M2 - Exam - Network and NLP

Malika Kuhlman & Michael Dybdahl

10/24/2019

Contents

OBS The notebook cannot be run in google Colab because of the *topicmodels* package.

Loading the *stop_words* that we will use in the tokenization.

```
set.seed(9)
data("stop_words")
```

Definition of a problem statement and a short outline of the implementation

Problem statement

How are the words in the text-reviews and based on a tidy format of the reviews, can we then compute any topics based on this? Can we compute a model that predict if a review is rated as a good review, based or linked to the NLP results?

Description of data acquisition / how it was collected (by you or the publisher of the data)

The dataset contains reviews of some of amazons products, the dataset can be reached on this website https://data.world/datafiniti/consumer-reviews-of-amazon-products

There are serveral datasets about the same topic, but with different amount of observations, we chose the dataset with 5000 observations. The dataset contains 24 variables and 5.000 observations (reviews).

```r
reviews <- read_csv("https://query.data.world/s/foqg5o75hazenbwqdug534atoqiyp3")
```

Have a look at the data structure

```r
glimpse(reviews)
```

```
## Observations: 5,000
## Variables: 24
## $ id                  <chr> "AVqVGZNvQMlgsOJE6eUY", "AVqVGZNvQMlgsOJE6...
## $ dateAdded           <dttm> 2017-03-03 16:56:05, 2017-03-03 16:56:05,...
## $ dateUpdated         <dttm> 2018-10-25 16:36:31, 2018-10-25 16:36:31,...
## $ name                <chr> "Amazon Kindle E-Reader 6\" Wifi (8th Gene...
## $ asins               <chr> "B00ZV9PXP2", "B00ZV9PXP2", "B00ZV9PXP2", ...
## $ brand               <chr> "Amazon", "Amazon", "Amazon", "Amazon", "A...
## $ categories          <chr> "Computers,Electronics Features,Tablets,El...
## $ primaryCategories   <chr> "Electronics", "Electronics", "Electronics...
## $ imageURLs           <chr> "https://pisces.bbystatic.com/image2/BestB...
## $ keys                <chr> "allnewkindleereaderblack6glarefreetouchsc...
## $ manufacturer        <chr> "Amazon", "Amazon", "Amazon", "Amazon", "A...
## $ manufacturerNumber  <chr> "B00ZV9PXP2", "B00ZV9PXP2", "B00ZV9PXP2", ...
## $ reviews.date        <dttm> 2017-09-03 00:00:00, 2017-06-06 00:00:00,...
## $ reviews.dateAdded   <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA...
## $ reviews.dateSeen    <chr> "2018-05-27T00:00:00Z,2017-09-18T00:00:00Z...
## $ reviews.doRecommend <lgl> FALSE, TRUE, TRUE, TRUE, TRUE, FALSE, TRUE...
## $ reviews.id          <dbl> NA, NA, NA, 177283626, NA, NA, 187043823, ...
## $ reviews.numHelpful  <dbl> 0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, ...
## $ reviews.rating      <dbl> 3, 5, 4, 5, 5, 5, 5, 4, 5, 5, 5, 5, 5, 5, ...
## $ reviews.sourceURLs  <chr> "http://reviews.bestbuy.com/3545/5442403/r...
## $ reviews.text        <chr> "I thought it would be as big as small pap...
## $ reviews.title       <chr> "Too small", "Great light reader. Easy to ...
## $ reviews.username    <chr> "llyyue", "Charmi", "johnnyjojojo", "Kdper...
## $ sourceURLs          <chr> "https://www.newegg.com/Product/Product.as...
```

1. Data preparation (general)

1.1. Data cleaning

We see that the data contains of 5000 observations and 24 variables, we are not going to use all of variables, therefore we now select the one we will use. Therefore we will only describe the variables we will use.

```r
reviews <- reviews %>%
  select(reviews.username,
         dateAdded,
         name,
         primaryCategories,
         reviews.numHelpful,
         reviews.rating,
         reviews.text)
```

- **reviews.username:** contain the reviewsers username.
- **dateAdded:** coontain the date the reviews was written.
- **name:** contain the name of the product.
- **primaryCategories:** contain the PRimary category of the product, there are four different primary categories in our data; *Electronics*, *Electronics, Hardware*, *Electronics, Media*and *Office Supplies, Electronics*.
- **reviews.numHelpful:** contain a the value wether or not a reader found the review helpful.
- **reviews.text:** contain the reviews of the product.

Now we check for *NA* and drop them if any exists.

```
sum(is.na(reviews)) # First we check for NA values in the whole datasat.
```

```
## [1] 0
```

```
reviews = reviews %>%
  drop_na()
```

1.2. Recoding

We saw earlier that some of the variable are named *reviews.X*, this we want to change.

```
reviews <- reviews %>%
  rename(helpful = reviews.numHelpful,
         rating = reviews.rating,
         text = reviews.text,
         username = reviews.username,
         date = dateAdded)
```

Have a look at the dataset now.

```
glimpse(reviews)
```

```
## Observations: 5,000
## Variables: 7
## $ username          <chr> "llyyue", "Charmi", "johnnyjojojo", "Kdperry...
## $ date              <dttm> 2017-03-03 16:56:05, 2017-03-03 16:56:05, 2...
## $ name              <chr> "Amazon Kindle E-Reader 6\" Wifi (8th Genera...
## $ primaryCategories <chr> "Electronics", "Electronics", "Electronics",...
## $ helpful           <dbl> 0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0,...
## $ rating            <dbl> 3, 5, 4, 5, 5, 5, 5, 4, 5, 5, 5, 5, 5, 5, 5,...
## $ text              <chr> "I thought it would be as big as small paper...
```

Now we have a dataset with 4987 observations and 9 variables

For later use, we add a new *ID* number to each username, since som users have reviewed more than one product.

```
username_df <- reviews %>%
  select(username) %>%
  distinct(username,
           .keep_all = TRUE)

username_df$ID <- seq.int(nrow(username_df))
```

Now we join the new ID with the dataset.

```r
reviews <- reviews %>%
  left_join(username_df, by = "username") %>%
  select(ID, everything())

reviews$ID2 <- seq.int(nrow(reviews))
```

We change the rating from *doubble* to *factor*.

```r
reviews$rating <- as.factor(reviews$rating)
```

We change the date from *dttm* to *date*.

```r
reviews$date <- as.Date(as.POSIXct(reviews$date))
```

We create a new variable called *year*, where we put the year of every review.

```r
reviews <- reviews %>%
  mutate(year = as.factor(year(date)))
```

Now we want to look into wich years the dataset has reviews from and how many reviews there are written in each year.

```r
reviews %>%
  group_by(year) %>%
  count()
```

```
## # A tibble: 4 x 2
## # Groups:   year [4]
##   year      n
##   <fct> <int>
## 1 2015     44
## 2 2016   1026
## 3 2017   2495
## 4 2018   1435
```

We see we have four years

- **2015** with 44 reviews
- **2016** with 1026 reviews
- **2017** with 2495 reviews
- **2018** with 1435 reviews

2. Natural Language Processing

We now do all the NLP here, and afterwards we have a section containing all the Network analysis.

2.1. NLP - preparation

2.1.1. Tokenization

In this section we are going to do a tokenization by removing *stop words*, *meaningless words*, *non-alphanumeric characters*, and so on.

First we seperate every word in each review text.

```
reviews_tidy <- reviews %>%
  select(ID2, ID, text) %>%
  unnest_tokens(output = word, input = text)
```

```
head(reviews_tidy)
```

```
## # A tibble: 6 x 3
##     ID2    ID word
##   <int> <int> <chr>
## 1     1     1 i
## 2     1     1 thought
## 3     1     1 it
## 4     1     1 would
## 5     1     1 be
## 6     1     1 as
```

Then we remove *stop words*, using the stop_words we loaded in the begining.

```
reviews_tidy %<>% # I use "%<>%" which both assigns and pipe - instead of first assigning and then pipi
  anti_join(stop_words, by = "word") # I use anti_join instead of filter, because it should be a bit fa
```

Now we have a look at the most used words.

```
reviews_tidy %>%
  count(word, sort = TRUE) %>% # Count "word". "sort = TRUE" means that it will sort the words in desce
  head(20)
```

```
## # A tibble: 20 x 2
##    word        n
##    <chr>   <int>
##  1 tablet   1309
##  2 love     1090
##  3 easy      822
##  4 bought    785
##  5 kindle    764
##  6 amazon    694
##  7 echo      693
##  8 alexa     513
##  9 loves     506
## 10 screen    500
## 11 price     477
## 12 product   470
## 13 fire      448
## 14 kids      441
## 15 music     404
## 16 apps      378
## 17 device    359
## 18 books     357
## 19 games     344
## 20 time      337
```

We see that words like *tablet*, *love* and *easy* are the most used words. Since the reviews are about *Electronics* we think thats why the word *Tablet* are the most used word togehter with *kindle*, *alexa* and *echo* wich all are names of some of the products.

After a closer look into the words, we found some words that could be removed.

```
own_stopwords <- tibble(word = c("bought", "yrs", "yokod", "yo", "xm", "wouldnt", "wouldn", "woo", "wit
                        lexicon = "OWN")
```

```
reviews_tidy %<>% # I use "%<>%" which both assigns and pipe - instead of first assigning and then pipi
  anti_join(own_stopwords, by = "word") # I use anti_join instead of filter, because it should be a bit
```

Now we look at the top words again.

```
reviews_tidy %>%
  count(word, sort = TRUE) %>% # Count "word". "sort = TRUE" means that it will sort the words in desce
  head(20)
```

```
## # A tibble: 20 x 2
##    word        n
##    <chr>   <int>
##  1 tablet   1309
##  2 love     1090
##  3 easy      822
##  4 kindle    764
##  5 amazon    694
##  6 echo      693
##  7 alexa     513
##  8 loves     506
##  9 screen    500
## 10 price     477
## 11 product   470
## 12 fire      448
## 13 kids      441
## 14 music     404
## 15 apps      378
## 16 device    359
## 17 books     357
## 18 games     344
## 19 time      337
## 20 read      331
```

We see that the top words are different, since we removed some of our own stop words.

Here we remowe numbers.

```
reviews_tidy %<>%
  mutate(word = trimws(gsub("[^\\s]*[0-9][^\\s]*", "", word, perl = T))) %>%
  filter(str_length(word) > 1) # this filter out the words that are blank.
```

Here we remowe non-alphanumeric characters.

```
reviews_tidy %<>%
  mutate(word = word %>% str_remove_all("[^[:alnum:]]") ) %>% # alnum = Alphanumeric characters.
  filter(str_length(word) > 1) # filter out words with 1 character.
```

Now we look at the top 20 of the least used words.

```
reviews_tidy %>%
  count(word, sort = TRUE) %>% # Count "word". "sort = TRUE" means that it will sort the words in desce
  top_n(-20)
```

```
## # A tibble: 2,157 x 2
##    word              n
##    <chr>         <int>
##  1 abke              1
##  2 absent            1
##  3 absorbs           1
##  4 accelerometer     1
##  5 accept            1
##  6 acceptable        1
##  7 accesible         1
##  8 accesses          1
##  9 accessible        1
## 10 accident          1
## # ... with 2,147 more rows
```

We see that the list contains words, that are only used once, this could indicate that some mispelling have happend or its not a very usefull word. Therefore we remove words used less than two times.

```
reviews_tidy = reviews_tidy %>%
  add_count(ID, word, name = "nword") %>%
  filter(nword > 1) %>%
  select(-nword)
```

Have a look again

```
reviews_tidy %>%
  count(word, sort = TRUE) %>% # Count "word". "sort = TRUE" means that it will sort the words in desce
  top_n(-20)
```

```
## # A tibble: 872 x 2
##    word             n
##    <chr>        <int>
##  1 accessed         2
##  2 accessory        2
##  3 accidentally     2
##  4 accidently       2
##  5 accounts         2
##  6 actions          2
##  7 activate         2
##  8 activated        2
##  9 activities       2
## 10 acts             2
## # ... with 862 more rows
```

We see some different words now, but we still find these words usefull and very informative.

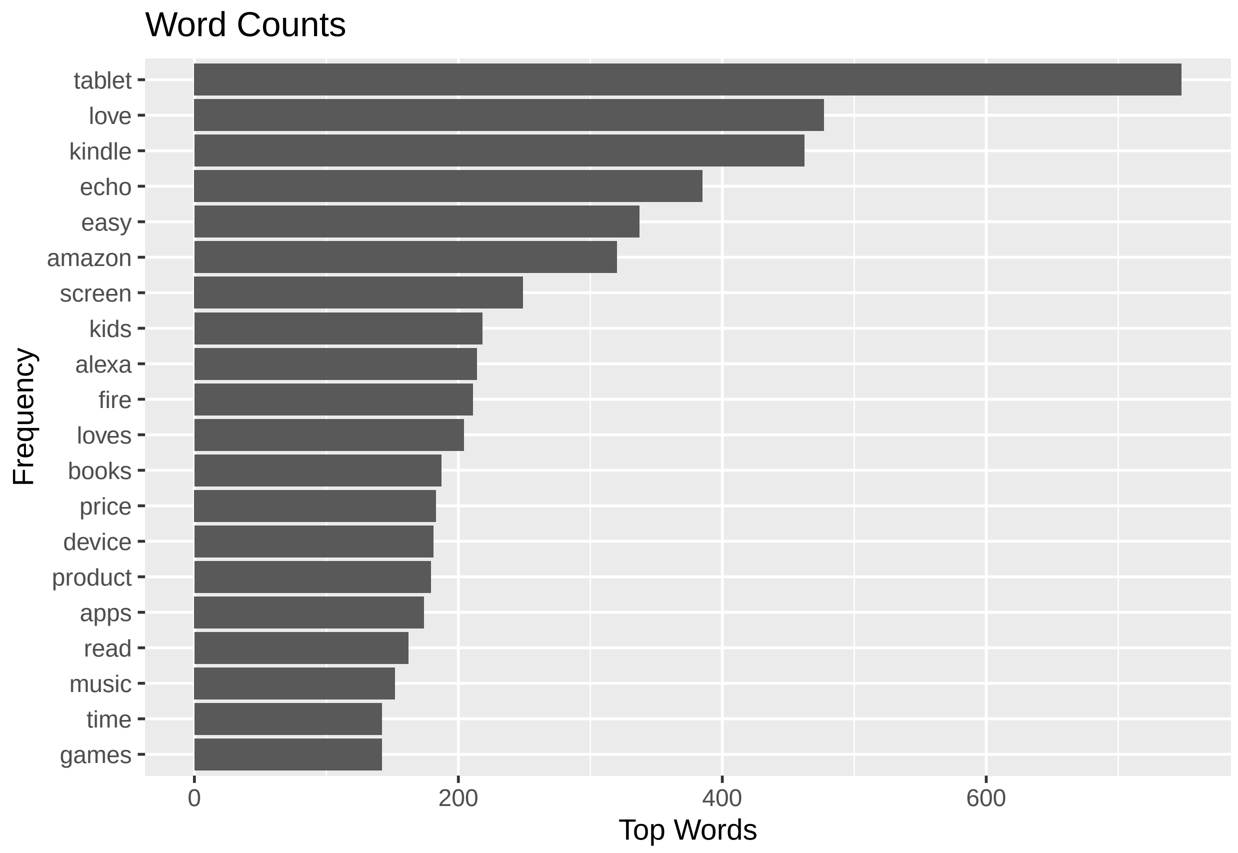
We now make varible only containing the words in descending order of how much they are used.

```
topwords <- reviews_tidy %>%
  count(word, sort = TRUE)

head(topwords)
```

```
## # A tibble: 6 x 2
##   word       n
##   <chr>  <int>
## 1 tablet   748
## 2 love     477
## 3 kindle   462
## 4 echo     385
## 5 easy      337
## 6 amazon    320
```

We use this new *topwords* variable to plot the topwords.

```
topwords %>%
  top_n(20, n) %>%
  ggplot(aes(x = word %>% fct_reorder(n), y = n)) + #fct_reorder factor the counts + it reorder the cou
  geom_col() +
  coord_flip() +
  labs(title = "Word Counts",
       x = "Frequency",
       y = "Top Words")
```



We see that the word *tablet* is way more than the rest. If we have a look at the other words, we see that overall the words are mainly the name of some electronics or the words are mainly about electronics.

For later use, we save this reviews_tidy with a different name

```
reviews_tidy_ML = reviews_tidy
```

2.2. Simple vectorization (Tf-idf)

In this section we add the *tf, term of frequency*, *idf, inverse document frequency* and *tf-idf, term frequency–inverse document frequency* to the tidy data.

```
reviews_tidy <- reviews_tidy %>%
  count(ID, word) %>%
  bind_tf_idf(ID, word, n) # I use this function to add the "tf", "idf" & "tf_idf" values.
```

We have a look at the dwords with the highest *tf_idf* values

```r
reviews_tidy %>%
  arrange(desc(tf_idf))
```

```
## # A tibble: 7,828 x 6
##       ID word         n    tf   idf tf_idf
##    <int> <chr>    <int> <dbl> <dbl>  <dbl>
##  1   102 äõ           2     1  7.51   7.51
##  2   238 äôre         2     1  7.51   7.51
##  3  1167 response     2     1  7.51   7.51
##  4  1250 bot          2     1  7.51   7.51
##  5  1334 charges      2     1  7.51   7.51
##  6  1412 mines        2     1  7.51   7.51
##  7  2150 reliable     2     1  7.51   7.51
##  8  2534 concept      2     1  7.51   7.51
##  9  2881 launcher     2     1  7.51   7.51
## 10  3094 lg           2     1  7.51   7.51
## # ... with 7,818 more rows
```

2.3. Topic modelling / Clustering (LSA)

Now we will perform a LSA, which is stable when attempting to do dimensionality reduction as preprocessing for supervised ML workflows, or for visualization.

We have loaded the quanteda package, which is for corpus-token based text analysis.

First we have to create a document-feature-matrix

```r
reviews_dfm <- reviews_tidy %>%
  count(ID, word) %>%
  cast_dfm(document = ID, term = word, value = n)

reviews_dfm
```

```
## Document-feature matrix of: 1,510 documents, 1,827 features (99.7% sparse).
```

Now we get ready for the *LSA*, by choosing the number of dimensions.

```r
reviews_lsa <- reviews_dfm %>%
  textmodel_lsa(nd = 5)
```

Let's have a look

```r
reviews_lsa %>%
  glimpse()
```

```
## List of 5
##  $ sk            : num [1:5] 22.7 16.5 15 13.5 13
##  $ docs          : num [1:1510, 1:5] 0.0119 0.0191 0.0109 0.0199 0.015 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:1510] "3" "4" "6" "7" ...
##   .. ..$ : NULL
##  $ features      : num [1:1827, 1:5] 0.012388 0.004371 0.016513 0.000999 0.211243 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:1827] "dark" "didnt" "happy" "im" ...
##   .. ..$ : NULL
##  $ matrix_low_rank: num [1:1510, 1:1827] 0.0209 0.0266 0.0103 0.0177 0.0258 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:1510] "3" "4" "6" "7" ...
##   .. ..$ : chr [1:1827] "dark" "didnt" "happy" "im" ...
##  $ data           :Formal class 'dgCMatrix' [package "Matrix"] with 6 slots
##   .. ..@ i       : int [1:7828] 0 41 48 49 50 541 1395 0 959 1025 ...
##   .. ..@ p       : int [1:1828] 0 7 10 31 33 187 191 214 216 296 ...
##   .. ..@ Dim     : int [1:2] 1510 1827
##   .. ..@ Dimnames:List of 2
##   .. ..@ x       : num [1:7828] 1 1 1 1 1 1 1 1 1 1 ...
##   .. ..@ factors : list()
##  - attr(*, "class")= chr "textmodel_lsa"
```

Now we take the *LSA documents* and put them into a tibble, and adding the id's as row names.

```
reviews_lsa_loading <- reviews_lsa$docs %>%
  as.data.frame() %>%
  rownames_to_column(var = "ID") %>%
  as_tibble()

reviews_lsa_loading %>%
  head()
```

```
## # A tibble: 6 x 6
##   ID        V1        V2     V3     V4     V5
##   <chr>  <dbl>     <dbl>  <dbl>  <dbl>  <dbl>
## 1 3     0.0119 0.00376   0.0156 0.0426 0.0151
## 2 4     0.0191 0.00234   0.0142 0.0476 0.0252
## 3 6     0.0109 0.0000639 0.0102 0.0109 0.0135
## 4 7     0.0199 0.00632   0.0110 0.0274 0.0141
## 5 8     0.0150 0.00290   0.0167 0.0482 0.0248
## 6 11    0.0409 0.00841   0.0228 0.0373 0.0538
```

We can nicely visualize it using *UMAP* dimensionality reduction for optimizing the visualization of the feature space.

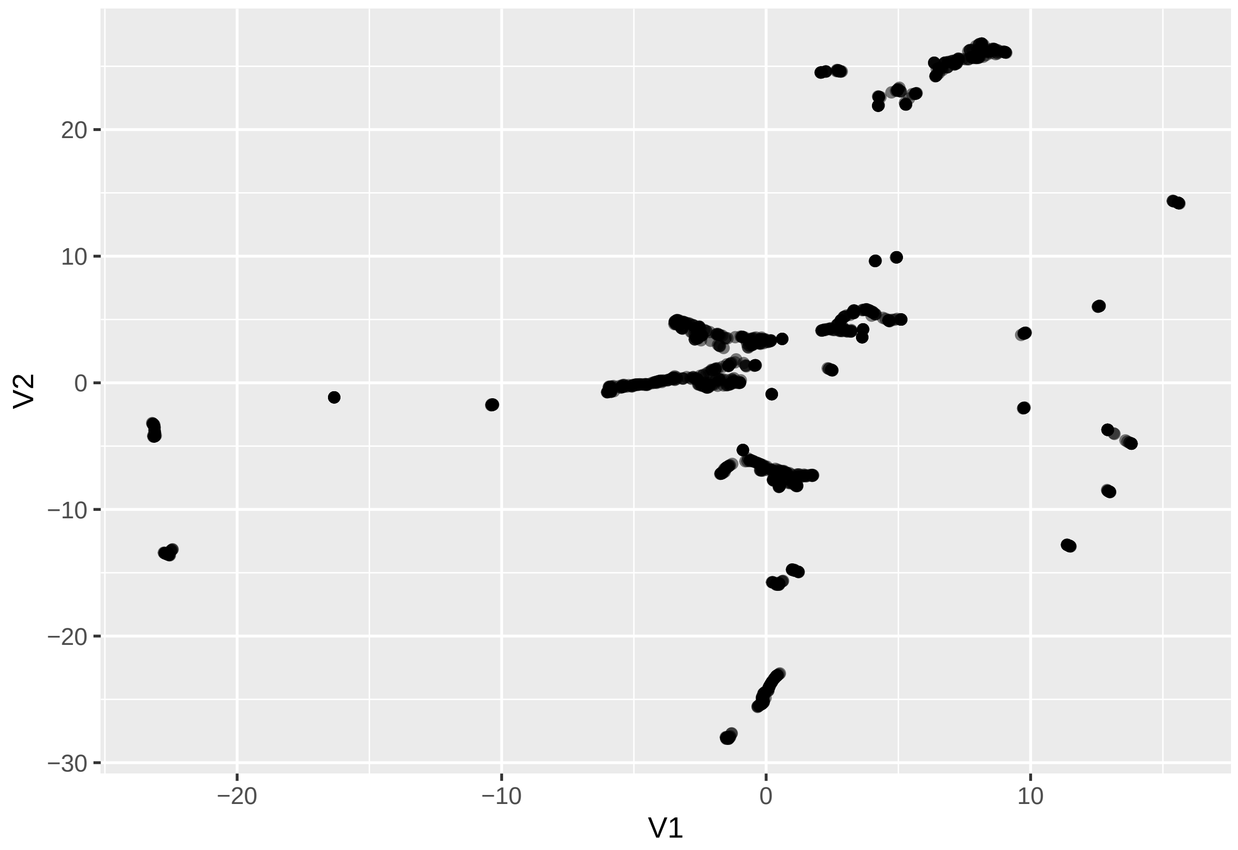
```
reviews_lsa_umap = umap(reviews_lsa_loading %>%
                          column_to_rownames("ID"),
                        n_neighbors = 15,
                        metric = "cosine",
                        min_dist = 0.01,
                        scale = TRUE,
                        verbose = TRUE,
                        n_threads = 8)
```

Then we put the result into a tibble for a nice layout.

```
reviews_lsa_umap  = reviews_lsa_umap %>%
  as.data.frame()
```

Now we can plot the result

```
reviews_lsa_umap %>%
  ggplot(aes(x = V1, y = V2)) +
  geom_point(alpha = 0.5)
```



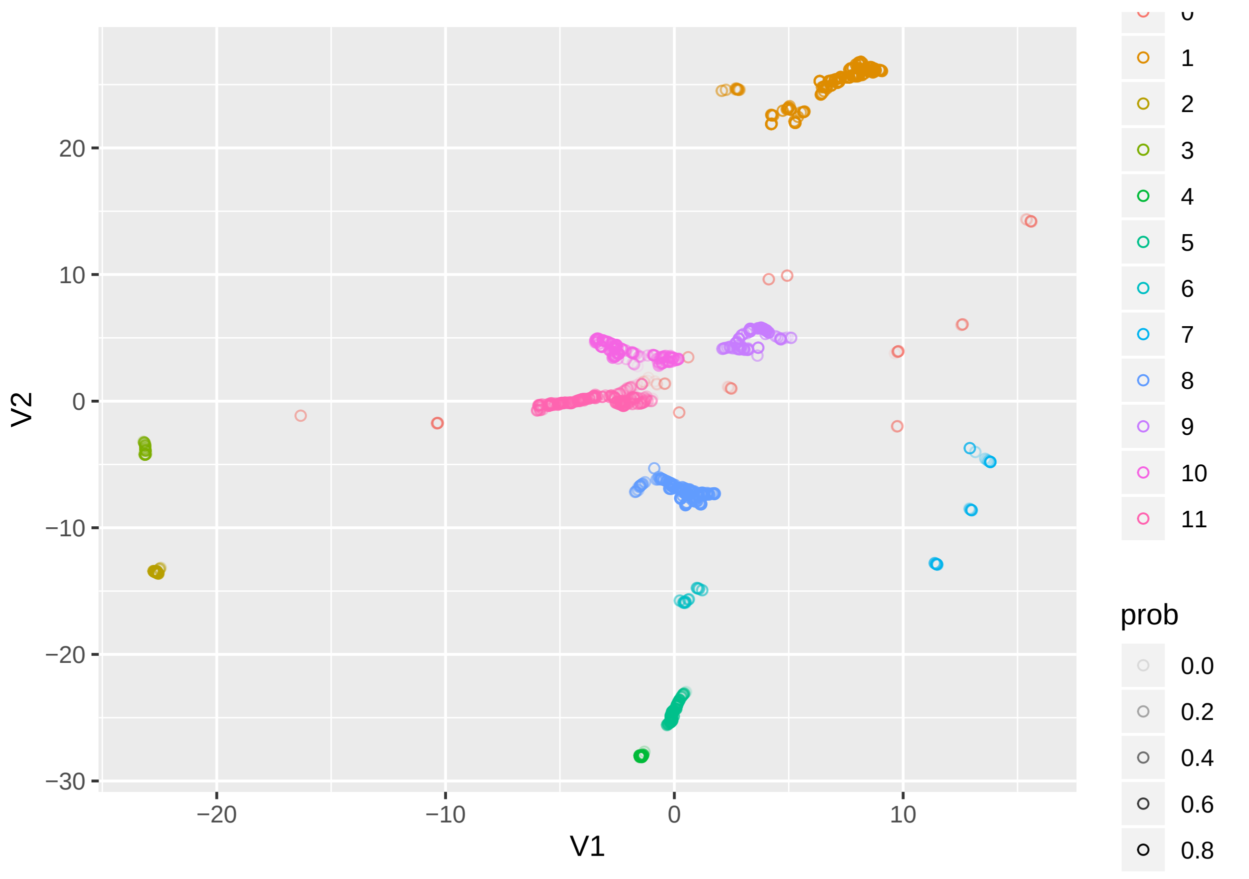
We see that the some of the points tend to cluster togehter.

We now try to find clusters by using the *hdbscan*.

```
reviews_lsa_hdbscan <- reviews_lsa_umap %>%
  as.matrix() %>%
  hdbscan(minPts = 50)
```

Let plot again, but now we color by clusters.

```
reviews_lsa_umap %>%
  bind_cols(cluster = reviews_lsa_hdbscan$cluster %>% as.factor(),
            prob = reviews_lsa_hdbscan$membership_prob) %>%
  ggplot(aes(x = V1, y = V2, col = cluster)) +
  geom_point(aes(alpha = prob), shape = 21)
```



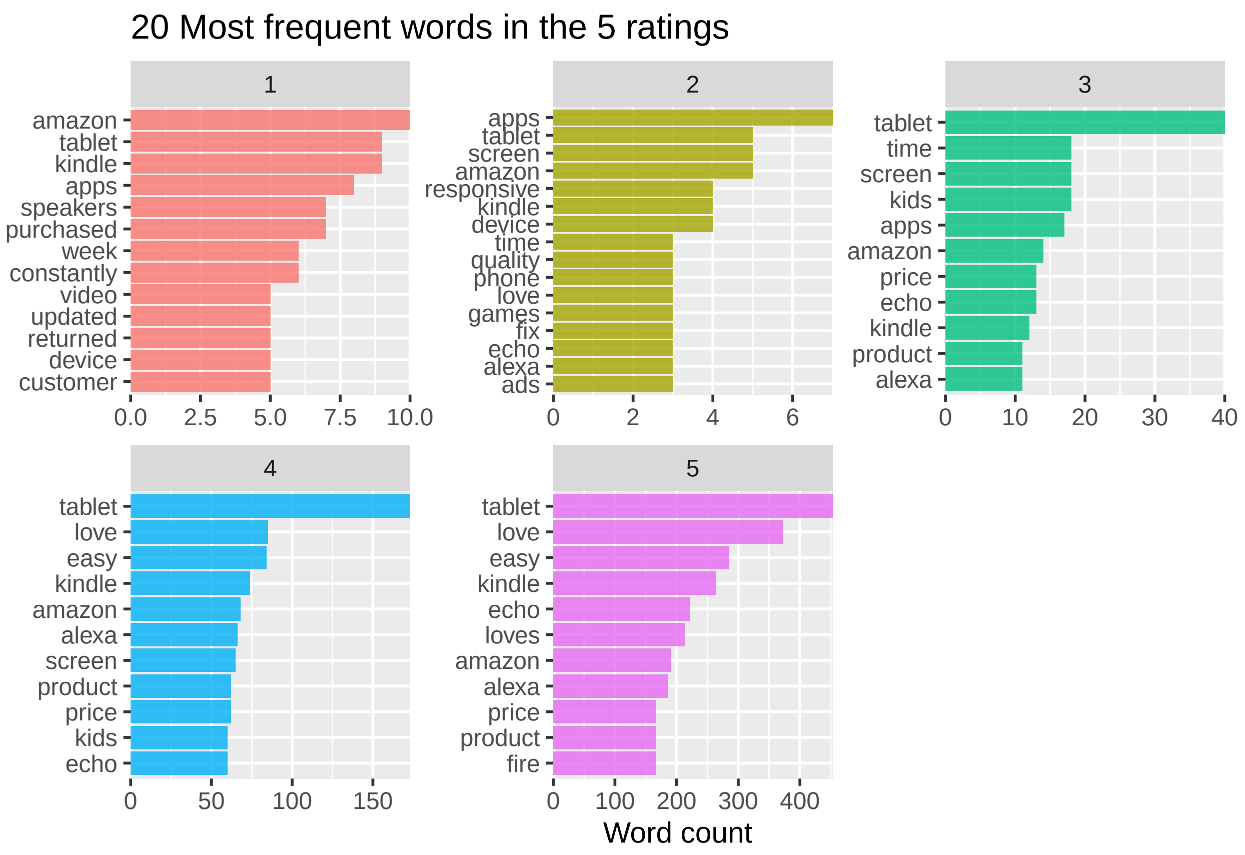
2.4. EDA / simple frequency-based analysis

Now we look closer into rating, we want to see of some words tend to show more in one rating then the others.

```
reviews_tidy3 = reviews_tidy %>%
  left_join(reviews, by = "ID")
```

Now we plot the 20 most used words in each rating.

```
reviews_tidy3 %>%
 count(rating, word, sort = TRUE) %>%
 group_by(rating) %>%
 top_n(10) %>%
 ungroup() %>%
 ggplot(aes(reorder_within(word, n, rating), n, fill = rating )) +
 geom_col(alpha = 0.8, show.legend = FALSE) +
 scale_x_reordered() +
 coord_flip() +
 facet_wrap(~rating, scales = "free") +
 scale_y_continuous(expand = c(0, 0)) +
 labs(x = NULL, y = "Word count", title = "20 Most frequent words in the 5 ratings")
```



we see that

- **Rating 1:** contains words like *constantly*, *returned*, *purchased* which could have a negative meaning and then there are a lot of words about the *electronics*.
- **Rating 2:** contains words like *responsive*, *time*, *fix* which also could have a negative meaning, and here we also see words describing the *electronics*
- **Rating 3:** contains words like the two other, but here the amount each words are used are almost the same for them all, except *tablet*. Therefore this looks loke the ones that don*t take this review seriously.
- **Rating 4:** containd words like *love*, *easy*, *price* and *product* this would usually have positive meaning in a text together with the describtion of the *electronic*.
- **Rating 5:** contain words like *love(s)*, *easy*, *price*, wich are way more positive words than the first two ratings. We also see that the amount each words are used are way higher here.

We now want to see why the amount each words are used differ that much through the ratings.

```
reviews %>%
  group_by(rating) %>%
  count()
```

```
## # A tibble: 5 x 2
## # Groups:   rating [5]
##   rating     n
##   <fct>  <int>
## 1 1         63
## 2 2         54
## 3 3        197
## 4 4       1208
## 5 5       3478
```

We see that most of the reviewers gave the product a rating at 4 or 5, this could explain the differens in the amount of the word.

2.5. Topic modelling / Clustering (LDA)

Preparing the Data

For this application, we have to leave the tidy data, since the *topicmodels* package requires a document-term matrix as imput. We can easily produce it using the cast_dtm() function of tidytext. This matrix has to be term-frequency weighted, we do so using the *weightTf* function of the tm package for the weighting argument

```
reviews_dtm = reviews_tidy %>%
  count(ID, word) %>% #add a count of words in each id
  cast_dtm(document = ID, #Column containing document IDs as string or symbol
           term = word, #Column containing terms as string or symbol
           value = n, #Column containing values as string or symbol
           weighting = tm::weightTf) #The weighting function for the DTM/TDM

reviews_dtm
```

```
## <<DocumentTermMatrix (documents: 1510, terms: 1827)>>
## Non-/sparse entries: 7828/2750942
## Sparsity           : 100%
## Maximal term length: 18
## Weighting          : term frequency (tf)
```

We see that the matrix is 100% sparse with 1510 documents and 1827 terms, which is an artefact of text data generally. Now we try to see if we could reduce that somewhat by deleting less often used terms.

```
reviews_dtm %>% removeSparseTerms(sparse = .99) #sparse is maximal allowed sparsity.
```

```
## <<DocumentTermMatrix (documents: 1510, terms: 74)>>
## Non-/sparse entries: 3597/108143
## Sparsity           : 97%
## Maximal term length: 9
## Weighting          : term frequency (tf)
```

With max allowed sparsity at 99% we only have 74 terms out of 1827, too little

```
reviews_dtm %>% removeSparseTerms(sparse = .999) #sparse is maximal allowed sparsity.
```

```
## <<DocumentTermMatrix (documents: 1510, terms: 806)>>
## Non-/sparse entries: 6807/1210253
## Sparsity           : 99%
## Maximal term length: 13
## Weighting          : term frequency (tf)
```

With max allowed sparsity at 99.9% we only have 806 terms out of 1827, we might have to accept a high level of sparsity in order to still have a meaningful number of unique words.

Now we can perform an LDA, using the more accurate Gibbs sampling as method.

```
reviews_lda <- reviews_dtm %>%
  LDA(k = 5,
      method = "Gibbs",
      control = list(seed = 9))
```

β Word-Topic Association

β is an output of the LDA model, indicating the propability that a word occurs in a certain topic. Therefore, loking at the top probability words of a topic often gives a good intuition regarding its properties.

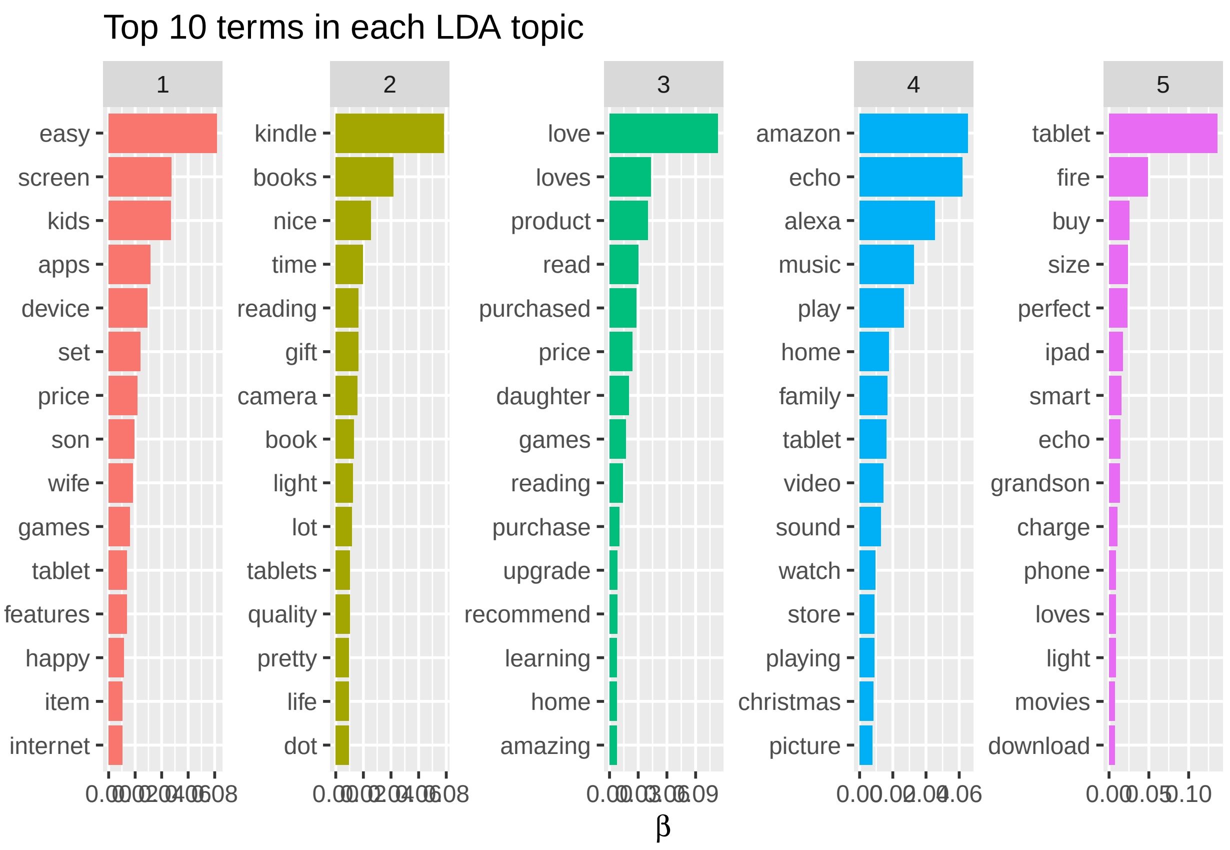
```
# LDA output is defined for tidy(), so we can easily extract it
lda_beta <- reviews_lda %>%
  tidy(matrix = "beta") %>%
  group_by(topic) %>%
  arrange(topic, desc(beta)) %>% #the higher beta, the more a word is "used" in a topic
  slice(1:15) %>%
  ungroup()

lda_beta %>% head()
```

```
## # A tibble: 6 x 3
##   topic term     beta
##   <int> <chr>   <dbl>
## 1     1 easy   0.0818
## 2     1 screen 0.0475
## 3     1 kids   0.0469
## 4     1 apps   0.0315
## 5     1 device 0.0292
## 6     1 set    0.0241
```

We see that topic one have words like *easy*, *screen* and *kids* with high β values.

Now we plot the top 10 words in each topic.

```
lda_beta %>%
  mutate(term = reorder_within(term, beta, topic)) %>% #reordering the term by beta in each topic
  group_by(topic, term) %>% #first group by topic and then by term
  arrange(desc(beta)) %>% #rearrange data in descending order by beta
  ungroup() %>%
  ggplot(aes(term, beta, fill = as.factor(topic))) + #aes for the plot
  geom_col(show.legend = FALSE) + #Don't show legend
  coord_flip() + #flip the graph, so the bars will be horizontal
  scale_x_reordered() +
  labs(title = "Top 10 terms in each LDA topic",
       x = NULL,
       y = expression(beta)) +
  facet_wrap(~ topic, nrow = 1, scales = "free")
```



- **Topic 1:** We see that topic one is about quality for *HIFI* stuff.
- **Topic 2:** We see that topic two is mainly about reading. The word *Kindle* are top 1, but words like *books*, *read(ing)*, *light*, are also in here.
- **Topic 3:** We see that topic three is re mess of different types of words.
- **Topic 4:** We see that topic four is about *HIFI*, since words like *echo*, *alexa* and *music* are highly rated here.
- **Topic 5:** We see that topic five is mainly about tablets, since the words like *tablet*, *fire* and *ipad* are in this topic. together with some other tablet related words.

Overall we can say that the topics are mainly split into categories and not rating like we did earlier.

2.6. Embedding-model based vectorization (GloVe)

First we create a corpus

```
reviews_corpus <- reviews %>%
  corpus(docid_field = "ID",
         text_field = "text")
```

Select features

Now we tokenize the corpus, we remove stopwords, punctuations, number and so on.

```
reviews_toks <- tokens(reviews_corpus, what = "word",
                       remove_numbers = TRUE, #Remove numbers
```

```
                     remove_punct = TRUE, #remove dots
                     remove_symbols = TRUE, #remove symbols
                     remove_separators = TRUE, #remove seperators
                     remove_url = TRUE, #remove url's
                     ngrams = 1) %>% #allowing unigrams
  tokens_tolower() %>% #only lower case letters
  tokens_remove(pattern = stopwords()) %>% #remove stopwords
  tokens_remove(pattern = own_stopwords$word) %>% #remove own words
  tokens_remove(pattern = c("[^[:alnum:]]")) #remove all non alpha numeric


reviews_toks %>% head(2)
```

```
## tokens from 2 documents.
## 1 :
##  [1] "thought"     "big"          "small"       "paper"       "turn"
##  [6] "just"        "like"         "palm"        "think"       "small"
## [11] "read"        "comfortable" "regular"     "kindle"      "definitely"
## [16] "recommend"   "paperwhite"   "instead"
##
## 2 :
## [1] "kindle"     "light"      "easy"       "use"        "especially"
## [6] "beach"
```

and then we keep the names of the features that occur five times or more.

```
feats <- dfm(reviews_toks, verbose = TRUE) %>%
    dfm_trim(min_termfreq = 5) %>%
    featnames()
```

Constructing the feature co-occurence matrix

```
reviews_fcm <- fcm(reviews_toks,
                 context = "window",
                 count = "weighted",
                 weights = 1 / (1:5),
                 tri = TRUE)
```

Fit word embedding model

Now we fit the *GloVe* model using *text2vec*

GloVe is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space.

```
glove <- GlobalVectors$new(word_vectors_size = 50, #number of words in each vector
                           vocabulary = featnames(reviews_fcm),
                           x_max = 10)

reviews_main <- fit_transform(reviews_fcm,
                            glove,
                            n_iter = 20) #number of iterations
```

```
## INFO [2019-10-23 16:07:54] 2019-10-23 16:07:54 - epoch 1, expected cost 0.0806
## INFO [2019-10-23 16:07:54] 2019-10-23 16:07:54 - epoch 2, expected cost 0.0479
## INFO [2019-10-23 16:07:54] 2019-10-23 16:07:54 - epoch 3, expected cost 0.0389
## INFO [2019-10-23 16:07:54] 2019-10-23 16:07:54 - epoch 4, expected cost 0.0336
## INFO [2019-10-23 16:07:54] 2019-10-23 16:07:54 - epoch 5, expected cost 0.0300
## INFO [2019-10-23 16:07:54] 2019-10-23 16:07:54 - epoch 6, expected cost 0.0273
## INFO [2019-10-23 16:07:54] 2019-10-23 16:07:54 - epoch 7, expected cost 0.0251
## INFO [2019-10-23 16:07:54] 2019-10-23 16:07:54 - epoch 8, expected cost 0.0234
## INFO [2019-10-23 16:07:54] 2019-10-23 16:07:54 - epoch 9, expected cost 0.0220
## INFO [2019-10-23 16:07:55] 2019-10-23 16:07:55 - epoch 10, expected cost 0.0207
## INFO [2019-10-23 16:07:55] 2019-10-23 16:07:55 - epoch 11, expected cost 0.0197
## INFO [2019-10-23 16:07:55] 2019-10-23 16:07:55 - epoch 12, expected cost 0.0188
## INFO [2019-10-23 16:07:55] 2019-10-23 16:07:55 - epoch 13, expected cost 0.0180
## INFO [2019-10-23 16:07:55] 2019-10-23 16:07:55 - epoch 14, expected cost 0.0173
## INFO [2019-10-23 16:07:55] 2019-10-23 16:07:55 - epoch 15, expected cost 0.0167
## INFO [2019-10-23 16:07:55] 2019-10-23 16:07:55 - epoch 16, expected cost 0.0161
## INFO [2019-10-23 16:07:55] 2019-10-23 16:07:55 - epoch 17, expected cost 0.0156
## INFO [2019-10-23 16:07:55] 2019-10-23 16:07:55 - epoch 18, expected cost 0.0152
## INFO [2019-10-23 16:07:55] 2019-10-23 16:07:55 - epoch 19, expected cost 0.0148
## INFO [2019-10-23 16:07:55] 2019-10-23 16:07:55 - epoch 20, expected cost 0.0144
```

Averagning learned word vectors

The two vectors are main and context. According to the *Glove* paper, averaging the two word vectors results in more accurate representation.

```r
reviews_context <- glove$components
dim(reviews_context)
```

```
## [1]   50 5613
```

We see that the dimension of this model is 50 x 5614.

```r
reviews_vectors <- as.dfm(reviews_main + t(reviews_context))
```

Examining term representations

Now we can find the six closest word vectors for *kindle*.

```r
new <- reviews_vectors["kindle", ]

# calculate the similarity
cos_sim <- textstat_simil(reviews_vectors, new,
                          margin = "documents", method = "cosine")

head(sort(cos_sim[, 1], decreasing = TRUE), 6)
```

```
##    kindle      fire    second       new      much     first
## 1.0000000 0.8198814 0.6760594 0.6581518 0.6310104 0.6266958
```

We see the words *fire*, *new*, *since*, *one* and *second* are the closest words to the word *kindle*

The six closest word vectors for *tablet*.

```r
new <- reviews_vectors["tablet", ]

# calculate the similarity
cos_sim <- textstat_simil(reviews_vectors, new,
                            margin = "documents", method = "cosine")

head(sort(cos_sim[, 1], decreasing = TRUE), 6)
```

```
##    tablet     great     price      good      fire   perfect
## 1.0000000 0.7479137 0.7307526 0.7285915 0.6670248 0.6637030
```

We see that the words like *good*, *great*, *price* and *kids* are the closest to the word *tablet*.

As seen earlier the data caontains mostly of high rated reviews. Therefore it makes sence that the word *tablet* follows by positive words like *great* and *good*.

Now we make an average-vector-representations for the words

```
reviews_main_tibble = reviews_main %>%
  as.data.frame() %>% #making a dataframe
  rownames_to_column(var = "word") %>% #Row names
  as_tibble() #for nice layout

reviews_main_tibble %>% head()
```

```
## # A tibble: 6 x 51
##   word      V1      V2      V3      V4      V5      V6      V7     V8
##   <chr>  <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>  <dbl>
## 1 thou~ -0.281 -0.0368  0.0152  0.133  -0.0720 -0.123  -0.531  -0.169
## 2 big    0.144 -0.261   0.207   0.521  -0.308  -0.279  -0.0329 -0.219
## 3 small -0.302  0.262   0.274   0.703   0.0587  0.179  -0.529  -0.422
## 4 paper  0.501  0.216  -0.245   0.182   0.0149 -0.456  -0.517   0.440
## 5 turn  -0.535 -0.0839  0.836  -0.0332  0.236   0.379  -0.286  -0.163
## 6 just  -0.473  0.0134 -0.276   0.249   0.144   0.0946 -0.519  -0.148
## # ... with 42 more variables: V9 <dbl>, V10 <dbl>, V11 <dbl>, V12 <dbl>,
## #   V13 <dbl>, V14 <dbl>, V15 <dbl>, V16 <dbl>, V17 <dbl>, V18 <dbl>,
## #   V19 <dbl>, V20 <dbl>, V21 <dbl>, V22 <dbl>, V23 <dbl>, V24 <dbl>,
## #   V25 <dbl>, V26 <dbl>, V27 <dbl>, V28 <dbl>, V29 <dbl>, V30 <dbl>,
## #   V31 <dbl>, V32 <dbl>, V33 <dbl>, V34 <dbl>, V35 <dbl>, V36 <dbl>,
## #   V37 <dbl>, V38 <dbl>, V39 <dbl>, V40 <dbl>, V41 <dbl>, V42 <dbl>,
## #   V43 <dbl>, V44 <dbl>, V45 <dbl>, V46 <dbl>, V47 <dbl>, V48 <dbl>,
## #   V49 <dbl>, V50 <dbl>
```

Now we make a average-vector-representations for the reviews.

```
reviews_tidy2 = reviews_toks %>%
  dfm() %>%
  tidy()

reviews_vector2 = reviews_tidy2 %>%
  inner_join(reviews_main_tibble, by = c("term" = "word")) #joining the data by word/term

head(reviews_vector2)
```

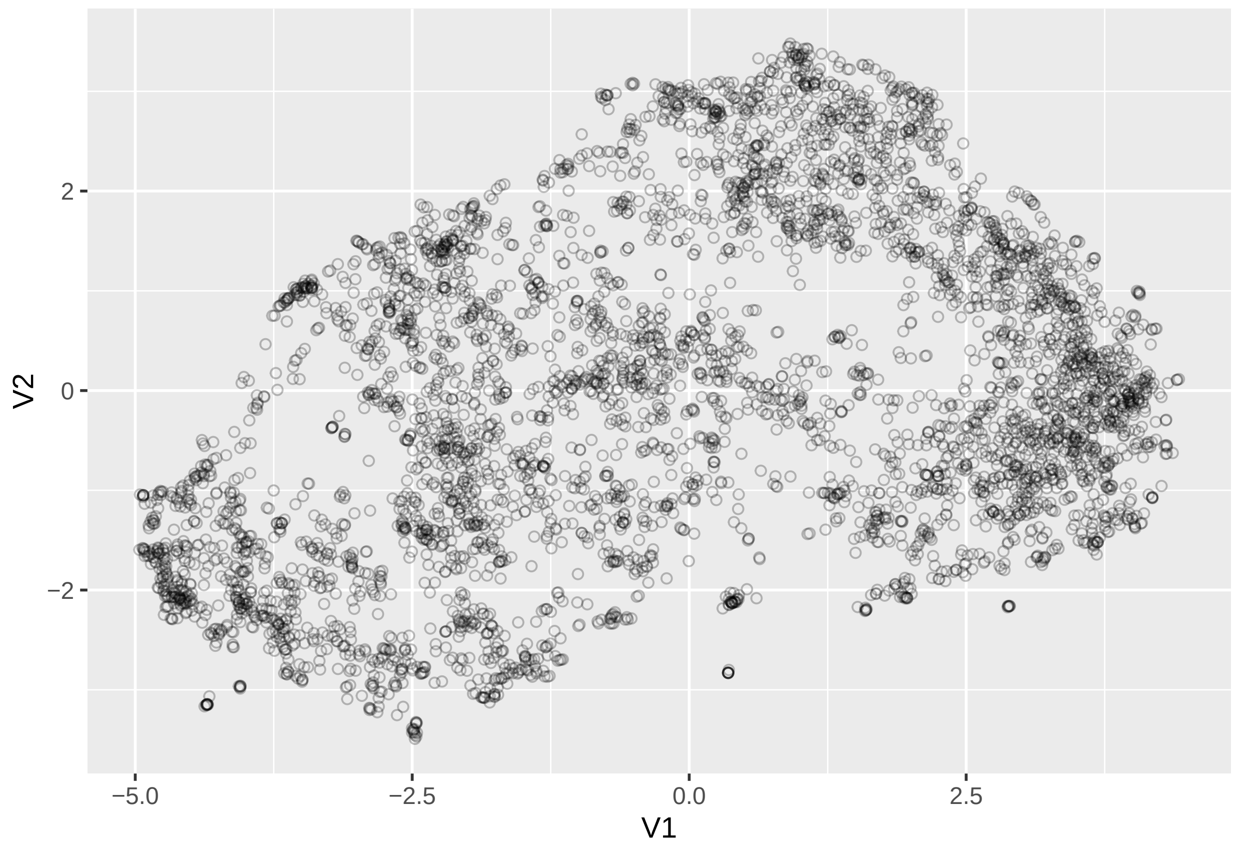
```
## # A tibble: 6 x 53
##   document term  count     V1      V2     V3    V4      V5     V6     V7
##   <chr>    <chr> <dbl>  <dbl>   <dbl>  <dbl> <dbl>   <dbl>  <dbl>  <dbl>
## 1 1        thou~     1 -0.281 -0.0368 0.0152 0.133 -0.0720 -0.123 -0.531
## 2 65       thou~     1 -0.281 -0.0368 0.0152 0.133 -0.0720 -0.123 -0.531
## 3 70       thou~     1 -0.281 -0.0368 0.0152 0.133 -0.0720 -0.123 -0.531
## 4 156      thou~     1 -0.281 -0.0368 0.0152 0.133 -0.0720 -0.123 -0.531
## 5 158      thou~     1 -0.281 -0.0368 0.0152 0.133 -0.0720 -0.123 -0.531
## 6 182      thou~     1 -0.281 -0.0368 0.0152 0.133 -0.0720 -0.123 -0.531
## # ... with 43 more variables: V8 <dbl>, V9 <dbl>, V10 <dbl>, V11 <dbl>,
## #   V12 <dbl>, V13 <dbl>, V14 <dbl>, V15 <dbl>, V16 <dbl>, V17 <dbl>,
## #   V18 <dbl>, V19 <dbl>, V20 <dbl>, V21 <dbl>, V22 <dbl>, V23 <dbl>,
## #   V24 <dbl>, V25 <dbl>, V26 <dbl>, V27 <dbl>, V28 <dbl>, V29 <dbl>,
## #   V30 <dbl>, V31 <dbl>, V32 <dbl>, V33 <dbl>, V34 <dbl>, V35 <dbl>,
## #   V36 <dbl>, V37 <dbl>, V38 <dbl>, V39 <dbl>, V40 <dbl>, V41 <dbl>,
## #   V42 <dbl>, V43 <dbl>, V44 <dbl>, V45 <dbl>, V46 <dbl>, V47 <dbl>,
## #   V48 <dbl>, V49 <dbl>, V50 <dbl>
```

```
reviews_vector2 = reviews_vector2 %>%
  select(-term, -count) %>% #remove term and count
  group_by(document) %>% #group the data by document
  summarise_all(mean) %>% #finding the mean of the coloumns for each document
  ungroup()
```

Let’s have a look at the data

```
reviews_vector2_umap <- umap(reviews_vector2 %>% column_to_rownames("document"),
                       n_neighbors = 15,
                       metric = "cosine",
                       min_dist = 0.01,
                       scale = TRUE,
                       verbose = TRUE) %>%
  as.data.frame()
```

```
reviews_vector2_umap %>%
  ggplot(aes(x = V1, y = V2)) +
  geom_point(shape = 21, alpha = 0.25)
```



It does not look like the data have any clusters right now. If we look closely we could argue about some few clusters at the plot, which might be *tablet* and/or *kindle* stuff.

3. Supervised / Unsupervised ML

3.1. Unsupervised ML

We do *kmeans* for the *tf_idf* values, and want 5 clusters since we have 5 different ratings in the data.

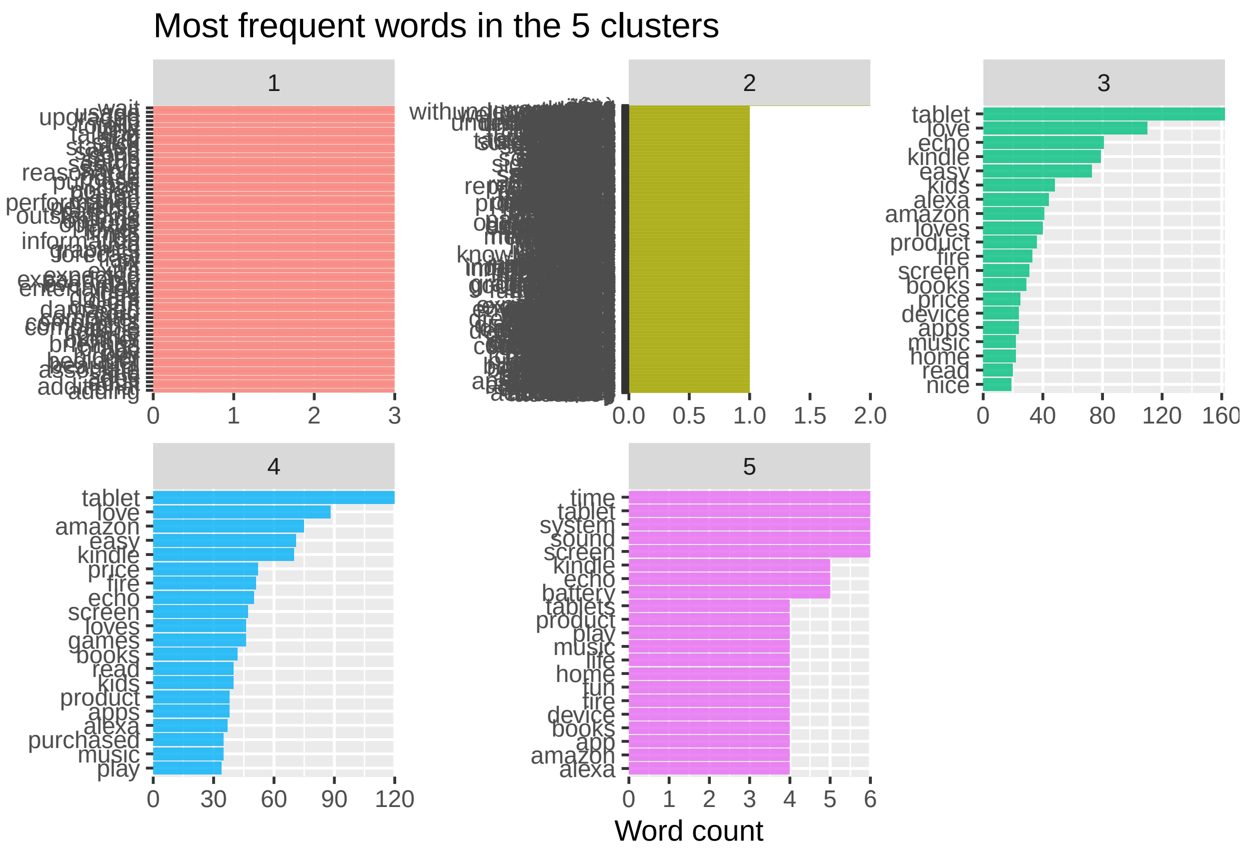
```
kmeans_tfidf = kmeans(reviews_tidy[,4:6], centers = 5, nstart = 20)
```

```
reviews_tidy = reviews_tidy %>%
  bind_cols(cluster = kmeans_tfidf$cluster)
```

Now we plot the reviews_tidy, and color by clusters.

```
reviews_tidy %>%
 mutate(cluster = as.factor(cluster)) %>%
 count(cluster, word, sort = TRUE) %>%
 group_by(cluster) %>%
 top_n(20,n) %>%
 ungroup() %>%
 ggplot(aes(reorder_within(word, n, cluster), n,
   fill = cluster
 )) +
 geom_col(alpha = 0.8, show.legend = FALSE) +
 scale_x_reordered() +
```

```r
  coord_flip() +
  facet_wrap(~cluster, scales = "free")  +
  scale_y_continuous(expand = c(0, 0)) +
  labs(x = NULL, y = "Word count",
    title = "Most frequent words in the 5 clusters")
```



Unfortunately the plots are messy. since some of the clusters contain words represented the same amount of times. Therefore we can only talk/discuss plot 1, 2 and 5.

The plots contain almost the same words, it is hard to find differences in these three plots.

3.2. Supervised ML

3.2.1. Method 1

Now we want to get ready for modeling. We split the data into training and testing sets, using *rsample*. We use the original data, reviews, because the reviews$text are our individual observations.

```r
reviews_split <- reviews %>%
  select(ID2) %>%
  initial_split() #by default it splits 3/4

train_data <- training(reviews_split) #train data

test_data <- testing(reviews_split) #test data
```

Now we transform our training data from a tidy structure to a sparse matrix, wich we will use for the machine learning algorithm.

```
sparse_words =  reviews_tidy_ML %>%
  count(ID2, word) %>%
  inner_join(train_data) %>% #keep rows from Tuwets_tidy where there are matching values in y, and all
  cast_sparse(ID2, word, n) #Create a sparse matrix from row names, column names, and values in a table
```

```
dim(sparse_words)
```

```
## [1] 1881 1754
```

We see we get a sparse matrix with 1885 rows and 1764 columns. This means we have 1885 training observations and 1764 features at this point.

Now we build a dataframe with response variable to associate each of the *rownames()* of the sparse matrix. We select the *rating*, since it will be our predictor later.

```
word_rownames = as.integer(rownames(sparse_words))
reviews_rating = reviews %>% select(ID2, rating)
```

```
reviews_joined = data_frame(ID2 = word_rownames) %>%
  left_join(reviews_rating, by = "ID2" )
```

Now we want to train our classification model. We use *glmnet* to fit a logistic regression model with *LASSO* regularization. It's a great fit for text classification because the variable selection that *LASSO* regularization performs can tell you which words are important for our prediction problem.

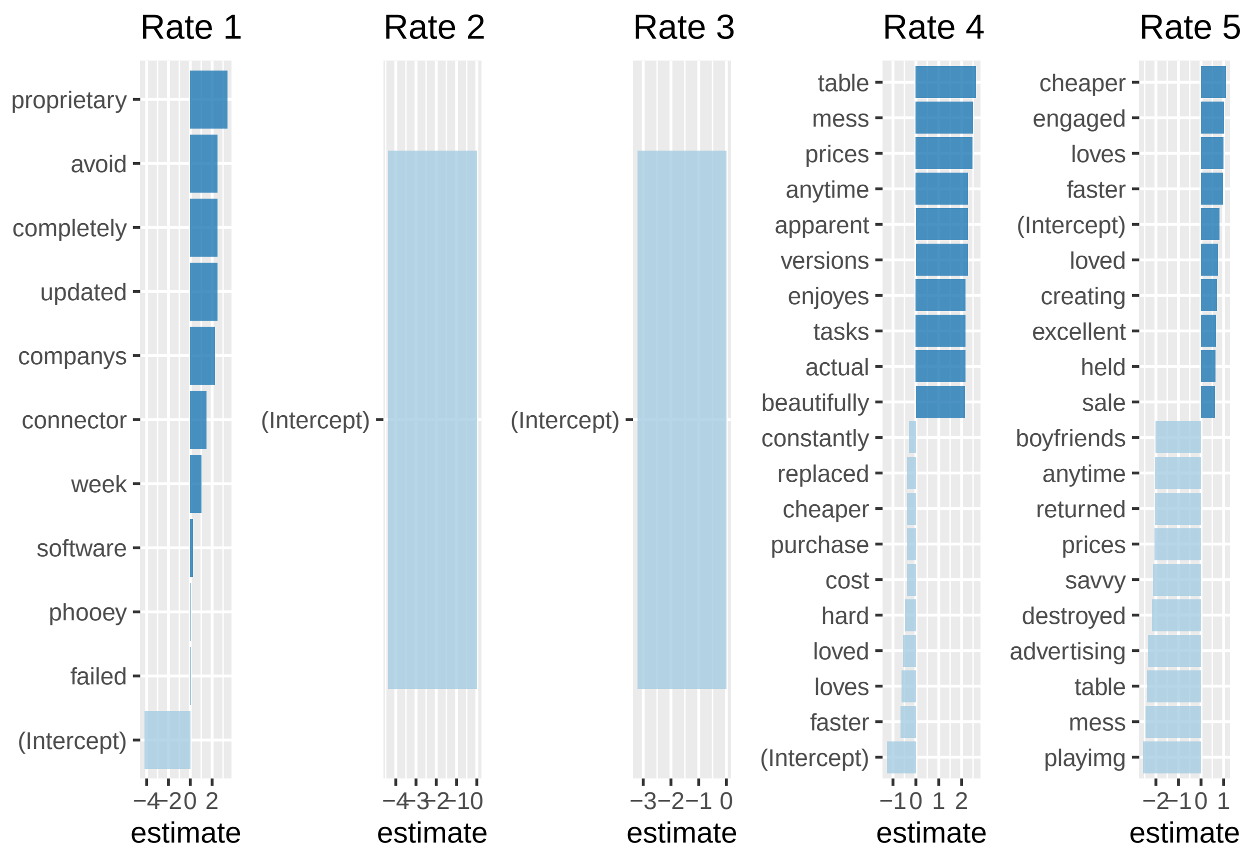
We make five models, since we have five types of ratings.

```
#Model for rate 1
model_1 = cv.glmnet(sparse_words, # x variable = matrix .
                    reviews_joined$rating == 1, #response variable
                    family = "binomial",
                    keep = TRUE)

#Model for rate 2
model_2 <- cv.glmnet(sparse_words, #x variable = matrix .
                    reviews_joined$rating == 2,  #response variable
                    family = "binomial",
                    keep = TRUE)

#Model for rate 3
model_3 <- cv.glmnet(sparse_words, #x variable = matrix .
                    reviews_joined$rating == 3, #response variable
                    family = "binomial",
                    keep = TRUE)

#Model for rate 4
model_4 <- cv.glmnet(sparse_words, #x variable = matrix .
                    reviews_joined$rating == 4, #response variable
                    family = "binomial",
                    keep = TRUE)

#Model for rate 5
model_5 <- cv.glmnet(sparse_words, #x variable = matrix .
                    reviews_joined$rating == 5, #response variable
                    family = "binomial",
                     keep = TRUE)
```

Now we use *broom* to check the five models coeffecients. We also check for wich coefficients are largest in size, in each direction

```
coef_1 <- model_1$glmnet.fit %>%
  tidy() %>%
  filter(lambda == model_1$lambda.1se) %>% #lambda with error within 1 standard error
  group_by(estimate > 0) %>%
  top_n(10, abs(estimate)) %>%
  ungroup() %>%
  ggplot(aes(fct_reorder(term, estimate),  #plotting the top words in each direction
             estimate,
             fill = estimate > 0)) +
  geom_col(alpha = 0.8, show.legend = FALSE) +
  coord_flip() +
  scale_fill_brewer(palette = "Paired") +
  labs( title = "Rate 1",
       x = NULL)


coef_2 <- model_2$glmnet.fit %>%
  tidy() %>%
  filter(lambda == model_2$lambda.1se) %>% #lambda with error within 1 standard error
  group_by(estimate > 0) %>%
  top_n(10, abs(estimate)) %>%
  ungroup() %>%
  ggplot( aes( fct_reorder( term, estimate),
               estimate,
               fill = estimate > 0)) +
  geom_col(alpha = 0.8, show.legend = FALSE) +
  coord_flip() +
  scale_fill_brewer(palette = "Paired") +
  labs( title = "Rate 2",
       x = NULL)


coef_3 <- model_3$glmnet.fit %>%
  tidy() %>%
  filter(lambda == model_3$lambda.1se) %>% #lambda with error within 1 standard error
  group_by(estimate > 0) %>%
  top_n(10, abs(estimate)) %>%
  ungroup() %>%
  ggplot(aes(fct_reorder(term, estimate),
             estimate,
             fill = estimate > 0)) +
  geom_col(alpha = 0.8, show.legend = FALSE) +
  coord_flip() + #Horizontal barplot
  scale_fill_brewer(palette = "Paired") +
  labs(title = "Rate 3",
       x = NULL)


coef_4 <- model_4$glmnet.fit %>%
```

```r
  tidy() %>%
  filter(lambda == model_4$lambda.1se) %>% #lambda with error within 1 standard error
  group_by(estimate > 0) %>%
  top_n(10, abs(estimate)) %>%
  ungroup() %>%
  ggplot(aes(fct_reorder(term, estimate),
             estimate,
             fill = estimate > 0)) +
  geom_col(alpha = 0.8, show.legend = FALSE) +
  coord_flip() + #Horizontal barplot
  scale_fill_brewer(palette = "Paired") +
  labs(title = "Rate 4",
       x = NULL)


coef_5 <- model_5$glmnet.fit %>%
  tidy() %>%
  filter(lambda == model_5$lambda.1se) %>% #lambda with error within 1 standard error
  group_by(estimate > 0) %>%
  top_n(10, abs(estimate)) %>%
  ungroup() %>%
  ggplot(aes(fct_reorder(term, estimate),
             estimate,
             fill = estimate > 0)) +
  geom_col(alpha = 0.8, show.legend = FALSE) +
  coord_flip() + #Horizontal barplot
  scale_fill_brewer(palette = "Paired") +
  labs(title = "Rate 5",
       x = NULL)

grid.arrange(coef_1, coef_2, coef_3, coef_4, coef_5, nrow = 1)
```



We see that not all the plots are very good, but we can see the meaning of some words for some topics.

Now we try to classify the rating for the test set.

```
model <- cv.glmnet(sparse_words,
                   reviews_joined$rating,
                   family = "multinomial",
                   keep = TRUE)

coefs <- model$glmnet.fit %>%
  tidy() %>%
  filter(lambda == model$lambda.1se)

intercept <- coefs %>%
  filter(term == "(Intercept)") %>%
  pull(estimate)

classifications <- reviews_tidy_ML %>%
  inner_join(test_data) %>% #Predicting the test data
  inner_join(coefs, by = c("word" = "term")) %>%
  group_by(class, ID2) %>%
  summarize(score = sum(estimate)) %>%
  mutate(probability = plogis(0 + score))
```

Let's have a look.

```
classifications
```

```
## # A tibble: 604 x 4
## # Groups:   class [5]
##    class   ID2 score probability
##    <chr> <int> <dbl>       <dbl>
##  1 1       227 1.32        0.789
##  2 1       305 2.54        0.927
##  3 1       516 1.33        0.791
##  4 1       545 0.667       0.661
##  5 1       664 4.40        0.988
##  6 1       665 1.27        0.781
##  7 1       878 1.33        0.791
##  8 1       879 0.667       0.661
##  9 1      1782 0.667       0.661
## 10 1      1839 3.47        0.970
## # ... with 594 more rows
```

We need to add the original rating, before we can say anything.

```
res <- classifications %>%
  group_by(ID2) %>%
  filter(probability == max(probability)) %>%
  left_join(reviews, by = "ID2")
```

Now we put the result in a table, where we can see if the classification did good or not.

```
result <- table(res$rating, res$class)

result
```

```
##
##       1   2   3   4   5
##   1   5   0   0   0   1
##   2   0   1   1   0   2
##   3   0   0  11   2   7
##   4   2   0   1  53  53
##   5   4   5  11  58 236
```

We see that the model is not that good, since it predicts $\sim 50\%$ correct.

3.2.2. Method 2

Here we try do like we saw in M1.

We select our variables to the classification models.

```
reviews_tidy_ML <- reviews_tidy3 %>%
  select(tf, idf,
         tf_idf,
         helpful,
         rating,
         primaryCategories,
         year) %>%
  mutate(good_review = as.logical(as.integer(rating) >= 4, rating)) %>%
  select(-rating) # I set class == 0, so class 0 is TRUE and the rest FALSE:
```

Splitting the data into train- and test set.

```r
index <- createDataPartition(y = reviews_tidy_ML$good_review,
                             p = 0.75,
                             list = FALSE) # 75% to 25% split

training <- reviews_tidy_ML[index,]
test <- reviews_tidy_ML[-index,]
```

Preprocessing using the recipes package.

```r
reci <- recipe(good_review ~ ., data = training) %>%
  step_dummy(all_nominal(), -all_outcomes()) %>% # create dummy variables for the nominal variables
  step_center(all_numeric(), -all_outcomes()) %>% # Centers all numeric variables to mean = 0
  step_scale(all_numeric(), -all_outcomes()) %>% # this scales the numeric variables
  step_zv(all_predictors())  # Removed predictors with zero variance

reci %<>%
  prep(data = train)
```

Setting the x and y values.

```r
# Predictors
x_train <- bake(reci, new_data = training) %>% select(-good_review) # I remove class from the predictor
y_train <- training %>% pull(good_review) %>% as.factor()
# test: split in y and x
x_test <- bake(reci, new_data = test) %>% select(-good_review)
y_test <- test %>% pull(good_review) %>% as.factor()
```

Setting the workflow.

```r
ctrl <- trainControl(method = "cv",
                     number = 4)
metric <- "Kappa" # I use Kappa because the class 0 (hate speech) is so low represented in the data.
```

Fitting the model - logistic regression.

```r
review_fit_log <- train(x = x_train,
                 y = y_train,
                 trControl = ctrl,
                 metric = metric,
                 method = "glm",
                 family = "binomial")
```

```r
review_fit_log
```

```
## Generalized Linear Model
##
## 17694 samples
##    10 predictor
##     2 classes: 'FALSE', 'TRUE'
##
## No pre-processing
## Resampling: Cross-Validated (4 fold)
## Summary of sample sizes: 13270, 13271, 13270, 13271
## Resampling results:
##
##   Accuracy   Kappa
##   0.9206511  0
```

Prediction and evaluating.

```r
pred_log <- predict(review_fit_log, newdata = x_test)
```

```r
print("Confusion matrix for logistic model")
```

```
## [1] "Confusion matrix for logistic model"
```

```r
confusionMatrix(pred_log, y_test, positive = 'TRUE')
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction FALSE TRUE
##      FALSE     0    0
##      TRUE    468 5429
##
##                Accuracy : 0.9206
##                  95% CI : (0.9134, 0.9274)
##     No Information Rate : 0.9206
##     P-Value [Acc > NIR] : 0.5123
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 1.0000
##             Specificity : 0.0000
##          Pos Pred Value : 0.9206
##          Neg Pred Value :    NaN
##              Prevalence : 0.9206
##          Detection Rate : 0.9206
##    Detection Prevalence : 1.0000
##       Balanced Accuracy : 0.5000
##
##        'Positive' Class : TRUE
##
```

Fitting the model - decision tree

```r
reviews_fit_dt <- train(x = x_train,
                y = y_train,
                trControl = ctrl,
                metric = metric,
                method = "rpart")
```

```r
reviews_fit_dt
```

```
## CART
```

```
##
## 17694 samples
##    10 predictor
##     2 classes: 'FALSE', 'TRUE'
##
## No pre-processing
## Resampling: Cross-Validated (4 fold)
## Summary of sample sizes: 13270, 13271, 13271, 13270
## Resampling results across tuning parameters:
##
##   cp          Accuracy   Kappa
##   0.01495726  0.9434272  0.4251322
##   0.02207977  0.9299197  0.1796601
##   0.02250712  0.9299197  0.1796601
##
## Kappa was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.01495726.
```

Predicting and evaluating

```r
pred_dt <- predict(reviews_fit_dt, newdata = x_test)
```

```r
print("Confusion matrix for decision tree")
```

```
## [1] "Confusion matrix for decision tree"
```

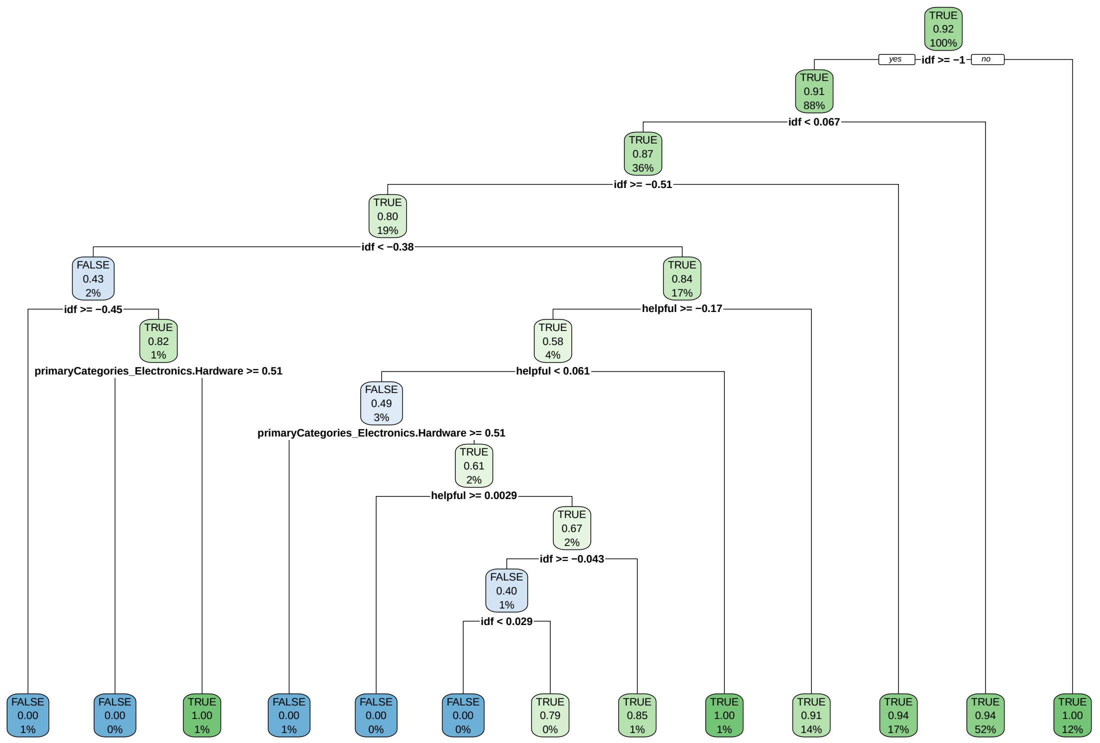
```r
confusionMatrix(pred_dt, y_test, positive = 'TRUE')
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction FALSE TRUE
##      FALSE   134    0
##      TRUE    334 5429
##
##                Accuracy : 0.9434
##                  95% CI : (0.9372, 0.9491)
##     No Information Rate : 0.9206
##     P-Value [Acc > NIR] : 7.679e-12
##
##                   Kappa : 0.4249
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 1.0000
##             Specificity : 0.2863
##          Pos Pred Value : 0.9420
##          Neg Pred Value : 1.0000
##              Prevalence : 0.9206
##          Detection Rate : 0.9206
##    Detection Prevalence : 0.9773
##       Balanced Accuracy : 0.6432
##
##        'Positive' Class : TRUE
##
```

We had some problems with the margins therefore we changed it.

```r
par(mar = c(1,1,1,1)) #The margins for the next plot are too large
```

```r
reviews_fit_dt$finalModel %>%
  rpart.plot()
```



Fitting the model - Random forest

```r
reviews_fit_rf <- train(x = x_train,
                y = y_train,
                trControl = ctrl,
                metric = metric,
                method = "ranger",
                importance = "impurity", # To define how to measure variable importantce (later more)
                num.trees = 25
                )
```

```r
reviews_fit_rf
```

```
## Random Forest
##
## 17694 samples
##    10 predictor
##     2 classes: 'FALSE', 'TRUE'
##
## No pre-processing
## Resampling: Cross-Validated (4 fold)
## Summary of sample sizes: 13271, 13270, 13270, 13271
## Resampling results across tuning parameters:
##
```

```
##   mtry  splitrule   Accuracy   Kappa
##    2    gini        0.9255679  0.1072797
##    2    extratrees  0.9206511  0.0000000
##    6    gini        0.9540520  0.6316107
##    6    extratrees  0.9511698  0.5874154
##   10    gini        0.9499264  0.6216231
##   10    extratrees  0.9502091  0.6222944
##
## Tuning parameter 'min.node.size' was held constant at a value of 1
## Kappa was used to select the optimal model using the largest value.
## The final values used for the model were mtry = 6, splitrule = gini
##  and min.node.size = 1.
```

Predicting and evaluating

```
pred_rf <- predict(reviews_fit_rf, newdata = x_test)
```

```
print("Confusion matrix for Random forest")
```

```
## [1] "Confusion matrix for Random forest"
```

```
confusionMatrix(pred_rf, y_test, positive = 'TRUE')
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction FALSE TRUE
##      FALSE   272   56
##      TRUE    196 5373
##
##                Accuracy : 0.9573
##                  95% CI : (0.9518, 0.9623)
##     No Information Rate : 0.9206
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6613
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9897
##             Specificity : 0.5812
##          Pos Pred Value : 0.9648
##          Neg Pred Value : 0.8293
##              Prevalence : 0.9206
##          Detection Rate : 0.9111
##    Detection Prevalence : 0.9444
##       Balanced Accuracy : 0.7854
##
##        'Positive' Class : TRUE
##
```

4. Network Analysis

In this section we are going to construct a network based on the reviews data.

We did several constructions and tried interpret different attributes to the network, but for some reason we could not include any attributes. For example we wanted to try include *rating* and *primaryCategories* and use them for coloring or shaping the nodes

Creating smaller sample of reviews.

We first tried to create a network inclusing the whole dataset, but because of the size, we where not be able to run the network/graph *(the system crashed).*

Instead we resampled the review dataset to only 500 observations - so we instead got a datasat consiting of 500 reviews instead of 5.000.

```
reviews_sampled <- reviews[sample(nrow(reviews),500),]
```

Now we create a tidy data as we did earlier, just on the resampled set.

```
reviews_sampled_tidy <- reviews_sampled %>%
  select(ID, text) %>%
  unnest_tokens(output = word, input = text) %>%
  anti_join(stop_words %>% bind_rows(own_stopwords), by = "word") %>%
  mutate(word = trimws(gsub("[^\\s]*[0-9][^\\s]*", "", word, perl = T))) %>%
  filter(str_length(word) > 1) %>%# this filter out the words that are blank.
  mutate(word = word %>% str_remove_all("[^[:alnum:]]") ) %>% # alnum = Alphanumeric characters.
  filter(str_length(word) > 1) # filter out words with 1 character.
```

To make a network not with too many words, we only use the 50 highest counted words in the tidy set.

```
sampled_50 <- reviews_sampled_tidy %>%
  count(word, sort = TRUE) %>% # we count the words
head(50) # we take the head of the 50 most frequent words.
```

Now we want the samlped tidy to only consist of the 50 most frequent words and here we use “left_join” to this.

First we get a big dataset, but this is because we have all the words with “NA” values, which is the words that where not a part of the 50 most frequent words. Therefore we use *drop_na()* to drop the rows with NA values.

```
reviews_sampled_tidy_joined <- reviews_sampled_tidy %>%
  left_join(sampled_50, by = "word") %>%
  drop_na()
```

Let’s have a look.

```
glimpse(reviews_sampled_tidy_joined)
```

```
## Observations: 1,800
## Variables: 3
## $ ID   <int> 1825, 1825, 1825, 3594, 3594, 817, 817, 817, 817, 577, 57...
## $ word <chr> "tablet", "tablet", "games", "recommend", "kindle", "alex...
## $ n    <int> 127, 127, 39, 22, 89, 40, 98, 19, 98, 21, 46, 25, 42, 16,...
```

We see that the data now consists of 1933 observations and 3 variables.

Now we can create the nodelist from the sampled tidy dataset.

```
nodes_reviews <- reviews_sampled_tidy_joined %>%
  group_by(ID) %>%
  select(word,ID) %>%
  ungroup() %>%
  distinct(word) # distinct does so we only retain the unique words from the nodelist -> so we dont get
```

Then we create a dataframe that we are going to be use for *left_joining* the edges (word.x and word.y)

```
nodes4join <- reviews_sampled_tidy_joined %>%
  select(ID, word)
```

```
g_reviews <- reviews_sampled_tidy_joined %>%
  left_join(nodes4join, by = c("ID" = "ID"))
```

```
g_reviews <- g_reviews[c("word.x", "word.y")]
```

Here we transform the dataframe into a graph using *igraph*.

We set *directed = FALSE*, because it is an undirected network.

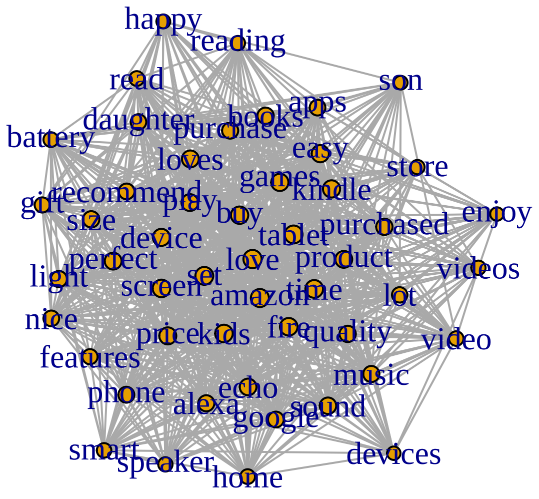
```
g_reviews <- graph_from_data_frame(d = g_reviews,
                                   directed = FALSE,
                                   vertices = nodes_reviews)
```

We use *simplify* to remove loops and multiple edges.

```
g_reviews <- simplify(g_reviews,
                      remove.multiple = TRUE,
                      remove.loops = TRUE,
                      edge.attr.comb = igraph_opt("edge.attr.comb"))
```

Now we can plot the network.

```
plot(g_reviews,
     vertex.size = 1 + sqrt(degree(g_reviews, mode = "all"))) # we assign the size to the degree and ad
```



At the graph we see that the size based on the degree is not that different between the words, but is seems as *device*, *tablet* & *love* is connected by *ID* to many words. This is not that surprising, because these where also some of the topwords from earlier.

Lets have a look at each nodes degree.

```r
degree(g_reviews) %>%
  as.data.frame()
```

```
##            .
## tablet    47
## games     43
## recommend 35
## kindle    43
## alexa     37
## love      49
## features  32
## video     29
## screen    45
## perfect   41
## price     42
## enjoy     24
## echo      40
## gift      32
## purchased 38
## set       47
## time      46
## kids      45
## reading   28
## lot       33
## sound     39
## speaker   28
## apps      35
## product   41
## buy       44
## devices   24
## purchase  37
## amazon    48
## home      30
## music     37
## fire      45
## easy      43
## quality   38
## size      38
## read      30
## books     38
## battery   30
## nice      35
## loves     41
## device    42
## smart     29
## play      41
## happy     26
## daughter  34
## phone     36
## light     37
## google    35
## son       25
## videos    28
## store     28
```

We can see that fx. *tap* only has degree 25, but *love* has degree 49. This makes sence, that not that many reviews has *tap* in the text, but rather more has *love* inside *(probably because the many high rated reviews)*.