Text as Data: Homework 3

Jeff Ziegler

August 17, 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

```
# import libraries
import json
import nltk
from nltk.corpus import stopwords
import re
import os
from urllib import urlopen

```

```python
# a) download {nyt_ac.json} from gitHub
with open('Documents/Git/WUSTL_textAnalysis/nyt_ac.json') as data_file:
    NYTjson = json.load(data_file)

# b) inspect the structure of the data
type(NYTjson)
# shows that json object is a list
# check the first elements as example
NYTjson[0].keys()
# two initial dictionaries (body, meta)
NYTjson[0]['body'].keys()
# within body, info like title and text

# c) write each story into new .txt file
for story in range(0,len(NYTjson)):
  # give title based on order
  with open('Documents/Git/WUSTL_textAnalysis/NYTstories/' +
  str(story) + '.txt', 'w') as file:
    # since we know where the relevant info is stored
    # use dump to write json as string by key
    file.write(json.dumps(NYTjson[story]['body']))

# check that files wrote properly
os.listdir('Documents/Git/WUSTL_textAnalysis/NYTstories/')
```

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

```python

# d) task: Using NYTjson file, and create a DTM
# of the 1000 most used terms
# be sure to:
# discard word order
# remove stop words
# apply porter stemmer

# create function to use Porter stemmer
def porterStem(unstemmedList):
  return [nltk.stem.PorterStemmer().stem(words) for words in unstemmedList]

# load a set of stop words from nlkt
# with the other stop word additions
stopWords = stopwords.words('english')
```

```
# apply Porter stemmer to stop words
stopWords = porterStem(stopWords)

# edit the text of pressReleases
for story in range(0,len(NYTjson)):
  # create key in each NYT story with cleaned text
  NYTjson[story]['body']['cleanText'] = []
  # remove capitalization
  textTokens = NYTjson[story]['body']['body_text'].lower()
  # discard punctuation by removing non-word characters
  textTokens = re.sub('\W', ' ', textTokens)
  # and apply Porter stem to tokenized story
  textTokens = porterStem(nltk.word_tokenize(textTokens))
  # remove stop words
  textTokens = [x for x in textTokens if x not in stopWords]
  # then append unigram tokens
  NYTjson[story]['body']['cleanText'] = textTokens

# now that we have cleaned text, need to create DTM
# create empty dictionaries to be filled with counts of unigrams
unigramDict = {}

# for each story
for story in range(0,len(NYTjson)):
  # add counts to totals
    for word in NYTjson[story]['body']['cleanText']:
        if word not in unigramDict:
            unigramDict[word] = 1
        else:
            unigramDict[word] += 1


# sort unigrams and trigrams into new lists
mostNunigrams = []

# create function to take the most used words
def extractTopN(topsList, mostNgrams):
  # loop over dictionary and append new list
  # by value, rather than key
  # then sort list
  return sorted(topsList, key=topsList.get, reverse=True)

# extract the most used unigrams
mostNunigrams = extractTopN(unigramDict, mostNunigrams)
# take only the top 1000
mostNunigrams = mostNunigrams[:1000]

# now that we have the most 1000 used unigrams
# create DTM matrix and write it to .csv
# task: we need to check whether each of the top 1000 words
# is in each NYT story, and count their frequency
# will iterate over each NYT story
# first , open up .csv writer
with open('Documents/Git/WUSTL_textAnalysis/NYTstoriesDTM.csv', 'wb') as f:
    w = csv.writer(f)
    # create header to be written to .csv as variable names
    # 1st column is desk name, preceding columns
    # represent top 1000 unigrams
    csvHeader = mostNunigrams
    csvHeader.insert(0,'desk')
    # write header first
    w.writerow(csvHeader)
    # then, we need to create counts of all the words
  # in each document (NOT across authors or
  # documents like the previous problem)
  # so, for each story
    for story in range(0,len(NYTjson)):
    # create a clear row
  rowEntry = []
  # and give each unigram count in that story
  for unigram in mostNunigrams:
    rowEntry.append(NYTjson[story]['body']['cleanText'].count(unigram))
  rowEntry.insert(0, NYTjson[story]['meta']['dsk'])
  # write row
  w.writerow(rowEntry)
```

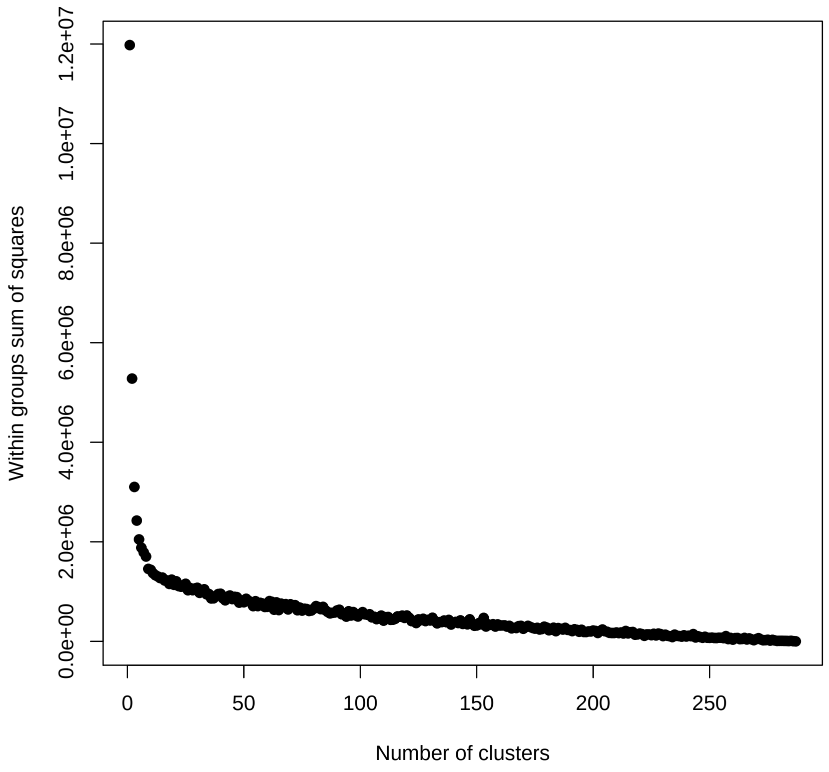
Clustering Methods

1) Using the `kmeans` function, create a plot of the `kmeans` objective function as the number of clusters varies from 2 to $N-1$.

```
# Problem 1

# load libraries and .csv files
library(foreign); library(fpc); library(data.table); library(glmnet); library(
    e1071)
NYTstoryDTM <- read.csv("~/Documents/Git/WUSTL_textAnalysis/NYTstoriesDTM.csv"
    , row.names=NULL)[,-2]

# a) task: use kmeans function to creat
# plot of the kmeans objective function as the
# number of clusters varies from 2 to N-1

# the data must be scaled for comparison
# we'll just use euclidean distance
euclideanNYTdtm <- as.matrix(dist(NYTstoryDTM[,-1], method = "euclidean"))

# to see which number of clusters is appropriate
# we want to vary the objective function of a range of K
# first , create k cluster points for 2 to N-1
# cluster points represent centroids for cluster j (c_j)
```

```
objectiveFunc <- (nrow(euclideanNYTdtm) - 1 * sum(apply(euclideanNYTdtm, 2,
    var)))
# then assign each object to the group w/ closest centroid
# after all objects have been assigned
# recalculate positions of the cluster centroids.
# repeat until centroids no longer move
for(i in 2:nrow(euclideanNYTdtm) - 1){
  objectiveFunc[i] <- sum(kmeans(euclideanNYTdtm, centers = i)$withinss)
}
# last, plot objectiveFunc
pdf("~/Documents/Git/WUSTL_textAnalysis/HW3kmeansClusters.pdf")
plot(2:nrow(euclideanNYTdtm) - 1, objectiveFunc, pch=19, xlab="Number of
    clusters", ylab="Within groups sum of squares")
dev.off()
```

Figure 1: `kmeans` objective function as number of clusters varies.



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 $\boldsymbol{\theta}_k$ is the cluster center for cluster k and define $\bar{\boldsymbol{\theta}}_{-k} = \frac{\sum_{j \neq k} \boldsymbol{\theta}_j}{K-1}$ or the average of the centers not k. Define

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

```r

# b) apply K-Means with 6 clusters
# set seed
set.seed(4); NYTclusters6 <- kmeans(euclideanNYTdtm, 6)

# c) task: label each cluster using computer and hand methods
# i) the final cluster centers are computed as the mean for each
    feature
# within each final cluster
# in other words, they reflect the characteristics of the "exemplar"
    document for each cluster

# so, if theta_k is the cluster center for cluster k
# mean(theta_{-k}) = (sum_(theta_j)/(K-1))
# or the average of the centers not k
# then Diff_k = theta_k - mean(theta_{-k})
# use the top ten words from Diff_k to label the clusters

# so, first make data frame of cluster centers
clusterCenters <- as.data.frame(NYTclusters6$centers)
# write a function that:
# finds theta_k, which is row k of clusterCenters
# and theta_{-k}, which is rows not-k of cluster centers divided by
    number of clusters - 1
# to create Diff_k
top10words <- function(k, clusters){
  internalFunc <- function(k, clusters){
    Diff_k <- as.data.frame(centers[k,] - colSums(centers[-(k),])/(
    max(clusters)-1))
    # finally, take top 10 words
    return(colnames(Diff_k[,order(Diff_k,decreasing=T)][1:10]))
  }
  # set up matrix for top 10 words
  keywords <- matrix(NA, nrow=10, ncol=k)
  # take top 10 words for each cluster
  for (i in 1:k){
    keywords[,i] <- internalFunc(i, clusterCenters)
  }
  return (keywords)
}
# execute top10words function
top10words(6, clusterCenters)
```

Use the top ten words from Diff_k to label the clusters

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

```r
# ii) sample and read texts assigned to each cluster and produce a hand label
# sample 2 stories from cluster 1
exploreClusters <- function(seed, k, nDocs){
  set.seed(seed)
  for(i in sample(which(NYTclusters6$cluster==k),2)){
    suppressWarnings(print(readLines(paste("~/Documents/Git/WUSTL_textAnalysis
    /NYTstories/",i,".txt", sep=''))))
  }
}

# these articles don't seem to match well, maybe
# they are grouped together cause they are talking
# about "battles" and competing (on the field and in the courtroom)
exploreClusters(5, 5, 2)
```

```
[1] ""title": "Red-Zone Forays Put Giants in the Black",
"body_text": "The Giants did not exactly become an offensive
juggernaut in Sunday's victory over Minnesota. They gained
283 yards, collected 83 yards in penalties from the Vikings and
absorbed five sacks, and quarterback Kurt Warner fumbled
three snaps... <truncated>
[2]""title": "G.O.P. in Ohio Can Challenge Voters at Polls",
"body_text": "In a day of see-sawing court rulings, a Federal appeals
court ruled early Tuesday morning that the Republican Party could
place thousands of people inside polling places to challenge the
eligibility of voters, a blow to Democrats who argued those challengers
 will intimidate minority voters... <truncated>
```

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

```python
# Problem 2

# download list of positive and negative stop words
# 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, porter stemmed dictionaries
positiveWords = loadWords('positive', stemmer='Porter').pop(1)
negativeWords = loadWords('negative', stemmer='Porter').pop(1)

# 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
def storyInfo(story, documentContent):
  # first, need to discard punctuation
  removedPunctuation = re.sub('\W', ' ', str(NYTjson[story]['body']['cleanText']))
  # capitalization
  removedCaps = removedPunctuation.lower()
  # and tokenization
  reducedStatements = nltk.word_tokenize(removedCaps)

    # append documentContent with relevant info
  documentContent.append({
  # add to statementIter
  'electionTiming': NYTjson[story]['meta']['publication_day_of_month'],
  'desk': NYTjson[story]['meta']['dsk'],
  # record the number of ___ in statements w/ no punctuation, caps,
  # and reduced tokens:
  # non-stop words so that we can look at rates in analysis
  'NonstopWords': len([x for x in reducedStatements if x not in stopWords]),
    # number of words in each positive and negative using:
    # (1) Porter stem
    'NposPorter': wordCount([nltk.stem.PorterStemmer().stem(y) for y in reducedStatements], positiveWords),
    'NnegPorter': wordCount([nltk.stem.PorterStemmer().stem(y) for y in
    reducedStatements], negativeWords),})

# create empty list to fill with statement info
storyCharacteristics = []
# execute story info extraction on each story
for i in range(0,len(NYTjson)):
  # execute statementInfo function for each statement
  storyInfo(i, storyCharacteristics)

# with data now assigned to dictionary
# write content to .csv to analyze in R
with open('Documents/Git/WUSTL_textAnalysis/storySentimentInfo.csv', 'wb') as
    f:
    w = csv.DictWriter(f, fieldnames=('electionTiming', 'desk', 'NonstopWords'
    ,
     'NposPorter', 'NnegPorter'))
    w.writeheader()
    for item in storyCharacteristics:
      w.writerow(item)
```

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

```
# Problem 2

# c) How does the score change before and after the election?
# How does the score vary across desks?
# first, load data
storyInfo <- read.csv("~/Documents/Git/WUSTL_textAnalysis/storySentimentInfo.
    csv")
# then create new dataframe w/ word rate for each column
rateDF <- cbind(storyInfo[,c(1:2)], storyInfo[,c(4:5)]/storyInfo[,3][row(
    storyInfo[,c(4:5)])])
# see if positive word rate
posToneLM <- lm(NposPorter ~ as.factor(electionTiming) + desk,rateDF)
negToneLM <- lm(NnegPorter ~ as.factor(electionTiming) + desk,rateDF)
summary(posToneLM);summary(negToneLM)
# generally, it doesn't seem like there are more positive
# words over the course of the election (day before the election is reference)
# some desks like health and culture are more "upbeat"
# and then the editorial, national, and foreign ("hard news") is slightly
# more negative
```

Table 1: Regression results of sentimental change.

	Dependent variable:	
	NposPorter	NnegPorter
	(1)	(2)
Election day	−0.003	−0.001
	(0.002)	(0.002)
Day after election	0.001	−0.002
	(0.002)	(0.002)
Editorial Desk	0.013***	0.015***
	(0.004)	(0.003)
Foreign Desk	−0.005	0.013***
	(0.003)	(0.003)
Health and Fitness	0.016***	0.011**
	(0.006)	(0.005)
Metropolitan Desk	−0.001	0.010*
	(0.006)	(0.006)
National Desk	0.003	0.007***
	(0.002)	(0.002)
Science Desk	0.002	0.001
	(0.006)	(0.006)
Sports Desk	0.002	−0.0002
	(0.003)	(0.003)
The Arts/Cultural Desk	0.011***	0.005**
	(0.003)	(0.003)
Constant	0.054***	0.033***
	(0.002)	(0.002)
Observations	288	288
R^2	0.144	0.144
Note:	*p<0.1; **p<0.05; ***p<0.01	

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 $\boldsymbol{\theta}_k$ 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?

```r
# Problem 3
# run naive bayes w/ leave one out validation
# find random validation observation
# and create training and test set
# first , create dichotomy noting the specified categories
# Business/Financial desk and National Desk
NYTstoryDTM$row.names <- ifelse(as.character(NYTstoryDTM$row.names)=='Business
    /Financial Desk' |
                                        as.character(NYTstoryDTM$row.names)=='
    National Desk', 1, 0)
set.seed(1)
randomObservation <- sample(which(NYTstoryDTM$row.names==1), 1)
trainingSet <- NYTstoryDTM[-randomObservation,]
validation <- NYTstoryDTM[randomObservation,]

# run naive bayes classifier
nbResults <- naiveBayes(trainingSet[,1]~., data=trainingSet)
# predict validation observation
default_pred <- predict(nbResults, as.matrix(validation[,-1]))
# predicts 0, which is incorrect

# c) use lasso and ridge
# run cv.glmnet w/ 10 folds and alpha = 1 for lasso, alpha=0 for ridge
lassoModel <- cv.glmnet(x = as.matrix(trainingSet[,-1]), y = trainingSet[,1],
    alpha = 1,
                          nfolds=10, family="binomial", type.measure="mse")
ridgeModel <- cv.glmnet(x = as.matrix(trainingSet[,-1]), y = trainingSet[,1],
    alpha = 0,
                          nfolds=10, family="binomial", type.measure="mse")
# predict category for both models
# by seeing if predicted probabilites > 0.5
predictedProbs <- c(1/(1 + exp(-predict(lassoModel, newx = as.matrix(
    validation[,-1]), s = lassoModel$lambda.min))),
                     1/(1 + exp(-predict(ridgeModel, newx = as.matrix(
    validation[,-1]), s = ridgeModel$lambda.min))))
# [1] 0.6638116 0.6748607
# both seem to do better than the naive bayes
```