Text Mining Examples

# 1. Reuters-21578 데이터

데이터 설명

* 로이터 통신 뉴스 데이터로 21578개의 문서(토픽, 저자, 위치 등에 대한 메타데이터 존재)로 구성

library(tm)

## Loading required package: NLP

library(XML)  
  
# XML 포맷의 데이터를 parsing하여 Corpus로 저장  
reut21578 = system.file("texts","crude", package = "tm")  
(reuters = Corpus(DirSource(reut21578),  
 readerControl = list(reader = readReut21578XMLasPlain)))

## <<VCorpus>>  
## Metadata: corpus specific: 0, document level (indexed): 0  
## Content: documents: 20

# 데이터 구조 탐색  
str(reuters[1])

## List of 1  
## $ 127:List of 2  
## ..$ content: chr "Diamond Shamrock Corp said that\neffective today it had cut its contract prices for crude oil by\n1.50 dlrs a barrel.\n The "| \_\_truncated\_\_  
## ..$ meta :List of 16  
## .. ..$ author : chr(0)   
## .. ..$ datetimestamp: POSIXlt[1:1], format: NA  
## .. ..$ description : chr ""  
## .. ..$ heading : chr "DIAMOND SHAMROCK (DIA) CUTS CRUDE PRICES"  
## .. ..$ id : chr "127"  
## .. ..$ language : chr "en"  
## .. ..$ origin : chr "Reuters-21578 XML"  
## .. ..$ topics : chr "YES"  
## .. ..$ lewissplit : chr "TRAIN"  
## .. ..$ cgisplit : chr "TRAINING-SET"  
## .. ..$ oldid : chr "5670"  
## .. ..$ topics\_cat : chr "crude"  
## .. ..$ places : chr "usa"  
## .. ..$ people : chr(0)   
## .. ..$ orgs : chr(0)   
## .. ..$ exchanges : chr(0)   
## .. ..- attr(\*, "class")= chr "TextDocumentMeta"  
## ..- attr(\*, "class")= chr [1:2] "PlainTextDocument" "TextDocument"  
## - attr(\*, "class")= chr [1:2] "VCorpus" "Corpus"

# 숫자 제거  
reuters = tm\_map(reuters, removeNumbers)  
# 공백 제거   
reuters = tm\_map(reuters, stripWhitespace)  
# 불용어 제거   
reuters = tm\_map(reuters, removeWords,  
 stopwords("english"))  
# 구두점 제거   
reuters = tm\_map(reuters, removePunctuation,  
 preserve\_intra\_word\_dashes = TRUE)  
# 소문자로 변환  
reuters = tm\_map(reuters, content\_transformer(tolower))  
  
# 형태소분석  
library(SnowballC)  
reuters = tm\_map(reuters, stemDocument)  
  
# document-term-matrix 생성  
dtm = DocumentTermMatrix(reuters,   
 control=list(weighting=weightTf))  
inspect(dtm[1:5,1:5])

## <<DocumentTermMatrix (documents: 5, terms: 5)>>  
## Non-/sparse entries: 1/24  
## Sparsity : 96%  
## Maximal term length: 10  
## Weighting : term frequency (tf)  
##   
## Terms  
## Docs abdul-aziz abil abl abroad accept  
## 127 0 0 0 0 0  
## 144 0 2 0 0 0  
## 191 0 0 0 0 0  
## 194 0 0 0 0 0  
## 211 0 0 0 0 0

# 20회 이상 빈출어  
findFreqTerms(dtm, 20)

## [1] "barrel" "bpd" "crude" "dlrs" "last" "market" "mln"   
## [8] "oil" "opec" "price" "reuter" "said" "the"

# opec과 상관계수가 0.8이상  
findAssocs(dtm, "opec", 0.8)

## $opec  
## meet analyst emerg oil they buyer name said want   
## 0.92 0.87 0.87 0.87 0.85 0.84 0.83 0.83 0.83   
## tri   
## 0.82

# sparse한 용어 제거   
dtm2 = removeSparseTerms(dtm, 0.5)  
dtm2

## <<DocumentTermMatrix (documents: 20, terms: 9)>>  
## Non-/sparse entries: 138/42  
## Sparsity : 23%  
## Maximal term length: 6  
## Weighting : term frequency (tf)

# 워드클라우드   
freq = colSums(as.matrix(dtm2))  
freq2 = apply(as.matrix(dtm2), 2, function(x) sum(x>0))  
  
library(wordcloud)

## Loading required package: RColorBrewer

wordcloud(names(freq2), freq2, colors=rainbow(20))  
  
# 네트워크  
require("graph")

## Loading required package: graph

## Loading required package: BiocGenerics

## Loading required package: parallel

##   
## Attaching package: 'BiocGenerics'

## The following objects are masked from 'package:parallel':  
##   
## clusterApply, clusterApplyLB, clusterCall, clusterEvalQ,  
## clusterExport, clusterMap, parApply, parCapply, parLapply,  
## parLapplyLB, parRapply, parSapply, parSapplyLB

## The following objects are masked from 'package:stats':  
##   
## IQR, mad, xtabs

## The following objects are masked from 'package:base':  
##   
## anyDuplicated, append, as.data.frame, cbind, colnames,  
## do.call, duplicated, eval, evalq, Filter, Find, get, grep,  
## grepl, intersect, is.unsorted, lapply, lengths, Map, mapply,  
## match, mget, order, paste, pmax, pmax.int, pmin, pmin.int,  
## Position, rank, rbind, Reduce, rownames, sapply, setdiff,  
## sort, table, tapply, union, unique, unsplit

##   
## Attaching package: 'graph'

## The following object is masked from 'package:XML':  
##   
## addNode

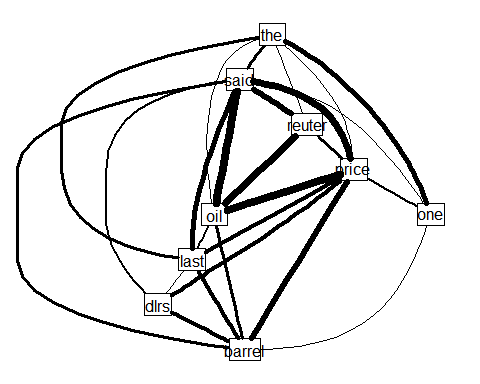
require("Rgraphviz")

## Loading required package: Rgraphviz

## Loading required package: grid



nNodes = nrow(dtm2)  
nA = list()  
nA$fixedSize = rep(FALSE, nNodes)  
nA$fontsize = rep(20, nNodes)  
nA = lapply(nA, function(x){names(x) <- Terms(dtm2); x})  
plot(dtm2, term=Terms(dtm2), corThreshold=0.1,  
 weighting=TRUE, nodeAttrs=nA)



# 2. 휴대폰 스팸 문자 필터링

목적: 휴대폰 SMS에서 스팸문자/정상문자를 구분하는 스팸 필터

<https://github.com/stedy/Machine-Learning-with-R-datasets/blob/master/sms_spam.csv에서> 다운로드 가능

## 2.1. 데이터 입력 및 처리

자료 처리

# 자료 입력 및 변환  
sms\_raw = read.csv("d:/work/sms\_spam.csv")  
str(sms\_raw)

## 'data.frame': 5559 obs. of 2 variables:  
## $ type: Factor w/ 2 levels "ham","spam": 1 1 1 2 2 1 1 1 2 1 ...  
## $ text: Factor w/ 5156 levels "'An Amazing Quote'' - Sometimes in life its difficult to decide whats wrong!! a lie that brings a smile or the truth that bring"| \_\_truncated\_\_,..: 1651 2557 257 626 3308 190 357 3392 2726 1079 ...

sms\_raw$type = factor(sms\_raw$type)  
  
str(sms\_raw$type)

## Factor w/ 2 levels "ham","spam": 1 1 1 2 2 1 1 1 2 1 ...

table(sms\_raw$type)

##   
## ham spam   
## 4812 747

# 텍스트 처리: 소문자로 변환, 숫자, and but 등 stopwords, 구두점, 공백 제거  
library(tm)  
sms\_corpus = Corpus(VectorSource(sms\_raw$text))  
  
print(sms\_corpus)

## <<VCorpus>>  
## Metadata: corpus specific: 0, document level (indexed): 0  
## Content: documents: 5559

inspect(sms\_corpus[1:3])

## <<VCorpus>>  
## Metadata: corpus specific: 0, document level (indexed): 0  
## Content: documents: 3  
##   
## [[1]]  
## <<PlainTextDocument>>  
## Metadata: 7  
## Content: chars: 49  
##   
## [[2]]  
## <<PlainTextDocument>>  
## Metadata: 7  
## Content: chars: 23  
##   
## [[3]]  
## <<PlainTextDocument>>  
## Metadata: 7  
## Content: chars: 43

corpus\_clean = tm\_map(sms\_corpus, tolower)  
corpus\_clean = tm\_map(corpus\_clean, removeNumbers)  
corpus\_clean = tm\_map(corpus\_clean, removeWords, stopwords())  
corpus\_clean = tm\_map(corpus\_clean, removePunctuation)  
corpus\_clean = tm\_map(corpus\_clean, stripWhitespace)  
corpus\_clean = tm\_map(corpus\_clean, PlainTextDocument)  
  
inspect(sms\_corpus[1:3])

## <<VCorpus>>  
## Metadata: corpus specific: 0, document level (indexed): 0  
## Content: documents: 3  
##   
## [[1]]  
## <<PlainTextDocument>>  
## Metadata: 7  
## Content: chars: 49  
##   
## [[2]]  
## <<PlainTextDocument>>  
## Metadata: 7  
## Content: chars: 23  
##   
## [[3]]  
## <<PlainTextDocument>>  
## Metadata: 7  
## Content: chars: 43

inspect(corpus\_clean[1:3])

## <<VCorpus>>  
## Metadata: corpus specific: 0, document level (indexed): 0  
## Content: documents: 3  
##   
## [[1]]  
## <<PlainTextDocument>>  
## Metadata: 7  
## Content: chars: 29  
##   
## [[2]]  
## <<PlainTextDocument>>  
## Metadata: 7  
## Content: chars: 17  
##   
## [[3]]  
## <<PlainTextDocument>>  
## Metadata: 7  
## Content: chars: 13

# 메시지별로 document - term 행렬 작성  
sms\_dtm = DocumentTermMatrix(corpus\_clean)  
sms\_dtm

## <<DocumentTermMatrix (documents: 5559, terms: 7991)>>  
## Non-/sparse entries: 42610/44379359  
## Sparsity : 100%  
## Maximal term length: 40  
## Weighting : term frequency (tf)

# 훈련 및 시험자료 분할  
sms\_raw\_train = sms\_raw[1:4169, ]  
sms\_raw\_test = sms\_raw[4170:5559, ]  
  
sms\_dtm\_train = sms\_dtm[1:4169, ]  
sms\_dtm\_test = sms\_dtm[4170:5559, ]  
  
sms\_corpus\_train = corpus\_clean[1:4169]  
sms\_corpus\_test = corpus\_clean[4170:5559]  
  
prop.table(table(sms\_raw\_train$type))

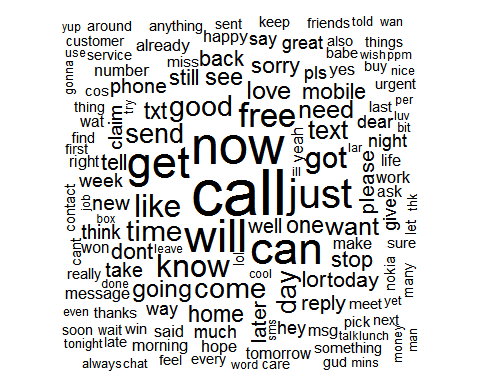
##   
## ham spam   
## 0.8647158 0.1352842

prop.table(table(sms\_raw\_test$type))

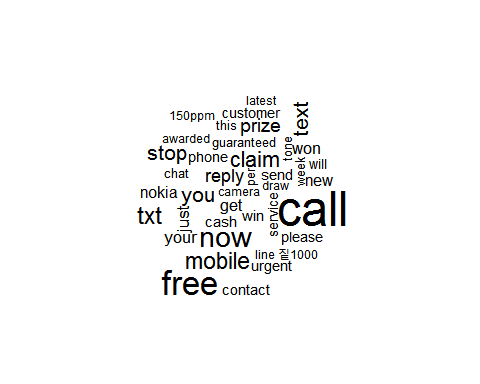
##   
## ham spam   
## 0.8683453 0.1316547

워드 클라우드를 이용한 시각화

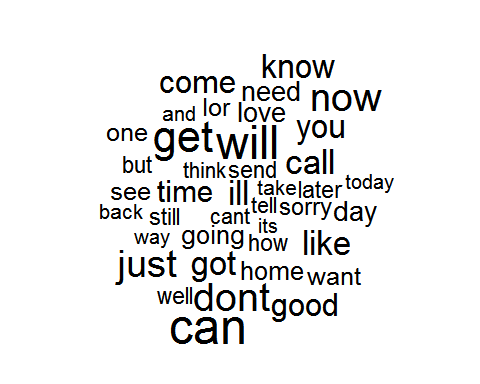
library(wordcloud)  
  
# random.order = F이면 빈도가 많은 단어가 중심에 나타남  
wordcloud(sms\_corpus\_train, min.freq = 30, random.order = FALSE)



# 그룹별로 시각화  
spam = subset(sms\_raw\_train, type == "spam")  
ham = subset(sms\_raw\_train, type == "ham")  
# max.words = 40개 이내의 단어, scale: 폰트 크기 범위   
wordcloud(spam$text, max.words = 40, scale = c(3, 0.5))



wordcloud(ham$text, max.words = 40, scale = c(3, 0.5))



# 자주 사용되는 단어에 대한 indicator 변수 생성  
head(findFreqTerms(sms\_dtm\_train, 5))

## [1] "abiola" "able" "abt" "accept" "access" "account"

sms\_dict = findFreqTerms(sms\_dtm\_train, 5) # 저장  
sms\_train = DocumentTermMatrix(sms\_corpus\_train, list(dictionary = sms\_dict))  
sms\_test = DocumentTermMatrix(sms\_corpus\_test, list(dictionary = sms\_dict))  
  
# 건수가 0 또는 1이상으로 변수 변환 (특정 단어가 들어가면 스팸일 확률)  
convert\_counts = function(x) {  
 x = ifelse(x > 0, 1, 0)  
 x = factor(x, levels = c(0, 1), labels = c("No", "Yes"))  
}  
  
sms\_train = apply(sms\_train, MARGIN = 2, convert\_counts)  
sms\_test = apply(sms\_test, MARGIN = 2, convert\_counts)

svm과 단순베이즈 분류 결과 비교

# 훈련  
library(e1071)  
sms\_classifier = naiveBayes(sms\_train, sms\_raw\_train$type)  
  
# 모형 평가  
sms\_test\_pred = predict(sms\_classifier, sms\_test)  
  
library(gmodels)  
CrossTable(sms\_test\_pred, sms\_raw\_test$type,  
 prop.chisq = FALSE, prop.t = FALSE, prop.r = FALSE,  
 dnn = c('predicted', 'actual'))

##   
##   
## Cell Contents  
## |-------------------------|  
## | N |  
## | N / Col Total |  
## |-------------------------|  
##   
##   
## Total Observations in Table: 1390   
##   
##   
## | actual   
## predicted | ham | spam | Row Total |   
## -------------|-----------|-----------|-----------|  
## ham | 1202 | 31 | 1233 |   
## | 0.996 | 0.169 | |   
## -------------|-----------|-----------|-----------|  
## spam | 5 | 152 | 157 |   
## | 0.004 | 0.831 | |   
## -------------|-----------|-----------|-----------|  
## Column Total | 1207 | 183 | 1390 |   
## | 0.868 | 0.132 | |   
## -------------|-----------|-----------|-----------|  
##   
##

# 라플라스 수정  
sms\_classifier2 = naiveBayes(sms\_train, sms\_raw\_train$type, laplace = 1)  
sms\_test\_pred2 = predict(sms\_classifier2, sms\_test)  
CrossTable(sms\_test\_pred2, sms\_raw\_test$type,  
 prop.chisq = FALSE, prop.t = FALSE, prop.r = FALSE,  
 dnn = c('predicted', 'actual'))

##   
##   
## Cell Contents  
## |-------------------------|  
## | N |  
## | N / Col Total |  
## |-------------------------|  
##   
##   
## Total Observations in Table: 1390   
##   
##   
## | actual   
## predicted | ham | spam | Row Total |   
## -------------|-----------|-----------|-----------|  
## ham | 1203 | 31 | 1234 |   
## | 0.997 | 0.169 | |   
## -------------|-----------|-----------|-----------|  
## spam | 4 | 152 | 156 |   
## | 0.003 | 0.831 | |   
## -------------|-----------|-----------|-----------|  
## Column Total | 1207 | 183 | 1390 |   
## | 0.868 | 0.132 | |   
## -------------|-----------|-----------|-----------|  
##   
##

## 연습문제

단어의 포함 여부가 아닌 각 문서에서 단어의 출현빈도를 입력변수로 하여 string 커널을 사용한 SVM과 k-NN으로 분류해 보고 예제의 결과와 비교하시오. 여기서 string 커널은 문자열의 분류를 위하여 제안된 커널로서 kernlab 패키지의 ksvm의 옵션 중 kernel = stringdot을 사용할 수 있다.

# 3. Internet Movie Database 데이터

데이터 설명

* 태그라인은 한 두 줄의 문구로 이루어진 영화에 대한 간략한 설명으로 영화제목과 링트되며 영화 포스터나 비디오 표지 등 광고에 사용됨
* 출처 : [IMDB](http://www.imdb.com)
* <http://www.informit.com/promotions/modeling-techniques-in-predictive-analytics-141183> 에서 7장 데이터 및 코드 다운로드 받을 수 있음

데이터

# "제목" (연도)로 시작

* 다음 줄에는 테그라인 내용이 나옴

데이터 처리 함수

library(tm)   
library(stringr)

##   
## Attaching package: 'stringr'

## The following object is masked from 'package:graph':  
##   
## boundary

library(grid)   
library(ggplot2)

##   
## Attaching package: 'ggplot2'

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

library(latticeExtra) # text horizon plot

## Loading required package: lattice

##   
## Attaching package: 'latticeExtra'

## The following object is masked from 'package:ggplot2':  
##   
## layer

library(wordcloud) # wordcloud  
library(cluster)   
  
# 표준화 함수   
standardize <- function(x) {(x - mean(x)) / sd(x)}  
  
# "invalid multibyte string" 메시지가 나오지 않도록 변환  
bytecode.convert <- function(x) {iconv(enc2utf8(x), sub = "byte")}  
  
NLINES <- 345317 # 데이터의 라인수   
input.data.file.name <- "d:/work/MTinPA/data/taglines\_copy\_data.txt"   
nlines\_to\_read <- 10000 # 블럭 크기   
  
# 디버그용 프린트 함수   
debub.print.mode <- FALSE  
debug.print <- function(title,date,tagline,status) {  
 cat("\n title =",title," date = ", date," tagline",  
 tagline, " status = ",status,"\n")  
 }  
  
  
# 텍스트 입력에 대한 parsing 함수  
tagline.parser <- function(input.list) {  
# 1915년 부터 올해까지를 유효 연도로   
 valid.years <- 1915:as.numeric(format(Sys.time(), "%Y"))  
 valid.years.strings.four <- paste("(",as.character(valid.years),sep="")  
  
 text <- input.list[[1]]  
 status <- input.list[[2]]  
 title <- input.list[[3]]  
 date <- input.list[[4]]  
 tagline <- input.list[[5]]  
   
 nitems <- length(text)  
 ncount <- 1 # 초기화   
 tagline\_data.store <- NULL  
   
 while(ncount < nitems) {   
 # 디버그 프린트   
 if (debub.print.mode) debug.print(title,date,tagline,status)   
 if (status == "indicator" | status == "begin") {  
 if (ncount <= nitems) {  
 ncount <- ncount + 1  
 status <- "initialtitle"  
 title <- " " # 초기값  
 date <- " "   
 tagline <- " "   
 }  
 }  
   
 if (status == "initialtitle") {  
 if (ncount <= nitems) {  
 title <- text[ncount]  
 ncount <- ncount + 1  
 if (ncount <= nitems) {  
 test\_date <- text[ncount]  
 if (substring(test\_date,1,5) %in% valid.years.strings.four) {  
 date <- test\_date  
 ncount <- ncount + 1  
 status <- "tagline"  
 }  
 if (!(substring(test\_date,1,5) %in% valid.years.strings.four)) {   
 if (test\_date == "#") {  
 status <- "indicator"  
 }   
 if (test\_date != "#") {  
 title <- paste(title, test\_date)   
 ncount <- ncount + 1   
 status <- "moretitle"  
 }   
 }   
 }   
 }  
 }  
   
 if (status == "moretitle") {  
 if (ncount <= nitems) {  
 ncount <- ncount + 1  
 if (ncount <= nitems) {  
 test\_date <- text[ncount]  
 if (substring(test\_date,1,5) %in% valid.years.strings.four) {  
 date <- test\_date  
 ncount <- ncount + 1  
 status <- "tagline"  
 }  
 if (!(substring(test\_date,1,5) %in% valid.years.strings.four)) {   
 if (test\_date == "#") {  
 status <- "indicator"  
 }   
 if (test\_date != "#") {  
 title <- paste(title, test\_date)   
 ncount <- ncount + 1   
 }   
 }   
 }   
 }  
 }   
   
 if (status == "tagline") {  
 if (ncount <= nitems) {  
 new\_text <- text[ncount]  
 if (new\_text == "#") {  
 tagline\_data.store <- rbind(tagline\_data.store,  
 data.frame(title, date, tagline, stringsAsFactors = FALSE))  
 status <- "indicator"  
 }   
 if (new\_text != "#") {  
 if (substring(new\_text,1,1) == "{") {  
 ncount <- ncount + 1  
 status <- "comment"  
 }  
 if (substring(new\_text,1,1) != "{") {  
 tagline <- paste(tagline, new\_text)  
 ncount <- ncount + 1  
 }  
 }   
 }   
 }  
   
 if (status == "comment") {  
 if (ncount <= nitems) {  
 new\_text <- text[ncount]   
 if (substring(new\_text,nchar(new\_text),nchar(new\_text)) == "}") {  
 ncount <- ncount + 1  
 status <- "tagline"  
 }  
 if (substring(new\_text,nchar(new\_text),nchar(new\_text)) != "}") {  
 ncount <- ncount + 1  
 }  
 }  
 }   
 } # while 루프 종료  
list(tagline\_data.store, status, title, date, tagline) # return list  
} # 함수 종료

데이터 읽기

cat("\n\n","NUMBER OF LINES READ: ")

##   
##   
## NUMBER OF LINES READ:

skip <- 0 # 건너뛸 줄수 초기화   
nlines\_read\_so\_far <- 0 # 현재까지 읽은 줄수 초기화   
  
  
status <- "begin" # 초기화   
title <- " "   
date <- " "   
tagline <- " "   
  
data.store <- NULL # 데이터 프레임 초기화   
  
while(nlines\_read\_so\_far < NLINES) {  
  
 if ((NLINES - nlines\_read\_so\_far) < nlines\_to\_read)   
 nlines\_to\_read <- (NLINES - nlines\_read\_so\_far)  
   
 text <- scan(file = input.data.file.name, what = "character",  
 skip = nlines\_read\_so\_far, nlines = nlines\_to\_read)  
   
 # 텍스트를 변환하여 "invalid multibyte string" 메시지가 나오지 않도록   
 text <- bytecode.convert(text)  
  
 input.list <- list(text, status, title, date, tagline)   
   
 # 블럭을 태그라인 parser로 parsing  
 output.list <- tagline.parser(input.list)   
   
 new\_data\_for\_store <- output.list[[1]]  
 status <- output.list[[2]]  
 title <- output.list[[3]]  
 date <- output.list[[4]]  
 tagline <- output.list[[5]]  
   
 data.store <- rbind(data.store, new\_data\_for\_store)  
   
 nlines\_read\_so\_far <- nlines\_read\_so\_far + nlines\_to\_read  
   
 cat(" ","nlines\_read\_so\_far:",nlines\_read\_so\_far)  
}

## nlines\_read\_so\_far: 10000 nlines\_read\_so\_far: 20000 nlines\_read\_so\_far: 30000 nlines\_read\_so\_far: 40000 nlines\_read\_so\_far: 50000 nlines\_read\_so\_far: 60000 nlines\_read\_so\_far: 70000 nlines\_read\_so\_far: 80000 nlines\_read\_so\_far: 90000 nlines\_read\_so\_far: 1e+05 nlines\_read\_so\_far: 110000 nlines\_read\_so\_far: 120000 nlines\_read\_so\_far: 130000 nlines\_read\_so\_far: 140000 nlines\_read\_so\_far: 150000 nlines\_read\_so\_far: 160000 nlines\_read\_so\_far: 170000 nlines\_read\_so\_far: 180000 nlines\_read\_so\_far: 190000 nlines\_read\_so\_far: 2e+05 nlines\_read\_so\_far: 210000 nlines\_read\_so\_far: 220000 nlines\_read\_so\_far: 230000 nlines\_read\_so\_far: 240000 nlines\_read\_so\_far: 250000 nlines\_read\_so\_far: 260000 nlines\_read\_so\_far: 270000 nlines\_read\_so\_far: 280000 nlines\_read\_so\_far: 290000 nlines\_read\_so\_far: 3e+05 nlines\_read\_so\_far: 310000 nlines\_read\_so\_far: 320000 nlines\_read\_so\_far: 330000 nlines\_read\_so\_far: 340000 nlines\_read\_so\_far: 345317

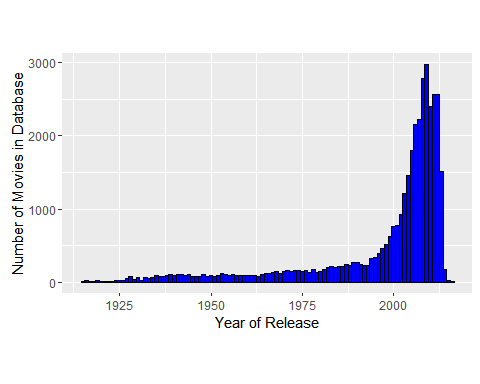
# output.list에 전체 영화 정보가 있으면 추가  
if ((!is.null(output.list[[3]])) &   
 (!is.null(output.list[[4]])) &  
 (!is.null(output.list[[5]]))) {  
 title <- output.list[[3]]  
 date <- output.list[[4]]  
 tagline <- output.list[[5]]   
 data.store <- rbind(data.store,   
 data.frame(title, date, tagline, stringsAsFactors = FALSE))  
}

데이터 청소

# 날짜: (로 시작하여 )로 끝남, 숫자를 제외한 것은 제거   
data.store$replace.date <- str\_replace\_all(data.store$date, "[^.(0-9)]", "")  
  
# 1915년부터 현재 년도까지 유효  
valid.years <- 1915:as.numeric(format(Sys.time(), "%Y"))  
valid.years.strings <- paste("(",as.character(valid.years),")",sep="")  
  
# 연도가 유효한지 체크   
data.store$valid <-   
 ifelse((data.store$replace.date %in% valid.years.strings),"YES","NO")  
  
# 유효한 데이터만 추출  
valid.data.store <- subset(data.store, subset = (valid == "YES"))  
  
# 제목에 연도를 붙여 유일한 키를 생성  
valid.data.store$movie <- paste(valid.data.store$title, valid.data.store$date)  
  
# 연도에서 괄호를 없애고 정수로 변환  
valid.data.store$replace.date <-   
 str\_replace(valid.data.store$replace.date,"[(]","")  
valid.data.store$replace.date <-   
 str\_replace(valid.data.store$replace.date,"[)]","")  
valid.data.store$year <- as.integer(valid.data.store$replace.date)  
  
# 제목과 태그라인 텍스트를 텍스트 변수로 저장  
valid.data.store$text <-   
 paste(valid.data.store$title, valid.data.store$tagline)  
  
# replace.date 제거하고 변수 재정렬  
movies <- valid.data.store[,c("movie","year","title","tagline","text")]

연도별 영화의 도수

ggplot.object <- ggplot(data = movies, aes(x = year)) +  
 geom\_histogram(binwidth = 1, fill = "blue", colour = "black") +  
 labs(x = "Year of Release",   
 y = "Number of Movies in Database") +  
 coord\_fixed(ratio = 1/50)   
ggplot.object



지난 1974~2013년까지 분석하기 위해 데이터 정리

years.list <- 1974:2013  
document.collection <- NULL   
for (index.for.year in seq(along=years.list)) {  
 working.year.data.frame =   
 subset(movies, subset = (year == years.list[index.for.year]))  
  
 tagline\_text <- NULL  
 for(index.for.movie in seq(along = working.year.data.frame$movie))   
 tagline\_text <-   
 paste(tagline\_text, working.year.data.frame$tagline[index.for.movie])  
   
 document <- PlainTextDocument(x = tagline\_text, author = "Tom",  
 description = paste("movie taglines for ",  
 as.character(years.list[index.for.year]),sep = ""),  
 id = paste("movies\_",as.character(years.list[index.for.year]),sep=""),   
 heading = "taglines",  
 origin = "IMDb", language = "en\_US",   
 localmetadata = list(year = years.list[index.for.year]))   
  
 # 생성된 문서에 유일한 이름 부여   
 if (years.list[index.for.year] == 1974) Y1974 <- document   
 if (years.list[index.for.year] == 1975) Y1975 <- document   
 if (years.list[index.for.year] == 1976) Y1976 <- document   
 if (years.list[index.for.year] == 1977) Y1977 <- document   
 if (years.list[index.for.year] == 1978) Y1978 <- document   
 if (years.list[index.for.year] == 1979) Y1979 <- document   
 if (years.list[index.for.year] == 1980) Y1980 <- document   
 if (years.list[index.for.year] == 1981) Y1981 <- document   
 if (years.list[index.for.year] == 1982) Y1982 <- document   
 if (years.list[index.for.year] == 1983) Y1983 <- document   
 if (years.list[index.for.year] == 1984) Y1984 <- document   
 if (years.list[index.for.year] == 1985) Y1985 <- document   
 if (years.list[index.for.year] == 1986) Y1986 <- document   
 if (years.list[index.for.year] == 1987) Y1987 <- document   
 if (years.list[index.for.year] == 1988) Y1988 <- document   
 if (years.list[index.for.year] == 1989) Y1989 <- document   
 if (years.list[index.for.year] == 1990) Y1990 <- document   
 if (years.list[index.for.year] == 1991) Y1991 <- document   
 if (years.list[index.for.year] == 1992) Y1992 <- document   
 if (years.list[index.for.year] == 1993) Y1993 <- document   
 if (years.list[index.for.year] == 1994) Y1994 <- document   
 if (years.list[index.for.year] == 1995) Y1995 <- document   
 if (years.list[index.for.year] == 1996) Y1996 <- document   
 if (years.list[index.for.year] == 1997) Y1997 <- document   
 if (years.list[index.for.year] == 1998) Y1998 <- document   
 if (years.list[index.for.year] == 1999) Y1999 <- document   
 if (years.list[index.for.year] == 2000) Y2000 <- document   
 if (years.list[index.for.year] == 2001) Y2001 <- document   
 if (years.list[index.for.year] == 2002) Y2002 <- document   
 if (years.list[index.for.year] == 2003) Y2003 <- document   
 if (years.list[index.for.year] == 2004) Y2004 <- document   
 if (years.list[index.for.year] == 2005) Y2005 <- document   
 if (years.list[index.for.year] == 2006) Y2006 <- document   
 if (years.list[index.for.year] == 2007) Y2007 <- document   
 if (years.list[index.for.year] == 2008) Y2008 <- document   
 if (years.list[index.for.year] == 2009) Y2009 <- document   
 if (years.list[index.for.year] == 2010) Y2010 <- document   
 if (years.list[index.for.year] == 2011) Y2011 <- document   
 if (years.list[index.for.year] == 2012) Y2012 <- document   
 if (years.list[index.for.year] == 2013) Y2013 <- document   
}   
   
document.collection <- c(Y1974,Y1975,Y1976,Y1977,Y1978,Y1979,  
 Y1980,Y1981,Y1982,Y1983,Y1984,Y1985,Y1986,Y1987,Y1988,Y1989,  
 Y1990,Y1991,Y1992,Y1993,Y1994,Y1995,Y1996,Y1997,Y1998,Y1999,  
 Y2000,Y2001,Y2002,Y2003,Y2004,Y2005,Y2006,  
 Y2007,Y2008,Y2009,Y2010,Y2011,Y2012,Y2013)  
  
# 공백 제거   
document.collection <- tm\_map(document.collection, stripWhitespace)  
  
# 대문자를 소문자로   
document.collection <- tm\_map(document.collection, tolower)  
  
# 숫자 제거  
document.collection <- tm\_map(document.collection, removeNumbers)  
  
# 구두점 제거  
document.collection <- tm\_map(document.collection, removePunctuation)  
  
# 영어의 stopwords 제거  
document.collection <- tm\_map(document.collection,   
 removeWords, stopwords("english"))

NLP가 아닌 bag of words 접근법으로 분석하기 위해 terms-by-document 행렬 생성

library(slam)  
document.collection <- tm\_map(document.collection, PlainTextDocument)  
initial.movies.tdm <- TermDocumentMatrix(document.collection)  
  
# 많이 나오지 않는 용어를 제거   
examine.movies.tdm <- removeSparseTerms(initial.movies.tdm, sparse = 0.25)  
top.words <- Terms(examine.movies.tdm)  
print(top.words)

## [1] "action" "adventure" "alive" "alone" "always"   
## [6] "america" "american" "americas" "another" "anything"   
## [11] "away" "back" "bad" "battle" "beautiful"   
## [16] "became" "become" "begins" "behind" "best"   
## [21] "beyond" "big" "biggest" "black" "blood"   
## [26] "body" "born" "boy" "boys" "business"   
## [31] "call" "came" "can" "cant" "century"   
## [36] "challenge" "chance" "city" "classic" "come"   
## [41] "comedy" "comes" "coming" "cop" "cops"   
## [46] "crime" "dangerous" "dark" "day" "days"   
## [51] "dead" "deadly" "death" "deep" "desire"   
## [56] "destroy" "didnt" "die" "different" "director"   
## [61] "doesnt" "dont" "dream" "dreams" "earth"   
## [66] "easy" "end" "enemy" "enough" "even"   
## [71] "ever" "every" "everyone" "everything" "evil"   
## [76] "experience" "eyes" "face" "family" "fantasy"   
## [81] "far" "fast" "father" "fear" "feel"   
## [86] "fight" "fighting" "film" "find" "first"   
## [91] "force" "forever" "forget" "found" "four"   
## [96] "friend" "friends" "full" "fun" "funny"   
## [101] "future" "game" "get" "gets" "girl"   
## [106] "girls" "goes" "going" "good" "got"   
## [111] "great" "greatest" "guys" "happen" "happens"   
## [116] "hard" "head" "heart" "hell" "help"   
## [121] "hero" "hes" "high" "history" "home"   
## [126] "hope" "horror" "hot" "house" "human"   
## [131] "isnt" "journey" "just" "justice" "keep"   
## [136] "kids" "kill" "killer" "killing" "kind"   
## [141] "king" "know" "knows" "land" "last"   
## [146] "law" "left" "legend" "let" "life"   
## [151] "lifetime" "like" "little" "live" "lives"   
## [156] "living" "look" "looking" "lost" "lot"   
## [161] "love" "loved" "loves" "made" "magic"   
## [166] "make" "man" "mans" "master" "may"   
## [171] "meet" "men" "million" "mind" "money"   
## [176] "mother" "movie" "much" "murder" "music"   
## [181] "must" "mystery" "name" "need" "never"   
## [186] "new" "next" "night" "nightmare" "nothing"   
## [191] "now" "old" "one" "original" "party"   
## [196] "passion" "past" "people" "perfect" "picture"   
## [201] "place" "play" "pleasure" "power" "powerful"   
## [206] "race" "ready" "real" "reality" "really"   
## [211] "remember" "revenge" "ride" "right" "rock"   
## [216] "run" "save" "say" "school" "secret"   
## [221] "see" "seen" "sex" "sexual" "shes"   
## [226] "show" "side" "someone" "something" "sometimes"   
## [231] "space" "stand" "star" "stars" "still"   
## [236] "stop" "story" "streets" "summer" "survival"   
## [241] "survive" "sweet" "take" "takes" "tale"   
## [246] "tell" "terrifying" "terror" "thats" "theres"   
## [251] "theyre" "thing" "things" "think" "thought"   
## [256] "three" "thriller" "time" "today" "together"   
## [261] "took" "tough" "town" "true" "truth"   
## [266] "turn" "two" "ultimate" "universe" "video"   
## [271] "want" "wanted" "wants" "war" "watch"   
## [276] "way" "weapon" "welcome" "wife" "wild"   
## [281] "will" "win" "without" "woman" "women"   
## [286] "wont" "world" "worlds" "worst" "wrong"   
## [291] "year" "years" "york" "youll" "young"   
## [296] "youre" "youve"

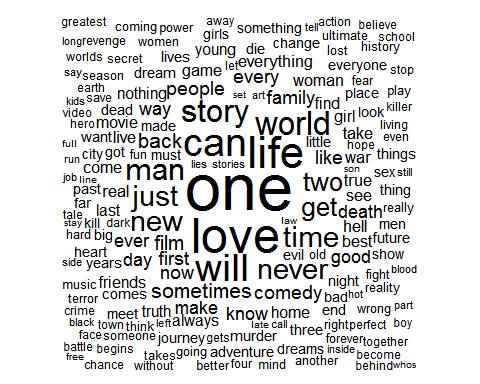
# 분석에서 사용하지 않을 추가 단어들 제거  
more.stop.words <- c("cant","didnt","doesnt","dont","goes","isnt","hes",  
 "shes","thats","theres","theyre","wont","youll","youre","youve")   
document.collection <- tm\_map(document.collection,   
 removeWords, more.stop.words)  
   
# 최종적으로 terms-by-document 행렬 생성  
movies.tdm <- TermDocumentMatrix(document.collection)  
  
# 저장  
save("movies","document.collection","movies.tdm",  
 file = "000\_movies\_data.Rdata")   
  
# 많이 나오지 않는 단어들 제거   
examine.movies.tdm <- removeSparseTerms(movies.tdm, sparse = 0.25)  
  
top.words <- Terms(examine.movies.tdm)  
print(top.words)

## [1] "action" "adventure" "alive" "alone" "always"   
## [6] "america" "american" "americas" "another" "anything"   
## [11] "away" "back" "bad" "battle" "beautiful"   
## [16] "became" "become" "begins" "behind" "best"   
## [21] "beyond" "big" "biggest" "black" "blood"   
## [26] "body" "born" "boy" "boys" "business"   
## [31] "call" "came" "can" "century" "challenge"   
## [36] "chance" "city" "classic" "come" "comedy"   
## [41] "comes" "coming" "cop" "cops" "crime"   
## [46] "dangerous" "dark" "day" "days" "dead"   
## [51] "deadly" "death" "deep" "desire" "destroy"   
## [56] "die" "different" "director" "dream" "dreams"   
## [61] "earth" "easy" "end" "enemy" "enough"   
## [66] "even" "ever" "every" "everyone" "everything"  
## [71] "evil" "experience" "eyes" "face" "family"   
## [76] "fantasy" "far" "fast" "father" "fear"   
## [81] "feel" "fight" "fighting" "film" "find"   
## [86] "first" "force" "forever" "forget" "found"   
## [91] "four" "friend" "friends" "full" "fun"   
## [96] "funny" "future" "game" "get" "gets"   
## [101] "girl" "girls" "going" "good" "got"   
## [106] "great" "greatest" "guys" "happen" "happens"   
## [111] "hard" "head" "heart" "hell" "help"   
## [116] "hero" "high" "history" "home" "hope"   
## [121] "horror" "hot" "house" "human" "journey"   
## [126] "just" "justice" "keep" "kids" "kill"   
## [131] "killer" "killing" "kind" "king" "know"   
## [136] "knows" "land" "last" "law" "left"   
## [141] "legend" "let" "life" "lifetime" "like"   
## [146] "little" "live" "lives" "living" "look"   
## [151] "looking" "lost" "lot" "love" "loved"   
## [156] "loves" "made" "magic" "make" "man"   
## [161] "mans" "master" "may" "meet" "men"   
## [166] "million" "mind" "money" "mother" "movie"   
## [171] "much" "murder" "music" "must" "mystery"   
## [176] "name" "need" "never" "new" "next"   
## [181] "night" "nightmare" "nothing" "now" "old"   
## [186] "one" "original" "party" "passion" "past"   
## [191] "people" "perfect" "picture" "place" "play"   
## [196] "pleasure" "power" "powerful" "race" "ready"   
## [201] "real" "reality" "really" "remember" "revenge"   
## [206] "ride" "right" "rock" "run" "save"   
## [211] "say" "school" "secret" "see" "seen"   
## [216] "sex" "sexual" "show" "side" "someone"   
## [221] "something" "sometimes" "space" "stand" "star"   
## [226] "stars" "still" "stop" "story" "streets"   
## [231] "summer" "survival" "survive" "sweet" "take"   
## [236] "takes" "tale" "tell" "terrifying" "terror"   
## [241] "thing" "things" "think" "thought" "three"   
## [246] "thriller" "time" "today" "together" "took"   
## [251] "tough" "town" "true" "truth" "turn"   
## [256] "two" "ultimate" "universe" "video" "want"   
## [261] "wanted" "wants" "war" "watch" "way"   
## [266] "weapon" "welcome" "wife" "wild" "will"   
## [271] "win" "without" "woman" "women" "world"   
## [276] "worlds" "worst" "wrong" "year" "years"   
## [281] "york" "young"

# 빈출어들로부터 사전 만들기  
top.words.dictionary <- c(top.words)  
   
# 생성한 사전을 이용하여 terms-by-documents 행렬 만들기  
top.words.movies.tdm <- TermDocumentMatrix(document.collection,   
 list(dictionary = top.words.dictionary))

워드클라우드

set.seed(1234)   
wordcloud(document.collection, min.freq = 10,  
 max.words = 300,  
 random.order=FALSE,  
 random.color=FALSE,  
 rot.per=0.0,  
 colors="black",  
 ordered.colors=FALSE,   
 use.r.layout=FALSE,  
 fixed.asp=TRUE)



## 연습문제

* document.collection에 대하여 wordcloud를 그려보시오.
* top.words.movies.tdm에 대하여 적절한 군집분석을 실시하고 결과를 해석해 보시오.