Bag of words,TFIDF

CHINDU

Bag of words

The bag of words uses vector to specify which words are in each text.

Lets look at an example to understand the bag of words

Text1 <- c("John is sleeping") Text2<- c("John is awake") Text3<- c("John is missing")

We need to create a vector (Clean_vector) of the unique words in all of the text but first we need to carry out these steps 1) Lower/upper case all words 2) Remove stop words 3) Remove punctuations 4) carry out stemming / lemmatization

the clean vector will look like clean_vector<-("john", "sleeping", "awake", "missing")

Now lets see how each text looks in vector form we compare te clean_vector to each text. If the clean_vector contains the word in the text the 1 else 0

From the clean_vector we see that text 1 has john and sleeping but not awake and missing hence: Text1_Vector <- c(1,1,0,0)

Text2_Vector <- c(1,0,1,0) Text3_Vector <- c(1,0,0,1)

#Tidytext representation

The the original representation a document that does not contains the words receives a 0 for that word, in the tidytext representation, word pairs that do not exisst are left out.

```
# to find the words that appear in each review
datax<- read.csv("Womens Clothing ECommerce Reviews.csv")
str(datax)
```

```
## 'data.frame':    23486 obs. of  10 variables:
##  $ Clothing.ID            : int  0 1 1 1 2 3 4 5 6 7 ...
##  $ Age                    : int  26 50 36 24 28 36 28 39 39 39 ...
##  $ Title                  : Factor w/ 13994 levels "","\"beach business\"",..: 1 7365 11540 6984 501
##  $ Review.Text            : Factor w/ 22635 levels "","- this really is lovely. the overall design f
##  $ Rating                 : int  5 5 5 2 4 5 5 5 5 5 ...
##  $ Recommended.IND        : int  1 1 1 0 1 1 1 1 1 1 ...
##  $ Positive.Feedback.Count: int  0 0 0 1 0 0 0 0 0 0 ...
##  $ Division.Name          : Factor w/ 4 levels "","General","General Petite",..: 2 4 4 4 2 2 2 2 2 2
##  $ Department.Name        : Factor w/ 7 levels "","Bottoms","Dresses",..: 5 4 4 4 6 6 6 6 6 5 ...
##  $ Class.Name             : Factor w/ 21 levels "","Blouses","Casual bottoms",..: 14 11 11 11 10 19
```

```
library(dplyr)
data<-datax %>% select(Clothing.ID,Review.Text)
data$Review.Text<-as.character((data$Review.Text))
data$Clothing.ID<-as.factor(data$Clothing.ID)
```

```r
# Tokenize, remove stop words and do a word count by clothing id
words<- data %>%
  unnest_tokens(output= "word", token="words",input=Review.Text) %>%
  anti_join(stop_words)
```

```
## Joining, by = "word"
```

Clothing.ID and word pairs that do not exist in a review are left out in the tidytext representation.

Lets understatnd what the reviews are generally saying about the clothes with ID 1028

```r
words%>%
filter(Clothing.ID==1078) %>%
  count(word,sort=TRUE)
```

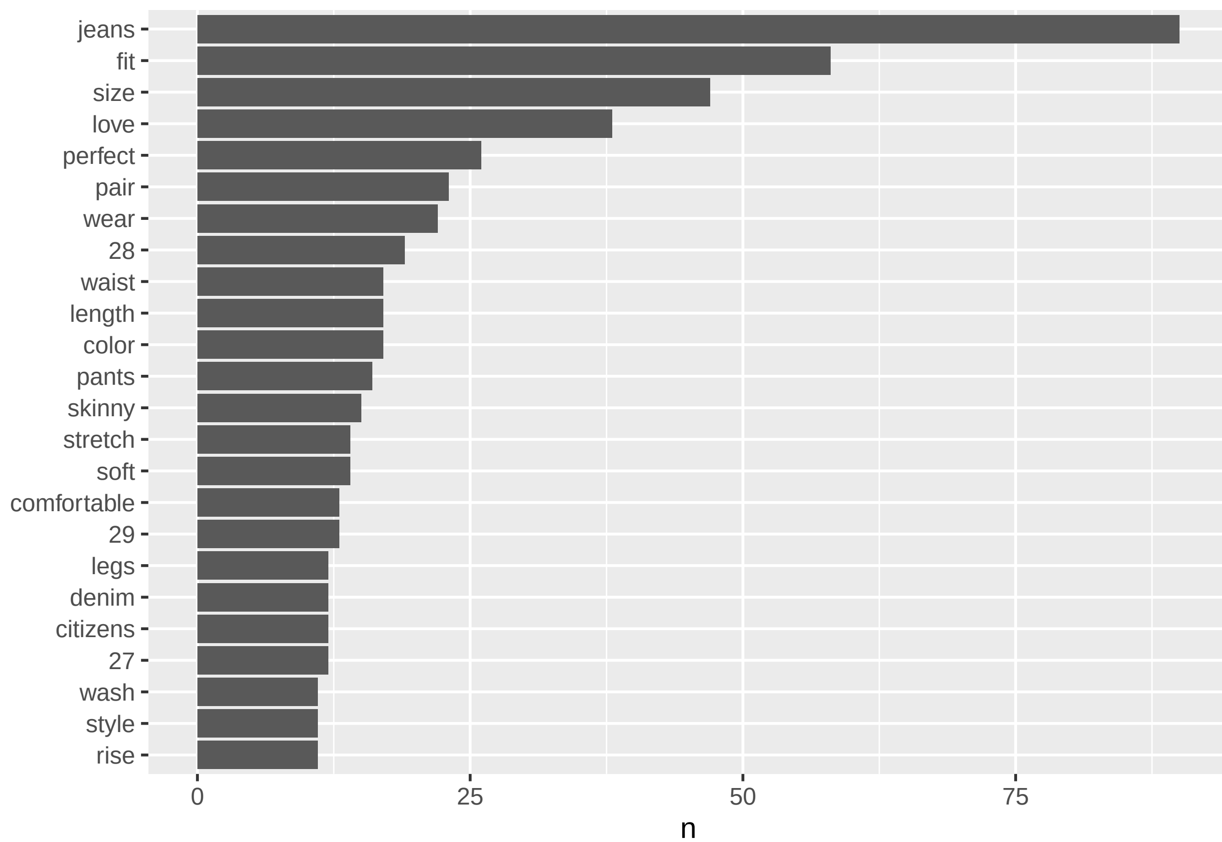
```
## # A tibble: 3,134 x 2
##    word            n
##    <chr>       <int>
##  1 dress        1406
##  2 size          397
##  3 love          372
##  4 fit           315
##  5 fabric        268
##  6 wear          264
##  7 flattering    198
##  8 perfect       182
##  9 color         173
## 10 comfortable   165
## # ... with 3,124 more rows
```

```r
library(ggplot2)
```

```
##
## Attaching package: 'ggplot2'

## The following object is masked from 'package:NLP':
##
##     annotate
```

```r
words%>%
  filter(Clothing.ID==1028) %>%
  count(word, sort = TRUE) %>%
  filter(n>10)%>%
  mutate(word = reorder(word, n)) %>%
  ggplot(aes(word, n)) +
  geom_col() +
  xlab(NULL) +
  coord_flip()
```



Most of the customers who left a review loved the product and they think is perfect. Such analysis help us to have a quick overview of how customer reacts to a product instead of reading through every reviews.

#Sparse matrix

parse matrices can become computational nightmares as the number of text documents and the number of unique words grow. Creating word representations with tweets can easily create sparse matrices because emojis, slang, acronyms, and other forms of language are used.

```
# How many unique words are there?
unique_words <-words %>%
  count(word,sort=TRUE)
unique_words
```

```
## # A tibble: 14,143 x 2
##    word           n
##    <chr>      <int>
##  1 dress      10553
##  2 love        8948
##  3 size        8768
##  4 top         7405
##  5 fit         7318
##  6 wear        6439
##  7 fabric      4790
##  8 color       4605
##  9 perfect     3772
## 10 flattering  3517
## # ... with 14,133 more rows
```

```r
# Count by id and word
unique_words_by_id <- words %>%
  count(Clothing.ID,word,sort=TRUE)
unique_words_by_id
```

```
## # A tibble: 175,019 x 3
##    Clothing.ID word      n
##    <fct>       <chr> <int>
##  1 1078        dress  1406
##  2 1094        dress  1127
##  3 1081        dress   869
##  4 1110        dress   786
##  5 1095        dress   527
##  6 862         top     481
##  7 1080        dress   426
##  8 1083        dress   411
##  9 1078        size    397
## 10 1086        dress   381
## # ... with 175,009 more rows
```

```r
# Find the size of the matrix
size <-nrow(data) * nrow(unique_words)
size
```

```
## [1] 332162498
```

```r
#Find percent of entries that would have a value
percent <-nrow(unique_words_by_id)/size
percent
```

```
## [1] 0.0005269078
```

TFIDF

Calculating TFIDF values relies on this bag-of-words representation, but takes into account how often a word appears in an article/reivew, and how often that word appears in the collection of articles/review.

To determine how meaningful words would be when comparing different reviews, calculate the TFIDF weights for the words

TF: Term frequency - Proportion of words in a text that are that term Text1 <- "john is awake" Text2 <- "john is sleeping" Text3 <- "john is missing" John is 1/4 words in text1, tf = 0.25 IDF: Inverse document frequency - weight of how common a term is across all documents IDF= log N/x N: total number of documents in the corpus x: number of documents where the term appears John:IDF = log(3/3) TFIDF for "John": text1: 1/4 * log(3/3)

```r
#Calculating the TFIDF matrix
words<-data %>%
  unnest_tokens(output='word',token="words",input=Review.Text)%>%
  anti_join(stop_words) %>%
  count(Clothing.ID,word,sort=TRUE) %>%
  bind_tf_idf(word,Clothing.ID,n)
```

```
## Joining, by = "word"
```

```
words
```

```
## # A tibble: 175,019 x 6
##    Clothing.ID word      n     tf   idf tf_idf
##    <fct>       <chr> <int>  <dbl> <dbl>  <dbl>
##  1 1078        dress  1406 0.0655 1.26  0.0828
##  2 1094        dress  1127 0.0679 1.26  0.0858
##  3 1081        dress   869 0.0705 1.26  0.0891
##  4 1110        dress   786 0.0692 1.26  0.0875
##  5 1095        dress   527 0.0666 1.26  0.0842
##  6 862         top     481 0.0321 0.901 0.0290
##  7 1080        dress   426 0.0671 1.26  0.0848
##  8 1083        dress   411 0.0705 1.26  0.0892
##  9 1078        size    397 0.0185 0.589 0.0109
## 10 1086        dress   381 0.0608 1.26  0.0769
## # ... with 175,009 more rows
```

We can use the tfidf values to access how similar two articles/reviews are if we use something called then cosine similarity.

Cosine similarity is the similarity between two vectors and is defined as the measure of the angles formed when representing the vectors in a multi dimensional space.

We can use the pairwise similarity function provided by the r package to calculate the cosine similarity of each pair of reviews

#Pairwise similarity pairwise_similairty(tbl, item, feature, value, ..) tbl: A table or tibble item: the item to compare (articles, tweets,etc) Feature: column describing the link between the item (i.e words) value: the column of values (i.e.n or tf_idf)

use cases - Find duplicates/similar pieces of text - use in clustering and classification analysis

```
# Calculate cosine similarity using tf_idf values
words %>%
  pairwise_similarity(Clothing.ID, word, tf_idf) %>%
  arrange(desc(similarity))
```

```
## # A tibble: 1,196,362 x 3
##    item1 item2 similarity
##    <fct> <fct>      <dbl>
##  1 1166  393        1
##  2 393   1166       1
##  3 1094  1078       0.952
##  4 1078  1094       0.952
##  5 1110  1078       0.943
##  6 1078  1110       0.943
##  7 1081  1078       0.943
##  8 1078  1081       0.943
##  9 1081  1094       0.937
## 10 1094  1081       0.937
## # ... with 1,196,352 more rows
```

The similarity measures how similar two reviews are, with 1 representing that the two reviews are exactly the same and 0 representing that the two reviews have nothing in common.