Text Analysis and Prediction

YZ Analytics

A key part of this project is learning how to extract features from text. In the case of our data with wine reviews, the largest body of text we have is from the description variable. First we'll load in our data.

```
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)
```

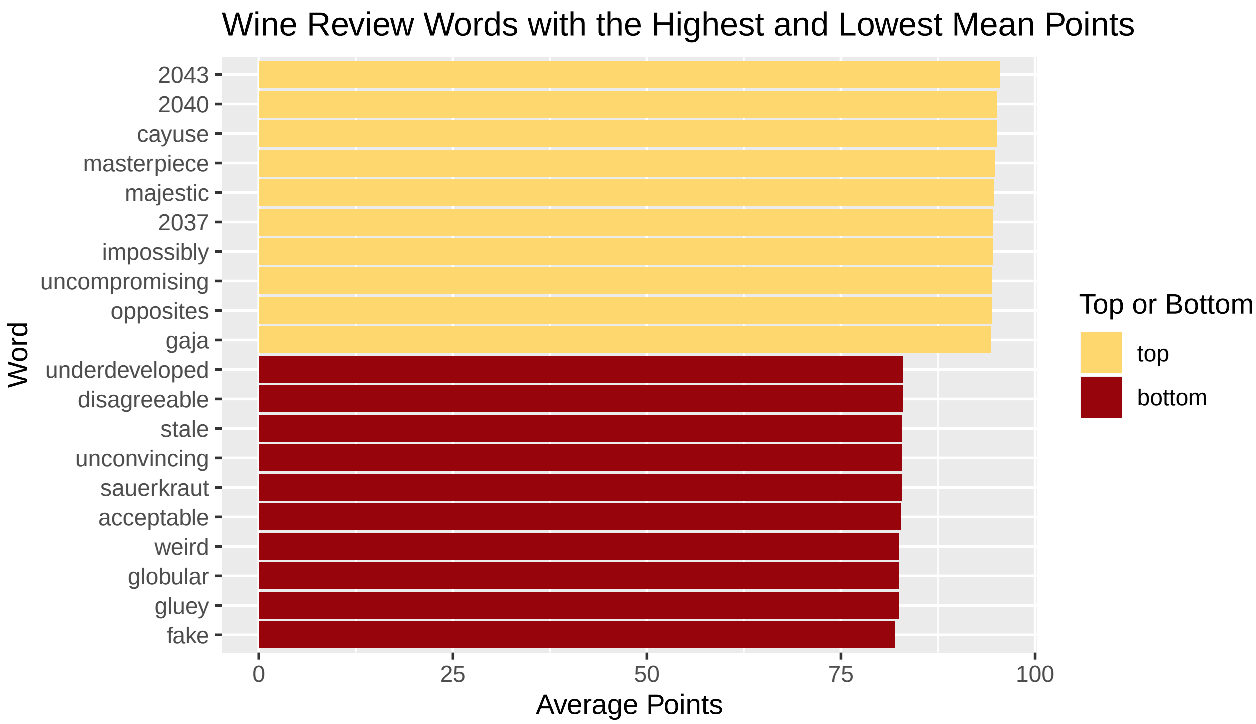
```
## Warning: Missing column names filled in: 'X1' [1]
```

First, we can look at some exploratory plots. For example, following code from Kaggle user nnnnick (2018), we can look at the words in the wine description with the highest and lowest mean scores:

```
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))
```

```
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('#fed86f', '#97040b')) +
    ggtitle("Wine Review Words with the Highest and Lowest Mean Points") +
    xlab("Word") +
    ylab("Average Points") +
    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)

A DTM is a matrix in which the rows correspond to documents in a corpus (in our case, each wine description constitutes a document) and each column corresponds to terms, or words. This code to create a DTM for our wine dataset is based on Ho (2018).

```
#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 resulting DTM 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:

```r
#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":

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

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

Which 13 wines?

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

[1] "Scattershot aromas of generic berry and cinnamon smell forced and fake. This has a tannic scrubbing mouthfeel and artificial flavors of chocolate and cheap oak. A green note and burn on the finish do nothing to help this along."

Yikes. This seems like a bad wine.

From our document-term matrix, we could create a list of the top words by frequency and use those in our predictive model. However, if we were basing our top words by term frequency, we would be including words that are used so often that they probably don't have much meaning. Therefore, a better way to rank our words would be term frequency-inverse document frequency, which will be explained in the next section.

Term Frequency-Inverse Document Frequency

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.

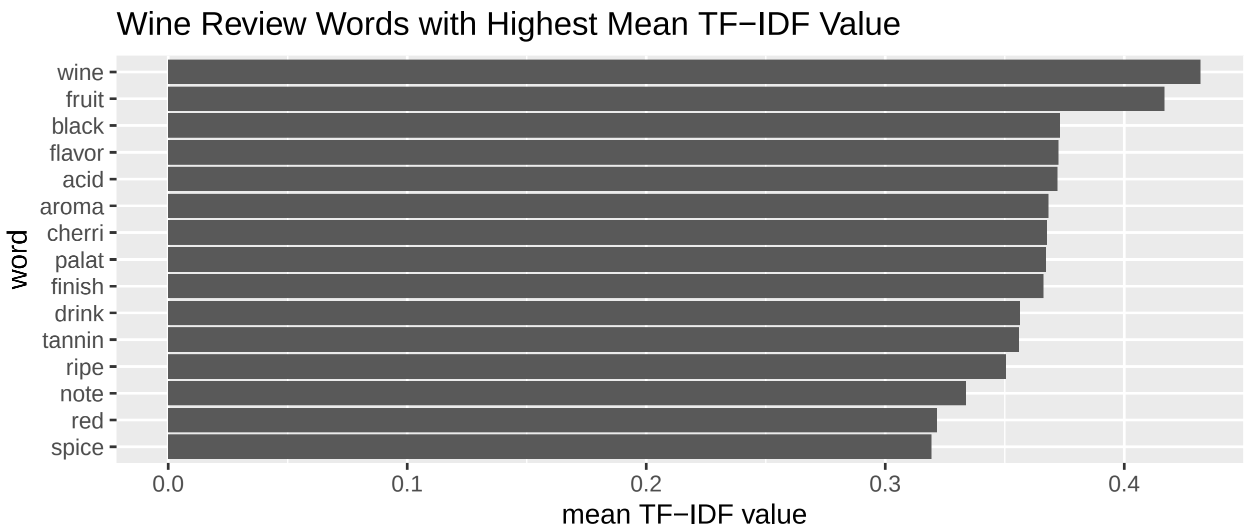
TF-IDF will be useful in our prediction model becuase we can include the words with the higest mean TF-IDF values in our model. Let’s see what these words with the highest mean TF-IDF values are. The following code has been modified from the one of the cran.R vignettes of the `textmineR` package (Jones, 2019).

```
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
```

```
tfidf_mat <- t(dtm[ , tf_mat$term ]) * tf_mat$idf #calculating TF-IDF
tfidf <- t(tfidf_mat)
tfidf_means <- colMeans(tfidf) #calculating mean values
tfidf_means <- as.data.frame(tfidf_means)
tfidf_means$word <- rownames(tfidf_means)
top200 <- tfidf_means %>% arrange(desc(tfidf_means)) %>% top_n(200, tfidf_means) #top 200
```

```
# Plot the words with highest mean TF-IDF values
tfidf_means %>%
  arrange(desc(tfidf_means)) %>%
  top_n(15, tfidf_means) %>%
  ggplot(aes(reorder(word, tfidf_means), tfidf_means)) +
  geom_bar(stat = "identity") +
  ggtitle("Wine Review Words with Highest Mean TF-IDF Value") +
  xlab("word") +
  ylab("mean TF-IDF value") +
  coord_flip()
```



"Wine" and "fruit" have especially high mean TF-IDF values, followed by "black," "flavor," "acid," and "aroma."

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

```
# Match dtm column names to the words with top 200 mean tf-idf values
dtm_small <- dtm[, colnames(dtm) %in% top200$word]
ncol(dtm_small)
```

```
## [1] 200
```

We're left with a document-term matrix with the 200 words with the highest mean tf-idf value.

Prediction

Let's look at predicting wines. We are looking to build a model that can be implemented for an average user of our web app to input values and text and receive an output of points.

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. We will remove variables that either 1) are factor variables with more than 1024 categories, or 2) are variables that are not necessarily relevant to an average person looking to explore wine. An example of variables in the latter category are `taster_name` and `taster_twitter_handle`.

```
dtm_df <- as.data.frame(as.matrix(dtm_small))
dtm_df$id <- c(0:129970)
Wine_dtm <- left_join(Wine, 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))
```

Now we will split our data into training and test sets.

```
# Test/Train split
set.seed(1)
```

```r
smp_size <- floor(0.8 * nrow(Wine_dtm))
train_ind <- sample(seq_len(nrow(Wine_dtm)), size = smp_size)
train <- Wine_dtm[train_ind, ]
test <- Wine_dtm[-train_ind, ]
```

Now we can apply a gradient boosting algorithm to create our prediction model. First, let's try a boosting model with 5 trees. This is not a lot of trees, so we'd expect that the model wouldn't do so well with prediction on our test set.

```r
set.seed(1)
boost_wine10 <- gbm(points ~ .,
                   data = train,
                   distribution = "gaussian",
                   n.trees = 5,
                   interaction.depth = 4)

boost_estimate10 <- predict(boost_wine10, # predict on test set
                         newdata = test,
                         n.trees = 5,
                         na.action = NULL)
mse_boost10 <- mean((boost_estimate10 - test$points)^2)
sqrt_mse10 <- sqrt(mse_boost10) # calculate MSE
```

With this model, the prediction is off by an average of 2.6593525 points. That's not great. Let's compare this square root of the MSE to that of a model where we use 500 trees.

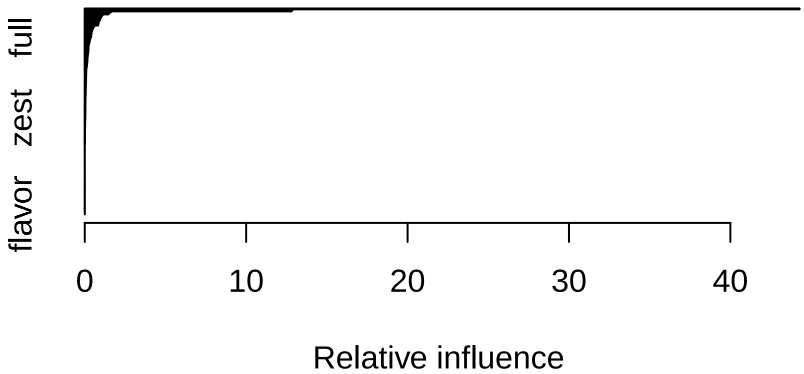
```r
set.seed(1)
boost_wine <- gbm(points ~ .,
                   data = train,
                   distribution = "gaussian",
                   n.trees = 500,
                   interaction.depth = 4)

# Save model to .rds file
saveRDS(boost_wine, "boost_wine500.rds")
```

Running the above chunk takes too long, so we've loaded the model in the chunk below. We can check to see how well this model with 500 trees does with prediction on the test set:

```r
boost_wine <- readRDS("boost_wine500.rds") # load model
boost_estimate <- predict(boost_wine,
                         newdata = test,
                         n.trees = 500,
                         na.action = NULL)
mse_boost <- mean((boost_estimate - test$points)^2)
sqrt_mse <- sqrt(mse_boost)
```

The square root of the mean squared error is 1.834457 - much smaller than that of the model with only 5 trees. Let's look at the most important features in this more accurate model.

```r
top_n(summary(boost_wine), 10, rel.inf)
```




```
##         var    rel.inf
## 1      price 44.2872441
## 2    variety 12.8744278
## 3   province 12.8074735
## 4       rich  1.6195324
## 5    complex  1.5243999
## 6      simpl  1.4679213
## 7       long  1.1043904
## 8     delici  1.0599025
## 9      black  0.9946763
## 10 concentr  0.9875296
```

Unsurprisingly, the variable with the most relative influence is price, followed by variety, then province. Stemmed words that are the most important are "rich", "complex," "simpl."

Let's see how this model predicts the points of a wine that is not in the data set.

For example, we can predict the points of a Portuguese Red Touriga Nacional wine that is $35 from Dão with the description: "This is a solidly structured wine that has big tannins in place. That will change as the wine ages further, bringing the rich black fruits forward and reveling in the perfumed acidity of the wine. Drink from 2021."

Below, we have created a function called `estimatepoints` that uses inputs of country, price, description, province, and variety and returns a points estimate based on our model.

```
estimatepoints <- function(country, price, description, province, variety) {
  newwine <- data.frame(id = 1, country, price, description, province, variety) %>%
    mutate(description = as.character(description))

  # Create DTM for new wine
  dtm_newwine <- CreateDtm(newwine$description,
              doc_names = newwine$id,
              ngram_window = c(1, 1),
              lower = TRUE,
              remove_punctuation = TRUE,
              remove_numbers = TRUE,
              stem_lemma_function = wordStem)

  dtm_newwine2 <- dtm_newwine[, colnames(dtm_newwine) %in% top200$word] # words in top 200
  dtm_newwine_df <- as.data.frame(as.matrix(t(dtm_newwine2)))
  otherwords <- top200 %>%
    filter(!word %in% dtm_newwine_df) # find the top 200 words not in new wine but in dtm
  dtm_newwine_df[otherwords$word] <- 0 # fill the words not in new wine to df with 0
```

```r
  newwine <- cbind(newwine, dtm_newwine_df) # fill in rest of data

  # Estimate points of new wine
  test_estimate <- predict(boost_wine,
                           newdata = newwine,
                           n.trees = 100,
                           na.action = NULL)

  return(test_estimate)
}

# Save function & top200 data frame for use in Shiny app
save(top200, estimatepoints, file="estimatepoints_function.Rda")
```

So now to predict the number of points our Portuguese wine would receive, we can just plug in the input values.

```r
estimatepoints(country = "Portugal",
               price = 35,
               description = "This is a solidly structured wine that has big tannins in place.
               That will change as the wine ages further, bringing the rich black fruits
               forward and reveling in the perfumed acidity of the wine. Drink from 2021.",
               province = "Dão",
               variety = "Touriga Nacional, Portuguese Red")
```

```
## [1] 88.83818
```

Our model predicts this wine would receive about 89 points. Not bad.

Model with just description

The model we are using to predict points uses variables like price, province, and variety, which all have much higher relative influence compared to just the words in the description. What if we removed these variables with higher relative influence and looked just at how well words in the description can predict points?

We will select just the points variable and the word variables and set up a training and test set to create the same gradient boosting model with just the word variables.

```r
Wine_justwords <- Wine_dtm %>%
  select(-country, -price, -province, -variety) # leaves just the DTM of words

# Train/Test split
set.seed(1)
smp_size2 <- floor(0.8 * nrow(Wine_justwords))
train_ind2 <- sample(seq_len(nrow(Wine_justwords)), size = smp_size2)
train2 <- Wine_justwords[train_ind2, ]
test2 <- Wine_justwords[-train_ind2, ]
```

Now we can fit the model. Again, this algorithm takes a while to run, so the code to create the model is shown in the first code chunk below, but we will just load the model object into the next code chunk for analysis.

```r
set.seed(1)
boost_wine_words <- gbm(points ~ .,
                   data = train2,
                   distribution = "gaussian",
                   n.trees = 500,
                   interaction.depth = 4)
```

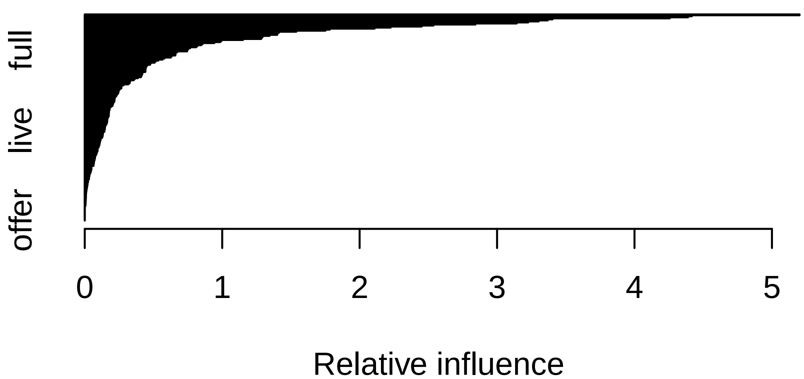
```r
# Save model to .rds file
saveRDS(boost_wine_words, "boost_wine_words.rds")
```

```r
boost_wine_words <- readRDS("boost_wine_words.rds") # load model
boost_estimate_words <- predict(boost_wine_words,
                                newdata = test2,
                                n.trees = 500,
                                na.action = NULL)
mse_boost_words <- mean((boost_estimate_words - test2$points)^2)
sqrt_mse_words <- sqrt(mse_boost_words); sqrt_mse_words
```

```
## [1] 2.177103
```

The model with just words performs a little worse than our model with price, variety, and province included with a mean prediction error of 2.1771026 points. We can look at which words have the most relative influence in this model:

```r
top_n(summary(boost_wine_words), 10, rel.inf)
```




```
##         var  rel.inf
## 1      rich 5.201027
## 2  vineyard 4.417346
## 3   complex 4.389741
## 4     simpl 4.256652
## 5  concentr 3.403552
## 6     black 3.368652
## 7      year 3.303258
## 8      long 3.226283
## 9     power 3.142659
## 10     eleg 2.842253
```

The word "rich" has the highest relative influence, followed by words like "vineyard," "complex," the stemmed "simpl," and the stemmed "concentr."

However, because the model with price, province, and variety included has a lower MSE, and because these variables would not be difficult to find for an average wine drinker, we will use the model with 500 trees with price, province, and variety included to build our prediction engine.