

Inleiding text mining in R

sessie 01

Longhow Lam

Contents

1	Inleiding	1
2	The basics	2
2.1	Reguliere expressies	2
2.2	functies uit het package stringr	3
2.3	Wordclouds	3
2.4	Talen	4
2.5	PDF files	4
3	Tokenization , part-of-speech tagging en lemmatization	4
4	Het text2vec package	4
4.1	iterators	4
4.2	vocabulaires	5
4.3	term document matrices	5
4.4	afstanden	5
4.5	Collocations	5
5	GLMNET	5
6	Termen prijzen in huis beschrijvingen	5
7	Sentiment prediction	8
7.1	Nederlands voor gedefinieerd	8
7.2	training sentiment	8
8	Glove word embeddings	10
9	Latente Dirichlet Allocatie	10

1 Inleiding

Deze cursus geeft een inleiding in text mining in R. De volgende punten zullen worden behandeld:

- String en text basics (stringr manipulaties, word clouds)
- Tokenization, lematization
- Het text2vec package
- Distance measures
- Latent Direchlet Allocation
- Latent semantic indexing
- GLMnet modellen
- Sentiment analyse

- word embeddings (glove)

We gebruiken in deze cursus een aantal data sets die ik eerder al eens gescraped heb. Het betreft: Iens, GTST, Jaap en Ajax, RTL en NOS nieuws.

Er wordt uitgegaan van een basis kennis van R. Deze cursus bestaat uit twee sessies van ongeveer 3 uur. Er zijn wat begeleidende powerpoint slides.

2 The basics

Character data in R kan je bewerken/manipuleren met het **stringr** package. Voordat we daar verder op in gaan is het handig om te weten wat reguliere expressies zijn.

2.1 Reguliere expressies

Dit is een soort ‘mini’ taaltje om character patronen te specificeren om er vervolgens bijvoorbeeld op te zoeken of filteren. Eerst wat dummy character data.

```
test = c("Mijn nummer is 06-12345678", "dit is .. een 1628EP postcode test", "foutje?: 0234XD", "dit is  
test
```

Een paar voorbeelden van reguliere expressies.

```
## een digit, en precies 1 digit
patroon = "\\d"
str_extract(test, patroon)

## 1 of meerdere digits
patroon = "\\d+"
str_extract(test, patroon)

## precies twee letters
patroon = "[[:alpha:]]{2}"
str_extract(test, patroon)

## Een postcode
patroon = "\\d{4}\\s?[[:alpha:]]{2}"
str_extract(test, patroon)

## Maar een postcode begint niet met een 0
patroon = "[1-9]\\d{3}\\s?[[:alpha:]]{2}"
str_extract(test, patroon)

## special punctuation characters !"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~
patroon = "[[:punct:]]"
str_extract(test, patroon)
```

Sommige mensen zijn de wildcard notatie gewend om te zoeken in strings, deze wildcard notatie kan je vertalen naar reguliere expressies met **glob2rx**. Een paar voorbeelden zullen hieronder geven.

```
patroon = glob2rx("*23*")
str_extract(test, patroon)
```

Voor een cheatsheet over reguliere expressies zie [regex cheatsheet](#)

2.2 functies uit het package stringr

In het package stringr zijn diverse functies om met characters te werken, een aantal zullen we laten zien

```
testset = titanic::titanic_train %>% select(Name)
## maak extra kolom aan in titanic die namen zoekt
testset = testset %>%
  mutate(
    naamlengte = str_length(Name),
    Ownes = str_detect(Name, 'Owen'),
    Plek = str_locate(testset$Name, "Mr") %>% .[,1],
    Name2 = str_replace(Name, "Mr", "")
  )

str_locate(testset$Name, "Mr")
```

2.3 Wordclouds

Wordclouds zijn nog steeds een ‘mooie’ manier om teksten weer te geven. Er zijn twee packages in R die je kan gebruiken voor wordclouds in R, wordcloud en wordcloud2.

```
ajax = readRDS("data/Ajax.Rds")
ajax = ajax %>% mutate_if(is.factor, as.character)

## pak 5 verslagen
verslagjes = paste(
  ajax$Verslag[1:3],
  collapse = " "
)

wordcounts = tibble(w = stringi::stri_extract_all_words(
  str_to_lower(verslagjes)
) %>%
  unlist
) %>%
group_by(w) %>%
summarise(n=n())

wordcloud(wordcounts$w, wordcounts$n)
wordcloud2(wordcounts)
```

Er zitten wat woorden in die we weg willen halen

```
stopwoorden = c("ajax", "de", "een", "het", "en", "met", "in", "van", "op", "er")

w = stringi::stri_extract_all_words(
  str_to_lower(verslagjes)
) %>%
  unlist

w = w[!w %in% stopwoorden]

wordcounts2 = data.frame(w, stringsAsFactors = FALSE) %>%
group_by(w) %>%
```

```
summarise(n=n()) %>% filter (n>2)
```

```
wordcloud2(wordcounts2)
```

2.4 Talen

In deze cursus ligt de focus op de Nederlandse taal. Soms krijg je teksten waar misschien ook andere talen in staan. Je kan dit detecteren met de `detect_language` functie uit het `cld2` (Google's compact language detector) package

```
# installeer package als het nodig is  
# devtools::install_github("ropensci/cld2")
```

```
cld2::detect_language("To be or not to be")  
cld2::detect_language("ga toch weg, jij rotzak")  
cld2::detect_language(url('https://www.rtlnieuws.nl/nederland/maastunnel-twee-jaar-dicht-van-noord-naar-'))
```

2.5 PDF files

```
pdf_text("RestaurantsBlacklist/99.pdf")
```

3 Tokenization , part-of-speech tagging en lemmatization

```
library(udpipe)  
dl <- udpipe_download_model(language = "dutch")  
dl
```

```
udmodel_dutch <- udpipe_load_model(file = "dutch-ud-2.0-170801.udpipe")  
x <- udpipe_annotate(udmodel_dutch, x = "Ik ging op reis en ik nam mee: mijn laptop, mijn zonnebril en  
x <- as.data.frame(x)  
x
```

```
udmodel_dutch <- udpipe_load_model(file = "dutch-ud-2.0-170801.udpipe")  
x <- udpipe_annotate(udmodel_dutch, x = RTLN$bodie[1])  
x <- as.data.frame(x)  
x
```

4 Het text2vec package

Het werkpaardje voor mij in R voor textmining

4.1 iterators

prepare_fun

$$\min_{\beta_0, \beta} \frac{1}{N} \sum_{i=1}^N w_i l(y_i, \beta_0 + \beta^T x_i) + \lambda \left[(1 - \alpha) \|\beta\|_2^2 / 2 + \alpha \|\beta\|_1 \right]$$

Figure 1:

4.2 vocabulaires

4.3 term document matrices

4.4 afstanden

4.5 Collocations

5 GLMNET

Het R package `glmnet` fit regressie modellen met zogenaamde elastic net penalties. In tegenstelling tot gewone regressie waar parameters ongelimiteerd mogen zijn, minimaliseren we in `glmnet` een loss met penalties, zie onderstaande minimalisatie probleem.

Speciale waarden voor α zijn:

- $\alpha = 1$, lasso regression
- $\alpha = 0$, ridge regression
- $0 < \alpha < 1$, least angle regression

6 Termen prijzen in huis beschrijvingen

Een toepassing van GLMNET, ik heb huisbeschrijvingen van de website jaap.nl gescrept.

```
jaap = readRDS("data/jaap.RDs")
```

We gaan eerst wat data opschonen, niet te korte beschrijvingen, te dure en te goedkope huizen er ook halen.

```
jaap = jaap %>%
  filter(
    longdescription != ""
  ) %>%
  mutate(
    Nwoorden = str_count(longdescription, pattern = '[\\w]+'),
    huisbeschrijving = str_to_lower(longdescription)
  ) %>%
  filter(
    prijs > 50000,
    prijs < 1250000,
    Nwoorden > 20,
    Nwoorden < 1500
  )
```

```

# create an index column
jaap$id = 1:dim(jaap)[1]

##### text mining part #####

## tokenize

jaap_tokens = jaap$huysbeschrijving %>% word_tokenizer

### use the tokens to create an iterator and vocabular
iter = itoken(
  jaap_tokens,
  ids = jaap$id,
  progressbar = TRUE
)

## remove some Dutch stopwords
stw = tm::stopwords("nl")

vocab = create_vocabulary(
  iter,
  stopwords = stw,
  ngram = c(ngram_min = 1L, ngram_max = 3L)
)

pruned_vocab = prune_vocabulary(vocab, doc_proportion_min = 0.0035 )

#### create document term matrix
vectorizer = vocab_vectorizer(pruned_vocab)
dtm = create_dtm(iter, vectorizer)
dim(dtm)

# define tfidf model
tfidf = TfIdf$new()
# fit model to train data and transform train data with fitted model
dtm_tfidf = fit_transform(dtm, tfidf)

##### fit price models #####

## split into test and train

Ntrain = floor(0.8*nrow(dtm))
tr.idx = sample(1:nrow(dtm), size = Ntrain)

dtm_train = dtm[tr.idx,]
dtm_train_tfidf = dtm_tfidf[tr.idx,]
dim(dtm_train)
dim(dtm_train_tfidf)

dtm_test = dtm[-tr.idx,]
dtm_test_tfidf = dtm_tfidf[-tr.idx,]

```

```

dim(dtm_test)

target_train = jaap$prijs[tr.idx]
target_test = jaap$prijs[-tr.idx]

#### glmnet

glmnet_prijmodel = cv.glmnet(
  x = dtm_train,
  y = target_train,
  # L1 penalty
  alpha = 1,
  # high value is less accurate, but has faster training
  thresh = 1e-4,
  # again lower number of iterations for faster training
  maxit = 1e4,
  type.measure="mae"
)

plot(glmnet_prijmodel)
print(paste("mse =", round(min(glmnet_prijmodel$cvm), 4)))

### extract terms with 'large' negative or positive value

tmp = coef(glmnet_prijmodel)
wordvalue = as.numeric(tmp)
names(wordvalue) = tmp@Dimnames[[1]]

### 100 'largest' and 'smallest' terms
neg100 = sort(wordvalue)[1:100]
pos100 = sort(wordvalue, decreasing = TRUE)[1:100]

outterms = tibble::tibble(
  NegWords = names(neg100),
  NegValue = neg100,
  PosWords = names(pos100),
  PosValues = pos100
)

## R squared calculation on the hold out test set
test_huizenprijs = predict(glmnet_prijmodel, newx = dtm_test)[,1]
R2 = 1 - sum((target_test - test_huizenprijs)^2) / sum((target_test - mean(target_test))^2)
R2

##### xgboost #####

param = list(
  max_depth = 10,
  nthread = 4
)

xgbmodel = xgboost(

```

```

params = param,
data = dtm_train,
label = target_train,
nrounds = 150,
print_every_n = 5L
)

## some model diagnostics
varimp = xgb.importance(colnames(dtm_train), model = xgbmodel)
head(varimp, 25)

xgb.plot.tree(colnames(dtm_train), model = xgbmodel, n_first_tree = 3)

p = xgb.plot.multi.trees(
  model = xgbmodel,
  feature_names = colnames(dtm_train),
  features_keep = 30
)
print(p)

## R squared calculation on the hold out test set
test_huizenprijs = predict(xgbmodel, newdata = dtm_test)
R2 = 1 - sum((target_test - test_huizenprijs)^2) / sum((target_test - mean(target_test))^2)
R2

```

7 Sentiment prediction

Ik maak onderscheid tussen voorgedefineerd sentiment met woorden lijsten, en voorspelling aan de hand van training data.

7.1 Nederlands voor gedefinieerd

Voor Nederlands kan je sentiment bepalen via het Python package pattern.nl (python 2.7 alleen), dit kan je in R aanroepen via reticulate

```

library(reticulate)
pattern = import("pattern.nl")
pattern$sentiment("Ik heb deze laptop gekocht wat een vreselijk slecht apparaat")

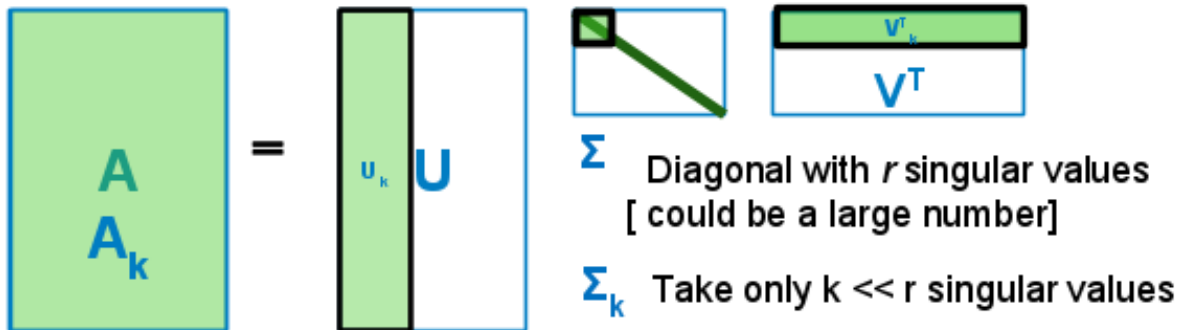
pattern$sentiment("Ik heb deze laptop gekocht. Vreselijk goed ding")
pattern$sentiment("Ik heb deze laptop gekocht, niet eens zo slecht")

```

7.2 training sentiment

Voor het trainen van sentiment heb je gelabelde data nodig. per document heb je een score of POS/NEG label nodig. Ik gebruik hier als voorbeeld de IENS reviews die ik een tijdje geleden heb gescrapet. Per review

Matrix SVD decomposition: $A = U\Sigma V^T$



A datapoint d can now be represented by k dimensional point $\hat{d} = U_k^T d$.

Figure 2:

heb je ook een cijfer die de reviewer heeft gegeven. Deze gebruik ik als target.

```
Iens = readRDS("data/IensReviews.rds")
```

Als eerste controleren op taal en voor deze analyse pakken we alleen de NL reviews.

```
talen = detect_language(Iens$Review)
table(talen)
```

Doe glmnet direct op dtm

De term document matrix is ontzetten sparse, om deze wat compacter te maken, zonder veel informatie verlies kunnen we een SVD trick toe passen.

Doe een svd en daarna een predictive model

Als voorbeeldje kan je ook eens kijken naar mijn zzzontje :-) plaatje SVD.... uit mijn slide

```
## enkele teksten evatten wat rare characters
Sys.setlocale('LC_ALL', 'C')

it = itoken(Iens$Review)
voc = create_vocabulary(it) %>%
  prune_vocabulary(
    doc_proportion_max = 0.1,
    term_count_min = 5
  )

vectorizer = vocab_vectorizer(voc)
dtm = create_dtm(it, vectorizer)
dim(dtm)
```

De dtm is een vrij grote sparse matrix met veel nullen

```
sum(dtm > 0) / (1.0*dim(dtm)[1]*dim(dtm)[2])
```

Pas de SVD toe en neem nu 50 dimensies

```
tfidf = TfIdf$new()
lsa = LSA$new(n_topics = 50)

# pipe friendly transformation
doc_embeddings = dtm %>%
  fit_transform(tfidf) %>%
  fit_transform(lsa)

dim(doc_embeddings)
doc_embeddings[1:25,]
```

8 Glove word embeddings

Een word embedding is een rij getallen die een woord representeert. Deze rij getallen / vectoren worden getrained uit een corpus. Zie bijvoorbeeld de site van het text2vec package

In een traditionele term document matrix zijn de volgende twee documenten vrij ongerelateerd

- De president spreekt met de media
- Trump praat tegen de pers

maar uit een grotere set documenten zou je kunnen leren dat De president en Trump gerelateerd zijn en dat geldt ook voor de media en de pers.

We gaan word embeddings leren op een grote verzameling van RTL Nieuws artikelen

EMBED SCRIPT

Check mijn shiny app

<http://145.131.21.163:3838/sample-apps/wordembeddings/>

en code op mijn github

9 Latente Dirichlet Allocatie

Hoe kan je nou een beeld krijgen van veel documenten zonder deze allemaal zelf te lezen? Een manier is een Latente Dirichlet Allocatie (LDA) model fitten op de documenten. Met deze techniek kan je een groot aantal documenten indelen in een kleiner aantal topics. Elk topic kenmerkt zich door een aantal woorden, waarvan sommige belangrijker dan anderen.

Het LDA model is een generatief model, het gaat er van uit dat documenten volgens een bepaald stramen worden gegenereerd. Heel in het kort:

Kies een kans mixture van (zeg even 3) topics, bijvoorbeeld 1/4 Topic A, 1/4 Topic B en 1/2 Topic C. Elk topic heeft weer een kansverdeling van woorden, bijvoorbeeld:

Topic A 30% ministerie, 20% politie, 50% veiligheid Topic B 25% buitenhof, 25% Den haag, 10% kamer, 40% politiek Topic C 15% inenten, 30% bijbel 30% tegen 25% geloof

Een document wordt dan als volgt gegenereerd: Kies met bovenstaande kansen de topics, en dan per gekozen topic kies een aantal woorden volgens de woorden kansverdeling van het desbetreffende topic.

Van te voren kennen we deze kansverdelingen niet, maar gegeven een aantal documenten proberen we deze kans verdelingen te achterhalen. Dit is het trainen/fitten van een LDA model.

In R kan je met het `text2vec` package ook LDA modellen trainen. We zullen hier de GTST data set gebruiken. Dit zijn 4090 samenvattingen van GTST afleveringen van 1-oktober-1990 tot en met 31-05-2012.

```
GTST = readRDS("data/GTST_samenvattingen.RDs")
```

Laten we een topic model maken met 15 topics

```
GTST$id = 1:dim(GTST)[1]
tokens = GTST$text_of_1month %>%
  tolower %>%
  word_tokenizer

# stopwoorden er uit. Ik heb een lijstje die ik nog aanvul
stopwoorden = c(
  readRDS("data/stopwoorden.RDs"),
  letters)

it = itoken(tokens, ids = GTST$id, progressbar = FALSE)
voc = create_vocabulary(it, stopwords = stopwoorden) %>%
  prune_vocabulary(
    term_count_min = 5
  )

vectorizer = vocab_vectorizer(voc)
dtm = create_dtm(it, vectorizer)
dim(dtm)

lda_model = LDA$new(
  n_topics = 15,
  doc_topic_prior = 0.1,
  topic_word_prior = 0.01
)

doc_topic_distr = lda_model$fit_transform(
  x = dtm,
  n_iter = 1000,
  convergence_tol = 0.001,
  n_check_convergence = 25,
  progressbar = FALSE
)
```

Wat zit er in output object? Het is een verdeling van topics over alle documenten. Voor document 1 hebben we de volgende topic verdeling

```
dim(doc_topic_distr)

barplot(doc_topic_distr[1, ], xlab = "topic",
  ylab = "proportion", ylim = c(0, 1),
  names.arg = 1:ncol(doc_topic_distr))
```

Het `lda_model` object bevat nu ook de woorden die bij de topics horen

```
lda_model$get_top_words(n = 20, lambda = 1)
```

Plot of topics in twee dimensionale ruimte met behulp van het LDAvis package

```
lda_model$plot()
```

9.0.1 Scoren van nieuwe documenten

Als je een LDA model hebt gemaakt kan je deze gebruiken om nieuwe documenten te scoren en zo van deze documenten de meest waarschijnlijke topic bepalen.

```
## pak nu als test gewoon weer wat GTST afleveringen, zeg evel de eerste 10
```

```
TEST = GTST[1:10,]
```

```
new_dtm = itoken(  
  TEST$text_of_1month,  
  tolower,  
  word_tokenizer,  
  ids = TEST$id  
) %>%  
create_dtm(vectorizer)
```

```
dim(new_dtm)
```

```
new_doc_topic_distr = lda_model$transform(new_dtm)
```

De matrix new_doc_distr bevat per document de kans op een topic, hoe kan je per document de meest waarschijnlijke topic halen?

```
## kijk eens naar een document, zeg 6
```

```
new_doc_topic_distr[6,]
```

```
which.max(new_doc_topic_distr[6,])
```

```
## gebruik purrr om een functie toe te passen op rijen van een matrix
```

```
rows <- function(m){ map(seq_len(nrow(m)), ~m[.,] ) }
```

```
purrr::map_int(rows(new_doc_topic_distr), which.max)
```