Unit 12: Natural Language Processing, Text Processing, and Feature Engineering

Nicole Friedl

2025-08-07

Assignment overview

To begin, download the following from the course web book (Unit 12):

- `hw_unit_12_nlp.qmd` (notebook for this assignment)
- `alcohol_tweets.csv` ~5k public Tweets collected from people in New York State from July 2013 - July 2014. Tweets were filtered to contain drinking related keywords (e.g., drunk, beer, party), and were labeled by Amazon MTurk workers to identify tweets that were about the user drinking alcohol. The data set contains the following variables:
  - `tweet_id`: unique Twitter id of the user
  - `user_drinking`: labeled yes/no if the tweet is about the user drinking
  - `text`: The raw text of the tweet. Note: raw text is listed as NA in this dataset if the tweet only contained an image or gif (i.e., no text was present)
- `glove_twitter.csv`: These are GloVe vectors - pretrained semantic embeddings (i.e., features of word meaning) derived from Twitter data by a group from Stanford. We provide you with a CSV version of one of the representation sets, but you can look at the other data and related tutorial on Stanford's website. They have vectors of varying sizes, generated from different large language corpora. These are very useful for examining language semantics.
- A caution about these Twitter data: these are raw language data from Twitter's platform. We have not curated the data in any way. As a result, the text includes content that some may well find offensive and users of these data should take note. Words used and topics discussed could be harsh, offensive, or inflammatory. Foul language is certainly present.

Your goal is to build the best logistic regression model that you can to predict whether a tweet is about a user drinking alcohol (or not). Similar to the neural networks homework, you will have a lot of flexibility in how you approach this assignment. We will include minimum steps you must consider for model building, but feel free to expand EDA beyond the steps listed in order to build the best model you can.

Your assignment is due Wednesday at 8:00 PM. Let's get started!

Setup

Load packages, paths, and function scripts you may need, including parallel processing code.

```
options(conflicts.policy = "depends.ok")
devtools::source_url("https://github.com/jjcurtin/lab_support/blob/main/fun_ml.R?raw=true")
```

```
i SHA-1 hash of file is "32a0bc8ced92c79756b56ddcdc9a06e639795da6"
```

```
tidymodels_conflictRules()

library(tidyverse)
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.5
v forcats   1.0.0     v stringr   1.5.1
v ggplot2   3.5.1     v tibble    3.2.1
v lubridate 1.9.4     v tidyr     1.3.1
v purrr     1.0.4
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to becom
```

```
library(tidymodels)
```

```
-- Attaching packages -------------------------------------- tidymodels 1.3.0 --
v broom        1.0.7     v rsample      1.2.1
v dials        1.4.0     v tune         1.3.0
v infer        1.0.7     v workflows    1.2.0
```

```
v modeldata    1.4.0     v workflowsets 1.1.0
v parsnip      1.3.0     v yardstick    1.3.2
v recipes      1.2.1
-- Conflicts ----------------------------------------- tidymodels_conflicts() --
x scales::discard() masks purrr::discard()
x dplyr::filter()   masks stats::filter()
x recipes::fixed()  masks stringr::fixed()
x dplyr::lag()      masks stats::lag()
x yardstick::spec() masks readr::spec()
x recipes::step()   masks stats::step()
```

```r
library(Matrix, exclude = c("expand", "pack", "unpack"))
library(magrittr, exclude = c("set_names", "extract"))
library(rsample)
library(parsnip)
library(recipes)
library(workflows)
library(stringr)

devtools::source_url("https://github.com/jjcurtin/lab_support/blob/main/fun_plots.R?raw=true"
```

```
i SHA-1 hash of file is "def6ce26ed7b2493931fde811adff9287ee8d874"
```

```r
devtools::source_url("https://github.com/jjcurtin/lab_support/blob/main/fun_eda.R?raw=true")
```

```
i SHA-1 hash of file is "c045eee2655a18dc85e715b78182f176327358a7"
```

```r
options(tibble.width = Inf, dplyr.print_max=Inf)
rerun_setting <- FALSE

cl <- parallel::makePSOCKcluster(parallel::detectCores(logical = FALSE))
doParallel::registerDoParallel(cl)

path_data <- "homework/data"
```

Part 1: Read in and split data

Since we do not have that much data, we will be splitting into a validation set for considering model configurations. However, you should still only look at the training portion of this split during EDA.

Read in `alcohol_tweets.csv`

```
data <- read_csv(here::here(path_data, "alcohol_tweets.csv"),
                     show_col_types = FALSE) |>
  glimpse()
```

```
Rows: 4,944
Columns: 3
$ id            <dbl> 4.717116e+17, 4.645540e+17, 4.680948e+17, 4.703539e+17, ~
$ user_drinking <chr> "no", "yes", "no", "yes", "no", "no", "no", "no", "no", ~
$ text          <chr> "\"@ShigDollaz: Shot Dollaz We Got Back Wit Ya! Fuck Yo ~
```

For this assignment it is helpful to see (and potentially copy) the `id` values from tweets, which we can't do easily if the number is in scientific notation. To change this in R, set the relevant option called `scipen`. You can do this by running the code below, which says that any integer with 20 decimal places or fewer, will be in raw integer form, not scientific notation. This can be useful with ID-type variables like we use here.

```
options(scipen = 20)
```

Glove embeddings

The file containing the GloVe embeddings is very large, so we will use `fread()`.

```
glove_embeddings <- data.table::fread(here::here(path_data, 'glove_twitter.csv')) |>
  as_tibble()
```

Validation splits

Use the provided splits across all model configurations you consider

```
set.seed(12345)

splits <- data |>
  validation_split(strata = "user_drinking")
```

```
Warning: `validation_split()` was deprecated in rsample 1.2.0.
i Please use `initial_validation_split()` instead.
```

```r
# Pull out indices of training data
training_ind <- splits |>
  unlist(recursive = FALSE)

training_ind <- training_ind$splits$in_id

data_train <- data |>
  slice(training_ind)
```

Part 1: Cleaning EDA

In this section, use the tidytext package to get a better sense of the data to guide model building in Part 2. Use data_train to identify what cleaning steps you may want to take. We will apply your identified cleaning steps to the full dataset and resplit again before building models.

Initial Cleaning

At a minimum, complete the following steps:

- Clean the tweets. Visual inspection of text data is really important. Are there any special characters or parsing errors that need to be handled in the text? For example, how will you handle NA tweets that were originally just images? This text is likely to have other characters that you want to consider for modeling the outcome. Take steps to process the text in a way that serves your modeling objective.
- Classing variables. Check if `id` and `user_drinking` variables are the correct class. What is the distribution of the outcome?
- Tokenization. Tokenize your text into both unigrams and bigrams. The help page for `unnest_tokens()` can help you understand your options for tokenization in tidytext.
- Stopwords. Load the stopwords that you plan to use in your workflow. Think about which stopwords you will use and why, and have those ready. Look at the tokenized data in order to consider how stopwords will impact modeling.

```r
library(tidytext)

clean_tweets <- function(text) {
  text <- ifelse(is.na(text), "", text)

  text |>
```

```
    str_to_lower() |>
    str_replace_all("http\\S+|www\\S+", "") |>
    str_replace_all("@\\w+", "") |>
    str_replace_all("#(\\w+)", "\\1") |>
    str_replace_all("^rt:", "") |>
    str_replace_all("[[:punct:]]", " ") |>
    str_replace_all("\\d+", "") |>
    str_squish()
}

data <- data |>
  mutate(clean_text = clean_tweets(text))

data |>
  select(text, clean_text) |>
  head(5)
```

```
# A tibble: 5 x 2
  text
  <chr>
1 "\"@ShigDollaz: Shot Dollaz We Got Back Wit Ya! Fuck Yo Leg Up Now You Will N~
2 "Chris is my DD while I drink by myself Wednesday afternoon and get WASTED ce~
3 "Only inviting fam to my grad party cause im tryna rack in that $$$$ to pay f~
4 "N/A"
5 "Straight from the airport.  When in Rome... er, Rochester! (@ Dinosaur Bar-B~
  clean_text
  <chr>
1 shot dollaz we got back wit ya fuck yo leg up now you will never run again ay~
2 chris is my dd while i drink by myself wednesday afternoon and get wasted cel~
3 only inviting fam to my grad party cause im tryna rack in that $$$$ to pay fo~
4 n a
5 straight from the airport when in rome er rochester dinosaur bar b que
```

```
unigrams <- data |>
  unnest_tokens(word, clean_text, token = "words") |>
  filter(word != "")

unigrams |>
  count(word, sort = TRUE) |>
  head(20)
```

```
# A tibble: 20 x 2
```

```
   word      n
   <chr> <int>
 1 a      2791
 2 n      1745
 3 i      1452
 4 the    1129
 5 to      809
 6 and     749
 7 my      716
 8 of      490
 9 party   484
10 s       484
11 you     460
12 up      457
13 drunk   421
14 in      419
15 at      415
16 is      399
17 m       372
18 it      355
19 beer    342
20 me      337
```

```r
bigrams <- data |>
  unnest_tokens(bigram, clean_text, token = "ngrams", n = 2) |>
  filter(!is.na(bigram))

bigrams |>
  count(bigram, sort = TRUE) |>
  head(20)
```

```
# A tibble: 20 x 2
   bigram          n
   <chr>       <int>
 1 n a          1722
 2 i m           367
 3 fucked up     211
 4 it s          113
 5 turn up       107
 6 in the        102
 7 don t          86
 8 drinking a     86
```

```
 9 at the        85
10 m at          82
11 to the        77
12 and i         75
13 on the        72
14 a beer        58
15 i can         56
16 of the        54
17 can t         53
18 i just        53
19 for the       51
20 a bottle      50
```

```
data("stop_words")

custom_stopwords <- tribble(
  ~word, ~lexicon,
  "rt", "custom",
  "amp", "custom",
  "https", "custom",
  "http", "custom"
)

all_stopwords <- bind_rows(stop_words, custom_stopwords)

unigrams_filtered <- unigrams |>
  anti_join(all_stopwords, by = "word")

unigrams_filtered |>
  count(word, sort = TRUE) |>
  head(20)
```

```
# A tibble: 20 x 2
   word        n
   <chr>   <int>
 1 party     484
 2 drunk     421
 3 beer      342
 4 shot      309
 5 bar       243
 6 club      238
 7 fucked    215
```

```
 8 wine        192
 9 drinking    146
10 bottle      118
11 night       112
12 time        105
13 tonight     105
14 ny          100
15 life         96
16 day          94
17 lol          90
18 don          89
19 love         88
20 alcohol      79
```

Explore tokens

At a minimum, complete the following steps (for both unigram and bigram tokens):

- Display the total number of tokens used across all tweets
- Display the total number of unique tokens used across all tweets
- Plot the frequency distribution of the 1000 most common tokens
- Review the top 1000 tokens

```r
library(ggplot2)

total_unigrams <- nrow(unigrams)
unique_unigrams <- unigrams |>
  distinct(word) |>
  nrow()

cat("Total unigram tokens:", total_unigrams, "\n")
```

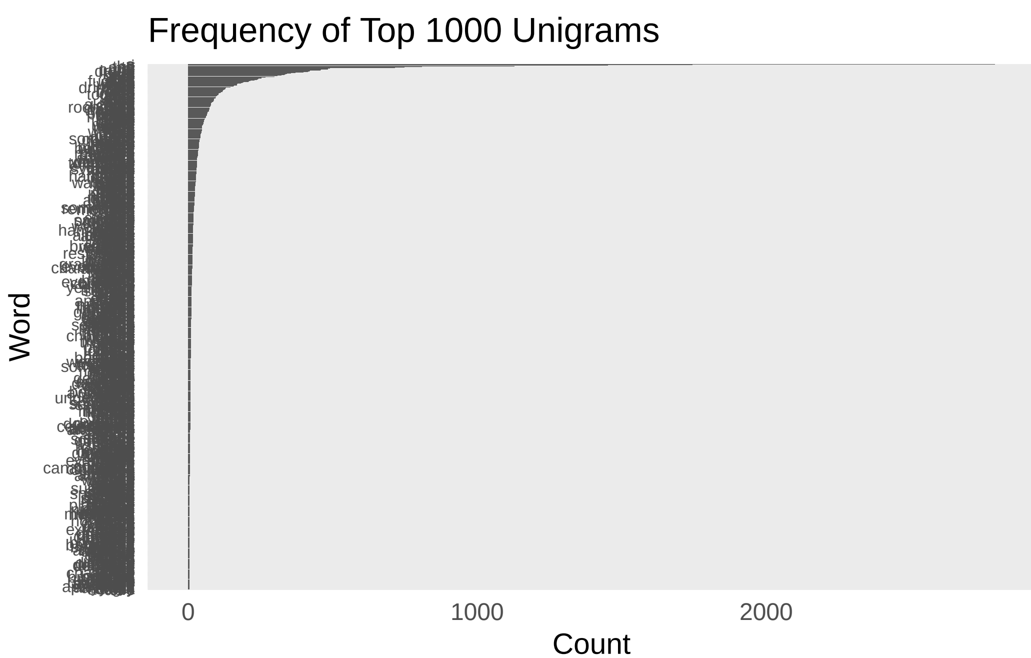
```
Total unigram tokens: 44255
```

```r
cat("Unique unigram tokens:", unique_unigrams, "\n")
```

```
Unique unigram tokens: 5972
```

```r
top_unigrams <- unigrams |>
  count(word, sort = TRUE) |>
  head(1000)

ggplot(top_unigrams, aes(x = reorder(word, n), y = n)) +
  geom_col() +
  coord_flip() +
  labs(x = "Word", y = "Count", title = "Frequency of Top 1000 Unigrams") +
  theme_minimal() +
  theme(axis.text.y = element_text(size = 6))
```




```r
print("Top 20 unigrams:")
```

```
[1] "Top 20 unigrams:"
```

```r
top_unigrams |>  head(20)
```

```
# A tibble: 20 x 2
   word      n
   <chr> <int>
 1 a      2791
 2 n      1745
```

```
 3 i        1452
 4 the      1129
 5 to        809
 6 and       749
 7 my        716
 8 of        490
 9 party     484
10 s         484
11 you       460
12 up        457
13 drunk     421
14 in        419
15 at        415
16 is        399
17 m         372
18 it        355
19 beer      342
20 me        337
```

```r
total_bigrams <- nrow(bigrams)
unique_bigrams <- bigrams |>
  distinct(bigram) |>
  nrow()

cat("Total bigram tokens:", total_bigrams, "\n")
```

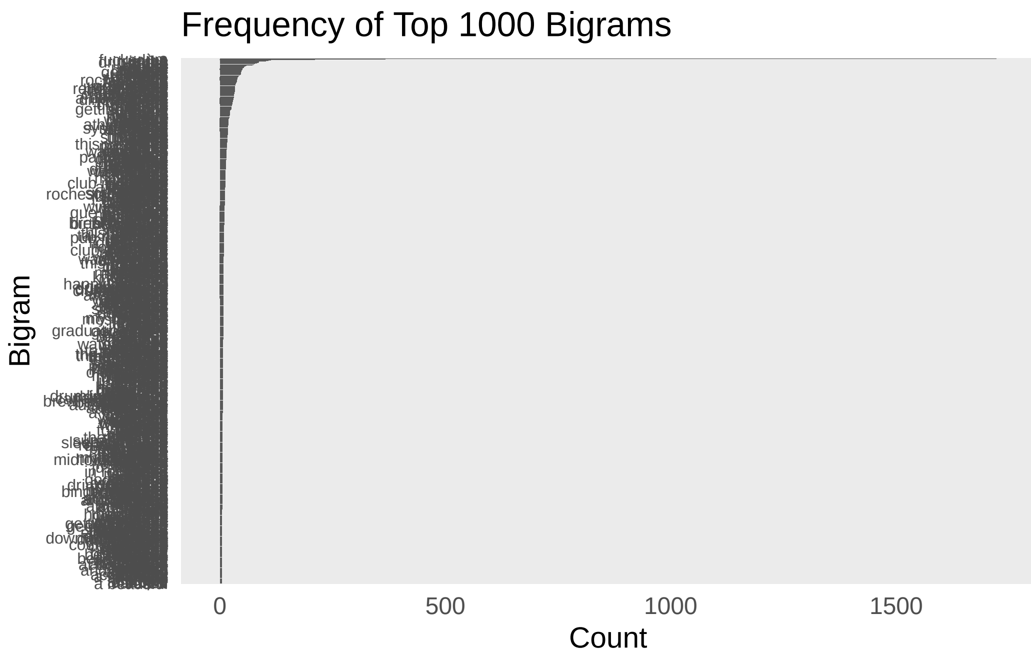
```
Total bigram tokens: 39312
```

```r
cat("Unique bigram tokens:", unique_bigrams, "\n")
```

```
Unique bigram tokens: 22801
```

```r
top_bigrams <- bigrams |>
  count(bigram, sort = TRUE) |>
  head(1000)

ggplot(top_bigrams, aes(x = reorder(bigram, n), y = n)) +
  geom_col() +
  coord_flip() +
  labs(x = "Bigram", y = "Count", title = "Frequency of Top 1000 Bigrams") +
  theme_minimal() +
  theme(axis.text.y = element_text(size = 6))
```




```r
print("Top 20 bigrams:")
```

```
[1] "Top 20 bigrams:"
```

```r
top_bigrams |>  head(20)
```

```
# A tibble: 20 x 2
   bigram          n
   <chr>       <int>
 1 n a          1722
 2 i m           367
 3 fucked up     211
 4 it s          113
 5 turn up       107
 6 in the        102
 7 don t          86
 8 drinking a     86
 9 at the         85
10 m at           82
11 to the         77
12 and i          75
13 on the         72
```

```
14 a beer        58
15 i can         56
16 of the        54
17 can t         53
18 i just        53
19 for the       51
20 a bottle      50
```

Final cleaning

Based on your EDA above, apply your desired cleaning steps to your full data set. Call this "cleaned_data".

```r
cleaned_data <- data |>
  mutate(user_drinking = factor(user_drinking, levels = c("no", "yes"))) |>
  select(id, user_drinking, text, clean_text)

glimpse(cleaned_data)
```

```
Rows: 4,944
Columns: 4
$ id            <dbl> 471711578795278336, 464553997081526272, 4680948221090365~
$ user_drinking <fct> no, yes, no, yes, no, no, no, no, no, no, no, no, yes, y~
$ text          <chr> "\"@ShigDollaz: Shot Dollaz We Got Back Wit Ya! Fuck Yo ~
$ clean_text    <chr> "shot dollaz we got back wit ya fuck yo leg up now you w~
```

Resplit

Run this code chunk to apply the same split as we did at the beginning to the fully cleaned data

```r
set.seed(12345)

splits <- cleaned_data |>
  validation_split(strata = "user_drinking")
```

Part 2: Model Building

Now you will consider multiple model configurations to predict `user_drinking` from the tweet text!

Each model you train must:

- Use the provided validation splits provided below
- Be a glmnet
- Use balanced accuracy as the metric
- Specify all tweet cleaning/feature engineering steps using `tidytext` recipes

At a minimum, fit at least 3 model configurations:

- A Unigram BoW model
- An n-gram BoW model (using either bigrams, combination of unigrams and bigrams, etc.)
- A model using pretrained Twitter embeddings (provided in homework files in web book, you can use `data.table::fread()` to open this file)

You may end up considering more than just three model configurations. Across these configurations, also consider:

- Impact of different cleaning steps on different models
- Number of features retained in your document term matrix
- The stopwords that are important for these data in particular
- Stemming

Let's do it!

Consider Configurations

Use as many chunks as you'd like below to consider model configurations. Save the resampled performance for each configuration you consider.

```r
#Practicing helper functions

evaluate_model <- function(wf, splits) {
  train_data <- splits$splits[[1]]$data[splits$splits[[1]]$in_id, ]

  val_indices <- setdiff(1:nrow(splits$splits[[1]]$data), splits$splits[[1]]$in_id)
  val_data <- splits$splits[[1]]$data[val_indices, ]

  wf_fit <- wf |> fit(train_data)

  class_preds <- wf_fit |> predict(val_data)
  prob_preds <- wf_fit |> predict(val_data, type = "prob")

  results <- bind_cols(
    class_preds,
    prob_preds,
    val_data |> select(user_drinking)
  )

  metrics <- yardstick::bal_accuracy(
    results,
    truth = user_drinking,
    estimate = .pred_class
  )

  return(metrics)
}
```

```r
# Unigram boW model
library(textrecipes)
library(rsample)
library(stopwords)

# Alternative approach using training() function
train_data <- training(splits$splits[[1]])

# Then create the recipe
unigram_recipe <- recipe(user_drinking ~ clean_text, data = train_data) |>
  step_tokenize(clean_text) |>
  step_stopwords(clean_text) |>
  step_tokenfilter(clean_text, min_times = 5) |>
  step_tokenfilter(clean_text, max_tokens = 100) |>
```

```r
  step_tfidf(clean_text)

glmnet_spec <- logistic_reg(penalty = 0.01, mixture = 1) |>
  set_engine("glmnet") |>
  set_mode("classification")

unigram_wf <- workflow() |>
  add_recipe(unigram_recipe) |>
  add_model(glmnet_spec)

unigram_metrics <- evaluate_model(unigram_wf, splits)
print("Unigram Model Performance:")
```

```
[1] "Unigram Model Performance:"
```

```r
unigram_metrics
```

```
# A tibble: 1 x 3
  .metric      .estimator .estimate
  <chr>        <chr>          <dbl>
1 bal_accuracy binary         0.644
```

```r
# Unigrams and Bigrams
train_data <- splits$splits[[1]]$data[splits$splits[[1]]$in_id, ]

ngram_recipe <- recipe(user_drinking ~ clean_text, data = train_data) |>
  step_tokenize(clean_text,
                engine = "tokenizers", token = "words") |>
  step_stopwords(clean_text) |>
  step_ngram(clean_text, num_tokens = 2, min_num_tokens = 1) |>
  step_tokenfilter(clean_text, min_times = 5) |>
  step_tokenfilter(clean_text, max_tokens = 100) |>
  step_tfidf(clean_text) |>
  step_normalize(all_predictors())

ngram_wf <- workflow() |>
  add_recipe(ngram_recipe) |>
  add_model(glmnet_spec)

ngram_metrics <- evaluate_model(ngram_wf, splits)
print("Ngram Model Performance:")
```

```
[1] "Ngram Model Performance:"
```

```r
ngram_metrics
```

```
# A tibble: 1 x 3
  .metric      .estimator .estimate
  <chr>        <chr>          <dbl>
1 bal_accuracy binary         0.645
```

```r
# Stem
stemmed_recipe <- recipe(user_drinking ~ clean_text, data = train_data) |>
  step_tokenize(clean_text,
                engine = "tokenizers", token = "words") |>
  step_stopwords(clean_text) |>
  step_stem(clean_text) |>
  step_tokenfilter(clean_text, min_times = 5) |>
  step_tokenfilter(clean_text, max_tokens = 1000) |>
  step_tfidf(clean_text) |>
  step_normalize(all_predictors())

stemmed_wf <- workflow() |>
  add_recipe(stemmed_recipe) |>
  add_model(glmnet_spec)

stem_metrics <- evaluate_model(stemmed_wf, splits)
```

```
Warning: max_tokens was set to 1000, but only 100 was available and selected.
```

```r
print("Stem Model Performance:")
```

```
[1] "Stem Model Performance:"
```

```r
stem_metrics
```

```
# A tibble: 1 x 3
  .metric      .estimator .estimate
  <chr>        <chr>          <dbl>
1 bal_accuracy binary         0.639
```

Part 3: Best Model

Since we did NOT hold out an independent test set and selected our model configuration based on cross validated performance, these next steps are subject to some degree of optimization bias!

Print the cross-validated performance of your best performing model

```
model_metrics <- tibble(
  Model = c("Unigram BoW", "N-gram", "Stemmed"),
  Balanced_Accuracy = c(
    unigram_metrics$.estimate,
    ngram_metrics$.estimate,
    stem_metrics$.estimate
  )
)

model_metrics |>
  arrange(desc(Balanced_Accuracy))
```

```
# A tibble: 3 x 2
  Model       Balanced_Accuracy
  <chr>                   <dbl>
1 N-gram                  0.645
2 Unigram BoW             0.644
3 Stemmed                 0.639
```

```
best_model_name <- model_metrics |>
  arrange(desc(Balanced_Accuracy)) |>
  slice(1) |>
  pull(Model)

cat("The best performing model is:", best_model_name, "\n")
```

```
The best performing model is: N-gram
```

```
best_wf <- if(best_model_name == "Unigram BoW") {
  unigram_wf
} else if(best_model_name == "N-gram") {
```

```
  ngram_wf
} else {
  stemmed_wf
}
```

Train your top performing model on the full data set

```
best_fit <- fit(ngram_wf, data = cleaned_data)

best_fit_model <- best_fit |> extract_fit_parsnip()

tidy(best_fit_model) |>
  arrange(desc(abs(estimate))) |>
  slice_head(n = 20)
```

```
Loaded glmnet 4.1-8
```

```
# A tibble: 20 x 3
   term                         estimate penalty
   <chr>                           <dbl>   <dbl>
 1 tfidf_clean_text_club         -0.448     0.01
 2 (Intercept)                   -0.436     0.01
 3 tfidf_clean_text_party        -0.412     0.01
 4 tfidf_clean_text_shot         -0.392     0.01
 5 tfidf_clean_text_drinking      0.362     0.01
 6 tfidf_clean_text_wine          0.326     0.01
 7 tfidf_clean_text_fucked       -0.318     0.01
 8 tfidf_clean_text_beer          0.252     0.01
 9 tfidf_clean_text_drunk         0.148     0.01
10 tfidf_clean_text_tequila       0.141     0.01
11 tfidf_clean_text_turn         -0.137     0.01
12 tfidf_clean_text_drink         0.131     0.01
13 tfidf_clean_text_get_drunk     0.112     0.01
14 tfidf_clean_text_water        -0.102     0.01
15 tfidf_clean_text_m             0.0908    0.01
16 tfidf_clean_text_alcohol       0.0864    0.01
17 tfidf_clean_text_people       -0.0833    0.01
18 tfidf_clean_text_pub           0.0787    0.01
19 tfidf_clean_text_b            -0.0689    0.01
20 tfidf_clean_text_n             0.0535    0.01
```

Plot variable importance scores of your top performing model

- Do these make sense to you? Why or why not?

  This shows the top 20 most important variables identified by the N-gram model, ranked by their importance score, which suggests they are strong predictors in the model's task.

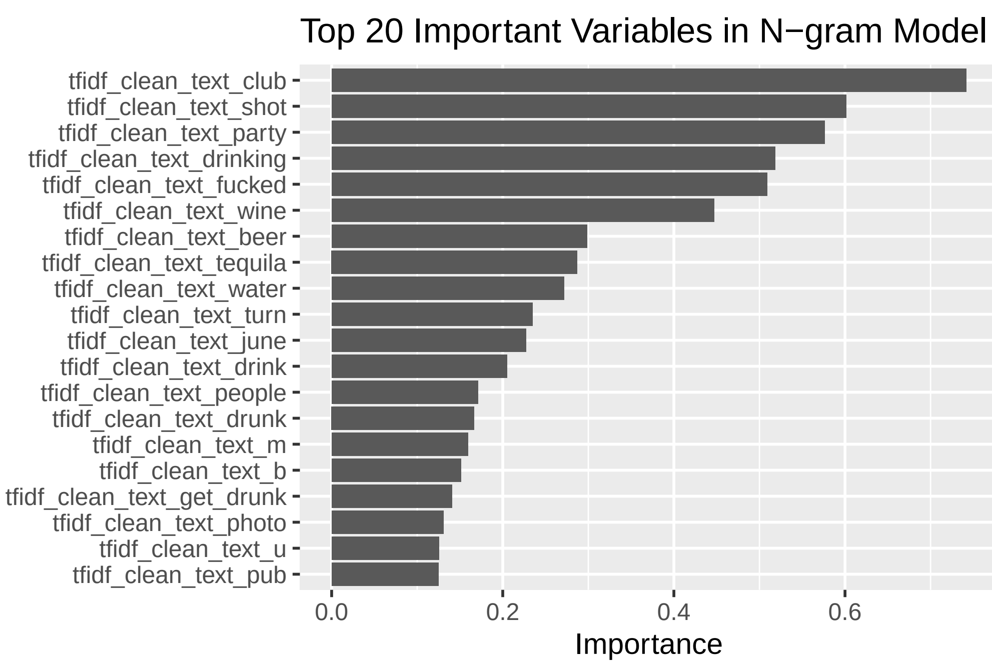
```
library(vip)
```

```
Attaching package: 'vip'


The following object is masked from 'package:utils':

    vi
```

```
best_fit |>
  extract_fit_parsnip() |>
  vip(num_features = 20) +
  labs(title = paste("Top 20 Important Variables in", best_model_name, "Model"))
```



```
if (FALSE) {
  coefs <- best_fit |>
    extract_fit_parsnip() |>
    tidy() |>
    filter(estimate != 0) |>
    arrange(desc(abs(estimate)))

  coefs |>
    head(20) |>
    mutate(term = forcats::fct_reorder(term, abs(estimate))) |>
    ggplot(aes(x = term, y = estimate, fill = estimate > 0)) +
    geom_col() +
    coord_flip() +
    labs(
      title = paste("Top 20 Important Variables in", best_model_name, "Model"),
      x = "Term",
      y = "Coefficient Estimate",
      fill = "Positive Effect"
    ) +
    theme_minimal()
}
```

Make a confusion matrix of your model's predictions

- What do you notice about the predictions that your model is making?

  Matrix shows that the N-gram model has some predictive power, but it's clearly biased towards predicting "no". The model struggles with identifying "yes" instances, indicating a need for improvement.

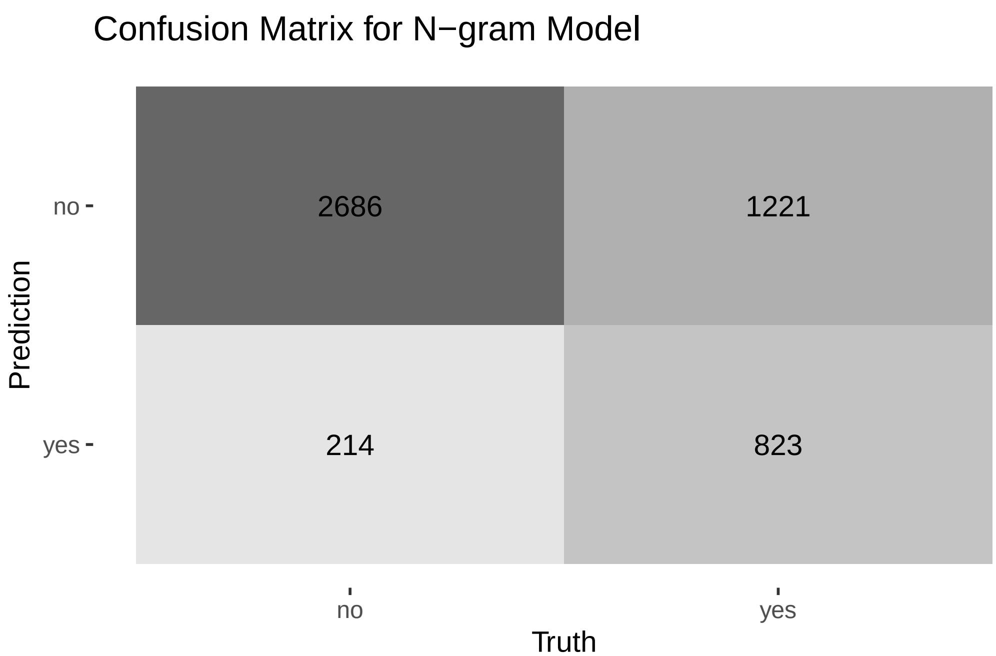
```
predictions <- best_fit |>
  predict(cleaned_data) |>
  bind_cols(
    best_fit |> predict(cleaned_data, type = "prob"),
    cleaned_data |> select(user_drinking)
  )

conf_mat <- predictions |>
  conf_mat(truth = user_drinking, estimate = .pred_class)

conf_mat
```

```
          Truth
Prediction   no  yes
       no  2686 1221
       yes  214  823
```

```r
conf_mat |>
  autoplot(type = "heatmap") +
  labs(title = paste("Confusion Matrix for", best_model_name, "Model"))
```



Knit this file and submit your knitted html. Make sure to leave yourself enough time to knit. Nice job completing this assignment - we are proud of you.