Working with text features in prediction tasks

Eltecon Data Science Course by Emarsys

Tamás Koncz

November 3, 2021

About me

- Lead Data Scientist @ Emarsys AI Labs
- (Previous: employer: Morgan Stanley)
- Worked in data for ~10 years
- Educational background in Finance / Business IT
- CEU MSc in Business Analytics graduate
- Contact: *t.koncz@gmail.com*

NLP

- NLP: Natural Language Processing
- Industry hot topic: replacing computer vision
- Deep Learning dominates: word2vec, transformers, BERT, DALLE...
- Some partical applications:
  - Chatbots...
  - Search
  - Translations
  - Text Classification
- These methods still don't understand language!

About today

- Not going to be about those mentioned before :)
- (But all of them are built on what we'll cover)
- Introduce basic methods to work with text data:
  - Regular expressions
  - Tokenization
  - Sentiment analysis
- Using these for prediction: two weeks from now

Section 1

Regular expressions

Regular expressions

- Regular expressions (regex) are basically *smart keyword matching*
- Formally: "a sequence of characters that specifies a search pattern"
- Regex allows us to capture important patterns in our data

Basic regex example

```
bad_review <- "This movie was terrible."

good_review <- "I loved every minute of it!"
```

Basic regex example

```r
good_words <- c("good", "superb", "love", "like")

bad_words <- c("bad", "terrible", "hate")
```

```r
good_regex <- paste(good_words, collapse = "|")

bad_regex <- paste(bad_words, collapse = "|")

# note: "|" is "OR" in regex
```

Basic regex example

```
print(good_regex)
```

```
## [1] "good|superb|love|like"
```

```
print(bad_regex)
```

```
## [1] "bad|terrible|hate"
```

```
# note: "|" is "OR" in regex
```

Basic regex example

```
grepl(good_regex, good_review)
```

```
## [1] TRUE
```

```
grepl(bad_regex, good_review)
```

```
## [1] FALSE
```

```
grepl(bad_regex, bad_review)
```

```
## [1] TRUE
```

```
grepl(good_regex, bad_review)
```

```
## [1] FALSE
```

Literals, special characters and escaping

- Literals are what you write in normal language:
  - E.g. "." means "dot" in English
- In regex, there are certain special characters that make it powerful
  - E.g. in regex, "." means "match **ANY** character"
- So how do you match a literal "."? With escaping! `\.` = `"dot"`
  - (Due to string escaping, you'll have to use `\\.` in R... I'm sorry)

gsub() example I.

```
media_scores <- c("65%", "33%", "20%")
```

How to capture the numeric value of these scores?

```
gsub(pattern = "%", replacement = "", media_scores) %>%
    as.integer()
```

```
## [1] 65 33 20
```

gsub() example II.

```
year <- c("(2002)", "(1990)")
```

How to capture the year as a number?

```
gsub(pattern = "(\\(|\\))", replacement = "", year) %>%
    as.integer()
```

```
## [1] 2002 1990
```

Note: "(" and ")" are special characters, that's why we need escaping.

Exercise

Capture the movie id in the following string! Use the `text_features.R` file.

```
movie_url <- "https://www.rt.com/m/1003722-casino_royale"
```

(Expected answer: “1003722-casino_royale”)

Solution

```
gsub("https://www.rt.com/m/", "", movie_url)
```

```
## [1] "1003722-casino_royale"
```

More precisely:

```
gsub("https://www\\.rt\\.com/m/", "", movie_url)
```

```
## [1] "1003722-casino_royale"
```

Exercise II.

Let's say our task is a bit more complex:

```
movie_urls <- c(
    "https://www.rt.com/m/1003722-casino_royale",
    "https://www.rt.com/m/1003722-casino_royale/reviews",
    "https://www.rt.com/m/1003722-casino_royale/reviews?type=top_critics",
    "https://www.rt.com/m/1003722-casino_royale/pictures"
)
```

gsub() is not going to "scale"...

```
gsub("https://www\\.rt\\.com/m/", "", movie_urls)
```

```
## [1] "1003722-casino_royale"
## [2] "1003722-casino_royale/reviews"
## [3] "1003722-casino_royale/reviews?type=top_critics"
## [4] "1003722-casino_royale/pictures"
```

gsub() is not going to “scale”. . .

```
gsub(
    "/reviews", "",
    gsub("https://www\\.rt\\.com/m/", "", movie_urls)
)
```

```
## [1] "1003722-casino_royale"
## [2] "1003722-casino_royale"
## [3] "1003722-casino_royale?type=top_critics"
## [4] "1003722-casino_royale/pictures"
```

Capturing groups

```
stringr::str_match(
    string = movie_urls, pattern = ".*/m/(.+?)(/.*)?$"
)[, 2]
```

```
## [1] "1003722-casino_royale" "1003722-casino_royale" "1003722-casino_royale
## [4] "1003722-casino_royale"
```

Special characters:

- "*" - match any number of the preceding character
- "+" - match one or more of the preceding character
- "+?" - same as above, but non-greedy
- "?" - optional
- "()" - group of characters
- "$" - end of line

Exercise

```
cat(img_url)
```

```
## https://resizing.flixster.com/
## R1dBRE4KaDM5WfvLIS7-0aSZMIo=/206x305/v2/
## https://flxt.tmsimg.com/assets/p4248_p_v8_ad.jpg
```

Resized image

Solution

```
stringr::str_match(
    string = img_url,
    pattern = ".+/(https://.+\\.jpg)"
)[, 2]
```

```
## [1] "https://flxt.tmsimg.com/assets/p4248_p_v8_ad.jpg"
```

Original image

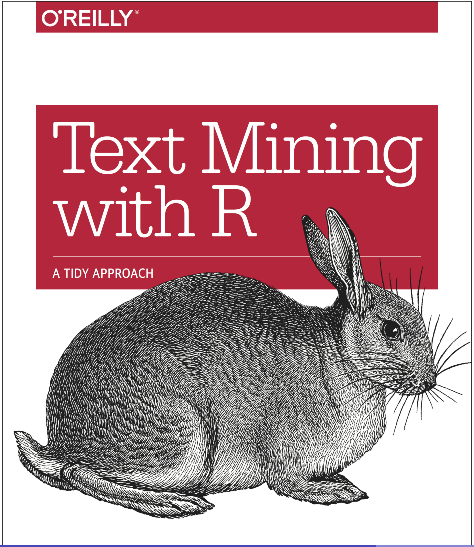
More on regex

- R4DS
- A Gentle Introduction to Regular Expressions with R
- A Complete Beginners Guide to Regular Expressions in R
- If you want to make a living out of regex: Mastering Regular Expressions

Section 2

Tidytext

Text Mining with R



Tokenization

A token is a meaningful unit of text, most often a word, that we are interested in using for further analysis, and tokenization is the process of splitting text into tokens.
- tidytextmining

Why we need tokenization?

```r
review <- data.table(review = "
    A few innovative sets,
    a wealth of eye-popping colors,
    and oodles of bared midriffs
    can't redeem this juvenile experiment
    in adolescent fantasy.
")
```

We understand this sentence, but computers can't.

Why we need tokenization?

```
review %>%
    unnest_tokens(output = "word", input = "review") %>%
    .[, .N, by = "word"] %>%
    .[order(-N)] %>%
    .[1:5]
```

```
##           word N
## 1:           a 2
## 2:          of 2
## 3:         few 1
## 4:  innovative 1
## 5:        sets 1
```

While words are still not meaningful for computers, by transforming text into a structured format (1 word / row), we can process it further for analysis.

Movie review data

```
reviews <- fread("data/james_bond_007_franchise_short_reviews.csv") %>%
    .[, .(movie_title, media_score, short_review)]
str(reviews, nchar.max = 55)
```

```
## Classes 'data.table' and 'data.frame':   540 obs. of  3 variables:
##  $ movie_title : chr  "Casino Royale" "Casino Royale" "Casino Royale" "Cas
##  $ media_score : int  25 25 25 25 25 25 25 25 25 25 ...
##  $ short_review: chr  "A tone-deaf spoof. Woody Allen steals the show." "A
##  - attr(*, ".internal.selfref")=<externalptr>
```

Exercise

Tokenize reviews & count the 10 most frequent words!

Solution

```
review_words_by_movie <- reviews[, .(short_review)] %>%
    unnest_tokens(output = word, input = short_review)

review_words_by_movie[, .N, by = "word"][order(-N)] %>%
    head(6)
```

```
##     word   N
## 1:   the 806
## 2:     a 416
## 3:   and 348
## 4:    of 340
## 5:  bond 297
## 6:    to 275
```

Not to useful, right? Do not worry. . .

Stop words

“. . . stop words are words that are not useful for an analysis, typically extremely common words such as “the”, “of”, “to”, and so forth in English." - tidytextmining

Stop words

```
stop_words <- tidytext::stop_words %>%
    data.table() %>%
    .[lexicon == "onix"]

stop_words[, word] %>% head(10)
```

```
##  [1] "a"        "about"   "above"   "across"  "after"   "again"   "against"
##  [8] "all"      "almost"  "alone"
```

Stop words

You can remove stop words from your data by an “anti join” to the original dataset:

```
review_words_by_movie <- review_words_by_movie %>%
    .[!stop_words, on = "word"]
```

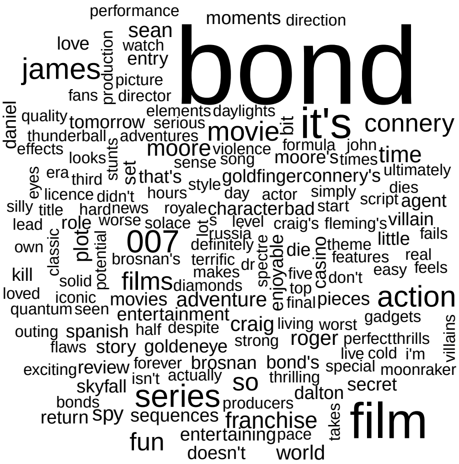
Most frequent words

```
word_freq <- review_words_by_movie %>%
    .[, .(num_occurance = .N), by = "word"] %>%
    .[order(-num_occurance)] %T>%
    head(10)
```

Something fancy - wordclouds

```
wordcloud::wordcloud(
    words = word_freq[num_occurance > 5, word],
    freq = word_freq[num_occurance > 5, num_occurance]
)
```

Something fancy - wordclouds



N-grams

“. . . we can also use the function to tokenize into consecutive sequences of words, called n-grams. By seeing how often word X is followed by word Y, we can then build a model of the relationships between them.” - tidytextmining

Bigram example

```
review_bigrams_by_movie <- reviews[, .(short_review)] %>%
    unnest_tokens(
        output = bigram, input = short_review,
        token = "ngrams", n = 2
    )

review_bigrams_by_movie[, .N, bigram][order(-N)] %>%
    head(10)
```

Bigram example

```
##          bigram   N
##  1:      of the 101
##  2:      in the  78
##  3: james bond  45
##  4:        is a  42
##  5:      to the  40
##  6:      one of  34
##  7:    the best  34
##  8:    the bond  33
##  9:    the film  32
## 10: the series  31
```

Exercise

Remove bigrams that have the first or second word on the stopword list!

Stop words and bigrams

```
stopwords <- stop_words[, word]
```

Regex should be as precise as possible. Example: “on” vs “bond”

```
grep(
    pattern = paste0(stopwords, collapse = "|"),
    "bond",
    value = TRUE
)
```

```
## [1] "bond"
```

Demo: "^" and "$"

```
test <- c("i am", "james", "ma'am")
grepl("am", test)
```

```
## [1] TRUE TRUE TRUE
```

```
grepl("^am|am$", test)
```

```
## [1]  TRUE FALSE  TRUE
```

```
grepl("^am|am$", test)
```

```
## [1]  TRUE FALSE  TRUE
```

```
grepl("^am[[:space:]]|[[:space:]]am$", test)
```

```
## [1]  TRUE FALSE FALSE
```

Solution

```
stopwords_regex <- paste(
    paste(paste0("^", stopwords, "[[:space:]]"), collapse = "|"),
    paste(paste0("[[:space:]]", stopwords, "$"), collapse = "|"),
    collapse = "|"
)

review_bigrams_by_movie <- review_bigrams_by_movie %>%
    .[!grepl(stopwords_regex, bigram)]

review_bigrams_by_movie[, .N, bigram][order(-N)] %>%
    head(10)
```

Solution

```
##              bigram  N
##  1:     james bond 45
##  2:      bond film 30
##  3:     bond films 19
##  4:     bond movie 16
##  5:    bond series 14
##  6:   daniel craig 14
##  7:   sean connery 14
##  8:    roger moore 14
##  9:         it’s a 13
## 10: casino royale 10
```

Section 3

Sentiment analysis

Sentiment analysis

... is computationally identifying and categorizing opinions and attitude in text. Typically, we are interested in negative-neutral-positive sentiments, but other scales are possible well.

Sentiment lexicons

```
tidytext::get_sentiments() %>% head()
```

```
## # A tibble: 6 x 2
##   word       sentiment
##   <chr>      <chr>
## 1 2-faces    negative
## 2 abnormal   negative
## 3 abolish    negative
## 4 abominable negative
## 5 abominably negative
## 6 abominate  negative
```

Sentiment lexicons - afinn

```
get_sentiments(lexicon = "afinn") %>% head(6)
```

```
## # A tibble: 6 x 2
##   word        value
##   <chr>       <dbl>
## 1 abandon        -2
## 2 abandoned      -2
## 3 abandons       -2
## 4 abducted       -2
## 5 abduction      -2
## 6 abductions     -2
```

Sentiment lexicons - bing

```
get_sentiments(lexicon = "bing") %>% head(6)
```

```
## # A tibble: 6 x 2
##   word        sentiment
##   <chr>       <chr>
## 1 2-faces     negative
## 2 abnormal    negative
## 3 abolish     negative
## 4 abominable negative
## 5 abominably negative
## 6 abominate  negative
```

Sentiment lexicons - nrc

```
get_sentiments(lexicon = "nrc") %>% head(6)
```

```
## # A tibble: 6 x 2
##   word      sentiment
##   <chr>     <chr>
## 1 abacus    trust
## 2 abandon   fear
## 3 abandon   negative
## 4 abandon   sadness
## 5 abandoned anger
## 6 abandoned fear
```

Sentiment lexicons - loughran

```
get_sentiments(lexicon = "loughran") %>% head(6)
```

```
## # A tibble: 6 x 2
##   word         sentiment
##   <chr>        <chr>
## 1 abandon      negative
## 2 abandoned    negative
## 3 abandoning   negative
## 4 abandonment  negative
## 5 abandonments negative
## 6 abandons     negative
```

Example

```
sentiment_scores <- get_sentiments(lexicon = "afinn") %>% data.table()
sentiment_scores[, .N, keyby = "value"]
```

```
##      value   N
##  1:     -5  16
##  2:     -4  43
##  3:     -3 264
##  4:     -2 966
##  5:     -1 309
##  6:      0   1
##  7:      1 208
##  8:      2 448
##  9:      3 172
## 10:      4  45
## 11:      5   5
```

Example

```
sentiment_scores[value == -5] %>% head(3)
```

```
##         word value
## 1:   bastard    -5
## 2:  bastards    -5
## 3:     bitch    -5
```

Example

```
sentiment_scores[value == 0] %>% head(3)
```

```
##           word value
## 1: some kind     0
```

Example

```
sentiment_scores[value == 5] %>% head(3)
```

```
##              word value
## 1: breathtaking     5
## 2:       hurrah     5
## 3:  outstanding     5
```

Example

```
review_sentiment_scores <- reviews[, .(short_review)] %>%
    unnest_tokens(output = word, input = short_review, drop = FALSE) %>%
    merge(sentiment_scores, by = "word") %>%
    .[, .(sentiment_score = sum(value)), by = c("short_review")]
```

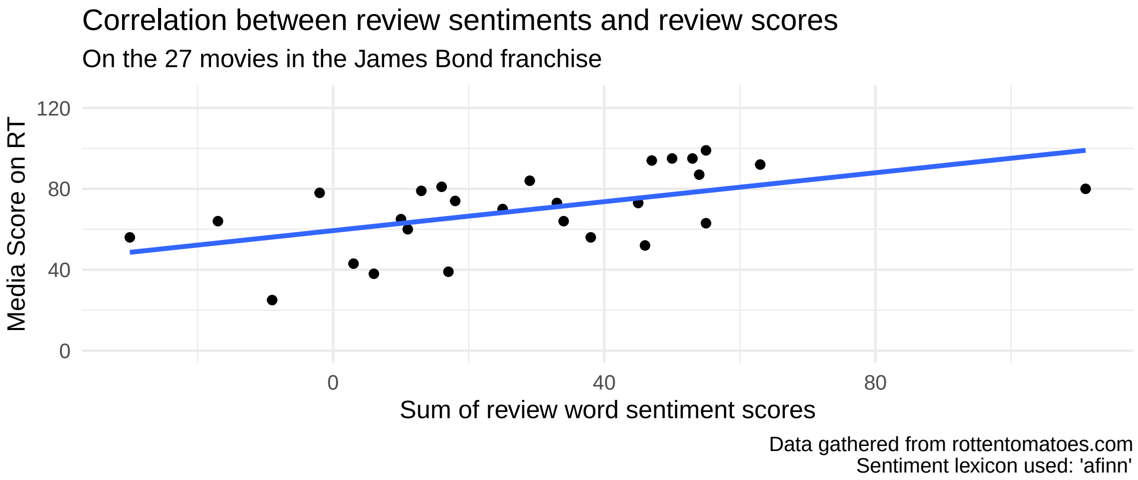
note: be careful with merging! not all words have sentiment scores

Example

```
movies_w_sentiments <- reviews %>%
    merge(review_sentiment_scores, by = "short_review") %>%
    .[,
        .(movie_sentiment_score = sum(sentiment_score)),
        by = c("movie_title", "media_score")
    ]
```

note: normally pls don't join on text fields...

Example


Correlation between review sentiments and review scores
On the 27 movies in the James Bond franchise
Media Score on RT
120
80
40
0
0
40
80
Sum of review word sentiment scores
Data gathered from rottentomatoes.com
Sentiment lexicon used: 'afinn'

Section 4

Homework

Homework

- Work on the `data/marvel_reviews_raw.csv` file!
- Clean data: media scores should be an integer.
- Get the "main" (first in order of appearance) actor for each movie
  - Hint: remember capturing groups!
  - (But you can choose any method you prefer)
- Based on the "bing" lexicon (!), calculate the some type of sentiment score for each movie
- By actor, plot avg sentiment score against their avg media score
- Deadline: start of next class, November 10th

For 2 bonus points

- Repeat the previous exercise with the following modifications:
- Instead of using just the main actor, you should capture the whole cast
- For each cast member, take the movie's sentiment score (calculated as before)
- Calculate the avg. sentiment score by actor
- Visualize from best to worst the sentiment the actors' movies got on average