STAT495 (Advanced Data Analysis): Final Project

Tam Tran-The and Jordan Browning

December 19, 2016

INTRODUCTION

Our motivations

Online shopping is hugely important in today's world as consumers shift more and more away from brick and mortar stores and increasingly purchase products from online. While online shopping was once more commonly used to buy items hard to find in stores, it is now often used for more mundane purchases like groceries and drug store products. As more purchases are made online, companies must shift their focus from brick and mortar locations to their web presence, and thus they must strengthen their decision making in this sphere in order to edge out competitors.

Word of mouth has always played a major role with consumers. It influences their purchases because people often believe what others tell them about a product perhaps more than a clearly biased marketing campaign. Online reviews are a sort of word of mouth in the online shopping sphere. While businesses have historically had to use surveys in order to determine customer opinion, the great amount of information available online provides a much cheaper and faster way to get public opinion. By mining product reviews, businesses can get a summary of opinions on a product and use this to inform a variety of business decisions.

Description of problem

Amazon currently sells over 480 million products on their website. Most of their popular products can have a thousand to more than ten thousand reviews overall. So how can a company like Amazon with millions of products that are constantly receiving thousands of reviews make any sense of these data? That's when sentiment analysis, or opinion mining, comes into place. By extracting humans' attitudes or opinions from these reviews, sellers like Amazon can gain invaluable insight into consumer preferences and concerns, which then help them make informed marketing decisions or even create entirely new products and brands.

Description of project

Our project sought to explore how sentiment analysis is used to summarize the opinions and attitudes of consumers. We want to split reviews into their polarity, either positive, negative, or neutral, as well their dominant emotion, like sad or fearful. In doing this, we would show how a business might use this tool to gain insight into consumer preferences and concerns.

DATA

While it is certainly possible to scrape an Amazon page to grab reviews, we chose to use a dataset already compiled from Amazon. However, we did include a separate document that shows the scraping process included in the final report folder. These are separate because the final report was made on the R server, while the scraper required to be done locally as it needed packages that the server did not have the tools to compile.

Description of dataset

The dataset we used was created by Julian McCauley, a UCSD professor and include Amazon review information between May 1996 and July 2014. It is available at http://jmcauley.ucsd.edu/data/amazon/. While the entirety of his data is nearly 143 million observations, we chose to only focus on three categories: clothing, shoes, and jewelry; health and personal care; and sports and outdoors. The analyses we did could be applied to any of the categories since we arbitrarily chose these categories.

The following variables were included in the dataset:

- **asin**: ID of the product; class: character
- **reviewText**: content of a user review; class: character
- **helpful**: score other users give for a specific review; class: numeric
- **overall**: score the reviewer gives for the product he/she reviews; class: numeric
- **reviewerID**: ID of reviewer; class: character
- **reviewerName**: name of reviewer; class: character
- **reviewTime**: MDY format for when the review is posted; class: date

DATA WRANGLING

The data wrangling process involved taking the data files from McCauley's site and pulling out the text for each review so that we could perform sentiment analyses. The files were in JSON format, so below we show how we streamed in the data. We did this with the other two categories as well.

```
library(rjson)
library(jsonlite)

clothing_file <- "Clothing_Shoes_and_Jewelry_5.json"
clothing <- stream_in(file(clothing_file))
```

We then counted up how many reviews there were by product. Because we wanted the top 10 or so products per category by review count, we filtered the clothing category by only the products that had more than 197 product reviews.

```
library(dplyr)
library(mosaic)

# count number of reviews for each product
num_clothing <- clothing %>%
  group_by(asin) %>%
  summarize(count = n()) %>%
  filter(count >= 197) %>%
  arrange(desc(count))
```

Because the dataset did not include product names, we had to manually add these in using the ASIN number to search on Amazon.

```
# add product names -- issues with 2 and 4
name <- c("Owl Pendant Long Chain Necklace", "Printed Brushed Leggings", "Long Sleeve Skull T Shirts",
    "Skechers Performance Women's Go Walk Slip-On Walking Shoe", "Champion Women's Unwind Sport Slip-On
    "Leegoal Retro Peacock Crystal Necklace Pendant Jewelry", "Glamorise Women's No-Bounce Full-Support
    "SE JT6221 16-Piece Watch Repair Tool Kit", "Levi's Men's 501 Original-Fit Jean",
    "ExOfficio Men's Give-N-Go Boxer Brief", "Leegoal Vintage Steampunk Nautical Style Antiqued Bronze (

num_clothing <- cbind(num_clothing, names)
```

We then joined the dataframe with the counts with the original dataframe and had to filter again to get the top 10 or so products. This allowed us to include count totals in the original dataframe.

```
clothing <- left_join(clothing, num_clothing, by = "asin")

# we have to filter again because we merged with the whole dataset, so we
# had even those products that won't be included
clothing <- filter(clothing, count >= 197)
```

We then cleaned up the review text for each product and made it usable for our function.

```
load("clothing.Rda")
reviews <- clothing$reviewText
review_list <- lapply(reviews, function(x) iconv(x, "latin1", "ASCII", sub=""))
review_list <- lapply(reviews, function(x) gsub("htt.*",' ',x))
reviews <- unlist(reviews)
clothing$cleanText <- reviews
```

DATA ANALYSIS/ METHODS

Sentiment analysis

Sentiment analysis refers to the use of natural language processing (NLP) and text mining to identify and extract subjective information from the source materials. One of the common tasks in sentiment analysis is classifying the polarity of a text, in other words, determining whether the expressed opinion in the text is positive, negative, or neutral. Beyond polarity, sentiment analysis also looks at emotional states, such as “angry”, “sad”, and “happy”, of the given document.

Our project is based on a package named “sentiment” which was created by Timothy Jurka during his time as a PhD student at UC Davis. Jurka is currently managing the data team that builds the LinkedIn News Feed and has been known in the computer science community for his contribution to text classification algorithms. Since this package was removed from CRAN a couple months ago, we tried to replicate the codes he used to write the package and applied them to our own data set.

There are two main functions we employed to extract subjective information from any given text: emotions classification and polarity classification. Both of them can be performed based on either a Naives Bayes classifier or a simple voting system that is trained on a particular lexicon.

Prepare the text for sentiment analysis

The intuition behind Naive Bayes is that it relies on a very simple representation of the text. In other words, it uses a subset of words instead of the whole document, and the order of words does not matter in this case. That's why we needed to clean up and transformed the text input into a document term matrix (DTM) so that we would be able to perform findFreqTerms() function on the object to pick out important words.

The code chunk below serves the purpose of tidying up the text by performing multiple text mining operations on a corpus and then transform it into a DTM object.

```
create_matrix <- function(textColumns, language="english", minDocFreq=1, minWordLength=3,
                          removeNumbers=TRUE, removePunctuation=TRUE, removeSparseTerms=0,
                          removeStopwords=TRUE, stemWords=FALSE, stripWhitespace=TRUE,
                          toLower=TRUE, weighting=weightTf) {

      stem_words <- function(x) {
        # split the whole text into single words
        split <- strsplit(x," ")
        #return the common root/ stem of these words in english
        return(wordStem(split[[1]], language=language))
    }

  #list of operations to tidy the text
    control <- list(language=language, tolower=toLower, removeNumbers=removeNumbers,
                    removePunctuation=removePunctuation, stripWhitespace=stripWhitespace,
                    minWordLength=minWordLength, stopwords=removeStopwords,
                    minDocFreq=minDocFreq, weighting=weighting)

      #if the argument stemWords is true,
      #add the list of stem_words to the list of operations
    if (stemWords == TRUE) control <- append(control, list(stemming=stem_words), after=6)

      #transform text into character matrix
    trainingColumn <- apply(as.matrix(textColumns), 1, paste, collapse=" ")
    trainingColumn <- sapply(as.vector(trainingColumn, mode="character"),
                             iconv, to="UTF8", sub="byte")

  #transform the given text into a corpus
  #and perform all the operations that are defined in the control list
    corpus <- Corpus(VectorSource(trainingColumn),readerControl=list(language=language))
    #transform corpus into a DTM matrix object
    matrix <- DocumentTermMatrix(corpus, control=control);
      #remove sparse terms (terms whose sparsity is above a certain threshold)
    if (removeSparseTerms > 0) matrix <- removeSparseTerms(matrix,removeSparseTerms)

    #since we probably just remove a large object,
    #it's safe call garbage collection to prevent R from returning memory to the operating system
    gc()
    #return the resultant matrix
    return(matrix)
}
```

Emotions classification

With the bag of words from the DTM object, we checked which word from the object actually appeared in the emotions lexicon. The emotions lexicon we used in this project was compiled by Carlo Strapparava and Alessandro Valitutti, two European computer scientists specialized in natural language processing. The lexicon includes all kinds of opinion-bearing words or sentiment phrases of idioms that represent five basic emotions anger, disgust, fear, joy, sadness, and surprise, and the categories to which they belong.

As for the algorithm that serves as a voting system (verbose=TRUE), score for each category is simply the number of words of that category which appear in the input text. In order words, for every match between a word in the document and a word in the lexicon, total score for that category is updated with +1. Category with the highest number of words showing up in the text is the predicted output.

As for the algorithm that is based on Naive Bayes classifier, we would calculate score for each category using Bayes theorem.

- Let D be the list of words from the input document where each word is represented by d_i and C be the set of classes used in the classification (anger, disgust, joy, fear, sadness, surprise) where each class is represented by c_i.
- P(C|D) is the conditional probability of class C given document D. P(C) is the prior probability of class C. P(D) is the prior probability of document D.
- We want to find P(C|D), which is equivalent to $P(C = c_i|D) = \frac{P(D|C=C_i)*P(C=c_i)}{P(D)}$
- Since we use the same document, the marginal probability P(D) can be disregarded and the equation becomes: $P(C = c_i|D) = P(D|C = C_i) * P(C = c_i) = P(C = c_i) * P(d_1|C = C_i) * P(d_2|C = C_i) * ... * P(d_n|C = C_i)$. This means that to find in which class we should classify a new document, we must estimate the product of the probability of each word of the document given a particular class, multiplied by the probability of the particular class. After calculating the above for all the classes of set C, we select the one with the highest probability.
- We know that: $P(C = c_i) = \frac{totalnumberofwordsofclassC}{totalnumberofwordsinlexicon}$ and $P(d_i|C = c_i) = \frac{1}{totalnumberofwordsofclassCinthelexicon}$
- Specifically, we have this table:

```
emotion_class <- c("Anger", "Disgust", "Fear", "Joy", "Sadness", "Surprise", "Total")
num_words <- c(335, 70, 195, 553, 274, 95, 1542)
table1 <- data.frame(emotion_class, num_words)
class_prob <- c()
for (i in 1:length(emotion_class)) {
  class_prob[i] <- num_words[i]/num_words[7]
}
class_prob[7] <- NA
table1 <- table1 %>% mutate(class_prob=class_prob)
cond_prob <- c()
for (i in 1:length(emotion_class)) {
  cond_prob[i] <- 1/num_words[i]
}
cond_prob[7] <- NA
table1 <- table1 %>% mutate(cond_prob=cond_prob)
options(xtable.comment = FALSE)
xtable(table1)
```

	emotion_class	num_words	class_prob	cond_prob
1	Anger	335.00	0.22	0.00
2	Disgust	70.00	0.05	0.01
3	Fear	195.00	0.13	0.01
4	Joy	553.00	0.36	0.00
5	Sadness	274.00	0.18	0.00
6	Surprise	95.00	0.06	0.01
7	Total	1542.00		

As we can see, some of these probabilities are pretty small, and multiplying them together may even lead the product to be smaller. In other words, since computers can handle numbers with specific decimal point accuracy, a number so small will not be able to fit in memory and thus will be rounded to zero, which then renders our analysis useless. To avoid this, instead of maximizing the product of the probabilities, we will maximize the sum of their logarithms: $log(P(C = c_i)) + log(P(d_1|C = C_i)) + log(P(d_2|C = c_i)) + ... + log(P(d_3|C = c_i))$

We now have all the theoretical knowledge to follow through the code chunk below.

```
classify_emotion <- function(textColumns, algorithm="bayes", prior=1.0, verbose=FALSE,...) {
    #tidy up input text and transform it into a DTM
  matrix <- create_matrix(textColumns,...)
  #import the emotions lexicon
    lexicon <- read.csv("emotions.csv", header=FALSE)

    #create a list of number of words that belong to each category
    counts <- list(anger=length(which(lexicon[, 2]=="anger")),
                   disgust=length(which(lexicon[, 2]=="disgust")),
                   fear=length(which(lexicon[, 2]=="fear")),
                   joy=length(which(lexicon[, 2]=="joy")),
                   sadness=length(which(lexicon[, 2]=="sadness")),
                   surprise=length(which(lexicon[, 2]=="surprise")), total=nrow(lexicon))
    #initialize an empty vector to store overall results
    documents <- c()

    #run a for loop through the text DTM
    for (i in 1:nrow(matrix)) {

      #if chose the voting system algorithm, simply print out the document
        if (verbose) print(paste("DOCUMENT",i))

      #set initial score for each category to 0
        scores <- list(anger=0, disgust=0, fear=0, joy=0, sadness=0, surprise=0)
        #select the text element from the matrix
        doc <- matrix[i,]
        #find words with the highest tf.idf scores (words with high importance to the document)
        words <- findFreqTerms(doc,lowfreq=1)

        #set number of words for each emotion to 0
        numanger <- 0
        numdisgust <- 0
        numfear <- 0
        numjoy <- 0
        numsad <- 0
```

```
        numsurprise <- 0

        #scan through all words in the text
        for (word in words) {
                #scan through each category/emotion
            for (key in names(scores)) {
                #choose the list of words that belong to one particular category
                emotions <- lexicon[which(lexicon[,2]==key),]
                #match word from the text with that list of words from the lexicon
                index <- pmatch(word,emotions[,1],nomatch=0)
                #if index > 0, which means there is a match
                if (index > 0) {
                    #save the word and its category into entry
                    entry <- emotions[index,]

                    #extract category
                    category <- as.character(entry[[2]])
                    #find the number of words of that category in the lexicon
                    count <- counts[[category]]

                    #update number of word of each category
                    if (category=="anger") numanger <- numanger+1
                    if (category=="disgust") numdisgust <- numdisgust+1
                    if (category=="fear") numfear <- numfear+1
                    if (category=="joy") numjoy <- numjoy+1
                    if (category=="sadness") numsad <- numsad+1
                    if (category=="surprise") numsurprise <- numsurprise+1

                    #initialize score to be 1
                    score <- 1.0
                    #if use bayes, score for each word in the class is the logarithm of P(d_i|C=c_i)
                    if (algorithm=="bayes") score <- abs(log(score/count))

                    #if use voting system, score for each word in the class is just 1.0
                    if (verbose) {
                        print(paste("WORD:", word,"CAT:", category, "SCORE:", score))
                    }
                    #add up these scores
                    scores[[category]] <- scores[[category]]+score
                }
            }
        }

            #if use bayes, score of the class is the logarithm of P(C=c_i)
        if (algorithm=="bayes") {
            for (key in names(scores)) {
                #count number of words of a specific class in the lexicon
                count <- counts[[key]]
                #count total number of words in the lexicon
                total <- counts[["total"]]
                #calculate class probability and take logarithm
                score <- abs(log(count/total))
                #update total score
```

```
                scores[[key]] <- scores[[key]]+score
            }
        } else {
          #otherwise, if use voting system, score of the class is basically 0
            for (key in names(scores)) {
                scores[[key]] <- scores[[key]]+0.000001
            }
        }

            #the predicted output for the text is the class with highest score
        best_fit <- names(scores)[which.max(unlist(scores))]
        #if the predicted class is disgust
        #and the difference between the class' score and 3.09234 is smaller than 0.01,
        #the output is reclassified to NA
        if (best_fit == "disgust" && as.numeric(unlist(scores[2]))-3.09234 < .01) best_fit <- NA

    #combine the summary statistics together
        documents <- rbind(documents,c(numanger, numdisgust, numfear, numjoy, numsad, numsurprise,
                                       scores$anger, scores$disgust, scores$fear, scores$joy,
                                       scores$sadness, scores$surprise, best_fit))
    }

    #rename columns
    colnames(documents) <- c("# ANGER", "# DISGUST", "# FEAR", "# JOY", "# SADNESS",
                             "# SURPRISE", "ANGER SCORE", "DISGUST SCORE", "FEAR SCORE",
                             "JOY SCORE", "SADNESS SCORE", "SURPRISE SCORE", "BEST_FIT")
    #return result
    return(documents)
}
```

Polarity classification

The way polarity classification was set up is very similar to that of emotions classification. The main difference is that this classification was trained on a subjectivity lexicon, instead of a emotions lexicon. The subjectivity lexicon we used for this function was compiled by Janyce Wiebe, a Computer Science Professor at University of Pittsburgh who is specialized in artificial intelligence and natural language processing (NLP), especially subjectivity analysis. This lexicon includes words that are splitted into 2 polars: positive and negative, and then in each polar, they are splitted further into strong subjective and weak subjective words. For example, `acceptable` is a weak positive and `admirable` is a strong positive, whereas `anomaly` is a weak negative and `absurd` is a strong negative.

As for the algorithm that serves as a voting system (verbose=TRUE), score for each category basically represents the number of words in that category which appear in the input text. However, each weak subjective word carries the weight of 1, while each strong subjective word carries the weight of 0.5. Thus, for every match between a word in the document and a word in the lexicon, total score for that category is updated with either +0.5 if match is weak subjective or +1 if match is strong subjective.

Using the same idea of Bayes theorem, we try to find: $P(C = c_i|D) = P(D|C = C_i) * P(C = c_i) = P(C = c_i) * P(d_1|C = C_i) * P(d_2|C = C_i) * ... * P(d_n|C = C_i)$.

- We know that: $P(C = c_i) = \frac{totalnumberofwordsofclassC}{totalnumberofwordsinlexicon}$.

- As for $P(d_i|C = c_i)$, to take subjectivity annotations into account, we consider a weak subjective as one word and a strong subjective as half a word. In mathematical notation: $P(d_i|C = c_i) = \frac{1}{totalnumberofwordsofclassCinthelexicon}$ for weak subjective and $P(d_i|C = c_i) = \frac{0.5}{totalnumberofwordsofclassCinthelexicon}$ for strong subjective.
- We then again have the following table:

```
polarity_class <- c("Postive", "Negative", "Total")
num_words <- c(2324, 4175, 6518)
table2 <- data.frame(polarity_class, num_words)
class_prob <- c()
for (i in 1:length(polarity_class)) {
  class_prob[i] <- num_words[i]/num_words[3]
}
class_prob[3] <- NA
table2 <- table2 %>% mutate(class_prob=class_prob)
weak_prob <- c()
for (i in 1:length(polarity_class)) {
  weak_prob[i] <- 1/num_words[i]
}
weak_prob[3] <- NA
table2 <- table2 %>% mutate(weak_prob=weak_prob)
strong_prob <- c()
for (i in 1:length(polarity_class)) {
  strong_prob[i] <- 0.5/num_words[i]
}
strong_prob[3] <- NA
table2 <- table2 %>% mutate(strong_prob=strong_prob)
options(xtable.comment = FALSE)
xtable(table2)
```

	polarity_class	num_words	class_prob	weak_prob	strong_prob
1	Postive	2324.00	0.36	0.00	0.00
2	Negative	4175.00	0.64	0.00	0.00
3	Total	6518.00			

Again, due to float point underflow, our probabilities are rounded to zero. To avoid this, we use the same trick: instead of maximizing the product of the probabilities, we will maximize the sum of their logarithms: $log(P(C = c_i))+log(P(d_1|C = C_i))+log(P(d_2|C = c_i))+...+log(P(d_3|C = c_i))$.

```
classify_polarity <- function(textColumns, algorithm="bayes", pstrong=0.5, pweak=1.0,
                              verbose=FALSE,...) {
    #tidy up input text and transform it into a DTM
  matrix <- create_matrix(textColumns,...)
  #import the subjectivity lexicon
    lexicon <- read.csv("subjectivity.csv",header=FALSE)

    #create a list of number of words that belong to each category
    counts <- list(positive=length(which(lexicon[,3]=="positive")),
                   negative=length(which(lexicon[,3]=="negative")), total=nrow(lexicon))

    #initialize an empty vector to store overall results
    documents <- c()
```

```
#run a for loop through the text DTM
for (i in 1:nrow(matrix)) {

  #if chose the voting system algorithm, simply print out the document
    if (verbose) print(paste("DOCUMENT",i))

  #set initial score for each category to 0
    scores <- list(positive=0, negative=0)
    #select the text element from the matrix
    doc <- matrix[i,]
    #find words with the highest tf.idf scores (words with high importance to the document)
    words <- findFreqTerms(doc, lowfreq=1)

    #set number of words for each polar to 0
    numpos <- 0
    numneg <- 0

    #scan through all words in the text
    for (word in words) {
      #match word from the text with the whole list of words in the lexicon
        index <- pmatch(word,lexicon[,1], nomatch=0)
        #if there's a match, which means the word in the text appears in the lexicon
        if (index > 0) {
          #save the index of the match
            entry <- lexicon[index,]
            #extract its subjectivity (strong or weak)
            polarity <- as.character(entry[[2]])
            #extract its polar (negative or positive)
            category <- as.character(entry[[3]])
            #find number of words of this polar in the lexicon
            count <- counts[[category]]

    #update number of word of each polar
            if (category=="positive") numpos <- numpos+1
            if (category=="negative") numneg <- numneg+1

            #weak subjective word has weight of 1
            score <- pweak
                    #strong subjective word has weight of 0.5
            if (polarity == "strongsubj") score <- pstrong
            #if use bayes, score for each word in the class is the logarithm of P(d_i|C=c_i)
            if (algorithm=="bayes") score <- abs(log(score/count))

            #if use voting system, score for each word in the class is just 1 for weak subjective
            #and 0.5 for strong subjective
            if (verbose) {
                print(paste("WORD:",word,"CAT:",category,"POL:",polarity,"SCORE:",score))
            }

    #add up these scores
            scores[[category]] <- scores[[category]]+score
        }
    }
```

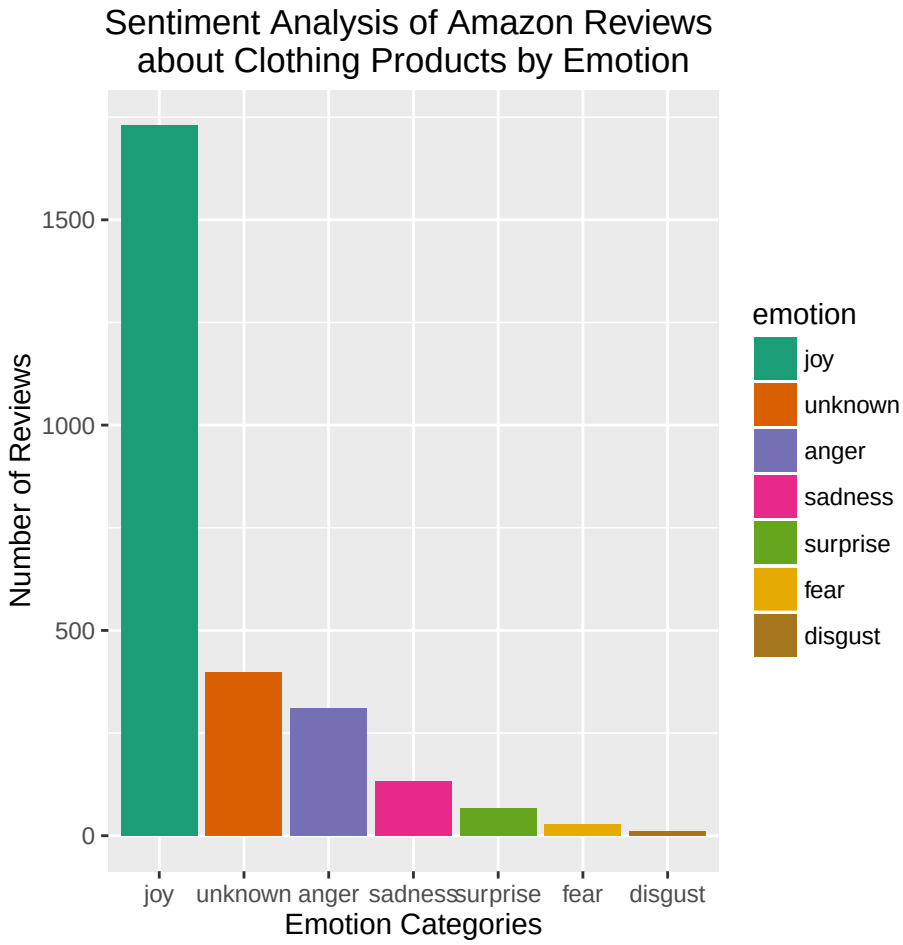
```
    #if use bayes, score of the class is the logarithm of P(C=c_i)
        if (algorithm=="bayes") {
            for (key in names(scores)) {
              #count number of words of a specific class in the lexicon
                count <- counts[[key]]
                #count total number of words in the lexicon
                total <- counts[["total"]]
                #calculate class probability and take logarithm
                score <- abs(log(count/total))
                #update total score
                scores[[key]] <- scores[[key]]+score
            }
        } else {
          #otherwise, if use voting system, score of the class is basically 0
            for (key in names(scores)) {
                scores[[key]] <- scores[[key]]+0.000001
            }
        }
            #the predicted output for the text is the class with highest score
        best_fit <- names(scores)[which.max(unlist(scores))]
        #calculate ratio between score for positive and score for negative
        ratio <- as.integer(abs(scores$positive/scores$negative))
        #if ratio=1, predicted output is neutral
        if (ratio==1) best_fit <- "neutral"
    #combine summary statistics
        documents <- rbind(documents, c(numpos, numneg, scores$positive, scores$negative,
                                        abs(scores$positive/scores$negative), best_fit))
        if (verbose) {
            print(paste("POS:", scores$positive, "NEG:",scores$negative, "RATIO:",
                        abs(scores$positive/scores$negative)))
            cat("\n")
        }
    }

    #rename columns
    colnames(documents) <- c("#POS","#NEG","POS SCORE","NEG SCORE","POS/NEG","BEST_FIT")
    #return results
    return(documents)
}
```

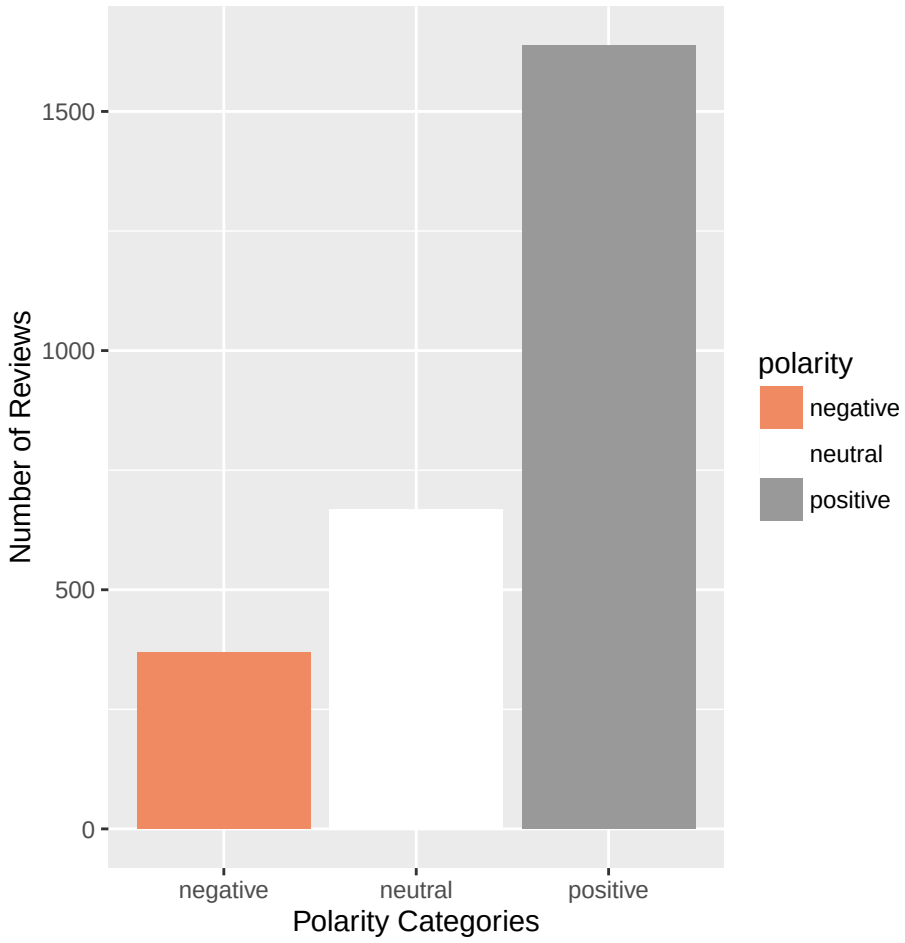
FINDINGS

Below we apply the sentiment analysis algorithms, using Naive Bayes, to the clothing data:

```
#classify emotion
class_emo <- classify_emotion(clothing$cleanText, algorithm="bayes")
# get emotion best fit
emotion <- class_emo[,13]
# substitute NA's by "unknown"
emotion[is.na(emotion)] <- "unknown"
```

```r
#classify polarity
class_pol <- classify_polarity(clothing$cleanText, algorithm="bayes")
# get polarity best fit
polarity <- class_pol[,6]

#create data frame to store results
general1 <- data.frame(text=clothing$cleanText, emotion=emotion,
                       polarity=polarity, rating = clothing$overall, stringsAsFactors=FALSE)
general <- within(general1,
  emotion <- factor(emotion, levels=names(sort(table(emotion), decreasing=TRUE))))
```

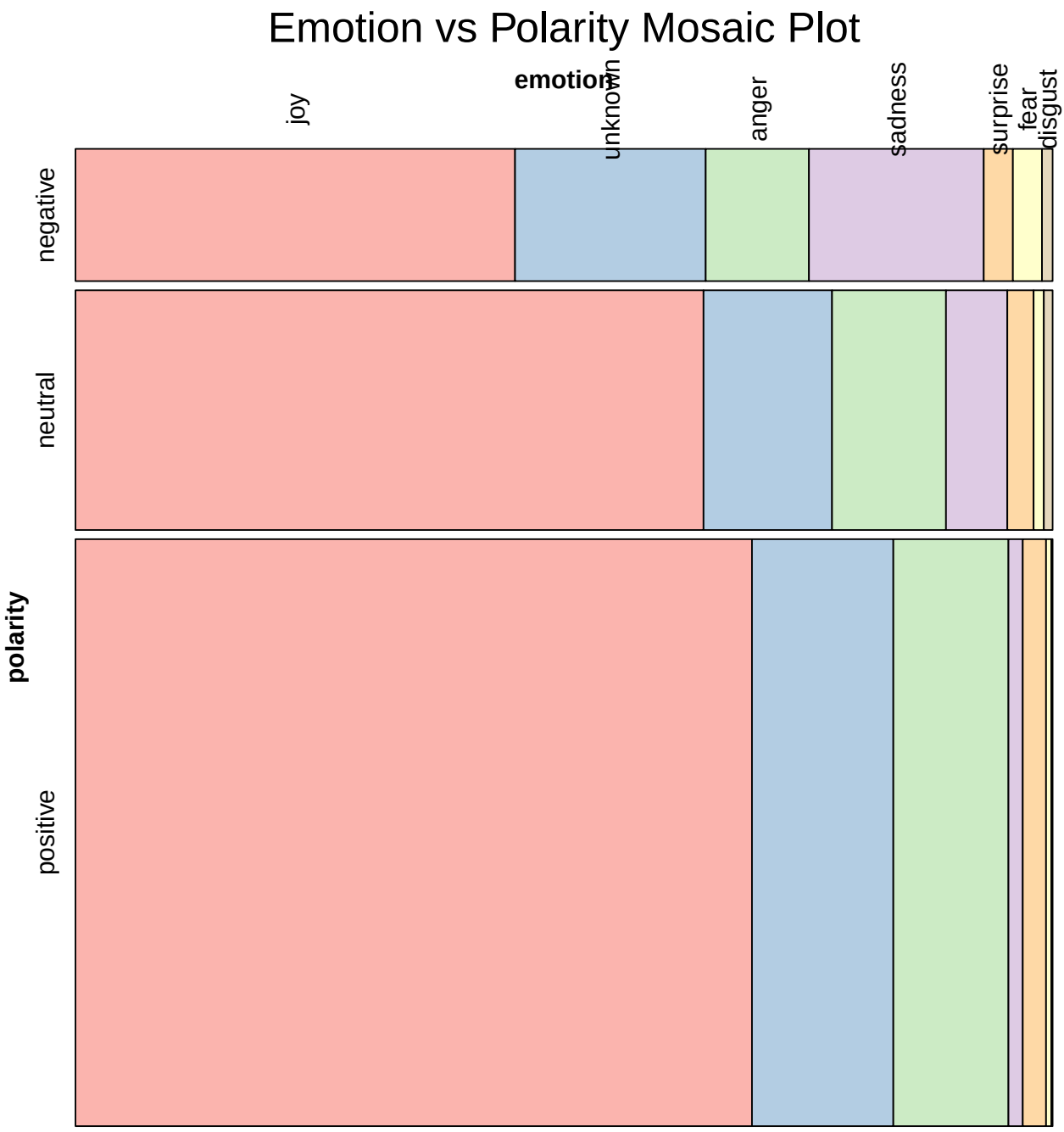




We can also visualize the frequencies of words stratified by emotional categories across reviews.

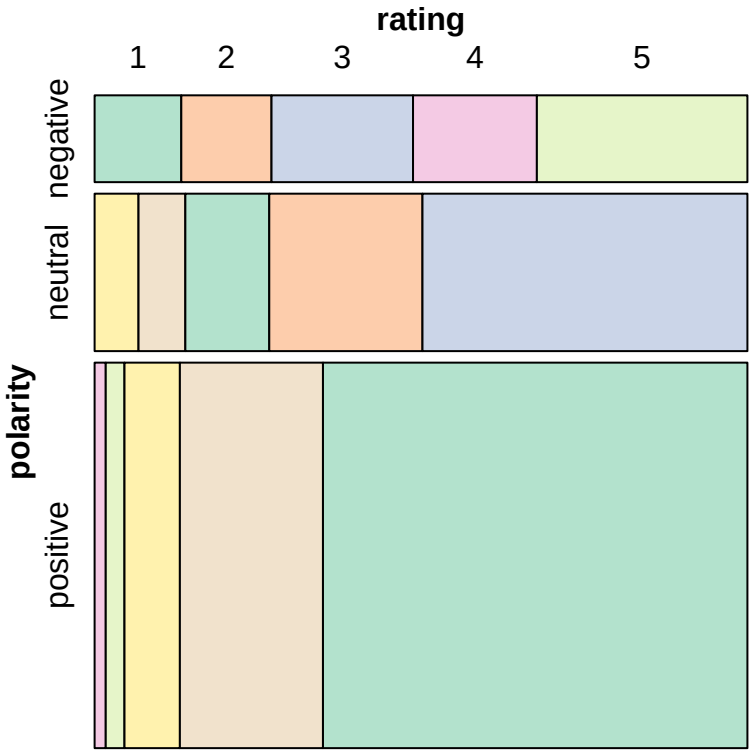
```
# separating text by emotion
emos <- levels(factor(general$emotion))
nemo <- length(emos)
emo_docs <- rep("", nemo)
for (i in 1:nemo)
{
   tmp <- general1$text[general$emotion == emos[i]]
   emo_docs[i] <- paste(tmp, collapse=" ")
}

# remove stopwords
emo_docs <- removeWords(emo_docs, stopwords("english"))
# create corpus
corpus <- Corpus(VectorSource(emo_docs))
tdm <- TermDocumentMatrix(corpus)
tdm <- as.matrix(tdm)
colnames(tdm) <- emos

# comparison word cloud
comparison.cloud(tdm, colors=brewer.pal(nemo, "Dark2"),
   scale=c(3,.5), random.order=FALSE, title.size=1.5)
```

To find the overlap between emotions and polarity classification, we create a mosaic plot showing the cross-sectional distribution of reviews in various emotional states through negative, neutral and positive categories.



The emotion joy appears to be most dominant no matter in which polar the review is classified. Positive reviews take up a larger part than negative ones. In addition, as we expected, the positive class has more joyful reviews whereas the negative class includes more sad and angry reviews. Surprise, disgust and fear show up very little compared to other emotions.

Also using mosaic plot, we visualize the relationship between ratings and polarity classification.



5-star reviews take up the largest chunk of not only positive but also negative class. In fact, about 55% of our reviews are rated 5-stars. We think this is the case partly because we only include reviews of popular products in our data set and these products are usually well-received by the customers.

END PRODUCT: SHINY APP

The final product of our work was put into a Shiny app. The code for this app is included in the appendix of this report. The app is available at: https://r.amherst.edu/apps/m5ttranthe/Amazon%20sentiment%20analysis/

This app allows you to select one of three categores – clothing, health, or sports. You can then choose one of the top ten or so reviewed products from those categories. The app then has three tabs. The first tab lets you see a bar graph describing that product's reviews broken into polarity and emotions. The second tab has a word cloud which lets you get a sense of the most prominent words used, separated into their categories of emotions. The third tab is not contingent on what product is chosen and lets a user put in a review of their own, which is then analyzed by the sentiment analysis algorithms and a score is shown for polarity and emotion.

LIMITATIONS AND FUTURE WORK

If we had more time, we would have made the Shiny app take in any product key, scrape its reviews, and perform sentiment analysis on them. This is certainly doable by using the scraper in

the other part of the final report, however this would be a highly time-intensive process. Because we precomputed all of the emotions and polarities shown in the Shiny app to make it run faster, waiting for a product's reviews to be scraped and then analyzed would be taxing on the Shiny. If one is willing to be patient, though, it would create a highly interesting visual that could help them understand the way reviewers feel about any product of their choice.

In addition, we would have tried to explore different methods of sentiment analysis to see which packages and methods seemed to be the most accurate. There were certainly some misclassifications with the algorithm we used, and while some of them were the result of simply the reviewer using too many positive words in a negative review or vice versa, some of the errors were more nebulous, so it would be useful to see if there were some packages better at understanding the nuance of reviews.

Sentiment analysis is a hugely important tool to utilize all the information available on the internet. Customers commonly conceive as review systems as something for them to help inform their decisions. However, while it may not be abundantly clear how businesses can use review systems other than to build reputations, sentiment analysis provides a way for them to extract information and make better decisions. Examples of ways in which it can be used are to see what product features are associated with negativity and improve those features, see what is being associated with positivity about their products and stress that in marketing campaigns, and to detect spam reviews purposely trying to demote/promote a product. When Marc Maier came to speak to the class about MassMutual, he described a data science project in which they were trying to compare responses of MassMutual Twitter followers to the company's tweets with responses to its competitors' tweets to figure out what content gets the best response from followers. Sentiment analysis is useful here in summarizing and understanding those customer responses.

The most major finding of our project is that sentiment analysis allows business, specifically Amazon in this case, to condense reviews quickly, figure out its consumers' opinions about its products, and determine if the plurality of reviews for a product is positive or negative as well as the dominant emotion behind them.

SHINY APP APPENDIX

```
library(rjson)
library(jsonlite)
library(dplyr)
library(mosaic)
library(shinythemes)
library(tm)
library(SnowballC)
library(wordcloud)
library(cowplot)
library(ggplot2)

# we display these for the user to choose a category on first drop-down
categories <- c("Clothing, Shoes, and Jewelry", "Health and Personal Care",
                "Sports and Outdoors")

# load dataframes with top ten reviews from each category
# files written instead of streamed from json to save time
```

```
setwd("/home/mhc/class17/m5ttranthe/Git/STAT495-Group1/Data")
load("clothing.Rda")
load("health.Rda")
load("sports.Rda")


# SENTIMENT PACAKGE FUNCTIONS BY TIMOTHY JURKA

create_matrix <- function(textColumns, language="english", minDocFreq=1,
                          minWordLength=3, removeNumbers=TRUE, removePunctuation=TRUE,
                          removeSparseTerms=0, removeStopwords=TRUE, stemWords=FALSE,
                          stripWhitespace=TRUE, toLower=TRUE, weighting=weightTf) {

  stem_words <- function(x) {
    split <- strsplit(x," ")
    return(wordStem(split[[1]], language=language))
  }

  control <- list(language=language, tolower=toLower, removeNumbers=removeNumbers,
                  removePunctuation=removePunctuation,
                  stripWhitespace=stripWhitespace, minWordLength=minWordLength,
                  stopwords=removeStopwords,
                  minDocFreq=minDocFreq,weighting=weighting)

  if (stemWords == TRUE) control <- append(control,
                                           list(stemming=stem_words), after=6)

  trainingColumn <- apply(as.matrix(textColumns), 1, paste, collapse=" ")
  trainingColumn <- sapply(as.vector(trainingColumn, mode="character"), iconv,
                           to="UTF8", sub="byte")

  corpus <- Corpus(VectorSource(trainingColumn),
                   readerControl=list(language=language))
  matrix <- DocumentTermMatrix(corpus,control=control);
  if (removeSparseTerms > 0) matrix <- removeSparseTerms(matrix, removeSparseTerms)

  gc()
  return(matrix)
}

classify_emotion <- function(textColumns, algorithm="bayes", prior=1.0,
                             verbose=FALSE,...) {
  matrix <- create_matrix(textColumns,...)
  lexicon <- read.csv("emotions.csv", header=FALSE)

  counts <- list(anger=length(which(lexicon[,2]=="anger")),
                 disgust=length(which(lexicon[,2]=="disgust")),
                 fear=length(which(lexicon[,2]=="fear")),
                 joy=length(which(lexicon[,2]=="joy")),
                 sadness=length(which(lexicon[,2]=="sadness")),
                 surprise=length(which(lexicon[,2]=="surprise")),
                 total=nrow(lexicon))
  documents <- c()
```

```
for (i in 1:nrow(matrix)) {
  if (verbose) print(paste("DOCUMENT", i))
  scores <- list(anger=0, disgust=0, fear=0, joy=0, sadness=0, surprise=0)
  doc <- matrix[i,]
  words <- findFreqTerms(doc, lowfreq=1)

  numanger <- 0
  numdisgust <- 0
  numfear <- 0
  numjoy <- 0
  numsad <- 0
  numsurprise <- 0

  for (word in words) {
    for (key in names(scores)) {
      emotions <- lexicon[which(lexicon[,2]==key),]
      index <- pmatch(word, emotions[,1], nomatch=0)
      if (index > 0) {
        entry <- emotions[index,]

        category <- as.character(entry[[2]])
        count <- counts[[category]]

        if (category=="anger") numanger <- numanger+1
        if (category=="disgust") numdisgust <- numdisgust+1
        if (category=="fear") numfear <- numfear+1
        if (category=="joy") numjoy <- numjoy+1
        if (category=="sadness") numsad <- numsad+1
        if (category=="surprise") numsurprise <- numsurprise+1

        score <- 1.0
        if (algorithm=="bayes") score <- abs(log(score*prior/count))

        if (verbose) {
          print(paste("WORD:",word,"CAT:",category,"SCORE:",score))
        }

        scores[[category]] <- scores[[category]]+score
      }
    }
  }

  if (algorithm=="bayes") {
    for (key in names(scores)) {
      count <- counts[[key]]
      total <- counts[["total"]]
      score <- abs(log(count/total))
      scores[[key]] <- scores[[key]]+score
    }
  } else {
    for (key in names(scores)) {
      scores[[key]] <- scores[[key]]+0.000001
    }
```

```
    }

    best_fit <- names(scores)[which.max(unlist(scores))]
    if (best_fit=="disgust" && as.numeric(unlist(scores[2]))-3.09234 < .01) best_fit <- NA
    documents <- rbind(documents, c(numanger, numdisgust, numfear, numjoy, numsad,
                                    numsurprise, scores$anger, scores$disgust,
                                    scores$fear, scores$joy, scores$sadness,
                                    scores$surprise, best_fit))
  }

  colnames(documents) <- c("# ANGER", "# DISGUST", "# FEAR", "# JOY", "# SADNESS",
                           "# SURPRISE", "ANGER SCORE", "DISGUST SCORE",
                           "FEAR SCORE", "JOY SCORE", "SADNESS SCORE",
                           "SURPRISE SCORE", "BEST_FIT")
  return(documents)
}

classify_polarity <- function(textColumns, algorithm="bayes", pstrong=0.5, pweak=1.0,
                              prior=1.0, verbose=FALSE,...) {
  matrix <- create_matrix(textColumns,...)
  lexicon <- read.csv("subjectivity.csv", header=FALSE)

  counts <- list(positive=length(which(lexicon[,3]=="positive")),
                 negative=length(which(lexicon[,3]=="negative")),
                 total=nrow(lexicon))
  documents <- c()

  for (i in 1:nrow(matrix)) {
    if (verbose) print(paste("DOCUMENT",i))
    scores <- list(positive=0, negative=0)
    doc <- matrix[i,]
    words <- findFreqTerms(doc, lowfreq=1)

    numpos <- 0
    numneg <- 0

    for (word in words) {
      index <- pmatch(word, lexicon[,1], nomatch=0)
      if (index > 0) {
        entry <- lexicon[index,]
        polarity <- as.character(entry[[2]])
        category <- as.character(entry[[3]])
        count <- counts[[category]]

        if (category=="positive") numpos <- numpos+1
        if (category=="negative") numneg <- numneg+1

        score <- pweak
        if (polarity == "strongsubj") score <- pstrong
        if (algorithm=="bayes") score <- abs(log(score*prior/count))

        if (verbose) {
          print(paste("WORD:", word,"CAT:", category,"POL:", polarity,
```

```
                      "SCORE:", score))
        }

        scores[[category]] <- scores[[category]]+score
      }
    }

    if (algorithm=="bayes") {
      for (key in names(scores)) {
        count <- counts[[key]]
        total <- counts[["total"]]
        score <- abs(log(count/total))
        scores[[key]] <- scores[[key]]+score
      }
    } else {
      for (key in names(scores)) {
        scores[[key]] <- scores[[key]]+0.000001
      }
    }

    best_fit <- names(scores)[which.max(unlist(scores))]
    ratio <- as.integer(abs(scores$positive/scores$negative))
    if (ratio==1) best_fit <- "neutral"
    documents <- rbind(documents, c(numpos, numneg, scores$positive, scores$negative,
                                    abs(scores$positive/scores$negative), best_fit))
    if (verbose) {
      print(paste("POS:", scores$positive, "NEG:", scores$negative,
                  "RATIO:", abs(scores$positive/scores$negative)))
      cat("\n")
    }
  }

  colnames(documents) <- c("#POS", "#NEG", "POS SCORE", "NEG SCORE",
                           "POS/NEG", "BEST_FIT")
  return(documents)
}

# OUR HELPER FUNCTIONS

#inputs a product name and returns clean product reviews
grab_reviews <- function(category, product) {
  product_reviews <- filter(category, name==product)
  return(product_reviews$cleanText)
}

# does the sentiment analysis and returns a dataframe with results
sentimental_analysis <- function(review_txt) {
  #classify emotion
  class_emo <- classify_emotion(review_txt, algorithm="bayes", prior=1.0)
  # get emotion best fit
  emotion <- class_emo[,13]
  # substitute NA's by "unknown"
  emotion[is.na(emotion)] <- "unknown"
```

```
  #classify polarity
  class_pol <- classify_polarity(review_txt, algorithm="bayes")
  # get polarity best fit
  polarity <- class_pol[,6]


  # data frame with results
  general <- data.frame(text=review_txt, emotion=emotion,
                        polarity=polarity, stringsAsFactors=FALSE)

  # sort data frame
  general <- within(general,
                    emotion <- factor(emotion,
                                      levels=names(sort(table(emotion), decreasing=TRUE))))

  return(general)
}


# SHINY BEGINS HERE

ui <- shinyUI(fluidPage(

  theme = shinytheme("cerulean"),

  # Application title
  titlePanel("Sentiment Analysis of Amazon Reviews"),

  sidebarLayout(
    sidebarPanel(
      # first drop-down menu to choose word
      selectInput("category",
                  label = "Choose an Amazon category",
                  choices = categories,
                  selected = "Clothing, Shoes, and Jewelry")),
    mainPanel(
      # second drop-down menu to choose product
      uiOutput("productSelector"),

      actionButton("visualize", "Visualize!"),

      tabsetPanel(
        tabPanel("Bar Graph", plotOutput("bargraphs")),
        tabPanel("Word Cloud", plotOutput("wordcloud")),
        tabPanel("Create Your Own Review",
                 textInput("newReview", "What do you think about this product?"),
                 actionButton("button", "Analyze"),
                 verbatimTextOutput("analyzeUser1"),
                 verbatimTextOutput("analyzeUser2"),
                 verbatimTextOutput("analyzeUser3"))
      )
    ))
))
```

```
server <- shinyServer(function(input, output) {
  # find the poems with the shakespearean word
  output$productSelector <- renderUI({

    if(input$category == "Clothing, Shoes, and Jewelry") {
      # group list of products
      clothing_products <- clothing %>%
        group_by(name) %>%
        summarize(count = n())

      # what we want to show user
      products <- as.character(clothing_products[["name"]])
    }

    else if(input$category == "Health and Personal Care") {
      # group list of products
      health_products <- health %>%
        group_by(name) %>%
        summarize(count = n())

      # what we want to show user
      products <- as.character(health_products[["name"]])
    }

    else if(input$category == "Sports and Outdoors") {
      # group list of products
      sports_products <- sports %>%
        group_by(name) %>%
        summarize(count = n())

      # what we want to show user
      products <- as.character(sports_products[["name"]])
    }

    # make second drop-down menu
    selectInput("products",
                label = "Choose a product",
                choices = products)

  })

  products <- eventReactive(input$visualize, {
    input$products
  })

  # show the positivity plot
  output$bargraphs <- renderPlot({
    # go from category name used in app to the dataframe name
    if(input$category == "Clothing, Shoes, and Jewelry") {
      cat <- clothing
    }
    if(input$category == "Health and Personal Care") {
```

```
    cat <- health
  }
  if(input$category == "Sports and Outdoors") {
    cat <- sports
  }

  #make product name back into string
  product <- as.String(products())

  #get the reviews and do sentimental analysis
  review_text <- grab_reviews(cat, product)
  general <- sentimental_analysis(review_text)

  # plot distribution of polarity
  polarityplot <- ggplot(general, aes(x=polarity)) +
    geom_bar(aes(y=..count.., fill=polarity)) +
    scale_fill_brewer(palette="RdGy") +
    labs(x="Polarity Categories", y="Number of Reviews",
         title = "Polarity")

  # plot distribution of emotions
  emotionplot <- ggplot(general, aes(x=emotion)) +
    geom_bar(aes(y=..count.., fill=emotion)) +
    scale_fill_brewer(palette="Dark2") +
    labs(x="Emotion Categories", y="Number of Reviews",
         title = "Emotions")

  plot_grid(polarityplot, emotionplot)


})

# show the emotions plot
output$wordcloud <- renderPlot({
  # go from category name used in app to the dataframe name
  if(input$category == "Clothing, Shoes, and Jewelry") {
    cat <- clothing
  }
  if(input$category == "Health and Personal Care") {
    cat <- health
  }
  if(input$category == "Sports and Outdoors") {
    cat <- sports
  }

  #make product name back into string
  product <- as.String(input$products)

  #get the reviews and do sentimental analysis
  review_text <- grab_reviews(cat, product)
  general <- sentimental_analysis(review_text)

  # separating text by emotion
```

```
  emos <- levels(factor(general$emotion))
  nemo <- length(emos)
  emo_docs <- rep("", nemo)
  for (i in 1:nemo)
  {
    tmp <- review_text[general$emotion == emos[i]]
    emo_docs[i] <- paste(tmp, collapse=" ")
  }

  # remove stopwords
  emo_docs <- removeWords(emo_docs, stopwords("english"))
  # create corpus
  corpus <- Corpus(VectorSource(emo_docs))
  tdm <- TermDocumentMatrix(corpus)
  tdm <- as.matrix(tdm)
  colnames(tdm) <- emos

  # comparison word cloud
  comparison.cloud(tdm, colors = brewer.pal(nemo, "Dark2"),
                   scale = c(3,.5), random.order = FALSE, title.size = 1.5)
})

# create your own review feature

newGraphs <- eventReactive(input$button, {
  input$newReview
})

output$analyzeUser1 <- renderText({
  #classify emotion
  class_emo <- classify_emotion(newGraphs(), algorithm="bayes", prior=1.0)
  print(paste("#ANGRY:", class_emo[1], "#DISGUST:", class_emo[2], "#FEAR:",
              class_emo[3], "#JOY:",class_emo[4], "#SADNESS:", class_emo[5],
              "#SURPRISE:", class_emo[6], "\n",
              "ANGER SCORE:", round(as.numeric(class_emo[7]),3),
              "DISGUST SCORE:", round(as.numeric(class_emo[8]),3),
              "FEAR SCORE:", round(as.numeric(class_emo[9]),3),
              "JOY SCORE:", round(as.numeric(class_emo[10]),3),
              "SADNESS SCORE:", round(as.numeric(class_emo[11]),3),
              "SURPRISE SCORE:", round(as.numeric(class_emo[12]),3),"\n",
              "BEST FIT:", class_emo[13]))
})

output$analyzeUser2 <- renderText({
  #classify polarity
  class_pol <- classify_polarity(newGraphs(), algorithm="bayes")
  print(paste("#POS:", class_pol[1], "#NEG:", class_pol[2], "\n",
              "POS SCORE:", round(as.numeric(class_pol[3]),3),
              "NEG SCORE:", round(as.numeric(class_pol[4]),3),
              "\n","POS/NEG:", round(as.numeric(class_pol[5]),3),
              "BEST FIT:", class_pol[6]))
})
```

```
  output$analyzeUser3 <- renderText({
    #classify emotion
    class_emo <- classify_emotion(newGraphs(), algorithm="bayes", prior=1.0)
    # get emotion best fit
    emotion <- class_emo[,13]
    # substitute NA's by "unknown"
    emotion[is.na(emotion)] <- "unknown"

    #classify polarity
    class_pol <- classify_polarity(newGraphs(), algorithm="bayes")
    # get polarity best fit
    polarity <- class_pol[,6]

    print(c(polarity, emotion))

  })


})

# Run the application
shinyApp(ui = ui, server = server)
```