Assignment 3

Aulia Dini Rafsanjani   Chia Wen Cheng   Wenqing Qian

```r
library(tidyverse)
library(readxl)
library(tidytext)
library(corpus)
library(rpart)
library(rpart.plot)
library(rattle)
library(caret)
library(class)
```

Data split and hand-coding

```r
set.seed(622)
posts <- read.csv("posts.csv")
pid <- sample(x = 1:nrow(posts), size = 200)
posts_sample <- posts[pid,]
posts_sample <- posts_sample %>%
  select(-X) %>%
  rename(id = X.1)
# write.csv(x = posts_sample[1:67,], file = "posts/posts1.csv", row.names = FALSE)
# write.csv(x = posts_sample[68:133,], file = "posts/posts2.csv", row.names = FALSE)
# write.csv(x = posts_sample[134:200,], file = "posts/posts3.csv", row.names = FALSE)
```

```r
# posts_lab1 <- read_xlsx("posts/labelled_posts1.xlsx") %>%
#   mutate(date_utc = as.POSIXct(date_utc)) %>%
#   select(-collect_time)
# posts_lab2 <- read_xlsx("posts/labelled_posts2.xlsx") %>%
#   select(-collect_time)
# posts_lab3 <- read_xlsx("posts/labelled_posts3.xlsx") %>%
#   select(-collect_time)
# posts_lab <- posts_lab1 %>%
#   bind_rows(posts_lab2) %>%
#   bind_rows(posts_lab3)
# xlsx::write.xlsx(x = posts_lab, file = "posts/labelled_posts.xlsx", row.names = FALSE)

posts_pn <- read_xlsx("posts/labelled_posts.xlsx")
posts_pn <- posts_pn %>%
  filter(label != "neutral") %>%   # Drop neutral posts
  mutate(label = factor(label))
nrow(posts_pn)
```

```
## [1] 81
```

```
glimpse(posts_pn)
```

```
## Rows: 81
## Columns: 9
## $ id        <dbl> 616, 595, 156, 846, 1211, 1069, 793, 1141, 981, 862, 340, 10~
## $ date_utc  <dttm> 2023-02-27 05:00:00, 2023-02-28 05:00:00, 2023-02-26 05:00:~
## $ timestamp <dbl> 1677484415, 1677594303, 1677418221, 1677867953, 1677090312, ~
## $ title     <chr> "When you?re starving but they?re all out of Vegan burgers a~
## $ text      <chr> NA, "From The Guardian (very vegan friendly news publication~
## $ subreddit <chr> "ShittyVeganFoodPorn", "exvegans", "vegan", "52weeksofcookin~
## $ comments  <dbl> 15, 7, 4, 5, 15, 3, 1, 8, 17, 11, 25, 7, 7, 15, 5, 1046, 13,~
## $ url       <chr> "https://www.reddit.com/r/ShittyVeganFoodPorn/comments/11d66~
## $ label     <fct> negative, negative, negative, positive, positive, negative, ~
```

We have 81 posts left after filtering out the "neutral" posts.

Pre-processing

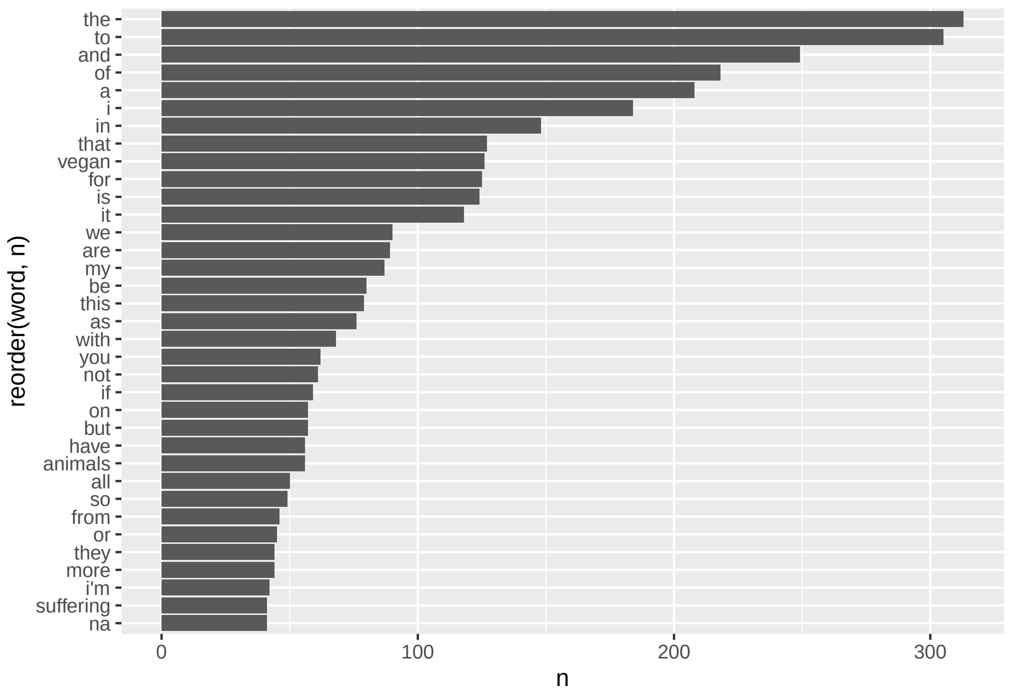
Since only some posts contain texts (some without texts may contain images we didn't scrapped), we created a new variable (column) that combined the post title and post text into one string for each post.

```
posts_pn$title_text <- paste(posts_pn$title, posts_pn$text)
```

Tokenize: split up the titles and texts into individual words

We first used `unnest_tokens` to break down the strings into individual strings with just one word and get the most frequently used words.

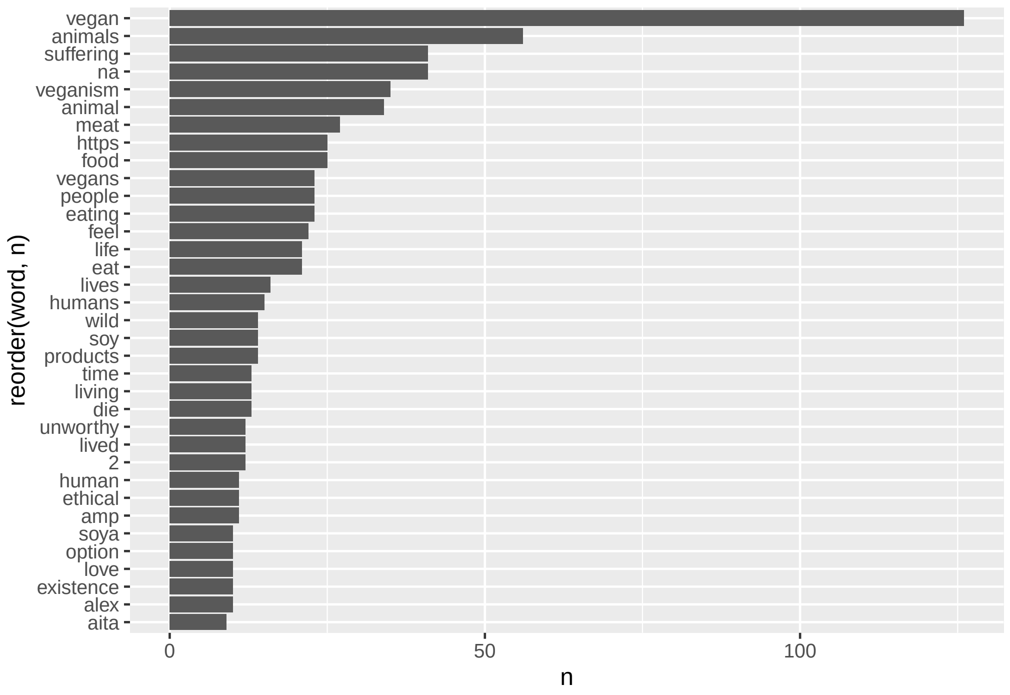
```
posts_pn %>%
  unnest_tokens(word, "title_text") %>%
  count(word, sort = TRUE) %>%
  arrange(desc(n)) %>%
  head(35) %>%
  ggplot(aes(x = reorder(word, n), y = n)) +
  geom_bar(stat = "identity") +
  coord_flip()
```



Exclude stop words: remove frequent words that are too generally seen but uninformative

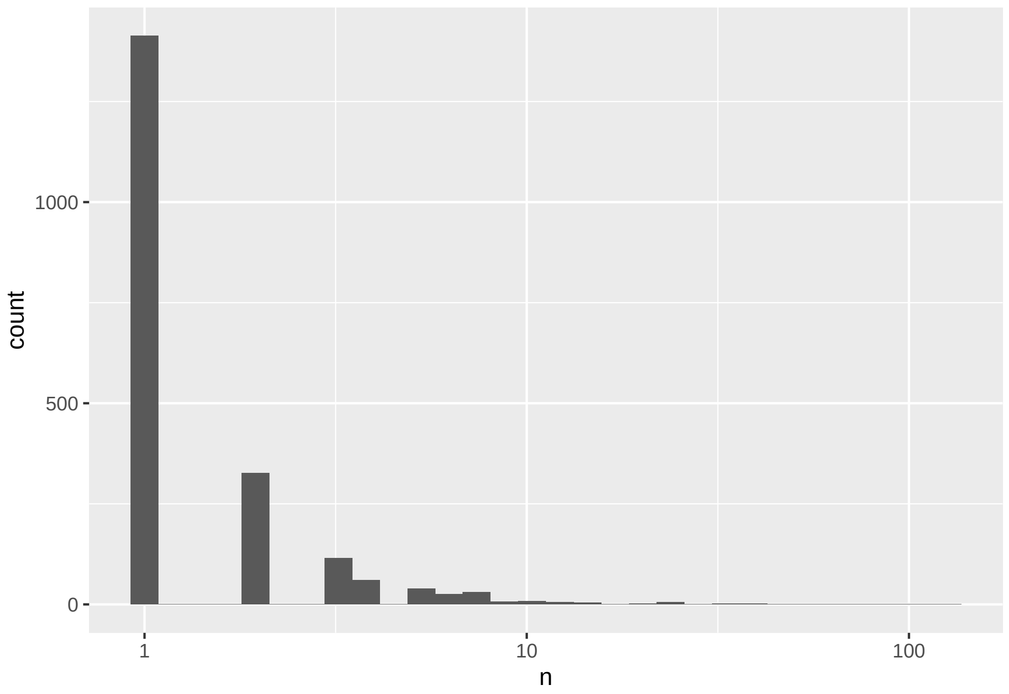
There were a lot of stop words in the result. Like we'd done in assignment 2, we'd like to remove them from the graph to catch more relevant words regarding our topic of "vegan." To do this, we used the `stop_words` from the `tidytext` package and use an `anti_join` to remove all instances of that word.

```r
posts_pn %>%
  unnest_tokens(word, "title_text") %>%
  anti_join(stop_words) %>%
  count(word, sort = TRUE) %>%
  arrange(desc(n)) %>%
  head(35) %>%
  ggplot(aes(x = reorder(word, n), y = n)) +
  geom_bar(stat = "identity") +
  coord_flip()
```



We now had top 35 most frequently appeared words in the titles or texts we'd collected excluding the stop words. We'd also like to take a look at the distribution of word counts by using a histogram on a log scale.

```
posts_pn %>%
  unnest_tokens(word, "title_text") %>%
  anti_join(stop_words) %>%
  count(word, sort = TRUE) %>%
  ggplot(aes(n)) +
  geom_histogram() +
  scale_x_log10()
```

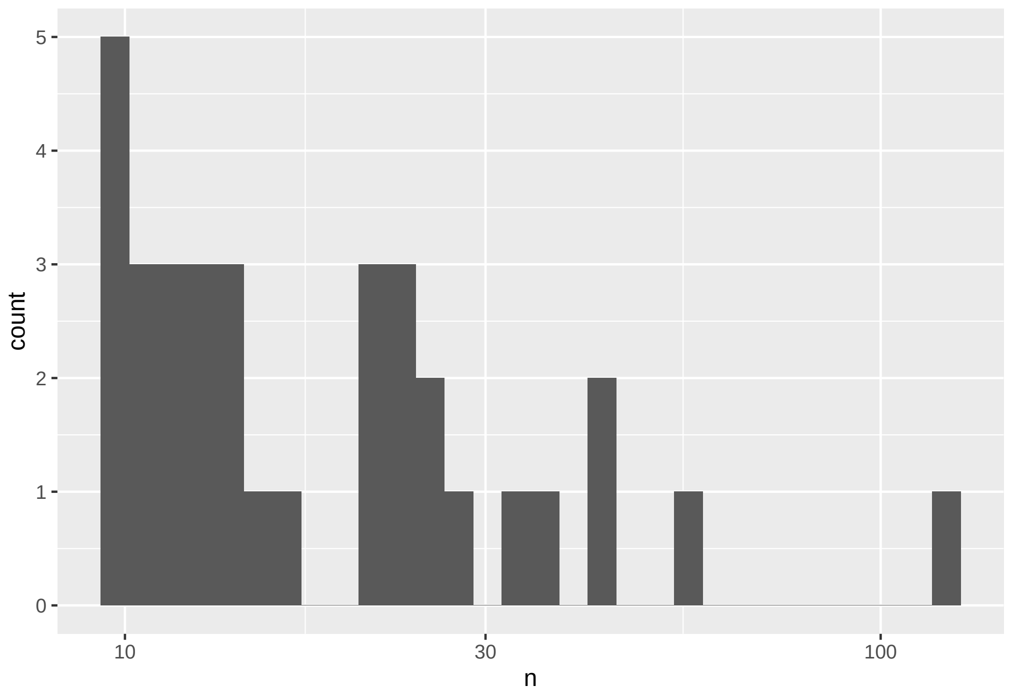


We had more than 1,300 words that were used only once and several words that were infrequently used.

Remove infrequent words: remove infrequent words that may be mistakes or may skew our models of fit

We removed some of the frequently occurred words since they may simply be typos or would affect our model of prediction performance. A frequency of 10 was chosen for this process because from the chart shown above, 10-time looked like a distinguishable threshold of the times of usage.

```
posts_pn %>%
  unnest_tokens(word, "title_text") %>%
  anti_join(stop_words)%>%
  count(word, sort = TRUE) %>%
  filter(n >= 10) %>%
  ggplot(aes(n)) +
  geom_histogram() +
  scale_x_log10()
```

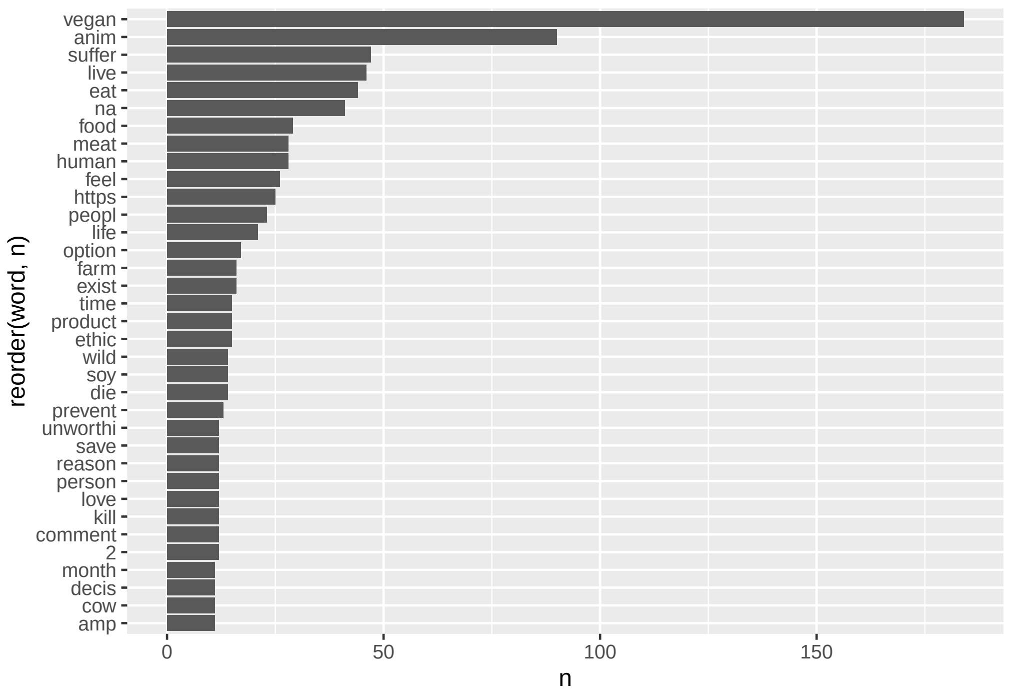


Now we got a histogram chart summing words that were used from 10 times to more than 100 times. There were 5 words used 10 times, and then the highest number of words used for the following frequencies was 3.

Stem words to groups: gather similar words to a single root

We then cleaned up our data a bit more by stemming in order to group similar words together under a single root.

```
posts_pn %>%
  unnest_tokens(word, "title_text") %>%
  anti_join(stop_words)%>%
  mutate(word = corpus::text_tokens(word, stemmer = "en") %>% unlist()) %>%
  count(word, sort = TRUE) %>%
  filter(n >= 10) %>%
  arrange(desc(n)) %>%
  head(35) %>%
  ggplot(aes(x = reorder(word, n), y = n)) +
  geom_bar(stat = "identity") +
  coord_flip()
```



By grouping similar words with Snowball Stemming Library built in the corpus function, we got a more concise result of top 35 used words, excluding stop words, in titles and texts we'd collected.

We did not remove digits or numbers from our data of collection because people seemed to be used to relate their sentiments or statements with numbers indicating their years of being a 'vegan.' Other usage of numbers included asking or sharing a recipe, which were usually neutral and had been excluded for the further steps.

Creating features for machine learning: Bag of Words

After cleaning the data, we created features for our machine learning models. We took all of the words in our corpus and counted the number of times that they appeared in the titles and texts of our collected data. We would like a result of a sparse data frame, in which the columns represent each word, and rows represent the titles and texts.

Like what we've done in the pre-processing, we extracted all of the words in our collected data and broke them down into individual words, removed stop words as well as infrequent words that appeared less than ten times, and stemmed similar words into groups.

```r
word_list <- posts_pn %>%
  unnest_tokens(word, "title_text") %>%
  anti_join(stop_words) %>%
  mutate(word = corpus::text_tokens(word, stemmer = "en") %>% unlist()) %>%   # stemming
  count(word, sort = TRUE) %>%   # count by words
  filter(n >= 10)%>%   # set threshold of frequency to 10
  pull(word)
```

Next, we generated counts of words that each posts contained. To do this, we took the bag of words for each combined title and text, and then find which words occurred how many times, and included that in our final dataset before conducting machine learning.

```
post_features <- posts_pn %>%
  unnest_tokens(word, "title_text") %>%
  anti_join(stop_words)%>%
  mutate(word = corpus::text_tokens(word, stemmer = "en") %>% unlist()) %>%
  filter(word %in% word_list) %>%        # filter for only words in the wordlist
  count(id, word) %>%                    # count word usage by ID
  spread(word, n) %>%                    # convert to wide format
  replace(is.na(.), 0)                   # replace NAs with 0

class(post_features)   # no need to change the type of this data since it's already a data frame
```

```
## [1] "tbl_df"     "tbl"        "data.frame"
```

It looked like we had a greatest difference of 40 and 0 in counts of our bagged words. But since the count 40 only appeared once and we had a bunch of 0s in our counts, we decided not to transform them.

We added an additional variable indicating the length of the combined titles and texts in number of words for each post.

```
# create a variable with length of title_text in number of words
post_length <- data.frame(post_features$id,
                          nchar(posts_pn$title_text, type = "chars", keepNA = TRUE))
colnames(post_length)[1] <- "id"
colnames(post_length)[2] <- "length of post"
```

We did not recognize any other stop words that should be excluded from our dataset. We had concerns on “https,” but it informed the shared habits of including urls among people posting relative posts of vegan. So leaving it for our prediction models should not be an issue.

We merged the counts of words bagged with our original dataset. “id” is used to get the labels matched up with our features.

```
full_data <- posts_pn %>%
  right_join(post_features, by = "id") %>%
  right_join(post_length, by = "id") %>%
  select(-title, -text, -timestamp, -date_utc, -subreddit,
         -comments, -url, -title_text) # Remove extra variables
```

We now had the full data with id for identifying specific rows and for splitting training and testing data for prediction, our features of each bagged word, and the sentimentally positive or negative labels.

Train and test split

To get repetitive results every time, we used the function `set.seed()`. 30% of our data was used for testing and the rest for training.

```
# 30% holdout sample
set.seed(937)
test <- full_data %>%
  sample_frac(.3)

# Rest in the training set
```

```r
train <- full_data %>%
  anti_join(test, by = "id") %>%
  select(-id)
test <- test %>%
  select(-id)
```

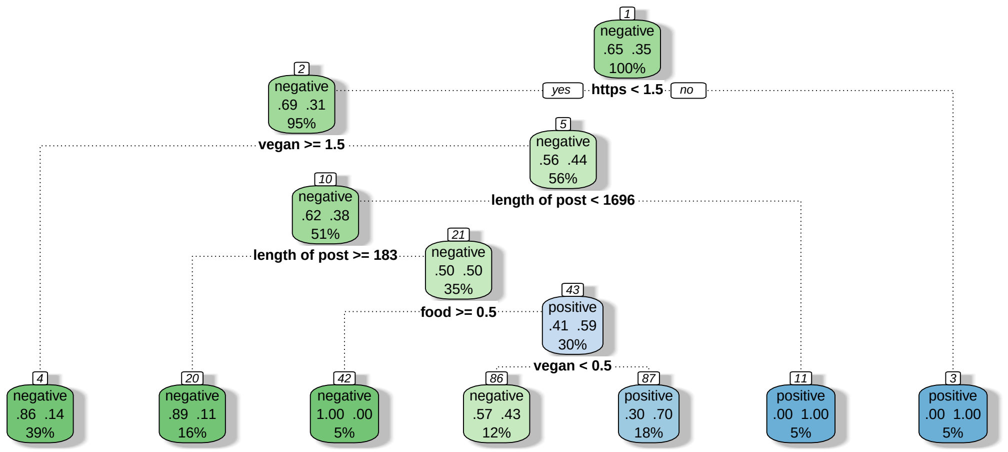
Sentiment prediction

```r
# Create separate training and testing features and labels objects
train_features <- train %>% select(-label)
test_features <- test %>% select(-label)

train_label <- train$label
test_label <- test$label
```

Decision tree

Fitting model

```r
treemod <- rpart(label ~ .,
                 data = train,
                 method = "class",
                 control = rpart.control(minbucket = 3))

fancyRpartPlot(treemod, sub = "")
```



The decision tree model shows that if our keyword "vegan" appears frequently, the post will be more likely to be negative. Besides, posts that are very long or contain many urls are inclined to be positive.

Evaluating model

```
pred_tree <- predict(object = treemod, newdata = test)
test_pred_tree <- data.frame(score = pred_tree[,2], actual = test$label)

test_pred_tree <- test_pred_tree %>%
  arrange(desc(score))
test_pred_tree$pred <- "negative"

top_scores <- floor(nrow(test_pred_tree) * 0.4)
test_pred_tree$pred[1:top_scores] <- "positive"
test_pred_tree$pred <- as.factor(test_pred_tree$pred)

pred_tab_tree <- table(Prediction = test_pred_tree$pred,
                       Reference = test_pred_tree$actual)   # confusion matrix
pred_tab_tree
```

```
##           Reference
## Prediction negative positive
##   negative        6        9
##   positive        6        3
```

```
(pred_tab_tree[1, 1] + pred_tab_tree[2, 2]) / sum(pred_tab_tree)   # accuracy
```

```
## [1] 0.375
```

```
precision(pred_tab_tree, relevant = "positive")   # precision
```

```
## [1] 0.3333333
```

```
recall(pred_tab_tree, relevant = "positive")   # recall
```

```
## [1] 0.25
```

The threshold for cutoff is set to 40%. Unfortunately the first decision tree model doesn't perform really well, especially when predicting positive posts.

Looping through models

Function `rpart.control` has provided many parameters about decision trees, among which `minbucket` (minimum number of observations in terminal nodes) and `maxdepth` (maximum depth of trees) will be adjusted in this case. Different thresholds for cutoff will also be tested.

```
# minbucket
min_term_node <- c(3, 5, 10, 15, 20)
# maxdepth
max_depth <- c(2, 3, 4, 5)
# threshold
percent <- c(0.3, 0.4, 0.5, 0.6, 0.7)

nmods_tree <- length(min_term_node) * length(max_depth) * length(percent)
```

```r
results_tree <- data.frame(min_term_node = rep(NA, nmods_tree),
                           max_depth = rep(NA, nmods_tree),
                           percent = rep(NA, nmods_tree),
                           accuracy = rep(NA, nmods_tree),
                           precision = rep(NA, nmods_tree),
                           recall = rep(NA, nmods_tree),
                           f_score = rep(NA, nmods_tree))

mod_num_tree <- 1

# The loop
for(i in 1:length(min_term_node)) {
  for(j in 1:length(max_depth)) {
    for (k in 1:length(percent)) {
      b <- min_term_node[i]
      d <- max_depth[j]
      p <- percent[k]

      treemod <- rpart(label ~ .,
                       data = train,
                       method = "class",
                       control = rpart.control(minbucket = b, maxdepth = d))

      pred_tree <- predict(object = treemod, newdata = test)
      test_pred_tree <- data.frame(score = pred_tree[,2], actual = test$label)
      test_pred_tree <- test_pred_tree %>%
        arrange(desc(score))

      test_pred_tree$pred <- "negative"
      top_scores <- floor(nrow(test_pred_tree) * p)
      test_pred_tree$pred[1:top_scores] <- "positive"
      test_pred_tree$pred <- as.factor(test_pred_tree$pred)

      pred_tab_tree <- table(test_pred_tree$pred, test_pred_tree$actual)
      results_tree[mod_num_tree,] <-
        c(b,
          d,
          p,
          (pred_tab_tree[1, 1] + pred_tab_tree[2, 2]) / sum(pred_tab_tree),
          precision(pred_tab_tree, relevant = "positive"),
          recall(pred_tab_tree, relevant = "positive"),
          F_meas(pred_tab_tree, relevant = "positive"))

      mod_num_tree <- mod_num_tree + 1
    }
  }
}
```

```r
head(results_tree)
```

```
##   min_term_node max_depth percent  accuracy precision     recall   f_score
## 1             3         2     0.3 0.2916667 0.1428571 0.08333333 0.1052632
## 2             3         2     0.4 0.2083333 0.1111111 0.08333333 0.0952381
## 3              3          2     0.5 0.2500000 0.2500000 0.25000000 0.2500000
## 4              3          2     0.6 0.2500000 0.2857143 0.33333333 0.3076923
## 5              3          2     0.7 0.3333333 0.3750000 0.50000000 0.4285714
## 6              3          3     0.3 0.2916667 0.1428571 0.08333333 0.1052632
```

```
# Best accuracy? Top 5 in descending order
results_tree %>%
  arrange(desc(accuracy)) %>%
  head()
```

```
##   min_term_node max_depth percent  accuracy precision    recall   f_score
## 1             3         3     0.7 0.5833333 0.5625000 0.7500000 0.6428571
## 2             3         4     0.7 0.5833333 0.5625000 0.7500000 0.6428571
## 3             5         2     0.6 0.5833333 0.5714286 0.6666667 0.6153846
## 4             5         3     0.6 0.5833333 0.5714286 0.6666667 0.6153846
## 5             3         3     0.5 0.5000000 0.5000000 0.5000000 0.5000000
## 6             3         3     0.6 0.5000000 0.5000000 0.5833333 0.5384615
```

```
# Best recall? Top 5 in descending order
results_tree %>%
  arrange(desc(recall)) %>%
  head()
```

```
##   min_term_node max_depth percent  accuracy precision    recall   f_score
## 1             3         3     0.7 0.5833333 0.5625000 0.7500000 0.6428571
## 2             3         4     0.7 0.5833333 0.5625000 0.7500000 0.6428571
## 3             5         2     0.6 0.5833333 0.5714286 0.6666667 0.6153846
## 4             5         2     0.7 0.5000000 0.5000000 0.6666667 0.5714286
## 5             5         3     0.6 0.5833333 0.5714286 0.6666667 0.6153846
## 6             5         3     0.7 0.5000000 0.5000000 0.6666667 0.5714286
```

```
# Best precision? Top 5 in descending order
results_tree %>%
  arrange(desc(precision)) %>%
  head()
```

```
##   min_term_node max_depth percent  accuracy precision    recall   f_score
## 1             5         2     0.6 0.5833333 0.5714286 0.6666667 0.6153846
## 2             5         3     0.6 0.5833333 0.5714286 0.6666667 0.6153846
## 3             3         3     0.7 0.5833333 0.5625000 0.7500000 0.6428571
## 4             3         4     0.7 0.5833333 0.5625000 0.7500000 0.6428571
## 5             3         3     0.5 0.5000000 0.5000000 0.5000000 0.5000000
## 6             3         3     0.6 0.5000000 0.5000000 0.5833333 0.5384615
```

```
# Best F-score? Top 5 in descending order
results_tree %>%
  arrange(desc(f_score)) %>%
  head()
```

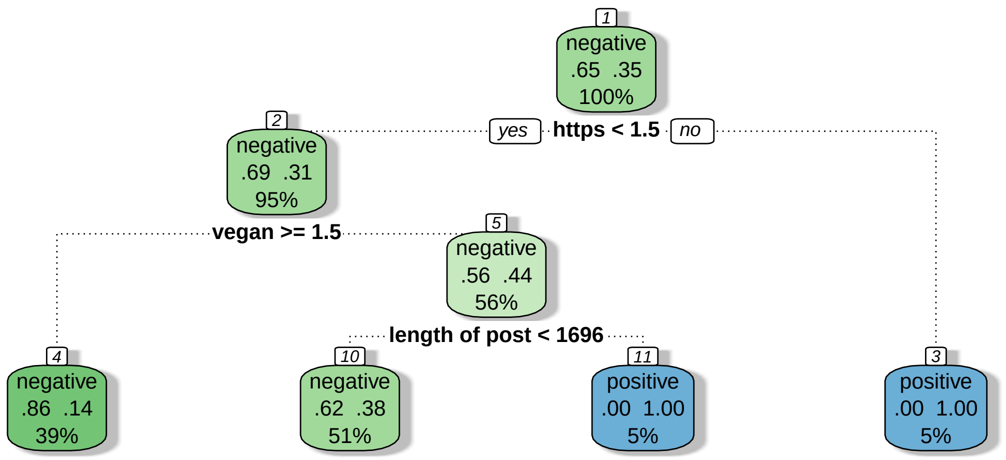
```
##   min_term_node max_depth percent  accuracy precision    recall   f_score
## 1             3         3     0.7 0.5833333 0.5625000 0.7500000 0.6428571
## 2             3         4     0.7 0.5833333 0.5625000 0.7500000 0.6428571
## 3              5          2     0.6 0.5833333 0.5714286 0.6666667 0.6153846
## 4              5          3     0.6 0.5833333 0.5714286 0.6666667 0.6153846
## 5              5          2     0.7 0.5000000 0.5000000 0.6666667 0.5714286
## 6              5          3     0.7 0.5000000 0.5000000 0.6666667 0.5714286
```

Results show that when we set the minimum number of observations for each terminal node to 5, the maximum depth of tree to 2 or 3, and use 0.6 as the cutoff point, the model will reach the highest precision. When the minimum number of observations for each terminal node is 3, the maximum depth of tree is either 3 or 4, and the threshold for cutoff is 0.7, the model will reach the highest recall. All these 4 models have the highest accuracy. Furthermore, we also calculated the F-score—a combination of precision and recall—of each model. Models with the highest recall perform the best in terms of F-scores.

Prediction with final model

A decision tree model with minimum terminal node size of 3, maximum depth of 3, and using 0.7 as threshold to classify will be applied to make predictions, due to its better performance measured in metrics calculated in the last section and its parsimony to the one with equal capacity but having larger potential depth. The model looks like:

```
treemod_best <- rpart(label ~ .,
                      data = train,
                      method = "class",
                      control = rpart.control(minbucket = 3, maxdepth = 3))

fancyRpartPlot(treemod_best, sub = "")
```



Sentiment scores of test posts predicted by the selected decision tree model are:

```
pred_tree_best <- predict(object = treemod_best, newdata = test)
test_pred_tree_best <- data.frame(score = pred_tree_best[,2], actual = test$label)
test_pred_tree_best <- test_pred_tree_best %>%
  arrange(desc(score))
test_pred_tree_best
```

```
##        score   actual
## 4  1.0000000 negative
## 7  1.0000000 positive
## 11 1.0000000 negative
## 14 1.0000000 negative
## 17 1.0000000 negative
## 1  0.3793103 negative
## 5  0.3793103 negative
## 9  0.3793103 positive
## 10 0.3793103 positive
## 13 0.3793103 positive
## 15 0.3793103 positive
## 18 0.3793103 positive
## 20 0.3793103 positive
## 21 0.3793103 negative
## 22 0.3793103 positive
## 23 0.3793103 positive
## 24 0.3793103 negative
## 2  0.1363636 negative
## 3  0.1363636 negative
## 6  0.1363636 positive
## 8  0.1363636 negative
## 12 0.1363636 negative
## 16 0.1363636 positive
## 19 0.1363636 positive
```

The performance metrics of this model are:

```
test_pred_tree_best$pred <- "negative"
top_scores_best <- floor(nrow(test_pred_tree_best) * 0.7)
test_pred_tree_best$pred[1:top_scores_best] <- "positive"
test_pred_tree_best$pred <- as.factor(test_pred_tree_best$pred)

pred_tab_tree_best <- table(Prediction = test_pred_tree_best$pred,
                            Reference = test_pred_tree_best$actual)   # confusion matrix
pred_tab_tree_best
```

```
##           Reference
## Prediction negative positive
##   negative        5        3
##   positive        7        9
```

```
(pred_tab_tree_best[1, 1] + pred_tab_tree_best[2, 2]) / sum(pred_tab_tree_best)   # accuracy
```

```
## [1] 0.5833333
```

```
precision(pred_tab_tree_best, relevant = "positive")   # precision
```

```
## [1] 0.5625
```

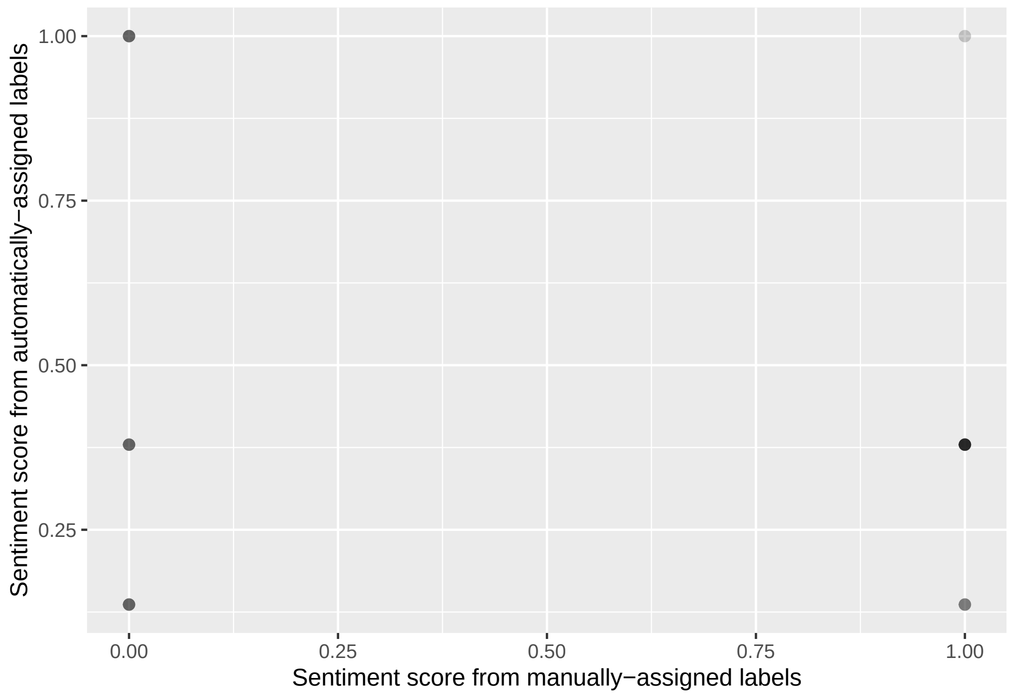
```r
recall(pred_tab_tree_best, relevant = "positive")    # recall
```

```
## [1] 0.75
```

Compared to the first decision tree model, this one outperforms it in terms of all three metrics we calculated, but it is still limited in predicting negative posts as we focused more on positive category when selecting models.

Comparing sentiment scores

```r
test_pred_tree_best %>%
  mutate(actual_score = ifelse(test = actual == "positive", yes = 1, no = 0)) %>%
  ggplot(data = ., mapping = aes(x = actual_score, y = score)) +
  geom_point(size = 2, alpha = 0.2) +
  labs(x = "Sentiment score from manually-assigned labels",
       y = "Sentiment score from automatically-assigned labels")
```



We assigned posts whose actual sentiment labels are “positive” a score of 1 and “negative” a score of 0. Since the structure of decision tree model is simple, there are only 3 distinct values among predicted sentiment scores. The color of points indicate the count of posts. We may find that most positive posts get the median score 0.3793103 since the according point look much darker, while the scores of negative posts distribute more evenly. In fact there is no highly correlated pattern between actual scores and predicted scores, which means the capacity of our model is sort of limited.

```r
test_pred_tree_best
```

```
##          score    actual     pred
## 4  1.0000000 negative positive
## 7  1.0000000 positive positive
## 11 1.0000000 negative positive
## 14 1.0000000 negative positive
## 17 1.0000000 negative positive
## 1  0.3793103 negative positive
## 5  0.3793103 negative positive
## 9  0.3793103 positive positive
## 10 0.3793103 positive positive
## 13 0.3793103 positive positive
## 15 0.3793103 positive positive
## 18 0.3793103 positive positive
## 20 0.3793103 positive positive
## 21 0.3793103 negative positive
## 22 0.3793103 positive positive
## 23 0.3793103 positive positive
## 24 0.3793103 negative negative
## 2  0.1363636 negative negative
## 3  0.1363636 negative negative
## 6  0.1363636 positive negative
## 8  0.1363636 negative negative
## 12 0.1363636 negative negative
## 16 0.1363636 positive negative
## 19 0.1363636 positive negative
```

```
pred_tab_tree_best
```

```
##            Reference
## Prediction negative positive
##   negative        5        3
##   positive        7        9
```

Further looking into automatically-assigned labels and manually-assigned labels, we noticed that 66.6666667% of posts (16 posts) in our test set are predicted to be positive and 33.3333333% to be negative (8 posts), while in fact only 50% of posts have positive sentiment. A major deficiency of this model is that it gives some negative posts pretty high sentiment scores and thus leads to misclassification. The model works better in predicting positive sentiment.

K-Nearest Neighbors

Fitting models

We start to build model using K-Nearest Neighbors. This method checks the class of closest k-neighbors of observation, and takes a vote of class among them to define which class of data will be. Since our data size is relatively small (n_train=57 and n_test=24), we decide to choose value of $k = 1$.

```
# Predicted values from K-NN, with k = 1
knnpred <- knn(train_features,test_features,train_label, k = 1)
```

The `knnpred` provides us the predicted value for each of `test_features` given. After that, we create data frame with predicted and actual value.

```r
pred_actual <- data.frame(predicted = knnpred, actual = test_label)
pred_actual %>% head()
```

```
##   predicted   actual
## 1  positive negative
## 2  negative negative
## 3  negative negative
## 4  negative negative
## 5  negative negative
## 6  negative positive
```

From this table, we can see that the result from predicted and actual value are relatively consistent each other. Some of manually assigned negative sentiment from vegan was correctly predicted by K-NN model with $k = 1$. Then, we evaluate the model performance using `confusionMatrix`.

Model evaluation

```r
pred_actual %>% table()
```

```
##           actual
## predicted  negative positive
##   negative       10        8
##   positive        2        4
```

```r
cm1 <- confusionMatrix(pred_actual %>% table(), positive = "positive")
cm1$overall["Accuracy"]    # accuracy
```

```
##  Accuracy
## 0.5833333
```

```r
cm1$byClass["Sensitivity"]    # recall
```

```
## Sensitivity
##   0.3333333
```

```r
cm1$byClass["Pos Pred Value"]    # precision
```

```
## Pos Pred Value
##      0.6666667
```

In the Confusion Matrix, Sensitivity represent recall score and Pos Pred Value represent precision score. For $k = 1$, the accuracy score is 0.58, the recall score is 0.33, and the precision score is 0.67. Based on this result, recall score is relatively small, so we decide to check various scenarios of k in K-NN method. The value of k is odd numbers to avoid ties.

Looping through models

```r
# look at k=1,3,5,7,9
nb <- c(1,3,5,7,9)

# define total running models
nmods <- length(nb)

# prepare data frame for the output
results <- data.frame(k = rep(NA, nmods),
                      accuracy = rep(NA, nmods),
                      recall = rep(NA, nmods),
                      precision = rep(NA, nmods))

# The model number that we will iterate on (aka models run so far)
mod_num <- 1

# The loop
for (i in 1:length(nb)){
  k <- nb[i]
  # Running the model and find the predictions
  knnpred <- knn(train_features, test_features, train_label, k = k)

  # Create table for actuals and predictions
  pred_actual <- data.frame(predicted = knnpred, actual = test_label)
  pred_actual %>% table()

  # Confusion Matrix
  cm <- confusionMatrix(pred_actual %>% table(), positive = "positive")

  # Store results
  results[mod_num,] <- c(k,
                         cm$overall["Accuracy"],
                         cm$byClass["Sensitivity"],
                         cm$byClass["Pos Pred Value"])

  # Increment the model number
  mod_num <- mod_num + 1
}

# All results are stored in the "results" dataframe
head(results)
```

```
##   k  accuracy    recall precision
## 1 1 0.5833333 0.3333333 0.6666667
## 2 3 0.5000000 0.1666667 0.5000000
## 3 5 0.5000000 0.1666667 0.5000000
## 4 7 0.4583333 0.0000000 0.0000000
## 5 9 0.4583333 0.0000000 0.0000000
```

```r
# Best recall? Top 5 in descending order
results %>% arrange(desc(recall)) %>% head()
```

```
##   k  accuracy    recall precision
## 1 1 0.5833333 0.3333333 0.6666667
## 2 3 0.5000000 0.1666667 0.5000000
## 3 5 0.5000000 0.1666667 0.5000000
## 4 7 0.4583333 0.0000000 0.0000000
## 5 9 0.4583333 0.0000000 0.0000000
```

```
# Best precision? Top 5 in descending order
results %>% arrange(desc(precision)) %>% head()
```

```
##   k  accuracy    recall precision
## 1 1 0.5833333 0.3333333 0.6666667
## 2 3 0.5000000 0.1666667 0.5000000
## 3 5 0.5000000 0.1666667 0.5000000
## 4 7 0.4583333 0.0000000 0.0000000
## 5 9 0.4583333 0.0000000 0.0000000
```

This table shows that as the highest score of accuracy, recall, and precision are reached when $k = 1$. As k increase, the value of three indicators decrease gradually. Starting with the value of $k = 7$, the recall and precision score start to be 0, means that there is not any agreement on actual positive with predicted positive using K-NN method. In other words, the observation are distinct each other, so there is a few similarities among them. Moreover, some journals stated that $k = 1$ tend to create over fitting model, therefore we choose $k = 3$ or $k = 5$ as the best model.

Prediction with final model and comparing sentiment scores

As we find the best $k = 3$ for KNN model, we fit the final model:

```
# Predicted values from K-NN, with k = 3
knnpred <- knn(train_features,test_features,train_label, k = 3)
```

Then, we create table to show our final prediction in compare to the manually-assigned sentiment.

```
pred_actual <- data.frame(predicted = knnpred, actual = test_label)
pred_actual
```

```
##    predicted   actual
## 1   positive negative
## 2   negative negative
## 3   negative negative
## 4   negative negative
## 5   negative negative
## 6   negative positive
## 7   negative positive
## 8   negative negative
## 9   positive positive
## 10  negative positive
## 11  negative negative
## 12  positive negative
## 13  negative positive
## 14  negative negative
## 15  negative positive
## 16  positive positive
## 17  negative negative
## 18  negative positive
## 19  negative positive
## 20  negative positive
## 21  negative negative
## 22  negative positive
## 23  negative positive
## 24  negative negative
```

The performance matrix for this model is :

```
pred_actual %>% table()
```

```
##           actual
## predicted  negative positive
##   negative       10       10
##   positive        2        2
```

```
cm1 <- confusionMatrix(pred_actual %>% table(), positive = "positive")
cm1$overall["Accuracy"]   # accuracy
```

```
## Accuracy
##      0.5
```

```
cm1$byClass["Sensitivity"]   # recall
```

```
## Sensitivity
##   0.1666667
```

```
cm1$byClass["Pos Pred Value"]   # precision
```

```
## Pos Pred Value
##            0.5
```

In this final model, the contingency table shows that the model is relatively good to predict negative sentiments. There are 10 observations accurately predicted to be negative. However, this model shows few accuracy of predicting positive sentiments. There are two actual positive which predicted to be positive based on K-NN method. Shortly, this model has a relatively low performance.

The accuracy score is 0.5, meaning that the model is able to predict true positive and true negative by 50%, among all observations. The recall score is 0.17, meaning that the model is able to predict true positive by 17%, among all actual positive observations. The precision score is 0.5, meaning that the model is able to predict true positive by 50%, among all positive prediction observations. We still suffer from low recall (17%), but we had moderate score of accuracy (50%) and precision (50%).