STA 380 Homework 2

Wenduo Wang

August 13, 2016

## Author attribution

Import tm and dplyr libraries. The former is the text mining library, and the latter provides tools for matrix operations.

## [1] 0

## used (Mb) gc trigger (Mb) max used (Mb)  
## Ncells 412579 22.1 750400 40.1 592000 31.7  
## Vcells 650033 5.0 1308461 10.0 1023697 7.9

library(tm)  
library(dplyr)

The first thing to do with the documents is to convert them to Corpus objects defined in tm library. While the texts are read in, a series of operations are performed, including enforcing lowercase, removing numbers & punctuations, and stripping white spaces. It is sometimes problematic to remove *stopwords* as such could change the meaning of the text. But given the abundance of data in this case, removing *stopwords* provides more benefits in terms of simplifying computation.

In detail, we defined a helper function that takes in a folder url which contains all the subfolders and documents, and then read in all the documents and convert them into a corpus object.

The training set and test set are converted into Corpus objects in the above way, and then the object is transformed into a DocumentTermMatrix and later normal matrix. No sparse terms are teased out at this stage.

get\_mat <- function(url, tfidf=1){  
 author\_list <- Sys.glob(url)  
  
 file\_list <- lapply(paste(author\_list, "/\*", sep=""), function(folder) Sys.glob(folder))  
   
 readFolder <- function(folder){  
 article\_list <- lapply(folder, function(fname) readPlain(elem=list(content=readLines(fname)), language="en", id=fname))  
 names(article\_list) <- folder  
 names(article\_list) <- sapply(names(article\_list), function(s) gsub(".txt", "", s))  
 names(article\_list) <- sapply(names(article\_list), function(s) gsub(".+/", "", s))  
 return(article\_list)  
 }  
   
 documents <- lapply(file\_list, readFolder)  
   
 names(documents) <- Sys.glob(url)  
 names(documents) <- sapply(names(documents), function(s) gsub(".+/", "", s))  
   
 documents\_corpus <- Corpus(VectorSource(documents))  
 names(documents\_corpus) <- names(documents)  
   
 documents\_corpus <- tm\_map(documents\_corpus, content\_transformer(tolower))  
 documents\_corpus <- tm\_map(documents\_corpus, content\_transformer(removeNumbers))  
 documents\_corpus <- tm\_map(documents\_corpus, content\_transformer(removePunctuation))  
 documents\_corpus <- tm\_map(documents\_corpus, content\_transformer(stripWhitespace))  
 documents\_corpus <- tm\_map(documents\_corpus, content\_transformer(removeWords), stopwords("en"))  
   
 # define an optional "control" parameter  
 # to weight the terms according to their tfidf index  
 if (tfidf) {  
 control <- list(weighting=function(x) weightTfIdf(x, normalize=FALSE))  
 documents\_DTM <- DocumentTermMatrix(documents\_corpus, control=control)  
 } else {  
 documents\_DTM <- DocumentTermMatrix(documents\_corpus)  
 }  
   
   
 documents\_mat <- as.matrix(documents\_DTM)  
   
 return(documents\_mat)  
}  
  
train\_mat <- get\_mat("data/ReutersC50/C50train/\*")

> Note: no visible binding for global variable '.Class'   
> Note: no visible binding for global variable '.Class'

test\_mat <- get\_mat("data/ReutersC50/C50test/\*")

Another problem when it comes to modeling and prediction, particularly in text mining, is the difference of words between the training set and test set. Here we define another helper function to remove the different columns, so only the shared words will be considered in the two datasets.

drop\_columns <- function(mat\_1, mat\_2){  
 drop\_cols <- c()  
 mat\_1\_terms <- colnames(mat\_1)  
 mat\_2\_terms <- colnames(mat\_2)  
   
 for (i in 1:length(mat\_1\_terms)){  
 if (!(mat\_1\_terms[i] %in% mat\_2\_terms)){  
 drop\_cols <- c(drop\_cols, i)  
 }  
 }  
   
 mat\_1 <- mat\_1[, -drop\_cols]  
}

Then using the above defined function, we transform the training and test sets. At this point, we also calculate the row vector length for training and test sets, which will be useful to calculate the vector angle between training and test vectors.

train\_dropped <- drop\_columns(train\_mat, test\_mat)  
train\_rscalar <- sqrt(diag(train\_dropped%\*%t(train\_dropped)))  
test\_dropped <- drop\_columns(test\_mat, train\_mat)  
test\_rscalar <- sqrt(diag(test\_dropped%\*%t(test\_dropped)))

### Model 1: matrix product and decide according to vector angle.

Important assumption: since the *test* set files are also stored in a folder-subfolder structure, it is assumes all the articles in the subfolder belong to the same author, which in effect expands each *unknown author's* terms matrix. This applies for model 2 as well.

# matrix multiplication between training and test vectors  
product <- train\_dropped %\*% t(test\_dropped)  
  
# scale the matrix product using training   
# and test vector lengths to get the cosine of their angle  
product <- product/matrix(rep(train\_rscalar, ncol(product)), ncol = ncol(product))  
product <- product/matrix(rep(test\_rscalar, ncol(product)), ncol = ncol(product))  
  
# in this model, if the two vectors pointing to the same direction  
# then they are seen as from the same author.  
# Therefore the predicted authors (in terms of their row index in the training set)  
# are the row indices corresponding to the max value in each column.  
authors <- sapply(1:ncol(product), function(n) which.max(product[, n]))  
  
# since we know the true authors are in the same order  
# in the test set as in the training set, we can calculate the prediction accuracy  
# by comparing the authors indices with the sequence from 1 to the number of authors.  
accuracy\_1 <- mean(authors==1:length(authors))  
cat("Model 1 accuracy:", accuracy\_1)

> Model 1 accuracy: 0.8

As seen from the output, this model gives accuracy of 80% (if the terms were not weighted according to their tfidf index, the accuracy is 30%).

### Model 2: Naive Bayes

We can also apply Naive Bayes method to determine the authors. Specifically, in this case our problem is to predict the probability

Where *x* can be any of the known authors from the training set. According to the general Bayes theorem, the above probability is equal to

Take logarithm of the expression and you will get the log probability terms as matrix expression

Where *smoothing count* is different from what was said durin the lecture, but is purely a tiny amount to avoid *infinity* values inside logarithm. Since we now in our case # articles by each author is the same, i.e. 50, the second term above is a constant, and the third term is independent of author and therefore is also a constant given the test set. We only need to calculate and compare the first term to determine the author.

However, for Naive Bayes method, the terms do not need to be converted to tfidf, so the *tfidf* flag is turned off to obtain the DTM object.

train\_mat <- get\_mat("data/ReutersC50/C50train/\*", tfidf=0)  
test\_mat <- get\_mat("data/ReutersC50/C50test/\*", tfidf=0)  
train\_dropped <- drop\_columns(train\_mat, test\_mat)  
test\_dropped <- drop\_columns(test\_mat, train\_mat)

Similar to model 1, the accuracy is measure as the fraction of correct attribution.

train\_prob <- log((train\_dropped+1/50)/sum(train\_dropped))  
author\_prob <- train\_prob %\*% t(test\_dropped)  
authors <- sapply(1:ncol(author\_prob), function(n) which.max(author\_prob[, n]))  
accuracy\_2 <- mean(authors==1:length(authors))  
cat("Model 2 accuracy:", accuracy\_2)

> Model 2 accuracy: 0.88

It is seen that Naive Bayes method gives a little higher accuracy over the vanilla vector angle method. But the different is small in this case: 88% vs 80%. Considering the small sample size (50 in total since each subfolder is assumed to belong to the same author) this makes it difficult to assert which model is better.

## Practice with association rule mining

## [1] 3

## used (Mb) gc trigger (Mb) max used (Mb)  
## Ncells 492591 26.4 1168576 62.5 1168576 62.5  
## Vcells 825680 6.3 14303478 109.2 17879331 136.5

Import arules library for association rule mining.

library(arules)

Import data with read.transactions() function from arules, which will automatically convert each row into a list of items separated by commas, and the returned object is a transactions object. This function will also drop duplicate items from each basket if rm.duplicates = TRUE.

groceries <- read.transactions("data/groceries.txt", sep=",", rm.duplicates = TRUE)

We can assign each user an id by converting the transactions to a list with as(from="transactions", to="list"), and define names() of the list, and then convert the list back to transactions.

groceries\_list <- as(groceries, "list")  
names(groceries\_list) <- as.character(1:length(groceries\_list))  
groceries <- as(groceries\_list, "transactions")

At this point, the *groceries* object is suitable for a priori analysis. Before applying the apriori function, we need to determine what the *support* and *confidence* thresholds, and *maxlen* value. This is essentially a heuristic process, so here let's first try a higher *support* level 0.01 and *confidence* threshold 0.55 (just a little bit more than 0.5), and see what we get.

params <- list(support=.01, confidence=.55, maxlen=4)  
grocery\_rules <- apriori(groceries, parameter = params)

> Apriori  
>   
> Parameter specification:  
> confidence minval smax arem aval originalSupport support minlen maxlen  
> 0.55 0.1 1 none FALSE TRUE 0.01 1 4  
> target ext  
> rules FALSE  
>   
> Algorithmic control:  
> filter tree heap memopt load sort verbose  
> 0.1 TRUE TRUE FALSE TRUE 2 TRUE  
>   
> Absolute minimum support count: 98   
>   
> set item appearances ...[0 item(s)] done [0.00s].  
> set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].  
> sorting and recoding items ... [88 item(s)] done [0.00s].  
> creating transaction tree ... done [0.00s].  
> checking subsets of size 1 2 3 4 done [0.00s].  
> writing ... [7 rule(s)] done [0.00s].  
> creating S4 object ... done [0.00s].

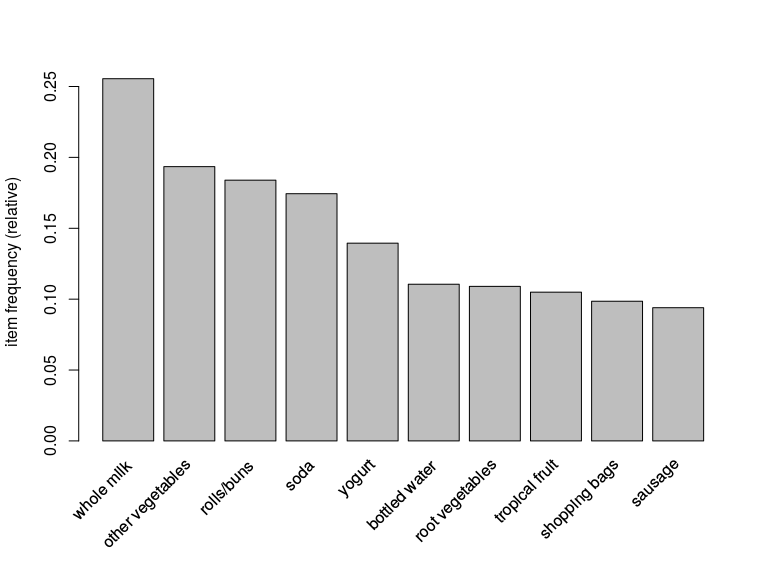
inspect(subset(grocery\_rules, subset=lift>=2))

> lhs rhs support confidence lift  
> 1 {curd,   
> yogurt} => {whole milk} 0.01006609 0.5823529 2.279125  
> 2 {butter,   
> other vegetables} => {whole milk} 0.01148958 0.5736041 2.244885  
> 3 {domestic eggs,   
> other vegetables} => {whole milk} 0.01230300 0.5525114 2.162336  
> 4 {citrus fruit,   
> root vegetables} => {other vegetables} 0.01037112 0.5862069 3.029608  
> 5 {root vegetables,   
> tropical fruit} => {other vegetables} 0.01230300 0.5845411 3.020999  
> 6 {root vegetables,   
> tropical fruit} => {whole milk} 0.01199797 0.5700483 2.230969  
> 7 {root vegetables,   
> yogurt} => {whole milk} 0.01453991 0.5629921 2.203354

Here we only selected the associations with *lift* greater than 2, which gives 7 in total. And among them are items such as *other vegetables* and *whole milk*, which are themselves frequent terms across all baskets. Such results provide limited information, so we need to look closer into more interesting and less ubiquitous items.

So a natural question to be asked here is, which are the most frequent items? Let's make a plot to show the top 10 frequent terms.

itemFrequencyPlot(groceries, topN=10)



So here we can see clearly that *whole milk* and *other vegetables* are indeed frequent terms containing relatively less information.

Therefore, let's lower *support* level to 0.001 to include less often items. Also, when printing out the associations, we raise the *lift* threshold to 10, which indicates highly correlated and dependent items.

params <- list(support=.001, confidence=.55, maxlen=4)  
grocery\_rules <- apriori(groceries, parameter = params)

> Apriori  
>   
> Parameter specification:  
> confidence minval smax arem aval originalSupport support minlen maxlen  
> 0.55 0.1 1 none FALSE TRUE 0.001 1 4  
> target ext  
> rules FALSE  
>   
> Algorithmic control:  
> filter tree heap memopt load sort verbose  
> 0.1 TRUE TRUE FALSE TRUE 2 TRUE  
>   
> Absolute minimum support count: 9   
>   
> set item appearances ...[0 item(s)] done [0.00s].  
> set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].  
> sorting and recoding items ... [157 item(s)] done [0.00s].  
> creating transaction tree ... done [0.00s].  
> checking subsets of size 1 2 3 4 done [0.01s].  
> writing ... [3314 rule(s)] done [0.00s].  
> creating S4 object ... done [0.00s].

inspect(subset(grocery\_rules, subset=lift>=10))

> lhs rhs support confidence lift  
> 1 {liquor,   
> red/blush wine} => {bottled beer} 0.001931876 0.9047619 11.23527  
> 2 {popcorn,   
> soda} => {salty snack} 0.001220132 0.6315789 16.69779  
> 3 {Instant food products,   
> soda} => {hamburger meat} 0.001220132 0.6315789 18.99565  
> 4 {ham,   
> processed cheese} => {white bread} 0.001931876 0.6333333 15.04549  
> 5 {baking powder,   
> flour} => {sugar} 0.001016777 0.5555556 16.40807  
> 6 {hard cheese,   
> whipped/sour cream,   
> yogurt} => {butter} 0.001016777 0.5882353 10.61522  
> 7 {hamburger meat,   
> whipped/sour cream,   
> yogurt} => {butter} 0.001016777 0.6250000 11.27867

Here we see some intriguing associations which are less frequent among all baskets but exhibits huge correlation in terms of *lift*, which is a measure of dependence. While in the previous case there were only 7 associations even with *lift* level higher than 2, here there are 7 associations with *lift* higher than 10.

Let's look at the association {liquor, red/blush wine} => {bottled beer}, it is intuitively this is some combination appealing to an alcohol lover. This intuition also holds for other association groups, such as {baking powder, flour} => {sugar} which is probably a part of common baking recipe.

And what about other associations?

params <- list(support=.001, confidence=.55, maxlen=4)  
grocery\_rules <- apriori(groceries, parameter = params)

> Apriori  
>   
> Parameter specification:  
> confidence minval smax arem aval originalSupport support minlen maxlen  
> 0.55 0.1 1 none FALSE TRUE 0.001 1 4  
> target ext  
> rules FALSE  
>   
> Algorithmic control:  
> filter tree heap memopt load sort verbose  
> 0.1 TRUE TRUE FALSE TRUE 2 TRUE  
>   
> Absolute minimum support count: 9   
>   
> set item appearances ...[0 item(s)] done [0.00s].  
> set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].  
> sorting and recoding items ... [157 item(s)] done [0.00s].  
> creating transaction tree ... done [0.00s].  
> checking subsets of size 1 2 3 4 done [0.01s].  
> writing ... [3314 rule(s)] done [0.00s].  
> creating S4 object ... done [0.00s].

inspect(subset(grocery\_rules, subset=(lift<=10 & lift>8)))

> lhs rhs support confidence lift  
> 1 {frozen vegetables,   
> specialty chocolate} => {fruit/vegetable juice} 0.001016777 0.6250000 8.645394  
> 2 {frozen fish,   
> other vegetables,   
> tropical fruit} => {pip fruit} 0.001016777 0.6666667 8.812724  
> 3 {flour,   
> root vegetables,   
> whole milk} => {whipped/sour cream} 0.001728521 0.5862069 8.177794  
> 4 {misc. beverages,   
> other vegetables,   
> tropical fruit} => {fruit/vegetable juice} 0.001016777 0.5882353 8.136841  
> 5 {citrus fruit,   
> fruit/vegetable juice,   
> grapes} => {tropical fruit} 0.001118454 0.8461538 8.063879  
> 6 {fruit/vegetable juice,   
> grapes,   
> tropical fruit} => {citrus fruit} 0.001118454 0.6875000 8.306588  
> 7 {citrus fruit,   
> grapes,   
> tropical fruit} => {fruit/vegetable juice} 0.001118454 0.6111111 8.453274  
> 8 {butter,   
> hard cheese,   
> yogurt} => {whipped/sour cream} 0.001016777 0.6250000 8.718972  
> 9 {butter,   
> hard cheese,   
> other vegetables} => {whipped/sour cream} 0.001220132 0.6000000 8.370213  
> 10 {butter,   
> hard cheese,   
> whole milk} => {whipped/sour cream} 0.001423488 0.6666667 9.300236  
> 11 {ham,   
> other vegetables,   
> tropical fruit} => {pip fruit} 0.001626843 0.6153846 8.134822  
> 12 {butter,   
> sliced cheese,   
> whole milk} => {whipped/sour cream} 0.001220132 0.6000000 8.370213  
> 13 {cream cheese,   
> sugar,   
> whole milk} => {domestic eggs} 0.001118454 0.5500000 8.668670  
> 14 {curd,   
> sugar,   
> yogurt} => {whipped/sour cream} 0.001016777 0.6250000 8.718972  
> 15 {butter,   
> other vegetables,   
> sugar} => {whipped/sour cream} 0.001016777 0.7142857 9.964539  
> 16 {citrus fruit,   
> cream cheese,   
> whole milk} => {domestic eggs} 0.001626843 0.5714286 9.006410  
> 17 {domestic eggs,   
> frankfurter,   
> tropical fruit} => {pip fruit} 0.001016777 0.6250000 8.261929  
> 18 {shopping bags,   
> tropical fruit,   
> whipped/sour cream} => {pip fruit} 0.001118454 0.6470588 8.553526

Above are associations with *lift* between 8 and 10. And here we can see some interesting combinations such as {butter, hard cheese, milk} => {whipped/sour cream}. Why is the customer buying such protein and fat heavy foots altogether? Probably it is simply because of the way these products are placed in the store. If some products are placed together, then they are more likely to be sold in a bundle. This can also be seen in {citrus fruit, grapes, tropical fruit} => {fruit/vegetable juice}.