Social Media Moderation with Sentiment Analysis

Introduction

Reddit is one of the internet's most popular online communities, with over 230 million unique monthly visitors from over 200 countries. Users post and share information, specifically in the form of comments. In 2015, over 8 million users left over 725 million comments. Users vote posts and comments up or down based on their opinion and earn "karma" for their participation[1]. Reddit is made up of various communities known as subreddits, centered around specific topics.

Reddit's Slogan:
"The front page of the Internet"

Popular posts and comments, earn "karma" but not all popular comments contribute to the community. If there are too many negative posts in a community people may be disincentivized to share and Reddit may lose users. In 2015, reddit introduced a new content policy and banned several offense communities that "generally make Reddit worse for everyone else" [2] and was not in keeping with their mission "to help people discover places where they can be their true selves, and empower our community to flourish."[3]

Although each comment has a "score" based on up and down votes, there is currently not a classification system that categorizes comments based on their overall sentiment. By moving beyond tracking only the most popular "up voted" or "down voted" comments, Reddit could track which comments, users and communities are negative or positive even if they are not highly popular.

[1] "By the Numbers: 60+ Amazing Reddit Statistics, http://expandedramblings.com/index.php/reddit-stats/
[2] "Reddit finally bans its most infamous racist communities because they 'made recruiting here more difficult', Business Insider, Matt Weinberger, 5 Aug 2015, http://www.businessinsider.com/reddit-bans-coontown-2015-8
[3] "About Reddit", https://about.reddit.com/

Social Media Moderation with Sentiment Analysis

By classifying comments by sentiment and identifying negative posts, reddit can ensure that it removes or flags users or comments who are consistently destructive to the community and rewards users who provide positive participation, not just those who are the most popular. Comments are the main source of user interaction on the site.

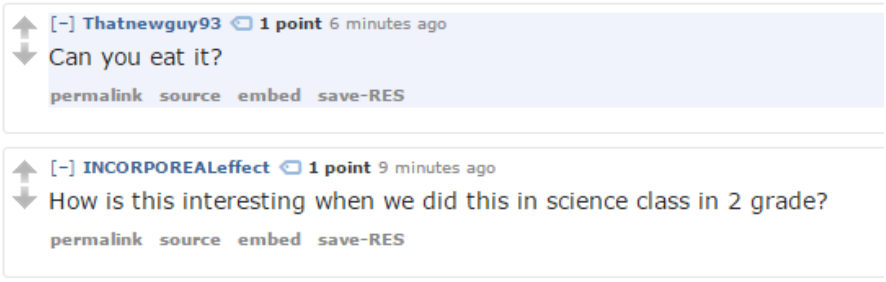


Figure 1 Sample Reddit Comment

This project aims to evaluate a method of classifying comments in a way that would be useful to categorize the general tone of individual comments, users, and communities. In order to correctly classify data on a site as large as reddit, we need an automated method. Since this is a classification problem, the model chosen was a naïve Bayes classification, a model well established for spam detection and sentiment analysis. In order to currently train the model, we needed to provide a set of pre classified data. The training set was classified with a simple, well established existing lexicon containing 6800 English language words divided into positive and negative[4]. Performing sentiment analysis on a sample of comments from Reddit allowed for the classification of words as "positive", "neutral" or "negative" with scores of 1, 0, and -1, respectively. A comment's score is the sum of the score of its constituent words after standardizing text. Although not all words are classified, this provides us with a good baseline to establish the overall tone of the comments.

[4] "Opinion Mining, Sentiment Analysis, and Opinion Spam Detection, Hu and Liu, https://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html

Social Media Moderation with Sentiment Analysis

Data Set

Data was collected from the reddit API outlined at http://www.reddit.com/dev/api using endpoint https://oauth.reddit.com/r/all/comments/.json. This endpoint provided a real time stream of comments from all subreddits as they were produced. Using an oAuth2 connection as a registered user application, over one million comments from almost 17,000 subreddits spanning a four-day period, 5 July to 8 July 8 2016 were collected. The scraping was performed in r using an "`RedditScrape.R`" script that utilized the `rCurl`, `jsonlite`, `httr` and `httpuv` libraries to download the relevant information in json format and then convert the data to data frames. Using the `r.utils` and `DBI` libraries, data was saved to a SQL database. Code was written to do an id check to ensure scraping could resume without duplicating comments by using the unique comment id. Another interesting feature of the API is it provided a "before" and "after" pointer to the previous and next comment id in the stream. This was also used to collect the data in a sequential manner. An alternative to scrapping live comments could be the examination of a set of 1.7 billion reddit comments loaded on Google Big Query[5] although significant data analysis may be cost prohibitive.

```
LoadData <- function(url.val) {
  # Loads comment data from a given reddit url.
  #
  # Args:
  #  url: The url of the reddit comments to scrape
  #
  # Returns:
  #  A data fame with comments, including subreddit, created date
  #  author, score (total, up and down  votes), and comment text
  init.req <- GET(url.val, config(token = my.token),
      user_agent("rpulldata v0.5 by /u/rdata"))
  init.req <- content(init.req, as = "text")
  main.data <- tryCatch({
    fromJSON(init.req)
  }, error = function(e) {
    cat("ERROR :",conditionMessage(e), "\n")
    # Sys.sleep(600)
    })
  if(!is.atomic(main.data[[2]]))
  {
    main <- main.data[[2]]$children$data
    main["before"]<- main.data[[2]]$before
    main["after"] <- main.data[[2]]$after
    main["inserteddate"] <- Sys.time()
  }

  reduced.main <- tryCatch({ main[, c("id","subreddit","created","author",
    "score","downs","ups","body","after","inserteddate")]
  }, error = function(e) {
    cat("ERROR :",conditionMessage(e), "\n")
    reduced.main <- data.frame(id = character(0), subreddit = character(0),
    created = numeric(0), author = character(0), score = integer(0),
    downs = integer(0), ups = integer(0), body = character(0),
    after = character(0))
  })
    return(reduced.main)
}
```

Figure 2 Partial Code to Load Data Set from Reddit API

[5] "1.7 billion reddit comments loaded on BigQuery", /u/fhoffa, https://www.reddit.com/r/bigquery/comments/3cej2b/17_billion_reddit_comments_loaded_on_bigquery/

The important fields in the data set are:

Field Name	Description	Example
id	Unique identifier per comment	d4z5y3s
subreddit	Community comment originated	AdviceAnimals
created	Time created (UTC Timestamp)	1467691477
author	User name of commenter	eric881
downs	Number of negative down votes	0
ups	Number of positive up votes	1
body	The text of the comment	I'm new to basketball trades and stuff, but is it just a small group of incidents that players go to other teams? Or are there multiple times players trade?
after		t1_d4z5y3s

In addition to the existing fields, several fields were added:

Field Name	Description	Example
score	Sentiment score using a basic integer score algorithm that subtracts negative words from positive words using an opinion lexicon	0
inserteddate	Time comment was inserted into the database	2016-07-04 16:04:37.693
score_category	“Positive”, “Neutral” or “Negative”	Neutral
created_time	Time created (date time format)	2016-07-05 04:04:37.000

Data Cleaning and Processing

The data was cleaned to remove unnecessary characters using `gsub()` and a basic regex and conversion to lower case, as well as using `strsplit` from `stringr` library. Comments that were duplicated as well as data with an author that included “moderator” or “bot” were removed. Comments that may have been posted by “bots” or automated programs may have skewed the results. Out of the larger data set, a subset was split into a training set of 22,500 comments, a validation set of 7,500 comments and finally tested on new data. Each comment in the training set was assigned a sentiment score based on the sum of the words used using the opinion lexicon described previously

using an r script named "`sentiment.R`". This sentence score was saved in the database along with the other data to provide the basis for determining the score_category. A simple update statement in SQL was used to classify the training data based on the score:

```
UPDATE Comments SET score_category =
CASE WHEN score < 0 THEN 'Negative'
     WHEN score > 0 THEN 'Positive'
     WHEN score = 0 THEN 'Neutral' END
```

The range of the scores was -55 to 47, with most comments clustered around a score of -1, 0, or 1, with a plurality with a score of 0. This suggests that most comments do not present strong sentiments on either the positive or negative side.

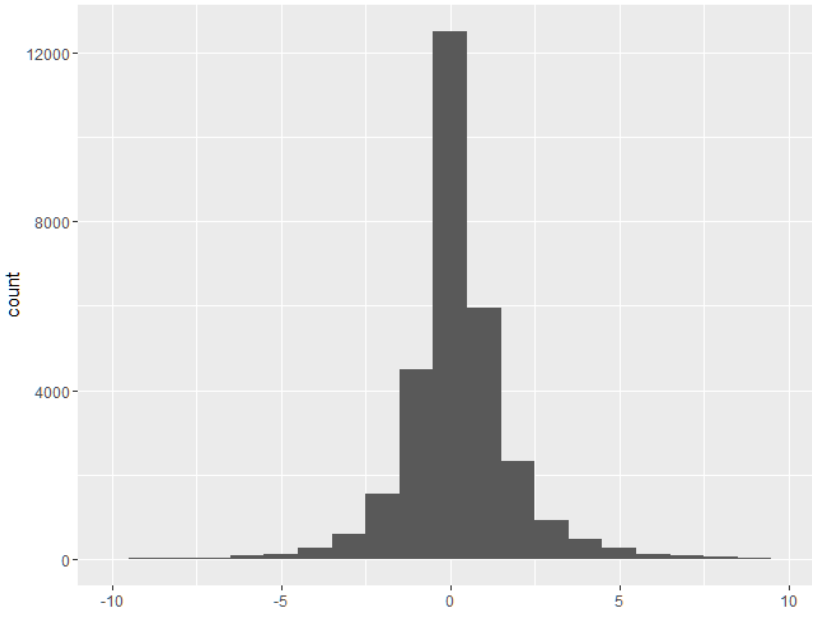


Figure 3 Sentiment vs Count (108 Records Not Shown)

Bin	*Frequency*
< -20	9
-20	10
-10	43
-1	7,180
0	12,516
1	5,953
10	4,234
20	56
> 20	8

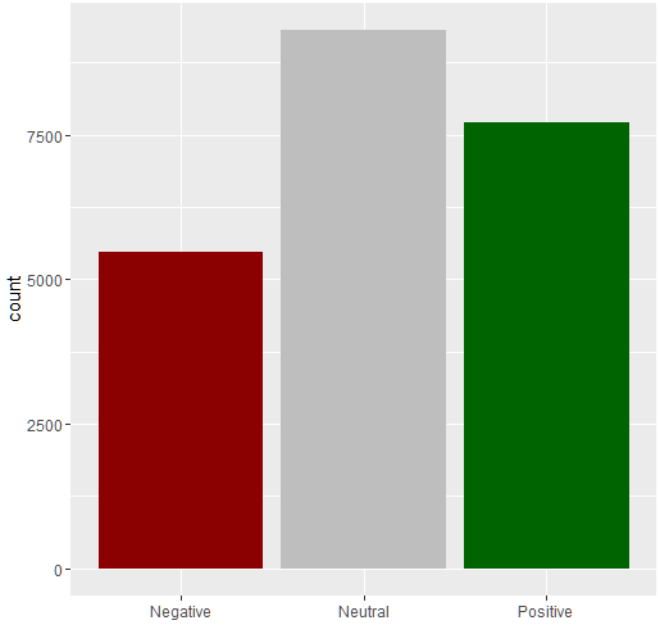


Figure 4 Sentiment Categories vs Count

Bin	*Frequency*
Negative	7,233
Neutral	12,516
Positive	10,251

Social Media Moderation with Sentiment Analysis

Analysis Approach

Once the comments in training set has been appropriately classified by sentiment, the naïve bayes model was ready to be developed with the "`comments_sent.R`" script. Using the `DBI` library data was imported from the database. An additional option was created to import from csv. The data frame column names were standardized, the score category was converted to a factor ("Positive", "Neutral" and "Negative") and basic exploration of the data with `table` and `str` was completed.

Positive	Negative
a+	2-faced
abound	2-faces
abounds	abnormal
abundance	abolish
abundant	abominable
accessable	abominably
accessible	abominate
acclaim	abomination
acclaimed	abort
acclamation	aborted
accolade	aborts
accolades	abrade
accommodative	abrasive
accomodative	abrupt
accomplish	abruptly
accomplished	abscond
accomplishment	absence
accomplishments	absent-minded
accurate	absentee
accurately	absurd

Figure 5 Sample Positive and Negative Words from Hu and Liu

```
> # examine the structure of the comments data
'data.frame':  30000 obs. of 2 variables:
 $ score_category: chr  "Neutral" "Neutral" "Negative" "Positive" ...
 $ body          : chr  "I'm new to basketball trades and stuff, but is it ju
st a small g

> table(comments_raw$score_category)

Negative  Neutral Positive
    7233    12516    10251
```

Next the text mining library `tm` was used to create a corpus, or collection of written text from the data set which was examined with the print and inspect commands. A series of transformations were made to clean the data using `tm_map` and commands to standardize the corpus as lower case, remove numbers, basic stop words (e.g. a, an, I, his, etc.), punctuation, and whitespace. The data set was then inspected again.

```
# clean up the corpus using tm_map()
corpus_clean <- tm_map(comments_corpus, content_transformer(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)
```

Social Media Moderation with Sentiment Analysis

The data corpus was then ready to be transformed into a Document Term Sparse Matrix, which helps show what terms are included and their frequency. Once completed the raw comments, document term matrix and cleaned text corpus were all split into training and test sets with 75% or 22,500 comments going to the training set and 25% or 7,5000 comments going to the test set. Within the training data `subset()` was used to create subsets of words used in comments scored as positive, neutral and negative that were output using the library `wordcloud`, as shown below:

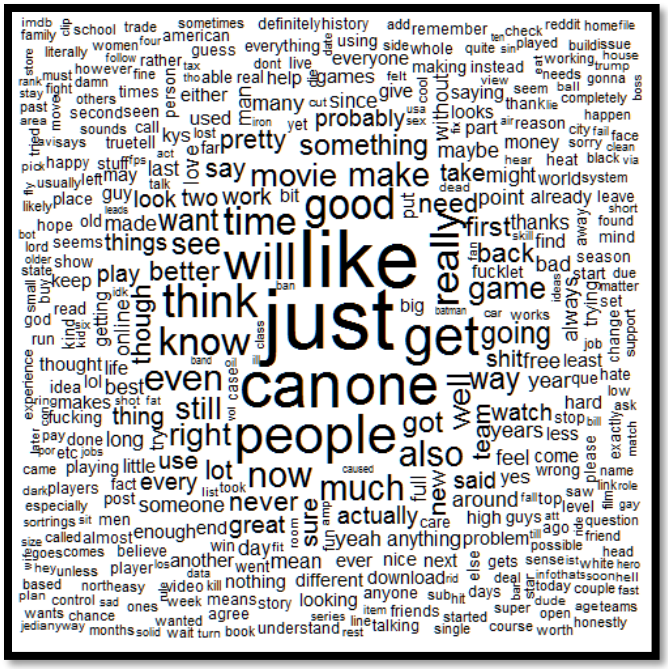


Figure 5 Training Data (All)

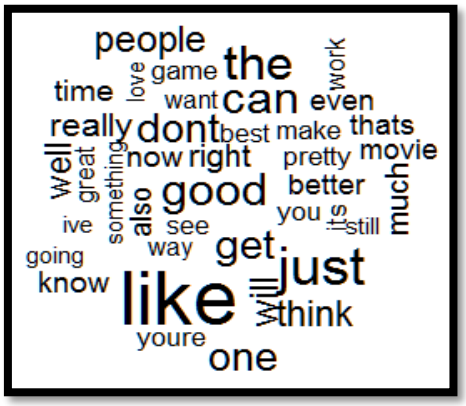


Figure 6 Training Data (Positive)

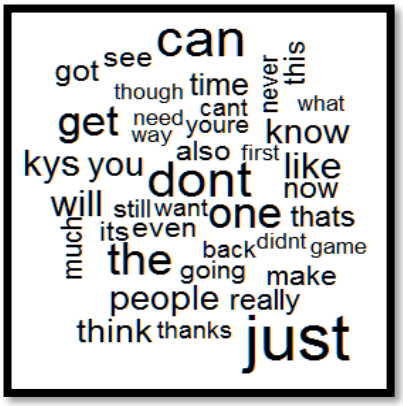


Figure 7 Training Data (Neutral)

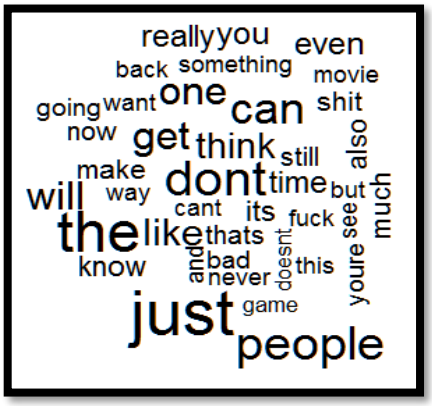


Figure 8 Training Data (Negative)

Training the Model

With the data classified and initial exploration complete, a function was applied to the document term matrix using apply to the data counts into “No” or “Yes” binary so that it could be trained using the naïve Bayes classifier in the `e1071` library as shown below

```
# convert counts to a factor
convert_counts <- function(x) {
  x <- ifelse(x > 0, 1, 0)
  x <- factor(x, levels = c(0, 1), labels = c("No", "Yes"))
}

# apply() convert_counts() to columns of train/test data
comments_train <- apply(comments_train, MARGIN = 2, convert_counts)
comments_test  <- apply(comments_test, MARGIN = 2, convert_counts)

## Step 3: Training the model on the data
library(e1071)
comments_classifier <- naiveBayes(comments_train,
                                  comments_raw_train$score_category)
```

Using the 2,500 comment test set the predict function was used to evaluate the model performance and the `gmodels` library was used to create a CrossTable:

```
Cell Contents
|-------------------------|
|                       N |
|           N / Col Total |
|-------------------------|

 Total Observations in Table:  7500

              | actual
   predicted |  Negative |   Neutral |  Positive | Row Total |
-------------|-----------|-----------|-----------|-----------|
    Negative |       642 |       186 |       153 |       981 |
             |     0.366 |     0.058 |     0.060 |           |
-------------|-----------|-----------|-----------|-----------|
     Neutral |       921 |      2726 |      1273 |      4920 |
             |     0.525 |     0.851 |     0.501 |           |
-------------|-----------|-----------|-----------|-----------|
    Positive |       190 |       293 |      1116 |      1599 |
             |     0.108 |     0.091 |     0.439 |           |
-------------|-----------|-----------|-----------|-----------|
Column Total |      1753 |      3205 |      2542 |      7500 |
             |     0.234 |     0.427 |     0.339 |           |
-------------|-----------|-----------|-----------|--------
```

Figure 9 Test Data Cross Table Results

The model performs reasonably well with an accuracy of 73% but a recall and precision of around 60% but could benefit from additional tuning.

Social Media Moderation with Sentiment Analysis

Analyzing the Model Results

Examining the cross table more closely we can look how the model performs in each category and discuss the possible implications of that in real world applications.

		Negative		Positive		Neutral	
True Positive	False Negative	642	1111	1116	1426	2726	479
False Positive	True Negative	339	5408	483	4475	2194	2101
	F1 Score	47.0%		53.9%		67.1%	

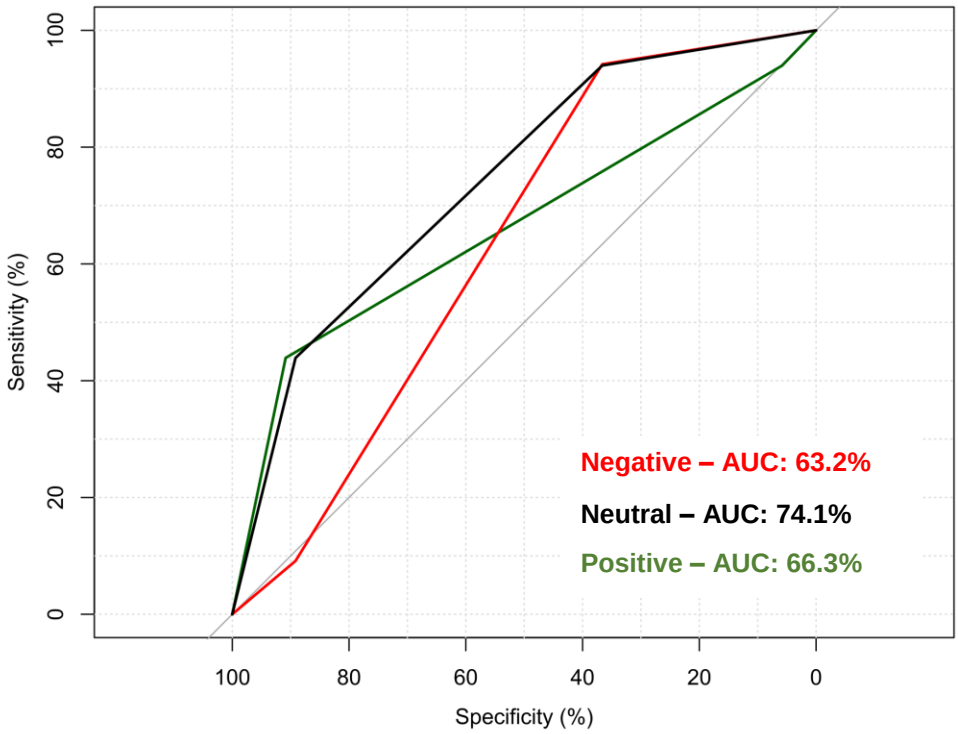
Figure 10 Model True and False Negatives and Positives

The model is most successful at classifying the neutral comments that make up the majority of the data set, correctly identifying 85% of them. However, the model was not as successful correctly identifying positive comments at 44% or negative comments at 37%. Over 50% of both the incorrectly classified positive and negative comments were classified as neutral. This indicates the model is unlikely to classify positive comments as negative or negative comments as positive. Examining the data further, we can use the `pROC` library to examine the area under the curve for the receiver operating characteristic. By plotting the true positive rate against the false positive rate we can visually see the performance of the classifier, yielding an average multi class area under the curve of 0.6787 or 68% using the command:

```
multiclass.roc(comments_raw_test$score_category,
as.numeric(comments_test_pred), percent=TRUE, plot=TRUE,
print.auc=TRUE, show.thres=TRUE, auc.polygon=TRUE, grid=TRUE)
```

Looking at the individual AUCs we see how the model performs for each sentiment:

Negative – AUC: 63.2%, Neutral – AUC: 74.1%, Positive – AUC: 66.3%



Potential Application

Reddit could begin to use a model similar to this to begin automatically “soft flag” individuals who consistently post negative comments, as well as those who post positive comments and bring them to the attention of subreddit moderators for further review. Additionally, fun features such as “mood badges” could increase community engagement and allow users further details beyond their “karma” (sum of up votes minus down votes). With a focus on applications that identify clearly positive and negative comments and ignore neutral comments some of the issues with the model would be mitigated.

Limitations and Future Improvements

Examining comments over a longer time period such as a year would be beneficial. Obtaining a dataset of past comments is difficult using the Reddit API and the sheer volume of data would have been unwieldy to process. However, the previously mentioned BigQuery data dump could provide a useful source for some analysis. Some of interesting questions that cannot be answered include location based information or other demographic based information. Because of the anonymous nature of the Reddit community there is no way to collect this type of information linked to specific individuals. Additionally, data exploration shows that not all comments are posted in English so translation may be useful. Using a more complex model such as a neural network instead of a simple classifier like naïve Bayes may also yield significant improvements. Improvements in the sentiment analysis classification of the training set using humans recruited via micro work platforms such as Amazon Mechanical Turk likely would also yield significant improvements to the model.

Acknowledgements

I would like to and acknowledge and thank my mentor Raghunandan Patthar for his guidance on this project, particularly in the area of the practical applications of classifiers and sentiment analysis in real world applications.

Appendix A – RedditScrape.R

```
# Author: Anthony J Buchanan
# File Description: Reddit scraper to pull in data from a given subreddit
# as an input. Outputs a csv with the reddit comments for the last year .
library(jsonlite)
library(RCurl)
library(dplyr)
library(R.utils)
library(httr)
library(httpuv)
library(DBI)
setwd("D:/Users/Anthony/Documents/Springboard")
RedditToken <- function(appscript, secretvalue) {
# Loads Reddit OAuth2 tokens.
  #
  # Args:
  #  appscript: The reddit application script code
  #  secret: The reddit secret api code
  #
  # Returns:
  #  Reddit authorization token

  # OAuth2 Settings
  reddit <- oauth_endpoint(
    authorize = "https://www.reddit.com/api/v1/authorize",
    access = "https://www.reddit.com/api/v1/access_token")
  app <- oauth_app("reddit", appscript, secretvalue)
  token <- oauth2.0_token(reddit, app, scope = c("read"),
    use_basic_auth = TRUE, user_agent("rpulldata v0.5 by /u/rdata"))
  return(token)
}
# Get the Reddit Token
my.token <- RedditToken("gOfXlYcHrvrWTg", "0aAflGdXeKPiaDcq_uaPeFBa1oo")
LoadData <- function(url.val) {
  # Loads comment data from a given reddit url.
  #
  # Args:
  #  url: The url of the reddit comments to scrape
  #
  # Returns:
  #  A data fame with comments, including subreddit, created date
  #  author, score (total, up and down  votes), and comment text
  init.req <- GET(url.val, config(token = my.token),
      user_agent("rpulldata v0.5 by /u/rdata"))
  init.req <- content(init.req, as = "text")
  main.data <- tryCatch({
    fromJSON(init.req)
  }, error = function(e) {
    cat("ERROR :",conditionMessage(e), "\n")
    # Sys.sleep(600)
    })
  if(!is.atomic(main.data[[2]]))
  {
    main <- main.data[[2]]$children$data
    main["before"]<- main.data[[2]]$before
    main["after"] <- main.data[[2]]$after
    main["inserteddate"] <- Sys.time()
  }

  reduced.main <- tryCatch({ main[, c("id","subreddit","created","author",
    "score","downs","ups","body","after","inserteddate")]
```

```
  }, error = function(e) {
    cat("ERROR :",conditionMessage(e), "\n")
    reduced.main <- data.frame(id = character(0), subreddit = character(0),
    created = numeric(0), author = character(0), score = integer(0),
    downs = integer(0), ups = integer(0), body = character(0),
    after = character(0))
  })
    return(reduced.main)
}
WriteData <- function(dframe) {
  # Writes comment data from a dataframe.
  #
  # Args:
  #  dframe: The dataframe to checka and write
  #
  # Returns:
  #  Number of rows in a data fame with comments not in the database,
  # including subreddit, created date, author, score (total, up and down votes),
  # and comment text
  idcheck <- dbGetQuery(devdb, paste("select id from Comments where id in (",
    paste("'",dframe$id,collapse="','",sep=""), "')",sep=""))
  ret.data <- anti_join(dframe, idcheck, by="id")
  if (nrow(ret.data) > 0)
  {
    dbWriteTable(devdb, name="Comments", ret.data, append = T,
                 overwrite = F)
  }
  return(nrow(ret.data))
}
subreddit <- "all"
url.start <- paste("https://oauth.reddit.com/r/",subreddit,
                   "/comments/.json?limit=100&t=year",sep="")
# Database Stuff
devdb <- dbConnect(RSQLServer::SQLServer(), server="localhost", port=1433,
                   properties=list(user="rdata", password="password"))
commentstart <- dbGetQuery(devdb,
  "select top 1 id from Comments order by inserteddate DESC")
if(nrow(commentstart) > 0)
{
  url.next <- paste(url.start,"&after=", gsub("t1_","",commentstart[1,1]),
    sep="")
} else {
  url.next <- url.start
}
total.data <- LoadData(url.next)
rows.written <- WriteData(total.data)
while (nrow(tail(total.data["after"],1)) > 0) {
    url.next <- paste(url.start,"&after=", gsub("t1_","",
    tail(total.data["after"],1)), sep="")
    total.data <- LoadData(url.next)
    rows.written <- WriteData(total.data)
    if (rows.written < 1)
    {
      total.data <- LoadData(url.start)
      rows.written <- WriteData(total.data)
    }
  }
# write.csv(total.data, paste("RedditYear",subreddit,"Comments.csv",sep=""))
dbDisconnect(devdb)
```

Social Media Moderation with Sentiment Analysis

Appendix B – `sentiment.R`

```
#' score.sentiment() implements a very simple algorithm to estimate
#' sentiment, assigning a integer score by subtracting the number
#' of occurrences of negative words from that of positive words.
#'
#' @param sentences vector of text to score
#' @param pos.words vector of words of postive sentiment
#' @param neg.words vector of words of negative sentiment
#' @param .progress passed to <code>laply()</code> to control of progress bar.
#' @returnType data.frame
#' @return data.frame of text and corresponding sentiment scores
library(DBI)
setwd("D:/Users/Anthony/Documents/Springboard")
hu.liu.pos = scan('positivew.txt', what='character', comment.char=';')
hu.liu.neg = scan('negativew.txt', what='character', comment.char=';')
# create a postive and negative word list
pos.words = c(hu.liu.pos)
neg.words = c(hu.liu.neg)
# Database Stuff
devdb <- dbConnect(RSQLServer::SQLServer(), server="localhost", port=1433,
                   properties=list(user="rdata", password="password"))
if(!exists("comments"))
   {
     comments <- dbGetQuery(devdb, "select body from Comments")
}
sentences <- comments[['body']]

scores.sentiment = function(sentences,pos.words,neg.words)
{
  require(plyr)
  require(stringr)
  scores = laply(sentences, function(sentence, pos.words, neg.words) {
  # clean up sentences with R's regex-driven global substitute, gsub():
  sentence = gsub('[[:punct:]]', '', sentence)
  sentence = gsub('[[:cntrl:]]', '', sentence)
  sentence = gsub('\\d+', '', sentence)
  # and convert to lower case:
  sentence = tolower(sentence)
    # split into words. str_split is in the stringr package
  word.list = strsplit(sentence, '\\s+')
  # sometimes a list() is one level of hierarchy too much
  words = unlist(word.list)
    # compare our words to the dictionaries of positive & negative terms
  pos.matches = match(words, pos.words)
  neg.matches = match(words, neg.words)
    # match() returns the position of the matched term or NA
  # we just want a TRUE/FALSE:
  pos.matches = !is.na(pos.matches)
  neg.matches = !is.na(neg.matches)
    # and conveniently enough, TRUE/FALSE will be treated as 1/0 by sum():
  score = sum(pos.matches) - sum(neg.matches)
    return(score)
  }, pos.words, neg.words)
    scores = data.frame(score=scores, text=sentences)
  return(scores)
}
reddit.sentiment <- scores.sentiment(sentences,pos.words,neg.words)
View(reddit.sentiment)
dbWriteTable(devdb, name="Score", reddit.sentiment, append = T, overwrite = T)
```

Social Media Moderation with Sentiment Analysis

Appendix C – comments_sent.R

```
# Student: Anthony J Buchanan, Mentor: Raghunandan Patthar
# File Description: Reads in data on Reddit comments and does sentiment analysis
library(DBI)
setwd("D:/Users/Anthony/Documents/Springboard")
# read the comments data into the comments data frame
 devdb <- dbConnect(RSQLServer::SQLServer(), server="localhost", port=1433,
                    properties=list(user="rdata", password="password"))

 comments_raw <- dbGetQuery(devdb, "select top 30000 score_category, body,
                 subreddit from Comments where author not like '%bot%' and
                 author not like '%moderator%' and body not in
                 (select body from Comments group by body having count(*) > 1)")
#comments_raw <- read.csv("comments_sent.csv", stringsAsFactors = FALSE)
  names(comments_raw)[1] <- "score_category" #rename because csv is off
  names(comments_raw)[2] <- "body"
  names(comments_raw)[3] <- "subreddit"
# examine the structure of the comments data
str(comments_raw)
# convert score category to factor.
comments_raw$score_category <- factor(comments_raw$score_category)
# examine the type variable more carefully
str(comments_raw$score_category)
table(comments_raw$score_category)
# build a corpus using the text mining (tm) package
library(tm)
comments_corpus <- Corpus(VectorSource(comments_raw$body))
# examine the comments corpus
print(comments_corpus)
inspect(comments_corpus[1:3])
# clean up the corpus using tm_map()
corpus_clean <- tm_map(comments_corpus, content_transformer(tolower))
corpus_clean <- tm_map(corpus_clean, removeNumbers)
corpus_clean <- tm_map(corpus_clean, removeWords, stopwords())
corpus_clean <- tm_map(corpus_clean, removePunctuation)
corpus_clean <- tm_map(corpus_clean, stripWhitespace)
corpus_clean <- tm_map(corpus_clean, PlainTextDocument)
# examine the clean corpus
inspect(comments_corpus[1:3])
inspect(corpus_clean[1:3])
# create a document-term sparse matrix
comments_dtm <- DocumentTermMatrix(corpus_clean)
#comments_dtm
# creating training and test datasets
comments_raw_train <- comments_raw[1:22500, ]
comments_raw_test  <- comments_raw[22501:30000, ]
comments_dtm_train <- comments_dtm[1:22500, ]
comments_dtm_test  <- comments_dtm[22501:30000, ]
comments_corpus_train <- corpus_clean[1:22500]
comments_corpus_test  <- corpus_clean[22501:30000]
# check that the proportion of score category is similar
prop.table(table(comments_raw_train$score_category))
prop.table(table(comments_raw_test$score_category))
# word cloud visualization
library(wordcloud)
wordcloud(comments_corpus_train, min.freq = 30, random.order = FALSE)
# subset the training data into positive, neutral and negative groups
positive <- subset(comments_raw_train, score_category == "Positive")
neutral  <- subset(comments_raw_train, score_category == "Neutral")
negative  <- subset(comments_raw_train, score_category == "Negative")
wordcloud(positive$body, max.words = 40, scale = c(3, 0.5))
```

```
wordcloud(neutral$body, max.words = 40, scale = c(3, 0.5))
wordcloud(negative$body, max.words = 40, scale = c(3, 0.5))
# indicator features for frequent words
findFreqTerms(comments_dtm_train, 5)
comments_dict <- findFreqTerms(comments_dtm_train, 5)
comments_train <- DocumentTermMatrix(comments_corpus_train,
                                     list(dictionary = comments_dict))
comments_test  <- DocumentTermMatrix(comments_corpus_test,
                                     list(dictionary = comments_dict))
# convert counts to a factor
convert_counts <- function(x) {
  x <- ifelse(x > 0, 1, 0)
  x <- factor(x, levels = c(0, 1), labels = c("No", "Yes"))
}
# apply() convert_counts() to columns of train/test data
comments_train <- apply(comments_train, MARGIN = 2, convert_counts)
comments_test  <- apply(comments_test, MARGIN = 2, convert_counts)
## Step 3: Training a model on the data ----
library(e1071)
comments_classifier <- naiveBayes(comments_train,
                        comments_raw_train$score_category)
# comments_classifier
## Step 4: Evaluating model performance ----
comments_test_pred <- predict(comments_classifier, comments_test)
library(gmodels)
CrossTable(comments_test_pred, comments_raw_test$score_category,
           prop.chisq = FALSE, prop.t = FALSE, prop.r = FALSE,
           dnn = c('predicted', 'actual'))
```