Text as Data: Homework 1

Jeff Ziegler

August 15, 2017

In this homework assignment we're going to analyze the first presidential debate from the 2012 election.

Problem 1

To analyze the debate, we first need to load the debate and parse the content. On the coursewebsite, you'll find the file `debate1.html`*. Download the file and open it in a browser. We will use* `BeautifulSoup` *to parse HTML file containing the debate transcript.*

- *Load the webpage into* `Python` *and use* `BeautifulSoup` *to create a searchable version of the debate. What tags can you use to identify statements?*

```
# import libraries
from bs4 import BeautifulSoup
from urllib import urlopen
import re
import os
import csv

# load .html file in gitHub folder (folder location will differ by user)
# find statements within <p> in HTML
pageText = BeautifulSoup(open('Documents/Git/WUSTL_textAnalysis/Debate1.
    html')).findAll('p')
```

- *Note that not all of the statements contain information about the speaker. Devise a rule to assign the unlabeled statements to speakers. For substantive reasons, we would like to define a single statement as any uninterrupted speech from a candidate. We'll say a candidate is interrupted when the transcript says that a new speaker has begun. In other words, cross talk doesn't count as an interruption. Create a list with just the text (not the tags) of each statement as an element. Some statements are split among several tags; these will need to be concatenated according to the rule you devised above. Remember to filter out notes about audience behavior.*

```
# we know that there are three speakers
# Speakers: FORMER GOV. MITT ROMNEY, R-MASS; PRESIDENT BARACK OBAMA;
# JIM LEHRER, MODERATOR
# but even if we didn't know which names to search for, it appears
# they are labeled by all caps
# which is how we'll identify who is speaking and speaker changes

# create empty vector to be filled with statements
statements = []
# prior speaker is set to NULL, but will be filled with the most
# recent speaker
priorSpeaker = ''

# iterate over each text block (excluding the introduction and ending)
for i in pageText[6:477]:
  # first, convert all <p> from bs4 object to strings to be searched
  # and get rid of HTML in strings
  # '\' will appear, but it's just to escape the apostrophes
    cleanedStatements = re.sub(re.compile('<.*?>'), '', str(i))
    # then check if there is a fully capitalized word at the beginning
    # of each statement
    speakerLabelled = re.search('^[A-Z]+:', cleanedStatements)
    # and if there is...
    if speakerLabelled:
      # record who the current speaker is (by checking which portion
      # of the string matched the regex))
      currentSpeaker = speakerLabelled.group()
      # if the current speaker matches the prior speaker, add cleaned
      # statement to the last full statement that was added
      # and remove current speaker from every other statement
      # except the first
        if currentSpeaker == priorSpeaker:
          # since no index is specified, .pop() removes and returns
          # the last item in the list
          # Note: there is an extra space added because otherwise
          # append will crunch words together
            statements.append(statements.pop() + " " +
            cleanedStatements.replace(currentSpeaker, ''))
        # if the current speaker is different than prior speaker,
        # add cleaned statement on its own
        else:
            statements.append(cleanedStatements)
            # and reset prior speaker to the most recently recorded speaker
            priorSpeaker = speakerLabelled.group(0)
    # if there is no speaker listed (does not match regex search),
    # add cleaned statement to the last full statement that was added
    else:
      statements.append(statements.pop() + " " + cleanedStatements)
```

Problem 2

Now we're going to do some more preprocessing to create a dataset that includes useful information about our texts. We will use a curated dictionary list from Neal Caren. The positive *words are at* `http://www.unc.edu/~ncaren/haphazard/positive.txt` *and the negative words are at* `http://www.unc.edu/~ncaren/haphazard/negative.txt`*.*

- Load the positive and negative words into `python`. Use the `porter`, `snowball` and `lancaster` stemmers from the `nltk` package to create stemmed versions of the dictionaries.

```python
# create function to load sentimental dictionaries
def loadWords(type, stemmer):
  # open url specifying positive or negative dictionary
  url = urlopen('http://www.unc.edu/~ncaren/haphazard/' + type + '.txt').read()
  # since they are in .txt files, we need to split each word
  # create the unstemmed dictionary
  unstemmedDict = url.split('\n')
  # determine which stemmer should be used
  # (1) Porter
  if stemmer=="Porter":
    # for each word in dictionary, stem
    stemmedDict = [nltk.stem.PorterStemmer().stem(word) for word in unstemmedDict]
  # (2) Snowball
  elif stemmer=="Snowball":
    # for each word in dictionary, stem
    stemmedDict = [nltk.stem.SnowballStemmer('english').stem(word) for word in unstemmedDict]
  # (3) Lancaster
  elif stemmer=="Lancaster":
    stemmedDict = [nltk.stem.LancasterStemmer().stem(word) for word in unstemmedDict]
  else:
    stemmedDict = unstemmedDict
  # return both stemmed and unstemmed dictionaries
  return [unstemmedDict, stemmedDict]

# get basic positive and negative, unstemmed dictionaries
positiveWords = loadWords('positive', stemmer="None").pop(0)
negativeWords = loadWords('negative', stemmer="None").pop(0)

# run dictionary acquisition and stemming function for all stemmers
# (1) Porter
stemmedPositivePorter = loadWords('positive', stemmer="Porter").pop(1)
stemmedNegativePorter = loadWords('negative', stemmer="Porter").pop(1)

# (2) Snowball
stemmedPositiveSnowball = loadWords('positive', stemmer="Snowball").pop(1)
stemmedNegativeSnowball = loadWords('negative', stemmer="Snowball").pop(1)

# (3) Lancaster
```

```
stemmedPositiveLancaster = loadWords('positive', stemmer="Lancaster").pop(1)
stemmedNegativeLancaster = loadWords('negative', stemmer="Lancaster").pop(1)
```

- *Using the original and stemmed dictionaries, we're going to create a statement by statement data set of the speech. The data set should have the following columns:*

  1) *Statement number (place in debate)*
  2) *Speaker*
  3) *Number of non-stop words spoken*
  4) *Number of positive words*
  5) *Number of negative words*
  6) *Number of lancaster stemmed positive words*
  7) *Number of lancaster stemmed negative words*
  8) *Number of porter stemmed positive words*
  9) *Number of porter stemmed negative words*
  10) *Number of snowball stemmed positive words*
  11) *Number of snowball stemmed negative words*

To create the data set, create a set of nested dictionaries that map each statement in the list created in Problem 1 to the each of the attributes described above. To calculate the values for items 3 - 11 above, you'll need to do the following to each statement:

- *Discard punctuation*
- *Remove capitalization*
- *Remove stop words with the list of words provided here:*
  `‘http://jmlr.org/papers/volume5/lewis04a/a11-smart-stop-list/english.stop’`
- *Tokenize the words*
- *Apply each of the stemmers, determining which of the words appear in the corresponding stemmed dictionaries*

Write your dataset as a .csv file and save it to a working directory. Turn it in with your homework.

```
# create function that will easily check how many words are in
# corresponding dictionary list
def wordCount(inputStatement, dictionaries):
  return len([x for x in inputStatement if x in dictionaries])
# create function to pull necessary info from each statement
```

```python
def statementInfo(statement, documentContent, count):
  # first, need to discard punctuation
  removedPunctuation = re.sub("\W", " ", i)
  # capitalization
  removedCaps = removedPunctuation.lower()
  # and tokenization
  reducedStatements = nltk.word_tokenize(removedCaps)

    # append documentContent with relevant info
  documentContent.append({
  # add to statementIter
  "statementNumber":  count,
  "speaker":re.search('^[A-Z]+', statement).group(),
  # record the number of ___ in statements w/ no punctuation, caps,
  # and reduced tokens:
  # non-stop words
  #"NstopWords": len([x for x in reducedStatements if x not in stop_words]),
  # number of positive words
  "NposWords": wordCount(reducedStatements, positiveWords),
    # number of negative words
    "NnegWords": wordCount(reducedStatements, negativeWords),
    # number of words in each positive and negative using:
    # (1) Porter stem
    "NposPorter": wordCount([nltk.stem.PorterStemmer().stem(y) for y in reducedStatements], stemmedPositivePorter),
    "NnegPorter": wordCount([nltk.stem.PorterStemmer().stem(y) for y in reducedStatements], stemmedNegativePorter),
    # (2) Snowball stem
    "NposSnowball": wordCount([nltk.stem.SnowballStemmer('english').stem(y) for y in reducedStatements], stemmedPositiveSnowball),
    "NnegSnowball": wordCount([nltk.stem.SnowballStemmer('english').stem(y) for y in reducedStatements], stemmedNegativeSnowball),
    # (3) Lancaster stem
    "NposLancaster": wordCount([nltk.stem.LancasterStemmer().stem(y) for y in reducedStatements], stemmedPositiveLancaster),
    "NnegLancaster": wordCount([nltk.stem.LancasterStemmer().stem(y) for y in reducedStatements], stemmedNegativeLancaster)})

# create empty list to fill with statement info
statementCharacteristics = []
# begin document iterations at 0
statementIter = 0
for i in statements:
  # execute statementInfo function for each statement
  # begin document iterations at 1
  statementIter +=1
  statementInfo(i, statementCharacteristics, count=statementIter)

# with data now assigned to dictionary
# write content to .csv
with open('Documents/Git/WUSTL_textAnalysis/statmentInfo.csv', 'wb') as f:
    w = csv.DictWriter(f, fieldnames=("statementNumber", "speaker",
    "NposWords", "NnegWords", "NposPorter", "NnegPorter",
    "NposSnowball", "NnegSnowball", "NposLancaster", "NnegLancaster"))
    w.writeheader()
    for item in statementCharacteristics:
      w.writerow(item)
```

Problem 3

Using our new data set, let's make some observations about the debate

- Load the data into `R`
- Create a visualization that compares the overall positive and negative word rate for Obama, Romney, and Lehrer. What patterns do you notice? There is no one right answer, be creative!
- Using your data set, examine trends in each candidate's statements and Lehrer's speeches. Do you notice any
  - i) Trends in the measured tone?
  - ii) Response to the other candidate's tone (examining who spoke previously)?
  - iii) Overall interesting patterns? (this is an intentionally vague question)