PSTAT 131 Final Project

Code

Tracy Sun

2023-12-15

Premise

r/AmITheAsshole is a popular subreddit where users can post about a situation and ask for feedback on who was in the wrong. Possible judgments include YTA/YWBTA (you're the asshole / you would be the asshole), NTA/YWNBTA (not the asshole / you wouldn't be the asshole), ESH (everyone sucks here), NAH (no assholes here), and INFO (more information needed). These scenarios range from mild to extreme and it is often hard to form a quick judgment on a situation from the title or the first few sentences, as some content creators who comment on the posts in this subreddit have pointed out. (https://www.youtube.com/watch?v=FPFJmupFg7k&t=1396s) Even with the entire post, there is always the possibility of the introduction of bias from the original poster leaving out key information or (even subconsciously) making themselves appear to be more innocent than they actually are, and as random strangers on the internet we will never understand the true extent of a particular story. However, with the power of machine learning, I will attempt to predict the judgments a particular post will receive and really get to the bottom of what makes someone the asshole.

Setup

Importing packages

Hide

```
library(textrecipes) # allows text processing steps in recipe
library(readxl) # read the data file (xlsx file)
# tidyverse packages
library(tidymodels)
library(tidyverse)
library(tidytext)
tidymodels_prefer()
library(themis)
# special matrix package
library(quanteda)
library(widyr)
library(furrr)
# text processing packages
library(textdata)
library(tokenizers)
# stopwords packages
library(stopwords)
library(SnowballC)
# pairwise association graphing
library(igraph)
library(ggraph)
library(irlba)
# SVM model packages
library(LiblineaR)
library(kernlab)
```

Reading in data

I gathered around 3000 posts and their related data using Python to access the Reddit API, then manually stitched together and cleaned the data in a separate file. I will load in the data here.

Hide

```
df <- read_xlsx('data_collection/aita_dataset_final.xlsx')
colnames(df)[1] = 'post_number'
head(df)
```

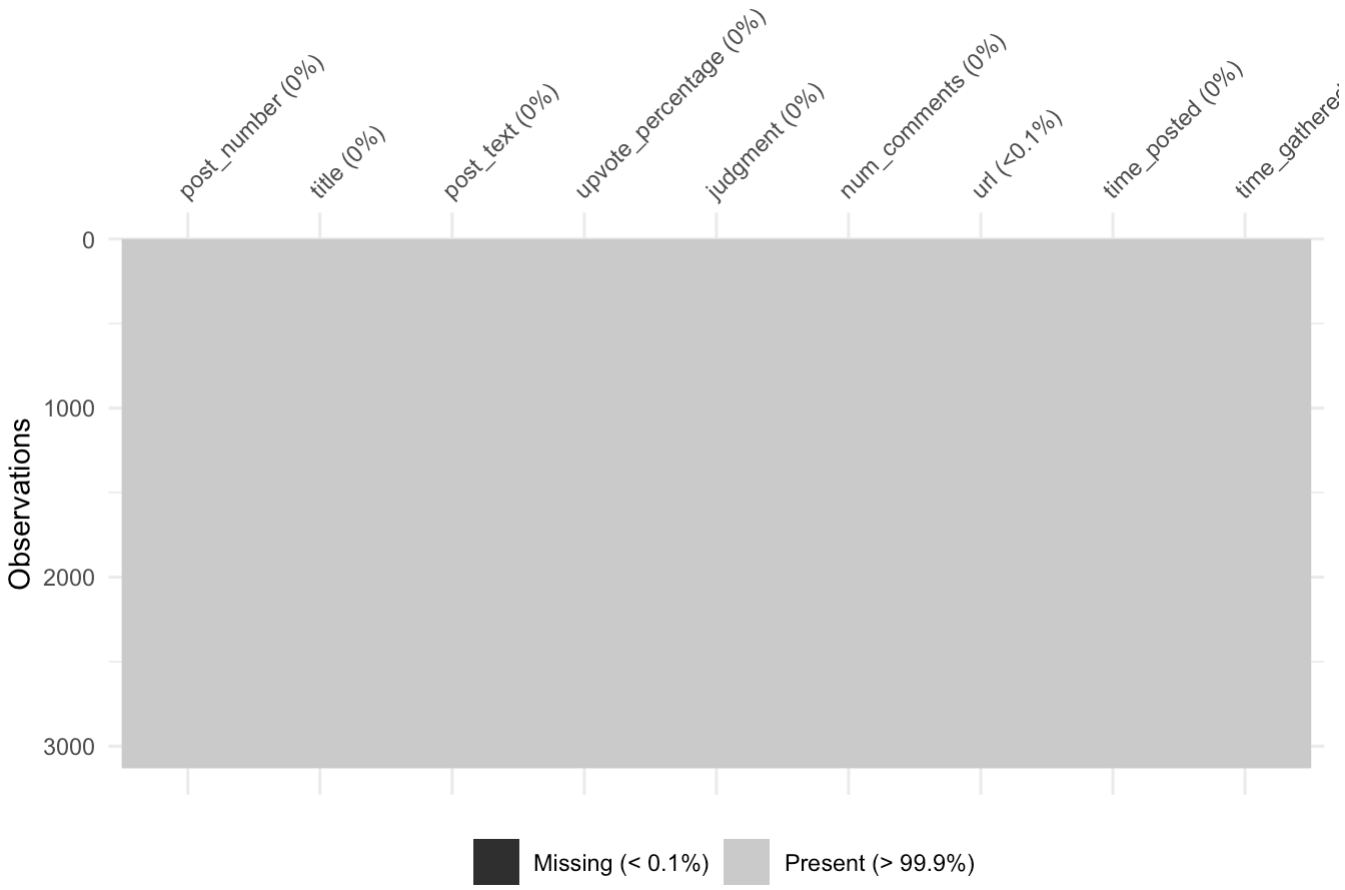
```
## # A tibble: 6 × 9
##   post_number title      post_text upvote_percentage judgment num_comments url
##         <dbl> <chr>      <chr>                <dbl> <chr>           <dbl> <chr>
## 1           0 "AITA for… "My brot…               92 NTA              5265 http…
## 2           1 "AITA For… "I'll tr…               98 NTA              2742 http…
## 3           2 "AITA for… "Edit: u…               98 NTA              3815 http…
## 4           3 "AITA for… "I am a …               92 NTA              3610 http…
## 5           4 "AITA for… "I am a …               94 NTA              3358 http…
## 6           5 "AITA for… "Note. M…               95 ESH              1991 http…
## # ℹ 2 more variables: time_posted <dttm>, time_gathered <dttm>
```

EDA

Breakdown of the judgments

Hide

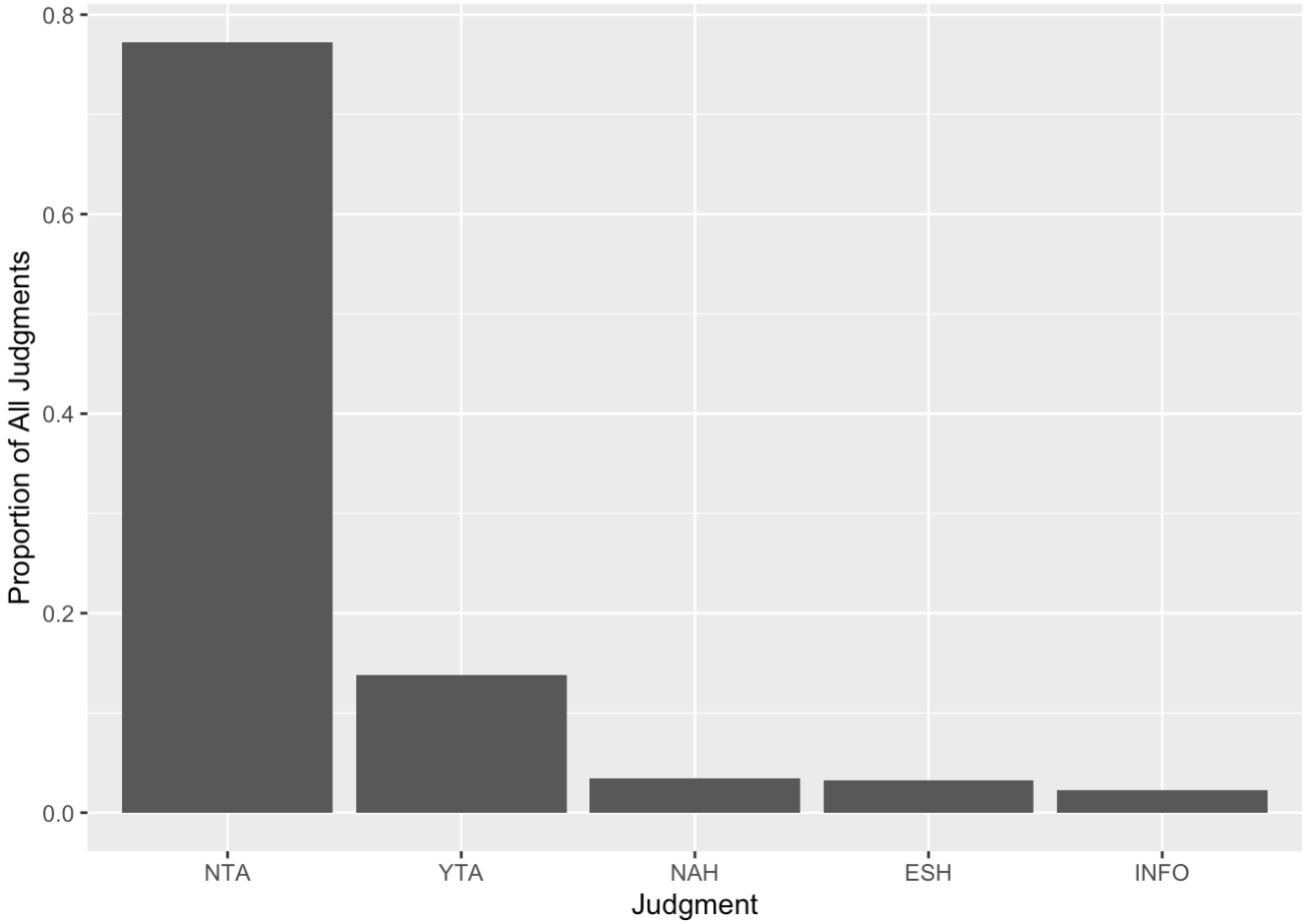
```
library(visdat)
vis_miss(df)
```



I made sure to get rid of missing values in the variables I will be using as predictors and the response variable (judgment).

Hide

```
df$judgment[df$judgment=="YWBTA"] <- "YTA"
df$judgment[df$judgment=="YWNBTA"] <- "NTA"
df$judgment <- as.factor(df$judgment)
df$judgment <- droplevels(df$judgment, exclude = c('YWBTA','YWNBTA'))
#summary(df['judgment'])

df %>%
  ggplot(aes
           (x = (forcats::fct_infreq(judgment)),
            y = after_stat(count)/sum(after_stat(count))
            )
         ) +
  geom_bar() +
  labs(x= "Judgment", y = "Proportion of All Judgments")
```



The NTA judgment is definitely overrepresented, composing nearly 80% of the data. It's great that the vast majority of people who post on the subreddit aren't actually in the wrong, but it does mean that I need to account for that in my model training.

Tokenization

The text data (posts and titles) is currently stored as sentences and paragraphs. In order to analyze the text data, we need to separate the words in it, a process known as tokenization.

Hide

```
post_text_vector = df %>% pull('post_text')
post_tokenized_vector = tokenize_words(post_text_vector)

title_text_vector = df %>% pull('title')
title_tokenized_vector = tokenize_words(title_text_vector)

post_tokenized_vector %>% head() %>% glimpse()
```

```
## List of 6
##  $ : chr [1:487] "my" "brother" "in" "law" ...
##  $ : chr [1:926] "i'll" "try" "to" "keep" ...
##  $ : chr [1:345] "edit" "update" "on" "profile" ...
##  $ : chr [1:403] "i" "am" "a" "high" ...
##  $ : chr [1:531] "i" "am" "a" "nurse" ...
##  $ : chr [1:303] "note" "my" "step" "daughter" ...
```

Hide

```
title_tokenized_vector %>% head() %>% glimpse()
```

```
## List of 6
##  $ : chr [1:27] "aita" "for" "telling" "my" ...
##  $ : chr [1:16] "aita" "for" "suing" "my" ...
##  $ : chr [1:15] "aita" "for" "bringing" "my" ...
##  $ : chr [1:13] "aita" "for" "pretending" "to" ...
##  $ : chr [1:18] "aita" "for" "bringing" "up" ...
##  $ : chr [1:6] "aita" "for" "making" "a" ...
```

Checking the most common words used

Hide

```
title_words_df <- df %>%
  unnest_tokens(word, title) %>%
  filter(!grepl('[0-9]', word))
title_words_df %>%
  count(word) %>%
  # group_by(post_number) %>%
  arrange(desc(n)) %>%
  slice(1:20)
```

```
## # A tibble: 20 × 2
##    word        n
##    <chr>   <int>
##  1 for      3297
##  2 my       3200
##  3 aita     2823
##  4 to       1588
##  5 a         941
##  6 not       722
##  7 i         615
##  8 the       606
##  9 and       464
## 10 telling   435
## 11 her       385
## 12 with      367
## 13 me        360
## 14 of        359
## 15 in        348
## 16 on        295
## 17 friend    294
## 18 wanting   281
## 19 after     272
## 20 at        272
```

Hide

```
post_words_df <- df %>%
  unnest_tokens(word, post_text) %>%
  filter(!grepl('[0-9]', word))
post_words_df %>%
  count(word) %>%
  # group_by(post_number) %>%
  arrange(desc(n)) %>%
  slice(1:20)
```

```
## # A tibble: 20 × 2
##    word      n
##    <chr> <int>
##  1 i     53617
##  2 and   46067
##  3 to    41445
##  4 the   34810
##  5 my    29652
##  6 a     26178
##  7 that  17451
##  8 she   16732
##  9 was   16513
## 10 me    16381
## 11 her   15725
## 12 of    15391
## 13 it    14707
## 14 for   14660
## 15 he    13733
## 16 in    12868
## 17 with  10702
## 18 but   10160
## 19 is     9277
## 20 have   8370
```

The most common words in the posts and titles are composed of words that can be considered “filler words”, such as pronouns and articles. While numbers that represent the ages of the poster and involved parties can be useful in predicting who is in the wrong, I find it simpler to remove them altogether due to inconsistencies in how they’re encoded in the posts and titles. We can remove these words by using a list of “stopwords” and remove the numbers as well. R has packages that can do this.

Taking out stopwords from the data

Hide

```
# premade stopwords lists mentioned in the smltar book (from least to most strict)
# snowball, smart, stopwords-iso

post_stopwords <- df %>%
                      unnest_tokens(word, post_text) %>%
                      filter(!grepl('[0-9]', word)) %>%
                      anti_join(get_stopwords(source = "stopwords-iso"))
post_stopwords
```

```
## # A tibble: 347,254 × 9
##    post_number title              upvote_percentage judgment num_comments url
##          <dbl> <chr>                          <dbl> <fct>           <dbl> <chr>
##  1           0 AITA for telling m…               92 NTA              5265 http…
##  2           0 AITA for telling m…               92 NTA              5265 http…
##  3           0 AITA for telling m…               92 NTA              5265 http…
##  4           0 AITA for telling m…               92 NTA              5265 http…
##  5           0 AITA for telling m…               92 NTA              5265 http…
##  6           0 AITA for telling m…               92 NTA              5265 http…
##  7           0 AITA for telling m…               92 NTA              5265 http…
##  8           0 AITA for telling m…               92 NTA              5265 http…
##  9           0 AITA for telling m…               92 NTA              5265 http…
## 10           0 AITA for telling m…               92 NTA              5265 http…
## # ℹ 347,244 more rows
## # ℹ 3 more variables: time_posted <dttm>, time_gathered <dttm>, word <chr>
```

Hide

```
title_stopwords <- df %>%
                       unnest_tokens(word, title) %>%
                       filter(!grepl('[0-9]', word)) %>%
                       anti_join(get_stopwords(source = "stopwords-iso"))
title_stopwords
```

```
## # A tibble: 16,520 × 9
##    post_number post_text           upvote_percentage judgment num_comments url
##          <dbl> <chr>                          <dbl> <fct>           <dbl> <chr>
##  1           0 "My brother in-law…               92 NTA              5265 http…
##  2           0 "My brother in-law…               92 NTA              5265 http…
##  3           0 "My brother in-law…               92 NTA              5265 http…
##  4           0 "My brother in-law…               92 NTA              5265 http…
##  5           0 "My brother in-law…               92 NTA              5265 http…
##  6           0 "My brother in-law…               92 NTA              5265 http…
##  7           0 "My brother in-law…               92 NTA              5265 http…
##  8           0 "My brother in-law…               92 NTA              5265 http…
##  9           0 "My brother in-law…               92 NTA              5265 http…
## 10           0 "My brother in-law…               92 NTA              5265 http…
## # ℹ 16,510 more rows
## # ℹ 3 more variables: time_posted <dttm>, time_gathered <dttm>, word <chr>
```

Hide

```
# proportion of words that were not removed by stopword removal compared to the ori
ginal
dim(title_stopwords %>% count(word))[1] / dim(title_words_df %>% count(word))[1]
```

```
## [1] 0.8603166
```

Hide

```
dim(post_stopwords %>% count(word))[1] / dim(post_words_df %>% count(word))[1]
```

```
## [1] 0.9519781
```

Removing stopwords and numbers reduced the total number of unique words for both posts and titles, but not by a large amount.

Stemming

Another way that we can further clean the text data is by using stemming. This collapses together related words like “tell” and “telling” into a single “stem” that they share (“tell”). However, the stems themselves might be different from the base word.

Hide

```
# most common stems for each post
post_stopwords_stemmed <- post_stopwords %>%
                                mutate(stem = wordStem(word)) %>%
                                count(post_number, stem, sort = TRUE)

post_stopwords_stemmed
```

```
## # A tibble: 238,593 × 3
##    post_number stem       n
##          <dbl> <chr>  <int>
##  1         518 kelli     30
##  2         108 cat       26
##  3         494 wife      26
##  4        1243 ticket    26
##  5        1898 hous      26
##  6         494 sister    25
##  7        2503 kid       25
##  8          66 pai       24
##  9         675 sister    24
## 10        1058 dog       24
## # ℹ 238,583 more rows
```

For this data, I won’t stem words, since the stems don’t make as much sense in terms of meaning and there are a lot of unique words in this data anyway.

Most common words (after removing stopwords)

Hide

```
post_stopwords %>%
  count(word) %>%
  arrange(desc(n))
```

```
## # A tibble: 20,141 × 2
##    word        n
##    <chr>   <int>
##  1 told     4976
##  2 time     3446
##  3 family   2674
##  4 mom      2439
##  5 i'm      2203
##  6 friends  1949
##  7 friend   1919
##  8 day      1894
##  9 feel     1884
## 10 house    1838
## # ℹ 20,131 more rows
```

Hide

```
title_stopwords %>%
  count(word) %>%
  arrange(desc(n))
```

```
## # A tibble: 3,369 × 2
##    word         n
##    <chr>    <int>
##  1 aita      2823
##  2 telling    435
##  3 friend     294
##  4 wibta      250
##  5 sister     171
##  6 family     165
##  7 wedding    151
##  8 mom        145
##  9 refusing   145
## 10 brother    144
## # ℹ 3,359 more rows
```

We can also look at the most common words within each post and title:

Hide

```
# most common words by post
post_stopwords %>%
  count(post_number, word) %>%
  group_by(post_number) %>%
  arrange(desc(n)) %>%
  slice(1:5)
```

```
## # A tibble: 15,633 × 3
## # Groups:   post_number [3,129]
##    post_number word          n
##          <dbl> <chr>     <int>
##  1           0 zoey          8
##  2           0 girls         6
##  3           0 wife          6
##  4           0 daughters     5
##  5           0 makeup        5
##  6           1 car          14
##  7           1 told          7
##  8           1 garage        4
##  9           1 guys          4
## 10           1 police        4
## # ℹ 15,623 more rows
```

Hide

```
# most common words by title
title_stopwords %>%
  count(post_number, word) %>%
  group_by(post_number) %>%
  arrange(desc(n)) %>%
  slice(1:5)
```

```
## # A tibble: 13,726 × 3
## # Groups:   post_number [3,130]
##    post_number word           n
##          <dbl> <chr>      <int>
##  1           0 aita           1
##  2           0 brother        1
##  3           0 daughter's     1
##  4           0 daughters      1
##  5           0 door           1
##  6           1 aita           1
##  7           1 girlfriend     1
##  8           1 impala         1
##  9           1 project        1
## 10           1 scrapyard      1
## # ℹ 13,716 more rows
```

We can visualize at the most common pairs of words that show up together. Code from this section of “Text Mining With R” (https://www.tidytextmining.com/nasa#word-co-ocurrences-and-correlations).

Pairwise word associations

Hide

```
post_word_pairs <- post_stopwords %>%
                pairwise_count(word, post_number, sort = TRUE, upper = FALSE)
post_word_pairs
```

```
## # A tibble: 5,621,016 × 3
##    item1 item2       n
##    <chr> <chr>   <dbl>
##  1 told  time     1215
##  2 told  aita     1059
##  3 time  aita      857
##  4 told  family    806
##  5 told  started   801
##  6 told  feel      800
##  7 told  day       787
##  8 time  feel      737
##  9 time  day       723
## 10 time  family    722
## # ℹ 5,621,006 more rows
```

Hide

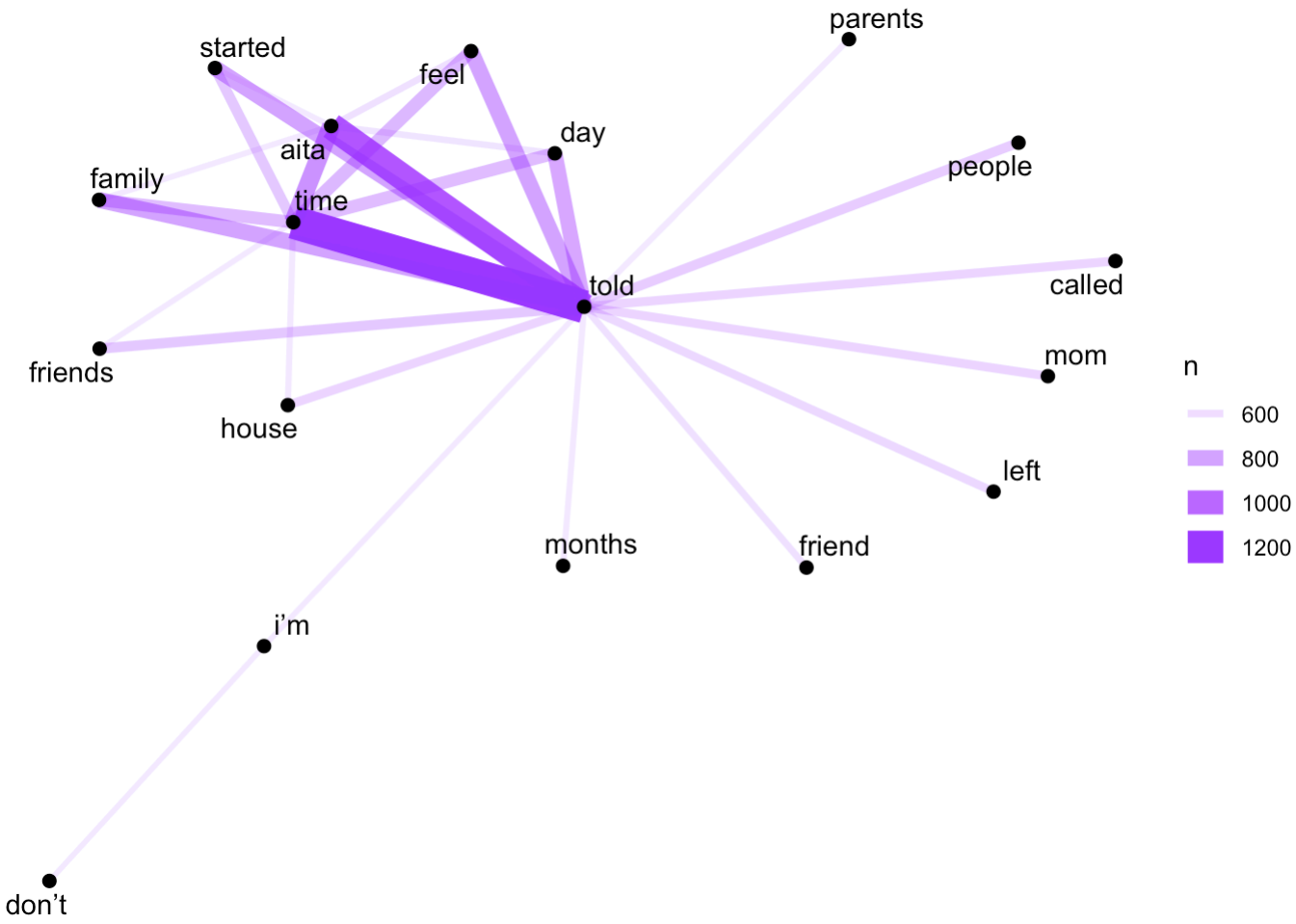
```
title_word_pairs <- title_stopwords %>%
                pairwise_count(word, post_number, sort = TRUE, upper = FALSE)
title_word_pairs
```

```
## # A tibble: 27,498 × 3
##    item1 item2        n
##    <chr> <chr>    <dbl>
##  1 aita  telling    414
##  2 aita  friend     259
##  3 aita  sister     161
##  4 aita  family     145
##  5 aita  refusing   139
##  6 aita  brother    138
##  7 aita  mom        134
##  8 aita  wedding    128
##  9 aita  husband    127
## 10 aita  wife       112
## # ℹ 27,488 more rows
```

Graphing the pairwise associations:

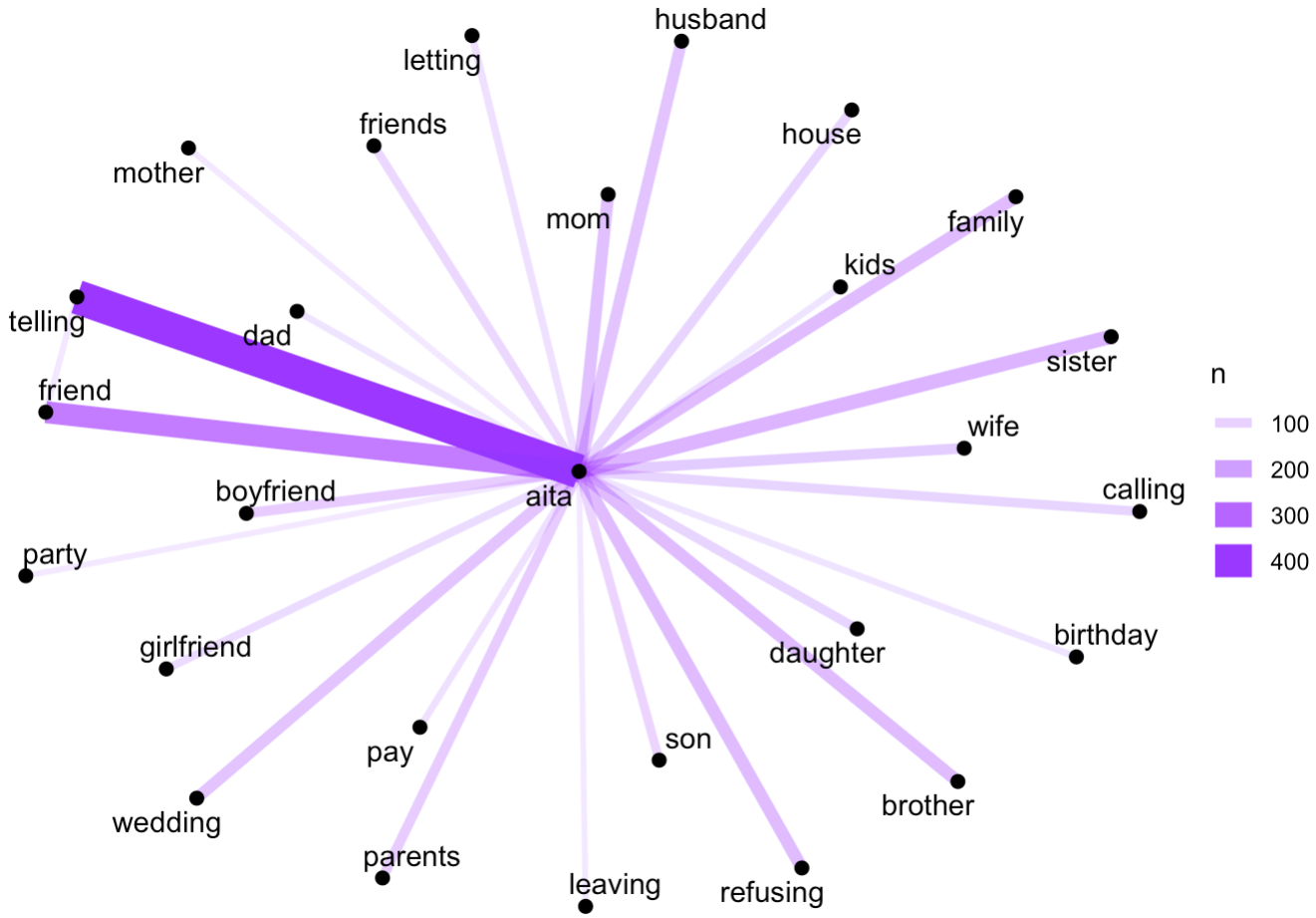
Hide

```
set.seed(27)
post_word_pairs %>%
  filter(n >= 550) %>%
  graph_from_data_frame() %>%
  ggraph(layout = "fr") +
  geom_edge_link(aes(edge_alpha = n, edge_width = n), edge_colour = "purple1") +
  geom_node_point(size = 2) +
  geom_node_text(aes(label = name), repel = TRUE,
                 point.padding = unit(0.2, "lines")) +
  theme_void()
```



Hide

```
title_word_pairs %>%
  filter(n >= 50) %>%
  graph_from_data_frame() %>%
  ggraph(layout = "fr") +
  geom_edge_link(aes(edge_alpha = n, edge_width = n), edge_colour = "purple1") +
  geom_node_point(size = 2) +
  geom_node_text(aes(label = name), repel = TRUE,
                 point.padding = unit(0.2, "lines")) +
  theme_void()
```



The words used are pretty centered around a select few pretty common ones. Notably, there are a lot of words pertaining to relationships (brother, daughter, girlfriend) and of course the name of the subreddit and the beginning of each title (AITA).

Text vectorization

One way to represent the words is to convert them into vectors based on the number of times they show up in the document and the words that they tend to occur with. This code was drawn from chapter 5.2 of SMLTAR (https://smltar.com/embeddings#understand-word-embeddings-by-finding-them-yourself).

Hide

```
post_text_reduced <- df %>%
  select(post_number, post_text) %>%
  unnest_tokens(word, post_text) %>%
  filter(!grepl('[0-9]', word)) %>%
  add_count(word) %>%
  # filter(n >= 50) %>%
  select(-n)

post_nested_words <- post_text_reduced %>%
  nest(words = c(word))

post_nested_words
```

```
## # A tibble: 3,129 × 2
##    post_number words
##          <dbl> <list>
##  1           0 <tibble [482 × 1]>
##  2           1 <tibble [918 × 1]>
##  3           2 <tibble [342 × 1]>
##  4           3 <tibble [403 × 1]>
##  5           4 <tibble [528 × 1]>
##  6           5 <tibble [300 × 1]>
##  7           6 <tibble [468 × 1]>
##  8           7 <tibble [301 × 1]>
##  9           8 <tibble [373 × 1]>
## 10           9 <tibble [288 × 1]>
## # ℹ 3,119 more rows
```

Hide

```
title_text_reduced <- df %>%
  select(post_number, title) %>%
  unnest_tokens(word, title) %>%
  filter(!grepl('[0-9]', word)) %>%
  add_count(word) %>%
  # filter(n >= 50) %>%
  select(-n)

title_nested_words <- title_text_reduced %>%
  nest(words = c(word))

title_nested_words
```

```
## # A tibble: 3,130 × 2
##    post_number words
##          <dbl> <list>
##  1           0 <tibble [27 × 1]>
##  2           1 <tibble [15 × 1]>
##  3           2 <tibble [15 × 1]>
##  4           3 <tibble [13 × 1]>
##  5           4 <tibble [18 × 1]>
##  6           5 <tibble [6 × 1]>
##  7           6 <tibble [15 × 1]>
##  8           7 <tibble [24 × 1]>
##  9           8 <tibble [10 × 1]>
## 10           9 <tibble [11 × 1]>
## # ℹ 3,120 more rows
```

Hide

```
slide_windows <- function(tbl, window_size) {
  skipgrams <- slider::slide(
    tbl,
    ~.x,
    .after = window_size - 1,
    .step = 1,
    .complete = TRUE
  )

  safe_mutate <- safely(mutate)

  out <- map2(skipgrams,
              1:length(skipgrams),
              ~ safe_mutate(.x, window_id = .y))

  out %>%
    transpose() %>%
    pluck("result") %>%
    compact() %>%
    bind_rows()
}
```

Hide

```
plan(multisession)

post_pmi <- post_nested_words %>%
  mutate(words = future_map(words, slide_windows, 4L)) %>%
  unnest(words) %>%
  unite(window_id, post_number, window_id) %>%
  pairwise_pmi(word, window_id)

post_pmi
save(post_pmi,file="post_pmi.Rda")
```

Hide

```
plan(multisession)

title_pmi <- title_nested_words %>%
  mutate(words = future_map(words, slide_windows, 4L)) %>%
  unnest(words) %>%
  unite(window_id, post_number, window_id) %>%
  pairwise_pmi(word, window_id)

title_pmi

save(title_pmi,file="title_pmi.Rda")
```

Hide

```
load(file = 'title_pmi.Rda')
load(file = 'post_pmi.Rda')
```

Hide

```
post_word_vectors <- post_pmi %>%
  widely_svd(
    item1, item2, pmi,
    nv = 100, maxit = 1000
  )

post_word_vectors
save(post_word_vectors,file="post_word_vectors.Rda")
```

Hide

```
title_word_vectors <- title_pmi %>%
  widely_svd(
    item1, item2, pmi,
    nv = 100, maxit = 1000
  )

title_word_vectors
save(title_word_vectors,file="title_word_vectors.Rda")
```

Hide

```
nearest_neighbors <- function(df, token) {
  df %>%
    widely(
      ~ {
        y <- .[rep(token, nrow(.)), ]
        res <- rowSums(. * y) /
          (sqrt(rowSums(. ^ 2)) * sqrt(sum(.[token, ] ^ 2)))

        matrix(res, ncol = 1, dimnames = list(x = names(res)))
      },
      sort = TRUE
    )(item1, dimension, value) %>%
    select(-item2)
}
```

Hide

```
load(file = 'post_word_vectors.Rda')
post_word_vectors %>%
  nearest_neighbors("happy")
```

```
## # A tibble: 21,157 × 2
##    item1         value
##    <chr>         <dbl>
##  1 happy         1
##  2 worried       0.578
##  3 close         0.530
##  4 upset         0.525
##  5 concerned     0.513
##  6 uncomfortable 0.489
##  7 excited       0.480
##  8 surprised     0.476
##  9 annoyed       0.475
## 10 mad           0.470
## # ℹ 21,147 more rows
```

After looking at training custom word embeddings on this data, I feel like this dataset is too small for it to work accurately. For example, the closest related word to “happy” is “worried”, followed by some more words relating to emotions, but many of them are more similar to “sad” than “happy”. This is understandable because people generally aren’t expressing happiness on this subreddit.

Downloading pretrained word embedding

Hide

```
glove6b <- embedding_glove6b(dimensions = 100,
                             dir = "/Users/trac.k.y/Documents/pstat131/aita",
                             return_path = TRUE,
                             manual_download = TRUE)
glove6b
```

```
## # A tibble: 400,000 × 101
##    token      d1      d2      d3      d4      d5      d6      d7      d8      d9
##    <chr>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
##  1 "the" -0.0382 -0.245   0.728  -0.400   0.0832  0.0440 -0.391   0.334  -0.575
##  2 ","   -0.108   0.111   0.598  -0.544   0.674   0.107   0.0389  0.355   0.0635
##  3 "."   -0.340   0.209   0.463  -0.648  -0.384   0.0380  0.171   0.160   0.466
##  4 "of"  -0.153  -0.243   0.898   0.170   0.535   0.488  -0.588  -0.180  -1.36
##  5 "to"  -0.190   0.0500  0.191  -0.0492 -0.0897  0.210  -0.550   0.0984 -0.201
##  6 "and" -0.0720  0.231   0.0237 -0.506   0.339   0.196  -0.329   0.184  -0.181
##  7 "in"   0.0857 -0.222   0.166   0.134   0.382   0.354   0.0129  0.225  -0.438
##  8 "a"   -0.271   0.0440 -0.0203 -0.174   0.644   0.712   0.355   0.471  -0.296
##  9 "\""  -0.305  -0.236   0.176  -0.729  -0.283  -0.256   0.266   0.0253 -0.0748
## 10 "'s"   0.589  -0.202   0.735  -0.683  -0.197  -0.180  -0.392   0.342  -0.606
## # ℹ 399,990 more rows
## # ℹ 91 more variables: d10 <dbl>, d11 <dbl>, d12 <dbl>, d13 <dbl>, d14 <dbl>,
## #   d15 <dbl>, d16 <dbl>, d17 <dbl>, d18 <dbl>, d19 <dbl>, d20 <dbl>,
## #   d21 <dbl>, d22 <dbl>, d23 <dbl>, d24 <dbl>, d25 <dbl>, d26 <dbl>,
## #   d27 <dbl>, d28 <dbl>, d29 <dbl>, d30 <dbl>, d31 <dbl>, d32 <dbl>,
## #   d33 <dbl>, d34 <dbl>, d35 <dbl>, d36 <dbl>, d37 <dbl>, d38 <dbl>,
## #   d39 <dbl>, d40 <dbl>, d41 <dbl>, d42 <dbl>, d43 <dbl>, d44 <dbl>, …
```

Tidying the GloVe embeddings a bit

Hide

```
tidy_glove <- glove6b %>%
  pivot_longer(contains("d"),
               names_to = "dimension") %>%
  rename(item1 = token)

tidy_glove
```

```
## # A tibble: 40,000,000 × 3
##    item1 dimension   value
##    <chr> <chr>       <dbl>
##  1 the   d1        -0.0382
##  2 the   d2        -0.245
##  3 the   d3         0.728
##  4 the   d4        -0.400
##  5 the   d5         0.0832
##  6 the   d6         0.0440
##  7 the   d7        -0.391
##  8 the   d8         0.334
##  9 the   d9        -0.575
## 10 the   d10        0.0875
## # ℹ 39,999,990 more rows
```

Modifying nearest neighbors function to work for GloVe embeddings

Hide

```
nearest_neighbors_glove <- function(df, token) {
  df %>%
    widely(
      ~ {
        y <- .[rep(token, nrow(.)), ]
        res <- rowSums(. * y) /
          (sqrt(rowSums(. ^ 2)) * sqrt(sum(.[token, ] ^ 2)))
        matrix(res, ncol = 1, dimnames = list(x = names(res)))
      },
      sort = TRUE,
      maximum_size = NULL
    )(item1, dimension, value) %>%
    select(-item2)
}
```

Hide

```
tidy_glove %>%
  nearest_neighbors_glove("happy")
```

```
## # A tibble: 400,000 × 2
##    item1  value
##    <chr>  <dbl>
##  1 happy  1
##  2 'm     0.841
##  3 feel   0.813
##  4 're    0.805
##  5 i      0.794
##  6 'll    0.792
##  7 really 0.790
##  8 glad   0.783
##  9 good   0.782
## 10 we     0.781
## # ℹ 399,990 more rows
```

This looks a bit better than the custom word embeddings.

We can use the embeddings to turn the data into a matrix.

Hide

```
tidy_post_words <- df %>%
  select(post_number, post_text) %>%
  unnest_tokens(word, post_text) %>%
  filter(!grepl('[0-9]', word)) %>%
  anti_join(stop_words) %>%
  add_count(word) %>%
  filter(n >= 50) %>%
  select(-n)

word_matrix <- tidy_post_words %>%
  inner_join(by = "word",
             tidy_glove %>%
               distinct(item1) %>%
               rename(word = item1)) %>%
  count(post_number, word) %>%
  cast_sparse(post_number, word, n)

glove_matrix <- tidy_glove %>%
  inner_join(by = "item1",
             tidy_post_words %>%
               distinct(word) %>%
               rename(item1 = word)) %>%
  cast_sparse(item1, dimension, value)

post_doc_matrix <- word_matrix %*% glove_matrix

glimpse(post_doc_matrix)
```

```
## Formal class 'dgCMatrix' [package "Matrix"] with 6 slots
##   ..@ i       : int [1:312600] 0 1 2 3 4 5 6 7 8 9 ...
##   ..@ p       : int [1:101] 0 3126 6252 9378 12504 15630 18756 21882 25008 28134
...
##   ..@ Dim     : int [1:2] 3126 100
##   ..@ Dimnames:List of 2
##   .. ..$ : chr [1:3126] "0" "1" "2" "3" ...
##   .. ..$ : chr [1:100] "d1" "d2" "d3" "d4" ...
##   ..@ x       : num [1:312600] -6.01 1.26 -1.52 1.77 1.39 ...
##   ..@ factors : list()
```

Hide

```
tidy_title_words <- df %>%
  select(post_number, title) %>%
  unnest_tokens(word, title) %>%
  filter(!grepl('[0-9]', word)) %>%
  anti_join(stop_words) %>%
  add_count(word) %>%
  filter(n >= 50) %>%
  select(-n)

title_word_matrix <- tidy_title_words %>%
  inner_join(by = "word",
             tidy_glove %>%
               distinct(item1) %>%
               rename(word = item1)) %>%
  count(post_number, word) %>%
  cast_sparse(post_number, word, n)

title_glove_matrix <- tidy_glove %>%
  inner_join(by = "item1",
             tidy_title_words %>%
               distinct(word) %>%
               rename(item1 = word)) %>%
  cast_sparse(item1, dimension, value)

title_doc_matrix <- title_word_matrix %*% title_glove_matrix
glimpse(title_doc_matrix)
```

```
## Formal class 'dgCMatrix' [package "Matrix"] with 6 slots
##   ..@ i       : int [1:302500] 0 1 2 3 4 5 6 7 8 9 ...
##   ..@ p       : int [1:101] 0 3025 6050 9075 12100 15125 18150 21175 24200 27225
...
##   ..@ Dim     : int [1:2] 3025 100
##   ..@ Dimnames:List of 2
##   .. ..$ : chr [1:3025] "0" "1" "2" "3" ...
##   .. ..$ : chr [1:100] "d1" "d2" "d3" "d4" ...
##   ..@ x       : num [1:302500] -1.2129 0.3976 -0.0242 -0.0242 0.9901 ...
##   ..@ factors : list()
```

The original dataset had 3130 entries, some posts and titles got removed because they did not have any words in common with the GloVe embeddings.

Model training

To train the model, I will split the data into training and testing sets. Since this is a relatively small dataset, I will leave most of the data (80%) for training the models on. The testing set will be used to evaluate model performance on data that it has not been trained on or seen before. I will also be using cross-validation, which splits the training data into multiple parts and simulates the training-testing dynamic within the training data. This will help give an accurate preliminary estimate of how the models will do on the testing data.

Splitting training data

Hide

```
set.seed(123)
df_split <- initial_split(df, strata = judgment, prop = 0.8)
df_train <- training(df_split)
df_test <- testing(df_split)
df_train$judgment <- droplevels(df_train$judgment, exclude = c('YWBTA','YWNBTA'))
df_test$judgment <- droplevels(df_test$judgment, exclude = c('YWBTA','YWNBTA'))

df_folds <- vfold_cv(df_train, v = 5, strata = judgment)
```

Setting up recipe

The steps we took to tokenize and vectorize the text data can be done in a tidymodels recipe with the help of the `textrecipes` package.

Hide

```
word_embeddings_recipe <-
  recipe(judgment ~ title + post_text + upvote_percentage + num_comments,
                data = df_train) %>%
                step_upsample(judgment, over_ratio = 0.4, skip=TRUE) %>%
                step_tokenize(post_text) %>%
                step_stopwords(post_text, stopword_source='stopwords-iso') %>%
                step_word_embeddings(post_text, embeddings = glove6b) %>%
                step_tokenize(title) %>%
                step_stopwords(title, stopword_source='stopwords-iso') %>%
                step_word_embeddings(title, embeddings = glove6b) %>%
                step_normalize(all_predictors())

word_embeddings_recipe %>% prep() %>% bake(new_data=df_train)
```

```
## # A tibble: 2,503 × 203
##    upvote_percentage num_comments judgment wordembed_post_text_d1
##                <dbl>        <dbl> <fct>                     <dbl>
##  1             0.683        2.88  NTA                       2.59
##  2             1.02         1.27  NTA                      -0.893
##  3             0.683        1.82  NTA                       0.608
##  4             0.794        1.66  NTA                       2.10
##  5             0.849        0.794 ESH                       0.544
##  6             1.02         3.39  NTA                       0.783
##  7             0.683        2.80  ESH                       0.666
##  8             0.683        1.17  NTA                       1.48
##  9             0.573        1.34  NTA                       0.558
## 10             0.517        4.30  NTA                      -2.11
## # ℹ 2,493 more rows
## # ℹ 199 more variables: wordembed_post_text_d2 <dbl>,
## #   wordembed_post_text_d3 <dbl>, wordembed_post_text_d4 <dbl>,
## #   wordembed_post_text_d5 <dbl>, wordembed_post_text_d6 <dbl>,
## #   wordembed_post_text_d7 <dbl>, wordembed_post_text_d8 <dbl>,
## #   wordembed_post_text_d9 <dbl>, wordembed_post_text_d10 <dbl>,
## #   wordembed_post_text_d11 <dbl>, wordembed_post_text_d12 <dbl>, …
```

Model fitting

I will fit a k-nearest neighbors, support vector machines based on a polynomial and a radial basis function, and a random forest model to this data. The SMLTAR book mentioned that while they are not as widely applicable as models such as aggregated tree models, SVMs have some properties that actually make them quite suited to doing classification on text data (briefly explained in this conference paper the book cited (https://link.springer.com/chapter/10.1007/BFb0026683)).

KNN model

Hide

```
knn_class <- nearest_neighbor(neighbors = tune()) %>%
  set_engine("kknn") %>%
  set_mode("classification")

knn_grid <- grid_regular(dials::neighbors(range = c(1, 1200)), levels = 20)

knn_wf <- workflow() %>%
  add_recipe(word_embeddings_recipe) %>%
  add_model(knn_class)

knn_res <- tune_grid(
  knn_wf,
  resamples = df_folds,
  grid = knn_grid
)
```

SVM polynomial

Hide

```
svm_polynom <- svm_poly(degree = tune(),
                        cost = tune(),
                        scale_factor = tune()) %>%
  set_mode("classification") %>%
  set_engine("kernlab")

svm_poly_grid <- grid_regular(dials::degree(range = c(1, 10)),
                              cost(),
                              scale_factor(),
                              levels = 5)

svm_poly_wf <- workflow() %>%
  add_recipe(word_embeddings_recipe) %>%
  add_model(svm_polynom)
```

Hide

```
svm_poly_res <- tune_grid(
  svm_poly_wf,
  resamples = df_folds,
  grid = svm_poly_grid,
  control = control_grid(verbose = TRUE)
)

save(svm_poly_res,file="svm_poly_res.Rda")
```

SVM radial basis function

Hide

```
svm_rbf_model <- svm_rbf(cost = tune(), rbf_sigma = tune()) %>%
  set_mode("classification") %>%
  set_engine("kernlab")

svm_rbf_wf <- workflow() %>%
  add_recipe(word_embeddings_recipe) %>%
  add_model(svm_rbf_model)

svm_rbf_grid <- grid_regular(cost(),
                             rbf_sigma(),
                             levels = 4)
```

Hide

```
svm_rbf_res <- tune_grid(
  svm_rbf_wf,
  resamples = df_folds,
  grid = svm_rbf_grid,
  control = control_grid(verbose = TRUE)
)

save(svm_rbf_res, file = 'svm_rbf_res.Rda')
```

Random forest

Hide

```
rf_model <- rand_forest(mtry = tune(),
                    trees = tune(),
                    min_n = tune()) %>%
  set_engine("ranger", importance = 'impurity') %>%
  set_mode("classification")

rf_grid <- grid_regular(mtry(range = c(1, 100)),
                        trees(range = c(1, 50)),
                        min_n(range = c(1, 20)),
                        levels = 10)

rf_wf <- workflow() %>%
  add_recipe(word_embeddings_recipe) %>%
  add_model(rf_model)
```

Hide

```
rf_res <- tune_grid(
  rf_wf,
  resamples = df_folds,
  grid = rf_grid
)

save(rf_res,file="rf_res.Rda")
```

Evaluation of performance on cross-validation folds

KNN model

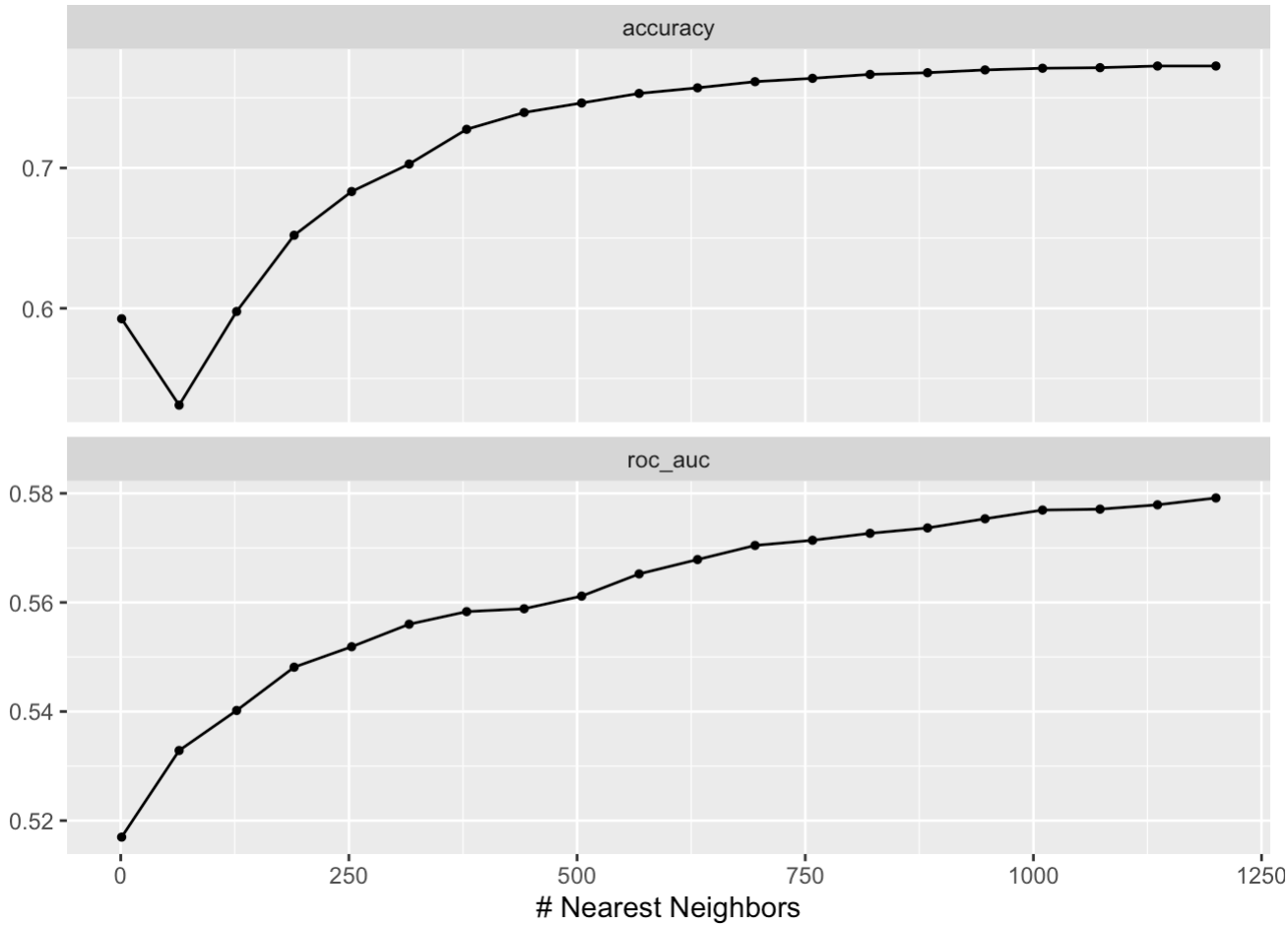
Hide

```
knn_metrics <- collect_metrics(knn_res)
knn_metrics %>%
  filter(.metric == "roc_auc") %>%
  select(.metric, mean, std_err, neighbors)
```

```
## # A tibble: 20 × 4
##    .metric  mean std_err neighbors
##    <chr>   <dbl>   <dbl>     <int>
##  1 roc_auc 0.517 0.00687         1
##  2 roc_auc 0.533 0.0146         64
##  3 roc_auc 0.540 0.0150        127
##  4 roc_auc 0.548 0.0146        190
##  5 roc_auc 0.552 0.0151        253
##  6 roc_auc 0.556 0.0157        316
##  7 roc_auc 0.558 0.0161        379
##  8 roc_auc 0.559 0.0160        442
##  9 roc_auc 0.561 0.0157        505
## 10 roc_auc 0.565 0.0159        568
## 11 roc_auc 0.568 0.0155        632
## 12 roc_auc 0.570 0.0156        695
## 13 roc_auc 0.571 0.0155        758
## 14 roc_auc 0.573 0.0152        821
## 15 roc_auc 0.574 0.0149        884
## 16 roc_auc 0.575 0.0151        947
## 17 roc_auc 0.577 0.0152       1010
## 18 roc_auc 0.577 0.0151       1073
## 19 roc_auc 0.578 0.0152       1136
## 20 roc_auc 0.579 0.0152       1200
```

Hide

```
autoplot(knn_res)
```



Hide

```
select_best(knn_res, metric = "roc_auc", neighbors)
```

```
## # A tibble: 1 × 2
##   neighbors .config
##       <int> <chr>
## 1      1200 Preprocessor1_Model20
```

Hide

```
select_best(knn_res, metric = "accuracy", neighbors)
```

```
## # A tibble: 1 × 2
##   neighbors .config
##       <int> <chr>
## 1      1136 Preprocessor1_Model19
```

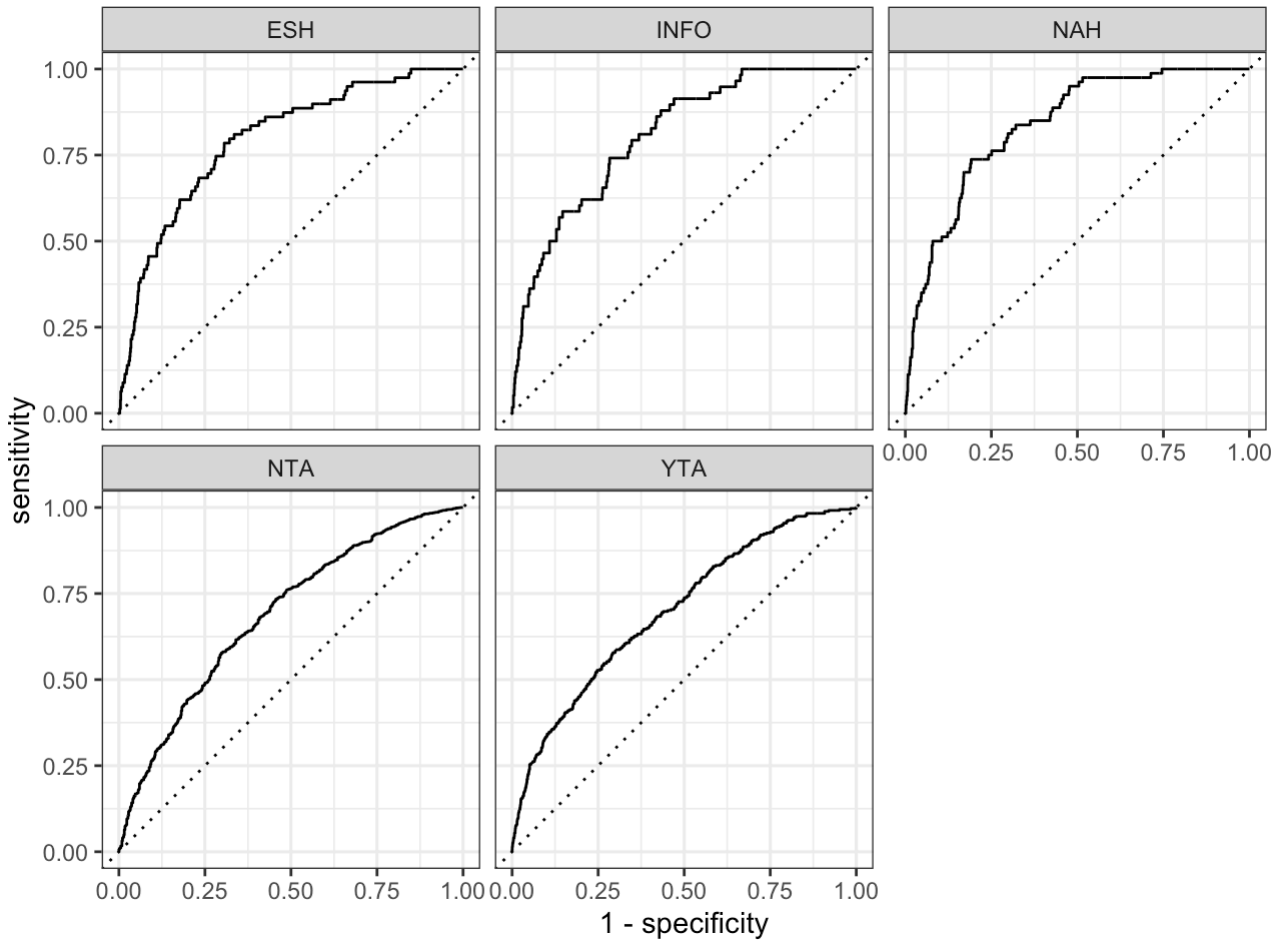
Hide

```
knn_roc_auc <- knn_metrics %>%
  filter(.metric == "roc_auc")
knn_roc_auc %>%
  filter(mean == max(knn_roc_auc$mean)) %>%
  select(.metric, neighbors, mean, std_err)
```

```
## # A tibble: 1 × 4
##   .metric neighbors  mean std_err
##   <chr>       <int> <dbl>   <dbl>
## 1 roc_auc      1200 0.579  0.0152
```

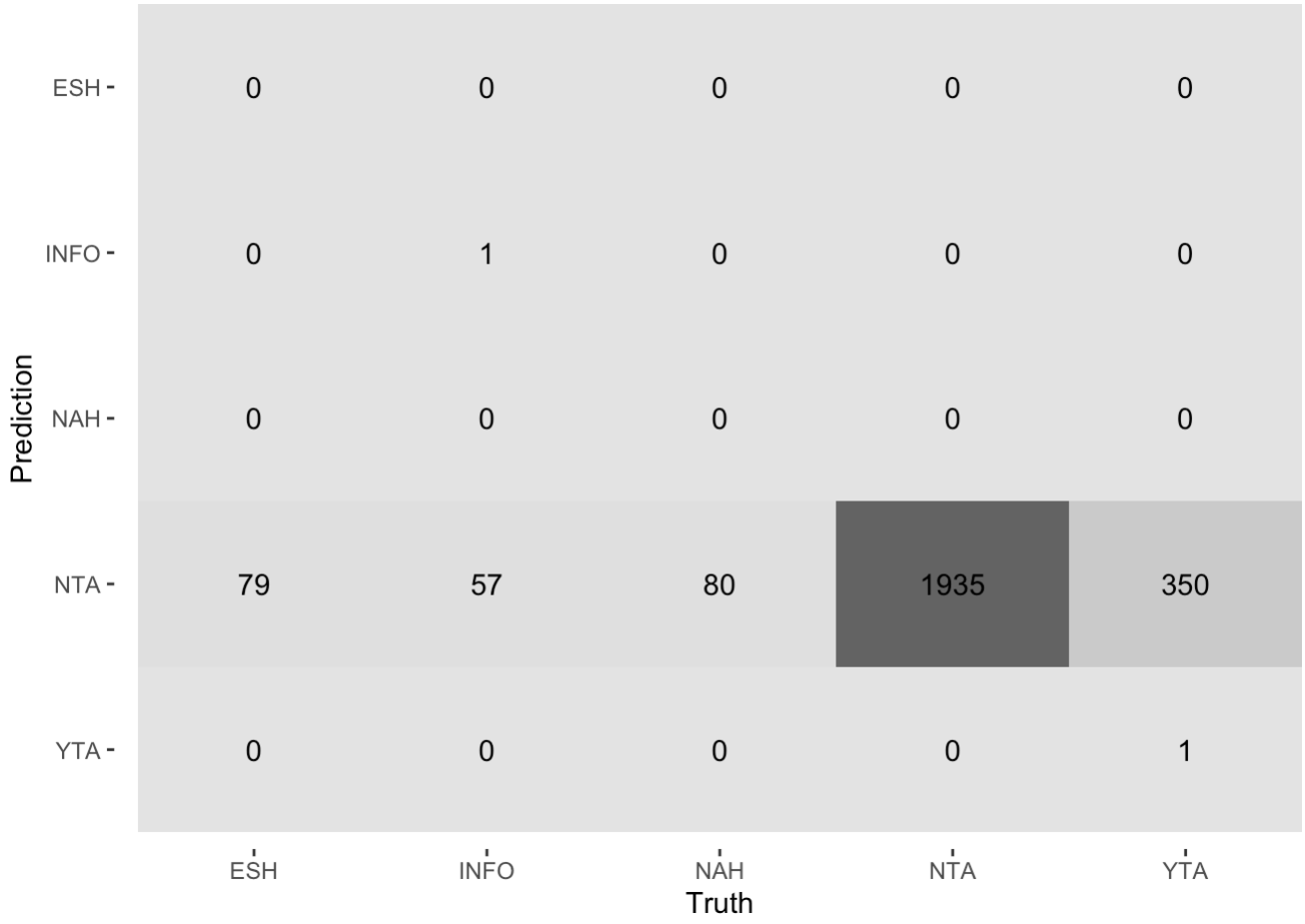
Hide

```
knn_final <- finalize_workflow(knn_wf,
                               select_best(knn_res, metric = "roc_auc", neighbors))
%>%
  fit(df_train)

knn_final_augmented <- augment(knn_final, new_data=df_train)

knn_final_augmented %>%
roc_curve(judgment, .pred_ESH:.pred_YTA) %>%
  autoplot()
```



Hide

```
autoplot(conf_mat(knn_final_augmented,
                  truth = judgment,
                  estimate = .pred_class),
         type = 'heatmap')
```

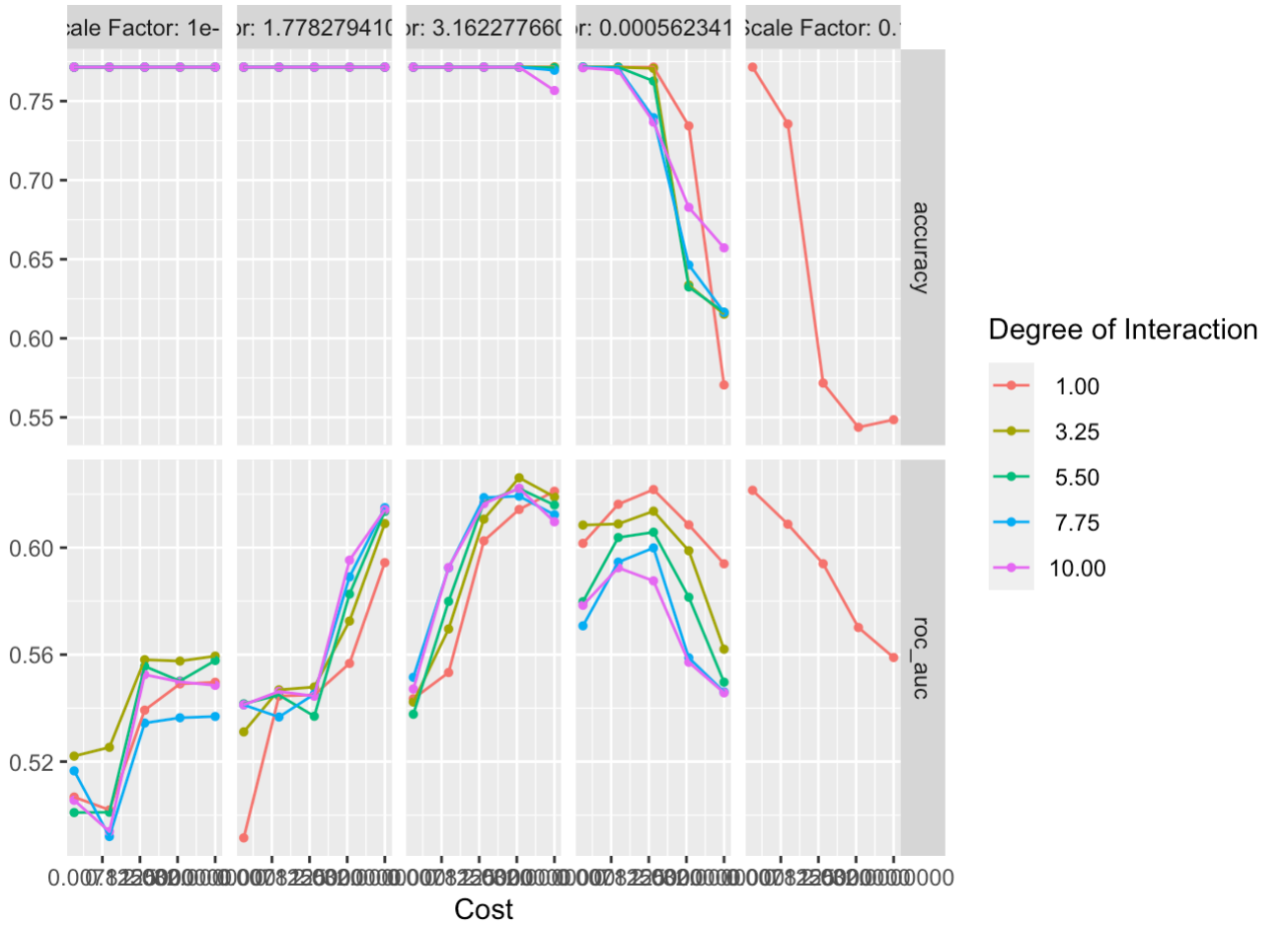


The KNN model did best with a higher number of neighbors. It's clear that the KNN model heavily overpredicts the NTA class even after some upsampling was done on the data. Since NTA is the most common class in the data, this is probably the primary cause of its ROC AUC being higher.

SVM polynomial

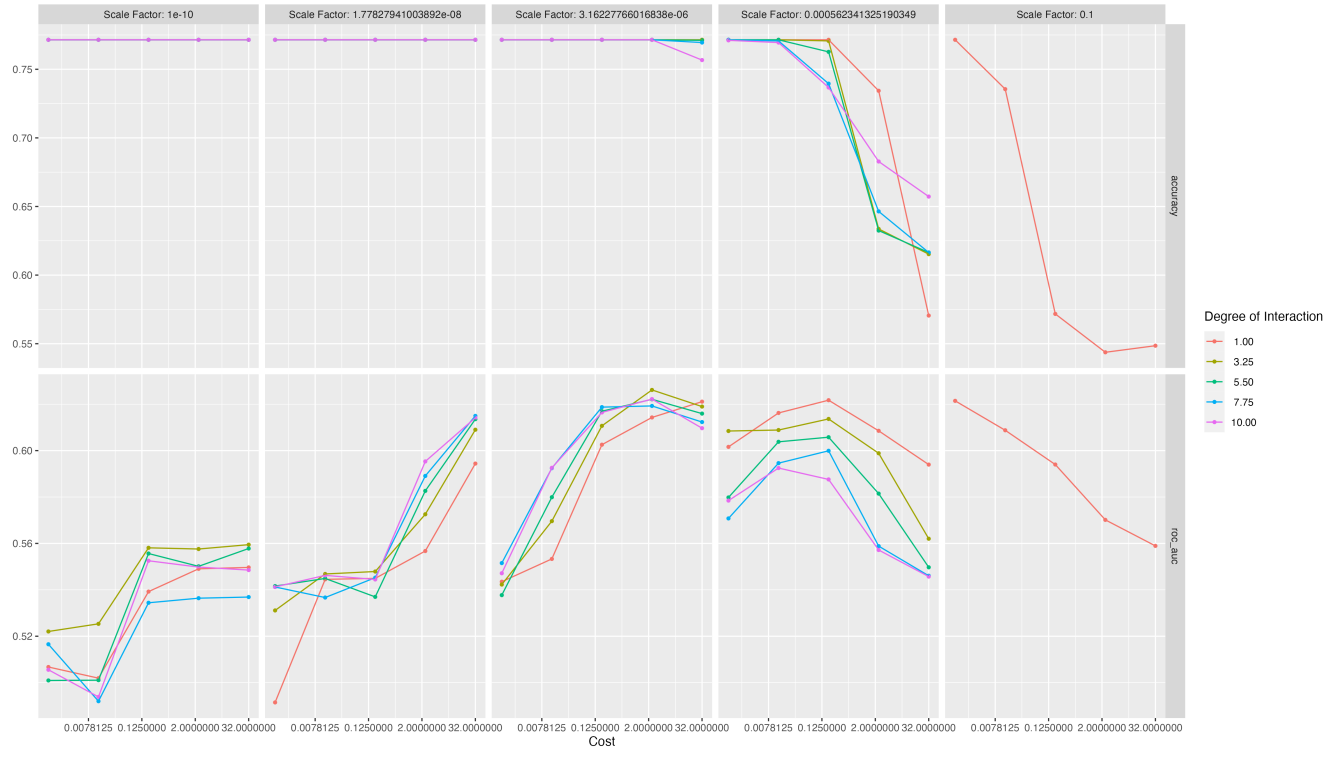
Hide

```
load(file = 'svm_poly_res.Rda')
autoplot(svm_poly_res)
```



Hide

```
ggsave('svm_poly_res.png', width=16, height=9)
```



It looks like a cost and scale factor in the middle is best, along with a polynomial border that is around cubic.

Hide

```
select_best(svm_poly_res, metric = "roc_auc", degree, cost, scale_factor)
```

```
## # A tibble: 1 × 4
##    cost degree scale_factor .config
##   <dbl>  <dbl>        <dbl> <chr>
## 1  2.38   3.25   0.00000316 Preprocessor1_Model067
```

Hide

```
select_best(svm_poly_res, metric = "accuracy", degree, cost, scale_factor)
```

```
## # A tibble: 1 × 4
##       cost degree scale_factor .config
##      <dbl>  <dbl>        <dbl> <chr>
## 1 0.000977      1 0.0000000001 Preprocessor1_Model001
```

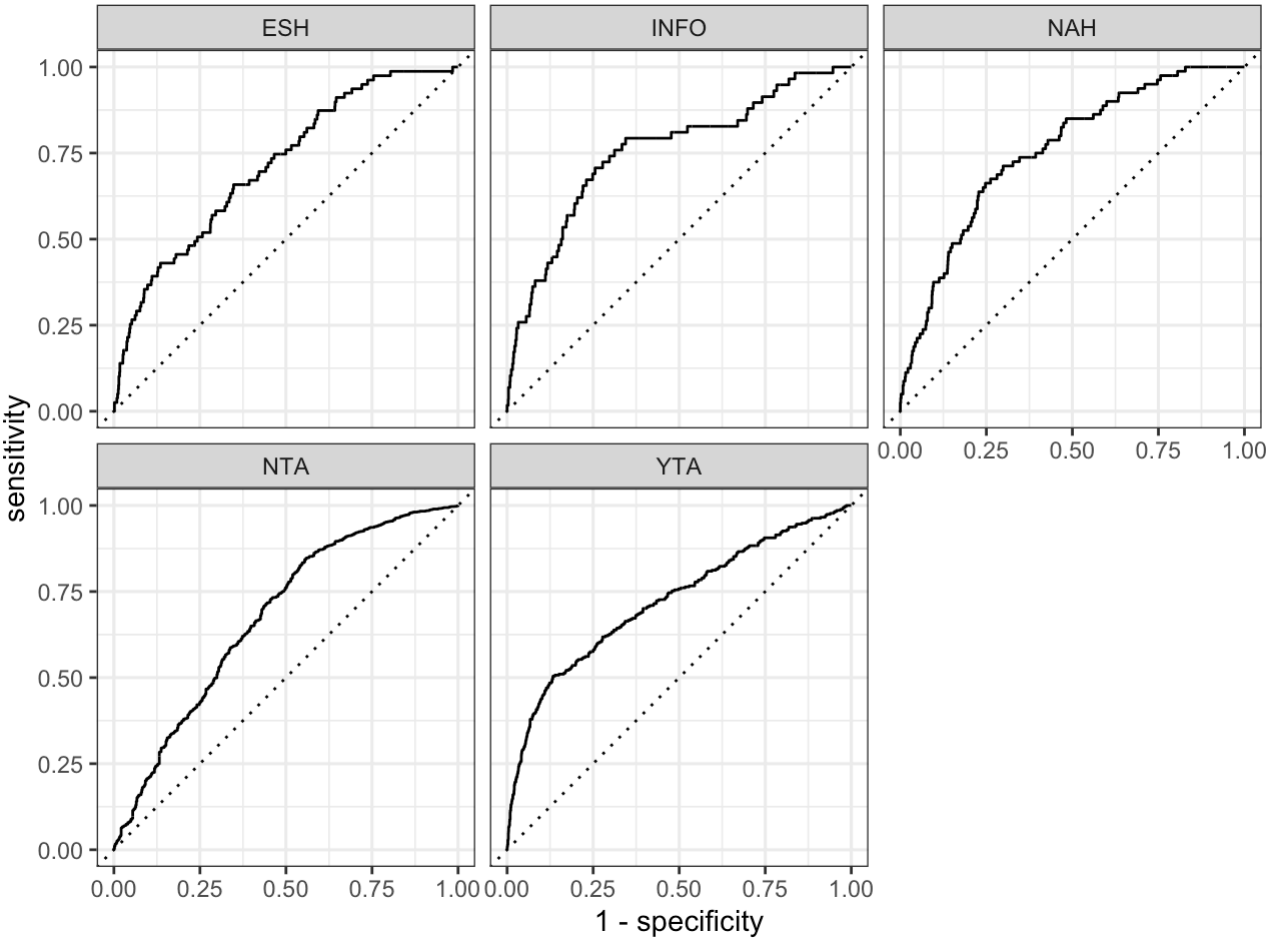
Hide

```
svm_poly_metrics <- collect_metrics(svm_poly_res)
svm_poly_roc_auc <- svm_poly_metrics %>%
  filter(.metric == "roc_auc")
svm_poly_roc_auc %>%
  filter(mean == max(svm_poly_roc_auc$mean)) %>%
  select(.metric, degree, cost, scale_factor, mean, std_err)
```

```
## # A tibble: 1 × 6
##   .metric degree  cost scale_factor  mean std_err
##   <chr>    <dbl> <dbl>        <dbl> <dbl>   <dbl>
## 1 roc_auc   3.25  2.38   0.00000316 0.626 0.00927
```

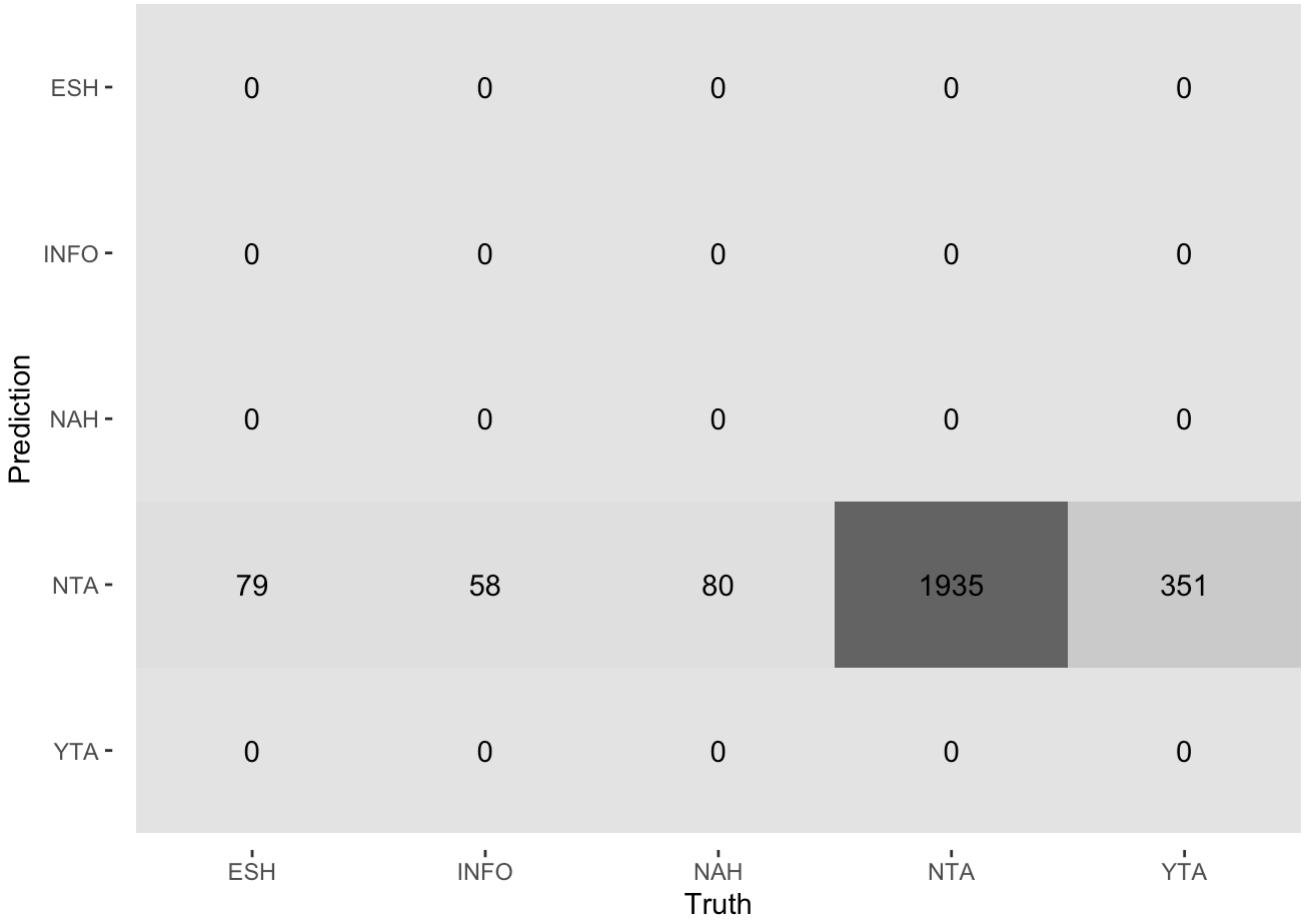
Hide

```
svm_poly_final <- finalize_workflow(svm_poly_wf,
                                    select_best(svm_poly_res, metric = "roc_auc",
                                                degree, cost, scale_factor)) %>%
  fit(df_train)

svm_poly_augmented <- augment(svm_poly_final, new_data = df_train)

svm_poly_augmented %>%
roc_curve(judgment, .pred_ESH:.pred_YTA) %>%
  autoplot()
```



Hide

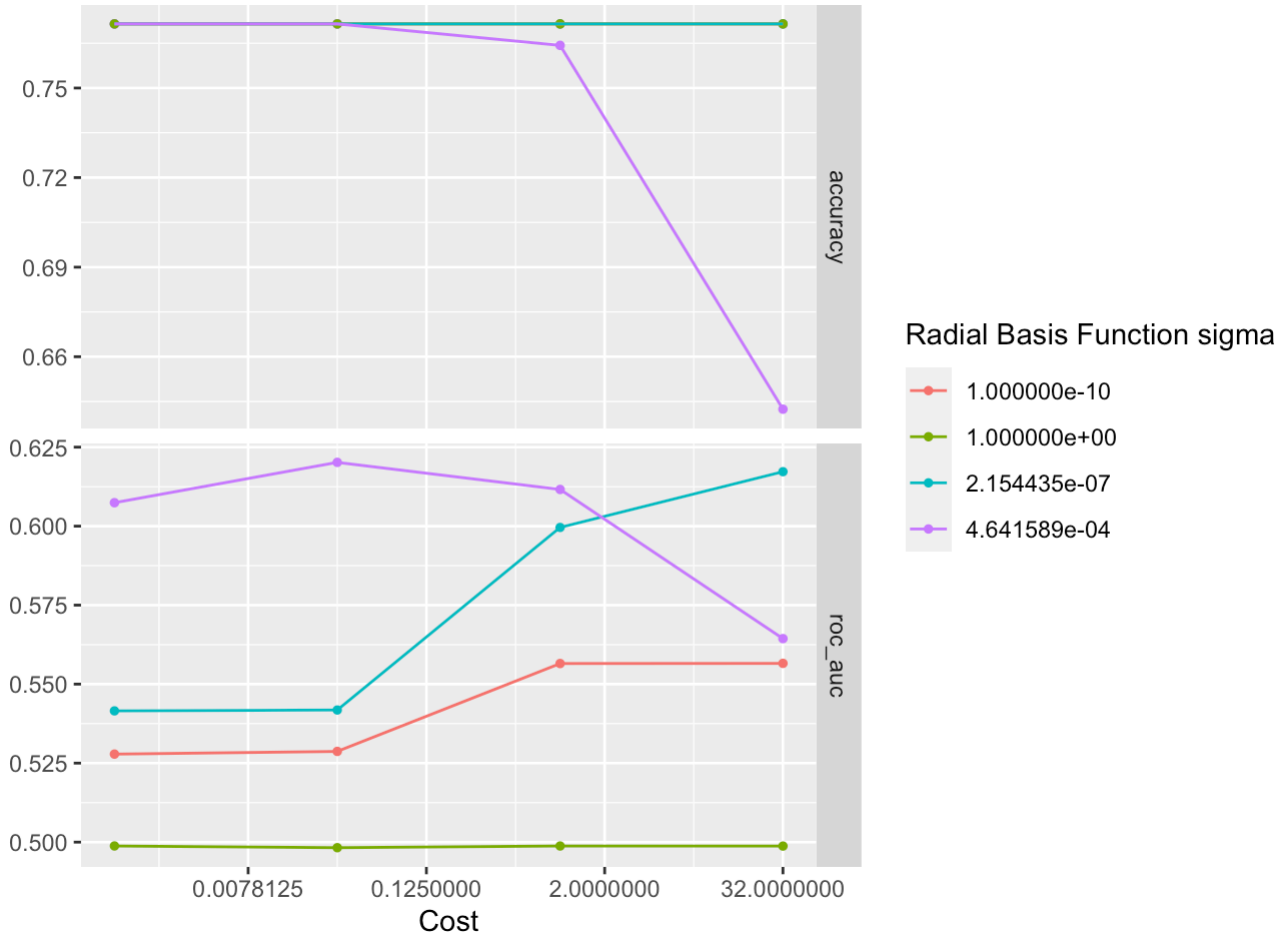
```
autoplot(conf_mat(svm_poly_augmented,
                  truth = judgment,
                  estimate = .pred_class),
         type = 'heatmap')
```



SVM radial basis function

Hide

```
load(file = 'svm_rbf_res.Rda')
autoplot(svm_rbf_res)
```



It looks like a combination of cost and radial sigma function values in the middle of the spectrum have the best results.

Hide

```
select_best(svm_rbf_res, metric = "roc_auc", cost, rbf_sigma)
```

```
## # A tibble: 1 × 3
##     cost rbf_sigma .config
##    <dbl>     <dbl> <chr>
## 1 0.0312  0.000464 Preprocessor1_Model10
```

Hide

```
select_best(svm_rbf_res, metric = "accuracy", cost, rbf_sigma)
```

```
## # A tibble: 1 × 3
##       cost    rbf_sigma .config
##      <dbl>        <dbl> <chr>
## 1 0.000977 0.0000000001 Preprocessor1_Model01
```

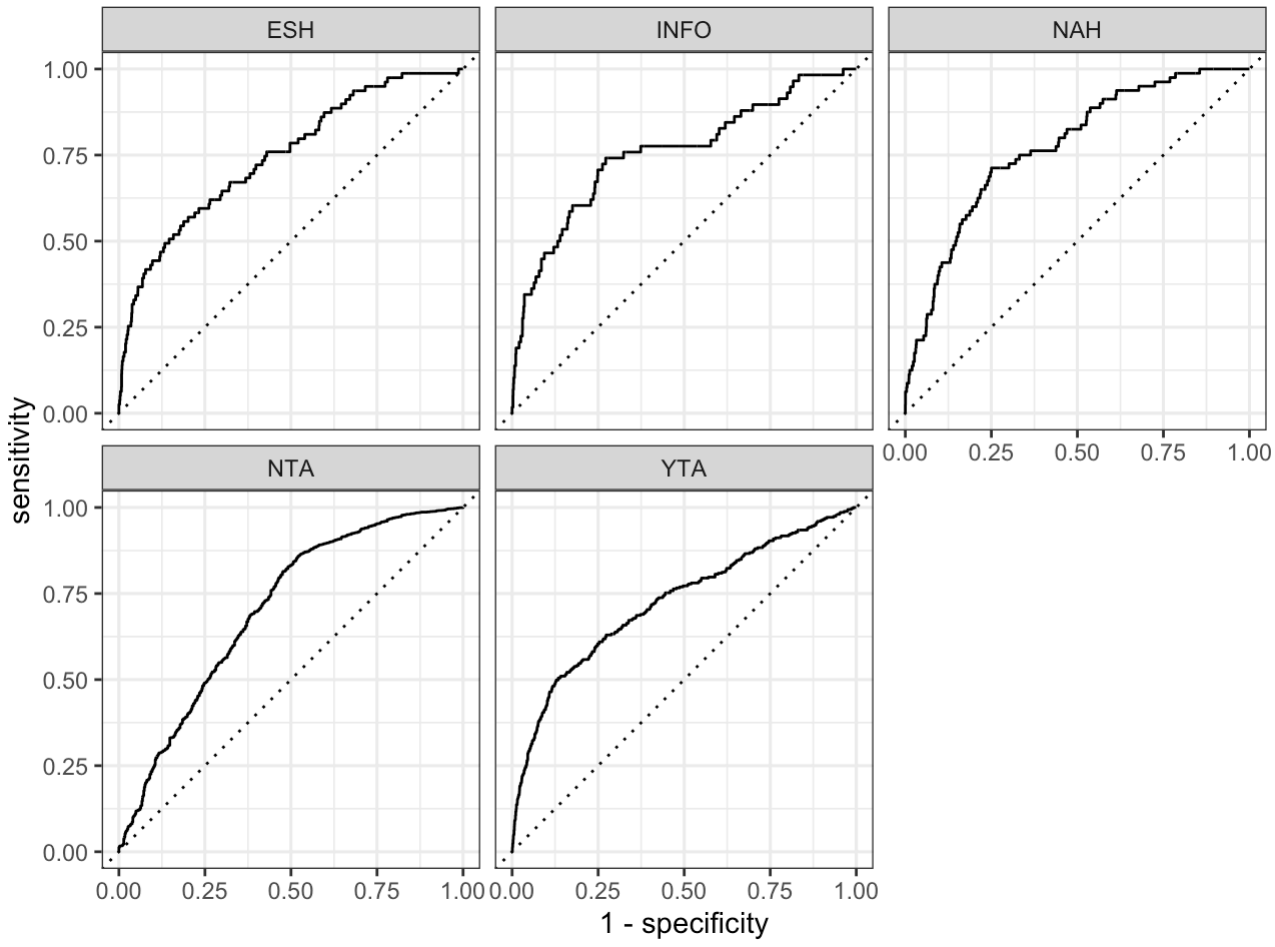
Hide

```
svm_rbf_metrics <- collect_metrics(svm_rbf_res)
svm_rbf_roc_auc <- svm_rbf_metrics %>%
  filter(.metric == "roc_auc")
svm_rbf_roc_auc %>%
  filter(mean == max(svm_rbf_roc_auc$mean)) %>%
  select(.metric, cost, rbf_sigma, mean, std_err)
```

```
## # A tibble: 1 × 5
##   .metric   cost rbf_sigma  mean std_err
##   <chr>    <dbl>     <dbl> <dbl>   <dbl>
## 1 roc_auc 0.0312  0.000464 0.620  0.0130
```

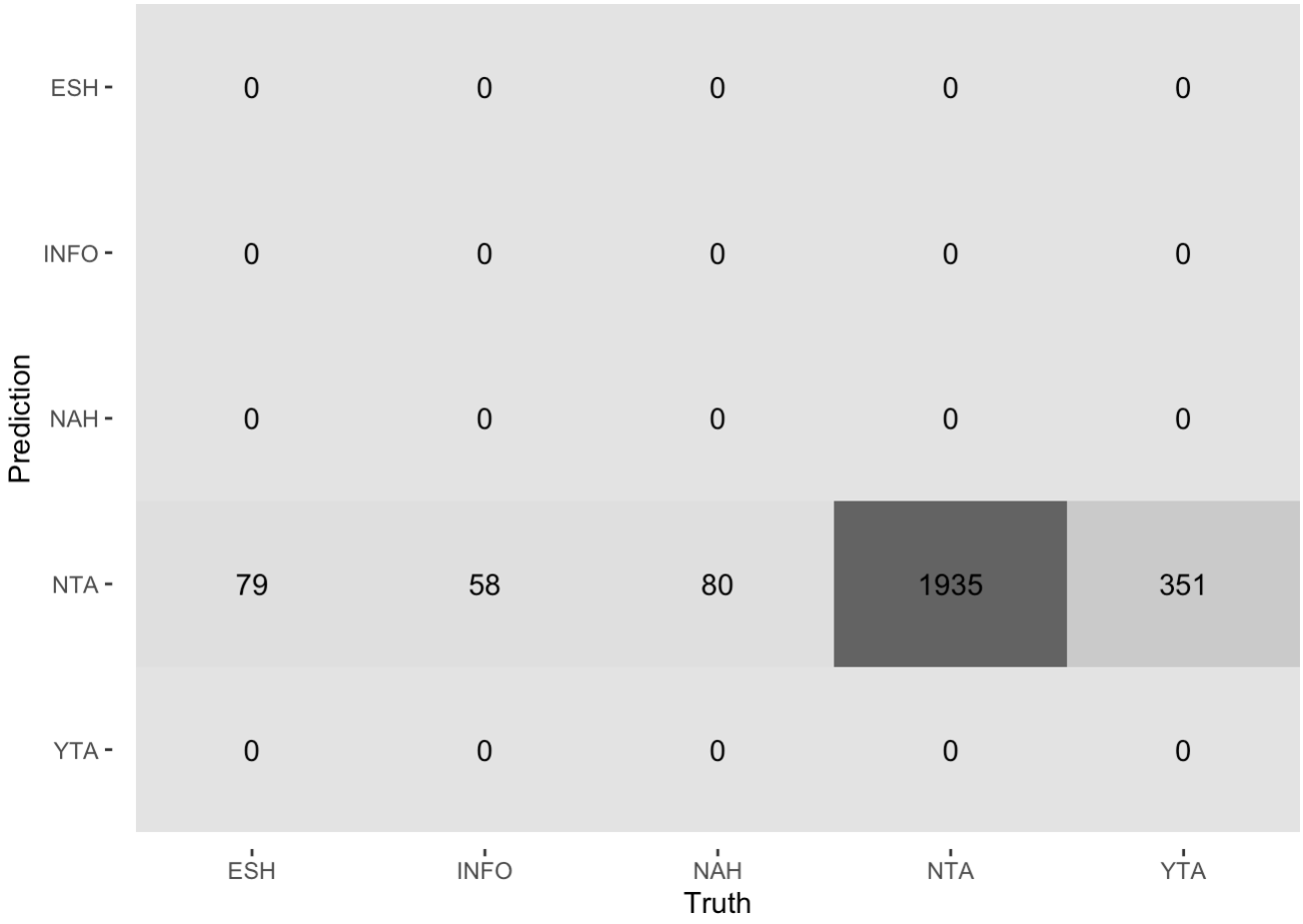
Hide

```
svm_rbf_final <- finalize_workflow(svm_rbf_wf,
                                   select_best(svm_rbf_res, metric = "roc_auc",
                                               cost, rbf_sigma)) %>%
  fit(df_train)

svm_rbf_augmented <- augment(svm_rbf_final, new_data = df_train)

svm_rbf_augmented %>%
roc_curve(judgment, .pred_ESH:.pred_YTA) %>%
  autoplot()
```



Hide

```
autoplot(conf_mat(svm_rbf_augmented,
                  truth = judgment,
                  estimate = .pred_class),
         type = 'heatmap')
```



Although both the SVM models had a higher ROC AUC value than KNN, this is due to them only predicting the NTA class.

Random forest

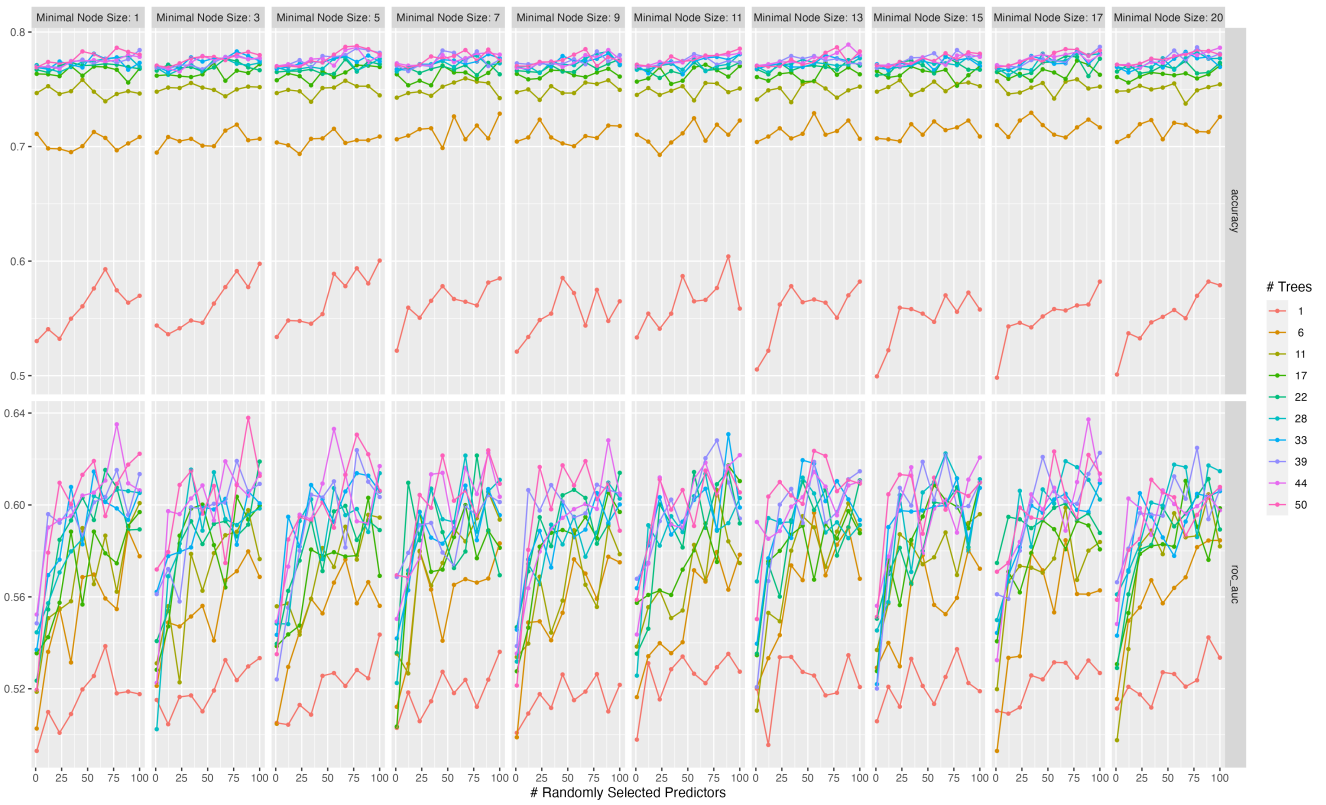
Hide

```
load(file = 'rf_res.Rda')
rf_metrics <- collect_metrics(rf_res)
rf_metrics %>%
  filter(.metric == "roc_auc") %>%
  select(.metric, mtry, trees, min_n, mean, std_err)
```

```
## # A tibble: 1,000 × 6
##    .metric  mtry trees min_n  mean std_err
##    <chr>   <int> <int> <int> <dbl>   <dbl>
##  1 roc_auc     1     1     1 0.493 0.00543
##  2 roc_auc    12     1     1 0.510 0.00558
##  3 roc_auc    23     1     1 0.501 0.00393
##  4 roc_auc    34     1     1 0.509 0.00590
##  5 roc_auc    45     1     1 0.520 0.00535
##  6 roc_auc    56     1     1 0.526 0.00634
##  7 roc_auc    67     1     1 0.539 0.0108
##  8 roc_auc    78     1     1 0.518 0.00479
##  9 roc_auc    89     1     1 0.519 0.00917
## 10 roc_auc   100     1     1 0.518 0.00532
## # ℹ 990 more rows
```

Hide

```
autoplot(rf_res)
ggsave('rf_res.png', width=15, height=9)
```



Hide

```
select_best(rf_res, metric = "roc_auc", mtry, trees, min_n)
```

```
## # A tibble: 1 × 4
##    mtry trees min_n .config
##   <int> <int> <int> <chr>
## 1    89    50     3 Preprocessor1_Model0199
```

Hide

```
select_best(rf_res, metric = "accuracy", mtry, trees, min_n)
```

```
## # A tibble: 1 × 4
##    mtry trees min_n .config
##   <int> <int> <int> <chr>
## 1    89    44    13 Preprocessor1_Model0689
```

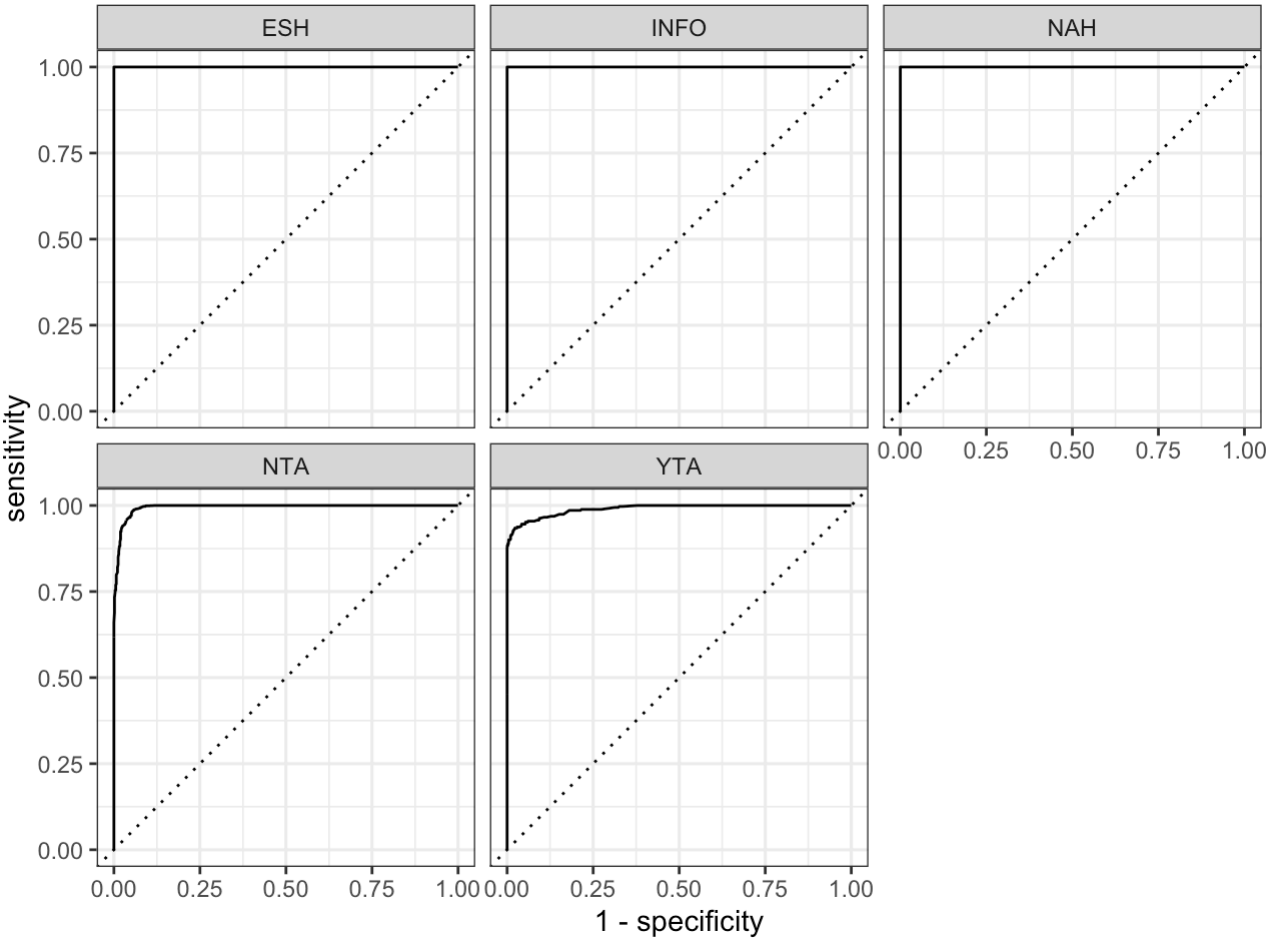
Hide

```
rf_roc_auc <- rf_metrics %>%
  filter(.metric == "roc_auc")
rf_roc_auc %>%
  filter(mean == max(rf_roc_auc$mean)) %>%
  select(.metric, mtry, trees, min_n, mean, std_err)
```

```
## # A tibble: 1 × 6
##   .metric  mtry trees min_n  mean std_err
##   <chr>   <int> <int> <int> <dbl>   <dbl>
## 1 roc_auc    89    50     3 0.638 0.00631
```

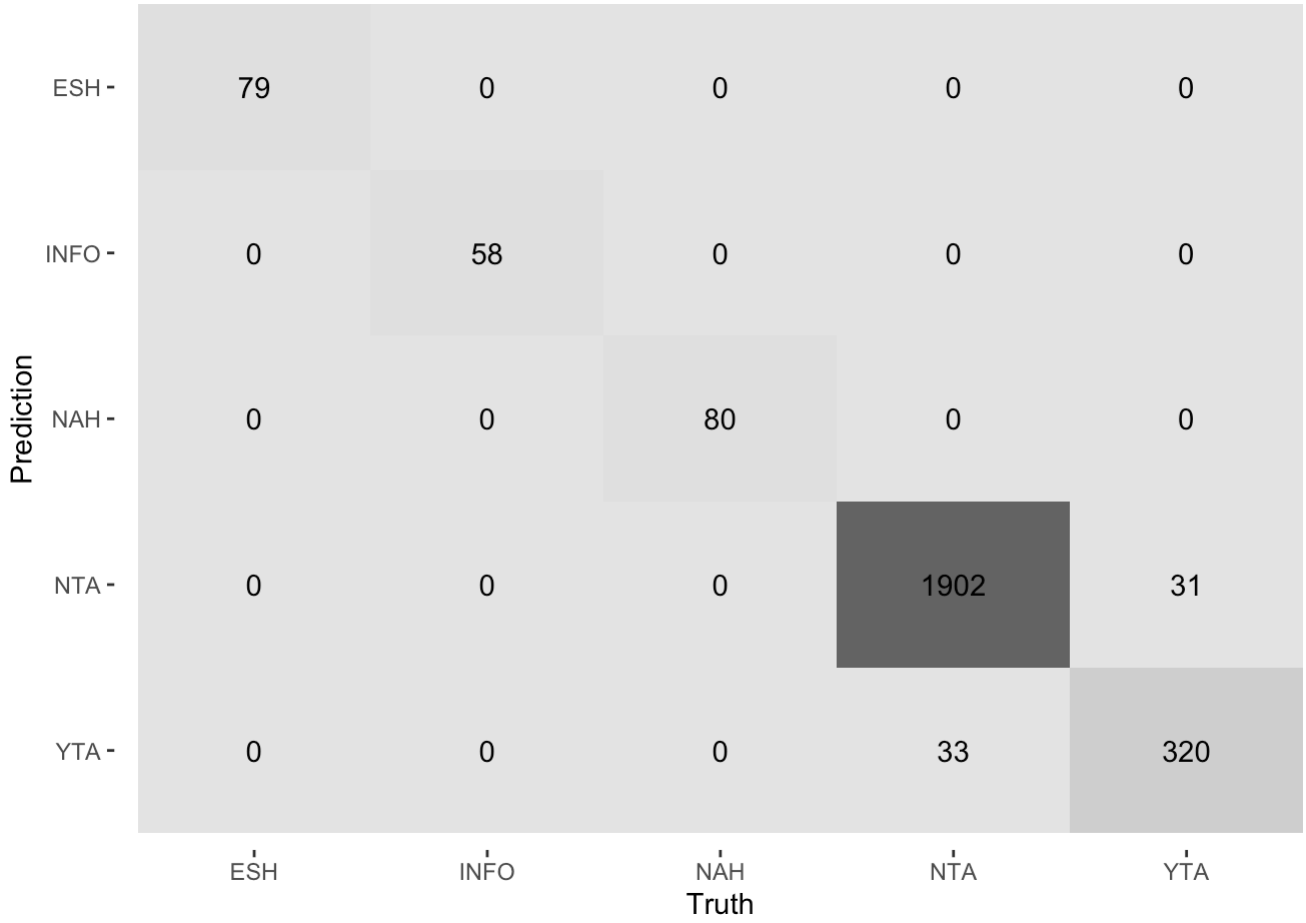
Hide

```
rf_final <- finalize_workflow(rf_wf,
                              select_best(rf_res, metric = "roc_auc",
                                          mtry, trees, min_n)) %>%
  fit(df_train)

rf_final_augmented <- augment(rf_final, new_data=df_train)

rf_final_augmented %>%
roc_curve(judgment, .pred_ESH:.pred_YTA) %>%
  autoplot()
```



Hide

```
autoplot(conf_mat(rf_final_augmented,
                  truth = judgment,
                  estimate = .pred_class),
         type = 'heatmap')
```



The random forest model did the best out of all the models on the training data in terms of both ROC AUC and looking at the results shown in the confusion matrix. In training the model, more trees and a middle amount of predictors resulted in better performance. I'll choose this model with the tuned parameters that got the best ROC AUC value as my final model.

Evaluating final model on testing data

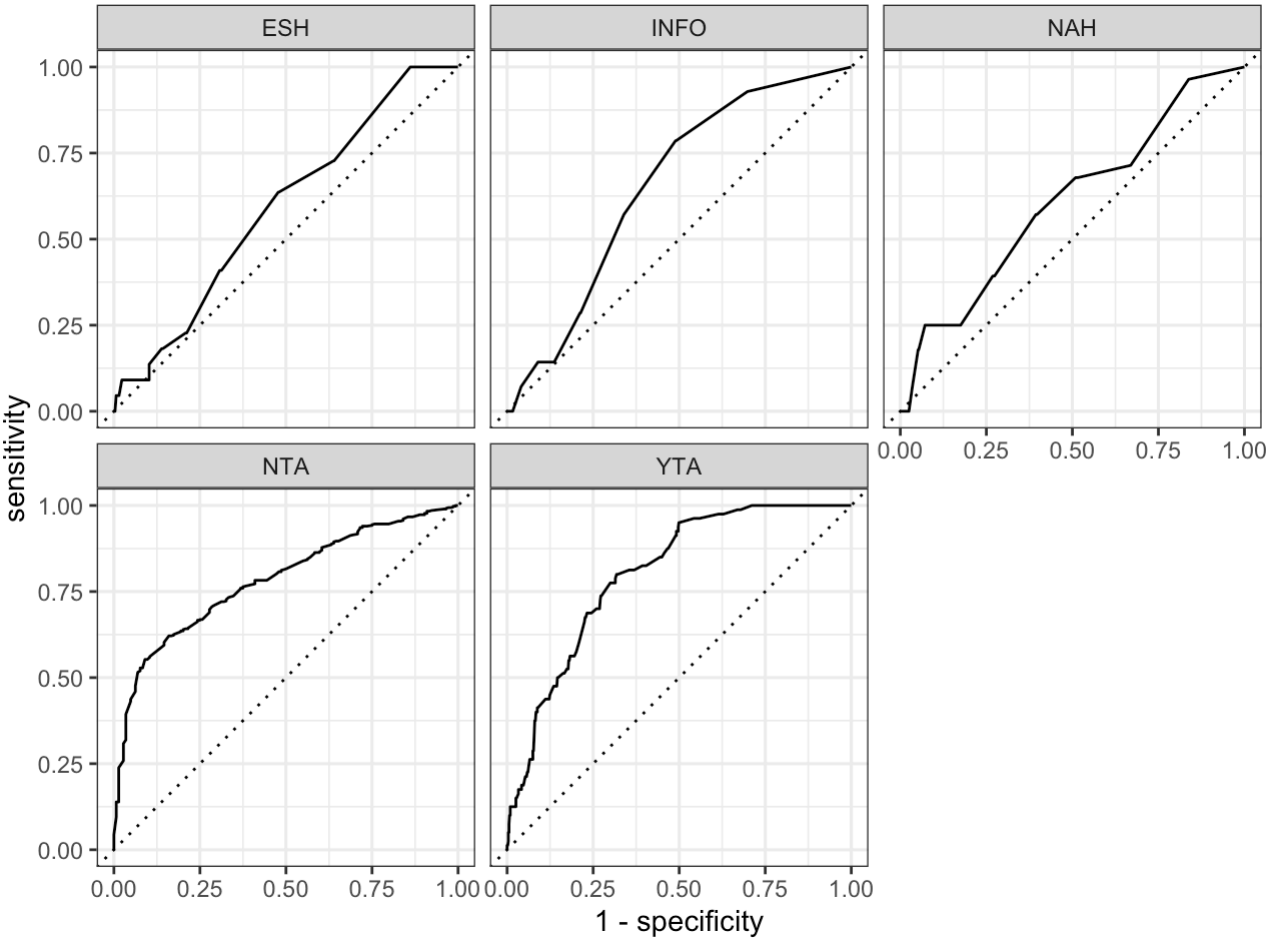
Hide

```
rf_test_augmented <- augment(rf_final, new_data=df_test)

roc_auc(rf_test_augmented, truth = judgment, .pred_ESH:.pred_YTA)
```

```
## # A tibble: 1 × 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc hand_till      0.623
```

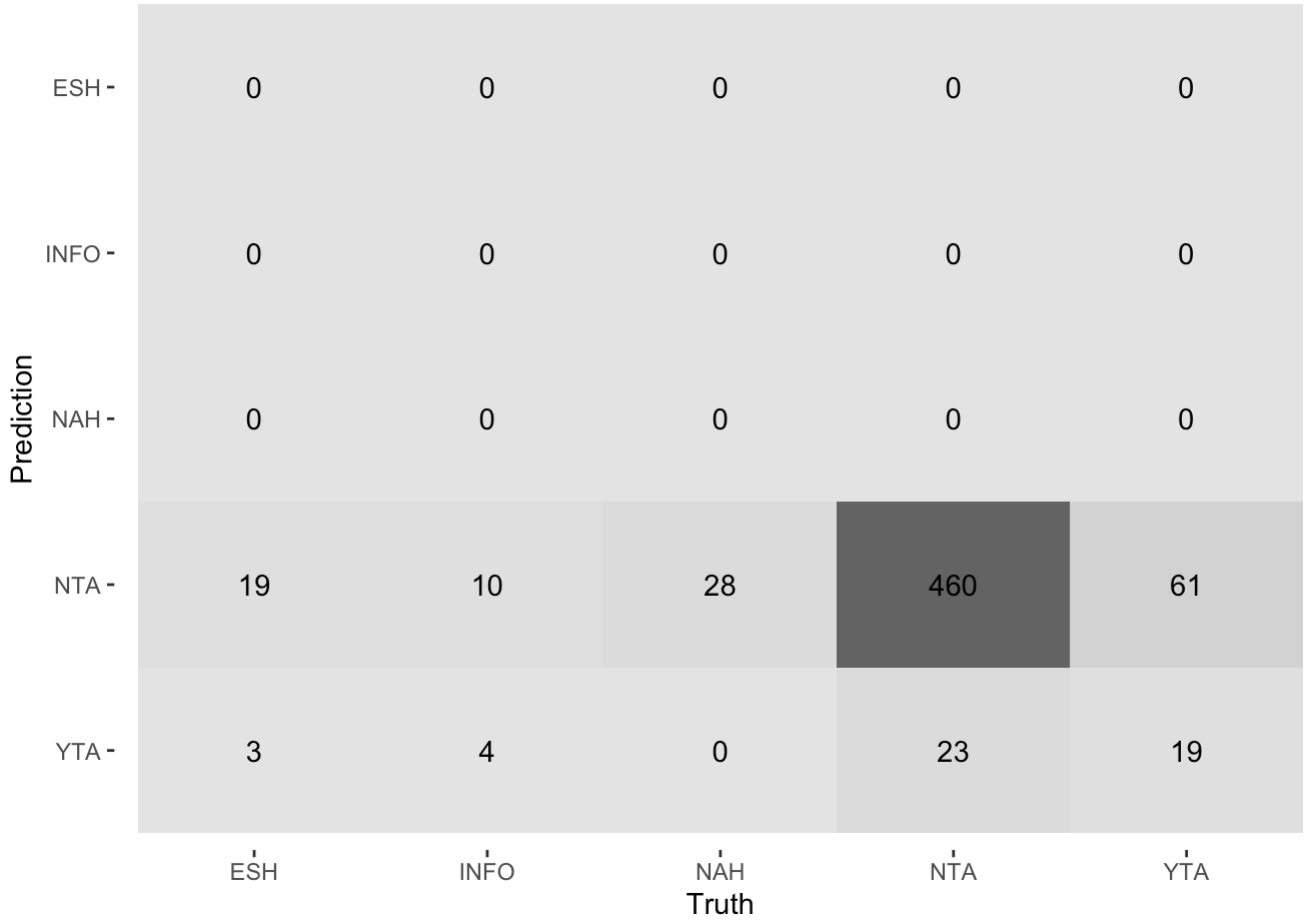
Hide

```
rf_test_augmented %>%
roc_curve(judgment, .pred_ESH:.pred_YTA) %>%
  autoplot()
```



Hide

```
autoplot(conf_mat(rf_test_augmented,
                  truth = judgment,
                  estimate = .pred_class),
         type = 'heatmap')
```



The random forest model didn't perform as well on the testing set, but that's usually to be expected. It did completely get the ESH, INFO, and NAH classes wrong due to basically never predicting them, but did at least okay on the NTA and YTA classes.

Conclusion

After testing all of these models, it's clear that this classification problem is hard to predict. By splitting the text data of the Reddit posts and titles into separate words, removing common filler words, and using pretrained word embeddings, I was able to reduce the number of features in the text data and convert them to vectors that could be analyzed by machine learning models while retaining as much of the original meaning and relationships as I could.

Out of all the models I tried, the random forest model did the best, as it was able to almost perfectly classify the posts in the training data. It was also able to maintain decent performance on the NTA and YTA posts in the testing data, but not the ESH, NAH, and INFO ones. This is probably because of the imbalance of the post categories, with there being very few posts of those three categories to the point where performing upsampling on those post categories contained too many duplicates, causing overfitting on those classes.

Ultimately the random forest model wasn't completely able to predict who was the asshole in each post, but neither can people, so it definitely has some potential. In the future, finding more posts of the minority categories to train it on could probably help its performance a lot.