Module 4-Political Naive Bayes

June 4, 2025

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.

[1]:
```
import sqlite3
import nltk
import random
import numpy as np
from collections import Counter, defaultdict

from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize


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

[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.

[3]:
```
convention_data = []
stop_words = set(stopwords.words('english'))

# 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 text, value in query_results:
    words = [word.lower() for word in word_tokenize(text) if word.isalpha()]
    removed_stop = [word for word in words if word not in stop_words]
    join = ' '.join(removed_stop)
    convention_data.append([join, value])
```

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

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

[4]:
```
[['also great back wisconsin lucky enough marry wife ann marie little three
decades ago progressive movement deep roots since today anniversary amendment
ratification point wisconsin first state ratify',
  'Democratic'],
 ['', 'Democratic'],
 ['name lakeisha cole met husband years ago started dating college graduated
college eloped two weeks deployed',
  'Democratic'],
 ['communist party china comparison joe biden aided abetted china rise years
terrible trade deals closed factories laid workers president trump stands china
cheating stealing lying joe biden allowed chinese fentanyl flood across southern
border president trump sanctioned chinese drug dealer poisoning kids joe said
chinese communist even competitor bad folks months unleashed plague world
president trump clear eyed chinese threat making china pay china giving fact
rooting joe biden america enemies give either joe biden would wrong weak next
four years last',
  'Republican'],
 ['trump pledge american workers definitely means lot today truly believe kids
going look one day tremendous could guess',
  'Republican'],
 ['years ago tonight suffragists based hermitage hotel nashville cheered
tennessee became deciding state ratify amendment granting women right vote year
casting first presidential vote joe biden women decide election replace donald
trump president respects us tennessee cast votes bernie sanders votes next
president united states joseph biden',
  'Democratic'],
 ['radical left believes federal government must involved every aspect lives
correct american wrongs believe federal government needs dictate americans live
work raise children process deprive people freedom prosperity security agenda
based government control agenda based freedom president trump cut taxes joe
biden wants raise taxes nearly trillion president achieved energy independence
united states joe biden would abolish fossil fuels fracking impose regime
climate change regulations would drastically increase cost living working
families fought free fair trade president stood china ended era economic
surrender joe biden cheerleader communist china wants repeal tariffs leveling
playing field american workers actually criticized president trump suspending
travel china outset pandemic',
```

```
  'Republican'],
 ['democrats going left', 'Republican'],
 ['give law enforcement police back power afraid act afraid lose pension afraid
lose jobs afraid able job desperately want suffer great people protect want
protect even higher level police misconduct justice system must hold wrongdoers
fully completely accountable never situation things going today must never allow
mob rule',
  'Republican'],
 ['nebraska', '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.

[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 2236 as features in the model.
```

[7]:
```
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()

    split = text.split()
    for word in split:
        if word in fw:
            ret_dict[word] = True

    return(ret_dict)
```

[8]:
```
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.

[9]:
```
featuresets = [(conv_features(text,feature_words), party) for (text, party) in␣
  ↪convention_data]
```

[10]:
```
random.seed(20220507)
random.shuffle(featuresets)

test_size = 500
```

[11]:
```
test_set, train_set = featuresets[:test_size], featuresets[test_size:]
classifier = nltk.NaiveBayesClassifier.train(train_set)
print(nltk.classify.accuracy(classifier, test_set))
```

```
0.494
```

[12]:
```
classifier.show_most_informative_features(25)
```

```
Most Informative Features
                   china = True           Republ : Democr =     27.1 : 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 =     15.8 : 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?

0.1.2 My Observations

Baed on the classifier, I think the classifier is doing a good job distinguishing between the Republicans and Democrats and that most of the words are linked to Republicans than Democrats

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.

[13]:
```python
cong_db = sqlite3.connect("congressional_data.db")
cong_cur = cong_db.cursor()
```

[14]:
```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
```

[25]:
```
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 candidate, party, tweet_text in results:
    if isinstance(tweet_text, bytes):
        tweet_text = tweet_text.decode('utf-8', errors='ignore')
    words = [word.lower() for word in word_tokenize(tweet_text) if word.
 ↪isalpha()]
    removed_stop = [word for word in words if word not in stop_words]
    join = ' '.join(removed_stop)
    tweet_data.append([join, 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…

[26]:
```
random.seed(20201014)

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

[30]:
```
for tweet, party in tweet_data_sample :
    features = conv_features(tweet, feature_words)
    estimated_party = classifier.classify(features)
    # 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 https
Actual party is Democratic and our classifer says Republican.

Here's our (cleaned) tweet: go tribe rallytogether https
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
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 help
```

```
putting lives line https
Actual party is Republican and our classifer says Republican.

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

Here's our (cleaned) tweet: cavs tie series 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 https
Actual party is Democratic and our classifer says Republican.

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

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

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

[31]:
```
# 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
```

```
    features = conv_features(tweet, feature_words)
    estimated_party = classifier.classify(features)

    results[party][estimated_party] += 1

    if idx > num_to_score :
        break
```

[32]:
```
results
```

[32]:
```
defaultdict(<function __main__.<lambda>()>,
            {'Republican': defaultdict(int,
                         {'Republican': 3767, 'Democratic': 605}),
             'Democratic': defaultdict(int,
                         {'Republican': 4799, 'Democratic': 831})})
```

0.2.1 Reflections

The results tell us that classsifer is predicting tweets as Republican by a lot compared to predicting Democratic tweets and this makes sense as the actual party tweets are Republican as well.