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. In [1]: import sqlite3 import nltk import random import numpy as np from nltk.corpus import stopwords from string import punctuation from collections import Counter, defaultdict # Some punctuation variations punctuation = set(punctuation) # speeds up comparison # Stopwords sw = stopwords.words("english") # Feel free to include your text patterns functions #from text_functions_solutions import clean_tokenize, get_patterns def remove stop(tokens) : # modify this function to remove stopwords tokens = [t for t in tokens if t not in sw and 'https' not in t] tokens = [t for t in tokens if t.isalpha() or t.isnumeric()] return tokens def remove_punctuation(text, punct_set=punctuation) : return("".join([ch for ch in text if ch==' ' or (ch not in punct set and ch.isalnum())])) def tokenize(text) : # modify this function to return tokens token = text.split() return token def prepare(text, pipeline) : tokens = str(text) for transform in pipeline : tokens = transform(tokens) return tokens my pipeline = [str.lower, remove punctuation, tokenize, remove stop] convention db = sqlite3.connect(r'C:\Users\gauth\Documents\Harini\San Diego University\applied text mining\2020 In [2]: 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 [3]: 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 for row in query results : # store the results in convention data text = row[0]text = ' '.join(prepare(text,my_pipeline)) convention data.append([text,row[1]]) Let's look at some random entries and see if they look right. In [4]: random.choices(convention data, k=10) [['military better equipped better resourced better manned military world president trump demolished terrorist Out[4]: isis caliphate middle east eliminated leader albaghdadi one worlds brutal terrorists president trump took decis ive action iranian terrorist mastermind gasem soleimani man responsible deaths hundreds american service men ir ag nato allies failed meet commitments upheld president trump demanded parody nato members increased contributi ons 100 billion year natos secretary general credits president donald j trump', 'Republican'], ['must interject one thing end chat said could favor said sure said tell get grandsons hair good humor sensiti ve years later vivid mind', 'Democratic'], ['please', 'Republican'], ['china would prefer joe biden', 'Republican'], ['born 15 heart condition known transposition greater arteries ended requiring transplant thankfully aca alrea dy taken effect thats something think time especially coverage potential coverage maxes initial hospital stay 3 million billed back insurance', 'Democratic'], ['washington dc welcome 2020 republican national convention tonight celebrating america land promise', 'Republican'], ['seen tragic shootings sides officers killing citizens citizens killing officers line duty police officers fe llow citizens live country families live communities unfortunately democrats turned backs brave police officers call defunding danger cities neighborhoods children isnt ironic democrat politicians personal security protect dont forego security replaced social workers especially since thats want police need funding less frequent psyc hological examinations nonlethal remote restraint technology deescalation use force training common sense solut ions president trump supports true sincere police reform thats right donald trump', 'Republican'], ['everything family weve always done everything family', 'Democratic'], ['time next year want see joe biden oval office', 'Democratic'], ['say crime time however time fair weve made mistakes none us want defined forever based worst decision prison became playwright mentor certified hospice volunteer ordained minister received special olympics event coordina tor year award work disabled women thing worse unjustly imprisoning body trying imprison mind transformation de scribed extraordinary truth thousands people like deserve opportunity come home never stopped fighting freedom christian faith prayers many kept hope alive', '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 [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 2373 as features in the model. def conv_features(text,fw) : In [6]: """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} 11 11 11 # Your code here ret dict = dict() for token in text.split(): if token in fw: ret dict[token] = True return(ret_dict) assert(len(feature words)>0) In [7]: 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. featuresets = [(conv features(text, feature words), party) for (text, party) in convention data] In [8]: In [9]: random.seed(20220507) random.shuffle(featuresets) test size = 500 In [10]: test_set, train_set = featuresets[:test_size], featuresets[test size:] classifier = nltk.NaiveBayesClassifier.train(train set) print(nltk.classify.accuracy(classifier, test set)) 0.498 In [11]: classifier.show_most_informative features(25) Republ: Democr = 25.8: 1.0
Democr: Republ = 23.8: 1.0
Republ: Democr = 21.5: 1.0
Republ: Democr = 19.2: 1.0
Republ: Democr = 18.2: 1.0
Democr: Republ = 17.8: 1.0
Republ: Democr = 17.1: 1.0
Republ: Democr = 16.1: 1.0
Republ: Democr = 14.9: 1.0
Republ: Democr = 13.0: 1.0
Republ: Democr = 10.9: 1.0 Most Informative Features china = True votes = True enforcement = True destroy = True freedoms = True climate = True supports = True crime = True media = True beliefs = True countries = True defense = True defund = True isis = True liberal = True religion = True trade = True flag = Truegreatness = True abraham = True drug = True department = True destroyed = True Republ : Democr = enemy = True 10.9 : 1.0 Republ : Democr = amendment = True 10.3 : 1.0 Write a little prose here about what you see in the classifier. Anything odd or interesting? **Observations** Most of the important features belongs to Republican party. • The model accuracy is ~50% only. China appears more in Rep vs. Dem. 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 [12]: cong db = sqlite3.connect(r'C:\Users\gauth\Documents\Harini\San Diego University\applied text mining\congression cong_cur = cong_db.cursor() In [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 from tqdm import tqdm In [14]: 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 tqdm(results) : # store the results in convention_data text = row[2]text = ' '.join(prepare(text,my_pipeline)) tweet data.append([text,row[1]]) 100%| | 664656/664656 [00:38<00:00, 1721 5.26it/s] 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 [15]: random.seed(20201014) tweet data sample = random.choices(tweet data, k=10) word cutoff = 0In [16]: 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) print(f"With a word cutoff of {word_cutoff}, we have {len(feature_words)} as features in the model.") With a word cutoff of 0, we have 119 as features in the model. In [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: bearlier today spoke house floor abt protecting health care women praised ppmarmont e work central coast Actual party is Democratic and our classifer says Republican. Here's our (cleaned) tweet: bgo tribe rallytogether Actual party is Democratic and our classifer says Democratic. Here's our (cleaned) tweet: bapparently 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 t irelessly keep people safe provide muchneeded help putting lives Actual party is Republican and our classifer says Republican. Here's our (cleaned) tweet: make even greater kag Actual party is Republican and our classifer says Republican. Here's our (cleaned) tweet: bwe cavs tie series 22 im repbarbaralee scared roadtovictory Actual party is Democratic and our classifer says Republican. Here's our (cleaned) tweet: bcongrats belliottsd new gig sd city hall glad continue Actual party is Democratic and our classifer says Republican. Here's our (cleaned) tweet: bwe really close 3500 raised toward match right whoot 7000 nonmath majors room help Actual party is Democratic and our classifer says Republican. Here's our (cleaned) tweet: btoday comment period plan expand offshore drilling opened public 60 days march 9 s hare oppose proposed program directly trump administration comments made email mail Actual party is Democratic and our classifer says Republican. Here's our (cleaned) tweet: bcelebrated 22 years eastside commitment amp saluted community leaders last awards 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. In [18]: # 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] = 0num to score = 10000random.seed(20220507) random.shuffle(tweet data) word cutoff = 5 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) print(f"With a word cutoff of {word cutoff}, we have {len(feature words)} as features in the model.") 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 With a word cutoff of 5, we have 47065 as features in the model.

results

4392

Reflections

defaultdict(<function main .<lambda>()>,

sum(results['Republican'].values())

sum(results['Democratic'].values())

{'Republican': defaultdict(int,

'Democratic': defaultdict(int,

For the Republican party, there are ~15% false predictions.

• For Democratic party, there are ~80% false predictions. The accuracy is much lower here.

{'Republican': 3678, 'Democratic': 714}),

{'Republican': 4620, 'Democratic': 990})})

In [19]:

Out[19]:

In [20]:

Out[20]:

In [21]:

Out[21]: