Differentiating Satire Vs Real News

Khizar Tahir

Executive summary:

In this project, we attempt to use text mining techniques and topic modeling to understand how real news headlines differ from satire in terms of usage of words, structure of writing, areas of discussion among other things. We then use our understanding of this to evaluate another set of headlines to see if we can identify some commonalities between the two sets of real news headlines and if some conclusions can be drawn to tell real news apart from satire.

Data:

For our analysis and validation, we use two datasets as described below:

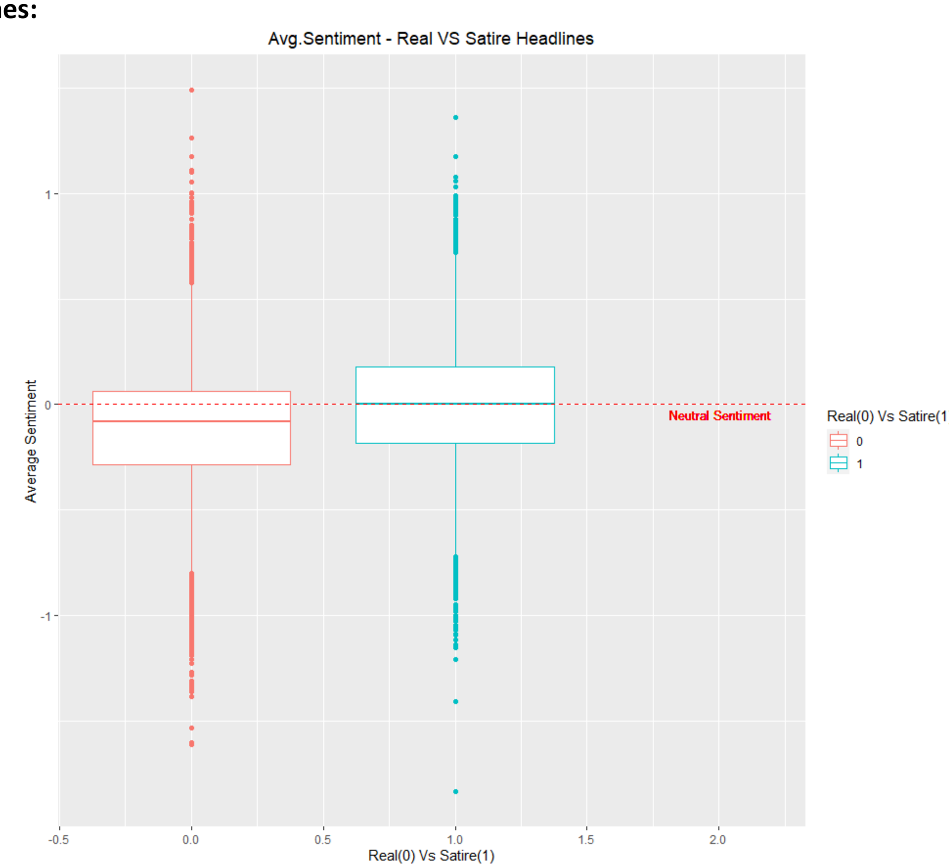
1. The first dataset was provided to us. It contained thousands of news headlines belonging to either of two categories: legitimate (real) news headlines extracted from mainstream news sources, and satire news headlines retrieved from websites that make satirical comments and interpretations of news for comedy and entertainment purposes. The real news headlines are labeled '0' and the satirical labeled '1'. This dataset is used to identify differences between real and satire news headlines.
2. The second dataset was collected by us by scraping headlines off of Google News. To avoid repetition of news headlines and to ensure validity and usefulness of data, we scraped the headlines at least 12 hours apart and fetched just over 50 headlines in each attempt. We repeated this to get 250+ real news headlines from Google News. This dataset is used to test how much our understanding is correct.

Understanding differences between Real and Satire:

In this section of the report, we will conduct several analyses, each aimed at identifying how does a particular analysis technique yields different results for real and satire news headlines. To achieve this in *most* analyses, we create 2 datasets from the given dataset: one containing only real news (label of '0'), and the other containing only satire news (label of '1'). We then conduct text mining techniques and compare results as shown in the sub-sections below.

PART 1 – Sentiments presented in the headlines:

We first use the whole given dataset to calculate a sentence-level sentiment score for each headline. Essentially, we try to identify what each headline means through the kind of emotions it expresses. On a numerical scale, the more negative a score, the more negative the sentiment expressed and vice versa. We separate the data by the two categories – real and satire, and plot the boxplot of scores shown on the right.



PART 1 – Observation:

The red boxplot is for real news headlines, and turquoise for satire. We can see that real news have a negative average sentiment score while satire has a neutral average sentiment score. Without worrying about the exact values, the important thing to note here is that <u>real news have lower sentiment scores than satire</u>.

PART 2 – Average length of headlines:

We want to see if length of headlines can be used to differentiate real headlines from fake. The table below shows the results:

Average number of words per headline	
Real news headlines	**Satire news headline**
11.8	13.9

PART 2 – Observation:

Although the difference is not huge, we can still observe that real news have lesser number of words per headline on average than satire news probably because satire requires more words to set the tone and achieve the humor.

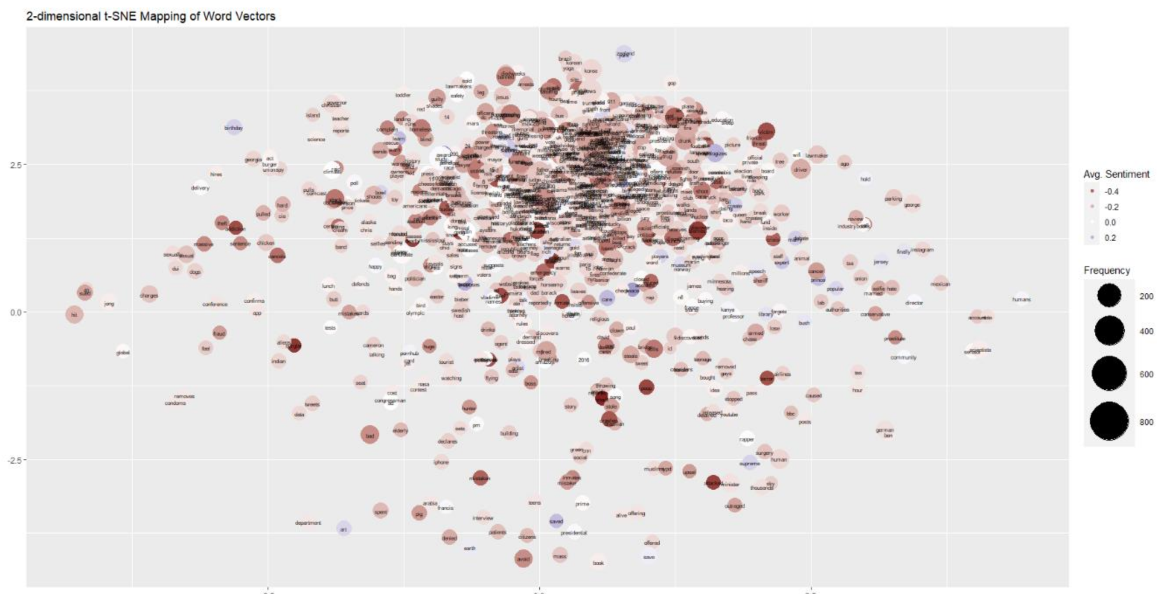
PART 3 – Form clusters of words used:

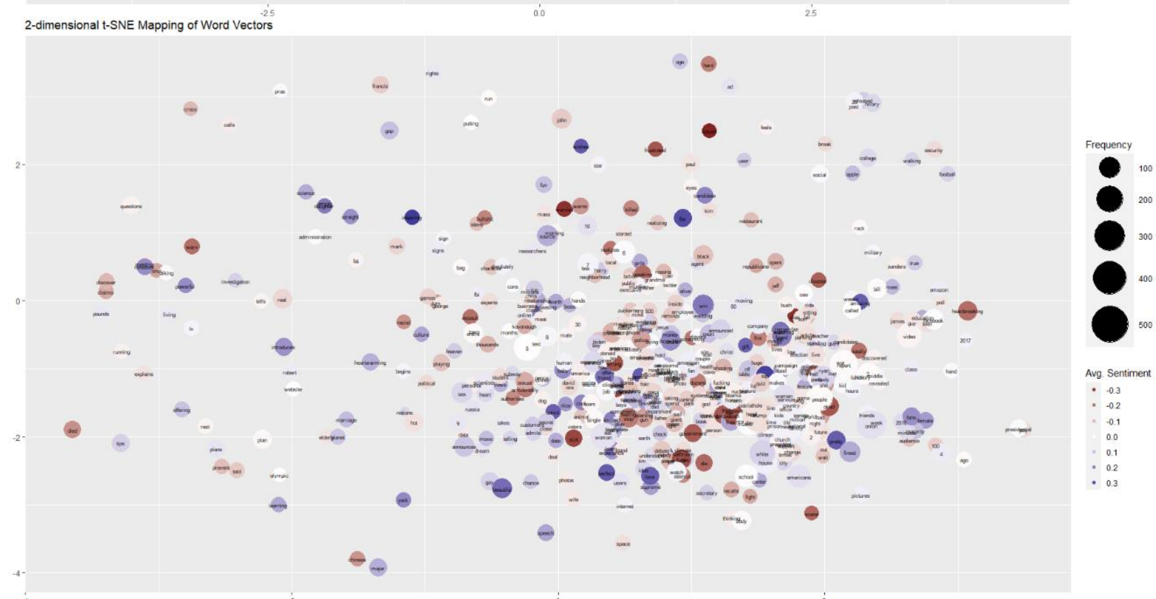
In this part, we plot t-SNE graphs that create naturally forming clusters such that words used with similar meaning are closer to each other and words different in meaning are further apart. We plot it for both real and satire headlines separately below.

PART 3 – Observation:

In a t-SNE plot, each circle represents a word used. Size of the circle represents the frequency, while color denotes sentiment: darker brown – more negative, darker blue – more positive.

The upper graph is for real headlines. Presence of more brown circles confirms our earlier observation of lower/negative sentiments. The concentration of circles shows that there are some specific topics/areas which are focused more by these – real news are not equally divided among different topics. Lower graph is for satire and shows more positive sentiments and a cluster around some topics. Clusters are too crowded to know topics though.





PART 4 – Most used words:

We now create visualizations to see which words are most commonly used in real news versus satire.

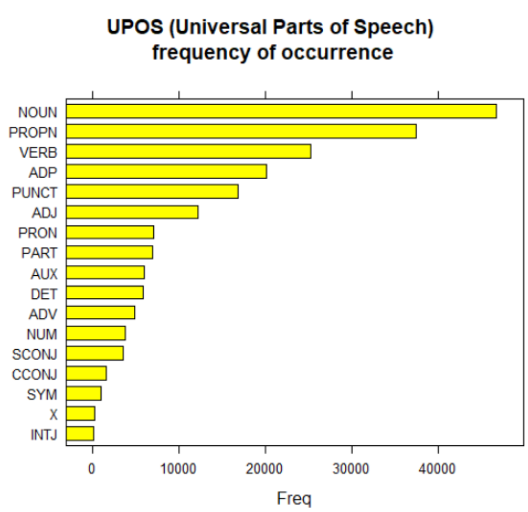
Real headlines wordcloud | **Satire headlines wordcloud**

PART 4 – Observation:

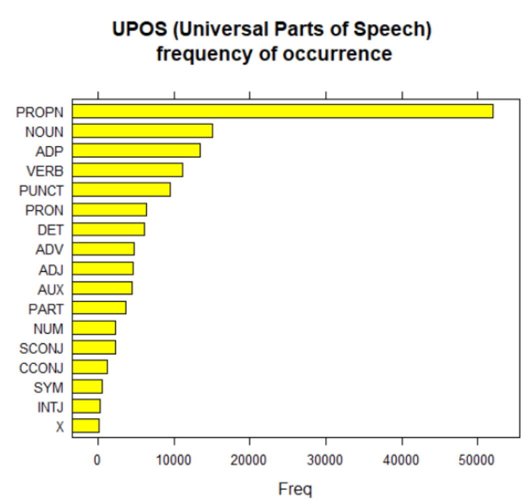
In a wordcloud representation, the size of the word corresponds with the frequency of the word in the evaluated text. From the real news headline wordcloud, we observe that the words 'police', 'says', 'arrested', 'school', 'sex', 'people' are used the most. Real news are more likely to cover actual stories like crimes, arrests, police intervention, or may be narrating what legitimate sources have said through usage of 'says'. From the satire wordcloud, we note that the words 'man', 'woman', 'news', 'life', 'trump', 'report' are used the most. Many of these words from satire wordcloud appear syllables that do not add value but we retain them as they explain a few things. These words are generic words mostly. Satire headlines seem to use generic 'man/woman' words to create a comment rather than give actual names – often because the stories are made up. Additionally, satire news focus on politicians which is expected.

PART 5 – Usage of Universal Parts of Speech (UPOS):

In this part of the analysis, we look at the way parts of speech are utilized in our 2 categories of news headlines. We create visualizations to help us observe differences easily.

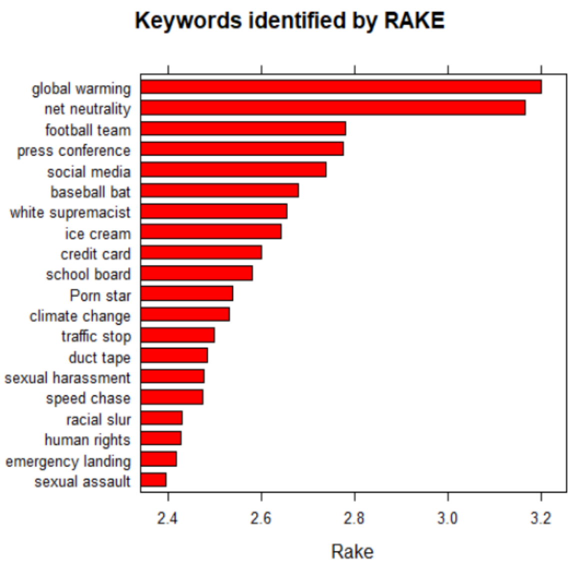


The left graph (real) shows a high usage of nouns, while the right graph (satire) has relatively less frequency of nouns. This confirms are earlier observation that satire tends to generalize while real news uses actuals names and other nouns.

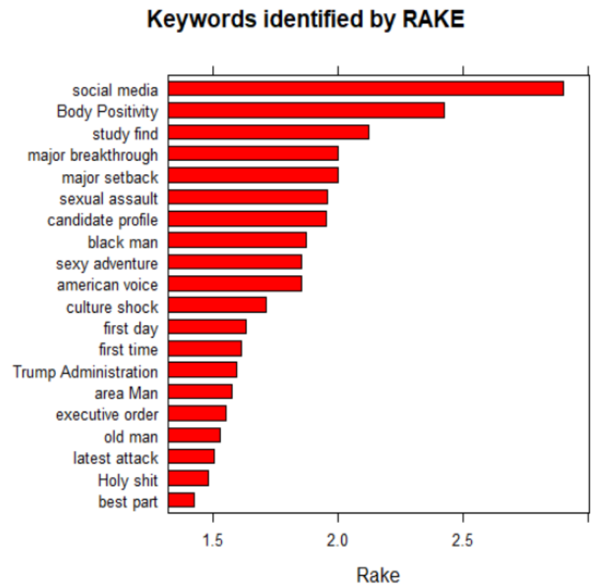


Next, we checked graphs for frequency of occurrences of specific parts of speech like nouns, adjectives, etc. But they did not give us any new insights other than we have already. Those graphs can still be found in the appendix.

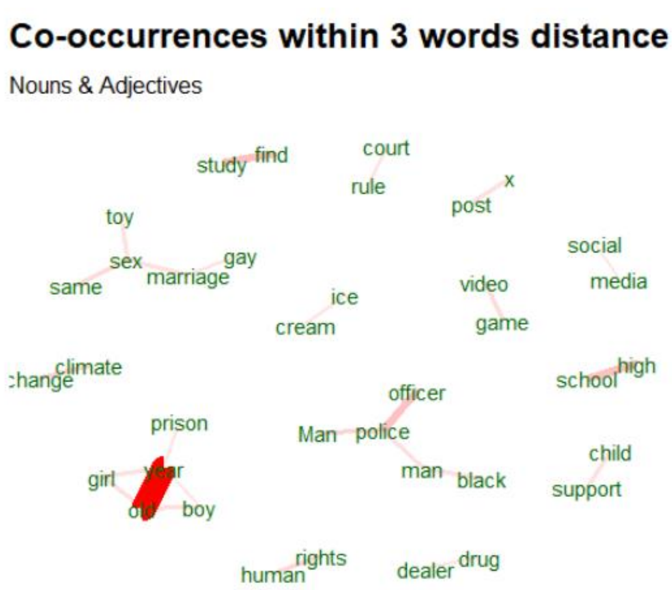
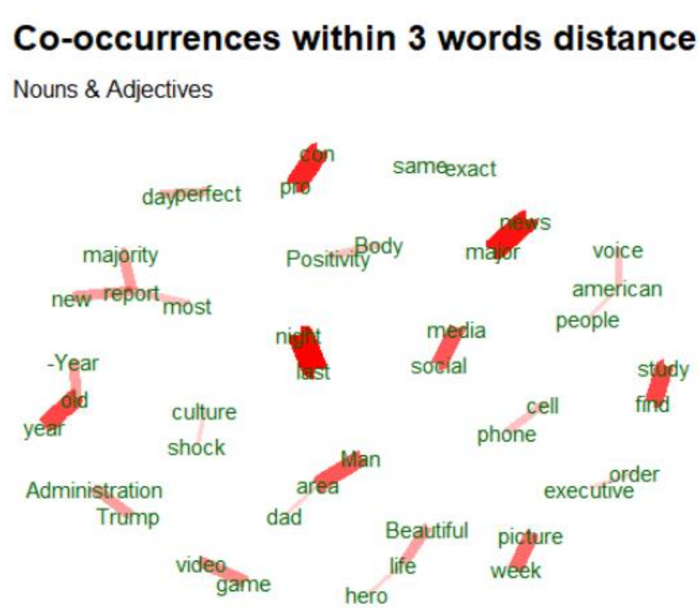
Our next step is to use a method 'RAKE' to determine key phrases in a body of text by analyzing the frequency of word appearance and its co-occurrence with other words in the text.



To the left is the RAKE keyword identification for real news showing keywords like global warming, net neutrality, press conference, etc. which are actual news topics. To the right is the RAKE keywords for satire showing social media as the most prominent keyword. This could be because satire uses social media to gather content for humor. Satire news also use attention-seeking keywords like major break-though/setback, study finds extensively.



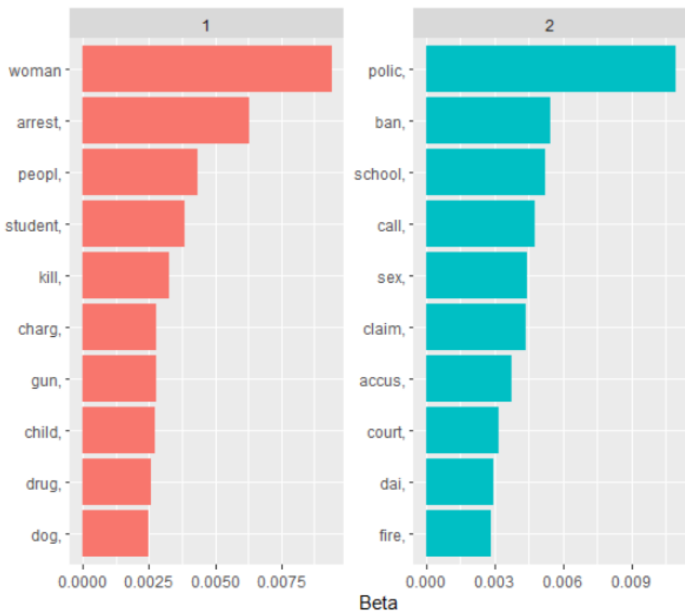
In the above analysis, we focused on words occurring together. Now, we focus on co-occurrences of words within 3-word distances. Basically, we identify what words appear near each other frequently with a maximum distance of 3 words in between. See the table below.

Real news headlines	Satire news headlines
	
Real news cover crime a lot and always tend to mention gender and age of a person being imprisoned. It also covers police arrest stories detailing ethnicities too when they do.	Satire news focuses on political headlines (Trump admin), social media, and other catchy words like study finds, major news, etc.

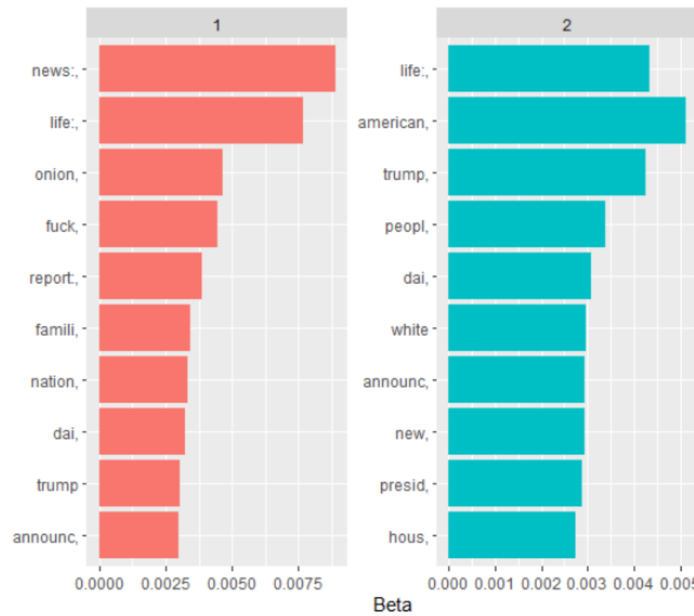
PART 6 – Topic modeling:

In this part, we aim to identify the topics covered in each of the 2 categories of news. Up till now, we were focused on keywords and their combinations. Now we are trying to find out the topics being discussed in these headlines. We use a topic modeling technique called LDA to help us do it. Although we know the categories of news, we still do not know what topics they might be covering. LDA allows us to run such analysis with limited knowledge of topics. It is smart at identifying on its own. Of course, we tailor it a bit to get the desired results. For example, we stem the words, meaning, each word is brought to its root word and then analysis run.

Real news headlines topics



Satire news headlines topics



PART 6 – Observation:

After iterating over the right number of topics to extract, we settled for 2 topics each. More resulted in repetition. For real headlines, the first topic was related to crimes possibly involving a student, woman, and a gun leading to a kill. The second topic also involved police but with a different crime – possible involving accusation of sexual assault.

For satire, both the topics were similar and revolved around President Trump, The White House, and other general life news. In summary, real news were focused on crimes or more alarming news while satire was focused on politics and other general remarks about everyday life things.

Summarizing observations – Real Vs Satire:

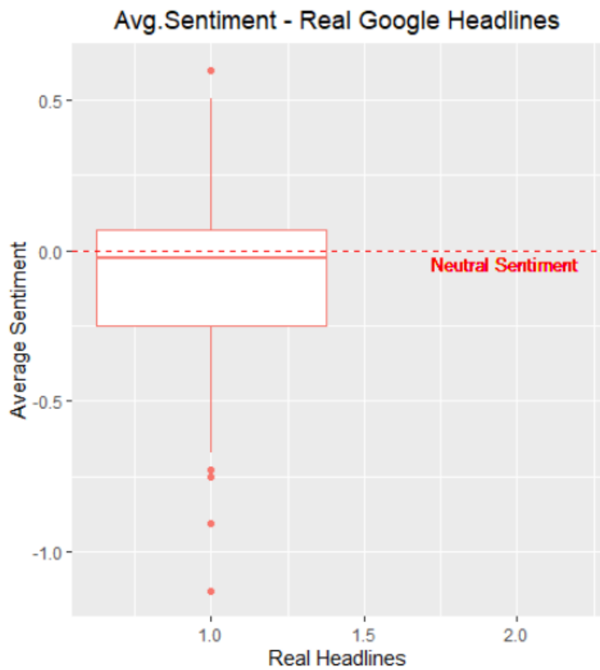
We now summarize the observations from the six analyses above.

- Real news tend to have a more negative sentiment than satire.
- Real news have lesser words per headline than satire.
- Real news are more likely to cover actual stories like crimes, arrests, police intervention, or may be narrations of other stories. They give specific details like age, names, etc. Satire tend to focus on political figures, or general humorous remarks on everyday things. Satire headlines also tend to generalize and be less specific to people or events, except for politics (wordclouds).
- Real news use more nouns than satire, mostly because of availability of information. Satire can be based on illegitimate sources or made-up stories.
- Real news have several keyword combinations even though theme remains of arrests and police. Satire has less, but more frequently used keyword combinations. Satire also uses strong attention-seeking keywords.

- From co-occurrences plots, we noticed that real news give more context and details. Satire news again revolves around the same topics – denoted by very thick joining lines.
- Real news topics are centered around crimes and arrests; satire around politics and generic humor.

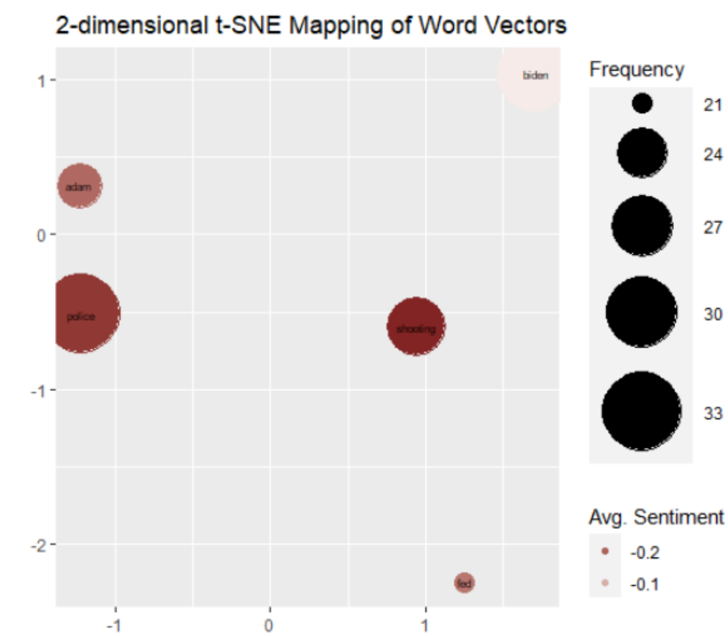
Applying observations to real Google headlines:

From our given dataset, we have made several observations about the differences in real versus satirical news headlines. Now, we run the same analyses on our fetched data from Google news and see which of our observations match with our perceived notions about real headlines. Because we have explained our analysis earlier, we will run them again quickly and without too much explanation.

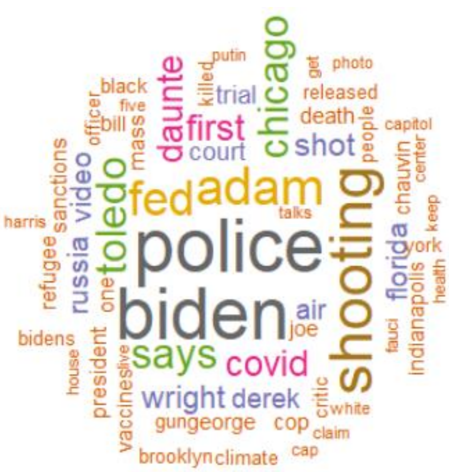


From the boxplot of sentiment scores from Google headlines on the left, we can see that the average sentiment score is negative. This is **in-line** with our observation in part 1. Google news is a legitimate source and like our previous set of real news, they may also be sharing actual news which we have seen to tend to be negative.

Average length of headline is 12.7 – closer to our original real.



The t-SNE plot on the right for Google headlines display only a few circles. This is because the dataset was small. Important things to note are that the sentiments are negative as observed from boxplot too. And, the most frequently appearing words are ‘Police’ and ‘Shooting’. This definitely rings a bell. These are the same kinds of words our given dataset’s real headlines generally talked about.

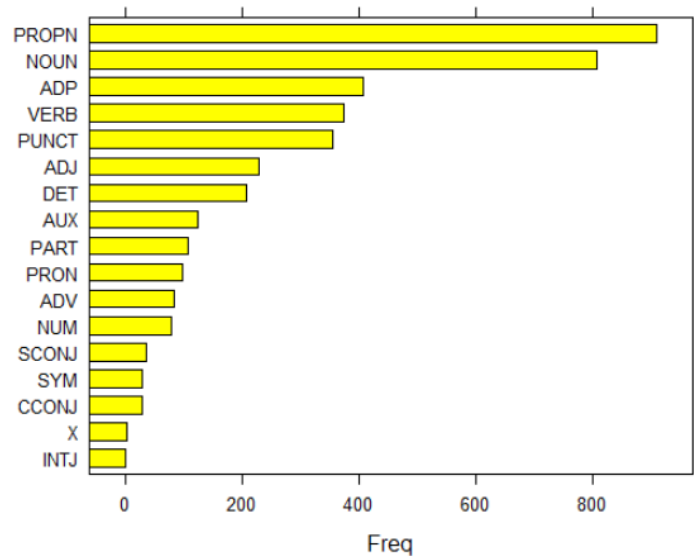


To further confirm our results, we plot a wordcloud of words for the Google headlines. Police and shootings again come out as 2 of most used. Biden is also used a lot as in reference to the President. ‘Adam’ and ‘Chicago’ are mentioned a few times. This is because of a recent killing incident of a teenager Adam Taledo by the Chicago Police.

An interesting fact is that this wordcloud also uses ‘says’ a lot. This is again in line with our observation from part 4 about real news narrating from others. Also, it mentions Covid too which is expected.

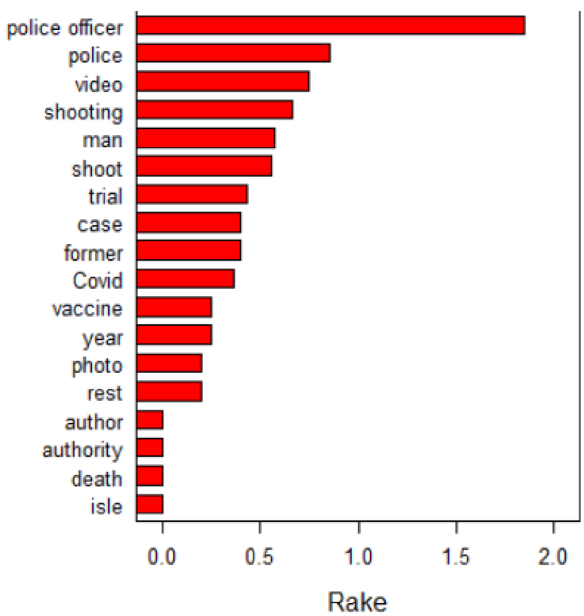
The bar graph of most frequently used Universal Parts of Speech (UPOS) for the google news headlines (on the right) shows a high usage of nouns. This is again similar to our observation in part 5 – real news tend to use more nouns. Since they are real, they have the actual information (names, etc.) to share with the audience.

UPOS (Universal Parts of Speech) frequency of occurrence



Other plots for specific parts of speech are not shown as they did not add much value, also because the counts did not vary much owing to a small dataset. They are attached in the appendix nonetheless.

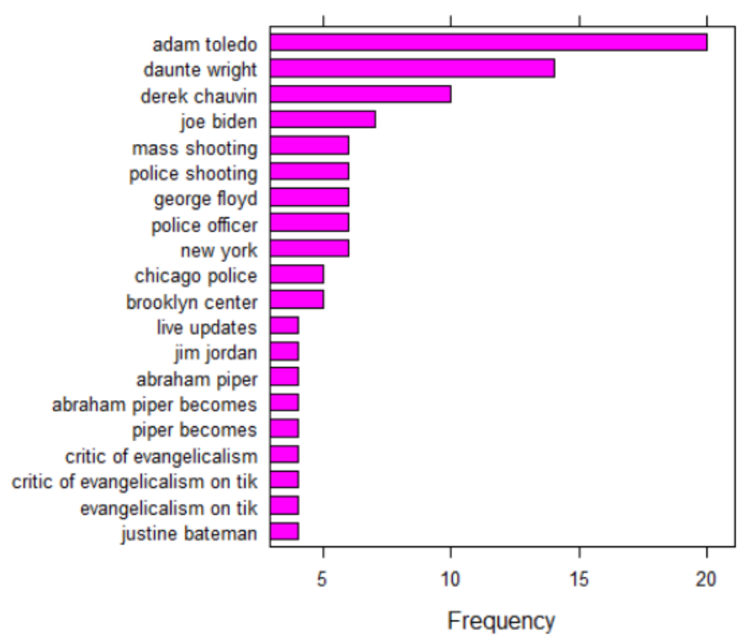
Keywords identified by RAKE



The RAKE extraction of keywords shows frequent usage of keywords ‘Police officer’, and other keywords related to the same theme. We can clearly see the dominance of news headlines by criminal incidences, or those involving police somehow.

The RAKE plot is different in this case from our earlier plot of real data. This seems to again be a result of small dataset. Since this data was pulled in 4-5 days, the same major incident still seems to dominate the news.

Keywords -simple noun phrases

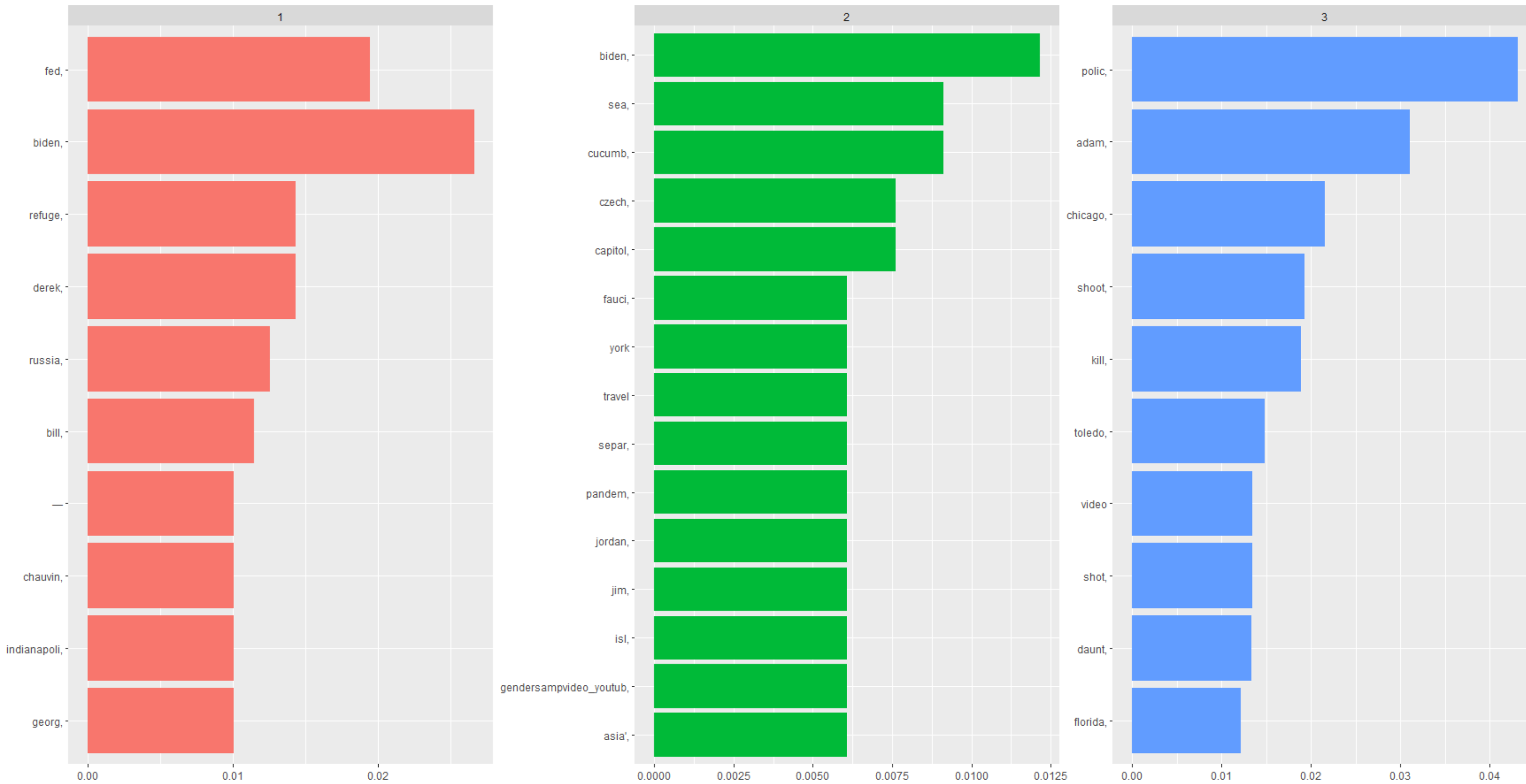


However, a plot of simple noun phrases keywords from google headlines shows most frequent mention of two individuals killed by police, Adam and Daunte. Although, it is a very specific instance, we can still conclude that the news covered follows the same theme of crimes, police, etc.

The co-occurrences plot with 3-word distance for google did not yield reliable results – probably due to small dataset not allowing enough sense to be made out of it. it can still be found in appendix.

Finally, we use the LDA technique to identify topics discussed in the google news headlines. We managed to get 3 topics as seen below. First focuses on FedEx shooting and other political news related to Biden and bill passed. The second emphasizes on Covid and its associated topics like Dr. Fauci and travelling. The third is solely about the killings of Adam and Daunte by the police.

Important takeaway is that 2 out 3 focus on crimes, killings, and police – a recurrent theme in real news headlines.



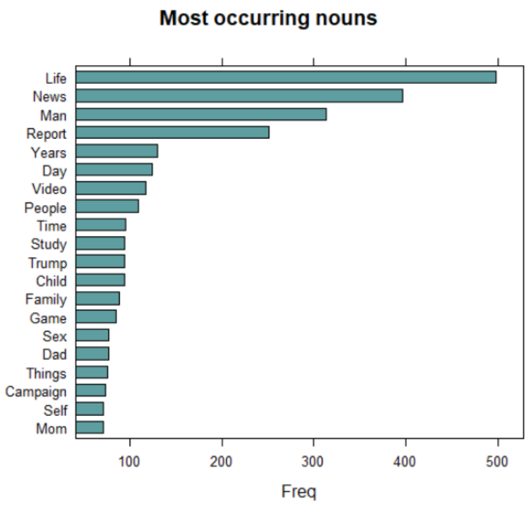
Conclusion:

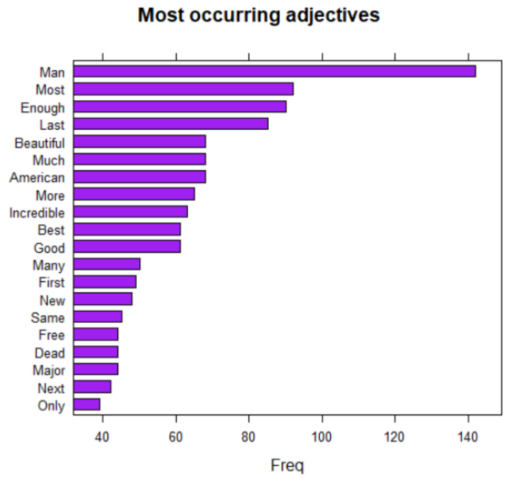
In this project, we conducted several text mining analyses and topic modeling techniques to distinguish real news headlines from satire. We then validated our observations and conclusions using a new real news dataset from google. We observed that almost all of our analyses on google headlines followed the patterns observed from the given real headlines data. The topics discussed, areas of focus, sentiments expressed in google news was in line with the real headlines from the given dataset. We can safely conclude that our observations allowed us to successfully identify differences between real and satire, and commonalities between two sets of real news headlines.

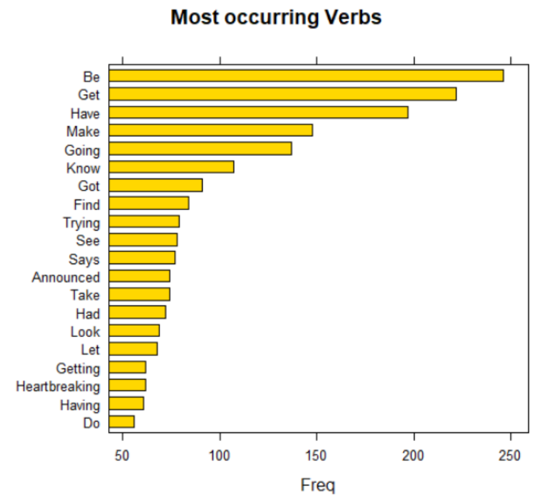
An extension to this project would be to repeat this for a bigger dataset from Google news fetched over a longer period of time to allow multiple global events to take place rather than just one. The high variety of data will lead to more robust solutions.

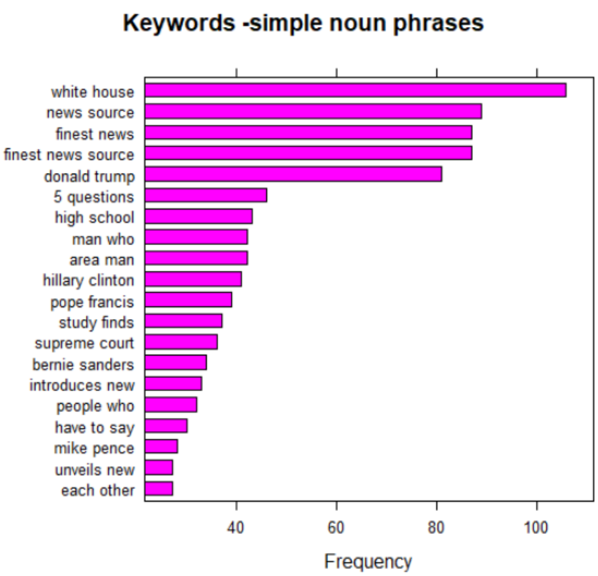
Appendix:

Satire news extra graphs

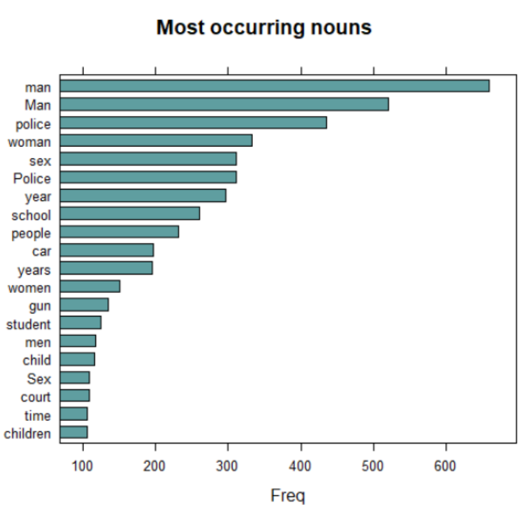


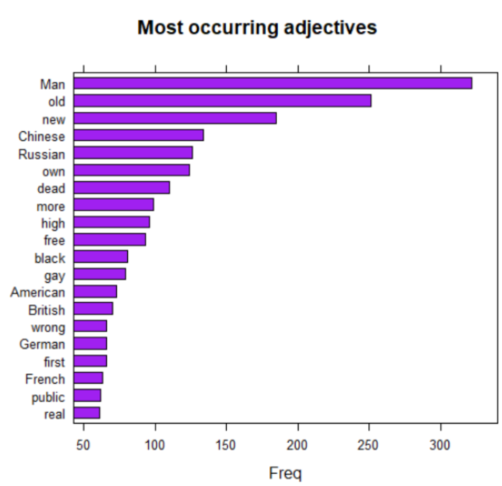


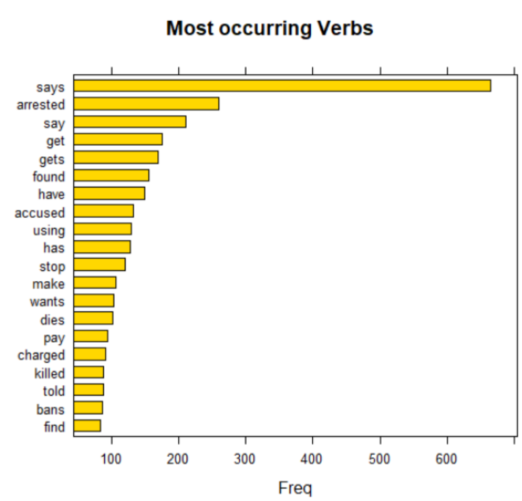


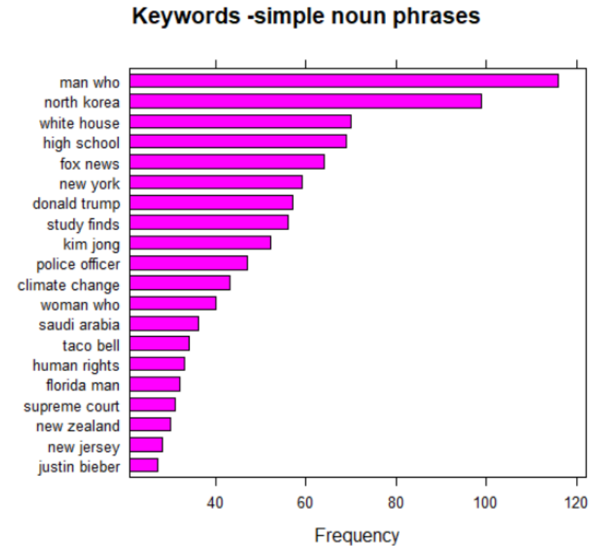


Real news extra graphs

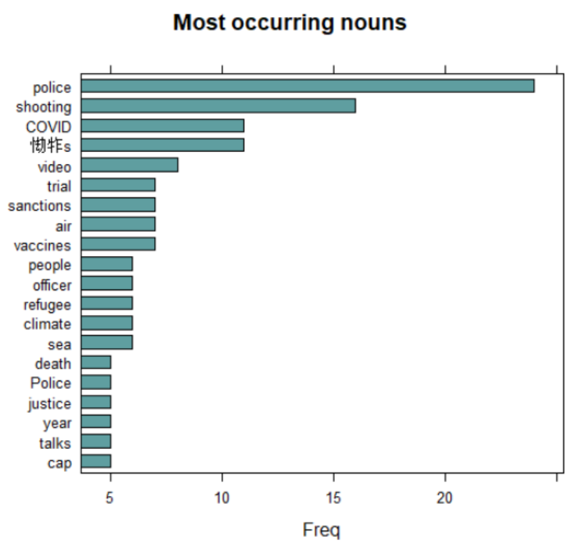


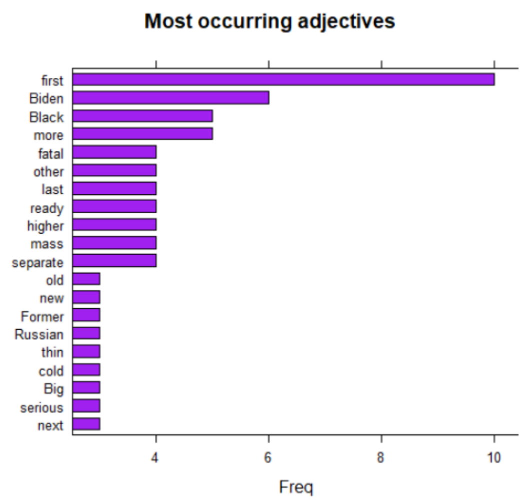


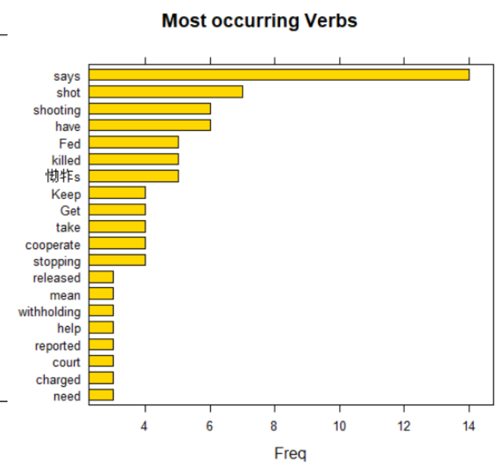


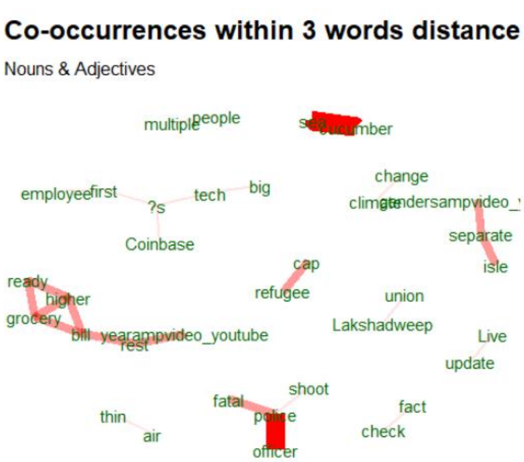


Google news extra graphs









:

CODE:

```
# EXTRACTING HEADLINES FROM GOOGLE

pacman::p_load(rvest, dplyr, stringr)

# extracting the whole news website page

google <-
read_html("https://news.google.com/")

# Step 3–extract the headlines and clean
using a regular expression in one step

# extracting the headlines and using stringr
for cleaning

# str_split("(?<=[a-z0-9aeiou!?\\.])(?=[A-Z])")
# for Google News in US Eng

#headline_all <-google %>%
html_nodes("article") %>% html_text("span")
%>% str_split("(?<=[a-z0-9!?\\.])(?=[A-Z])")

headline_all <-google %>%
html_nodes("article") %>% html_text("span")
%>% str_split("(?<=[a-z0-9aeiou!?\\.])(?=[A-
Z])")

# get only the headline title which is the first
element

headline_all <-sapply(headline_all, function(x)
x[1])

google_headlines<-data.frame(headlines=
headline_all, stringsAsFactors = F)

str(google_headlines)

write.csv(google_headlines, file="C:/RIT
Courses/MKTG 768 - Marketing
Analytics/Assignment
2/google_news_headlines v5.csv")

# -------------------------------------

# SENTENCE LEVEL SENTIMENTS USING
SENTIMENTR, PLOTTING AGAINST LABEL.

pacman::p_load(tidyr, dplyr, stringr,
data.table, sentimentr, ggplot2)

given_all = read.csv(file.choose(),
stringsAsFactors = F)

# create a rowid for the reviews

given_df <-given_all %>% mutate(id =
row_number())

# examine structure

str(given_all)

head(given_df)

# Step 2-define the lexicon and any changes
needed for our context

#get n rows-to see what we have in the
lexicon -

# Tyler Rinker is the author of sentimentr

nrow(lexicon::hash_sentiment_jockers_rinker
)#seems like 11,710 words

# words to replace-in this example, there are
switch, brand names etc.

replace_in_lexicon <-tribble(

  ~x, ~y,

  "switch", 0,      # not in dictionary

  "nintendo", 0,      # not in dictionary

  "red", 0,         # original score: -.6

  "amazeballs", .75, # not in dictionary

)

# create a new lexicon with modified
sentiment

review_lexicon <-
lexicon::hash_sentiment_jockers_rinker %>%

  filter(!x %in% replace_in_lexicon$x) %>%

  bind_rows(replace_in_lexicon) %>%

  setDT() %>%

  setkey("x")

# Step 3-start by getting the sentence level
sentiment for testing

# get sentence-level sentiment

sent_df <-given_df %>%

  get_sentences() %>%

  sentiment_by(by = c('id', 'label'),polarity_dt
= review_lexicon)

# Step 4-start by getting the sentence level
sentiment for testing

# check the relationship between star rating
and sentiment

ggplot(sent_df, aes(x = label, y =
ave_sentiment, color = factor(label), group =
label)) +

  geom_boxplot() +

  geom_hline(yintercept=0,
linetype="dashed", color = "red") +

  geom_text(aes(2.2, -0.07, label = "Neutral
Sentiment", vjust = 0, hjust=1), size = 3.5,
color = "red") +

  guides(color = guide_legend(title="Real(0)
Vs Satire(1)")) +

  ylab("Average Sentiment") +

  xlab("Real(0) Vs Satire(1)") +

  ggtitle("Avg.Sentiment - Real VS Satire
Headlines") +

  theme(plot.title = element_text(hjust = 0.5))

head(sent_df)

# ------------------------

# SEPARATE REAL VS SATIRE SENTIMENT DF

satire_df <- given_df %>% filter(label == 1)

head(satire_df)
```

```
real_df <- given_df %>% filter(label == 0)

head(real_df)


satire_sent_df <-satire_df %>%

  get_sentences() %>%

  sentiment_by(by = c('id', 'label'),polarity_dt
= review_lexicon)


satire_counts <- satire_sent_df %>%
summarise(mean = mean(word_count))

satire_counts


real_sent_df <-real_df %>%

  get_sentences() %>%

  sentiment_by(by = c('id', 'label'),polarity_dt
= review_lexicon)


real_counts <- real_sent_df %>%
summarise(mean = mean(word_count))

real_counts


#satire_counts <- satire_df %>%
group_by('id') %>% mutate(w_count =
str_count(satire_df$text, '\\w+'))

#head(satire_counts)


pacman::p_load(tidyr, dplyr, stringr,
data.table, sentimentr, ggplot2, text2vec, tm,
ggrepel)


glove_func <- function(main_df,
sentiment_df)

{

  tokens <-space_tokenizer(main_df$text
%>%tolower() %>%removePunctuation())


  it <-itoken(tokens, progressbar = FALSE)

  vocab <-create_vocabulary(it)


  vocab <- prune_vocabulary(vocab,
term_count_min = 3L)

  vectorizer <- vocab_vectorizer(vocab)


  tcm <- create_tcm(it, vectorizer,
skip_grams_window = 5L)


  glove = GloVe$new(rank = 100, x_max = 5)


  glove$fit_transform(tcm, n_iter = 20)



  word_vectors = glove$components


  #nintendo<-word_vectors[, "nintendo", drop
= F]


  #cos_sim = sim2(x = t(word_vectors), y =
t(nintendo), method = "cosine", norm = "l2")


  #head(sort(cos_sim[,1], decreasing = TRUE),
10)



  pacman::p_load(tm, Rtsne, tibble, tidytext,
scales)


  keep_words <-
setdiff(colnames(word_vectors), stopwords())


  word_vec <- word_vectors[, keep_words]


  train_df <-data.frame(t(word_vec)) %>%
rownames_to_column("word")


  tsne <-Rtsne(train_df[,-1], dims = 2,
perplexity = 50, verbose=TRUE, max_iter =
500)



  colors =
rainbow(length(unique(train_df$word)))

  names(colors) = unique(train_df$word)

  plot_df <-data.frame(tsne$Y) %>%

    mutate(word = train_df$word,col =
colors[train_df$word]) %>%

    left_join(vocab, by = c("word" = "term"))
%>%

    filter(doc_count >= 20)



  ggplot(plot_df, aes(X1, X2)) +

    geom_text(aes(X1, X2, label = word, color =
col), size = 3) +

    xlab("") +

    ylab("") +

    theme(legend.position = "none")




  word_sent <-main_df %>%

    left_join(sentiment_df, by = "id") %>%

    select(id, text, ave_sentiment) %>%

    unnest_tokens(word, text) %>%

    group_by(word) %>%

    summarise(count = n(),avg_sentiment =
mean(ave_sentiment),sum_sentiment =
sum(ave_sentiment),sd_sentiment =
sd(ave_sentiment)) %>%

    # remove stop words

    anti_join(stop_words, by = "word")

  # filter to words that appear at least 5 times

  pd_sent <-plot_df %>%

    left_join(word_sent, by = "word") %>%

    drop_na() %>%filter(count >= 5)



  ggplot(pd_sent, aes(X1, X2)) +

    geom_point(aes(X1, X2, size = count, alpha
= .1, color = avg_sentiment)) +

    geom_text(aes(X1, X2, label = word), size =
2) +
```

```r
   scale_colour_gradient2(low = muted("red"), mid = "white",high = muted("blue"), midpoint = 0) +

   scale_size(range = c(5, 20)) +xlab("") +

   ylab("") +

   ggtitle("2-dimensional t-SNE Mapping of Word Vectors") +

   guides(color = guide_legend(title="Avg. Sentiment"), size = guide_legend(title = "Frequency"), alpha = NULL) +

   scale_alpha(range = c(1, 1), guide = "none")

}


glove_func(real_df, real_sent_df)


# -------------------------------------


# FREQUENCIES AND WORDCLOUDS


pacman::p_load(dplyr, ggplot2, tidytext, wordcloud, wordcloud2, tm, RColorBrewer)


text <- satire_df$text


# Create a corpus

docs <- Corpus(VectorSource(text))


# clean data

docs <- docs %>%

  tm_map(removeNumbers) %>%

  tm_map(removePunctuation) %>%

  tm_map(stripWhitespace)

docs <- tm_map(docs, content_transformer(tolower))

docs <- tm_map(docs, removeWords, c(stopwords("english"), "new", "news", "trump", "onion", "can", "man", "just", "life", "woman",

                           "will", "women"))


#undesirable_words <-c("new", "news", "trump", "onion", "can")

#docs <- docs %>% filter(!word %in% undesirable_words)



dtm <- TermDocumentMatrix(docs)

matrix <- as.matrix(dtm)

words <- sort(rowSums(matrix),decreasing=TRUE)

df <- data.frame(word = names(words),freq=words)


set.seed(1234) # for reproducibility

wordcloud(words = df$word, freq = df$freq, scale=c(2,.3), min.freq = 20,

          max.words=100, random.order=FALSE, rot.per=0.35,

          colors=brewer.pal(8, "Dark2"))



# -----------------------------------


# TEXT MINING USING UDPIPE AND RAKE


pacman::p_load(dplyr, ggplot2, stringr, udpipe, lattice)


udmodel_english <-udpipe_load_model(file = "C:/RIT Courses/MKTG 768 - Marketing Analytics/Week 12/english-ewt-ud-2.5-191206.udpipe")


# Step 4–use udpipe to annotate the text in the headlines for 2016 and load into a frame

# this may take a while depending on how much data you are analyzing.

s <-udpipe_annotate(udmodel_english, satire_df$text) # satire

s2 <- udpipe_annotate(udmodel_english, real_df$text)


x <-data.frame(s2)

# Step 5–extract and display frequencies for universal parts of speech (upos) in text

stats <-txt_freq(x$upos)

stats$key <-factor(stats$key, levels = rev(stats$key))

barchart(key ~ freq, data = stats, col = "yellow", main = "UPOS (Universal Parts of Speech)\n frequency of occurrence", xlab = "Freq")



# Step 5–extract and display most occurring nouns in the headlines#

# NOUNS

stats <-subset(x, upos %in% c("NOUN"))

stats <-txt_freq(stats$token)

stats$key <-factor(stats$key, levels = rev(stats$key))

barchart(key ~ freq, data = head(stats, 20), col = "cadetblue", main = "Most occurring nouns", xlab = "Freq")



# Step 6–extract and displaymost occurring adjectives in the headlines#

# ADJECTIVES

stats <-subset(x, upos %in% c("ADJ"))

stats <-txt_freq(stats$token)

stats$key <-factor(stats$key, levels = rev(stats$key))

barchart(key ~ freq, data = head(stats, 20), col = "purple", main = "Most occurring adjectives", xlab = "Freq")



# Step 7–extract and display most occurring verbs in the headlines#

# VERBS

stats <-subset(x, upos %in% c("VERB"))

stats <-txt_freq(stats$token)

stats$key <-factor(stats$key, levels = rev(stats$key))

barchart(key ~ freq, data = head(stats, 20), col = "gold", main = "Most occurring Verbs", xlab = "Freq")
```

Step 8–finally use RAKE (Rapid Automatic Keyword Extraction algorithm) to

determine key phrases in a body of text by analyzing the frequency of word appearance

and its co-occurrence with other words in the text.#

RAKE

stats <-keywords_rake(x = x, term = "lemma", group = "doc_id",

relevant = x$upos %in% c("NOUN", "ADJ"))

stats$key <-factor(stats$keyword, levels = rev(stats$keyword))

barchart(key ~ rake, data = head(subset(stats, freq > 3), 20), col = "red", main = "Keywords identified by RAKE", xlab = "Rake")

Step 9–In English (and in many other languages a phrase can be formed simply with a

noun and a verb (e.g., cat meows) This may be useful for understanding context of a

sentence or a review or headlines especially if they are clickbait like. This step is to just

extract top phrases that are basically keyword-topics.#

display by plota sequence of POS tags (noun phrases / verb phrases)

x$phrase_tag <-as_phrasemachine(x$upos, type = "upos")

stats <-keywords_phrases(x = x$phrase_tag, term = tolower(x$token),

pattern = "(A|N)*N(P+D*(A|N)*N)*", is_regex = TRUE, detailed = FALSE)

stats <-subset(stats, ngram > 1 & freq > 3)

stats$key <-factor(stats$keyword, levels = rev(stats$keyword))

barchart(key ~ freq, data = head(stats, 20), col = "magenta", main = "Keywords -simple noun phrases", xlab = "Frequency")

Step 10–it would be helpful to explore the words that appear next to each other. We can

do this with just nouns and adjectives to explore

the patterns to get focus topic areas. #Adjust the ngram max levels if needed.It issetto4 to indicate that we want

co-occurrences within 3 words of each other.#

Collocation identification –basically words following one another)

stats<-keywords_collocation(x = x, term = "token", group = c("doc_id", "paragraph_id", "sentence_id"), ngram_max = 4)

How frequentlydo words occur in the same sentence(nouns and adjectives)

stats <-cooccurrence(x = subset(x, upos %in% c("NOUN", "ADJ")), term = "lemma", group = c("doc_id", "paragraph_id", "sentence_id"))

Co-occurrences: How frequent do words follow one another

stats <-cooccurrence(x = x$lemma, relevant = x$upos %in% c("NOUN", "ADJ"))#

Co-occurrences: How frequent do words follow one another even if we would

skip 2 words in between. You can adjust this if you need to.

stats <-cooccurrence(x = x$lemma, relevant = x$upos %in% c("NOUN", "ADJ"), skipgram = 2)

head(stats)

pacman::p_load(igraph, ggraph)

wordnetwork <-head(stats, 25)

wordnetwork <- graph_from_data_frame(wordnetwork)

ggraph(wordnetwork, layout = "fr") +

geom_edge_link(aes(width = cooc, edge_alpha = cooc), edge_colour = "red") +

geom_node_text(aes(label = name), col = "darkgreen", size = 4) +

theme_graph(base_family = "Arial Narrow") +

theme(legend.position = "none") +

labs(title = "Co-occurrences within 3 words distance", subtitle = "Nouns & Adjectives")

Topic modeling

library(tidyverse) # organize workflow and for all text work

library(tidytext) # contains the NLP methodswe need

library(topicmodels) # for LDA topic modelling–our main package

library(tm) # general text mining functions, DTM work.

library(SnowballC) # for stemmingwhen needed.

library(stringr) # for cleaning

#using read_csv instead of read.csv to avoid the stringAsFactors issue

#read_csv is also known to be faster at reading large datasets

reviews <- real_df #load the ChicagoReviews2kAirBnB.csv file

#Step 2 –Clean the data. This is an important step, check column names for your dataset

#clean the review data, our reviews are in the 'text' column of the dataset

reviews$text <- str_replace_all(reviews$text,"[^[:graph:]]", " ")

top_terms_by_topic_LDA <- function(input_text, # should be a column from a data frame

plot = T, # return a plot? TRUE by defult

number_of_topics = 4) # number of topics (4 by default)

{

create a corpus (type of object expected by tm) and document term matrix

Corpus <- Corpus(VectorSource(input_text))

```
  # make a corpus object

  DTM <- DocumentTermMatrix(Corpus)

  # get the count of words/document

  # remove any empty rows in our document term matrix (if there are any

  # we'll get an error when we try to run our LDA)

  unique_indexes <- unique(DTM$i)

  # get the index of each unique value

  DTM <- DTM[unique_indexes,]

  # get a subset of only those indexes

  # preform LDA & get the words/topic in a tidy text format

  lda <- LDA(DTM, k = number_of_topics, control = list(seed = 1234))

  topics <- tidy(lda, matrix = "beta")

  # get the top ten terms for each topic,

  # yes I made up the word informativeness

  top_terms <- topics  %>% # take the topics data frame and..

    group_by(topic) %>% # treat each topic as a different group

    top_n(10, beta) %>% # get the top 10 most informative words

    ungroup() %>% # ungroup

    arrange(topic, -beta) # arrange words in descending informativeness

  # if the user asks for a plot (TRUE by default)

  if(plot == T){

    # plot the top ten terms for each topic in order

    top_terms %>% # take the top terms

      mutate(term = reorder(term, beta)) %>% # sort terms by beta value

      ggplot(aes(term, beta, fill = factor(topic))) + # plot beta by theme

      geom_col(show.legend = FALSE) + # as a bar plot

      facet_wrap(~ topic, scales = "free") + # which each topic in a separate plot

      labs(x = NULL, y = "Beta") + # no x label, change y label

      coord_flip() # turn bars sideways

  }

  else{        # if the user does not request a plot

    # return a list of sorted terms instead

    return(top_terms)

  }

}


#Step 4– Test out the function to ensure everything works by starting with two topics.

# This step also allows you to identify any irrelevant words that may need to be

# eliminated and added to a stop word list.

# get just two topics to see how things pan out, carefully check words to see

# what needs to be added to stop words.

top_terms_by_topic_LDA(reviews$text, number_of_topics = 2)


#Step 5 – We are ready for the topic modeling now. Create a tidytext corpus

# Make a list of edited and customized stop words for our needs.

# pay attention to column names

reviewsCorpus <- Corpus(VectorSource(reviews$text))

reviewsDTM <- DocumentTermMatrix(reviewsCorpus)

# convert the document term matrix to a tidytext corpus

reviewsDTM_tidy <- tidy(reviewsDTM)


# Im going to add my own custom stop words that I don't think will be

# very informative in these reviews. My first test topic model indicated that

# I should add room and Chicago to the list. Case is not relevant

#custom_stop_words <- tibble(word = c("room", "chicago"))

# remove stopwords from the dataset of reviews

reviewsDTM_tidy_cleaned <- reviewsDTM_tidy %>% # take our tidy dtm and...

  anti_join(stop_words, by = c("term" = "word")) #%>% # remove English stopwords and...

  #anti_join(custom_stop_words, by = c("term" = "word")) # remove my custom stopwords


# reconstruct cleaned documents (so that each word shows up the correct number of times)

cleaned_documents <- reviewsDTM_tidy_cleaned %>%

  group_by(document) %>%

  mutate(terms = toString(rep(term, count))) %>%

  select(document, terms) %>%

  unique()


# check out what the cleaned documents look like (should just be a bunch of content words)

# in alphabetic order

head(cleaned_documents)


# Step 6 – Start obtaining topic models. If you know enough about the reviews you will

# have a great starting point in the number of topics sought, otherwise we will have to

# make several models.

#try out two topics, expand to 3 or 4 or more. I found three topics to be

# ideal in this specific 2k review set but in the larger set I needed 4.

top_terms_by_topic_LDA(cleaned_documents, number_of_topics = 2)

top_terms_by_topic_LDA(cleaned_documents, number_of_topics = 3)
```

top_terms_by_topic_LDA(cleaned_documents, number_of_topics = 4)

Step 7 – Stemming is a controversial topic when it comes to topic models.

Some research indicates that stemming actually harms the creation of topic models,

but some data scientists claim that stemming increases interpretability.

stem the words (e.g. convert each word to its stem, where applicable)

reviewsDTM_tidy_cleaned <- reviewsDTM_tidy_cleaned %>%

  mutate(stem = wordStem(term))

reconstruct our documents

cleaned_documents <- reviewsDTM_tidy_cleaned %>%

  group_by(document) %>%

  mutate(terms = toString(rep(stem, count))) %>%

  select(document, terms) %>%

  unique()

#Step 8 – Revisit the topic models and create the stemmed word topic models.

#try out the lower end of what was acceptable from step 6. This was 3 for me

#I then tried 4 for the larger set which seemed to work well enough.

now let's look at the new most informative terms

top_terms_by_topic_LDA(cleaned_documents$terms, number_of_topics = 2)

top_terms_by_topic_LDA(cleaned_documents$terms, number_of_topics = 4)

EXTRACTING HEADLINES FROM GOOGLE

pacman::p_load(rvest, dplyr, stringr)

extracting the whole news website page

google <- read_html("https://news.google.com/")

Step 3–extract the headlines and clean using a regular expression in one step

extracting the headlines and using stringr for cleaning

str_split("(?<=[a-z0-9aeiou!?\\.])(?=[A-Z])") # for Google News in US Eng

#headline_all <-google %>% html_nodes("article") %>% html_text("span") %>% str_split("(?<=[a-z0-9!?\\.])(?=[A-Z])")

headline_all <-google %>% html_nodes("article") %>% html_text("span") %>% str_split("(?<=[a-z0-9aeiou!?\\.])(?=[A-Z])")

get only the headline title which is the first element

headline_all <-sapply(headline_all, function(x) x[1])

google_headlines<-data.frame(headlines= headline_all, stringsAsFactors = F)

str(google_headlines)

write.csv(google_headlines, file="C:/RIT Courses/MKTG 768 - Marketing Analytics/Assignment 2/google_news_headlines v5.csv")

SENTENCE LEVEL SENTIMENTS USING SENTIMENTR, PLOTTING AGAINST LABEL.

pacman::p_load(tidyr, dplyr, stringr, data.table, sentimentr, ggplot2)

google_all = read.csv(file.choose(), stringsAsFactors = F)

create a rowid for the reviews

google_df <-google_all %>% mutate(id = row_number())

examine structure

str(google_all)

tail(google_df)

Step 2-define the lexicon and any changes needed for our context

#get n rows-to see what we have in the lexicon -

Tyler Rinker is the author of sentimentr

nrow(lexicon::hash_sentiment_jockers_rinker)#seems like 11,710 words

words to replace-in this example, there are switch, brand names etc.

```
replace_in_lexicon <-tribble(
  ~x, ~y,
  "switch", 0,      # not in dictionary
  "nintendo", 0,     # not in dictionary
  "red", 0,        # original score: -.6
  "amazeballs", .75, # not in dictionary
)
```

create a new lexicon with modified sentiment

```
review_lexicon <- lexicon::hash_sentiment_jockers_rinker %>%
  filter(!x %in% replace_in_lexicon$x) %>%
  bind_rows(replace_in_lexicon) %>%
  setDT() %>%
  setkey("x")
```

Step 3-start by getting the sentence level sentiment for testing

```
# get sentence-level sentiment

sent_df <-google_df %>%

  get_sentences() %>%

  sentiment_by(by = c('id', 'label'),polarity_dt = review_lexicon)


google_count <- sent_df %>% summarise(mean = mean(word_count))

google_count


# Step 4-start by getting the sentence level sentiment for testing

# check the relationship between star rating and sentiment

ggplot(sent_df, aes(x = label, y = ave_sentiment, color = factor(label), group = label)) +

  geom_boxplot() +

  geom_hline(yintercept=0, linetype="dashed", color = "red") +

  geom_text(aes(2.2, -0.07, label = "Neutral Sentiment", vjust = 0, hjust=1), size = 3.5, color = "red") +

  #guides(color = guide_legend(title="Real(0) Vs Satire(1)")) +

  ylab("Average Sentiment") +

  xlab("Real Headlines") +

  ggtitle("Avg.Sentiment - Real Google Headlines") +

  theme(plot.title = element_text(hjust = 0.5))


head(sent_df)


# -------------------------


# SEPARATE REAL VS SATIRE SENTIMENT DF


satire_df <- given_df %>% filter(label == 1)

head(satire_df)


real_df <- given_df %>% filter(label == 0)

head(real_df)

satire_sent_df <-satire_df %>%

  get_sentences() %>%

  sentiment_by(by = c('id', 'label'),polarity_dt = review_lexicon)


real_sent_df <-real_df %>%

  get_sentences() %>%

  sentiment_by(by = c('id', 'label'),polarity_dt = review_lexicon)


pacman::p_load(tidyr, dplyr, stringr, data.table, sentimentr, ggplot2, text2vec, tm, ggrepel)


glove_func <- function(main_df, sentiment_df)

{

  tokens <-space_tokenizer(main_df$text %>%tolower() %>%removePunctuation())


  it <-itoken(tokens, progressbar = FALSE)

  vocab <-create_vocabulary(it)


  vocab <- prune_vocabulary(vocab, term_count_min = 3L)


  vectorizer <- vocab_vectorizer(vocab)


  tcm <- create_tcm(it, vectorizer, skip_grams_window = 5L)


  glove = GloVe$new(rank = 100, x_max = 5)


  glove$fit_transform(tcm, n_iter = 20)


  word_vectors = glove$components


  #nintendo<-word_vectors[, "nintendo", drop = F]

  #cos_sim = sim2(x = t(word_vectors), y = t(nintendo), method = "cosine", norm = "l2")


  #head(sort(cos_sim[,1], decreasing = TRUE), 10)


  pacman::p_load(tm, Rtsne, tibble, tidytext, scales)


  keep_words <- setdiff(colnames(word_vectors), stopwords())


  word_vec <- word_vectors[, keep_words]


  train_df <-data.frame(t(word_vec)) %>% rownames_to_column("word")


  tsne <-Rtsne(train_df[,-1], dims = 2, perplexity = 50, verbose=TRUE, max_iter = 500)


  colors = rainbow(length(unique(train_df$word)))

  names(colors) = unique(train_df$word)


  plot_df <-data.frame(tsne$Y) %>%

    mutate(word = train_df$word,col = colors[train_df$word]) %>%

    left_join(vocab, by = c("word" = "term")) %>%

    filter(doc_count >= 20)


  ggplot(plot_df, aes(X1, X2)) +

    geom_text(aes(X1, X2, label = word, color = col), size = 3) +

    xlab("") +

    ylab("") +

    theme(legend.position = "none")
```

```
  word_sent <-main_df %>%

    left_join(sentiment_df, by = "id") %>%

    select(id, text, ave_sentiment) %>%

    unnest_tokens(word, text) %>%

    group_by(word) %>%

    summarise(count = n(),avg_sentiment = mean(ave_sentiment),sum_sentiment = sum(ave_sentiment),sd_sentiment = sd(ave_sentiment)) %>%

    # remove stop words

    anti_join(stop_words, by = "word")

  # filter to words that appear at least 5 times

  pd_sent <-plot_df %>%

    left_join(word_sent, by = "word") %>%

    drop_na() %>%filter(count >= 5)




  ggplot(pd_sent, aes(X1, X2)) +

    geom_point(aes(X1, X2, size = count, alpha = .1, color = avg_sentiment)) +

    geom_text(aes(X1, X2, label = word), size = 2) +

    scale_colour_gradient2(low = muted("red"), mid = "white",high = muted("blue"), midpoint = 0) +

    scale_size(range = c(5, 20)) +xlab("") +

    ylab("") +

    ggtitle("2-dimensional t-SNE Mapping of Word Vectors") +

    guides(color = guide_legend(title="Avg. Sentiment"), size = guide_legend(title = "Frequency"), alpha = NULL) +

    scale_alpha(range = c(1, 1), guide = "none")

}


glove_func(google_df, sent_df)



# ------------------------------------

# FREQUENCIES AND WORDCLOUDS


pacman::p_load(dplyr, ggplot2, tidytext, wordcloud, wordcloud2, tm, RColorBrewer)


text <- google_df$text


# Create a corpus

docs <- Corpus(VectorSource(text))


# clean data

docs <- docs %>%

  tm_map(removeNumbers) %>%

  tm_map(removePunctuation) %>%

  tm_map(stripWhitespace)

docs <- tm_map(docs, content_transformer(tolower))

docs <- tm_map(docs, removeWords, c(stopwords("english"), "new", "news", "trump", "onion", "can", "man", "just", "life", "woman",

                              "will", "women", "sea", "cucumber", "ampvideo_youtube", "ampvideoyoutube"))


#undesirable_words <-c("new", "news", "trump", "onion", "can")

#docs <- docs %>% filter(!word %in% undesirable_words)



dtm <- TermDocumentMatrix(docs)

matrix <- as.matrix(dtm)

words <- sort(rowSums(matrix),decreasing=TRUE)

df <- data.frame(word = names(words),freq=words)


set.seed(1234) # for reproducibility

wordcloud(words = df$word, freq = df$freq, scale=c(3,.3), min.freq = 5,

          max.words=100, random.order=FALSE, rot.per=0.35,

          colors=brewer.pal(8, "Dark2"))


# -----------------------------------


# TEXT MINING USING UDPIPE AND RAKE


pacman::p_load(dplyr, ggplot2, stringr, udpipe, lattice)


udmodel_english <-udpipe_load_model(file = "C:/RIT Courses/MKTG 768 - Marketing Analytics/Week 12/english-ewt-ud-2.5-191206.udpipe")


# Step 4–use udpipe to annotate the text in the headlines for 2016 and load into a frame

# this may take a while depending on how much data you are analyzing.

s <-udpipe_annotate(udmodel_english, satire_df$text) # satire

s2 <- udpipe_annotate(udmodel_english, real_df$text) # real

s3 <- udpipe_annotate(udmodel_english, google_df$text)


x <-data.frame(s3)



# Step 5–extract and display frequencies for universal parts of speech (upos) in text

stats <-txt_freq(x$upos)

stats$key <-factor(stats$key, levels = rev(stats$key))

barchart(key ~ freq, data = stats, col = "yellow", main = "UPOS (Universal Parts of Speech)\n frequency of occurrence", xlab = "Freq")



# Step 5–extract and display most occurring nouns in the headlines#

# NOUNS

stats <-subset(x, upos %in% c("NOUN"))
```

```
stats <-txt_freq(stats$token)

stats$key <-factor(stats$key, levels = rev(stats$key))

barchart(key ~ freq, data = head(stats, 20), col = "cadetblue", main = "Most occurring nouns", xlab = "Freq")



# Step 6–extract and displaymost occurring adjectives in the headlines#

# ADJECTIVES

stats <-subset(x, upos %in% c("ADJ"))

stats <-txt_freq(stats$token)

stats$key <-factor(stats$key, levels = rev(stats$key))

barchart(key ~ freq, data = head(stats, 20), col = "purple", main = "Most occurring adjectives", xlab = "Freq")



# Step 7–extract and display most occurring verbs in the headlines#

# VERBS

stats <-subset(x, upos %in% c("VERB"))

stats <-txt_freq(stats$token)

stats$key <-factor(stats$key, levels = rev(stats$key))

barchart(key ~ freq, data = head(stats, 20), col = "gold", main = "Most occurring Verbs", xlab = "Freq")



# Step 8–finally use RAKE (Rapid Automatic Keyword Extraction algorithm) to

# determine key phrases in a body of text by analyzing the frequency of word appearance

# and its co-occurrence with other words in the text.#

# RAKE

stats <-keywords_rake(x = x, term = "lemma", group = "doc_id",

                  relevant = x$upos %in% c("NOUN", "ADJ"))

stats$key <-factor(stats$keyword, levels = rev(stats$keyword))

barchart(key ~ rake, data = head(subset(stats, freq > 3), 20), col = "red", main = "Keywords identified by RAKE", xlab = "Rake")



# Step 9–In English (and in many other languages a phrase can be formed simply with a

# noun and a verb (e.g., cat meows) This may be useful for understanding context of a

# sentence or a review or headlines especially if they are clickbait like. This step is to just

# extract top phrases that are basically keyword-topics.#


# display by plota sequence of POS tags (noun phrases / verb phrases)

x$phrase_tag <-as_phrasemachine(x$upos, type = "upos")

stats <-keywords_phrases(x = x$phrase_tag, term = tolower(x$token),

                  pattern = "(A|N)*N(P+D*(A|N)*N)*", is_regex = TRUE, detailed = FALSE)

stats <-subset(stats, ngram > 1 & freq > 3)

stats$key <-factor(stats$keyword, levels = rev(stats$keyword))

barchart(key ~ freq, data = head(stats, 20), col = "magenta", main = "Keywords -simple noun phrases", xlab = "Frequency")


# Step 10–it would be helpful to explore the words that appear next to each other. We can

# do this with just nouns and adjectives to explore

# the patterns to get focus topic areas. #Adjust the ngram max levels if needed.It issetto4 to indicate that we want

# co-occurrences within 3 words of each other.#


# Collocation identification –basically words following one another)

stats<-keywords_collocation(x = x, term = "token", group = c("doc_id", "paragraph_id", "sentence_id"), ngram_max = 4)

# How frequentlydo words occur in the same sentence(nouns and adjectives)

stats <-cooccurrence(x = subset(x, upos %in% c("NOUN", "ADJ")), term = "lemma", group = c("doc_id", "paragraph_id", "sentence_id"))


## Co-occurrences: How frequent do words follow one another

stats <-cooccurrence(x = x$lemma, relevant = x$upos %in% c("NOUN", "ADJ"))#

# Co-occurrences: How frequent do words follow one another even if we would #

# skip 2 words in between. You can adjust this if you need to.

stats <-cooccurrence(x = x$lemma, relevant = x$upos %in% c("NOUN", "ADJ"), skipgram = 2)

head(stats)



pacman::p_load(igraph, ggraph)

wordnetwork <-head(stats, 25)

wordnetwork <-graph_from_data_frame(wordnetwork)

ggraph(wordnetwork, layout = "fr") +

  geom_edge_link(aes(width = cooc, edge_alpha = cooc), edge_colour = "red") +

  geom_node_text(aes(label = name), col = "darkgreen", size = 4) +

  theme_graph(base_family = "Arial Narrow") +

  theme(legend.position = "none") +

  labs(title = "Co-occurrences within 3 words distance", subtitle = "Nouns & Adjectives")


# --------------------------------------------


# Topic modeling


library(tidyverse) # organize workflow and for all text work

library(tidytext) # contains the NLP methodswe need

library(topicmodels) # for LDA topic modelling–our main package
```

```
library(tm) # general text mining functions, DTM work.

library(SnowballC) # for stemmingwhen needed.

library(stringr) # for cleaning


#using read_csv instead of read.csv to avoid the stringAsFactors issue

#read_csv is also known to be faster at reading large datasets

reviews <- google_df #load the ChicagoReviews2kAirBnB.csv file


#Step 2 –Clean the data. This is an important step, check column names for your dataset

#clean the review data, our reviews are in the 'text' column of the dataset

reviews$text <- str_replace_all(reviews$text,"[^[:graph:]]", " ")


top_terms_by_topic_LDA <- function(input_text, # should be a column from a data frame

                              plot = T, # return a plot? TRUE by defult

                              number_of_topics = 4) # number of topics (4 by default)

{

  # create a corpus (type of object expected by tm) and document term matrix

  Corpus <- Corpus(VectorSource(input_text))

  # make a corpus object

  DTM <- DocumentTermMatrix(Corpus)

  # get the count of words/document

  # remove any empty rows in our document term matrix (if there are any

  # we'll get an error when we try to run our LDA)

  unique_indexes <- unique(DTM$i)

  # get the index of each unique value

  DTM <- DTM[unique_indexes,]

  # get a subset of only those indexes

  # preform LDA & get the words/topic in a tidy text format

  lda <- LDA(DTM, k = number_of_topics, control = list(seed = 1234))

  topics <- tidy(lda, matrix = "beta")

  # get the top ten terms for each topic,

  # yes I made up the word informativeness

  top_terms <- topics  %>% # take the topics data frame and..

    group_by(topic) %>% # treat each topic as a different group

    top_n(10, beta) %>% # get the top 10 most informative words

    ungroup() %>% # ungroup

    arrange(topic, -beta) # arrange words in descending informativeness

  # if the user asks for a plot (TRUE by default)

  if(plot == T){

    # plot the top ten terms for each topic in order

    top_terms %>% # take the top terms

      mutate(term = reorder(term, beta)) %>% # sort terms by beta value

      ggplot(aes(term, beta, fill = factor(topic))) + # plot beta by theme

      geom_col(show.legend = FALSE) + # as a bar plot

      facet_wrap(~ topic, scales = "free") + # which each topic in a separate plot

      labs(x = NULL, y = "Beta") + # no x label, change y label

      coord_flip() # turn bars sideways

  }

  else{        # if the user does not request a plot

    # return a list of sorted terms instead

    return(top_terms)

  }

}


#Step 4– Test out the function to ensure everything works by starting with two topics.

# This step also allows you to identify any irrelevant words that may need to be

# eliminated and added to a stop word list.

# get just two topics to see how things pan out, carefully check words to see

# what needs to be added to stop words.

top_terms_by_topic_LDA(reviews$text, number_of_topics = 2)


#Step 5 – We are ready for the topic modeling now. Create a tidytext corpus

# Make a list of edited and customized stop words for our needs.

# pay attention to column names

reviewsCorpus <- Corpus(VectorSource(reviews$text))

reviewsDTM <- DocumentTermMatrix(reviewsCorpus)

# convert the document term matrix to a tidytext corpus

reviewsDTM_tidy <- tidy(reviewsDTM)


# Im going to add my own custom stop words that I don't think will be

# very informative in these reviews. My first test topic model indicated that

# I should add room and Chicago to the list. Case is not relevant

#custom_stop_words <- tibble(word = c("room", "chicago"))


# remove stopwords from the dataset of reviews

reviewsDTM_tidy_cleaned <- reviewsDTM_tidy %>% # take our tidy dtm and...

  anti_join(stop_words, by = c("term" = "word")) #%>% # remove English stopwords and...

#anti_join(custom_stop_words, by = c("term" = "word")) # remove my custom stopwords


# reconstruct cleaned documents (so that each word shows up the correct number of times)
```

```
cleaned_documents <-
reviewsDTM_tidy_cleaned %>%

  group_by(document) %>%

  mutate(terms = toString(rep(term, count)))
%>%

  select(document, terms) %>%

  unique()




# check out what the cleaned documents look
like (should just be a bunch of content words)

# in alphabetic order

head(cleaned_documents)




# Step 6 – Start obtaining topic models. If you
know enough about the reviews you will

# have a great starting point in the number of
topics sought, otherwise we will have to

# make several models.


#try out two topics, expand to 3 or 4 or more.
I found three topics to be

# ideal in this specific 2k review set but in the
larger set I needed 4.

top_terms_by_topic_LDA(cleaned_document
s, number_of_topics = 2)

top_terms_by_topic_LDA(cleaned_document
s, number_of_topics = 3)

top_terms_by_topic_LDA(cleaned_document
s, number_of_topics = 4)




# Step 7 – Stemming is a controversial topic
when it comes to topic models.

# Some research indicates that stemming
actually harms the creation of topic models,

# but some data scientists claim that
stemming increases interpretability.




# stem the words (e.g. convert each word to
its stem, where applicable)

reviewsDTM_tidy_cleaned <-
reviewsDTM_tidy_cleaned %>%

  mutate(stem = wordStem(term))


# reconstruct our documents

cleaned_documents <-
reviewsDTM_tidy_cleaned %>%

  group_by(document) %>%

  mutate(terms = toString(rep(stem, count)))
%>%

  select(document, terms) %>%

  unique()


#Step 8 – Revisit the topic models and create
the stemmed word topic models.



#try out the lower end of what was
acceptable from step 6. This was 3 for me

#I then tried 4 for the larger set which
seemed to work well enough.

# now let's look at the new most informative
terms


top_terms_by_topic_LDA(cleaned_document
s$terms, number_of_topics = 2)

top_terms_by_topic_LDA(cleaned_document
s$terms, number_of_topics = 3)

top_terms_by_topic_LDA(cleaned_document
s$terms, number_of_topics = 4)
```