

# Text as Data: Homework 3

August 19, 2017

In this homework we will analyze a collection of news stories from the New York Times from the November 1-3, 2004 (the day before, of, and after the 2004 general election). This data come from the New York Times Annotated Corpus and is for academic use only. We have done some preprocessing in order to simplify the homework tasks.

## 1 Preprocessing and Creating a Document-Term Matrix

- a) From the course github, download `nyt_ac.json`
- b) Using the `JSON` library in python, import the data. Use `type` to explore the structure of this data. How are this data organized?
- c) Extract the title and text from each story. Create an individual document for each story and write each of the files to a new directory
- d) Using the loaded `json` file, create a document term matrix of the 1000 most used terms. Be sure to:
  - Discard word order
  - Remove stop words
  - Apply the porter stemmer
- e) Include in your document-term matrix the *desk* from which the story originated, which we will include later

```
1 # import libraries
2 import json
3 import nltk
4 from nltk.corpus import stopwords
5 import re
6 import os
7 from urllib import urlopen
8
```

```

9 # a) download {nyt-ac.json} from gitHub
10 with open('Documents/Git/WUSTL.textAnalysis/nyt-ac.json') as data_file:
11     NYTjson = json.load(data_file)
12
13 # b) inspect the structure of the data
14 type(NYTjson)
15 # shows that json object is a list
16 # check the first elements as example
17 NYTjson[0].keys()
18 # two initial dictionaries (body, meta)
19 NYTjson[0]['body'].keys()
20 # within body, info like title and text
21
22 # c) write each story into new .txt file
23 for story in range(0, len(NYTjson)):
24     # give title based on order
25     with open('Documents/Git/WUSTL.textAnalysis/NYTstories/' +
26              str(story) + '.txt', 'w') as file:
27         # since we know where the relevant info is stored
28         # use dump to write json as string by key
29         file.write(json.dumps(NYTjson[story]['body']))
30
31 # check that files wrote properly
32 os.listdir('Documents/Git/WUSTL.textAnalysis/NYTstories/')

```

```

['0.txt',
 '1.txt',
 '10.txt',
 '100.txt',
 '101.txt',
 '102.txt',
 '103.txt',
 ...]

```

```

1
2 # d) task: Using NYTjson file , and create a DIM
3 # of the 1000 most used terms
4 # be sure to:
5 # discard word order
6 # remove stop words
7 # apply porter stemmer
8
9 # create function to use Porter stemmer
10 def porterStem(unstemmedList):
11     return [nlk.stem.PorterStemmer().stem(words) for words in unstemmedList]
12
13 # load a set of stop words from nlkt
14 # with the other stop word additions
15 stopWords = stopwords.words('english')

```

```

16 # apply Porter stemmer to stop words
17 stopWords = porterStem(stopWords)
18
19 # edit the text of pressReleases
20 for story in range(0, len(NYTjson)):
21     # create key in each NYT story with cleaned text
22     NYTjson[story]['body']['cleanText'] = []
23     # remove capitalization
24     textTokens = NYTjson[story]['body']['body_text'].lower()
25     # discard punctuation by removing non-word characters
26     textTokens = re.sub('\W', ' ', textTokens)
27     # and apply Porter stem to tokenized story
28     textTokens = porterStem(nltk.word_tokenize(textTokens))
29     # remove stop words
30     textTokens = [x for x in textTokens if x not in stopWords]
31     # then append unigram tokens
32     NYTjson[story]['body']['cleanText'] = textTokens
33
34 # now that we have cleaned text, need to create DIM
35 # create empty dictionaries to be filled with counts of unigrams
36 unigramDict = {}
37
38 # for each story
39 for story in range(0, len(NYTjson)):
40     # add counts to totals
41     for word in NYTjson[10]['body']['cleanText']:
42         if word not in unigramDict:
43             unigramDict[word] = 1
44         else:
45             unigramDict[word] += 1
46
47
48 # sort unigrams and trigrams into new lists
49 mostNunigrams = []
50
51 # create function to take the most used words
52 def extractTopN(topsList, mostNgrams):
53     # loop over dictionary and append new list
54     # by value, rather than key
55     # then sort list
56     return sorted(topsList, key=topsList.get, reverse=True)
57
58 # extract the most used unigrams
59 mostNunigrams = extractTopN(unigramDict, mostNunigrams)
60 # take only the top 1000
61 mostNunigrams = mostNunigrams[:1000]
62
63 # now that we have the most 1000 used unigrams
64 # create DIM matrix and write it to .csv
65 # task: we need to check whether each of the top 1000 words
66 # is in each NYT story, and count their frequency

```

```

67 # will iterate over each NYT story
68 # first, open up .csv writer
69 with open('Documents/Git/WUSTL-textAnalysis/NYTstoriesDTM.csv', 'wb') as f:
70     w = csv.writer(f)
71     # create header to be written to .csv as variable names
72     # 1st column is desk name, preceding columns
73     # represent top 1000 unigrams
74     csvHeader = mostNunigrams
75     csvHeader.insert(0, 'desk')
76     # write header first
77     w.writerow(csvHeader)
78     # then, we need to create counts of all the words
79 # in each document (NOT across authors or
80 # documents like the previous problem)
81 # so, for each story
82     for story in range(0, len(NYTjson)):
83         # create a clear row
84         rowEntry = []
85         # and give each unigram count in that story
86         for unigram in mostNunigrams:
87             rowEntry.append(NYTjson[story]['body']['cleanText'].count(unigram))
88         rowEntry.insert(0, NYTjson[story]['meta']['dsk'])
89         # write row
90         w.writerow(rowEntry)

```

## Clustering Methods

- 1) Using the `kmeans` function, create a plot of the `kmeans` objective function as the number of clusters varies from 2 to  $N - 1$ .
- 2) Apply K-Means with 6 clusters, being sure to use `set.seed` to ensure you can replicate your analysis
- 3) Label each cluster using computer and hand methods:

- i) Suppose  $\theta_k$  is the cluster center for cluster  $k$  and define  $\bar{\theta}_{-k} = \frac{\sum_{j \neq k} \theta_j}{K-1}$  or the average of the centers not  $k$ . Define

$$\text{Diff}_k = \theta_k - \bar{\theta}_{-k}$$

Use the top ten words from  $\text{Diff}_k$  to label the clusters

- ii) Sample and read texts assigned to each cluster and produce a hand label

## 2 Dictionary Classification Methods

- a) Download the list of positive (<http://www.unc.edu/~ncaren/haphazard/positive.txt>) and negative (<http://www.unc.edu/~ncaren/haphazard/negative.txt>) stop words from Neil Caren's website.

- b) Calculate a positive score and a negative score for each document and the difference between each score using the dictionaries

```
1 # Problem 2
2
3 # download list of positive and negative stop words
4 # create function to load sentimental dictionaries
5 def loadWords(type, stemmer):
6     # open url specifying positive or negative dictionary
7     url = urlopen('http://www.unc.edu/~ncaren/haphazard/' + type + '.txt').read
8     ()
9     # since they are in .txt files, we need to split each word
10    # create the unstemmed dictionary
11    unstemmedDict = url.split('\n')
12    # determine which stemmer should be used
13    # (1) Porter
14    if stemmer=='Porter':
15        # for each word in dictionary, stem
16        stemmedDict = [nltk.stem.PorterStemmer().stem(word) for word in
17            unstemmedDict]
18    # (2) Snowball
19    elif stemmer=='Snowball':
20        # for each word in dictionary, stem
21        stemmedDict = [nltk.stem.SnowballStemmer('english').stem(word) for word in
22            unstemmedDict]
23    # (3) Lancaster
24    elif stemmer=='Lancaster':
25        stemmedDict = [nltk.stem.LancasterStemmer().stem(word) for word in
26            unstemmedDict]
27    else:
28        stemmedDict = unstemmedDict
29    # return both stemmed and unstemmed dictionaries
30    return [unstemmedDict, stemmedDict]
31
32 # get basic positive and negative, porter stemmed dictionaries
33 positiveWords = loadWords('positive', stemmer='Porter').pop(1)
34 negativeWords = loadWords('negative', stemmer='Porter').pop(1)
35
36 # create function that will easily check how many words are in
37 # corresponding dictionary list
38 def wordCount(inputStatement, dictionaries):
39     return len([x for x in inputStatement if x in dictionaries])
40
41 # create function to pull necessary info from each statement
42 def storyInfo(story, documentContent):
43     # first, need to discard punctuation
44     removedPunctuation = re.sub('\W', ' ', str(NYTjson[story]['body']['cleanText
45         ']))
46     # capitalization
47     removedCaps = removedPunctuation.lower()
48     # and tokenization
49     reducedStatements = nltk.word_tokenize(removedCaps)
```

```

44
45     # append documentContent with relevant info
46     documentContent.append({
47     # add to statementIter
48     'electionTiming': NYTjson[story]['meta']['publication_day_of_month'],
49     'desk': NYTjson[story]['meta']['dsk'],
50     # record the number of --- in statements w/ no punctuation, caps,
51     # and reduced tokens:
52     # non-stop words so that we can look at rates in analysis
53     'NonstopWords': len([x for x in reducedStatements if x not in stopWords]),
54     # number of words in each positive and negative using:
55     # (1) Porter stem
56     'NposPorter': wordCount([nltk.stem.PorterStemmer().stem(y) for y in
57     reducedStatements], positiveWords),
58     'NnegPorter': wordCount([nltk.stem.PorterStemmer().stem(y) for y in
59     reducedStatements], negativeWords),})
60
61 # create empty list to fill with statement info
62 storyCharacteristics = []
63 # execute story info extraction on each story
64 for i in range(0, len(NYTjson)):
65     # execute statementInfo function for each statement
66     storyInfo(i, storyCharacteristics)
67
68 # with data now assigned to dictionary
69 # write content to .csv to analyze in R
70 with open('Documents/Git/WUSTL-textAnalysis/storySentimentInfo.csv', 'wb') as
71     f:
72     w = csv.DictWriter(f, fieldnames=('electionTiming', 'desk', 'NonstopWords',
73     'NposPorter', 'NnegPorter'))
74     w.writeheader()
75     for item in storyCharacteristics:
76         w.writerow(item)

```

- c) How does the score change before and after the election? How does the score vary across desks?

### 3 Supervised Learning with Naive Bayes

- a) Using the version of Naive Bayes outlined on slide 24 of lecture 14, write a function to estimate  $p(C_k)$  and  $\theta_k$  for an arbitrary collection of categories. Hint: to compute the probability of a document from a category, note you can work with the log of the probability equivalently.
- b) Let's focus on documents that came from Business/Financial desk and National Desk. Using leave-one out cross validation, calculate the accuracy of Naive Bayes to calculate the label.

c) Compare the performance of Naive Bayes to the performance of 2 of the following 3 algorithms using 10-fold cross validation:

- LASSO
- Ridge
- KRLS

How does Naive Bayes compare?