IS620 Web Analytics Final Project

A Better Sentiment Analysis System

Author: Partha Banerjee, CUNY MSDA

> **Final Project Requirement**
>
> "Your project should incorporate one or both of the two main themes of this course: network analysis and text processing. You need to show all of your work in a coherent workflow, and in a reproducible format, such as an IPython Notebook or an R Markdown document. If you are building a model or models, explain how you evaluate the “goodness” of the chosen model and parameters."

Preface

This project is to create a *model* which will be able to analyze the sentiment more accurately. The target is to train the model to predict “not cool” as negative instead of positive as predicted by majority of the models due to the positive word “cool”.

As you all know, sentiment analysis helps modern business many ways - a prospect buyers can use the sentiments of other buyers to decide about the product (s)he is planning to buy, producers can plan about their product lines based upon buyers sentiment, producers can take corrective measures to address negative sentiment about their product, marketers can use this for their research and recommendation etc. Social media like twitter, facebook play an important role to spread this sentiment very quickly.

In this project, I will first build the model and test its accuracy. Then I will get twitter data based upon user given hashtag and check their sentiment using the model. This project will comply the Project Requirements as follows:

- This is a text processing project.
- It is reproducible, written in IPython. I will share everything except my twitter credentials.
- I will also include the twitter data in .TXT format. This is to support the situation if twitter is unavailable (due to credentials etc.)
- Compare among different conditions and model's behavior.
- Finally, to deliver a model which can identify "not cool" as negative, not positive.

While working on this project, I got a good deal of help from the text book and the following sites:

- how to build a twitter sentiment analyzer? (http://ravikiranj.net/posts/2012/code/how-build-twitter-sentiment-analyzer/)
- Text Classification For Sentiment Analysis – Stopwords And Collocations (http://streamhacker.com/2010/05/24/text-classification-sentiment-analysis-stopwords-collocations/)

Now let's start without spending much time talking things.

Setup Environment

Let us start with setting up the environment with all necessary libraries in one place.

In [1]:
```
import re, math, collections, itertools, os
import nltk, nltk.classify.util, nltk.metrics
from nltk.classify import NaiveBayesClassifier
from nltk.metrics import BigramAssocMeasures
from nltk.probability import FreqDist, ConditionalFreqDist
from nltk.corpus import stopwords
from nltk.collocations import BigramCollocationFinder
```

Feature Evaluator

As we have learnt in our course, “features” are an important piece in sentiment analysis. Whatever someone is analyzing is an attempt to correlate the subject to the labels. In this code, the features will be the words in each review.

- The evaluate_features() function will splits up the review by line and then builds a two variables posFeatures and negFeatures containing the output of feature selection mechanism with ‘Positive’ or ‘Negative’ appended to it based upon the review file used.
- Then it separates the data into training and testing datasets for a Naive Bayes classifier which is the type of classifier I used.
- Now I train my classifier using NLTK's NaiveBayesClassifier and using my training dataset.
- Next test the classifier using test dataset by initiating two more variables - referenceSets and testSets. referenceSets will contain the actual values for the testing data and testSets will contain the predicted output.
- Each one of the testFeatures (the reviews that need testing) iterated through three things: an arbitrary identifier 'i', the features (or words) in the review, and the actual label (‘Positive’ or ‘Negative’).
- Finally display statistics of the model - accuracy, precision and recall. Referring NLTK
  - Accuracy: Given a list of reference values and a corresponding list of test values, return the fraction of corresponding values that are equal.
  - Precision: Given a set of reference values and a set of test values, return the fraction of test values that appear in the reference set.
  - Recall: Given a set of reference values and a set of test values, return the fraction of reference values that appear in the test set.
  - show_most_informative_features: Display top 25 features that were most helpful to the classifier in determining whether a review was positive or negative.
- And then return the classifier model.

Data Source

For building feature corpus, I have used *sentence polarity dataset v1.0 (http://www.cs.cornell.edu/people/pabo/movie-review-data/)* having 5,331 positive and 5,331 negative processed sentences / snippets introduced by Cornell professor Bo Pang in Pang/Lee ACL 2005 which was released in July 2005. Though this data collected on movie review, but we can still use this dataset to use for our purpose. The data is available freely for use after citation.

In [2]:
```
def evaluate_features(feature_fun):
    posFeatures = []
    negFeatures = []
    # http://stackoverflow.com/questions/367155/splitting-a-string-into-words-and-pu
nctuation
    # Breaks up the sentences into lists of individual words (as selected by the
    # input mechanism) and appends 'pos' or 'neg' after each list
    with open('./data/rt-polarity.pos', 'r') as posSentences:
        for i in posSentences:
            posWords = re.findall(r"[\w']+|[.,!?;]", i.rstrip())
            posWords = [feature_fun(posWords), 'Positive']
            posFeatures.append(posWords)
    with open('./data/rt-polarity.neg', 'r') as negSentences:
        for i in negSentences:
            negWords = re.findall(r"[\w']+|[.,!?;]", i.rstrip())
            negWords = [feature_fun(negWords), 'Negative']
            negFeatures.append(negWords)

    # Now we need to split the data into 80:20 ratio as training and
    # testing data for a Naive Bayes classifier.
    posCutoff = int(math.floor(len(posFeatures)*0.8))
    negCutoff = int(math.floor(len(negFeatures)*0.8))
    trainFeatures = posFeatures[:posCutoff] + negFeatures[:negCutoff]
    testFeatures = posFeatures[posCutoff:] + negFeatures[negCutoff:]

    # Trains a Naive Bayes Classifier using NLTK
    classifier = NaiveBayesClassifier.train(trainFeatures)

    # Initiates referenceSets and testSets
    referenceSets = collections.defaultdict(set)
    testSets = collections.defaultdict(set)

    # Puts correctly labeled sentences in referenceSets and the
    # predictively labeled version in testsets
    for i, (features, label) in enumerate(testFeatures):
        referenceSets[label].add(i)
        predicted = classifier.classify(features)
        testSets[predicted].add(i)

    # Display model statistics
    print 'Train on {:,} instances, Test on {:,} instances'.format( \
                            len(trainFeatures), len(testFeatures))
    print 'Accuracy:', nltk.classify.util.accuracy(classifier, testFeatures)
    print 'Positive precision:', nltk.metrics.precision(referenceSets['Positive'], \
                            testSets['Positive'])
    print 'Positive recall:', nltk.metrics.recall(referenceSets['Positive'], \
                            testSets['Positive'])
    print 'Negative precision:', nltk.metrics.precision(referenceSets['Negative'], \
                            testSets['Negative'])
    print 'Negative recall:', nltk.metrics.recall(referenceSets['Negative'], \
                            testSets['Negative'])
    print
    classifier.show_most_informative_features(25)

    # Finally return the model for future use
    return classifier
```

Baseline Bag of Words Feature Extraction

Now let us start with all the words in each review. The function make_complete_dict() builds a dictionary object that has each of the words in the review followed by ‘True’. Then run the test using the dictionary object.

In [3]:
```
def make_complete_dict(words):
    return dict([(word, True) for word in words])

monogramClassifier = evaluate_features(make_complete_dict)
```

```
Train on 8,528 instances, Test on 2,134 instances
Accuracy: 0.778819119025
Positive precision: 0.787996127783
Positive recall: 0.762886597938
Negative precision: 0.770208900999
Negative recall: 0.794751640112

Most Informative Features
              engrossing = True           Positi : Negati =     18.3 : 1.0
                  flaws = True           Positi : Negati =     13.7 : 1.0
                mediocre = True           Negati : Positi =     13.7 : 1.0
               absorbing = True           Positi : Negati =     13.0 : 1.0
                 generic = True           Negati : Positi =     13.0 : 1.0
                  boring = True           Negati : Positi =     12.4 : 1.0
              refreshing = True           Positi : Negati =     12.3 : 1.0
               inventive = True           Positi : Negati =     12.3 : 1.0
                    flat = True           Negati : Positi =     11.8 : 1.0
                    lame = True           Negati : Positi =     11.7 : 1.0
                 triumph = True           Positi : Negati =     11.7 : 1.0
            refreshingly = True           Positi : Negati =     11.7 : 1.0
                 routine = True           Negati : Positi =     11.0 : 1.0
              disturbing = True           Positi : Negati =     11.0 : 1.0
               affecting = True           Positi : Negati =     11.0 : 1.0
                 culture = True           Positi : Negati =     10.7 : 1.0
                touching = True           Positi : Negati =     10.7 : 1.0
                    dull = True           Negati : Positi =     10.6 : 1.0
                  stupid = True           Negati : Positi =     10.6 : 1.0
               wonderful = True           Positi : Negati =     10.6 : 1.0
             mesmerizing = True           Positi : Negati =     10.3 : 1.0
                   stale = True           Negati : Positi =     10.3 : 1.0
                chilling = True           Positi : Negati =     10.3 : 1.0
                mindless = True           Negati : Positi =     10.3 : 1.0
                provides = True           Positi : Negati =     10.2 : 1.0
```

The accuracy 77.88% is good, but we will try to find a better accuracy. The precisions and recalls are also pretty close to each other indicating that it is classifying everything fairly evenly.

From most informative features we see the chance of a word being positive vs. negative. For example, if ‘engrossing’ is in a review, there’s a 18:1 chance the review is positive.

Stopword Filtering

Let us now remove stopwords and see how the model behaves using accuracy parameters.

In [4]:
```
stopset = set(stopwords.words('english'))

def stopword_filtered_word_feats(words):
    return dict([(word, True) for word in words if word not in stopset])

nostopwordClassifier = evaluate_features(stopword_filtered_word_feats)
```

```
Train on 8,528 instances, Test on 2,134 instances
Accuracy: 0.77038425492
Positive precision: 0.766882516189
Positive recall: 0.77694470478
Negative precision: 0.773979107312
Negative recall: 0.763823805061

Most Informative Features
              engrossing = True           Positi : Negati =     18.3 : 1.0
                   flaws = True           Positi : Negati =     13.7 : 1.0
                mediocre = True           Negati : Positi =     13.7 : 1.0
               absorbing = True           Positi : Negati =     13.0 : 1.0
                 generic = True           Negati : Positi =     13.0 : 1.0
                  boring = True           Negati : Positi =     12.4 : 1.0
              refreshing = True           Positi : Negati =     12.3 : 1.0
               inventive = True           Positi : Negati =     12.3 : 1.0
                    flat = True           Negati : Positi =     11.8 : 1.0
                    lame = True           Negati : Positi =     11.7 : 1.0
                 triumph = True           Positi : Negati =     11.7 : 1.0
            refreshingly = True           Positi : Negati =     11.7 : 1.0
                 routine = True           Negati : Positi =     11.0 : 1.0
              disturbing = True           Positi : Negati =     11.0 : 1.0
               affecting = True           Positi : Negati =     11.0 : 1.0
                 culture = True           Positi : Negati =     10.7 : 1.0
                touching = True           Positi : Negati =     10.7 : 1.0
                    dull = True           Negati : Positi =     10.6 : 1.0
                  stupid = True           Negati : Positi =     10.6 : 1.0
               wonderful = True           Positi : Negati =     10.6 : 1.0
             mesmerizing = True           Positi : Negati =     10.3 : 1.0
                   stale = True           Negati : Positi =     10.3 : 1.0
                chilling = True           Positi : Negati =     10.3 : 1.0
                mindless = True           Negati : Positi =     10.3 : 1.0
                provides = True           Positi : Negati =     10.2 : 1.0
```

Accuracy has gone down from 77.88% to 77.03%. Also negative recall has gone down a bit. This is an indication that stopwords add information to sentiment analysis classification. So, we should not remove stopwords.

Bigram Collection

Let us now include bigrams to see the accuracy parameters. We will use NLTK library bigram features for this. The BigramCollocationFinder maintains 2 internal FreqDists -- one for individual word frequencies, another for bigram frequencies. Once it has these frequency distributions, it can score individual bigrams using a scoring function provided by BigramAssocMeasures, such chi-square. These scoring functions measure the collocation correlation of 2 words, basically whether the bigram occurs about as frequently as each individual word.

In [5]:
```
def bigram_word_features(words, score_fn=BigramAssocMeasures.chi_sq, n=200):
    bigram_finder = BigramCollocationFinder.from_words(words)
    bigrams = bigram_finder.nbest(score_fn, n)
    return dict([(ngram, True) for ngram in itertools.chain(words, bigrams)])

bigramClassifier = evaluate_features(bigram_word_features)
```

```
Train on 8,528 instances, Test on 2,134 instances
Accuracy: 0.787722586692
Positive precision: 0.792380952381
Positive recall: 0.779756326148
Negative precision: 0.783210332103
Negative recall: 0.795688847235

Most Informative Features
              engrossing = True           Positi : Negati =     18.3 : 1.0
                mediocre = True           Negati : Positi =     13.7 : 1.0
           (',', 'funny') = True           Positi : Negati =     13.7 : 1.0
                   flaws = True           Positi : Negati =     13.7 : 1.0
            ('dull', ',') = True           Negati : Positi =     13.7 : 1.0
               absorbing = True           Positi : Negati =     13.0 : 1.0
           ('to', 'care') = True           Negati : Positi =     13.0 : 1.0
            ('up', 'for') = True           Positi : Negati =     13.0 : 1.0
                 generic = True           Negati : Positi =     13.0 : 1.0
  ('examination', 'of') = True           Positi : Negati =     13.0 : 1.0
                 boring = True           Negati : Positi =     12.4 : 1.0
              refreshing = True           Positi : Negati =     12.3 : 1.0
               inventive = True           Positi : Negati =     12.3 : 1.0
          ('with', 'such') = True           Positi : Negati =     12.3 : 1.0
                    flat = True           Negati : Positi =     11.8 : 1.0
                    lame = True           Negati : Positi =     11.7 : 1.0
                triumph = True           Positi : Negati =     11.7 : 1.0
          ('the', 'title') = True           Negati : Positi =     11.7 : 1.0
              refreshingly = True           Positi : Negati =     11.7 : 1.0
            ('about', '.') = True           Negati : Positi =     11.0 : 1.0
                routine = True           Negati : Positi =     11.0 : 1.0
             disturbing = True           Positi : Negati =     11.0 : 1.0
              affecting = True           Positi : Negati =     11.0 : 1.0
               touching = True           Positi : Negati =     10.7 : 1.0
                culture = True           Positi : Negati =     10.7 : 1.0
```

I have tried several values for finding the optimal value for number of best bigrams to produce the result, but gotten the same value for the range between 50 and 500. I have settled using the 200 best bigrams from each file.

From the above, we can see that accuracy is now up from 77.88% to 78.77%. Also both positive and negative precision and recall values have increased. So we can conclude that including bigrams can increase classifier effectiveness.

Get into the Business

Now after seeing the benefits of bigram, let us start using our models to find the sentiments of tweets. But before extracting tweeter data, let me use a sample test data to see the results using the models.

In [6]:
```
# To prodice the sentiment of review comment using given model
def predictSentiment(tweet, classifier):
    twt = []
    for key, value in tweet.iteritems():
        twt.append(value)

    twFeatures = []
    for i in twt[0]:
        twWords = re.findall(r"[\w']+|[.,!?;]", i.rstrip())
        twWords = [bigram_word_features(twWords), 'tbd']
        twFeatures.append(twWords)

    for i, (features, label) in enumerate(twFeatures):
        predicted = classifier.classify(features)
        print "{} - {}".format(twt[0][i], predicted)
```

In [7]:
```
# Create a sample test data
test = {0: ["you look so cool", \
            "he looks not so cool", \
            "so cool", \
            "not so cool", \
            "so great", \
            "not so great"]}
```

Now, let me first use my model having no stopwords:

```
In [8]: predictSentiment(test, nostopwordClassifier)
```

```
you look so cool - Positive
he looks not so cool - Positive
so cool - Positive
not so cool - Positive
so great - Positive
not so great - Positive
```

And it failed to judge the sentiment correctly which we already thought based upon the statistics analysis above.

Now let us see how the model behave using bags of words.

```
In [9]: predictSentiment(test, monogramClassifier)
```

```
you look so cool - Positive
he looks not so cool - Negative
so cool - Positive
not so cool - Negative
so great - Positive
not so great - Positive
```

Much better result, though it has failed to predict "not so great" correctly.

Finally let us check how it behaves using the model with bigram:

```
In [10]: predictSentiment(test, bigramClassifier)
```

```
you look so cool - Positive
he looks not so cool - Negative
so cool - Positive
not so cool - Negative
so great - Positive
not so great - Negative
```

And bingo, it worked. The model is able to predict the sentiment more accurately.

Now let us do the final part of this project - get twitter data and run the prediction using bigram model.

Get Data from Twitter

Finally we need to have data for analyzing its sentiment and I extract the data from twitter. Data extraction is based upon the key word and time period. I have chosen them just for demonstrating my project.

In [11]:
```
import argparse, urllib, urllib2, json, random
import os, oauth2, datetime, re
from datetime import timedelta

class TwitterData:
    def __init__(self):
        self.currDate = datetime.datetime.now()
        self.weekDates = []
        self.weekDates.append(self.currDate.strftime("%Y-%m-%d"))
        for i in range(1,7):
            dateDiff = timedelta(days=-i)
            newDate = self.currDate + dateDiff
            self.weekDates.append(newDate.strftime("%Y-%m-%d"))

    def getTwitterData(self, keyword, time):
        self.weekTweets = {}
        if(time == 'lastweek'):
            for i in range(0,6):
                params = {'since': self.weekDates[i+1], 'until': \
                          self.weekDates[i]}
                self.weekTweets[i] = self.getData(keyword, params)
        elif(time == 'today'):
            for i in range(0,1):
                params = {'since': self.weekDates[i+1], 'until': \
                          self.weekDates[i]}
                self.weekTweets[i] = self.getData(keyword, params)
        return self.weekTweets

    def parse_config(self):
        config = {}
        if os.path.exists('config.json'):
            with open('config.json') as f:
                config.update(json.load(f))
        return config

    def oauth_req(self, url, http_method="GET", post_body=None,
                  http_headers=None):
        config = self.parse_config()
        consumer = oauth2.Consumer(key=config.get('consumer_key'), \
                                   secret=config.get('consumer_secret'))
        token = oauth2.Token(key=config.get('access_token'), \
                             secret=config.get('access_token_secret'))
        client = oauth2.Client(consumer, token)

        resp, content = client.request(
            url,
            method=http_method,
            body=post_body or '',
            headers=http_headers
        )
        return content

    def getData(self, keyword, params = {}):
        maxTweets = 50
        url = 'https://api.twitter.com/1.1/search/tweets.json?'
        data = {'q': keyword, 'lang': 'en', 'result_type': 'recent', \
                'count': maxTweets, 'include_entities': 0}

        if params:
            for key, value in params.iteritems():
                data[key] = value

        url += urllib.urlencode(data)

        response = self.oauth_req(url)
        jsonData = json.loads(response)

        tweets = []
        if 'errors' in jsonData:
            print "API Error"
            print jsonData['errors']
        else:
            for item in jsonData['statuses']:
                tweets.append(item['text'])
        return tweets
```

Get keyword and define time to extract data from twitter.

In [12]:
```
keyword = raw_input('Enter hash tag (with #) you want to retrieve? ')
```

```
Enter hash tag (with #) you want to retrieve? Trump
```

To make sure that the keyword is a hashtag and is in lower case.

In [13]:
```
if keyword[:1] != "#":
    keyword = "#" + keyword
keyword = keyword.lower()
print "Entered keyword:", keyword
```

```
Entered keyword: #trump
```

In [14]:
```
#keyword = '#Trump'
time = 'today'
twitterData = TwitterData()
tweets = twitterData.getTwitterData(keyword, time)
```

In [15]:
```
json.dump(tweets, open("./data/tweets.txt",'w'))
```

Data Cleanup

- Convert the tweets to lower case
- Remove Unicode
- Remove URLs
- Remove @usernames
- Remove additional white spaces

In [16]:
```
def processTweet(tweet):
    #Convert to lower case
    tweet = tweet.lower()
    #Remove unicode
    tweet = tweet.encode('ascii','ignore')
    #Remove www.* or https?://*
    tweet = re.sub('((www\.[^\s]+)|(https?://[^\s]+))',' ',tweet)
    tweet = tweet.replace("https://","")
    tweet = tweet.replace("https:","")
    #Remove @username
    tweet = re.sub('@[^\s]+',' ',tweet)
    #Remove additional white spaces
    tweet = re.sub('[\s]+', ' ', tweet)
    #Replace #word with word
    tweet = re.sub(r'#([^\s]+)', r'\1', tweet)
    #trim
    tweet = tweet.strip('\'"')
    #Finally remove rt from the begining
    if tweet[:3] == "rt ":
        tweet = tweet[3:]
    return tweet

processedTweet = {k: map(processTweet, v) for k, v in tweets.items()}
```

In [17]:
```
json.dump(processedTweet, open("./data/processedTweets.txt",'w'))
```

In [18]:
```
predictSentiment(processedTweet, bigramClassifier)
```

so any trump supporter could have this little coward arrested for making a death th
reat, correct? - Negative
 isn't it time to remove donald trump from the hall of fame considering the racist
rule and all? wwe trump - Negative
 donald trump live in iowa buildthewall trumpkin trumptrain trumptoday trumpmiami m
akeamericagreatagain trump - Positive
so any trump supporter could have this little coward arrested for making a death th
reat, correct? - Negative
nice try cont. new leftist anti-trump narrative they "care" abt gop "dilemma". spar
e the false "mournful state" pity. cnn trump2016 - Positive
rt pat buchanan's message to america! get behind trump!!! makeamericagreatagain -
Positive
so any trump supporter could have this little coward arrested for making a death th
reat, correct? - Negative
so any trump supporter could have this little coward arrested for making a death th
reat, correct? - Negative
intelligent analysis of trump + prospects for 2016 repub convention by for 4 min.
- Negative
oh boy: hoist on his own petard!! this should be fun! anonymous just declared war a
gainst donald trump - Negative
even die-hard hillaryclinton supporters can't think it's ok for trump to get more c
overage than all the democrats combined! democracy - Negative
left targets trump by finding nutty supporters--let's meet some obama & hillary
supporters - Negative
someone didn't like a trump campaign advertisement in cody, wyoming - Negative
anonyops: optrump: petition to ban trump from the uk now has over half a million si
gnatures! - Negative
to hillary clinton, donald trump is no longer a laughing matter trump will beat her
pants suit - Negative
sharpton has done zero for blacks and only enriched himself. now he's enraged black
ministers are warming to trump - Negative
left targets trump by finding nutty supporters--let's meet some obama & hillary
supporters - Negative
i typically vote for "none of the above", & antiestablishment & agenda. &am
p; right now trump is it! imvotingtrump2016 ht - Positive
left targets trump by finding nutty supporters--let's meet some obama & hillary
supporters - Negative
why dems and republicans are slamming trump? - Negative
so any trump supporter could have this little coward arrested for making a death th
reat, correct? - Negative
why dems and republicans are slamming trump? - Negative
we gave housesenate now they conspire to sabotage the will of the people's vote whe
n trump wins the nomination? - Negative
so any trump supporter could have this little coward arrested for making a death th
reat, correct? - Negative
that should make you stop talking abt trump and focus on true evil - Negative
palin blasts msm herd, naive pundits for opposing trumps temp muslim ban radicalisl
am - Negative
trump doesn't mind rich muslims... - Negative
so any trump supporter could have this little coward arrested for making a death th
reat, correct? - Negative
or maybe twitter should make this punk famous trump - Positive
this is what i have felt about how media is handling trump anti muslim rants. bad n
ews! - Negative
this is unexpected. moderate texas imam says he supports trump's proposal; then for
ced to resign from his mosque. http - Negative
palin blasts msm herd, naive pundits for opposing trumps temp muslim ban radicalisl
am - Negative
trump remarks 2 banmuslims r causin white zionazis 2 attack us in usa may a white z
ionazi return the favor on trump bantrump 2 speak - Negative
trump donaldchimp (apology to chimps) - Positive
 donald trump live in iowa trump trumpdemands trump2016 women4trump needtrumpnow un
itedwestand - Positive
i have hope now for this country! hollywood/media continue to lie about trump yet t
he people aren't listening anymore! - Negative
why i love america: because men like this make the world a better place, and this i
s who we are. trump cruz - Positive
 so any trump supporter could have this little coward arrested for making a deathth
reat, correct? - Negative
specialreport the gope is way off like always! trump is going to win....... - Nega
tive
i've never been so disgraced & disappointed w our govt & our worlds in cris
is & all they care abt is taking dwn t - Negative
left targets trump by finding nutty supporters--let's meet some obama & hillary
supporters - Negative
is the republican convention going to be open carry? gop trump gopwar - Negative
so any trump supporter could have this little coward arrested for making a death th
reat, correct? - Negative
left targets trump by finding nutty supporters--let's meet some obama & hillary
supporters - Negative
franklin graham behind trump on muslim ban trumpisright bit me gop i support tedcru
z islamistheproblem - Negative

```
so any trump supporter could have this little coward arrested for making a death th
reat, correct?  - Negative
this is unexpected. moderate texas imam says he supports trump's proposal; then for
ced to resign from his mosque. http - Negative
so any trump supporter could have this little coward arrested for making a death th
reat, correct?  - Negative
i guess that police order in nh that just endorsed trump is gonna be vanguard of hi
s brownshirts? disgusting!!.  - Negative
mark shields on ted cruz=donald trump + 60 iq points + better haircut+better academ
ic credentials - Negative
```

Conclusion

It appears that the bigram hypothesis is correct, including significant bigrams can increase classifier effectiveness. While this produces the result of my stated goal, I am sure it needs further improvement since the accuracy is still at 78.77%. To improve, I definitely want to get a relevant source of positive and negative data sources for training my model. Also I want to spend some more time to find the optimal level of bigram values to use (In my case, I have taken n=200). I am also interested to see the use of trigram, but this is a future plan.

Overall, I am satisfied that I have been able to produce the result I was looking for. This concludes my project.