Text analysis

YZ Analytics

```
Wine <- read_csv("../data/winemag-data-130k-v2.csv",
                 col_types = cols(
                     X1 = col_double(),
                     country = col_character(),
                     description = col_character(),
                     designation = col_character(),
                     points = col_double(),
                     price = col_double(),
                     province = col_character(),
                     region_1 = col_character(),
                     region_2 = col_character(),
                     taster_name = col_character(),
                     taster_twitter_handle = col_character(),
                     title = col_character(),
                     variety = col_character(),
                     winery = col_character()),
                 progress = FALSE
                 ) %>%
    rename(id = X1)
```

```
## Warning: Missing column names filled in: 'X1' [1]
```

Term Frequency-Inverse Document Frequency

Code based from https://www.tidytextmining.com/tfidf.html.

Term Frequency-Inverse Document Frequency (TF-IDF) is a statistic that evaluates how important a word is in a document or corpus. It is calculated through dividing the term frequency of how often a word appears in a document by its inverse document frequency, which is the inverse of the proportion of how many documents in a corpus have a certain term. Therefore, high frequency terms but with little importance such as “the” or “and” will have low TF-IDF values, and so TF-IDF can be used as a weighting measure in ranking.

To use TF-IDF in our prediction model, we will use a sum of the TF-IDF per description as a variable.

```
Wine_tfidf <- Wine %>%
  unnest_tokens(word, description) %>%
  count(points, id, word, sort = TRUE) %>%
  bind_tf_idf(word, id, n) %>%
  group_by(id) %>%
  summarise(tf_idf = sum(tf_idf))

Wine2 <- left_join(Wine, Wine_tfidf, by = "id")
```

First let’s look at TF-IDF by points:

```
Wine_points_tfidf <- Wine %>%
  unnest_tokens(word, description) %>%
  count(points, word, sort = TRUE) %>%
  bind_tf_idf(word, points, n)
```

```r
Wine_points_tfidf %>%
  filter(points == 100) %>%
  arrange(desc(tf_idf)) %>%
  head()
```

```
## # A tibble: 6 x 6
##   points word           n       tf   idf  tf_idf
##    <dbl> <chr>      <int>    <dbl> <dbl>   <dbl>
## 1    100 masseto        2 0.00150   1.95 0.00292
## 2    100 frog           2 0.00150   1.66 0.00248
## 3    100 cerretalto     1 0.000749  3.04 0.00228
## 4    100 fragility      1 0.000749  3.04 0.00228
## 5    100 master's       1 0.000749  3.04 0.00228
## 6    100 proclaim       1 0.000749  3.04 0.00228
```

We see that the words with the highest TF-IDF values are the unique words in the 100-point wine descriptions that occur only 1-2 in the vocabulary of all the descriptions.

Let's look specifically at the words with the highest TF-IDF values for 80-point wines:

```r
Wine_points_tfidf %>%
  filter(points == 80) %>%
  arrange(desc(tf_idf)) %>%
  head()
```

```
## # A tibble: 6 x 6
##   points word           n      tf   idf   tf_idf
##    <dbl> <chr>      <int>   <dbl> <dbl>    <dbl>
## 1     80 strange       19 0.00180 0.560 0.00101
## 2     80 weedy         19 0.00180 0.560 0.00101
## 3     80 acceptable    16 0.00152 0.647 0.000982
## 4     80 weird         12 0.00114 0.847 0.000965
## 5     80 pickled       18 0.00171 0.560 0.000956
## 6     80 tastes        64 0.00607 0.154 0.000936
```

These words occur more frequently than the words in the 100-point descriptions. However, the frequencies are pretty low. It might be useful to separate points into different levels (perhaps 80-86 is low rating, 97-93 is medium, and 94-100 is high).

```r
Wine$rating <- cut(Wine$points,
                   breaks=c(-Inf, 86, 93, Inf),
                   labels=c("low","medium","high"))
```

```r
Wine_rating_tfidf <- Wine %>%
  unnest_tokens(word, description) %>%
  count(rating, word, sort = TRUE) %>%
  bind_tf_idf(word, rating, n)
```

```r
Wine_rating_tfidf %>%
  filter(rating == "high") %>%
  arrange(desc(tf_idf)) %>%
  head()
```

```
## # A tibble: 6 x 6
##   rating word      n       tf   idf    tf_idf
##   <fct>  <chr> <int>    <dbl> <dbl>     <dbl>
## 1 high   2025    226 0.000673 0.405 0.000273
```

```
## 2 high   2030    219 0.000653 0.405 0.000265
## 3 high   2023    152 0.000453 0.405 0.000184
## 4 high   2026     84 0.000250 0.405 0.000101
## 5 high   2035     81 0.000241 0.405 0.0000979
## 6 high   2027     77 0.000229 0.405 0.0000930
```

Following https://www.kaggle.com/nnnnick/predicting-wine-ratings-using-lightgbm-text2vec, we can look at words with the highest and lowest mean scores:

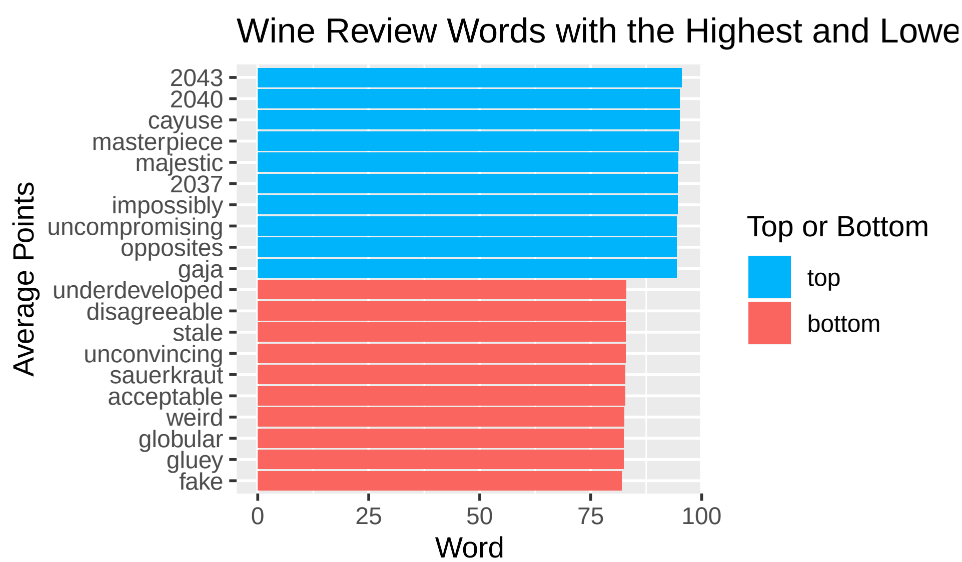
```r
wine_explore <- Wine %>%
    select(description, points) %>%
    mutate(description = gsub('[[:punct:] ]+',' ',tolower(description)))

words <- str_split(wine_explore$description, ' ')
all_words <- data.frame(points = rep(wine_explore$points, sapply(words, length)), words = unlist(words)
```

```r
words_grouped <- all_words %>%
    group_by(words) %>%
    summarize(
        points = mean(points),
        count = n()
    ) %>%
    filter(count > 10) %>%
    arrange(desc(points))

top <- words_grouped[1:10,] %>% cbind(top_bottom = 'top')
bottom <- words_grouped[(nrow(words_grouped) - 9):nrow(words_grouped),] %>% cbind(top_bottom = 'bottom'
top_bottom <- rbind(top, bottom)

ggplot(top_bottom, aes(x = reorder(words, points), y = points, fill = top_bottom)) +
    geom_bar(stat = 'identity') +
    coord_flip() +
    scale_fill_manual(values = c('#00b4fb', '#fa6560')) +
    ggtitle('Wine Review Words with the Highest and Lowest Mean Points', subtitle = NULL) +
    xlab('Average Points') +
    ylab('Word') +
    labs(fill = 'Top or Bottom')
```



Nobody wants to drink a wine that's described as "gluey" or "fake." How often do these words show up though, and how can we use them in a predictive model? To find out, we need to create a document-term matrix, which shows the the frequency of terms that occur in a collection of documents.

Creating a Document-Term Matrix

DTM created from code here: https://datawarrior.wordpress.com/2018/01/22/document-term-matrix-text-mining-in-r-and-pyt

```
#Create DTM
dtm <- CreateDtm(Wine$description,
                 doc_names = Wine$id,
                 ngram_window = c(1, 1),
                 lower = TRUE,
                 remove_punctuation = TRUE,
                 remove_numbers = TRUE,
                 stem_lemma_function = wordStem)
```

The document-term matrix that is created is huge - about 43 megabytes with close to 3 billion elements. We need to create some functions that will allow us to use the DTM:

```
#Create functions
get.token.occurrences<- function(dtm, token)
  dtm[, token] %>% as.data.frame() %>% rename(count=".") %>%
  mutate(token=row.names(.)) %>% arrange(-count)

get.total.freq<- function(dtm, token) dtm[, token] %>% sum

get.doc.freq<- function(dtm, token)
  dtm[, token] %>% as.data.frame() %>% rename(count=".") %>%
  filter(count>0) %>% pull(count) %>% length
```

Now we can see how many wines are actually described as "fake":

```
dtm %>% get.doc.freq(wordStem("fake"))
```

```
## [1] 13
```

Which 13 wines?

```
fakewines <- dtm %>% get.token.occurrences(wordStem("fake")) %>% head(13)
Wine$title[c(as.numeric(fakewines$token))]
```

```
##  [1] "Robert Stemmler 2005 Nugent Vineyard Pinot Noir (Russian River Valley)"
##  [2] "Funky Llama 2011 Merlot (Mendoza)"
##  [3] "Pierre Chardigny 2015 Vieilles Vignes  (Saint-Véran)"
##  [4] "Mellisoni 2016 Estate Pinot Grigio (Lake Chelan)"
##  [5] "Pradorey 2016 Tempranillo-Merlot Fermentado en Barrica Rosado (Ribera del Duero)"
##  [6] "Skylite 2005 Skylite Vineyard Merlot (Walla Walla Valley (WA))"
##  [7] "Black Stallion 2014 Cabernet Sauvignon (Napa Valley)"
##  [8] "Adega Cooperativa Ponte de Barca 2013 Ela Rosé (Vinho Verde)"
##  [9] "St. Julian 2013 Reserve Pinot Grigio (Lake Michigan Shore)"
## [10] "Cannonball 2010 Cabernet Sauvignon (California)"
## [11] "Finca Patagonia 2015 Expedicion Pinot Noir (Maule Valley)"
## [12] "Loken Cellars NV Reserve Lot 14 Rosé (California)"
## [13] "St. Andrews Estate 2000 Ceravolo Chardonnay (Adelaide Hills)"
```

Let's look at the description of the fourth wine, Love 2015 Cabernet Sauvignon (Vino de la Tierra de Castilla):

```r
Wine$description[27585] #27585 is the token number for the fourth wine
```

```
## [1] "Scattershot aromas of generic berry and cinnamon smell forced and fake. This has a tannic scrubl
```

Yikes. This seems like a bad wine.

Let's look at the most frequent terms:

```r
tf_mat <- TermDocFreq(dtm = dtm)
head(tf_mat[ order(tf_mat$term_freq, decreasing = TRUE) , ], 10)
```

```
##          term term_freq doc_freq       idf
## wine     wine     83107    66140 0.6755376
## flavor flavor     70968    65697 0.6822581
## fruit   fruit     63935    55692 0.8474748
## aroma   aroma     41052    40492 1.1662069
## finish finish     40466    40083 1.1763590
## acid     acid     39812    38586 1.2144218
## palat   palat     38636    37796 1.2351081
## drink   drink     33970    33244 1.3634370
## cherri cherri     33590    31328 1.4227991
## tannin tannin     32981    31960 1.4028262
```

Unsurprisingly, the most frequently used words in the descriptions are "wine," "flavor," and "fruit."

Now let's see how we can use the DTM in a data frame for prediction.

```r
# remove any tokens that were in 2000 or fewer documents
dtm_small <- dtm[ , colSums(dtm > 0) > 2000 ]
ncol(dtm_small)
```

```
## [1] 279
```

We're left with 279 words that are used in more than 2000 documents in the form of indicator variables per wine.

Prediction

Let's look at predicting wines.

Because we have a lot of missing data and a mixture of numerical and categorical data, methods like random forest are difficult to implement. Let's try gradient boosting, which in R can include categorical variables of up to 1024 categories (unlike randomForest, which only allows up to 53 categories per categorical variable).

First, we need to clean our data:

```r
dtm_df <- as.data.frame(as.matrix(dtm_small))
dtm_df$id <- c(0:129970)
Wine_dtm <- left_join(Wine2, dtm_df, by = "id")
Wine_dtm <- Wine_dtm %>%
  select(-id, -description, -designation, -region_1, -region_2, -taster_name,
         -taster_twitter_handle, -title, -winery) %>%
  mutate(country = as.factor(country),
         province = as.factor(province),
         variety = as.factor(variety))
```

```r
#Test/Train split
set.seed(1)
smp_size <- floor(0.8 * nrow(Wine_dtm))
```

```r
train_ind <- sample(seq_len(nrow(Wine_dtm)), size = smp_size)
train <- Wine_dtm[train_ind, ]
test <- Wine_dtm[-train_ind, ]
```

We'll use a gradient boosting algorithm to create our prediction model.

```r
#boosting
set.seed(1)
boost_wine <- gbm(points ~ .,
                  data = train,
                  distribution = "gaussian",
                  n.trees = 1000,
                  interaction.depth = 4)
```

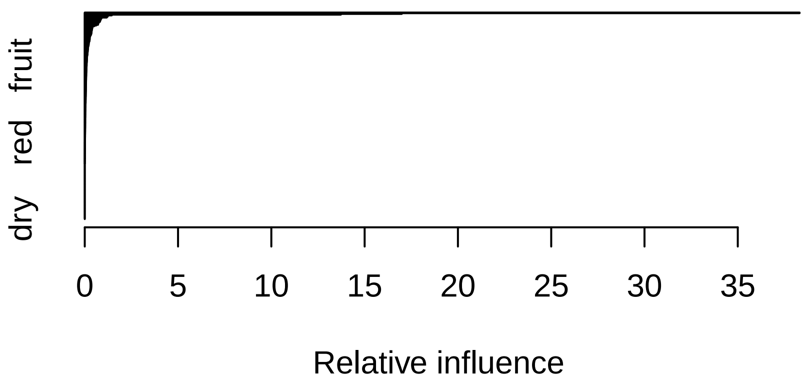
We can check to see how well our model does with prediction on the test set:

```r
boost_estimate <- predict(boost_wine,
                          newdata = test,
                          n.trees = 1000)
mse_boost <- mean((boost_estimate - test$points)^2); mse_boost
```

```
## [1] 2.947997
```

The mean squared error is 2.9479967. Let's look at the most important features in the model.

```r
top_n(summary(boost_wine), 20, rel.inf)
```




```
##         var    rel.inf
## 1   price.x 38.3115789
## 2   variety 16.9914220
## 3  province 13.7256859
## 4      rich  1.4476887
## 5   complex  1.2427373
## 6    tf_idf  1.2197357
## 7     simpl  1.1765287
## 8    delici  0.8982307
## 9      long  0.8788905
## 10    black  0.8493266
## 11 concentr  0.8398059
## 12   balanc  0.8201405
## 13    power  0.8087432
## 14 structur  0.7478065
## 15    miner  0.7353281
## 16     eleg  0.7169249
```

```
## 17    great  0.7077206
## 18 vineyard  0.6064132
## 19    spice  0.5047514
## 20     fine  0.4138690
```

Unsurprisingly, the variable with the most relative influence is price, followed by variety, then province. The words "rich" and "complex" have slightly higher influence than the tf_idf variable, then followed by the stemmed words "simpl," "delici," "long," and "black."