Naive Bayes on Political Text

Renetta Nelson

June 5, 2023

In this notebook we use Naive Bayes to explore and classify political data. See the `README.md` for full details.

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

from nltk.corpus import stopwords
from string import punctuation

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

In [ ]:
```
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 [ ]:
```
# Place any addtional functions or constants you need here.

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

# Stopwords
```

```python
sw = stopwords.words("english")

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


# and now our functions
def descriptive_stats(tokens, num_tokens = 5, verbose=True) :
    """
        Given a list of tokens, print number of tokens, number of unique tokens,
        number of characters, lexical diversity, and num_tokens most common
        tokens. Return a list of
    """

    # Place your Module 2 solution here

    num_characters = 0
    for i in tokens:
        num_characters = num_characters + len(i)


    # Fill in the correct values here.
    num_tokens = len(tokens)
    num_unique_tokens = len(set(tokens))
    lexical_diversity = num_unique_tokens / num_tokens
    num_characters = num_characters #len(list(tokens)) #(nltk.FreqDist(nltk.Text(tokens))).N()

    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 the five most common tokens

    return([num_tokens, num_unique_tokens,
            lexical_diversity,
            num_characters])




def remove_stop(tokens) :
    # modify this function to remove stopwords
```

```
    new_tokens = []
    rmstop_tokens = tokens.split(" ")
    #print(rmstop_tokens)
    #rmstop_tokens = rmstop_tokens.lower()

    for word in rmstop_tokens:
        if word not in sw:
            #word.lower()
            #print(word)
            new_tokens.append(word)


    #return (t for t in rmstop_tokens if t.lower() not in sw)
    return(new_tokens)

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

def tokenize(text) :
    """ Splitting on whitespace rather than the book's tokenize function. That
        function will drop tokens like '#hashtag' or '2A', which we need for Twitter. """

    # modify this function to return tokens
    #for i in text:
    #    text = i.split(" ")

    return(text)

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

    for transform in pipeline :
        tokens = transform(tokens)

    return(tokens)
```

In [ ]:

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

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
                            WHERE speaker != "Unknown";

                            ''')

for row in query_results :
    # store the results in convention_data

    results = prepare(row[0], my_pipeline)

    convention_data.append([results, row[1]])
```

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

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

Out[ ]:
```
[[['good'], 'Democratic'],
 [['eating', 'freezer', 'grandma', 'doesn’t', 'see'], 'Democratic'],
 [['that’s',
   'first',
   'time',
   'president',
   'trump',
   'showed',
   'iowans',
   'rely',
   '2019',
   '100',
   'year',
   'floods',
   'breached',
   'nearly',
   'every',
   'levy',
   'devastated',
   'communities',
   'large',
   'small',
   'along',
   'missouri',
   'river',
   'iowa',
   'nebraska',
   'missouri',
   'president',
   'approved',
   'request',
   'aid',
   'record',
   'time',
   'two',
   'days',
   'well',
   'year',
   'less',
   '24',
   'hours',
   'whether',
   'it’s',
   'providing',
   'needed',
```

```
  'relief',
  'farmers',
  'target',
  'china’s',
  'unfair',
  'trade',
  'practices',
  'hammering',
  'new',
  'free',
  'fair',
  'trade',
  'deals',
  'fighting',
  'workers',
  'small',
  'businesses',
  'hit',
  'hard',
  'covid19',
  'president',
  'vice',
  'president',
  'get',
  'things',
  'done',
  'president',
  'trump',
  'leadership',
  'country',
  'able',
  'bounce',
  'back',
  'setbacks',
  'see',
  'opportunity',
  'grow',
  'thrive'],
 'Republican'],
[['three', 'bond', 'forged', 'sorrow', 'expanded', 'joy', 'jill', 'entered'],
 'Democratic'],
[['hoped',
  'convention',
  'city',
  'festivals',
```

```
  'milwaukee',
  'wisconsin',
  'year',
  'course',
  'we’re',
  'able',
  'we’ll',
  'hearing',
  'several',
  'wisconsin’s',
  'leaders',
  'throughout',
  'convention',
  'starting',
  'congresswoman',
  'gwen',
  'moore'],
 'Democratic'],
[['i’m',
  'mariska',
  'hargitay',
  'started',
  'research',
  'play',
  'detective',
  'olivia',
  'benson',
  'law',
  'order',
  'svu',
  '20',
  'years',
  'ago',
  'shocked',
  'find',
  'many',
  'people',
  'including',
  'children',
  'experienced',
  'physical',
  'sexual',
  'abuse',
  'statistics',
  'fueled',
```

```
  'resolve',
  'committed',
  'movement',
  'end',
  'violence'],
 'Democratic'],
[['everything', 'family', 'we’ve', 'always', 'done', 'everything', 'family'],
 'Democratic'],
[['i’m',
  'mother',
  'five',
  'grandmother',
  'nine',
  'i’m',
  'inaudible',
  '012131',
  'power',
  'speaker',
  'awesome',
  'awesome',
  'want',
  'go',
  'arena',
  'prepared',
  'take',
  'punch',
  'also',
  'prepared',
  'throw',
  'punch',
  'children'],
 'Democratic'],
[['arizona'], 'Democratic'],
[['people', 'say', '“we’re', 'oppressed', 'united', 'states”'], '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 [ ]:

```
word_cutoff = 5

tokens = [w for t, p in convention_data for w in t]

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 2360 as features in the model.
```

In [ ]:

```
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()

    text2 = text.split(" ")

    for words in text2:
          if words in fw:
               ret_dict[words] = True
          else:
               continue
```

```
    #print(ret_dict)
    return(ret_dict)




def conv_features2(text,fw) :

    ret_dict = dict()

    for words in text:
          if words in fw:
               ret_dict[words] = True
          else:
               continue

    return(ret_dict)
```

In [ ]:
```
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 [ ]:
```
featuresets = [(conv_features2(text,feature_words), party) for (text, party) in convention_data]
```

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

test_size = 500
```

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

```
0.52
```

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

```
Most Informative Features
                 radical = True           Republ : Democr =     38.0 : 1.0
                   media = True           Republ : Democr =     35.9 : 1.0
                   votes = True           Democr : Republ =     22.8 : 1.0
             enforcement = True           Republ : Democr =     18.4 : 1.0
                   crime = True           Republ : Democr =     17.2 : 1.0
                 destroy = True           Republ : Democr =     16.2 : 1.0
                freedoms = True           Republ : Democr =     16.2 : 1.0
                   china = True           Republ : Democr =     15.4 : 1.0
                  earned = True           Republ : Democr =     14.1 : 1.0
                  defund = True           Republ : Democr =     13.0 : 1.0
                  lowest = True           Republ : Democr =     13.0 : 1.0
              prosperity = True           Republ : Democr =     13.0 : 1.0
                   taxes = True           Republ : Democr =     12.8 : 1.0
                    mike = True           Republ : Democr =     12.2 : 1.0
                 beliefs = True           Republ : Democr =     12.0 : 1.0
                religion = True           Republ : Democr =     12.0 : 1.0
                 violent = True           Republ : Democr =     12.0 : 1.0
                      50 = True           Republ : Democr =     11.6 : 1.0
                 african = True           Republ : Democr =     10.9 : 1.0
                received = True           Republ : Democr =     10.9 : 1.0
                 climate = True           Democr : Republ =     10.9 : 1.0
                  record = True           Republ : Democr =     10.0 : 1.0
              countries = True            Republ : Democr =      9.9 : 1.0
                  enemy = True            Republ : Democr =      9.9 : 1.0
                   seem = True            Republ : Democr =      9.9 : 1.0
```

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

My Observations

From my observations, the top features are mainly found more in the Republican twees. There were only two times that one of the top features were found more in the Democratic tweets than the Republican.

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

In [ ]:
```
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 [ ]:
```
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 results :
    # store the results in convention_data

    results2 = prepare(row[0], my_pipeline)

    tweet_data.append([results2, row[1]])
```

In [ ]:
```
featuresets2 = [(conv_features2(text,feature_words), party) for (text, party) in tweet_data]


test_set2, train_set2 = featuresets2[:test_size], featuresets2[test_size:]
classifier2 = nltk.NaiveBayesClassifier.train(train_set2)
```

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

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

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

    featuresets2 = conv_features2(tweet, feature_words)

    estimated_party = classifier2.classify(featuresets2)
    # 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: ['jimmy', 'panetta']
Actual party is Democratic and our classifer says Democratic.

Here's our (cleaned) tweet: ['marcy', 'kaptur']
Actual party is Democratic and our classifer says Democratic.

Here's our (cleaned) tweet: ['debbie', 'wasserman', 'schultz']
Actual party is Democratic and our classifer says Democratic.

Here's our (cleaned) tweet: ['dave', 'brat']
Actual party is Republican and our classifer says Republican.

Here's our (cleaned) tweet: ['antonio', 'sabàto', 'jr']
Actual party is Republican and our classifer says Republican.

Here's our (cleaned) tweet: ['marcia', 'fudge']
Actual party is Democratic and our classifer says Democratic.

Here's our (cleaned) tweet: ['scott', 'peters']
Actual party is Democratic and our classifer says Republican.

Here's our (cleaned) tweet: ['mariah', 'phillips']
Actual party is Democratic and our classifer says Democratic.

Here's our (cleaned) tweet: ['jimmy', 'panetta']
Actual party is Democratic and our classifer says Democratic.

Here's our (cleaned) tweet: ['lucille', 'roybalallard']
Actual party is Democratic and our classifer says Democratic.
```

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

In [ ]:
```
# 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.

    featuresets2 = conv_features2(tweet, feature_words)

    # get the estimated party
    estimated_party = classifier2.classify(featuresets2)

    results[party][estimated_party] += 1

    if idx > num_to_score :
        break
```

In [ ]:
```
results
```

Out[ ]:
```
defaultdict(<function __main__.<lambda>()>,
            {'Republican': defaultdict(int,
                         {'Republican': 1166, 'Democratic': 3125}),
             'Democratic': defaultdict(int,
                         {'Republican': 571, 'Democratic': 5140})})
```

Reflections

For Republicans, the Democratic party was classified more than Republicans. For Democratics, the Democratic party was classfied the most.