Text as Data: Homework 1

Jeff Ziegler

August 16, 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?*

```
1  # import libraries
2  from bs4 import BeautifulSoup
3  from urllib import urlopen
4  import re
5  import os
6  import csv
7
8  # load .html file in gitHub folder (folder location will differ by user)
```

- *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.*

```
1
2  # load .html file in gitHub folder (folder location will differ by user)
3  # find statements within <p> in HTML
4  pageText = BeautifulSoup(open('Documents/Git/WUSTL_textAnalysis/Debate1.
       html')).findAll('p')
5
```

```python
 6  # we know that there are three speakers
 7  # Speakers: FORMER GOV. MITT ROMNEY, R-MASS; PRESIDENT BARACK OBAMA;
 8  # JIM LEHRER, MODERATOR
 9  # but even if we didn't know which names to search for, it appears
10  # they are labeled by all caps
11  # which is how we'll identify who is speaking and speaker changes
12
13  # create empty vector to be filled with statements
14  statements = []
15  # prior speaker is set to NULL, but will be filled with the most
16  # recent speaker
17  priorSpeaker = ''
18
19  # iterate over each text block (excluding the introduction and ending)
20  for i in pageText[6:477]:
21    # first, convert all <p> from bs4 object to strings to be searched
22    # and get rid of HTML in strings
23    # '\' will appear, but it's just to escape the apostrophes
24      cleanedStatements = re.sub(re.compile('<.*?>'), '', str(i))
25      # then check if there is a fully capitalized word at the beginning
26      # of each statement
27      speakerLabelled = re.search('^[A-Z]+:', cleanedStatements)
28      # and if there is...
29      if speakerLabelled:
30        # record who the current speaker is (by checking which portion
31        # of the string matched the regex))
32        currentSpeaker = speakerLabelled.group()
33        # if the current speaker matches the prior speaker, add cleaned
34        # statement to the last full statement that was added
35        # and remove current speaker from every other statement
36        # except the first
37          if currentSpeaker == priorSpeaker:
38            # since no index is specified, .pop() removes and returns
39            # the last item in the list
40            # Note: there is an extra space added because otherwise
41            # append will crunch words together
42              statements.append(statements.pop() + " " +
43              cleanedStatements.replace(currentSpeaker, ''))
44        # if the current speaker is different than prior speaker,
45        # add cleaned statement on its own
46          else:
47              statements.append(cleanedStatements)
48              # and reset prior speaker to the most recently recorded
      speaker
49              priorSpeaker = speakerLabelled.group(0)
50      # if there is no speaker listed (does not match regex search),
51      # add cleaned statement to the last full statement that was added
52      else:
53        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
### Problem 2

# 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
  # by using set() instead of keeping them as lists
  # removes duplicates
  return [unstemmedDict, set(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)
```

```
37
38 # (2) Snowball
39 stemmedPositiveSnowball = loadWords('positive', stemmer="Snowball").pop
       (1)
40 stemmedNegativeSnowball = loadWords('negative', stemmer="Snowball").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.

```python
1  # create function that will easily check how many words are in
2  # corresponding dictionary list
3  def wordCount(inputStatement, dictionaries):
4      return len([x for x in inputStatement if x in dictionaries])
5  # create function to pull necessary info from each statement
6  def statementInfo(statement, documentContent, count):
7      # first, need to discard punctuation
8      removedPunctuation = re.sub("\W", " ", i)
9      # capitalization
10     removedCaps = removedPunctuation.lower()
11     # and tokenization
12     reducedStatements = nltk.word_tokenize(removedCaps)
13
14       # append documentContent with relevant info
15     documentContent.append({
16     # add to statementIter
17     "statementNumber":   count,
18     "speaker":re.search('^[A-Z]+', statement).group(),
19     # record the number of ___ in statements w/ no punctuation, caps,
20     # and reduced tokens:
21     # non-stop words
22     #"NstopWords": len([x for x in reducedStatements if x not in stop_words]),
23     # number of positive words
24     "NposWords": wordCount(reducedStatements, positiveWords),
25      # number of negative words
26      "NnegWords": wordCount(reducedStatements, negativeWords),
27      # number of words in each positive and negative using:
28      # (1) Porter stem
29      "NposPorter": wordCount([nltk.stem.PorterStemmer().stem(y) for y in
       reducedStatements], stemmedPositivePorter),
30      "NnegPorter": wordCount([nltk.stem.PorterStemmer().stem(y) for y in
       reducedStatements], stemmedNegativePorter),
31      # (2) Snowball stem
32      "NposSnowball": wordCount([nltk.stem.SnowballStemmer('english').stem(y)
       for y in reducedStatements], stemmedPositiveSnowball),
33      "NnegSnowball": wordCount([nltk.stem.SnowballStemmer('english').stem(y)
       for y in reducedStatements], stemmedNegativeSnowball),
34      # (3) Lancaster stem
35      "NposLancaster": wordCount([nltk.stem.LancasterStemmer().stem(y) for y in
       reducedStatements], stemmedPositiveLancaster),
36      "NnegLancaster": wordCount([nltk.stem.LancasterStemmer().stem(y) for y in
       reducedStatements], stemmedNegativeLancaster)})
37
38 # create empty list to fill with statement info
39 statementCharacteristics = []
40 # begin document iterations at 0
41 statementIter = 0
42 for i in statements:
43     # execute statementInfo function for each statement
44     # begin document iterations at 1
45     statementIter +=1
```

```
46    statementInfo(i, statementCharacteristics, count=statementIter)
47
48  # with data now assigned to dictionary
49  # write content to .csv
50  with open('Documents/Git/WUSTL_textAnalysis/statmentInfo.csv', 'wb') as f:
51      w = csv.DictWriter(f, fieldnames=("statementNumber", "speaker",
52      "NposWords", "NnegWords", "NposPorter", "NnegPorter",
53      "NposSnowball", "NnegSnowball", "NposLancaster", "NnegLancaster"))
54      w.writeheader()
55      for item in statementCharacteristics:
56        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)