chocolate breyer half chocolate half vanilla likes ice cream hidden ways',
  'Democratic'],
 ['return higher standard', 'Republican'],
 ['joe biden leader donnamarie w  3639  really wants best country georgia m
3642  understands respects democracy rule law us constitution jacqueline a  3647
move toward creating perfect union jacqueline a  3655  singing',
  'Democratic'],
 ['want help joe kamala make sure america stays strong united please go
joebidencom contribute anything possibly can tonight prouder loyal union member
passionate climate activist patriotic democrat donald trump call tweet tomorrow
washed horse face talent low ratings well due respect sir takes one know one
like introduce real american hero world war ii veteran ed good',
  'Democratic'],
 ['teaching jill is jill simply cares cares people dr', 'Democratic'],
 ['sweet grandkids yay official nominee onto next step electing joe biden kamala
harris november make sure plan vote text vote 30330 find how going talk topic
touches lives healthcare affordable care act gamechanging pandemic revealed
important protect improve it increasing access healthcare bringing cost always
priority joe biden joe us healthcare personal',
  'Democratic'],
 ['privilege … speaker 21  5551  nominate four years', 'Republican'],
 ['pushed edge anyone could expected bear', 'Democratic'],
 ['ranchers miners cowboys sheriffs farmers settlers pressed past mississippi
stake claim wild frontier legends born wyatt earp annie oakley davy crockett
buffalo bill americans built beautiful homesteads open range soon churches
communities towns time great centers industry commerce were americans build
future tear past nation revolution toppled tyranny fascism delivered millions
freedom laid railroads built great ships raised sky scrapers revolutionized
industry sparked new age scientific discovery set trends art music radio film
sport literature style confidence flair are whenever way life threatened heroes
answered call yorktown gettysburg normandy iwo jima american patriots raised
cannon blasts bullets bayonets rescue american liberty',
  'Republican'],
 ['people quandary present people search future attempting fulfill national
purpose create sustain society us equal',
  '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.

[287]:
```
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 and word not in sw :
        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 2336 as features in the model.
```

[288]:
```
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}

    """

    #split raw text by spaces, return anything in the fw arg, pass to a counter␣
 ↪to dictionarize,
    #reset all key values to true
    tokens = [t for t in text.split() if t in fw]
```

```
    ret_dict = Counter(tokens)
    ret_dict = dict.fromkeys(ret_dict, True)

    return(ret_dict)
```

[290]:
```
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.

[291]:
```
featuresets = [(conv_features(text,feature_words), party) for (text, party) in␣
 ↪convention_data]
```

[338]:
```
random.seed(20220507)
random.shuffle(featuresets)

test_size = 500
```

[339]:
```
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
```

[340]:
```
classifier.show_most_informative_features(25)
```

```
Most Informative Features
             enforcement = True           Republ : Democr =     36.5 : 1.0
                 radical = True           Republ : Democr =     36.5 : 1.0
                   votes = True           Democr : Republ =     22.5 : 1.0
                   media = True           Republ : Democr =     17.9 : 1.0
                 destroy = True           Republ : Democr =     17.4 : 1.0
                    race = True           Republ : Democr =     16.4 : 1.0
               greatness = True           Republ : Democr =     15.3 : 1.0
                   china = True           Republ : Democr =     15.0 : 1.0
                   allow = True           Republ : Democr =     14.3 : 1.0
                preserve = True           Republ : Democr =     14.3 : 1.0
                  lowest = True           Republ : Democr =     13.2 : 1.0
              prosperity = True           Republ : Democr =     13.2 : 1.0
                    mike = True           Republ : Democr =     12.4 : 1.0
                 defense = True           Republ : Democr =     12.2 : 1.0
                religion = True           Republ : Democr =     12.2 : 1.0
                      25 = True           Republ : Democr =     11.1 : 1.0
                 abraham = True           Republ : Democr =     11.1 : 1.0
```

```
           countries = True           Republ : Democr =     11.1 : 1.0
              earned = True           Republ : Democr =     11.1 : 1.0
                iran = True           Republ : Democr =     11.1 : 1.0
            recently = True           Republ : Democr =     11.1 : 1.0
           democracy = True           Democr : Republ =     11.0 : 1.0
            bringing = True           Republ : Democr =     10.5 : 1.0
              prison = True           Republ : Democr =     10.5 : 1.0
             culture = True           Republ : Democr =     10.0 : 1.0
```

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

### 0.1.2 My Observations

The majority of informative features listed seem to display a ratio of republican to democrat usage, rather than the other way around. Based on layman political knowledge, this lines up with expectations about what each party is likely to discuss at great length - Republicans being largely concerned about China's expansionist economic policy, freedoms, crime, and isis, while the terms with a democrat:republican ratio also coincide with political expectations - discussion of climate change and environmental policy is often democrat-led. The fact that "votes" tends to be used more often by democrats, however, seems unintuitive given that both parties discuss voting rights in different dimensions.

## 0.2 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.

[114]:
```python
cong_db = sqlite3.connect("congressional_data.db")
cong_cur = cong_db.cursor()
```

[375]:
```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
```

[376]:
```
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 result in results:
    #convert from bytes to string
    this_tweet = result[2].decode("utf-8")
    this_party = result[1]
    this_tweet = clean_text(this_tweet)
    tweet_data.append([this_tweet,this_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...

[377]:
```
random.seed(20201014)

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

**Developer Note:** The ambiguity of this exercise has forced me to create this loop twice, once for the old model, and a model based only off tweet data.

[378]:
```
word_cutoff = 10
tokens = [w for t, p in tweet_data for w in t.split()]
word_dist = nltk.FreqDist(tokens)
tw_feature_words = set()

for word, count in word_dist.items() :
    if count > word_cutoff and word not in sw :
        tw_feature_words.add(word)

print(f"With a word cutoff of {word_cutoff}, we have {len(tw_feature_words)} as␣
↪features in the twitter model.")
```

```
With a word cutoff of 10, we have 32977 as features in the twitter model.
```

[379]:
```
twitter_features = [(conv_features(text,tw_feature_words), party) for (text,␣
↪party) in tweet_data]
```

### 0.2.1 Somehow, we got an accuracy of around 74% on the NB and 90% on the old NB classifier. Let's compare it to a baseline.

[380]:
```
tw_test_set, tw_train_set = twitter_features[:test_size],␣
↪twitter_features[test_size:]
classifier_2 = nltk.NaiveBayesClassifier.train(tw_train_set)
print(nltk.classify.accuracy(classifier_2, tw_test_set))
print(nltk.classify.accuracy(classifier, tw_test_set))
```

```
0.738
0.908
```

[344]:
```
import pandas as pd
tweet_df = pd.DataFrame(tweet_data,columns=['text','party'])
```

### 0.2.2 Given that the dataset is balanced, our baseline would be guessing democrat.

Given this, an accuracy of 74% **Or** 90% is substantially better than our convention performance.

[442]:
```
tweet_df['party'].value_counts()/len(tweet_df)
```

[442]:
```
Democratic    0.565894
Republican    0.434106
Name: party, dtype: float64
```

## 0.3 Iterate over a sample of tweets with the superior NB Classifier based off our convention data:

[443]:
```
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: earlier today spoke house floor abt protecting
health care women praised ppmarmonte 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
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 tirelessly keep people safe provide
muchneeded help putting lives line
Actual party is Republican and our classifer says Republican.

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

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

Here's our (cleaned) tweet: congrats belliottsd 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 now
whoot that 7000 nonmath majors room   help us get
Actual party is Democratic and our classifer says Republican.

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

Here's our (cleaned) tweet: celebrated icseastla 22 years eastside commitment
amp saluted community leaders last night 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.

[444]:
```
# 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(conv_features(tweet,feature_words))

    results[party][estimated_party] += 1

    if idx > num_to_score :
```

```
        break
```

[445]:
```
results
```

[445]:
```
defaultdict(<function __main__.<lambda>()>,
            {'Republican': defaultdict(int,
                         {'Republican': 3735, 'Democratic': 651}),
             'Democratic': defaultdict(int,
                         {'Republican': 4663, 'Democratic': 953})})
```

### 0.3.1 Reflections

It seems that our classifier was equally accurate for both groups with an accuracy of about 84% for democratic tweets and about 84% for repulican tweets. This could, however, be a result of the volatility of sampling, and we might benefit from repeatedly sampling from our test set to get a better picture. However, this result is close enough to the 90% accuracy our convention NB classifier got when pointed at tweet data.