SanjayRegiPhilip_Assignment4.1_PoliticalNaiveBayes

February 6, 2023

1 Assignment 4.1 by Sanjay Regi Philip

1.1 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
import pandas as pd

# functions to support text pattern functions
from string import punctuation
from nltk.corpus import stopwords
import re

# Feel free to include your text patterns functions
punctuation = set(punctuation) # speeds up comparison
tw_punct = punctuation - {"#"}

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. """

    text = text.split(" ")
    text = list(filter(str.strip, text)) ## removes unnecessary white space␣
↪characters
    return(text)

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

```
    for transform in pipeline :
        tokens = transform(tokens)
    return(tokens)
```

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

1.1.1 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 = []

# preprocess text for tokenization
my_pipeline = [str.lower, remove_punctuation]

# 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 :
    convention_data = list(map(list,query_results))

for item in convention_data:
    item[0] = prepare(item[0],my_pipeline)
```

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

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

[4]:
```
[['montana', 'Republican'],
 ['thank you', 'Democratic'],
 ['i’m mariann budde bishop of the episcopal diocese of washington dc and i’m
honored to offer the benediction tonight hear these words from pastor civil
rights leader and peace activist william sloane coffin "may god give you the
grace never to sell yourself short grace to do something big for something good
grace to remember that the world is too dangerous now for anything but truth and
too small for anything but love" and now may the blessing of god the source of
all goodness truth and love inspire you inspire us all to realize dr king’s
dream of the beloved community congressman lewis’s dream of a just society
```

```
president lincoln’s dream of a more perfect union in this country in our time
amen',
  'Democratic'],
 ['i’m senator chris coons from delaware a small state where people expect to
see their senators and even sometimes their vice president at the supermarket at
a church festival out in their community joe fights for us because he knows our
struggles and hopes he knows the pain of loss and the worries of working parents
and he’s always brought that same personal concern he showed for jacquelyn to
getting things done as our senator and then as president obama’s vice
president',
  'Democratic'],
 ['and he knows that even the united states of america needs friends on this
planet before donald trump we used to talk about american exceptionalism the
only thing exceptional about the incoherent trump foreign policy is that it has
made our nation more isolated than ever before joe biden knows we aren’t
exceptional because we bluster that we are we are exceptional because we do
exceptional things on june 6th 1944 young americans gave their lives and the
beaches of normandy to liberate the world from tyranny out of the ashes of that
war we made peace and rebuilt the world that was and remains exceptional it is
the opposite of everything donald trump stands for this moment is a fight for
the security of america and the world only joe biden can make america lead like
america again if you agree text join to 30330 thank you',
  'Democratic'],
 ['nebraska', 'Republican'],
 ['millions of people and veterans and senior citizens rely on the postal system
for prescription medicines for their checks',
  'Democratic'],
 ['i work at a meatpacking plant making sure grocery store shelves stay full
they call us essential workers but we get treated like we’re expendable workers
are dying from covid and a lot of us don’t have paid sick leave or even quality
protective equipment we are human beings not robots not disposable we want to
keep helping you feed your family but we need a president who will have our
backs nebraska cast 33 votes for our next president joe biden',
  'Democratic'],
 ['in the most difficult times is when we stand closest together it’s out of
tragedy that we go stronger',
  'Democratic'],
 ['my name is lakeisha cole i met my husband 20 years ago when we started dating
while i was in college once i graduated from college we eloped two weeks after
that he deployed',
  '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.

[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 2514 as features in the model.
```

[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
    text = tokenize(text) ## tokenize the text into list of tokens
    ret_dict = dict.fromkeys(text, "True") ## add tokens to dictionary
    return(ret_dict)
```

[7]:
```
## ensure functions work as expected
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.

[8]:
```
## create features using convention data
featuresets = [(conv_features(text,feature_words), party) for (text, party) in␣
 ↪convention_data]
```

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

test_size = 500
```

[10]:
```
test_set, train_set = featuresets[:test_size], featuresets[test_size:]
classifier = nltk.NaiveBayesClassifier.train(train_set)
print(nltk.classify.accuracy(classifier, test_set))
```

```
0.44
```

[11]:
```
classifier.show_most_informative_features(25)
```

```
Most Informative Features
                   china = 'True'          Republ : Democr =     39.9 : 1.0
             enforcement = 'True'          Republ : Democr =     35.8 : 1.0
                 radical = 'True'          Republ : Democr =     35.8 : 1.0
                   votes = 'True'          Democr : Republ =     24.6 : 1.0
                freedoms = 'True'          Republ : Democr =     18.1 : 1.0
                 destroy = 'True'          Republ : Democr =     16.1 : 1.0
                  prison = 'True'          Republ : Democr =     16.1 : 1.0
                   media = 'True'          Republ : Democr =     15.3 : 1.0
                   trade = 'True'          Republ : Democr =     15.2 : 1.0
                    army = 'True'          Republ : Democr =     14.0 : 1.0
                 beliefs = 'True'          Republ : Democr =     14.0 : 1.0
                 between = 'True'          Republ : Democr =     14.0 : 1.0
                    isis = 'True'          Republ : Democr =     14.0 : 1.0
                   crime = 'True'          Republ : Democr =     13.0 : 1.0
                 defense = 'True'          Republ : Democr =     13.0 : 1.0
                 liberal = 'True'          Republ : Democr =     13.0 : 1.0
                  within = 'True'          Republ : Democr =     13.0 : 1.0
               countries = 'True'          Republ : Democr =     11.9 : 1.0
                  defund = 'True'          Republ : Democr =     11.9 : 1.0
                    iran = 'True'          Republ : Democr =     11.9 : 1.0
                 violent = 'True'          Republ : Democr =     11.9 : 1.0
                 climate = 'True'          Democr : Republ =     11.2 : 1.0
                  earned = 'True'          Republ : Democr =     10.9 : 1.0
```

```
   patriots = 'True'          Republ : Democr =     10.9 : 1.0
  wonderful = 'True'          Republ : Democr =     10.9 : 1.0
```

1.1.2 My Observations

It appears that the classifier works by looking for words that are most distintictive in Republican speeches vs Democratic speeches. Many of the most informative words are known to be tied closely to issues that are brought up in Republican media including topics around China, voting, crime, corruption, and more. The most informative features listed here resonate to me with what is expected to be vocalized in Republican speeches.

1.2 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.

[12]:
```
cong_db = sqlite3.connect("congressional_data.db")
cong_cur = cong_db.cursor()
```

[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
```

[14]:
```
# 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.
tweet_data = [(prepare(sublist[2].decode(), my_pipeline), sublist[1]) for␣
 ↪sublist in results]
```

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…

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

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

[16]:
```
## used to create features for the new tweet sample data set
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)
```

[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: earlier today i spoke on the house floor abt
protecting health care for women and praised ppmarmonte for their work on the
central coast httpstcowqgtrzt7vv
Actual party is Democratic and our classifer says Republican.

Here's our (cleaned) tweet: go tribe #rallytogether httpstco0nxutfl9l5
Actual party is Democratic and our classifer says Democratic.

Here's our (cleaned) tweet: apparently trump thinks its just too easy for
students overwhelmed by the crushing burden of debt to pay off student loans
#trumpbudget httpstcockyqo5t0qh
Actual party is Democratic and our classifer says Republican.

Here's our (cleaned) tweet: we're grateful for our first responders our rescue
personnel our firefighters our police and volunteers who have been working
tirelessly to keep people safe provide muchneeded help while putting their own
lives on the line

httpstcoezpv0vmiz3
Actual party is Republican and our classifer says Republican.

Here's our (cleaned) tweet: let's make it even greater  #kag
httpstcoy9qozd5l2z
Actual party is Republican and our classifer says Republican.

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

```
Here's our (cleaned) tweet: congrats to belliottsd on his new gig at sd city
hall we are glad you will continue to serve… httpstcofkvmw3cqdi
Actual party is Democratic and our classifer says Republican.

Here's our (cleaned) tweet: we are really close we have over 3500 raised toward
the match right now whoot that’s 7000 for the nonmath majors in the room   help
us get there httpstcotu34c472sd httpstcoqsdqkypsmc
Actual party is Democratic and our classifer says Republican.

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

Here's our (cleaned) tweet: celebrated icseastla’s 22 years of eastside
commitment amp saluted community leaders at last night’s awards dinner
httpstco7v7gh8givb
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.

[18]:
```python
## create features for larger tweet tokens dataset
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)
```

[19]:
```python
# 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
```

[20]:
```
results
```

[20]:
```
defaultdict(<function __main__.<lambda>()>,
            {'Republican': defaultdict(int,
                         {'Republican': 3982, 'Democratic': 296}),
             'Democratic': defaultdict(int,
                         {'Republican': 5274, 'Democratic': 450})})
```

1.2.1 Reflections

The results show that this Naive Bayes classifier seems to perform well when predicting Republican texts but not very well at prediciting Democrat texts. This is supported by the fact that the accuracy is roughly 44%, Precision is roughly 93%, and Recall is 43%. This shows that the model performs well when prediciting for the positive class, which in this model is Republican, but not very well at prediciting the negative class of this model which is Democrat.