

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 [24]: import sqlite3
import nltk
import random
import numpy as np
from collections import Counter, defaultdict
import string

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

In [53]: nltk.download('stopwords')
nltk.download('punkt')

# define function to clean and tokenize text
def clean_tokenize(text):
    #tokenize text
    tokens = word_tokenize(text)

    #convert to lowercase
    tokens = [word.lower() for word in tokens]

    #remove punctuations
    tokens = [word for word in tokens if word not in string.punctuation]

    #remove stopwords
    stop_words = set(stopwords.words('english'))
    tokens = [word for word in tokens if word not in stop_words]

    return tokens

# define function to clean and tokenize tweets
def clean_tokenize_tweet(text):
    text = text.decode('utf-8')
    tokens = word_tokenize(text)
    tokens = [word.lower() for word in tokens if word.isalpha()]
    stop_words = set(stopwords.words('english'))
    tokens = [word for word in tokens if word not in stop_words]
    return tokens

# define function to convert text to feature dictionary
def conv_features(text, fw):
    words = text.split()
    ret_dict = {}
    for word in words:
        if word in fw:
            ret_dict[word] = ret_dict.get(word, 0) + 1

[nltk_data] Downloading package stopwords to
[nltk_data]   /Users/samantharivas/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to
[nltk_data]   /Users/samantharivas/nltk_data...
[nltk_data]   Package punkt is already up-to-date!

In [21]: convention_db = sqlite3.connect("2020_Conventions.db")
convention_cur = convention_db.cursor()

In [22]: convention_cur.execute("SELECT name FROM sqlite_master WHERE type='table';")

# retrieve table names
tables = convention_cur.fetchall()
for table in tables:
    print(table[0])

conventions
```

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 [25]: 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. As part of your cleaning process,
# remove the stopwords from the text. The second element of the sublist
# should be the party.

query_results = convention_cur.execute(
    """
    SELECT text, party FROM conventions
    """)

for row in query_results:
    text = row[0]
    party = row[1]

    #clean, tokenize, remove stopwords
    cleaned_text = clean_tokenize(text)
    cleaned_text = [word for word in cleaned_text if word not in stop_words]

    convention_data.append([cleaned_text, party])

convention_db.close()

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

```
In [26]: random.choices(convention_data, k=5)
```

```
Out[26]: [[['top',
'consider',
'markist',
'liberal',
'activist',
'leading',
'mob',
'neighborhood',
'stood',
'outside',
'home',
'bull',
'horn',
'screaming',
'',
'',
'',
'stop',
'revolution.',
'',
'weeks',
'later',
'markist',
'activist',
'democrat',
'nomination',
'hold',
'seat',
'us',
'house',
'representatives',
'city',
'st',
'louis',
'',
'winning',
'general',
'election',
'markist',
'revolutionary',
'going',
'congresswoman',
'first',
'district',
'missouri',
'radicals',
'content',
'marching',
'streets',
'want',
'walk',
'halls',
'congress',
'want',
'take',
'want',
'power',
'joe',
'biden',
'',
'party',
'people',
'charge',
'future',
'future',
'children',
'republican'],
['singin', 'Democratic'],
[['massachusetts', 'Republican'],
['jon',
'honor',
'devotion',
'showing',
'returning',
'citizens',
'forgotten',
'believe',
'person',
'made',
'god',
'purpose',
'continue',
'give',
'americans',
'including',
'former',
'innates',
'best',
'chance',
'build',
'new',
'life',
'achieve',
'american',
'dream',
'great',
'american',
'dream',
'',
'john',
'ask',
'john',
'richard',
'say',
'words'],
'republican'],
[['family',
'stopped',
'ranching',
'seven',
'years',
'ago',
'regulations',
'became',
'overbearing',
'ranch',
'slowly',
'sold'],
'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 [28]: word_cutoff = 5

tokens = [w for t, p in convention_data for w in t] #'.split()' removed

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 2330 as features in the model.
```

```
In [40]: 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 text into words
    words = text.split()
    # initialize empty dictionary for feature words
    ret_dict = {}
    # iterate through each word in text
    for word in words:
        #check word in set of features
        if word in fw:
            ret_dict[word] = True # add word to dict
    return ret_dict
```

```
In [41]: assert(len(feature_words)>0)
assert(conv_features("donald is the president", feature_words) ==
        {'donald': True, 'president': True})
assert(conv_features("some people in america are citizens", feature_words) ==
        {'people': True, 'america': 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 [43]: featuresets = [(conv_features(" ".join(text), feature_words), party) for (text, party) in convention_data]
```

```
In [44]: random.seed(20220507)
random.shuffle(featuresets)

test_size = 500
```

```
In [63]: test_set, train_set = featuresets[:test_size], featuresets[test_size:]
classifier = nltk.NaiveBayesClassifier.train(train_set)
accuracy = nltk.classify.accuracy(classifier, test_set)

print(f"Classifier accuracy on text data: {accuracy:.3f}")

Classifier accuracy on text data: 0.622
```

```
In [46]: classifier.show_most_informative_features(25)

Most Informative Features

          thank = True                Republ : Democr =    4.7 : 1.0
                yes = True            Democr : Republ =   34.1 : 1.0
    california = True                Republ : Democr =    1.6 : 1.0
    colorado = True                  Republ : Democr =    1.6 : 1.0
    georgia = True                   Republ : Democr =    1.6 : 1.0
        good = True                  Republ : Democr =    1.6 : 1.0
    indiana = True                   Republ : Democr =    1.6 : 1.0
        iowa = True                  Republ : Democr =    1.6 : 1.0
    kentucky = True                  Republ : Democr =    1.6 : 1.0
    louisiana = True                 Republ : Democr =    1.6 : 1.0
        maine = True                 Republ : Democr =    1.6 : 1.0
    mississippi = True               Republ : Democr =    1.6 : 1.0
    missouri = True                 Republ : Democr =    1.6 : 1.0
    montana = True                   Republ : Democr =    1.6 : 1.0
    ohio = True                     Republ : Democr =    1.6 : 1.0
    pennsylvania = True              Republ : Democr =    1.6 : 1.0
    tennessee = True                Republ : Democr =    1.6 : 1.0
        texas = True                 Republ : Democr =    1.6 : 1.0
        utah = True                  Republ : Democr =    1.6 : 1.0
    vermont = True                   Republ : Democr =    1.6 : 1.0
    virginia = True                  Republ : Democr =    1.6 : 1.0
    washington = True                Republ : Democr =    1.6 : 1.0
    wisconsin = True                 Republ : Democr =    1.6 : 1.0
    delaware = True                  Democr : Republ =    1.1 : 1.0
        singin = None                Republ : Republ =    1.0 : 1.0
```

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

My Observations

From the classifier, several observations can be made. For instance, the word 'thank' appears more often in Republican texts compared to Democratic texts (displayed by the 4.7:1.0 ratio). From this, it can be concluded that Republican speeches express greater gratitude than Democratic speeches. It can also be noted that some states, such as California, Colorado, and Georgia, are labeled as informative features, indicating that references to certain states are more prevalent in speeches than others. An interesting word that appears is 'singin', which might allude to a difference in tone or style of communication between the two parties.

Part 2: Classifying Congressional Tweets

In this part we apply the classifier 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 [47]: cong_cur = sqlite3.connect("congressional_data.db")
cong_cur = cong_cur.cursor()

In [48]: 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 [54]: tweet_data = []

# prepare tweet data
for row in results:
    candidate, party, tweet_text = row
    cleaned_text = clean_tokenize_tweet(tweet_text)
    tweet_data.append([cleaned_text, party])

# 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.
```

```
In [55]: # use naive bayes classifier trained on convention speeches to classify the tweets
tweet_featuresets = [(conv_features(" ".join(text), feature_words), party) for (text, party) in tweet_data]
```

There are a lot of tweets here. Let's take a random sample and see how our classifier does. I'm guessing it won't be too great given the performance on the convention speeches...

```
In [56]: random.seed(20201014)

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

```
In [60]: test_size = 500
test_tweet_set, train_tweet_set = tweet_featuresets[:test_size], tweet_featuresets[test_size:]

tweet_classifier = nltk.NaiveBayesClassifier.train(train_tweet_set)
tweet_accuracy = nltk.classify.accuracy(tweet_classifier, test_tweet_set)

print(f"Classifier accuracy on tweet data: {tweet_accuracy:.3f}")

Classifier accuracy on tweet data: 0.452
```

```
In [61]: tweet_classifier.show_most_informative_features(25)

Most Informative Features

    indigenous = True                Democr : Republ =   43.6 : 1.0
    indivisible = True               Democr : Republ =   34.1 : 1.0
    corporations = True              Democr : Republ =   33.7 : 1.0
        equality = True               Democr : Republ =   25.0 : 1.0
        unborn = True                Republ : Democr =   24.4 : 1.0
        womb = True                  Republ : Democr =   18.7 : 1.0
        hbcus = True                 Democr : Republ =   16.2 : 1.0
    inequality = True                Democr : Republ =   16.1 : 1.0
    childcare = True                 Democr : Republ =   15.9 : 1.0
        gender = True                Democr : Republ =   13.5 : 1.0
    vermont = True                   Democr : Republ =   12.7 : 1.0
    communism = True                 Republ : Democr =   11.5 : 1.0
    racial = True                     Democr : Republ =   10.9 : 1.0
    marched = True                   Democr : Republ =   10.9 : 1.0
    planet = True                     Democr : Republ =   10.6 : 1.0
    empathy = True                    Democr : Republ =   10.4 : 1.0
    pelosi = True                    Republ : Democr =   10.3 : 1.0
    marginalized = True              Democr : Republ =   10.0 : 1.0
        lord = True                  Republ : Democr =    9.9 : 1.0
    liberal = True                    Republ : Democr =    9.3 : 1.0
        beto = True                   Democr : Republ =    9.3 : 1.0
        equal = True                  Democr : Republ =    8.7 : 1.0
        digest = True                 Republ : Democr =    8.4 : 1.0
    decency = True                    Democr : Republ =    8.4 : 1.0
    baltimore = True                 Democr : Republ =    8.3 : 1.0
```

```
In [68]: for tweet, party in tweet_data_sample:
    tweet_features = conv_features(" ".join(tweet), feature_words)
    estimated_party = tweet_classifier.classify(tweet_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 classifier says {estimated_party}.")
    print("")
```

Here's our (cleaned) tweet: ['earlier', 'today', 'spoke', 'house', 'floor', 'abt', 'protecting', 'health', 'care', 'wome n', 'praised', 'pmpamonte', 'work', 'central', 'coast', 'https']
Actual party is Democratic and our classifier says Democratic.

Here's our (cleaned) tweet: ['go', 'tribe', 'rallytogether', 'https']
Actual party is Democratic and our classifier says Democratic.

Here's our (cleaned) tweet: ['apparently', 'trump', 'thinks', 'easy', 'students', 'overwhelmed', 'crushing', 'burden', 'd ebt', 'pay', 'student', 'loans', 'trumpbudget', 'https']
Actual party is Democratic and our classifier says Democratic.

Here's our (cleaned) tweet: ['grateful', 'first', 'responders', 'rescue', 'personnel', 'firefighters', 'police', 'volunte ers', 'working', 'tirelessly', 'keep', 'people', 'safe', 'provide', 'help', 'putting', 'lives', 'line', 'https']
Actual party is Republican and our classifier says Democratic.

Here's our (cleaned) tweet: ['let', 'make', 'even', 'greater', 'kag', 'https']
Actual party is Republican and our classifier says Democratic.

Here's our (cleaned) tweet: ['cavs', 'tie', 'series', 'repbarbaralee', 'scared', 'roadtovictory']
Actual party is Democratic and our classifier says Democratic.

Here's our (cleaned) tweet: ['congrats', 'belliottsd', 'new', 'gig', 'sd', 'city', 'hall', 'glad', 'continue', 'https']
Actual party is Democratic and our classifier says Democratic.

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

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', 'm ade', 'email', 'mail', 'https']
Actual party is Democratic and our classifier says Democratic.

Here's our (cleaned) tweet: ['celebrated', 'iceastla', 'years', 'eastside', 'commitment', 'amp', 'saluted', 'community', 'leaders', 'last', 'night', 'awards', 'dinner', 'https']
Actual party is Democratic and our classifier says Democratic.

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

```
In [69]: # dictionary of counts by actual party and estimated party.
# first key is actual, second is estimated
parties = {'Republican': defaultdict(int), 'Democratic': defaultdict(int)}
results = defaultdict(lambda: defaultdict(int))

for p in parties:
    for pl in parties:
        results[p][pl] = 0

num_to_score = 10000
random.shuffle(tweet_data)

for idx, tp in enumerate(tweet_data):
    tweet, party = tp

    # convert tweets to feature dictionary
    tweet_features = conv_features(" ".join(tweet), feature_words)
    # get estimated party
    estimated_party = tweet_classifier.classify(tweet_features)

    results[party][estimated_party] += 1

    if idx > num_to_score:
        break
```

```
In [70]: results

Out[70]: defaultdict(<function __main__.<lambda>()>,
                    {'Republican': defaultdict(int,
                    {'Republican': 1203, 'Democratic': 3169}),
                    'Democratic': defaultdict(int,
                    {'Republican': 520, 'Democratic': 5110})})
```

Reflections

From the classifier, several observations can be made. The accuracy on tweet data is 0.452, slightly better than random guessing, with a noticeable bias towards predicting tweets as Democratic. This suggests that the classifier struggles with the informal and brief nature of tweets. Informative features such as 'indigenous', 'indivisible', and 'equality' appear more often in Democratic tweets, while 'unborn', 'womb', and 'lord' are more common in Republican tweets. These distinctions highlight the specific issues and terminologies each party emphasizes. The findings underscore the importance of context-specific feature engineering. To improve accuracy and generalization, incorporating additional linguistic features and using advanced NLP techniques could be beneficial.