Contents

Week 6

Text mining

- **Read:**
  - Silge and Robinson (2017), Text Mining with R, O'Reilly Media Inc.
  - Kwartler (2017), Text Mining in Practice with R, Wiley
- **Learning objectives:**
  - Define key terms: string, term, token/tokenization, document, corpus, regular expression, bag of words (BOW), bigram, n-gram, stop word, stemming, lemmatization, document-term matrix (DTM)
  - Identify the steps in the text mining workflow
  - Read a corpus into software (e.g., R or Python) and prepare it for analysis following the workflow
  - Know how to apply basic string-processing functions, e.g., convert to upper/lower case, find length, concatinate two strings, extract a substring, split a string into tokens based on a delimiter, find and replace based on a regular expression, and locate a pattern.
  - Use software to create descriptive statistics and visualizations
  - Compute tf-idf scores, explain the rationale for transformation, and use it to select terms
  - Incorporate text features in a subsequent analysis, e.g., clustering, prediction or dimensionality reduction, to solve some domain-specific problem

Week 7

Text mining

- **Read:**
- **Watch:** (?)
  - ???
- **Do:** Homework ?, page **??**
- **Learning objectives:**
  - asdf

References

- Books
  - Silge and Robinson (2017), Text Mining with R, O'Reilly Media Inc.
  - Kwartler (2017), Text Mining in Practice with R, Wiley
  - Bird, Loper and Klein (2009), Natural Language Processing with Python. O'Reilly Media Inc.
  - Text Analysis Fundamentals
  - Text mining workflow
- Software
  - R `tm` package
  - R Tidy Text package
  - Python natural language toolkit (NLTK)

Introduction to text mining

- **Structured data**: highly organized and easily decipherable. Think of a table with rows (for each "unit" or "entity") and fields (attributes, variables, etc.), e.g., a customer table might have fields customer id, add date, name, address, balance, etc.
- **Unstructured data**: does not have a pre-defined and consistent data model, e.g., text, images, video.
- Predictive analytics and our coverage of unsupervised learning have (mostly) dealt with structured data
- **Text mining** is the process of distilling actionable insights from text
- This module on text mining gives a taste of unstructured data:
  - Organize unstructured data
  - Create "feature variables"
  - Analyze features (e.g., with descriptive statistics, supervised or unsupervised methods), e.g.,
    * **Word cloud** visualize most common words
    * **Sentiment analysis** predict valence of a "document" from text
    * **Topic models** cluster documents
  - Identify and summarize insights

Units of TM analysis

- Our primary unit of analysis is a **document**, e.g., text of webpage, text of article/headline, product description, customer review, court record from a trial, new product idea.
- A collection of documents is a **corpus**
- A **word** is the unit in a **dictionary**
- An n**-gram** is a sequence of words, e.g., “New York” is a **bigram** and “The White House” is a **trigram**.
- A **token** (or `term`) is a meaningful unit of text (e.g., word or bigram) *that we are interested in using for further analysis.*
- **Tokenization** is the process of splitting text into tokens.
  - In English we often remove certain punctuation and look for a sequence of characters separated by white space (space, tabs, returns, etc.), but this does not work in other languages, e.g., Chinese
  - Sometimes punctuation important, e.g., ! for spam filters
  - Names like “O’Reilly” and contractions (e.g., don’t) should be one word.
  - Beware of “words” like “C++,” “B-52,” IP addresses (e.g., 129.105.48.123), and email addresses

Two TM Approaches

- Bag of words (BOW) model: consider a document as a set of words, ignoring order, grammar, etc. Limitations:
  - Words in negative context: e.g., steakhouse description: “there is nothing on the menu that a vegetarian would like” The keyword “vegetarian” caused the restaurant to be recommended to vegetarians.
  - Identical bags of words but opposite meanings:
    - ‘not an issue, phone is working’
    - ‘an issue, phone is not working’
  - n-gram model: considers n words at a time, e.g., bigrams ‘an issue,’ ‘issue phone,’ ‘phone is,’ ‘is not,’ and ‘not working’ (‘not working’ bigram probably important!)
  - n-grams tend to create many terms
- **Syntactic parsing** identifies the roles of words in a sentence with **part of speech (POS) tagging**, e.g., subject, object, adjectives, verb phrases, noun blocks, e.g.,

$$\underbrace{\text{Lebron James}}_{\text{named entity}}\ \underbrace{\underbrace{\text{hit}}_{\text{verb}}\ \underbrace{\text{a}}_{\text{article}}\ \underbrace{\text{tough}}_{\text{adjective}}\ \underbrace{\text{shot}}_{\text{noun}}}_{\text{verb phrase}}$$

- We focus on BOW

Text data matrices

- We track certain **terms** (words, bigrams, etc.) in documents
- The **document-term matrix** (DTM) gives the number of times (or 0-1 indicators) a term (column) occurs in each document (row)
- A **term-document matrix** (TDM) is the transpose
- We want to do useful things with the DTM:
  - Identify defining terms of a document with **tf–idf**, e.g., identify key terms in product descriptions
  - **Sentiment analysis**: estimate whether the documents are positive, negative or neutral
  - Identify "types" of documents with clustering or **topic models**, e.g., **latent Dirichlet allocation** (**LDA**)
  - Reduce the dimension (like PCA/factor analysis) with **latent semantic analysis** (**LSA**)
  - Predict something (regression or classification), e.g.,
    * customer review → purchase or total revenue
    * email text → spam,
    * news article → labeled types, e.g., sports, business, events, health, government
    * Who wrote each Federalist paper?
    * Classify good/bad ideas on crowd-sourcing website
    * Classify airline tweets by severity

Text mining basic workflow

---

1. Preparation
   - Maybe run spell checker or dictionary of synonyms
   - **Phrases**, e.g., "United Nations." I sometimes create a dictionary of phrases and do a search and replace, e.g., "New York" to "NewYork" (or use bigrams)
   - Maybe replace phone numbers with "PhoneNumber" (see regular expressions) NFL team with "NFLteam," etc.
   - Convert to lower case (beware: White House is not the same as white house)
   - Remove **stop words**, e.g., a, the, on, as. (e.g., RanksNL)
   - **Lemmatization** remove inflectional endings and return to base dictionary form of a word
   - **Stem** words, e.g., running $\rightarrow$ run. Lemmas are actual words while stems may not be
2. Select terms
3. Maybe transform data (e.g., tf-idf)
4. Fit model

Tiny Example

```
> library(tidyverse); library(tm)
> library(wordcloud)

> setwd("/Users/ecm/teach/textmine")

# read text data
> corpus = read.csv("tiny.txt", sep="|")
> corpus
  id                              str
1  1 the cat the dog the mouse the cat
2  2         the cat the milk the apple
3  3     the mouse the cheese the beer
4  4                 the mouse the dog
> dim(corpus)
[1] 4 2

# convert to tidy data structure
> text_df = data_frame(
+   id=as.character(corpus$id),
+   text = as.character(corpus$str))
> text_df
# A tibble: 4 × 2
     id text
  <chr>   <chr>
1    1   "the cat the dog the mouse the cat"
2    2   "the cat the milk the apple"
3    3   "the mouse the cheese the beer"
4    4   "the mouse the dog "

# tokenize
> text_df %>% unnest_tokens(word, text)
# A tibble: 24 × 2
   id    word
   <chr> <chr>
 1 1     the
 2 1     cat
 3 1     the
 4 1     dog
 5 1     the
 6 1     mouse
 7 1     the
 8 1     cat
 9 2     the
10 2     cat
# ... with 14 more rows
```

```
# Let's make bigrams instead
>  text_df %>% unnest_tokens(word, text,
      token="ngrams", n=2)

A tibble: 20 × 2
   id word
   <chr>   <chr>
 1 1       the cat
 2 1       cat the
 3 1       the dog
 4 1       dog the
 5 1       the mouse
 6 1       mouse the
 7 1       the cat
 8 2       the cat
 9 2       cat the
10 2       the milk
11 2       milk the
12 2       the apple
13 3       the mouse
14 3       mouse the
15 3       the cheese
16 3       cheese the
17 3       the beer
18 4       the mouse
19 4       mouse the
20 4       the dog
```

```
# let's remove stop words
> text_df2 = text_df %>%
+   unnest_tokens(word, text) %>%
+   anti_join(stop_words)
Joining, by = "word"
> text_df2
# A tibble: 12 × 2
   id    word
   <chr> <chr>
 1 1     cat
 2 1     dog
 3 1     mouse
 4 1     cat
 5 2     cat
 6 2     milk
 7 2     apple
 8 3     mouse
 9 3     cheese
10 3     beer
11 4     mouse
12 4     dog

# compute frequency distribution
> text_df2 %>% count(word, sort=T)
# A tibble: 7 × 2
  word       n
  <chr>  <int>
1 cat        3
2 mouse      3
3 dog        2
4 apple      1
5 beer       1
6 cheese     1
7 milk       1

# make word cloud
> text_df2 %>%
+   count(word) %>%
+   with(wordcloud(word, n))
```

```
# find counts by document id
> text_df2 %>%
+   group_by(id) %>%
+   count(word, sort=T) %>%
+   ungroup()
# A tibble: 11 × 3
   id    word       n
   <chr> <chr>  <int>
 1 1     cat        2
 2 1     dog        1
 3 1     mouse      1
 4 2     apple      1
 5 2     cat        1
 6 2     milk       1
 7 3     beer       1
 8 3     cheese     1
 9 3     mouse      1
10 4     dog        1
11 4     mouse      1

> as.matrix(cast_tdm(df3, term=word,
    document=id, value=n))
        Docs
Terms    1 2 3 4
  cat    2 1 0 0
  dog    1 0 0 1
  mouse  1 0 1 1
  apple  0 1 0 0
  milk   0 1 0 0
  beer   0 0 1 0
  cheese 0 0 1 0

> as.matrix(cast_dtm(df3, document=id,
     term=word, value=n))
    Terms
Docs cat dog mouse apple milk beer cheese
   1   2   1     1     0    0    0      0
   2   1   0     0     1    1    0      0
   3   0   0     1     0    0    1      1
   4   0   1     1     0    0    0      0
```

Wine Example

```
##################
# wine example
##################
wine = read.csv("wine.txt", sep="|")
wine
text_df = data_frame(
  id=as.character(wine$id),
  text = as.character(wine$desc))
text_df

(text_df2 = text_df %>%
  unnest_tokens(word, text))

text_df2 %>%
  count(word, sort=T)
text_df2 %>%
  count(word) %>%
  with(wordcloud(word, n))

# remove stop words first
text_df2 %>%
  count(word, sort=T) %>%
  anti_join(stop_words)
text_df2 %>%
  count(word) %>%
  anti_join(stop_words) %>%
  with(wordcloud(word, n))

# make bar plot
text_df2 %>%
  count(word, sort=T) %>%
  anti_join(stop_words) %>%
  filter(n > 2) %>%
  mutate(word = reorder(word, n)) %>%
  ggplot(aes(word, n)) +
  geom_col() +
  xlab(NULL) +
  coord_flip()
```

What is tf-idf?

- tf-idf is *term frequency* $\times$ *inverse document frequency*
- Let $\mathcal{D}$ be the corpus of documents
- $\text{tf}(d, t) =$ **term frequency**: for some document $d \in \mathcal{D}$, either
  - count: number of times that term t appears
  - fraction: divide count by total number of terms in document
  - indicator: 1 if term t appears, 0 otherwise
- $\text{idf}(t, \mathcal{D}) =$ **inverse document frequency**: how much information the word provides, i.e., if it's common or rare across documents. It's usually the log-scaled inverse fraction of the documents containing the term

$$\text{idf}(t, \mathcal{D}) = \log \frac{N}{|\{d \in \mathcal{D} : t \in d\}|}$$

  - N: total number of documents in corpus $N = |\mathcal{D}|$
  - $|\{d \in \mathcal{D} : t \in d\}|$: number of documents where term t appears, i.e., $\text{tf}(t, d) > 0$
- Automatically demotes stop words and common terms
- Promotes core terms over incidental ones

tf-idf applied to tiny example

```
> corpus = read.csv("tiny.txt", sep="|")
> (text_df = data_frame(
+   id=as.character(corpus$id),
+   text = as.character(corpus$str)))
  id    text
1 1     "the cat the dog the mouse the cat"
2 2     "the cat the milk the apple"
3 3     "the mouse the cheese the beer"
4 4     "the mouse the dog "

> # same as before, plus bind_tf_idf
> (df2=text_df %>%
+   unnest_tokens(word, text) %>%
+   group_by(id) %>%
+   count(word, sort=T) %>%
+   ungroup() %>%
+   bind_tf_idf(word, id, n))
   id    word       n    tf    idf tf_idf
 1 1     the        4 0.5   0     0
 2 2     the        3 0.5   0     0
 3 3     the        3 0.5   0     0
 4 1     cat        2 0.25  0.693 0.173
 5 4     the        2 0.5   0     0
 6 1     dog        1 0.125 0.693 0.0866
 7 1     mouse      1 0.125 0.288 0.0360
 8 2     apple      1 0.167 1.39  0.231
 9 2     cat        1 0.167 0.693 0.116
10 2     milk       1 0.167 1.39  0.231
11 3     beer       1 0.167 1.39  0.231
12 3     cheese     1 0.167 1.39  0.231
13 3     mouse      1 0.167 0.288 0.0479
14 4     dog        1 0.25  0.693 0.173
15 4     mouse      1 0.25  0.288 0.0719
```

```
> df2 %>% arrange(desc(tf_idf))
   id    word       n     tf   idf tf_idf
   <chr> <chr>  <int> <dbl> <dbl>  <dbl>
 1 2     apple      1 0.167 1.39  0.231
 2 2     milk       1 0.167 1.39  0.231
 3 3     beer       1 0.167 1.39  0.231
 4 3     cheese     1 0.167 1.39  0.231
 5 1     cat        2 0.25  0.693 0.173
 6 4     dog        1 0.25  0.693 0.173
 7 2     cat        1 0.167 0.693 0.116
 8 1     dog        1 0.125 0.693 0.0866
 9 4     mouse      1 0.25  0.288 0.0719
10 3     mouse      1 0.167 0.288 0.0479
11 1     mouse      1 0.125 0.288 0.0360
12 1     the        4 0.5   0     0
13 2     the        3 0.5   0     0
14 3     the        3 0.5   0     0
15 4     the        2 0.5   0     0

# Or we can cast it as a DTM or TDM
> df2 %>%
  cast_tdm(df2, document=id, term=word,
      value=tf_idf) %>%
+   as.matrix() %>%
+   round(digits=3)
        Docs
Terms        1     2     3     4
  the    0.000 0.000 0.000 0.000
  cat    0.173 0.116 0.000 0.000
  dog    0.087 0.000 0.000 0.173
  mouse  0.036 0.000 0.048 0.072
  apple  0.000 0.231 0.000 0.000
  milk   0.000 0.231 0.000 0.000
  beer   0.000 0.000 0.231 0.000
  cheese 0.000 0.000 0.231 0.000
```

- cat appears in 2/4=.5 documents, and 2/8=.25 terms in document 1, so tf-idf for cat in document1: .25*log(1/.5) = 0.1732868
- mouse appears in 3/4=.75 documents and 1/8 terms in document 1, so tf-idf for mouse in document1: log(1/.75)/8 = 0.03596026

Regular Expressions

Regular expressions match a pattern of characters.

- Meta characters

```
.       wildcard, matches a single char
^       start of a string
$       end of a string
[]      match one of the set of chars within []
[a-z]   matches a...z
[^abc]  matches a char that is not a, b, or c
a|b     matches either a or b, where a and b are strings
\       escape char (\t, \n, \b)
\b      match word boundary
\d      any digit, same as [0-9]
\D      any non-digit, same as [^0-9]
\s      any whitespace, same as [ \t\n\r\f\v]
\S      any non-whitespace char, same as [^ \t\n\r\f\v]
\w      any alphanum char, same as [a-zA-Z0-9_]
\W      any non-alphanum char,  same as [^a-zA-Z0-9_]
```

- Repetitions

```
*       matches 0 or more occurrences
+       matches one or more occurrences
?       matches 0 or 1 occurrences
{n}     exactly n repetitions, n>=0
{n,}    at least n reps
{,n}    at mos n reps
{m,n}   at least m and at most n reps
```

Additional references analyticsvidhya or ieva.rocks

Regular expression examples

```
> dat=read.csv("ex1.txt", sep="|")
> dat
  id                                str
1  1                today is 2/7/2022
2  2                  I like ice cream
3  3   a week from today is 2/14/2022
4  4                  my dog has fleas
5  5 the phone number is 312-789-1234
6  6     call 800-234-5678 to buy one

# find rows with "today"
```

```
> dat %>% filter(str_detect(str, "today"))
  id                              str
1  1              today is 2/7/2022
2  3 a week from today is 2/14/2022

# find rows beginning with "today"
> dat %>% filter(str_detect(str, "^today"))
  id               str
1  1 today is 2/7/2022
```

```
# find rows with a date
> dat %>% filter(str_detect(str, "[0-9]{1,2}[/-][0-9]{1,2}[/-][0-9]{4}"))
  id                              str
1  1              today is 2/7/2022
2  3 a week from today is 2/14/2022

# or we use \d instead of [0-9]
> dat %>% filter(str_detect(str, "\\d{1,2}[/-][0-9]{1,2}[/-][0-9]{4}"))
  id                              str
1  1              today is 2/7/2022
2  3 a week from today is 2/14/2022

# replace date with DATE
> dat %>%
  mutate(str=str_replace(str, "\\d{1,2}[/-][0-9]{1,2}[/-][0-9]{4}", "DATE"))
  id                                str
1  1                     today is DATE
2  2                  I like ice cream
3  3        a week from today is DATE
4  4                  my dog has fleas
5  5 the phone number is 312-789-1234
6  6     call 800-234-5678 to buy one
```

Headline example

```
library(tidyverse); library(tm)
library(SnowballC) # necessary for stemming
library(wordcloud); library(textstem)
setwd("/Users/ecm/teach/textmine")
corpus = read.csv("headline200b.txt", sep="|", quote="")
head(corpus,3)
dim(corpus)

(text_df = data_frame(   # text must be of type character
  article=as.character(corpus$article),
  text = as.character(corpus$headline))    )

# this is a test to see if str_replace works
# see https://www.rdocumentation.org/packages/stringr/versions/1.4.0
textdf2=text_df %>%
  mutate(text = str_replace_all(str_to_lower(text), "long island", "li")) %>%
  mutate(text = str_replace_all(text, "[’’]s", "")) # drop apostrophes
head(textdf2, 5)

textdf2 %>% unnest_tokens(word, text) # unigrams
textdf2 %>% unnest_tokens(bigram, text, token="ngrams", n=2) # bigrams
textdf2 %>% unnest_tokens(trigram, text, token="ngrams", n=3) # trigrams

# now remove stop words
head(stop_words)
table(stop_words$lexicon)
text_df %>% unnest_tokens(word, text) %>%
  anti_join(stop_words) %>%
  anti_join(data_frame(word=c("will", "heres"))) # additional stop words

# compare stemming with lemmatizing
wordStem(c("go", "went", "gone", "going"))
lemmatize_words(c("go", "went", "gone", "going"))
wordStem(c("good", "better", "best"))
lemmatize_words(c("good", "better", "best"))
wordStem(c("run", "ran", "running"))
lemmatize_words(c("run", "ran", "running"))
wordStem(c("university", "universities", "universal", "universe"))
lemmatize_words(c("university", "universities", "universal", "universe"))
getStemLanguages()
text_df %>% unnest_tokens(word, text) %>%
  anti_join(stop_words) %>%      # drop stop words
  mutate(word = wordStem(word))  # stem words

# in real life we would do it all at once
text_df2 = text_df %>%
```

```
  mutate(text = str_replace_all(text, "[Ll]ong [Ii]sland", "LI")) %>%  # replace phrases
  mutate(text = lemmatize_strings(str_to_lower(text))) %>%
  unnest_tokens(word, text) %>%
  anti_join(stop_words)   # remove stop words
text_df2


text_df2 %>%    # frequency distribution of corpus
  count(word, sort = TRUE) %>%
  print(n=50)

text_df2 %>%    # bar graph of most frequent words
  count(word, sort = TRUE) %>%
  filter(n > 5) %>%
  mutate(word = reorder(word, n)) %>%
  ggplot(aes(word, log(n, base=2))) +
  geom_col() +
  xlab(NULL) +
  coord_flip()

text_df2 %>%    # word cloud
  count(word) %>%
  with(wordcloud(word, n, max.words = 50))

df3 = text_df2 %>%        # tf-idf
  group_by(article) %>%
  count(word, sort = TRUE) %>%
  ungroup() %>%
  bind_tf_idf(word, article, n)
df3 %>% arrange(desc(tf_idf)) %>% print(n=50)
```

We're now ready to fit some models