Ulta Text Modeling

Andrew Mashhadi

01 June, 2023

Load Libraries

```
library(dplyr)
library(tidytext)
library(ROCR)
library(pROC)
library(caret)
library(broom)
library(ggplot2)
library(ggpubr)
library(randomForest)
library(purrr)
library(tm)
library(tidyr)
library(stringr)
library(SnowballC)
```

Modeling

Reading in the the full dataframe of webscraped data `full_df`.

```
## set data path
PATH_ULTA_TEXT_DIR <- Sys.getenv("ULTA_TEXT_DATA")
PATH_ULTA_PRODUCT_NAMES <-paste0(PATH_ULTA_TEXT_DIR, '/products/product_dict.csv')

## read in product names
product_names <- read.csv(PATH_ULTA_PRODUCT_NAMES)[, 2:3]

## get full paths to all data files
csv_files <- list.files(PATH_ULTA_TEXT_DIR, pattern = "csv$", full.names = T)
brand_names <- list.files(PATH_ULTA_TEXT_DIR, pattern = "csv$", full.names = F)

# initiallize df
full_df <- data.frame(matrix(ncol = 7, nrow = 0))
colnames(full_df) <- c('id', 'class', 'text', 'reviews',
                       'rating', 'price', 'brand')

## read in product names
for (i in 1:length(csv_files)) {
```

```r
  tmp <- read.csv(csv_files[i])[, 2:7]

  tmp <- tmp %>%
    mutate(brand=str_replace(brand_names[i], ".csv", ""))

  full_df <- rbind(full_df, tmp)

}

# remove duplicates (sometimes same product page woudl appear a few times)
full_df <- full_df %>%
  distinct()

full_df <- full_df %>%
  group_by(id) %>%
  mutate(rev_num=1:n()) %>%
  ungroup()


# join with product names found seperately
full_df <- full_df %>%
   inner_join(product_names %>%
                distinct())
```

```
## Joining, by = "id"
```

```r
ona_df_og <- full_df %>%
   na.omit() %>%
   filter(reviews != "")
```

Part (1)

Model to predict product rating from review text.

We begin with a setup for train/test split.

```r
# set seed
set.seed(7)

revs_df <- ona_df_og %>%
  select(c(id, rating, reviews)) %>%
  group_by(id) %>%
  summarise(review=paste0(reviews, collapse = " "),
            rating=mean(rating)) %>%
  ungroup()

# train test split
train.ind <- sample(1:nrow(revs_df),
                    replace = FALSE,
                    size=floor(0.80 * nrow(revs_df)))

train_revs <- revs_df[train.ind, ]
```

```r
# tidy revs data by brand
tidy_train_revs <- train_revs %>%
  unnest_tokens(word, review) %>%
  filter(!grepl('[0-9]', word)) %>%
  mutate(word = str_remove_all(word, "[:punct:]")) %>%
  anti_join(stop_words) %>%
  mutate(word = wordStem(word))

# cast tidy data into dtm
dtm_train <- tidy_train_revs %>%
  count(id, word, sort=TRUE) %>%
  cast_dtm(id, word, n)

# remove extremely sparse terms
dtm_train <- removeSparseTerms(dtm_train, 0.995)

# inspect corpus
dtm_train %>% inspect()
```

```
## <<DocumentTermMatrix (documents: 10437, terms: 1408)>>
## Non-/sparse entries: 477065/14218231
## Sparsity           : 97%
## Maximal term length: 13
## Weighting          : term frequency (tf)
## Sample             :
##          Terms
## Docs      dai dry feel hair im iv love product skin smell
##   2304475   1   1    4   36  2  3    0       1    0     1
##   2515554   1   9    3   22  2  3    0       6    0     0
##   2586825   1   4    3   30  0  4    0       8    0     3
##   2591424   1   2    4   26  4  2    5       9    0     3
##   2594719   3   0    1   12  1  5    1       7    5     0
##   2595586   2   3    2    0  0  3   11       5    6     5
##   2596248   4   1    6    0  2  3   11       8   18     7
##   2606383   2   3    4    0  6  1    2      10    6     3
##   2607973   2   4    4    0  1  2    4       8   19     0
##   2609407   1   5    0   34  3  2    2      18    0     4
```

```r
# make syntactically valid names
colnames(dtm_train) = make.names(colnames(dtm_train))

# convert dtm to df
train.data <- dtm_train %>%
  as.matrix() %>%
  as.data.frame()

# make id a column
train.data$id <- as.numeric(rownames(train.data))

# join the Y values, rating
train <- train.data %>%
  left_join(revs_df %>%
              select(c(id, rating)),
          by="id")
```

Validate using the training set with 5-fold CV. Then train on full training data.

```
## Validation for tuning mtry

rfGrid <- expand.grid(.mtry = c(floor(sqrt(ncol(train))),
                                floor(ncol(train)/6),
                                floor(ncol(train)/3),
                                floor(ncol(train)/2)))

tr.control <- trainControl(method = "cv",
                           number=4,
                           search = 'grid',
                           verboseIter = TRUE)

cv_rf_model <- train(rating ~ . -id,
                     data = train,
                     method = "rf",
                     metric = "RMSE",
                     tuneGrid = rfGrid,
                     ntree=100,
                     trControl = tr.control)

# print validation metrics
print(cv_rf_model)

# print validation metrics
ggplot(cv_rf_model) +
  geom_point(colour = "red", size = 4) +
  geom_line(colour = "red", size = 2) +
  ggtitle("4-Fold CV Tuning")

# save tuning image
ggsave(paste0("images/rev_cv_rf_tuning.png"))


## Train with full dataset now

start.time <- Sys.time() # start timer

model_rf <- randomForest(rating ~ .-id,
                         data = train,
                         mtry = floor(ncol(train)/6), # from tuning
                         ntree = 500)

elapsed.time <- round((Sys.time() - start.time), 3) # stop timer

# report train time
cat("Finished training in ", elapsed.time, " minutes\n")

# save time consuming data
```

```r
save.image("Rdata/models_checkpoint1.RData")
```

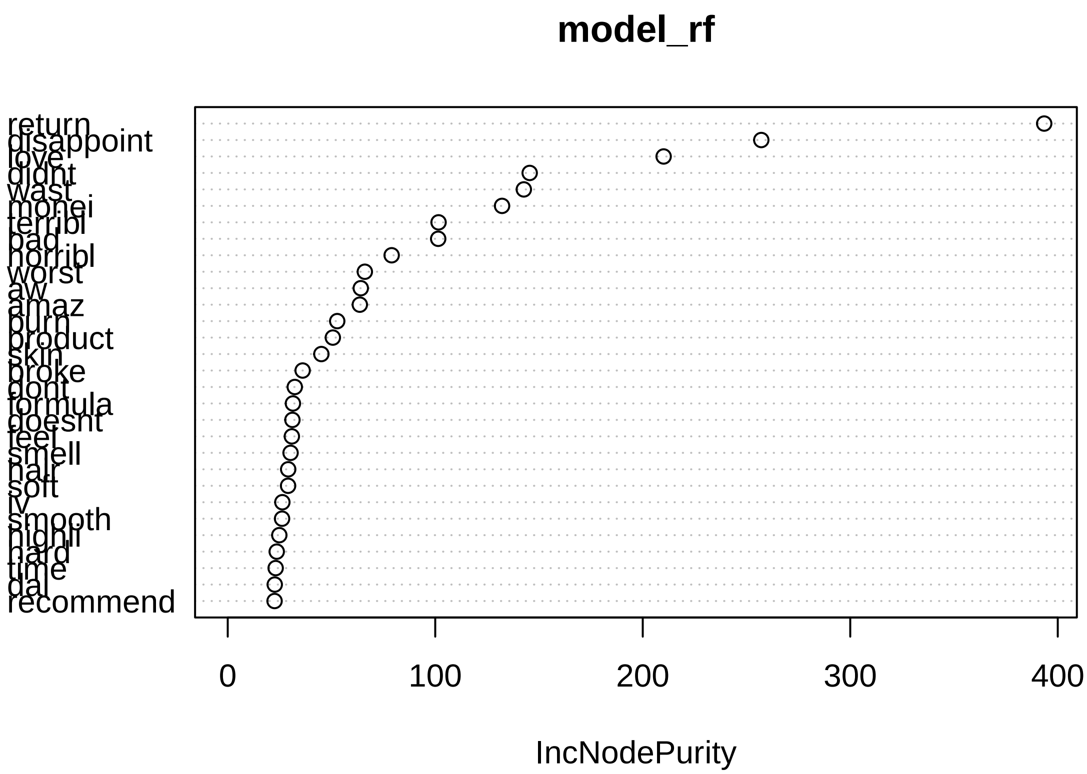
```r
load("Rdata/models_checkpoint1.RData")

# report train RMSE
preds <- predict(model_rf, newdata = train)
cat("Training RMSE: ", sqrt(mean((train$rating-preds)^2)))
```

```
## Training RMSE:  0.2675206
```

Feature importances.

```r
# setup imp object
imp <- varImpPlot(model_rf) %>%
  as.data.frame() %>%
  arrange(desc(IncNodePurity)) %>%
  head(n=15)
```

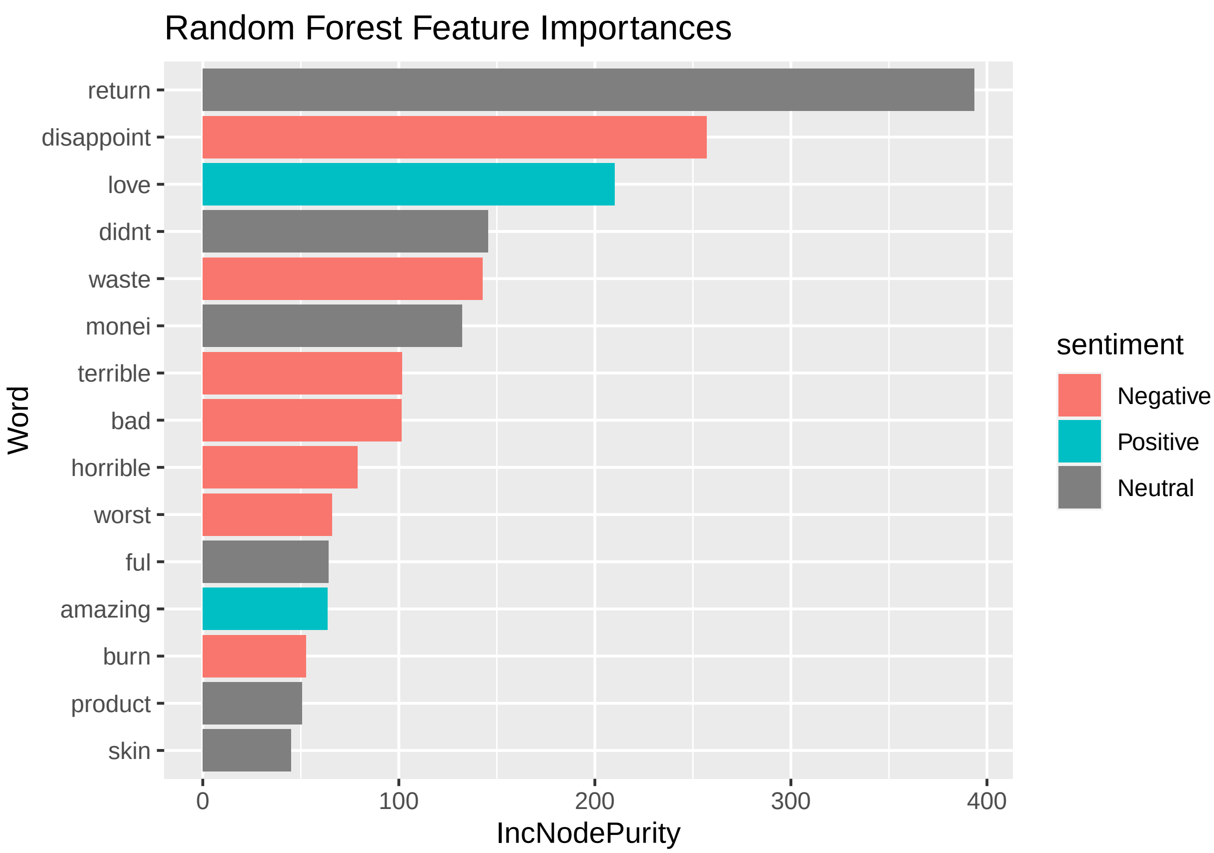



```r
imp$word <- rownames(imp) # row names to column

imp <- imp %>%
  mutate(across('word', str_replace, 'terribl', 'terrible')) %>%
  mutate(across('word', str_replace, 'wast', 'waste')) %>%
  mutate(across('word', str_replace, 'aw', 'ful')) %>%
  mutate(across('word', str_replace, 'amaz', 'amazing')) %>%
```

```
  mutate(across('word', str_replace, 'horribl', 'horrible')) %>%
  left_join(get_sentiments("bing"))

imp %>%
ggplot(aes(x=reorder(word, IncNodePurity), weight=IncNodePurity, fill=sentiment)) +
  geom_bar() +
  scale_fill_discrete(name="Variable Group") +
  ylab("IncNodePurity") +
  xlab("Word") +
  ggtitle("Random Forest Feature Importances") +
  coord_flip() +
  scale_fill_discrete(labels = c("Negative", "Positive", "Neutral"))
```




```
ggsave(paste0("images/reviews_feat_imp.png"))
```

Testing Performance.

```
test_revs <- revs_df[-train.ind, ]

# tidy revs data by brand
tidy_test_revs <- test_revs %>%
  unnest_tokens(word, review) %>%
  filter(!grepl('[0-9]', word)) %>%
  mutate(word = str_remove_all(word, "[:punct:]")) %>%
```

```r
  anti_join(stop_words) %>%
  mutate(word = wordStem(word))


# cast tidy data into dtm
dtm_test <- tidy_test_revs %>%
  count(id, word, sort=TRUE) %>%
  cast_dtm(id, word, n)

# make syntactically valid names
colnames(dtm_test) = make.names(colnames(dtm_test))

# convert dtm to df
test.data <- dtm_test %>%
  as.matrix() %>%
  as.data.frame()

# make id a column
test.data$id <- as.numeric(rownames(test.data))

# join the Y values, rating
test <- test.data %>%
  left_join(revs_df %>%
               select(c(id, rating)),
            by="id")

preds <- predict(model_rf, newdata = test)

cat("Testing RMSE: ", sqrt(mean((test$rating-preds)^2)), "\n")
```

```
## Testing RMSE:  0.6176011
```

```r
rss <- sum((preds - test$rating) ^ 2)  ## residual sum of squares
tss <- sum((test$rating - mean(test$rating)) ^ 2)  ## total sum of squares
rsq <- 1 - rss/tss

cat("Testing R^2: ", 1 - rss/tss)
```

```
## Testing R^2:  0.4269521
```

Part (2)

Model (find relationship) between sentiment and price.

First, we get the sentiments again.

```r
options(scipen=2)

tidy_revs <- ona_df_og %>%
   unnest_tokens(word, reviews) %>%
   filter(!grepl('[0-9]', word)) %>%
   mutate(word = str_remove_all(word, "[:punct:]")) %>%
```

```
  anti_join(stop_words)

# get sentiments for each review
rev_sents <- tidy_revs %>%
  inner_join(get_sentiments("afinn")) %>%
  group_by(id, rev_num) %>%
  summarise(sentiment = sum(value))

# treat each review equally (with weight) then get prop. positive
rev_sents <- rev_sents %>%
  filter(sentiment != 0) %>%
  mutate(sentiment=as.integer(sentiment > 0)) %>%
  group_by(id) %>%
  summarise(pos_sentiment=mean(sentiment))

# merge sentiments with reviews
ona_df <- ona_df_og %>%
  inner_join(rev_sents, by="id") %>%
  group_by(id) %>%
  slice(1) %>%
  ungroup() %>%
  select(-c(reviews, rev_num))
```

Now we model using the entire dataset.

```
df <- ona_df %>%
  filter(pos_sentiment != 0.5) %>%
  mutate(sentiment = case_when(
                          pos_sentiment < 0.5 ~ "negative",
                          pos_sentiment > 0.5 ~ "positive"))
df$sentiment <- factor(df$sentiment)


summary(glm(sentiment~price, data=df, family = "binomial"))
```

```
##
## Call:
## glm(formula = sentiment ~ price, family = "binomial", data = df)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.2251   0.3639   0.3776   0.3886   0.4008
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) 2.474844   0.053189  46.529  < 2e-16 ***
## price       0.005441   0.001489   3.654 0.000258 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
```

```
##
##     Null deviance: 5975.9  on 12095  degrees of freedom
## Residual deviance: 5960.0  on 12094  degrees of freedom
## AIC: 5964
##
## Number of Fisher Scoring iterations: 5
```

Significant. Over all product classes, it appears that an increase in price corresponds to a more positive sentiment.

Now we perform the same over each individual product type.

```r
# set up tibbles with nesting
nested_data <- df %>%
  select(c(class, sentiment, price)) %>%
  nest(data = c(-class))


# create log-reg model for each product category
nested_models <- nested_data %>%
  mutate(models = map(data, ~ glm(sentiment ~ price, .,
                                  family = "binomial")))


# store model summarys in unnested format
slopes <- nested_models %>%
  mutate(models = map(models, tidy)) %>%
  unnest(cols = c(models)) %>%
  filter(term=="price")

slopes %>%
  filter(p.value < 0.05) %>%
  arrange(p.value)
```

```
## # A tibble: 4 x 7
##   class           data               term  estimate std.error statistic p.value
##   <chr>           <list>             <chr>    <dbl>     <dbl>     <dbl>   <dbl>
## 1 Anti-Aging      <tibble [37 x 2]>  price  -0.0212   0.00919     -2.31  0.0210
## 2 Wax&Pomade      <tibble [43 x 2]>  price   0.152    0.0705       2.15  0.0313
## 3 FaceMoisturizer <tibble [474 x 2]> price   0.0193   0.00955      2.02  0.0436
## 4 Masks           <tibble [184 x 2]> price  -0.0486   0.0246      -1.97  0.0485
```

Now we perform the same over each individual brand.

```r
# set up tibbles with nesting
nested_data <- df %>%
  select(c(brand, sentiment, price)) %>%
  nest(data = c(-brand))


# create log-reg model for each product category
```

```r
nested_models <- nested_data %>%
  mutate(models = map(data, ~ glm(sentiment ~ price, .,
                                  family = "binomial")))

# store model summarys in unnested format
slopes <- nested_models %>%
  mutate(models = map(models, tidy)) %>%
  unnest(cols = c(models)) %>%
  filter(term=="price")

slopes %>%
  filter(p.value < 0.05) %>%
  arrange(p.value)
```

```
## # A tibble: 7 x 7
##   brand             data              term  estimate std.error statistic p.value
##   <chr>             <list>            <chr>    <dbl>     <dbl>     <dbl>   <dbl>
## 1 gimme-beauty      <tibble [24 x 2]> price    0.800    0.339       2.36  0.0183
## 2 ouidad            <tibble [57 x 2]> price    0.134    0.0622      2.15  0.0319
## 3 invisibobble      <tibble [19 x 2]> price    0.433    0.208       2.08  0.0374
## 4 fourth-ray-beauty <tibble [23 x 2]> price    0.629    0.305       2.06  0.0395
## 5 scunci            <tibble [64 x 2]> price    0.355    0.175       2.03  0.0427
## 6 burts-bees        <tibble [44 x 2]> price   -0.205    0.102      -2.00  0.0457
## 7 ogx               <tibble [35 x 2]> price   -0.373    0.187      -1.99  0.0461
```

Four different product categories and seven brands reported significant relationships (at the $\alpha = 0.5$ level) between the price and sentiment. Both have a variety of positive or negative coefficients (log-odds).

Part (3)

Model to predict rating from description text.

We begin with a setup for train/test split.

```r
# set seed
set.seed(7)

desc_df <- ona_df_og %>%
  select(c(id, rating, text, price, class)) %>%
  group_by(id) %>%
  filter(row_number()==1)%>%
  ungroup()

desc_df <- desc_df %>%
  mutate(rating= as.numeric(rating >= 4))

# train test split
train.ind <- sample(1:nrow(desc_df),
                    replace = FALSE,
                    size=floor(0.80 * nrow(desc_df)))

train_desc <- desc_df[train.ind, ]
```

```r
# tidy texts data by brand
tidy_train_desc <- train_desc %>%
  unnest_tokens(word, text) %>%
  filter(!grepl('[0-9]', word)) %>%
  mutate(word = str_remove_all(word, "[:punct:]")) %>%
  anti_join(stop_words) %>%
  mutate(word = wordStem(word))

# cast tidy data into dtm
dtm_train <- tidy_train_desc %>%
  count(id, word, sort=TRUE) %>%
  cast_dtm(id, word, n)

# remove extremely sparse terms
dtm_train <- removeSparseTerms(dtm_train, 0.995)

# inspect corpus
dtm_train %>% inspect()
```

```
## <<DocumentTermMatrix (documents: 10437, terms: 1404)>>
## Non-/sparse entries: 364599/14288949
## Sparsity           : 98%
## Maximal term length: 30
## Weighting          : term frequency (tf)
## Sample             :
##          Terms
## Docs      formula free hair help hydrat moistur natur oil skin smooth
##   2532647       3    6    0    1      3       1     1   1   13      1
##   2541744       3    6    0    2      4       1     2   2   13      0
##   2560931       5    8    0    2      3       2     1   1   19      0
##   2566481       1    3    0    1      4       2     2   2   11      1
##   2568145       4    9    0    3      4       0     1   1   15      3
##   2575568       2    8    0    3      3       4     1   1   13      1
##   2583153       2    0    0    1      2       2     0   1   12      1
##   2583373       5    7    0    1      4       0     2   1   14      1
##   2599934       0    1    0    4      1       6     0   4   18      2
##   2601273       3    6    0    2      4       1     2   2   13      0
```

```r
# make syntactically valid names
colnames(dtm_train) = make.names(colnames(dtm_train))

# convert dtm to df
train.data <- dtm_train %>%
  as.matrix() %>%
  as.data.frame()

# make id a column
train.data$id <- as.numeric(rownames(train.data))

# join the Y values, rating
train <- train.data %>%
  left_join(desc_df %>%
              select(c(id, rating)),
            by="id")
```

```
train$rating <- factor(train$rating)
```

Validate using the training set with 5-fold CV. Then train on full training data.

```
## Validation for tuning mtry

levels(train$rating) <- c("no", "yes")

rfGrid <- expand.grid(.mtry = c(floor(sqrt(ncol(train))/3),
                                floor(sqrt(ncol(train))/2),
                                floor(sqrt(ncol(train))),
                                floor(ncol(train)/6)))

tr.control <- trainControl(method = "cv",
                           number=4,
                           search = 'grid',
                           classProbs = TRUE,
                           verboseIter = TRUE,
                           summaryFunction = twoClassSummary)

cv_rf_model <- train(rating ~ . -id,
                     data = train,
                     method = "rf",
                     metric = "ROC",
                     tuneGrid = rfGrid,
                     ntree=100,
                     trControl = tr.control)

levels(train$rating) <- c(0, 1)

# print validation metrics
print(cv_rf_model)

# print validation metrics
ggplot(cv_rf_model) +
  geom_point(colour = "red", size = 4) +
  geom_line(colour = "red", size = 2) +
  ggtitle("4-Fold CV Tuning") +
  theme_minimal()

# save tuning image
ggsave(paste0("images/desc_cv_rf_tuning.png"))


## Train with full dataset now

start.time <- Sys.time() # start timer

model_rf <- randomForest(rating ~ .-id,
                         data = train,
                         mtry = floor(sqrt(ncol(train))/2),  # from tuning
```

```
                        ntree = 500)

elapsed.time <- round((Sys.time() - start.time), 3) # stop timer

# report train time
cat("Finished training in ", elapsed.time, " minutes\n")

save.image("Rdata/models_checkpoint2.RData")
```

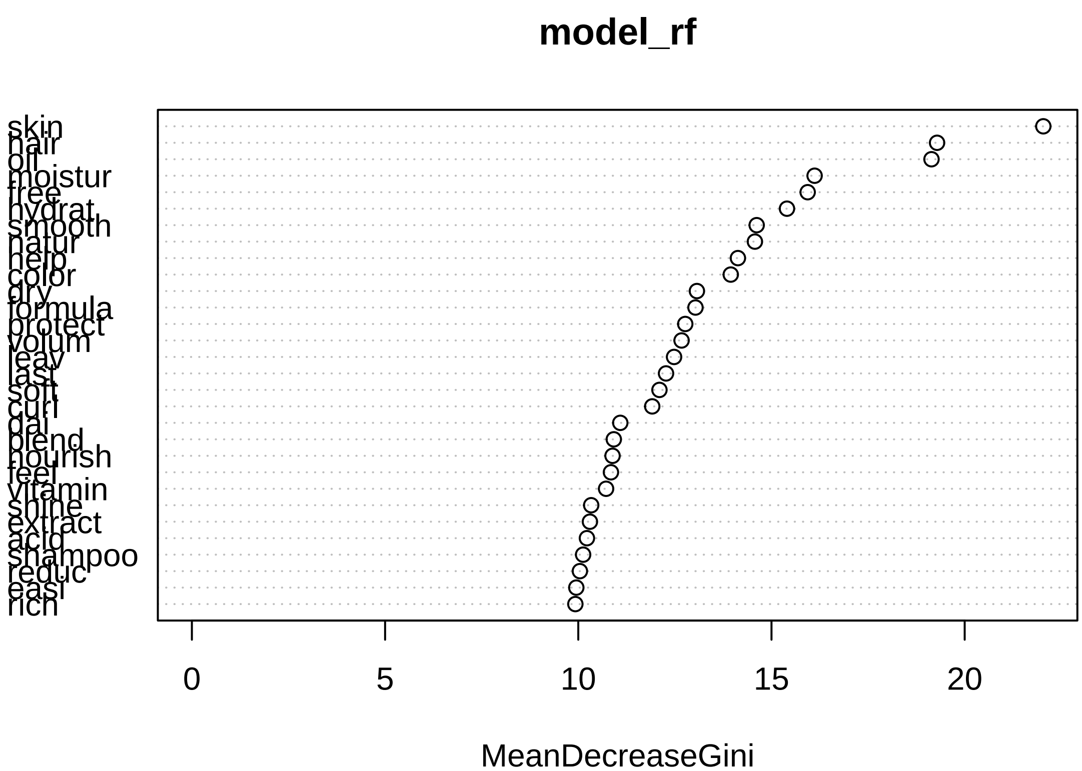
```
load("Rdata/models_checkpoint2.RData")

# report train metrics
preds <- predict(model_rf, newdata = train)
cat("Training Accuracy: ", mean(train$rating == preds))
```

```
## Training Accuracy:  0.977963
```

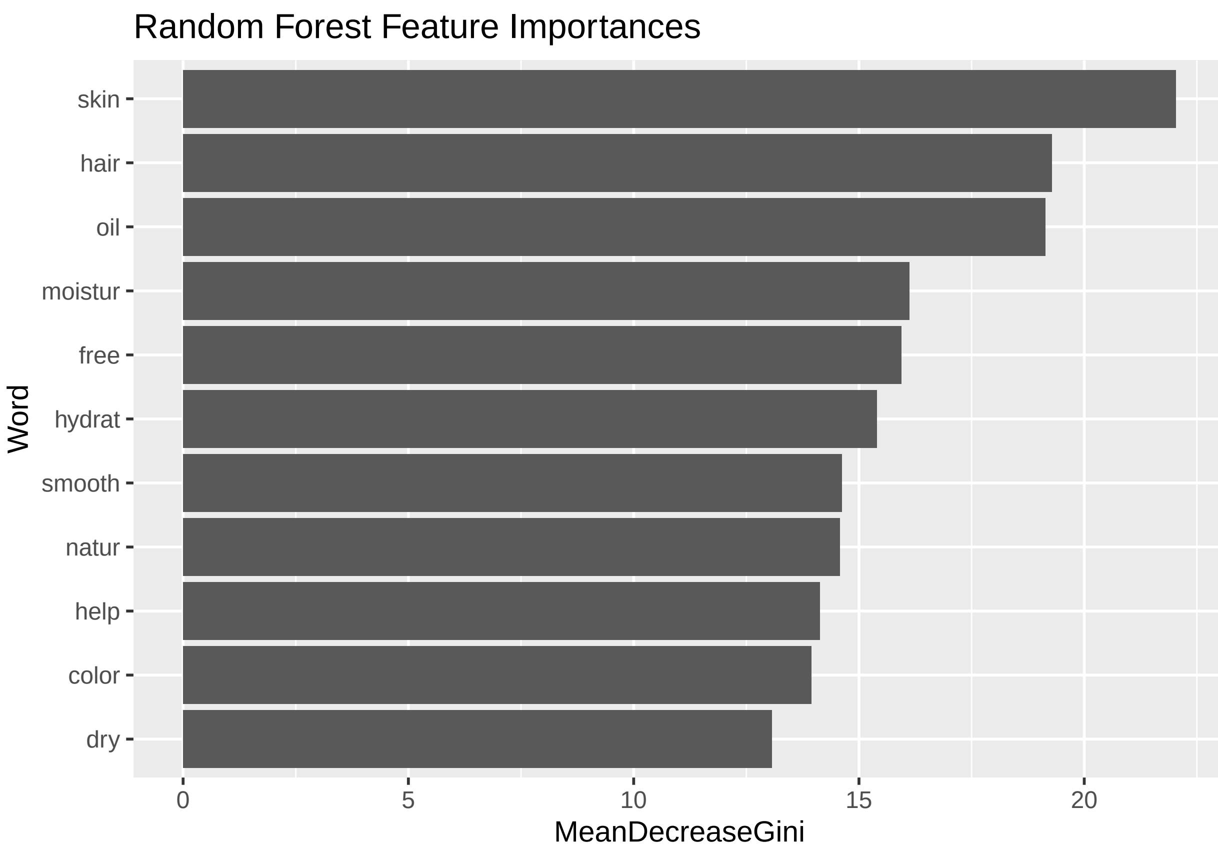
Feature importances.

```
# setup imp object
imp <- varImpPlot(model_rf) %>%
  as.data.frame() %>%
  arrange(desc(MeanDecreaseGini)) %>%
  head(n=11)
```



```
imp$word <- rownames(imp) # row names to column

imp %>%
ggplot(aes(x=reorder(word, MeanDecreaseGini), weight=MeanDecreaseGini)) +
  geom_bar() +
  scale_fill_discrete(name="Variable Group") +
  ylab("MeanDecreaseGini") +
  xlab("Word") +
  ggtitle("Random Forest Feature Importances") +
  coord_flip()
```




```
ggsave(paste0("images/desc_feat_imp.png"))
```

Notice the impact that hair and skin have on rating (as seen for sentiment analysis).

Threshold calculation.

```
test_desc <- desc_df[-train.ind, ]

# tidy revs data by brand
tidy_test_desc <- test_desc %>%
  unnest_tokens(word, text) %>%
  filter(!grepl('[0-9]', word)) %>%
  mutate(word = str_remove_all(word, "[:punct:]")) %>%
  anti_join(stop_words) %>%
```

```r
  mutate(word = wordStem(word))

# cast tidy data into dtm
dtm_test <- tidy_test_desc %>%
  count(id, word, sort=TRUE) %>%
  cast_dtm(id, word, n)

# make syntactically valid names
colnames(dtm_test) = make.names(colnames(dtm_test))

# convert dtm to df
test.data <- dtm_test %>%
  as.matrix() %>%
  as.data.frame()

# make id a column
test.data$id <- as.numeric(rownames(test.data))

# rename important vars
#test.data <- test.data %>%
#  rename(rating.x = rating)

# join the Y values, rating
test <- test.data %>%
  left_join(desc_df %>%
              select(c(id, rating)),
            by="id")

# change to factor
test$rating <- as.factor(test$rating)


# set seed for validation split
set.seed(77)

val.ind <- sample(1:nrow(test),
                  replace = FALSE,
                  size=500)

probs <- predict(model_rf, newdata = test[val.ind,], type="prob")[, 2]

val_metrics <- data.frame()
for (threshold in seq(0.31, 0.7, 0.01)){
  val_threshold <- as.factor(as.numeric(probs > threshold))
  val_table <- table(val_threshold, test$rating[val.ind])
  temp <- confusionMatrix(val_table, positive = "1")
  model <- "Random Forest"
  temp <- data.frame(cbind(model, cbind(threshold, t(temp$byClass), t(temp$overall))))
  val_metrics <- rbind(val_metrics, temp)
}

val_metrics %>%
  select(model, threshold, Accuracy, Sensitivity, Specificity, Balanced.Accuracy) %>%
```

```
mutate_at(c("Sensitivity", "Accuracy",
            "Specificity", "Balanced.Accuracy"), as.numeric) %>%
mutate(Gmean = sqrt(Sensitivity * Specificity)) %>%
arrange(desc(Accuracy))
```

```
##            model threshold Accuracy Sensitivity Specificity Balanced.Accuracy
## 1  Random Forest       0.5    0.656   0.9213836  0.19230769         0.5568457
## 2  Random Forest      0.55    0.654   0.8805031  0.25824176         0.5693725
## 3  Random Forest      0.49    0.652   0.9213836  0.18131868         0.5513512
## 4  Random Forest      0.51    0.652   0.9119497  0.19780220         0.5548759
## 5  Random Forest      0.53    0.652   0.8930818  0.23076923         0.5619255
## 6  Random Forest      0.48    0.650   0.9245283  0.17032967         0.5474290
## 7  Random Forest      0.54    0.650   0.8867925  0.23626374         0.5615281
## 8  Random Forest      0.56    0.650   0.8679245  0.26923077         0.5685776
## 9  Random Forest      0.52    0.648   0.8993711  0.20879121         0.5540811
## 10 Random Forest      0.57    0.646   0.8522013  0.28571429         0.5689578
## 11 Random Forest      0.31    0.644   0.9905660  0.03846154         0.5145138
## 12 Random Forest      0.32    0.644   0.9874214  0.04395604         0.5156887
## 13 Random Forest      0.33    0.644   0.9874214  0.04395604         0.5156887
## 14 Random Forest      0.34    0.644   0.9842767  0.04945055         0.5168636
## 15 Random Forest      0.35    0.644   0.9842767  0.04945055         0.5168636
## 16 Random Forest      0.36    0.644   0.9811321  0.05494505         0.5180386
## 17 Random Forest      0.47    0.644   0.9308176  0.14285714         0.5368374
## 18 Random Forest      0.37    0.642   0.9779874  0.05494505         0.5164662
## 19 Random Forest      0.39    0.642   0.9716981  0.06593407         0.5188161
## 20 Random Forest      0.45    0.642   0.9496855  0.10439560         0.5270406
## 21 Random Forest      0.46    0.640   0.9402516  0.11538462         0.5278181
## 22 Random Forest      0.38    0.638   0.9716981  0.05494505         0.5133216
## 23 Random Forest       0.4    0.638   0.9654088  0.06593407         0.5156714
## 24 Random Forest      0.59    0.638   0.8081761  0.34065934         0.5744177
## 25 Random Forest      0.41    0.636   0.9622642  0.06593407         0.5140991
## 26 Random Forest      0.42    0.636   0.9591195  0.07142857         0.5152740
## 27 Random Forest      0.43    0.636   0.9559748  0.07692308         0.5164490
## 28 Random Forest      0.44    0.636   0.9496855  0.08791209         0.5187988
## 29 Random Forest      0.58    0.636   0.8270440  0.30219780         0.5646209
## 30 Random Forest       0.6    0.630   0.7893082  0.35164835         0.5704783
## 31 Random Forest      0.62    0.618   0.7515723  0.38461538         0.5680939
## 32 Random Forest      0.61    0.614   0.7610063  0.35714286         0.5590746
## 33 Random Forest      0.63    0.612   0.7295597  0.40659341         0.5680766
## 34 Random Forest      0.64    0.612   0.7075472  0.44505495         0.5763011
## 35 Random Forest      0.65    0.610   0.6823899  0.48351648         0.5829532
## 36 Random Forest      0.66    0.600   0.6477987  0.51648352         0.5821411
## 37 Random Forest      0.67    0.582   0.6037736  0.54395604         0.5738648
## 38 Random Forest      0.68    0.574   0.5786164  0.56593407         0.5722752
## 39 Random Forest      0.69    0.568   0.5503145  0.59890110         0.5746078
## 40 Random Forest       0.7    0.568   0.5283019  0.63736264         0.5828323
##        Gmean
## 1  0.4209384
## 2  0.4768466
## 3  0.4087347
## 4  0.4247183
## 5  0.4539777
## 6  0.3968307
## 7  0.4577302
```

```
## 8  0.4833963
## 9  0.4333368
## 10 0.4934431
## 11 0.1951889
## 12 0.2083342
## 13 0.2083342
## 14 0.2206196
## 15 0.2206196
## 16 0.2321817
## 17 0.3646559
## 18 0.2318093
## 19 0.2531166
## 20 0.3148698
## 21 0.3293791
## 22 0.2310628
## 23 0.2522961
## 24 0.5247025
## 25 0.2518849
## 26 0.2617414
## 27 0.2711762
## 28 0.2889444
## 29 0.4999309
## 30 0.5268386
## 31 0.5376488
## 32 0.5213329
## 33 0.5446413
## 34 0.5611572
## 35 0.5744099
## 36 0.5784266
## 37 0.5730849
## 38 0.5722401
## 39 0.5740940
## 40 0.5802757
```

Testing Performance

```r
# get test metrics
probs <- predict(model_rf, newdata = test[-val.ind,], type="prob")[, 2]

threshold <- as.factor(as.numeric(probs > 0.5))
table <- table(threshold, test$rating[-val.ind])
confusionMatrix(table, positive = "1")
```

```
## Confusion Matrix and Statistics
##
##
## threshold    0    1
##         0  151   78
##         1  601 1280
##
##                Accuracy : 0.6782
##                  95% CI : (0.6578, 0.6981)
```

```
##     No Information Rate : 0.6436
##     P-Value [Acc > NIR] : 0.0004522
##
##                   Kappa : 0.1697
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9426
##             Specificity : 0.2008
##          Pos Pred Value : 0.6805
##          Neg Pred Value : 0.6594
##              Prevalence : 0.6436
##          Detection Rate : 0.6066
##    Detection Prevalence : 0.8915
##       Balanced Accuracy : 0.5717
##
##        'Positive' Class : 1
##
```

ROC Curve

```
# train probs
probs <- predict(model_rf, newdata = train, type="prob")[, 2]
pred_train <- prediction(probs, train$rating)
rocX <- roc(train$rating, probs)
auc_train <- rocX$auc

cat("ROC-AUC from Training Data: ", auc_train, "\n")
```

```
## ROC-AUC from Training Data:  0.9986156
```

```
# test probs
probs <- predict(model_rf, newdata = test[-val.ind, ], type="prob")[, 2]
pred_test <- prediction(probs, test$rating[-val.ind])
rocX <- roc(test$rating[-val.ind], probs)
auc_test <- rocX$auc

cat("ROC-AUC from Testing Data: ", auc_test, "\n")
```

```
## ROC-AUC from Testing Data:  0.6695758
```

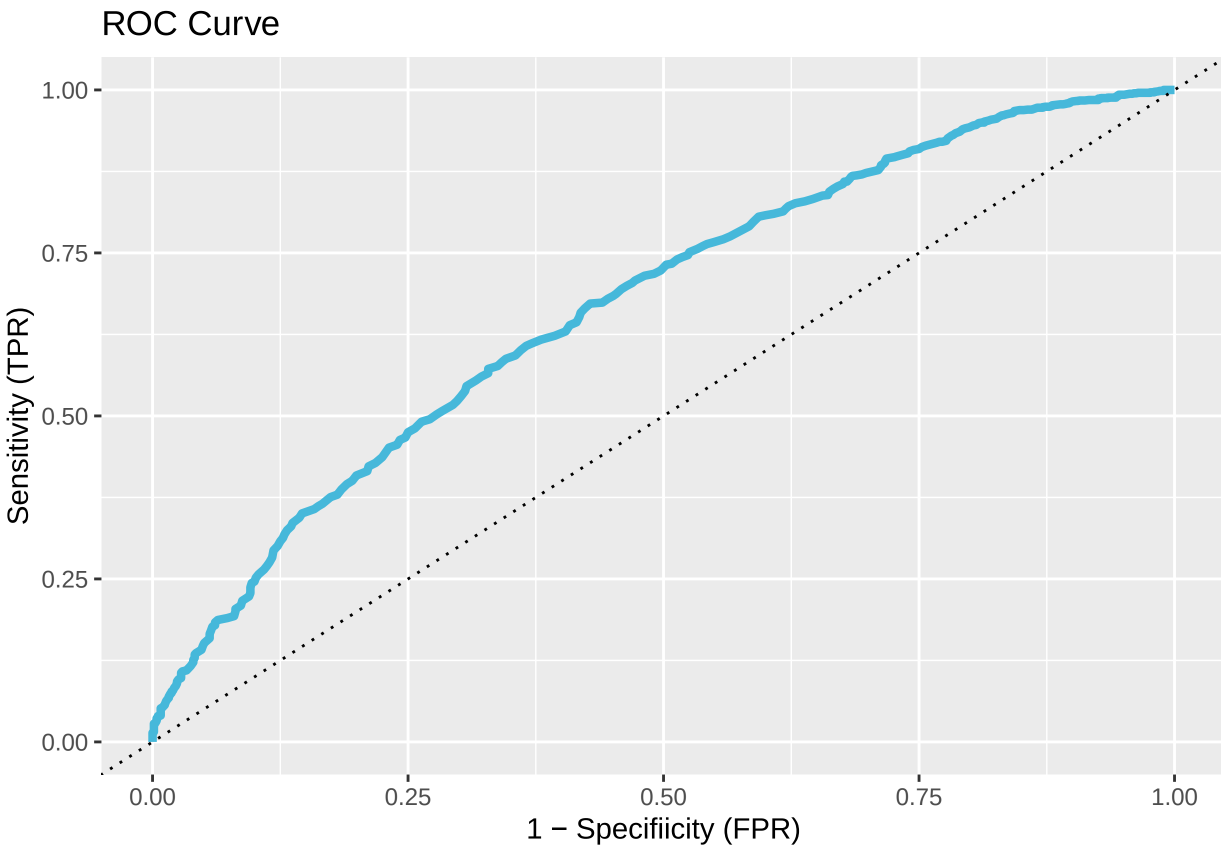
```
# get training and testing ROC curve

ptrain_df <- data.frame(x = performance(pred_train, "sens", "fpr")@x.values[[1]],
                     y = performance(pred_train, "sens", "fpr")@y.values[[1]])
ptest_df <- data.frame(x = performance(pred_test, "sens", "fpr")@x.values[[1]],
                     y = performance(pred_test, "sens", "fpr")@y.values[[1]])

cols <- c("#5CB85C", "#46B8DA")

ggplot() +
  geom_line(data = ptest_df, aes(x = x, y = y), color = cols[2], lwd=1.5) +
  labs(color = "Model") +
  xlab("1 - Specifiicity (FPR)") +
```

```
  ylab("Sensitivity (TPR)") +
  geom_abline(intercept = 0, slope = 1, linetype = "dotted",
              color = "black", size = 0.5) +
  theme(legend.position = "right") +
  theme(legend.position="none") +
  ggtitle("ROC Curve")
```




```
ggsave(paste0("images", "/desc_results_roc.png"))
```