CIS585 Project Report

Logan O’Connor

May 4, 2019

### Background

This analysis was done to determine whether or not video game developers who publish titles to Steam can get any meaningful feedback from the reviews players posted on the store page. This report will show the process of retrieving the reviews with Steam’s web API using the rvest and jsonlite package, text mining using the tm and hunspell packages, and finally visualizing the bigrams and their frequencies using ggwordcloud.

### Packages used

#Required packages  
require(rvest)

## Loading required package: rvest

## Loading required package: xml2

require(jsonlite)

## Loading required package: jsonlite

require(tm)

## Loading required package: tm

## Loading required package: NLP

require(ggwordcloud)

## Loading required package: ggwordcloud

## Loading required package: ggplot2

##   
## Attaching package: 'ggplot2'

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

require(dplyr)

## Loading required package: dplyr

##   
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':  
##   
## filter, lag

## The following objects are masked from 'package:base':  
##   
## intersect, setdiff, setequal, union

require(anytime)

## Loading required package: anytime

require(tidytext)

## Loading required package: tidytext

require(hunspell)

## Loading required package: hunspell

### Setting up variables for API request

steamappid <- "289070" #ID of the game/app in Steam  
url <- paste0('https://store.steampowered.com/appreviews/',steamappid,'?json=1&filter=all&language=english&day\_range=9223372036854775807&num\_per\_page=100&purchase\_type=all&start\_offset=')  
storeurl <- read\_html(paste0("https://store.steampowered.com/app/",steamappid))  
gamename <- tolower(storeurl %>% html\_nodes("div.apphub\_AppName") %>% html\_text())  
gamename <- strsplit(gamename, " ")[[1]]

Steam’s web API returns a page of JSON based on what parameters you give it. The documention for this API request can be read [here](https://partner.steamgames.com/doc/store/getreviews). Every application in the Steam Store has it’s own unique ID, the API uses this ID to tell what app we want to request reviews from. For this demonstration we picked the game “Sid Meier’s Civilization VI” and stored it’s ID in the steamappid variable. Next, using the paste0 function, create part of our API request URL. The parameters in the JSON request are set to return what kind of reviews are wanted, in this case it will be pulling all reviews that are english. The day range parameter is set to 9223372036854775807 due to it being the maximum amount of days it can be set to. The JSON request can only return a maximum of 100 reviews per request so the num\_per\_page parameter is set to match that. Finally the start\_offset is set to be blank because the script will be cycling through this based on how many reviews it is told to grab. Also in this section of code, using the Rvest package, the HTML of the store page for the game is read and the name of the game is stored in a variable and is then split into a list of words for purpose of removing any instance of the games name from the review data.

### Gathering review data

pages <- list() #The list that the raw JSON will be stored in  
i <- 0 #The starting offset value for the API request  
amount <- 1000 #amount of reviews  
numofentries <- 0  
  
while(numofentries < amount){  
   
 rawjson <- fromJSON(paste0(url,i)) #Getting JSON  
   
 message("Retrieving review ", numofentries, " out of ", amount, ".")  
   
 if (rawjson[["query\_summary"]][["num\_reviews"]] == 0) #Stopping the loop when there are no reviews left  
 {  
 message("Done")  
 break  
 }  
   
 pages[[i+1]] <- rawjson$reviews #Storing JSON in list  
 reviewdata\_df <- rbind\_pages(pages) #Turning list into a data frame  
 reviewdata\_df <- reviewdata\_df %>% distinct(recommendationid, .keep\_all = TRUE)#Removing any duplicate entries that might have snuck in  
 numofentries <- length(reviewdata\_df$recommendationid)  
   
 i <- i+100 #Moving to next set of reviews  
   
}

## Retrieving review 0 out of 1000.

## Retrieving review 100 out of 1000.

## Retrieving review 200 out of 1000.

## Retrieving review 297 out of 1000.

## Retrieving review 397 out of 1000.

## Retrieving review 497 out of 1000.

## Retrieving review 597 out of 1000.

## Retrieving review 696 out of 1000.

## Retrieving review 796 out of 1000.

## Retrieving review 896 out of 1000.

## Retrieving review 994 out of 1000.

message("Done")

## Done

message(numofentries, " reviews with the oldest being from ", anytime(tail(reviewdata\_df$timestamp\_created, 1)) )

## 1094 reviews with the oldest being from 2018-06-26 03:16:30

Here the in this while loop is where the JSON file that the Steam API returns is stored in a data frame. First before starting the loop an empty list needs to be made for the data to be stored in, after that the starting offset represented by the variable “i” is set and the amount of reviews wanting to be pulled is declared. More data is always better but this script can be quite intense on RAM. Depedning on how much RAM a person’s computer has it can allow for more or less reviews to be analyzed. The computer in this demostration had 8GB of RAM and after testing I found the most reviews that can be analyzed with 8GB of RAM without the script erroring is about 3000. For this example though 1000 reviews will be used. The number of reviews the loop has grabbed so far is also tracked to compare against the number of reviews requested so the loop will stop once the number of reviews wanted is reached. Sometimes the Steam API will return duplicate reviews in a different offset so to avoid this issue after each itteration of the reviews being added to the data frame duplicate entries are removed. After the loop is finished gathering the number of reviews requested it will display a message saying how many reviews it gatherd and the time stamp of the oldest review using the anytime package.

### Text Mining and Cleaning

#Organizing data  
  
negreview\_df <- reviewdata\_df[reviewdata\_df$voted\_up == FALSE,] #Data frame of negative reviews  
posreview\_df <- reviewdata\_df[reviewdata\_df$voted\_up == TRUE,] #Data frame of positive reviews  
#-----------------------------  
rm(pages, rawjson, reviewdata\_df) #Removing unused data to clear up memory

The newly created data frame of reviews is seperated into two data frames, one for the positive reviews and one for the negative. The lists and data frames used in the retrival process are removed from the enviroment to clear memory.

posreviews <- posreview\_df$review #Getting only the review part of the data frame  
negreviews <- negreview\_df$review  
posreviews <- iconv(posreviews, 'UTF-8', 'ASCII') #Removing emojis or other non-text characters  
negreviews <- iconv(negreviews, 'UTF-8', 'ASCII')

A new list is made of only the part of the data frame that contains reviews. The two lists are then converted into UTF-8 character types to remove emoji and other characters that Steam’s reviews allow but do not cooperate with some of text mining functions used in this script.

# Gathering list of misspelled words that will be cleaned and combined to be used later  
badpos <- hunspell(posreview\_df$review) #Getting list of misspelled words to remove  
badneg <- hunspell(negreview\_df$review)  
badpos <- unlist(badpos)  
badneg <- unlist(badneg)  
  
badpos <- iconv(badpos, 'UTF-8', 'ASCII')  
badneg <- iconv(badneg, 'UTF-8', 'ASCII')  
  
badneg <- removeNumbers(badneg)  
badpos <- removeNumbers(badpos)  
  
badpos <- removePunctuation(badpos)  
badneg <- removePunctuation(badneg)  
  
badpos <- removeWords(badpos, "")  
badneg <- removeWords(badneg, "")  
  
badpos <- unique(badpos)  
badneg <- unique(badneg)  
  
badwords <- c(badpos, badneg)  
  
badwords <- unique(badwords)  
# End of misspelled word cleaning

The hunspell package is used here to gather a list of words that are misspelled in each set of reviews. The list is then cleaned and combined into one list of misspelled words that apear in both sets of reviews. The words will be removed in the text cleaning process later.

posreviews <- VCorpus(VectorSource(posreviews)) #Creating Corpus of the reviews  
negreviews <- VCorpus(VectorSource(negreviews))  
  
clean\_corpus <- function(corpus, badwords){ #Function to clean the corpus   
 corpus <- tm\_map(corpus, stripWhitespace) #Getting rid of unwanted white space  
 corpus <- tm\_map(corpus, removePunctuation) #Getting rid of punctuation  
 corpus <- tm\_map(corpus, content\_transformer(tolower)) #making all characters lowercase  
 corpus <- tm\_map(corpus, removeWords, c("game", "review", "like", "can", "theres",   
 "just", "even", "games", "really", "1010", "dont",  
 "will", "ive", "one", "along", "doesnt", "well",   
 "much", "many", "also", "lets", "isnt", "good", "bad",   
 "not", "get", "great", "play", "playing", "played", "early",  
 "access", "b", "u", "®")) #List of unwanted words  
 corpus <- tm\_map(corpus, removeWords, stopwords("en")) #Removing stopwords  
 corpus <- tm\_map(corpus, removeNumbers)  
 corpus <- tm\_map(corpus, removeWords, gamename)  
   
 #Removing words that are misspelled  
 #Lists larger than 2000 can break the removewords function  
 #To fix the problem we do the removal in chunks of 2000 if the list is larger than 2000  
   
 if (length(badwords) > 2000){  
 s <- 1  
 e <- 2000  
   
 while (anyNA(badwords[s:e] == FALSE)){   
   
 corpus <- tm\_map(corpus, removeWords, tolower(badwords[s:e]))  
   
 if (anyNA(badwords[s:e]) == TRUE) break #If the function is calling values in the list that don't exist, break the loop  
   
 s <- s+2000 #Moving to next set of words  
 e <- e+2000  
   
 }  
   
 }  
   
 else{  
   
 corpus <- tm\_map(corpus, removeWords, tolower(badwords)) #If there is less than 2000 words remove them normally  
   
 }  
  
 return(corpus) #Returning the cleaned Corpus  
   
}  
  
posreviews <- clean\_corpus(posreviews, badwords) #Applying the corpus function to the review corpuses  
negreviews <- clean\_corpus(negreviews, badwords)

Here the reviews are converted into volatile corpuses so that the tm package can apply functions to them. A function to tidy and remove certain things from the review text is made. In this function first all extra whitespace and punctuation is removed as well as making all of the characters lowercase. Next a list of words that added no meaning to the result are removed as well as stop words, number characters, and the words in the name of the application or game. Next the list of misspelled words is used to remove those words from the reviews. Because of the way the regular expression for the removewords function is made it can’t handle all of words at once so the are removed in groups of 2000 if the list is larger than 2000. The function after it is done returns the tidy corpus.

### Creating the Visualization

find\_freq\_words <- function(corpus){ #Funcion to find the frequency of each word  
   
 BigramTokenizer <-  
 function(x)  
 unlist(lapply(ngrams(words(x), 2), paste, collapse = " "), use.names = FALSE)  
   
 dtm <- TermDocumentMatrix(corpus, control = list(tokenize = BigramTokenizer))  
   
 m <- as.matrix(dtm)  
 v <- sort(rowSums(m),decreasing=TRUE)  
 d <- data.frame(word = names(v),freq=v) #Creating a data frame of the terms and their frequencies  
 return(d) #Returning data frame  
   
}  
  
posreviewfreq <- find\_freq\_words(posreviews) #Applying frequency function   
negreviewfreq <- find\_freq\_words(negreviews)

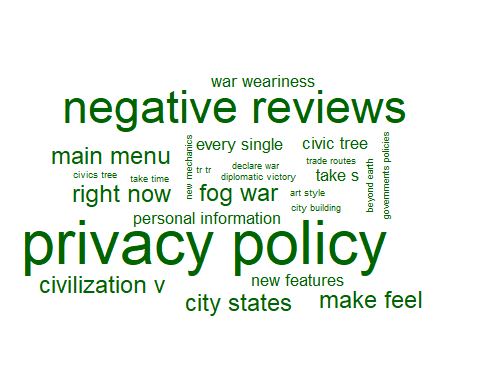
Another function is made to find the most frequent bigrams in each set of reviews. First the list of reviews is tokenized to find the bigrams then a term document matrix is made to find the frequency of each bigram in each set of reviews. The function returns a data frame that can be used for the visualization.

wc\_min\_freq <- 5 #Variable to adjust the minimum frequency required to be displayed on the word cloud  
wc\_max\_words <- 30 #Variable to adjust the maxium amount of words to be displayed on the word cloud  
wc\_shape <- "pentagon"  
  
create\_wordcloud <- function(wordfreq, wc\_min\_freq, wc\_max\_words, wccolor)  
{ #Function to create word cloud  
   
 set.seed(78)  
 ggwordcloud(wordfreq$word, wordfreq$freq, min.freq = wc\_min\_freq,   
 max.words = wc\_max\_words, shape = wc\_shape, color = wccolor)  
   
}  
  
poswordcloud <- create\_wordcloud(posreviewfreq, wc\_min\_freq, wc\_max\_words, "darkgreen") #Creating word cloud  
#poswordcloud #Displaying word cloud  
  
negwordcloud <- create\_wordcloud(negreviewfreq, wc\_min\_freq, wc\_max\_words, "darkred")  
#negwordcloud

For the visualization first the minimum word frequency is set. This is to prevent phrases with no meaning from showing up in the word cloud. In this example a phrase would have to have more than 5 occurrences to display in the word cloud. Next the maximum amount of phrases is set. This is to prevent a cluttered word cloud and in this example no more than 30 phrases will apear. The word cloud will display the most common phrases in various sizes, the size of the phrase is based on its frequency.

### Positive Reviews

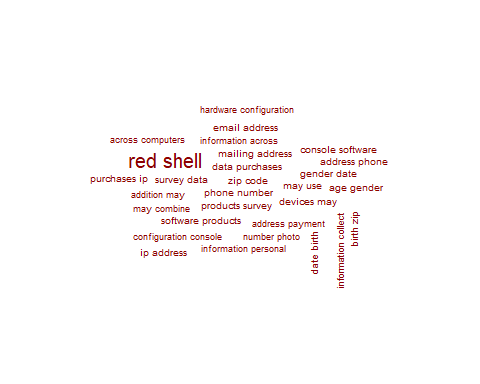
## One word could not fit on page. It has been removed.



## word freq  
## tech tree tech tree 14  
## privacy policy privacy policy 12  
## negative reviews negative reviews 10  
## city states city states 7  
## civilization v civilization v 7  
## fog war fog war 7

### Negative Reviews

## One word could not fit on page. It has been removed.



## word freq  
## personal information personal information 261  
## red shell red shell 130  
## ip address ip address 108  
## phone number phone number 108  
## zip code zip code 108  
## address phone address phone 107

Here are the results for each word cloud and their table of phrase frequencies. When looking at the negative reviews one can notice a lot of talk about personal information and the company Red Shell. The context behind this is that one of the developers for Civilization VI was apart of data controversy where the personal information of users was being shared with thrid party companies. This is only an example of a single game but based on what is shown here, if there is enough data it is possible to get meaningful information from these word clouds.