lab3a_estes

Andrew Estes

2/3/2022

Notes: To load in the data set, you'll have to use sep="^" since the columns are separated by tabs. Also, you'll want to make sure that you read the text columns in as character vectors, rather than as factors. You might make a"small" version of the data set to use while you're testing your code. Then once you know that your code runs, you can run it on the full data set. If you're writing your code by knitting a markdown file into HTML or PDF, you might investigate using the cache=TRUE option on specific code chunks to speed up knitting. That option means that the results of running that code chunk are saved, and the code is only re-run if you change something in the code chunk itself. Otherwise, the stored results are reloaded each time you re-knit the document. Make sure to think about pre-processing of the test and what sorts of words you want to remove before you run the analysis. With a data set of this size, some of my tuning code took a while to run, so you should be expecting that. But on my middle-of-the-road laptop, nothing took on the order of hours or days to complete. Feel free to adapt the code from my "headlines" example code, but please edit it to use only those parts you need, and remove any extraneous and perhaps too-verbose comments that you don't need. If you do adapt my code, it's good practice to add a note to that effect somewhere in your write-up.

Installing libraries, packages, and datasets

```
pacman::p_load(tidymodels,
               tidyverse,
               ggplot2,
               jsonlite,
               tidytext,
               sentimentr,
               wordcloud,
               LiblineaR,
               topicmodels,
               topicdoc,
               caret,
               igraph,
               ggraph,
               quanteda,
               SparseM,
               textdata,
               dplyr)
```

```
system.time({
#loading the data
df.original <- read.csv("pro_con.csv", sep="\t", stringsAsFactors = FALSE)
```

```
#copying dataframe, making Pro Con boolean
df <- df.original

#making Pro Con boolean
df <- df %>%
  mutate(ProCon = ifelse(df$type=="Pro", 1, 0))


#getting sentiment dictionary
#afinn <- get_sentiments("afinn")
#The AFINN lexicon assigns words with a score that runs between -5 and 5, with negative scores indicati

#bing <- get_sentiments("bing")
#The bing lexicon categorizes words in a binary fashion into positive and negative categories

nrc <- get_sentiments("nrc")
#The nrc lexicon categorizes words in a binary fashion ("yes"/"no") into categories of positive, negati

})
```

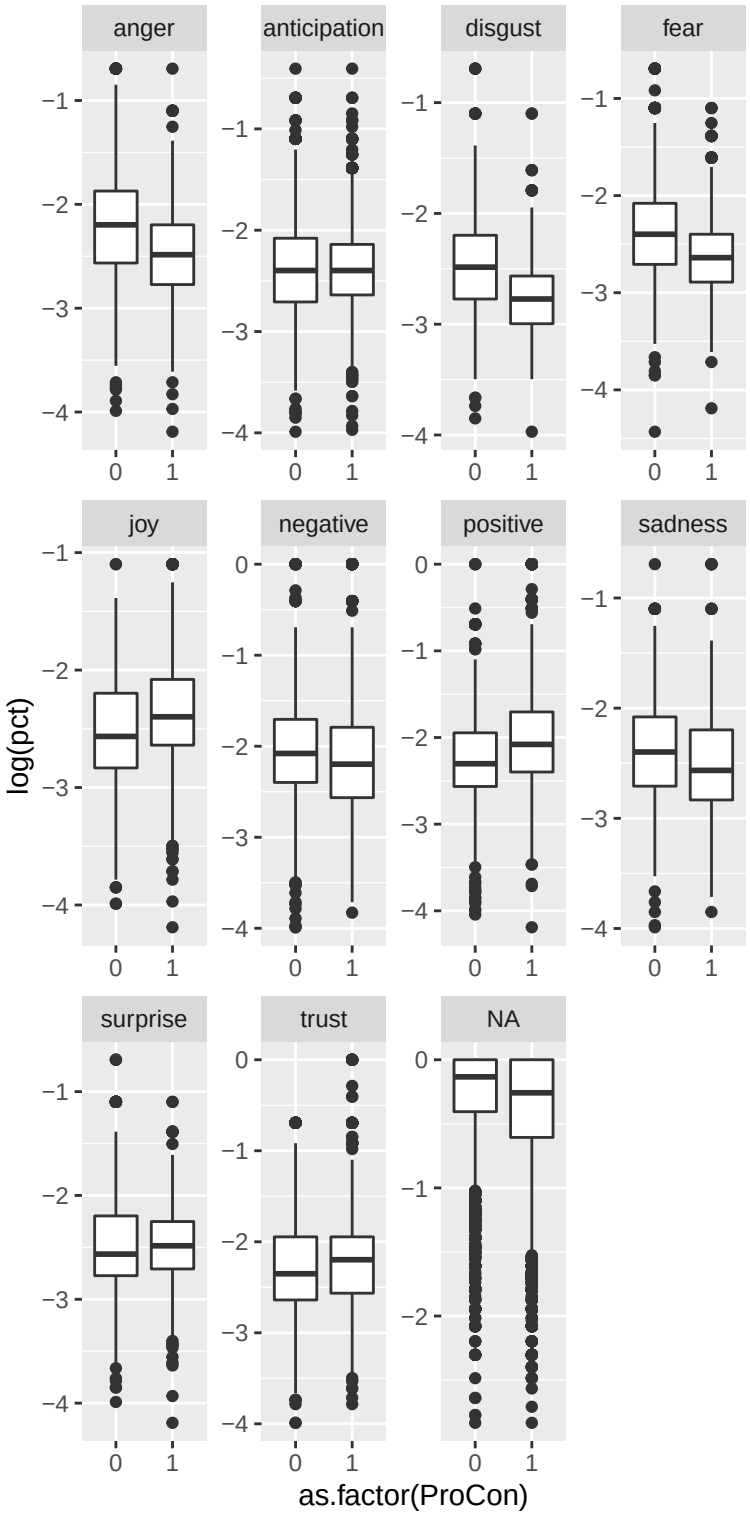
```
##    user  system elapsed
##    0.13    0.03    0.19
```

Part 1.

Calculate a single sentiment score for each review, and create a visualization that compares sentiment scores of the "Con" group to those of the "Pro" group. A) After creating that output, comment on whether there appears to be a significant difference in sentiment between the two groups.

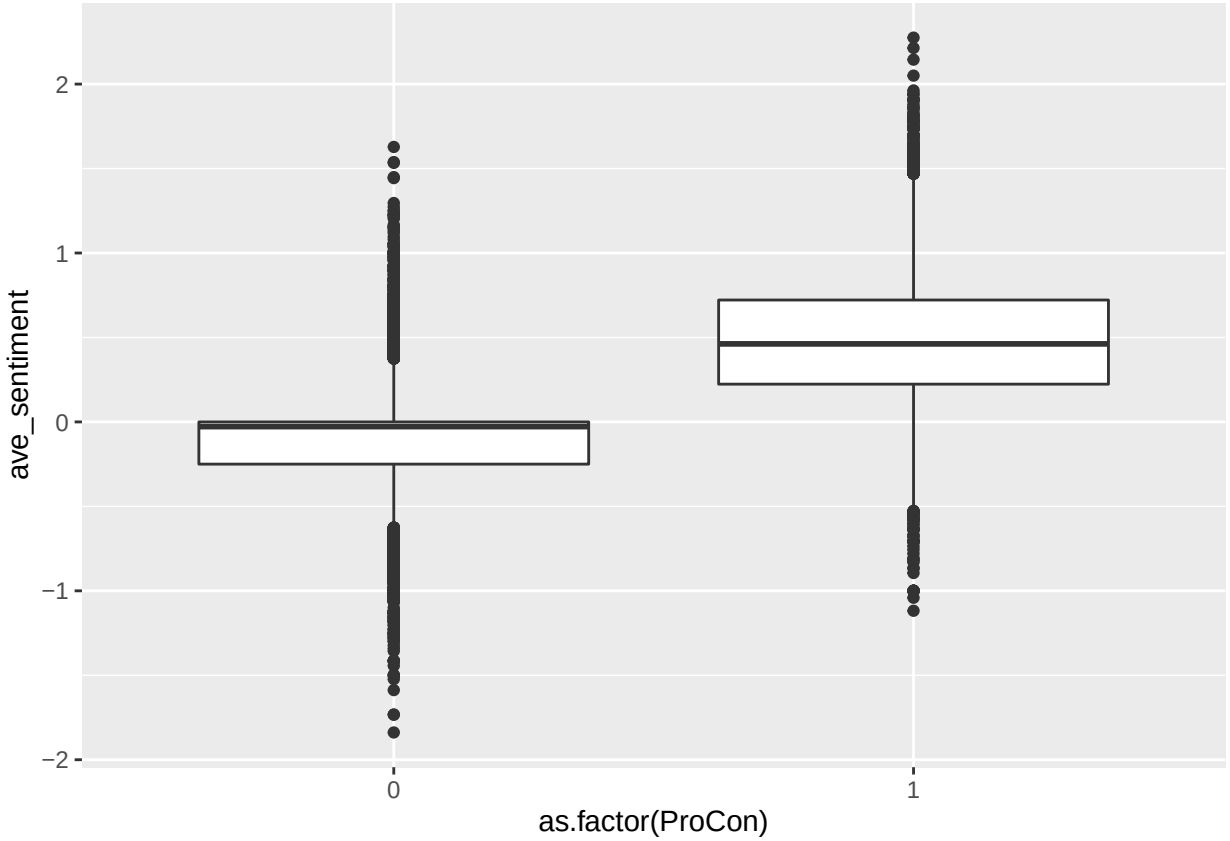
```
df.tidy <- df %>%
  unnest_tokens(output=word, input=review)

#df.afinn <- df.tidy %>% left_join(afinn) %>%
  #group_by(ProCon, id) %>%
  #summarize(n = n()) %>% mutate(pct = n/sum(n)) %>%
  #right_join(df)

#df.bing <- df.tidy %>% left_join(bing) %>%
  #group_by(id, sentiment, type) %>%
  #summarize(n = n()) %>% mutate(pct = n/sum(n)) %>%
  #right_join(df)

df.nrc <- df.tidy %>% left_join(nrc) %>%
  group_by(ProCon, id, sentiment) %>%
  summarize(n = n()) %>%
  mutate(pct = n/sum(n)) %>%
  ungroup() %>%
  right_join(df)
```

```
#gg.afinn <-
  #ggplot(df.afinn) +
  #geom_boxplot(aes(x=as.factor(ProCon), y=(pct))) +
  #facet_wrap(facets="type", scales="free")
#gg.afinn

#gg.bing <-
  #ggplot(df.bing) +
  #geom_boxplot(aes(x=as.factor(ProCon), y=log(pct))) +
  #facet_wrap(facets="sentiment", scales="free")
#gg.bing

gg.nrc <-
  ggplot(df.nrc) +
  geom_boxplot(aes(x=as.factor(ProCon), y=log(pct))) +
  facet_wrap(facets="sentiment", scales="free")
gg.nrc
```



There does not appear to be a significant difference between "pro" and "con" headlines using the NRC text library classification. There is still the possibility that the total average sentiment will show a distinction between "Pro" and "Con" so below uses the SentimentR package to find the average.

```
df.sent <- sentiment_by(df$review, by=df$id)

sent.gg <- df %>% left_join(df.sent) %>%
  ggplot() + geom_boxplot(aes(x=as.factor(ProCon), y=ave_sentiment))
sent.gg
```



It does appear that the average sentiment for a "Pro" review is .5 higher than a "Con" review. Interestingly enough, a "Con" review has a near zero mid-point.

B) Find the "Con" review with the highest sentiment score and the "Pro" review with the lowest sentiment score.

C) Print out each of the reviews you found in part (b), and comment on why you think they might have "fooled" the sentiment algorithm the way they did (if they did).

```
df.con <- df.nrc %>%
  filter(type == "Con") %>%
  arrange(desc(pct)) %>%
  na.omit()
head(df.con)
```

```
## # A tibble: 6 x 7
##   ProCon    id sentiment     n   pct type  review
##    <dbl> <int> <chr>     <int> <dbl> <chr> <chr>
## 1      0 23669 negative      1     1 Con   cheap
## 2      0 24758 positive      1     1 Con   Charger
## 3      0 25265 positive      3     1 Con   FOCUS, FOCUS, FOCUS
## 4      0 26402 negative      1     1 Con   Cheap
## 5      0 26578 negative      1     1 Con   cheap
## 6      0 26722 negative      3     1 Con   Noise, noise, noise
```

```
df.pro <- df.nrc %>%
  filter(type == "Pro") %>%
  arrange(desc(pct)) %>%
  na.omit()
head(df.pro)
```

```
## # A tibble: 6 x 7
##   ProCon    id sentiment     n   pct type  review
##    <dbl> <int> <chr>     <int> <dbl> <chr> <chr>
## 1      1    41 negative      1     1 Pro   cheap
## 2      1    88 positive      1     1 Pro   inexpensive
## 3      1   222 negative      2     1 Pro   Cheap, small.
## 4      1   234 negative      1     1 Pro   Cheap
## 5      1   248 negative      1     1 Pro   Cheap
## 6      1   356 negative      1     1 Pro   cheap
```

Both "Pro" and "Con" had similar reviews - mostly regarding the word "Cheap." That is understandable as a "cheap food" has a different connotation than "cheapskate." Another major issue is the lack of other words, only one review of the 12 listed had two or more distinct words. 9 of the 12 reviews were one-word responses. And the remaining two reviews just repeated the same word three times.

Part 2.

Create an SVM model that uses the text of a review to predict whether it is a positive ("Pro") review or negative ("Con") review. A) Create a training and test set from your data set of product reviews

```
# Set seed for reproducibility.
set.seed(68954)
split.train <- slice_sample(df, prop=0.7) %>% arrange(id)
split.test <- df %>% anti_join(split.train) %>% arrange(id)

# We'll remove symbols we don't want, as well as stop words. Then we'll stem
# the words.
split.train.tokens <- tokens(split.train$review, remove_punct=TRUE,
                      remove_symbols=TRUE, remove_numbers=TRUE) %>%
  tokens_remove(pattern=stopwords(language="en")) %>%
  tokens_wordstem()
# Create a DFM from the training set. Note: This matrix can be very large,
# and is created as a "sparse matrix" to save memory. Commands that try to
# transform it into a regular matrix might die for lack of memory.
split.train.dfm <- dfm(split.train.tokens)
# Reduce size of DFM by removing words that don't occur often
split.train.dfm <- dfm_trim(split.train.dfm, min_termfreq=5)
# Response variable.
split.train.y <- as.factor(split.train$type)


# Test set
split.test.tokens <- tokens(split.test$review, remove_punct=TRUE,
                      remove_symbols=TRUE, remove_numbers=TRUE) %>%
  tokens_remove(pattern=stopwords(language="en")) %>%
  tokens_wordstem()
split.test.dfm <- dfm(split.test.tokens) %>%
  dfm_match(featnames(split.train.dfm))

split.test.y <- as.factor(split.test$type)
```

B) Use cross-validation with the training set to do some model tuning for the cost tuning parameter of the LiblineR command. Plot a graph of accuracy vs. cost.

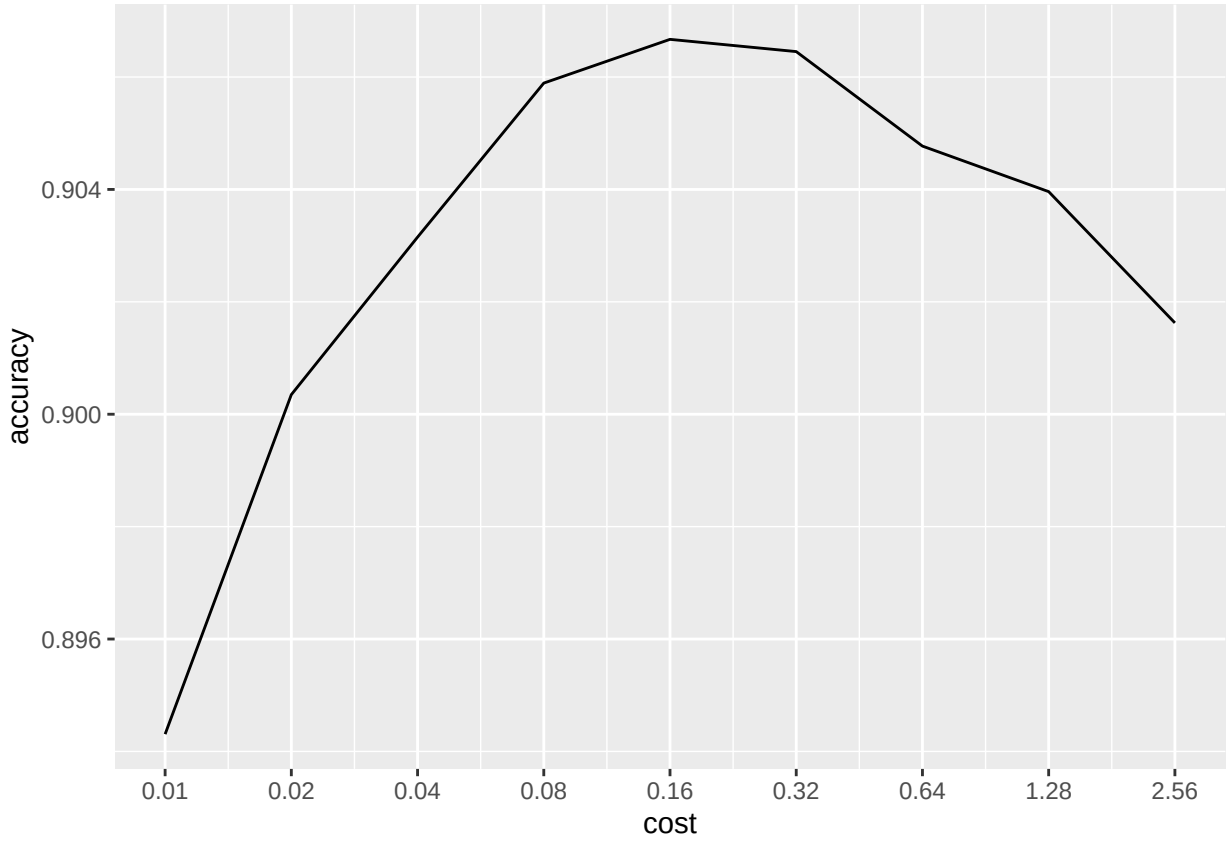
C) Using the best value of the cost parameter that you found in the last part, create a support vector machine using the entire training set to predicts whether a review is in the "Con" group or the "Pro" group.

```
split.train.csr <- as(split.train.dfm, "matrix.csr")
# LiblineaR does several types of linear models, and type=2 specifies SVM.
split.svm <- LiblineaR(data=split.train.csr, target=split.train.y, type=2)

split.test.csr <- as(split.test.dfm, "matrix.csr")
split.test.predictions <- predict(split.svm, newx=split.test.csr)[[1]]
confusionMatrix(split.test.y, split.test.predictions)
```

```
## Confusion Matrix and Statistics
##
```

```
##           Reference
## Prediction  Con  Pro
##        Con 6210  600
##        Pro  751 6202
##
##                Accuracy : 0.9018
##                  95% CI : (0.8967, 0.9068)
##     No Information Rate : 0.5058
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8037
##
##  Mcnemar's Test P-Value : 4.485e-05
##
##             Sensitivity : 0.8921
##             Specificity : 0.9118
##          Pos Pred Value : 0.9119
##          Neg Pred Value : 0.8920
##              Prevalence : 0.5058
##          Detection Rate : 0.4512
##    Detection Prevalence : 0.4948
##       Balanced Accuracy : 0.9020
##
##        'Positive' Class : Con
##
```

```
# Get costs on different orders of magnitude.
cost.vec <- 0.01*2^(0:8)
tuning.results <-
  map_dbl(cost.vec, ~LiblineaR(data=split.train.csr, target=split.train.y,
                               cost=.x, cross=10, type=2)
          )
data.frame(cost=cost.vec, accuracy=tuning.results) %>%
  ggplot() +
  geom_line(aes(x=cost, y=accuracy)) +
  scale_x_log10(breaks=cost.vec)
```



D) Then, apply your SVM model to your test set to create a confusion matrix and get an estimate for the accuracy of your SVM when being used to make predictions with new data. (You should be able to get at least 90% accuracy.)

```
cost.max <- cost.vec[which.max(tuning.results)]

split.best.svm <-
  LiblineaR(data=split.train.csr, target=split.train.y, type=2, cost=cost.max)

split.test.best.predictions <- predict(split.best.svm, newx=split.test.csr)[[1]]

confusionMatrix(split.test.y, split.test.best.predictions)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  Con  Pro
##        Con 6264  546
##        Pro  753 6200
##
##                Accuracy : 0.9056
##                  95% CI : (0.9006, 0.9105)
##     No Information Rate : 0.5098
##     P-Value [Acc > NIR] : < 2.2e-16
##
```

```
##                  Kappa : 0.8113
##
##  Mcnemar's Test P-Value : 1.093e-08
##
##             Sensitivity : 0.8927
##             Specificity : 0.9191
##          Pos Pred Value : 0.9198
##          Neg Pred Value : 0.8917
##              Prevalence : 0.5098
##          Detection Rate : 0.4551
##    Detection Prevalence : 0.4948
##       Balanced Accuracy : 0.9059
##
##        'Positive' Class : Con
##
```

E) Does it appear that the accuracy estimate from cross-validation matches up with the accuracy you had in categorizing your test set? Yes, the accuracy estimates were both over 90%.

Part 3

The paper in which this data set was first used referred to a data set that contained reviews of 15 consumer electronics products. No details were given as to what those products were, but we can guess some of them based on what was written. In particular, some reviews mentioned cameras. Lets see if we can do some topic modeling for the problems people encountered with cameras.

A) Create a new data set that consists of all the "Con" reviews that also mention the word "camera." (You should get 757 reviews in this smaller data set.)

```
df.camera <- df %>%
  rename_all(tolower) %>%
  mutate(review = tolower(review)) %>%
  filter(type== "Con" & grepl("camera", review))
```

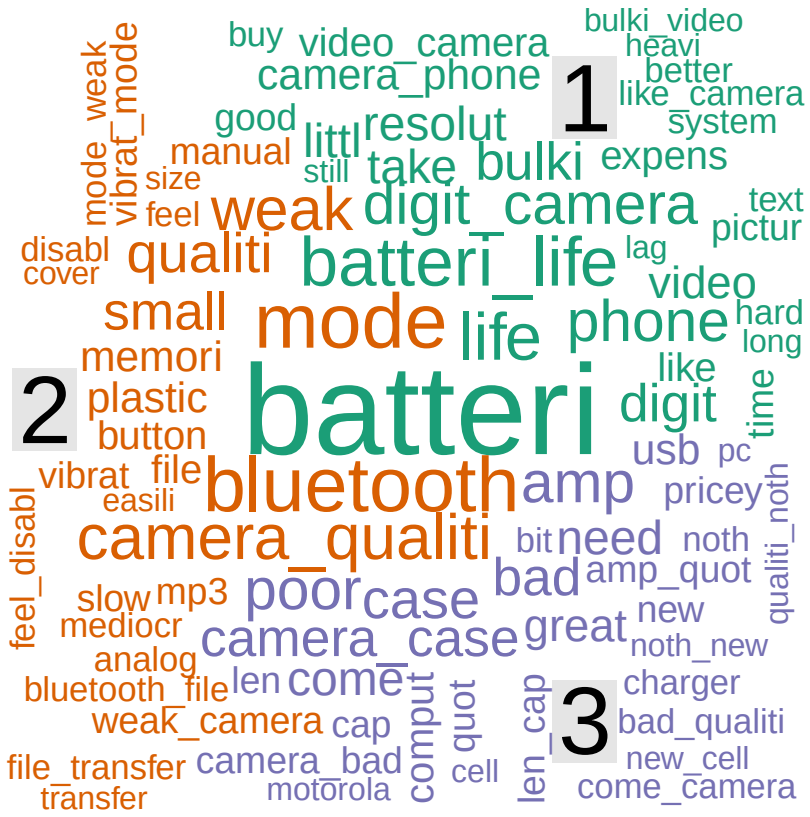
B) Do some tuning with coherence and perplexity to find an optimal number of topics for a topic model. Remember that in choosing a k, you're trying to balance getting a low perplexity, getting a high coherence, and not creating so many categories that you can't make sense of them. (For a very diverse data set, you might get a lot of categories, but when a question is more focused, you'd likely be trying to get fewer categories. Complains about cameras seems relatively focused.) You also want to avoid making so many categories that you're basically putting each review in its own category.

```
df.real.tokens <- tokens(df.camera$review, remove_punct=TRUE,
                         remove_symbols=TRUE, remove_numbers=TRUE) %>%
  tokens_remove(pattern=stopwords(language="en")) %>%
  tokens_wordstem() %>%
  tokens_ngrams(n=c(1,2))

df.real.dfm <- dfm(df.real.tokens) %>%
  dfm_trim(min_docfreq = 2/length(df.real.tokens),
           max_docfreq = 0.25, docfreq_type="prop")

df.real.dtm <- convert(df.real.dfm, to="topicmodels")

# Do the LDA analysis.
df.real.lda <- LDA(df.real.dtm, k=3)

# Make a topic prediction for each headline
df.real.topics <- topics(df.real.lda)

# We can print out terms related to each category in text form.
terms(df.real.lda,10)
```

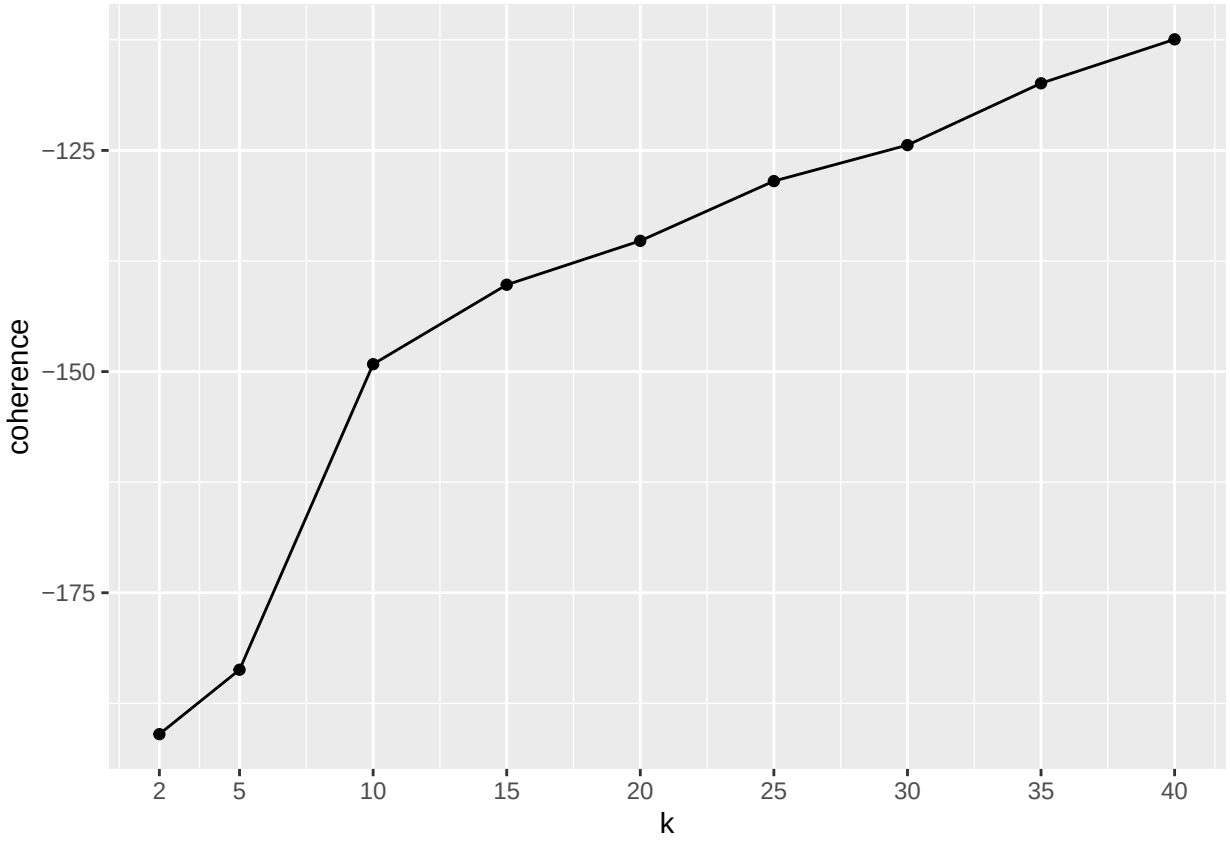
```
##       Topic 1        Topic 2     Topic 3
##  [1,] "batteri"      "qualiti"   "batteri"
##  [2,] "batteri_life" "life"      "poor"
##  [3,] "phone"        "digit"     "digit_camera"
##  [4,] "digit"        "button"    "slow"
##  [5,] "qualiti"      "bluetooth" "use"
##  [6,] "digit_camera" "mode"      "memori"
##  [7,] "bulki"        "small"     "bad"
##  [8,] "littl"        "slow"      "need"
```

```
##  [9,] "time"          "weak"       "amp"
## [10,] "expens"        "pictur"     "case"
```

```
# Convert back to DFM, just because it's easy to use the `groups` option
df.real.dfm <- as.dfm(df.real.dtm)


dfm_group(df.real.dfm, groups=df.real.topics) %>%
  convert(to="matrix") %>% t() %>%
  comparison.cloud(max.words=100)
```

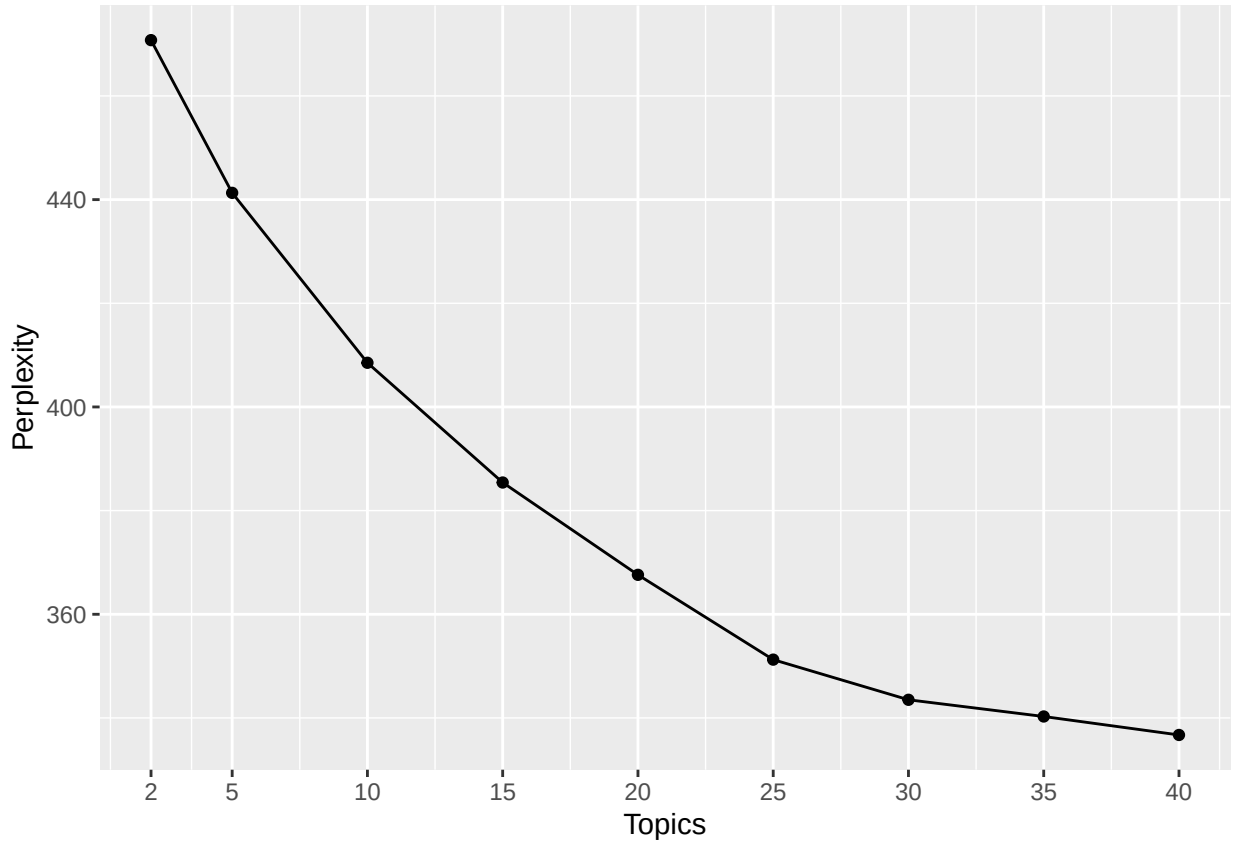



```
k.vec <- c(2, seq(5, 40, by=5))

coherence.vec <- map_dbl(k.vec,
  ~mean(topic_coherence(
    LDA(df.real.dtm, k=.x, control=list(nstart=5, best=TRUE)),
    df.real.dtm)
    )
)

data.frame(k=k.vec, coherence=coherence.vec) %>%
  ggplot(aes(x=k, y=coherence)) +
  geom_line() +
  geom_point() +
  scale_x_continuous(breaks=k.vec)
```




```
perp.vec <- rep(0, length(k.vec))

n.folds <- 5

folds <- sample(1:n.folds, nrow(df.real.dtm), replace=TRUE)

for(j in 1:n.folds){
  train.tokens <- tokens(df.camera$review[folds!=j], remove_punct=TRUE,
                         remove_symbols=TRUE, remove_numbers=TRUE) %>%
    tokens_remove(pattern=stopwords(language="en")) %>%
    tokens_wordstem() %>%
    tokens_ngrams(n=c(1,2))
  train.dfm <- dfm(train.tokens) %>%
    dfm_trim(min_docfreq = 2/length(train.tokens),
             max_docfreq = 0.25, docfreq_type="prop")
  test.tokens <- tokens(df.camera$review[folds==j], remove_punct=TRUE,
                        remove_symbols=TRUE, remove_numbers=TRUE) %>%
    tokens_remove(pattern=stopwords(language="en")) %>%
    tokens_wordstem() %>%
    tokens_ngrams(n=c(1,2))
  test.dfm <- dfm(test.tokens) %>%
    dfm_match(featnames(train.dfm))
  train.dtm <- convert(train.dfm, to="topicmodels")
  test.dtm <- convert(test.dfm, to="topicmodels")
  for(i in 1:length(k.vec)){
    # Do the LDA analysis.
```

```
    train.lda <- LDA(train.dtm, k=k.vec[i])
    perp.vec[i] <- perp.vec[i] + perplexity(train.lda, newdata=test.dtm)
  }
}

perp.vec <- perp.vec/n.folds

data.frame(Topics=k.vec, Perplexity=perp.vec) %>%
  ggplot(aes(x=Topics, y=Perplexity)) +
  geom_line() +
  geom_point() +
  scale_x_continuous(breaks=k.vec)
```




```
df.real.lda20 <- LDA(df.real.dtm, k=20, control=list(nstart=5, best=TRUE))
terms(df.real.lda20,10)
```

```
##       Topic 1       Topic 2      Topic 3     Topic 4          Topic 5
##  [1,] "bluetooth"   "manual"     "len"       "qualiti"        "littl"
##  [2,] "mode"        "focus"      "need"      "poor"           "low"
##  [3,] "weak"        "imag"       "cover"     "problem"        "still"
##  [4,] "vibrat"      "light"      "lack"      "poor_qualiti"   "good"
##  [5,] "plastic"     "size"       "cap"       "qualiti_camera" "speakerphon"
##  [6,] "feel"        "35mm"       "len_cap"   "call"           "hard"
##  [7,] "file"        "one"        "display"   "direct"         "softwar"
##  [8,] "transfer"    "open"       "bodi"      "slow"           "film"
```

```
##  [9,] "weak_camera" "con"         "len_cover" "price"          "mediocr"
## [10,] "vibrat_mode" "35mm_camera" "bigger"    "download"       "suck"
##       Topic 6       Topic 7          Topic 8       Topic 9
##  [1,] "pictur"      "poor"           "heavi"       "expens"
##  [2,] "take"        "qualiti"        "none"        "can"
##  [3,] "photo"       "camera_qualiti" "amp"         "better"
##  [4,] "usb"         "slow"           "must"        "resolut"
##  [5,] "set"         "poor_camera"    "best"        "get"
##  [6,] "connect"     "design"         "size"        "megapixel"
##  [7,] "power"       "camera_poor"    "size_camera" "megapixel_camera"
##  [8,] "camera_take" "camera_slow"    "avail"       "pricey"
##  [9,] "time"        "volum"          "charg"       "much"
## [10,] "use"         "media"          "slow"        "use"
##       Topic 10         Topic 11        Topic 12       Topic 13
##  [1,] "batteri"        "button"        "digit"        "featur"
##  [2,] "life"           "sound"         "digit_camera" "phone"
##  [3,] "batteri_life"   "side"          "system"       "color"
##  [4,] "use"            "camera_button" "text"         "camera_phone"
##  [5,] "short"          "annoy"         "compar"       "amp"
##  [6,] "short_batteri"  "speaker"       "lag"          "quot"
##  [7,] "camera_batteri" "keypad"        "text_system"  "amp_quot"
##  [8,] "slow"           "limit"         "camera_text"  "find"
##  [9,] "quick"          "etc"           "bulki_video"  "want"
## [10,] "turn"           "dial"          "first"        "camera_featur"
##       Topic 14        Topic 15      Topic 16         Topic 17
##  [1,] "zoom"          "batteri"     "bulki"          "bad"
##  [2,] "limit"         "like"        "video"          "phone"
##  [3,] "difficult"     "eat"         "video_camera"   "charger"
##  [4,] "bit"           "memori"      "bag"            "noth"
##  [5,] "work"          "eat_batteri" "camera_bag"     "bad_camera"
##  [6,] "menus"         "recharg"     "sometim"        "qualiti"
##  [7,] "optic"         "time"        "suffici"        "camera_bad"
##  [8,] "optic_zoom"    "long"        "camera_suffici" "new"
##  [9,] "compact"       "mp3"         "break"          "weak"
## [10,] "camera_comput" "like_camera" "memori"         "bad_qualiti"
##       Topic 18      Topic 19       Topic 20
##  [1,] "case"        "small"        "flash"
##  [2,] "larg"        "memori"       "great"
##  [3,] "includ"      "card"         "realli"
##  [4,] "camera_case" "come"         "flash_camera"
##  [5,] "case_includ" "stick"        "function"
##  [6,] "recept"      "screen"       "scratch"
##  [7,] "buy"         "memori_card"  "easili"
##  [8,] "may"         "memori_stick" "user"
##  [9,] "mani"        "come_camera"  "great_camera"
## [10,] "larg_camera" "small_memori" "just"
```

C) Write a short description of some of the important themes you see represented in these negative reviews involving cameras. Try to write it as a short memo to your boss, who's asked you to summarize an analysis of consumer feedback. (It doesn't have to be long, but it should be well written.) It appears that many of the complaints involve batteries/battery life. Other issues include mechanical features (lens cover, shutter, buttons, vibrations).