MC ML knit 2

Andreas Methling

30/11/2021

```r
library(readr)
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --

## v ggplot2 3.3.5     v dplyr   1.0.7
## v tibble  3.1.4     v stringr 1.4.0
## v tidyr   1.1.3     v forcats 0.5.1
## v purrr   0.3.4

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```r
data_start <- read_csv("C:/Users/andre/Desktop/lyrics-data.csv")
```

```
## Rows: 209522 Columns: 5

## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (5): ALink, SName, SLink, Lyric, Idiom

##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
artists_data <- read_csv("C:/Users/andre/Downloads/artists-data.csv")
```

```
## Rows: 3242 Columns: 6

## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (4): Artist, Link, Genre, Genres
## dbl (2): Songs, Popularity

##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
artists = artists_data %>%
  group_by(Artist) %>%
  count(Genre) %>%
  pivot_wider(names_from = Genre, values_from = n) %>%
  replace_na(list(Pop = 0, "Hip Hop" = 0, Rock = 0, "Funk Carioca" = 0, "Sertanejo" = 0, Samba = 0 )) %
  ungroup() %>%
  left_join(artists_data, by = c("Artist")) %>%
  select(-c(Genre, Genres, Popularity)) %>%
  distinct()
```

```
glimpse(data_start)
```

```
## Rows: 209,522
## Columns: 5
## $ ALink <chr> "/10000-maniacs/", "/10000-maniacs/", "/10000-maniacs/", "/10000~
## $ SName <chr> "More Than This", "Because The Night", "These Are Days", "A Camp~
## $ SLink <chr> "/10000-maniacs/more-than-this.html", "/10000-maniacs/because-th~
## $ Lyric <chr> "I could feel at the time. There was no way of knowing. Fallen l~
## $ Idiom <chr> "ENGLISH", "ENGLISH", "ENGLISH", "ENGLISH", "ENGLISH", "ENGLISH"~
```

```
data = data_start %>%
  filter(Idiom == "ENGLISH") %>%
  rename("Link" = "ALink") %>%
  inner_join(artists, by = c("Link")) %>%
  distinct() %>%
  mutate(name = paste(Artist, SName))%>%
  rename(text=Lyric) %>%
  filter(Rock==1, Pop==1) %>%
  select(name, text)%>%
  distinct(name, .keep_all = T)


data %>%
  count(name, sort = T)
```

```
## # A tibble: 3,348 x 2
##    name                                             n
##    <chr>                                        <int>
##  1 10000 Maniacs A Campfire Song                    1
##  2 10000 Maniacs A Room For Everything              1
##  3 10000 Maniacs Across The Fields                  1
##  4 10000 Maniacs All That Never Happens             1
##  5 10000 Maniacs Among The Americans                1
##  6 10000 Maniacs Angels, From The Realms Of Glory   1
##  7 10000 Maniacs Anthem For Doomed Youth            1
##  8 10000 Maniacs Arbor Day                          1
##  9 10000 Maniacs Back O' The Moon                   1
## 10 10000 Maniacs Because The Night                  1
## # ... with 3,338 more rows
```

Make labels

```
library(tidytext)
text_tidy = data %>% unnest_tokens(word, text, token = "words")

head(text_tidy)
```

```
## # A tibble: 6 x 2
##   name                        word
##   <chr>                       <chr>
## 1 10000 Maniacs More Than This i
## 2 10000 Maniacs More Than This could
## 3 10000 Maniacs More Than This feel
## 4 10000 Maniacs More Than This at
## 5 10000 Maniacs More Than This the
## 6 10000 Maniacs More Than This time
```

```
text_tidy %<>%
  filter(str_length(word) > 2 ) %>%
  group_by(word) %>%
  ungroup() %>%
  anti_join(stop_words, by = 'word')
```

```
library(hunspell)
text_tidy %>%
  mutate(stem = hunspell_stem(word)) %>%
  unnest(stem) %>%
  count(stem, sort = TRUE)
```

```
## # A tibble: 8,047 x 2
##    stem      n
##    <chr> <int>
##  1 love   5789
##  2 time   3466
##  3 feel   3005
##  4 yeah   2512
##  5 baby   2215
##  6 gonna  2166
##  7 day    2042
##  8 wanna  1944
##  9 life   1769
## 10 heart  1730
## # ... with 8,037 more rows
```

```
text_tidy %<>%
  mutate(stem = hunspell_stem(word)) %>%
  unnest(stem) %>%
   select(-word) %>%
  rename(word = stem)
```

```
top_10000_words=text_tidy %>%
  count(word,sort = T) %>%
  head(10000) %>%
  select(word)

data_top_10000=top_10000_words %>%
  left_join(text_tidy, by= c("word"))
```

nrc multiclass

```
library(magrittr)
```

```
##
## Vedhæfter pakke: 'magrittr'

## Det følgende objekt er maskeret fra 'package:purrr':
##
##     set_names

## Det følgende objekt er maskeret fra 'package:tidyr':
##
##     extract
```

```
sentiment_nrc <- text_tidy %>%
  inner_join(get_sentiments("nrc"))
```

```
## Joining, by = "word"
```

```
multi_data=sentiment_nrc %>%
  filter(sentiment %in% c("negative", "positive", "joy", "fear")) %>%
  count(name, sentiment) %>%
  pivot_wider(names_from = sentiment, values_from = n, values_fill = 0) %>%
  mutate(label= pmax(positive, joy, negative, fear)) %>%
  mutate(label= ifelse(label == fear, "fear", ifelse(label == positive, "positive", ifelse(label == nega
  select(name, label) %>%
  inner_join(data)
```

```
## Joining, by = "name"
```

```
multi_data_new=sentiment_nrc %>%
  filter(sentiment %in% c("trust", "sadness", "joy", "fear")) %>%
  count(name, sentiment) %>%
  pivot_wider(names_from = sentiment, values_from = n, values_fill = 0) %>%
  mutate(label= pmax(trust, joy, sadness, fear)) %>%
  mutate(label= ifelse(label == fear, "fear", ifelse(label == trust, "trust", ifelse(label == sadness,
  select(name, label) %>%
  rename(y= label)%>%
  inner_join(data)
```

```
## Joining, by = "name"
```

```r
multi_data_new %>%
  count(y)
```

```
## # A tibble: 4 x 2
##   y           n
##   <chr>   <int>
## 1 fear      759
## 2 joy      1119
## 3 sadness   722
## 4 trust     726
```

```r
multi_data_new %<>%
  select(-name)


multi_data %>%
  count(label)
```

```
## # A tibble: 4 x 2
##   label        n
##   <chr>    <int>
## 1 fear       185
## 2 joy          5
## 3 negative  1106
## 4 positive  2042
```

```r
library(rsample)

split= initial_split(multi_data_new, prop = 0.75)

train_data= training(split)
test_data= testing(split)
```

```r
library(recipes)
```

```
##
## Vedhæfter pakke: 'recipes'

## Det følgende objekt er maskeret fra 'package:stringr':
##
##     fixed

## Det følgende objekt er maskeret fra 'package:stats':
##
##     step
```

```r
train_data <- recipe(y~., data = train_data) %>%
  themis::step_downsample(y) %>%
  prep() %>%
  juice()
```

```
## Registered S3 methods overwritten by 'themis':
##   method                  from
##   bake.step_downsample    recipes
##   bake.step_upsample      recipes
##   prep.step_downsample    recipes
##   prep.step_upsample      recipes
##   tidy.step_downsample    recipes
##   tidy.step_upsample      recipes
##   tunable.step_downsample recipes
##   tunable.step_upsample   recipes
```

```
train_data %>%
  count(y)
```

```
## # A tibble: 4 x 2
##   y           n
##   <fct>   <int>
## 1 fear      538
## 2 joy       538
## 3 sadness   538
## 4 trust     538
```

And can now see that the classes are evenly distributed.

```
library(textdata)

glove6b <- embedding_glove6b(dimensions = 100)
```

We create the three recipies we want to use.

```
library(textrecipes)


tf_idf_rec <- recipe(y~., data = train_data) %>%
  step_tokenize(text) %>%
  step_stem(text) %>%
  step_stopwords(text) %>%
  step_tokenfilter(text, max_tokens = 1000) %>%
  step_tfidf(all_predictors())


embeddings_rec <- recipe(y~., data = train_data) %>%
  step_tokenize(text) %>%
  step_stem(text) %>%
  step_stopwords(text) %>%
  step_tokenfilter(text, max_tokens = 1000) %>%
  step_word_embeddings(text, embeddings = embedding_glove6b())


hash_rec <- recipe(y~., data = train_data) %>%
  step_tokenize(text) %>%
  step_stem(text) %>%
```

```
  step_stopwords(text) %>%
  step_tokenfilter(text, max_tokens = 1000) %>%
  step_texthash(text, num_terms = 100)
```

Define models Term frequency

We define three models:

All models are coded to do multiclass predcitions. We set some of the parameters for tuning.

Logistic model

```
library(tidymodels)
```

```
## Registered S3 method overwritten by 'tune':
##   method                   from
##   required_pkgs.model_spec parsnip

## -- Attaching packages -------------------------------------- tidymodels 0.1.3 --

## v broom        0.7.9      v tune         0.1.6
## v dials        0.0.10     v workflows    0.2.3
## v infer        1.0.0      v workflowsets 0.1.0
## v modeldata    0.1.1      v yardstick    0.0.8
## v parsnip      0.1.7

## -- Conflicts ----------------------------------------- tidymodels_conflicts() --
## x scales::discard()     masks purrr::discard()
## x magrittr::extract()   masks tidyr::extract()
## x dplyr::filter()       masks stats::filter()
## x recipes::fixed()      masks stringr::fixed()
## x dplyr::lag()          masks stats::lag()
## x magrittr::set_names() masks purrr::set_names()
## x yardstick::spec()     masks readr::spec()
## x recipes::step()       masks stats::step()
## * Use tidymodels_prefer() to resolve common conflicts.
```

```
model_lg <- multinom_reg(penalty = tune(), mixture = 1) %>%
  set_engine("glmnet") %>%
  set_mode("classification")
```

KNN model

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

Random Forrest

```
model_rf <-
  rand_forest() %>%
  set_engine("ranger", importance = "impurity") %>%
  set_mode("classification")
```

Workflow

We create workflows for each recipe.

tf_idf

```
workflow_general_tf <- workflow() %>%
  add_recipe(tf_idf_rec)

workflow_lg_tf <- workflow_general_tf %>%
  add_model(model_lg)

workflow_knn_tf <- workflow_general_tf %>%
  add_model(model_knn)

workflow_rf_tf <- workflow_general_tf %>%
  add_model(model_rf)
```

Embeding

```
workflow_general_emb <- workflow() %>%
  add_recipe(embeddings_rec)

workflow_lg_emb <- workflow_general_emb %>%
  add_model(model_lg)

workflow_knn_emb <- workflow_general_emb %>%
  add_model(model_knn)

workflow_rf_emb <- workflow_general_emb %>%
  add_model(model_rf)
```

hash

```
workflow_general_hash <- workflow() %>%
  add_recipe(hash_rec)

workflow_lg_hash <- workflow_general_hash %>%
  add_model(model_lg)
```

```
workflow_knn_hash <- workflow_general_hash %>%
  add_model(model_knn)

workflow_rf_hash <- workflow_general_hash %>%
  add_model(model_rf)
```

Hyper tuneing

We use vfold_cv to create resampled data. to perfrom hypertuning and fitting.

```
set.seed(100)

k_folds_data <- train_data %>%
  vfold_cv(strata = y,
           v = 3,
           repeats = 3)
```

Define Grids

We define the grids we want to use for the hypertuning

```
logistic_grid <- grid_regular(parameters(model_lg), levels = 3)
knn_grid <- grid_regular(parameters(model_knn), levels = 5, filter = c(neighbors > 1))
```

The level defines the amount of parameters that should be considered.

Define tuning process

We define which measures we want to be able to choose best parameters from.

```
model_control <- control_grid(save_pred = TRUE)
model_metrics <- metric_set(accuracy, sens, spec, mn_log_loss, roc_auc)
```

Tune Models

We tune the three different models

```
# Tune hash models
linear_hash_res <- tune_grid(
  model_lg,
  hash_rec,
  grid = logistic_grid,
  control = model_control,
  metrics = model_metrics,
  resamples = k_folds_data
)
```

```r
knn_hash_res <- tune_grid(
  model_knn,
  hash_rec,
  grid = knn_grid,
  control = model_control,
  metrics = model_metrics,
  resamples = k_folds_data
)
```

```r
# Tune embed models
linear_embed_res <- tune_grid(
  model_lg,
  embeddings_rec,
  grid = logistic_grid,
  control = model_control,
  metrics = model_metrics,
  resamples = k_folds_data
)


knn_embed_res <- tune_grid(
  model_knn,
  embeddings_rec,
  grid = knn_grid,
  control = model_control,
  metrics = model_metrics,
  resamples = k_folds_data
)
```

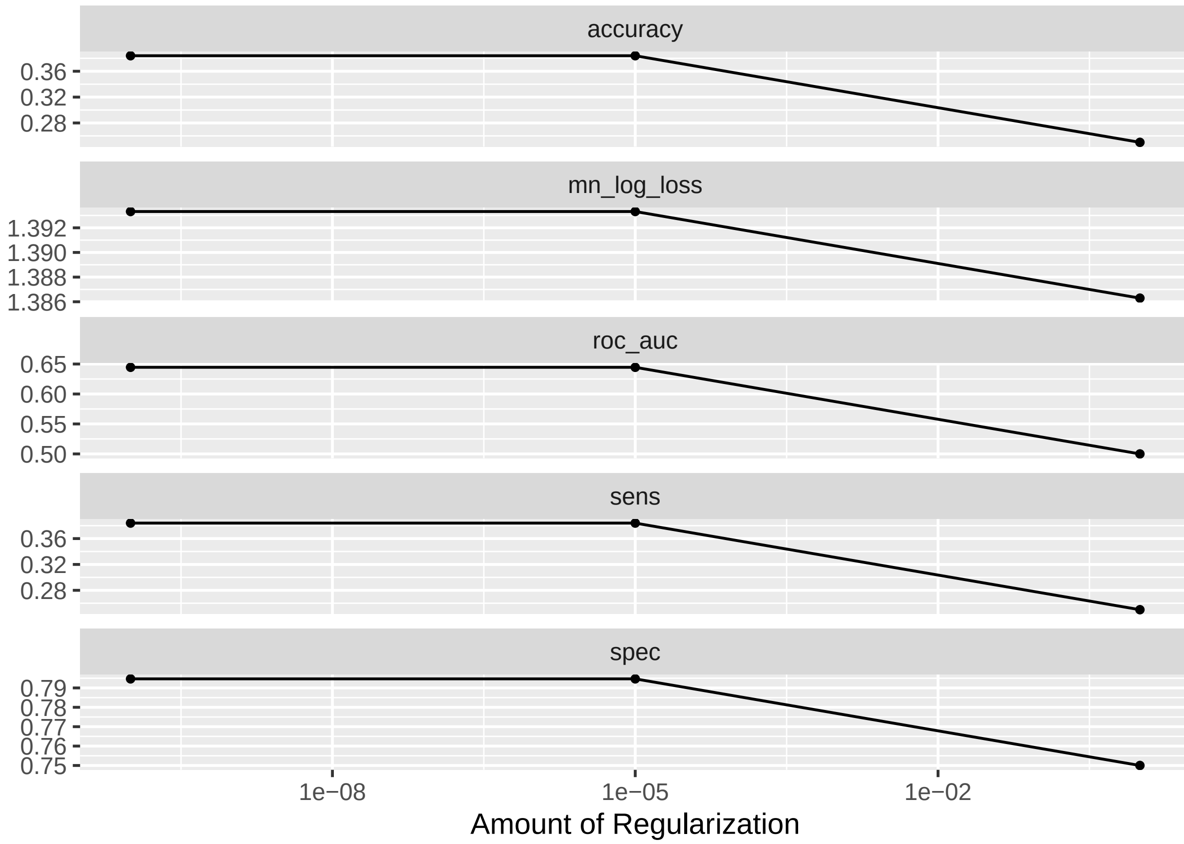
```r
# Tune tf-idf models
linear_tf_res <- tune_grid(
  model_lg,
  tf_idf_rec,
  grid = logistic_grid,
  control = model_control,
  metrics = model_metrics,
  resamples = k_folds_data
)


knn_tf_res <- tune_grid(
  model_knn,
  tf_idf_rec,
  grid = knn_grid,
  control = model_control,
  metrics = model_metrics,
  resamples = k_folds_data
)
```

Best parameters

We look at the different optimizations and choose the best parameters.

linear_embed_res We use autoplot

```
linear_hash_res %>% autoplot()
```

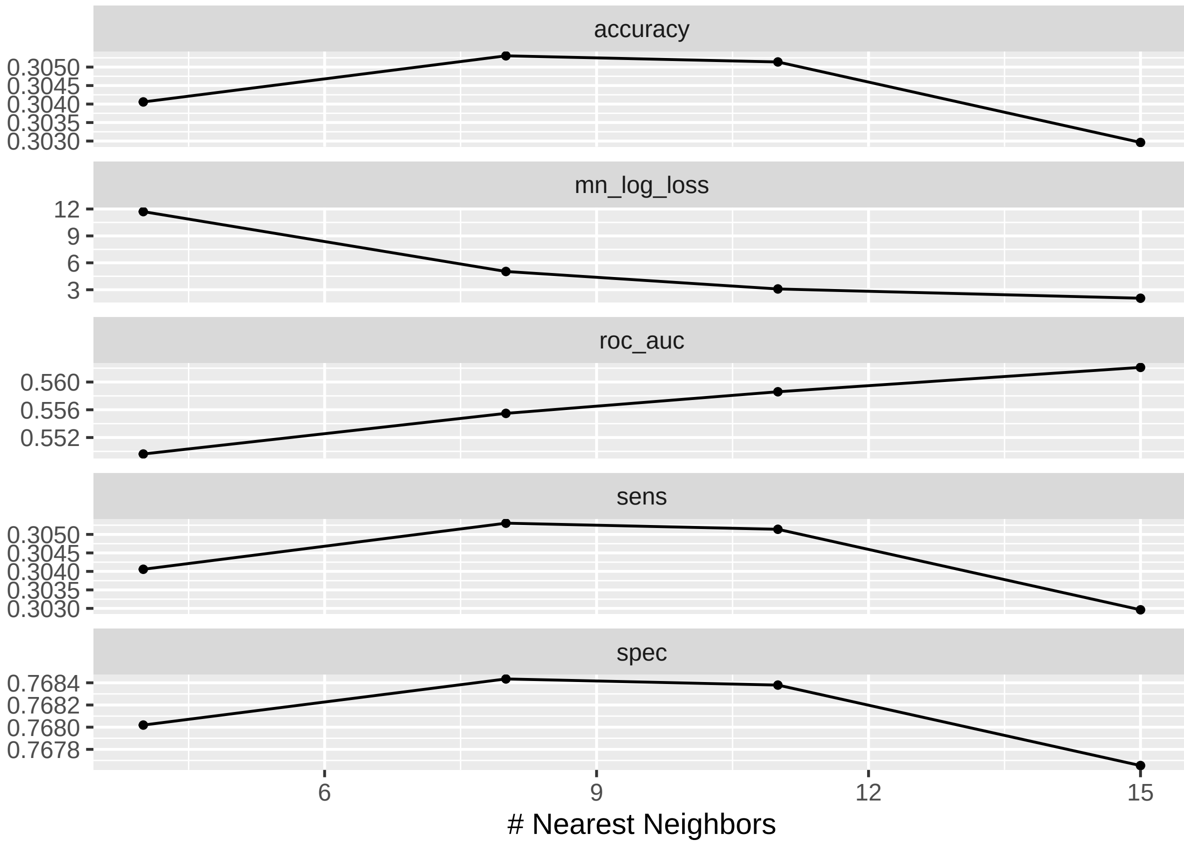



```
best_param_linear_hash_res <- linear_hash_res %>% select_best(metric = 'accuracy')
best_param_linear_hash_res
```

```
## # A tibble: 1 x 2
##        penalty .config
##          <dbl> <chr>
## 1 0.0000000001 Preprocessor1_Model1
```

knn_embed_res We use autoplot

```
knn_hash_res %>% autoplot()
```

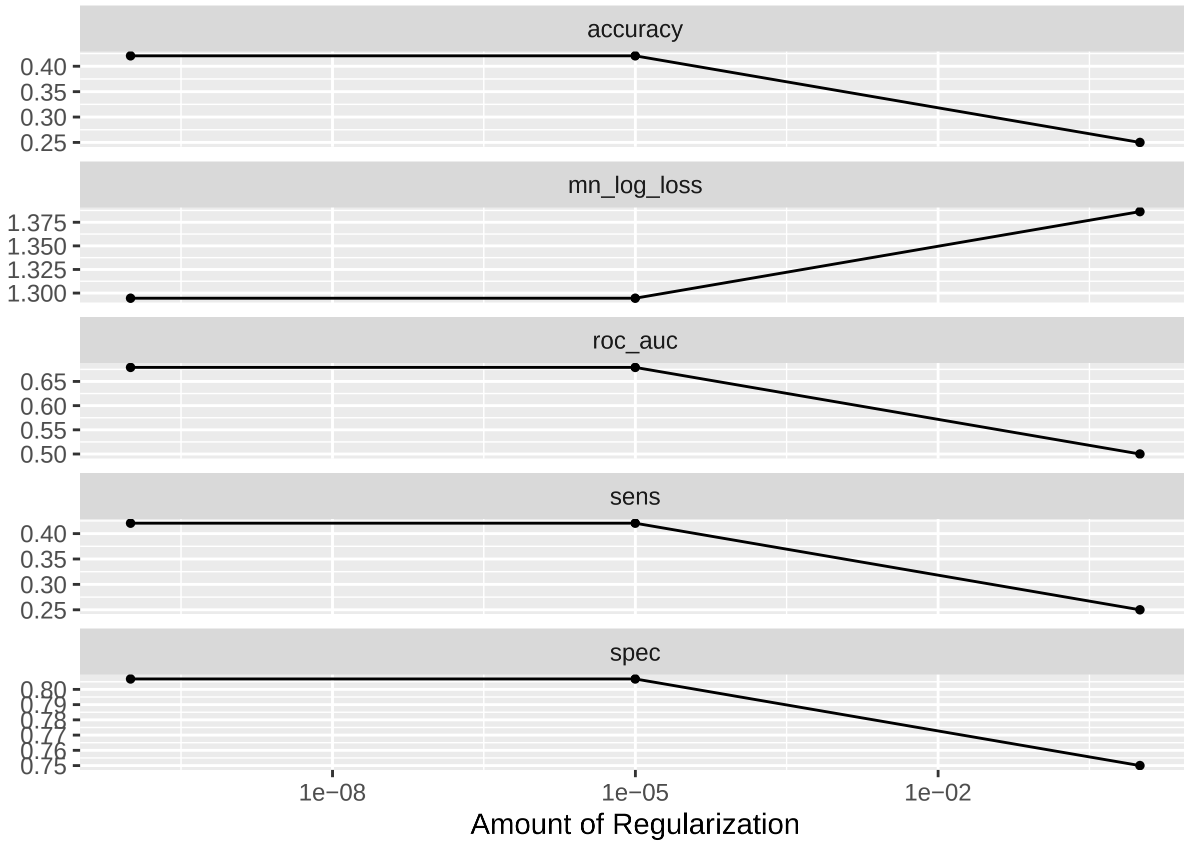



```
best_param_knn_hash_res <- knn_hash_res %>% select_best(metric = 'accuracy')
best_param_knn_hash_res
```

```
## # A tibble: 1 x 2
##   neighbors .config
##       <int> <chr>
## 1         8 Preprocessor1_Model2
```

linear_embed_res We use autoplot

```
linear_embed_res %>% autoplot()
```

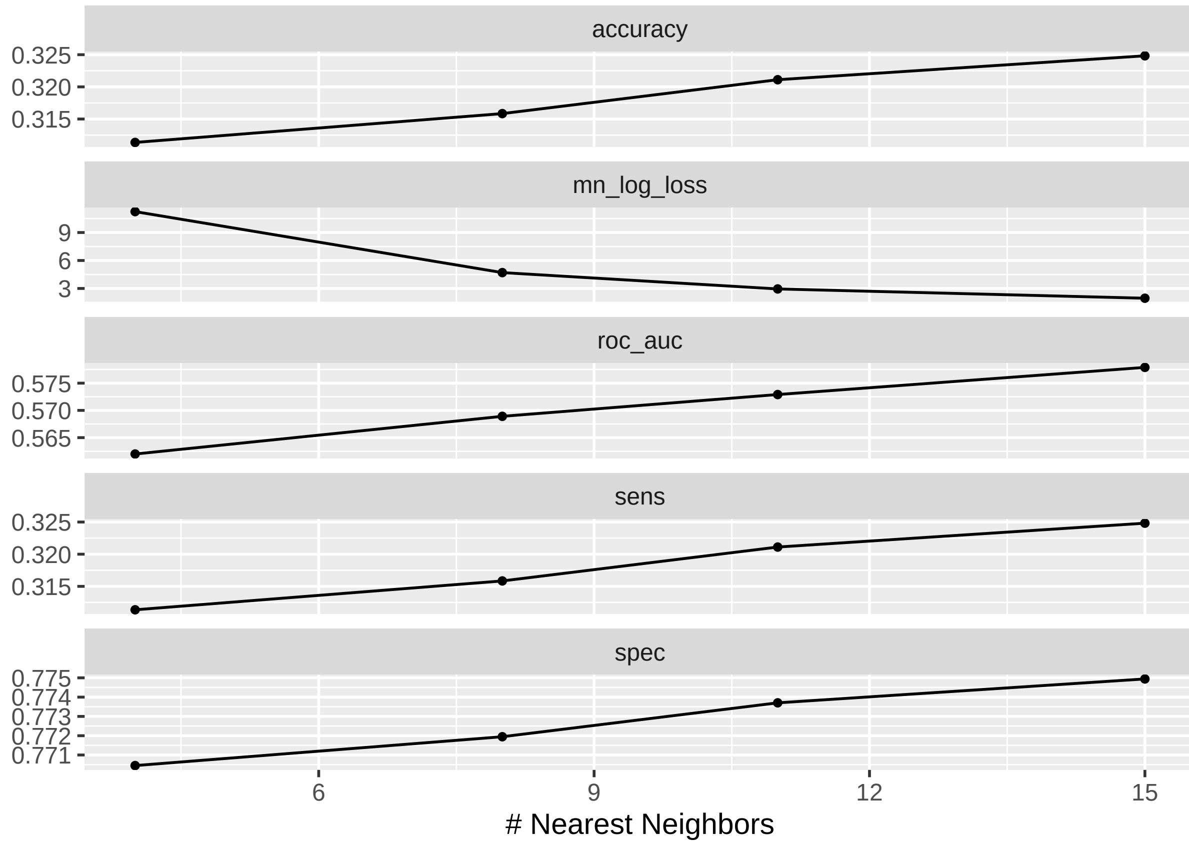



```r
best_param_linear_embed_res <- linear_embed_res %>% select_best(metric = 'accuracy')
best_param_linear_embed_res
```

```
## # A tibble: 1 x 2
##        penalty .config
##          <dbl> <chr>
## 1 0.0000000001 Preprocessor1_Model1
```

knn_embed_res We use autoplot

```r
knn_embed_res %>% autoplot()
```

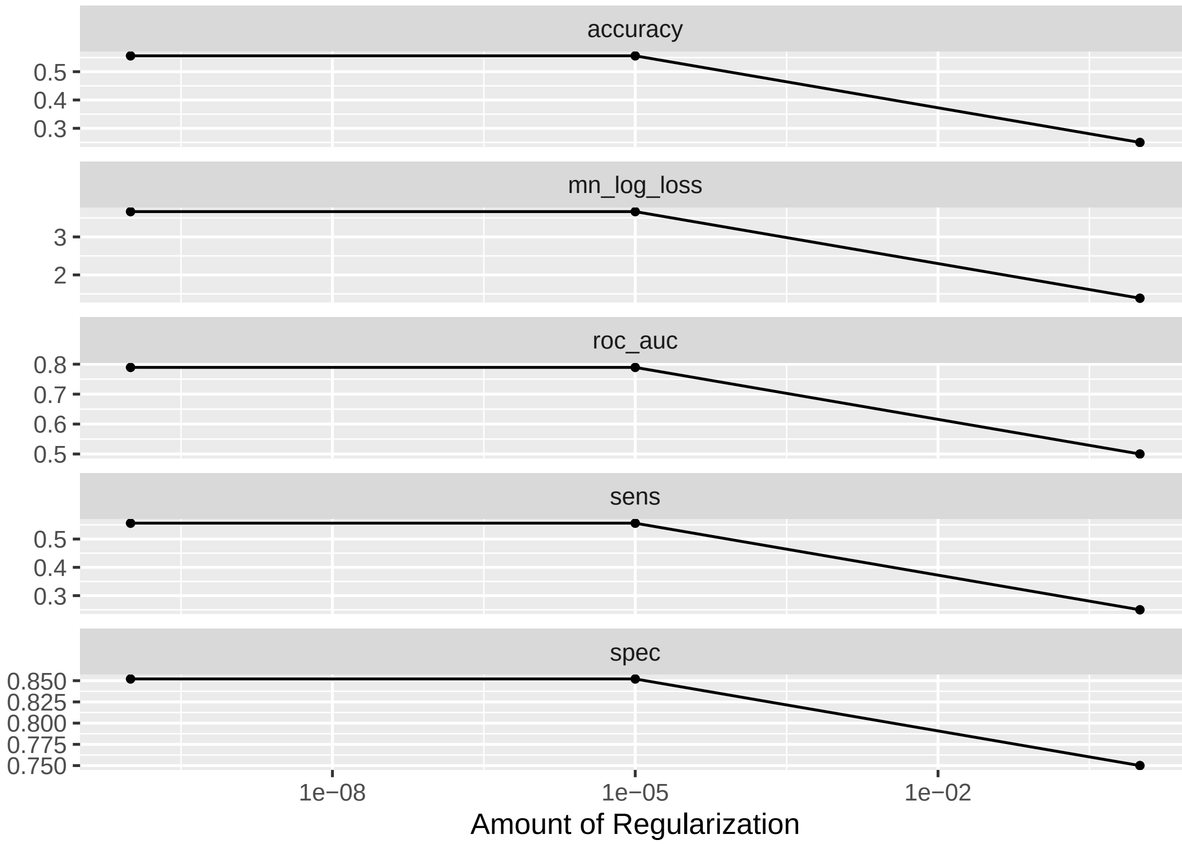



```r
best_param_knn_embed_res <- knn_embed_res %>% select_best(metric = 'accuracy')
best_param_knn_embed_res
```

```
## # A tibble: 1 x 2
##   neighbors .config
##       <int> <chr>
## 1        15 Preprocessor1_Model4
```

linear_tf_res We use autoplot

```r
linear_tf_res %>% autoplot()
```

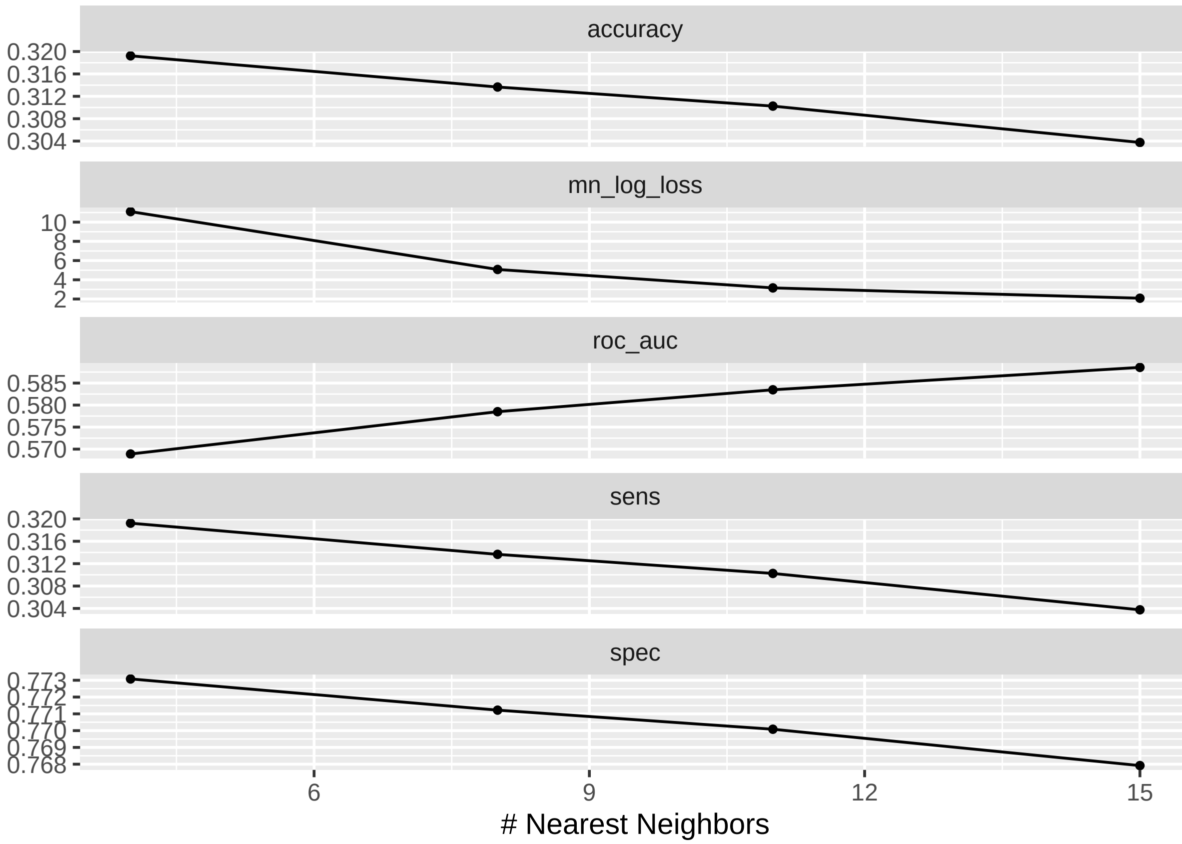



```
best_param_linear_tf_res <- linear_tf_res %>% select_best(metric = 'accuracy')
best_param_linear_tf_res
```

```
## # A tibble: 1 x 2
##        penalty .config
##          <dbl> <chr>
## 1 0.0000000001 Preprocessor1_Model1
```

knn_tf_res We use autoplot

```
knn_tf_res %>% autoplot()
```




```
best_param_knn_tf_res <- knn_tf_res %>% select_best(metric = 'accuracy')
best_param_knn_tf_res
```

```
## # A tibble: 1 x 2
##   neighbors .config
##       <int> <chr>
## 1         4 Preprocessor1_Model1
```

Finalize workflows

We now fit the best parameters into the workflow of the two models that needed hypertuning.

Hash

```
workflow_final_lg_hash <- workflow_lg_hash %>%
  finalize_workflow(parameters = best_param_linear_hash_res)

workflow_final_knn_hash <- workflow_knn_hash %>%
  finalize_workflow(parameters = best_param_knn_hash_res)
```

Tf-idf

```
workflow_final_lg_tf <- workflow_lg_tf %>%
  finalize_workflow(parameters = best_param_linear_tf_res)

workflow_final_knn_tf <- workflow_knn_tf %>%
  finalize_workflow(parameters = best_param_knn_tf_res)
```

Embedings

```
workflow_final_lg_emb <- workflow_lg_emb %>%
  finalize_workflow(parameters = best_param_linear_embed_res)

workflow_final_knn_emb <- workflow_knn_emb %>%
  finalize_workflow(parameters = best_param_knn_embed_res)
```

Evaluate models

here we us the resampled data to evaluate the models.

Logistic regression

```
log_res_hash <-
  workflow_final_lg_hash %>%
  fit_resamples(
    resamples = k_folds_data,
    metrics = metric_set(
      recall, precision, f_meas,
      accuracy, kap,
      roc_auc, sens, spec),
    control = control_resamples(
      save_pred = TRUE)
    )

log_res_hash %>% collect_metrics(summarize = TRUE)
```

hash

```
## # A tibble: 8 x 6
##   .metric   .estimator  mean     n std_err .config
##   <chr>     <chr>      <dbl> <int>   <dbl> <chr>
## 1 accuracy  multiclass 0.384     9 0.00615 Preprocessor1_Model1
## 2 f_meas    macro      0.383     9 0.00610 Preprocessor1_Model1
## 3 kap       multiclass 0.179     9 0.00821 Preprocessor1_Model1
## 4 precision macro      0.384     9 0.00591 Preprocessor1_Model1
## 5 recall    macro      0.384     9 0.00615 Preprocessor1_Model1
```

```
## 6 roc_auc   hand_till  0.645     9 0.00451 Preprocessor1_Model1
## 7 sens      macro      0.384     9 0.00615 Preprocessor1_Model1
## 8 spec      macro      0.795     9 0.00205 Preprocessor1_Model1
```

```r
log_res_tf <-
  workflow_final_lg_tf %>%
  fit_resamples(
    resamples = k_folds_data,
    metrics = metric_set(
      recall, precision, f_meas,
      accuracy, kap,
      roc_auc, sens, spec),
    control = control_resamples(
      save_pred = TRUE)
    )

log_res_tf %>% collect_metrics(summarize = TRUE)
```

Tf_idf

```
## # A tibble: 8 x 6
##   .metric   .estimator  mean     n std_err .config
##   <chr>     <chr>      <dbl> <int>   <dbl> <chr>
## 1 accuracy  multiclass 0.556     9 0.00635 Preprocessor1_Model1
## 2 f_meas    macro      0.556     9 0.00630 Preprocessor1_Model1
## 3 kap       multiclass 0.408     9 0.00847 Preprocessor1_Model1
## 4 precision macro      0.558     9 0.00631 Preprocessor1_Model1
## 5 recall    macro      0.556     9 0.00635 Preprocessor1_Model1
## 6 roc_auc   hand_till  0.789     9 0.00290 Preprocessor1_Model1
## 7 sens      macro      0.556     9 0.00635 Preprocessor1_Model1
## 8 spec      macro      0.852     9 0.00212 Preprocessor1_Model1
```

```r
log_res_emb <-
  workflow_final_lg_emb %>%
  fit_resamples(
    resamples = k_folds_data,
    metrics = metric_set(
      recall, precision, f_meas,
      accuracy, kap,
      roc_auc, sens, spec),
    control = control_resamples(
      save_pred = TRUE)
    )

log_res_emb %>% collect_metrics(summarize = TRUE)
```

Embeding

```
## # A tibble: 8 x 6
##   .metric   .estimator  mean     n std_err .config
##   <chr>     <chr>      <dbl> <int>   <dbl> <chr>
## 1 accuracy  multiclass 0.421     9 0.00471 Preprocessor1_Model1
## 2 f_meas    macro      0.419     9 0.00453 Preprocessor1_Model1
## 3 kap       multiclass 0.227     9 0.00628 Preprocessor1_Model1
## 4 precision macro      0.421     9 0.00437 Preprocessor1_Model1
## 5 recall    macro      0.421     9 0.00471 Preprocessor1_Model1
## 6 roc_auc   hand_till  0.679     9 0.00561 Preprocessor1_Model1
## 7 sens      macro      0.421     9 0.00471 Preprocessor1_Model1
## 8 spec      macro      0.807     9 0.00157 Preprocessor1_Model1
```

KNN model

```
knn_res_hash <-
  workflow_final_knn_hash %>%
  fit_resamples(
    resamples = k_folds_data,
    metrics = metric_set(
      recall, precision, f_meas,
      accuracy, kap,
      roc_auc, sens, spec),
    control = control_resamples(
      save_pred = TRUE)
    )

knn_res_hash %>% collect_metrics(summarize = TRUE)
```

Hash

```
## # A tibble: 8 x 6
##   .metric   .estimator   mean     n std_err .config
##   <chr>     <chr>       <dbl> <int>   <dbl> <chr>
## 1 accuracy  multiclass 0.305      9 0.00573 Preprocessor1_Model1
## 2 f_meas    macro      0.301      9 0.00635 Preprocessor1_Model1
## 3 kap       multiclass 0.0737     9 0.00764 Preprocessor1_Model1
## 4 precision macro      0.307      9 0.00623 Preprocessor1_Model1
## 5 recall    macro      0.305      9 0.00573 Preprocessor1_Model1
## 6 roc_auc   hand_till  0.555      9 0.00479 Preprocessor1_Model1
## 7 sens      macro      0.305      9 0.00573 Preprocessor1_Model1
## 8 spec      macro      0.768      9 0.00191 Preprocessor1_Model1
```

```
knn_res_tf <-
  workflow_final_knn_tf %>%
  fit_resamples(
    resamples = k_folds_data,
    metrics = metric_set(
      recall, precision, f_meas,
```

```r
      accuracy, kap,
      roc_auc, sens, spec),
    control = control_resamples(
      save_pred = TRUE)
    )

knn_res_tf %>% collect_metrics(summarize = TRUE)
```

TF-idf

```
## # A tibble: 8 x 6
##   .metric   .estimator   mean     n std_err .config
##   <chr>     <chr>       <dbl> <int>   <dbl> <chr>
## 1 accuracy  multiclass 0.319      9 0.00675 Preprocessor1_Model1
## 2 f_meas    macro      0.311      9 0.00695 Preprocessor1_Model1
## 3 kap       multiclass 0.0923     9 0.00900 Preprocessor1_Model1
## 4 precision macro      0.326      9 0.00832 Preprocessor1_Model1
## 5 recall    macro      0.319      9 0.00675 Preprocessor1_Model1
## 6 roc_auc   hand_till  0.569      9 0.00447 Preprocessor1_Model1
## 7 sens      macro      0.319      9 0.00675 Preprocessor1_Model1
## 8 spec      macro      0.773      9 0.00225 Preprocessor1_Model1
```

```r
knn_res_emb <-
  workflow_final_knn_emb %>%
  fit_resamples(
    resamples = k_folds_data,
    metrics = metric_set(
      recall, precision, f_meas,
      accuracy, kap,
      roc_auc, sens, spec),
    control = control_resamples(
      save_pred = TRUE)
    )

knn_res_emb %>% collect_metrics(summarize = TRUE)
```

Embedings

```
## # A tibble: 8 x 6
##   .metric   .estimator   mean     n std_err .config
##   <chr>     <chr>       <dbl> <int>   <dbl> <chr>
## 1 accuracy  multiclass 0.325      9 0.00572 Preprocessor1_Model1
## 2 f_meas    macro      0.321      9 0.00590 Preprocessor1_Model1
## 3 kap       multiclass 0.0998     9 0.00762 Preprocessor1_Model1
## 4 precision macro      0.322      9 0.00585 Preprocessor1_Model1
## 5 recall    macro      0.325      9 0.00572 Preprocessor1_Model1
## 6 roc_auc   hand_till  0.578      9 0.00602 Preprocessor1_Model1
## 7 sens      macro      0.325      9 0.00572 Preprocessor1_Model1
## 8 spec      macro      0.775      9 0.00191 Preprocessor1_Model1
```

Random forest model

```
rf_res_hash <-
  workflow_rf_hash %>%
  fit_resamples(
    resamples = k_folds_data,
    metrics = metric_set(
      recall, precision, f_meas,
      accuracy, kap,
      roc_auc, sens, spec),
    control = control_resamples(
      save_pred = TRUE)
    )

rf_res_hash %>% collect_metrics(summarize = TRUE)
```

hash

```
## # A tibble: 8 x 6
##   .metric   .estimator  mean     n std_err .config
##   <chr>     <chr>      <dbl> <int>   <dbl> <chr>
## 1 accuracy  multiclass 0.436     9 0.00563 Preprocessor1_Model1
## 2 f_meas    macro      0.433     9 0.00533 Preprocessor1_Model1
## 3 kap       multiclass 0.248     9 0.00750 Preprocessor1_Model1
## 4 precision macro      0.433     9 0.00524 Preprocessor1_Model1
## 5 recall    macro      0.436     9 0.00563 Preprocessor1_Model1
## 6 roc_auc   hand_till  0.705     9 0.00316 Preprocessor1_Model1
## 7 sens      macro      0.436     9 0.00563 Preprocessor1_Model1
## 8 spec      macro      0.812     9 0.00188 Preprocessor1_Model1
```

```
rf_res_tf <-
  workflow_rf_tf %>%
  fit_resamples(
    resamples = k_folds_data,
    metrics = metric_set(
      recall, precision, f_meas,
      accuracy, kap,
      roc_auc, sens, spec),
    control = control_resamples(
      save_pred = TRUE)
    )

rf_res_tf %>% collect_metrics(summarize = TRUE)
```

TF-idf

```
## # A tibble: 8 x 6
##   .metric   .estimator  mean     n std_err .config
```

```
##   <chr>     <chr>      <dbl> <int>   <dbl> <chr>
## 1 accuracy  multiclass 0.552     9 0.00585 Preprocessor1_Model1
## 2 f_meas    macro      0.551     9 0.00560 Preprocessor1_Model1
## 3 kap       multiclass 0.403     9 0.00781 Preprocessor1_Model1
## 4 precision macro      0.553     9 0.00534 Preprocessor1_Model1
## 5 recall    macro      0.552     9 0.00585 Preprocessor1_Model1
## 6 roc_auc   hand_till  0.806     9 0.00304 Preprocessor1_Model1
## 7 sens      macro      0.552     9 0.00585 Preprocessor1_Model1
## 8 spec      macro      0.851     9 0.00195 Preprocessor1_Model1
```

```r
rf_res_emb <-
  workflow_rf_emb %>%
  fit_resamples(
    resamples = k_folds_data,
    metrics = metric_set(
      recall, precision, f_meas,
      accuracy, kap,
      roc_auc, sens, spec),
    control = control_resamples(
      save_pred = TRUE)
    )

rf_res_emb %>% collect_metrics(summarize = TRUE)
```

Embedings

```
## # A tibble: 8 x 6
##   .metric   .estimator  mean     n std_err .config
##   <chr>     <chr>      <dbl> <int>   <dbl> <chr>
## 1 accuracy  multiclass 0.383     9 0.00575 Preprocessor1_Model1
## 2 f_meas    macro      0.380     9 0.00559 Preprocessor1_Model1
## 3 kap       multiclass 0.177     9 0.00767 Preprocessor1_Model1
## 4 precision macro      0.380     9 0.00594 Preprocessor1_Model1
## 5 recall    macro      0.383     9 0.00575 Preprocessor1_Model1
## 6 roc_auc   hand_till  0.650     9 0.00664 Preprocessor1_Model1
## 7 sens      macro      0.383     9 0.00575 Preprocessor1_Model1
## 8 spec      macro      0.794     9 0.00192 Preprocessor1_Model1
```

Compare performance

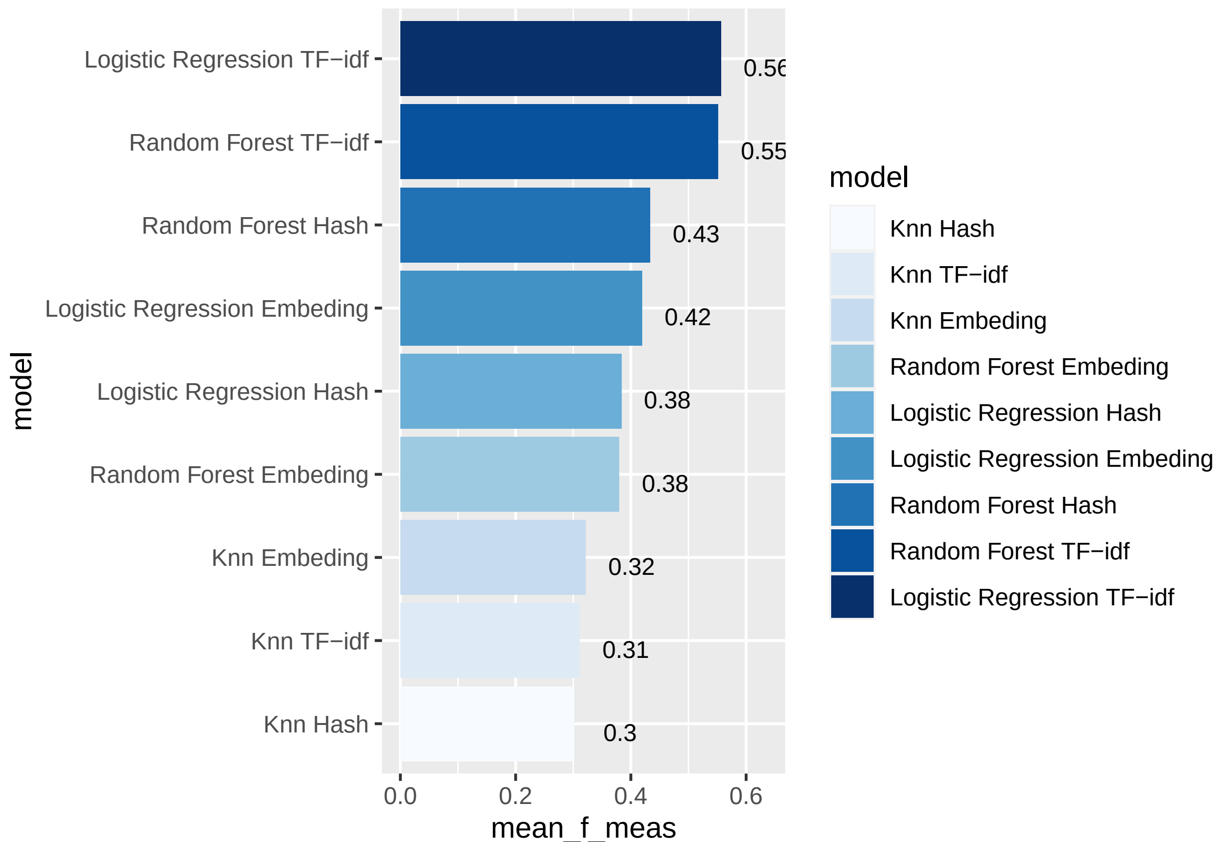
We get a summary for the performed models. We add the model name to each metric to keep the models appart from each other later on.

```r
log_metrics_tf <-
  log_res_tf %>%
  collect_metrics(summarise = TRUE) %>%
  mutate(model = "Logistic Regression TF-idf")

log_metrics_emb <-
  log_res_emb %>%
```

```r
  collect_metrics(summarise = TRUE) %>%
  mutate(model = "Logistic Regression Embeding")

log_metrics_hash <-
  log_res_hash %>%
  collect_metrics(summarise = TRUE) %>%
  mutate(model = "Logistic Regression Hash")

rf_metrics_tf <-
  rf_res_tf %>%
  collect_metrics(summarise = TRUE) %>%
  mutate(model = "Random Forest TF-idf")

rf_metrics_emb <-
  rf_res_emb %>%
  collect_metrics(summarise = TRUE) %>%
  mutate(model = "Random Forest Embeding")

rf_metrics_hash <-
  rf_res_hash %>%
  collect_metrics(summarise = TRUE) %>%
  mutate(model = "Random Forest Hash")

knn_metrics_tf <-
  knn_res_tf %>%
  collect_metrics(summarise = TRUE) %>%
  mutate(model = "Knn TF-idf")

knn_metrics_emb <-
  knn_res_emb %>%
  collect_metrics(summarise = TRUE) %>%
  mutate(model = "Knn Embeding")

knn_metrics_hash <-
  knn_res_hash %>%
  collect_metrics(summarise = TRUE) %>%
  mutate(model = "Knn Hash")
```

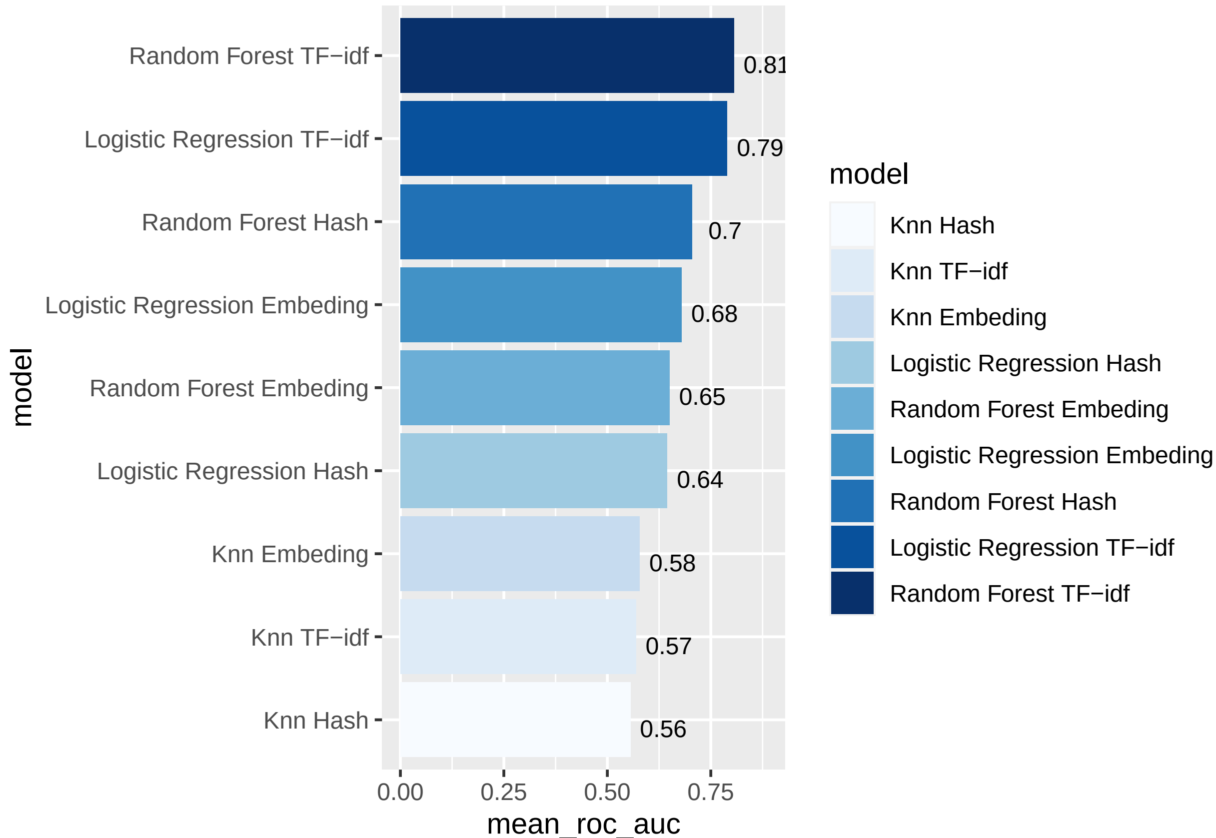
```r
model_compare <- bind_rows(
                          log_metrics_tf,
                          log_metrics_emb,
                          log_metrics_hash,
                          rf_metrics_tf,
                          rf_metrics_emb,
                          rf_metrics_hash,
                          knn_metrics_tf,
                          knn_metrics_emb,
                          knn_metrics_hash
                           )



model_comp <-
  model_compare %>%
```

```
  select(model, .metric, mean, std_err) %>%
  pivot_wider(names_from = .metric, values_from = c(mean, std_err))


model_comp %>%
  arrange(mean_f_meas) %>%
  mutate(model = fct_reorder(model, mean_f_meas)) %>%
  ggplot(aes(model, mean_f_meas, fill=model)) +
  geom_col() +
  coord_flip() +
  scale_fill_brewer(palette = "Blues") +
   geom_text(
     size = 3,
     aes(label = round(mean_f_meas, 2), y = mean_f_meas + 0.08),
     vjust = 1
  )
```




```
model_comp %>%
  arrange(mean_roc_auc) %>%
  mutate(model = fct_reorder(model, mean_roc_auc)) %>%
  ggplot(aes(model, mean_roc_auc, fill=model)) +
  geom_col() +
  coord_flip() +
  scale_fill_brewer(palette = "Blues") +
     geom_text(
```

```
    size = 3,
    aes(label = round(mean_roc_auc, 2), y = mean_roc_auc + 0.08),
    vjust = 1
  )
```



Choose model

The best model seems to be Random Forest using TF-idf we also look at the second best model which is the Logistic Regression model using TF-idf

So we only continue with the two best ones.

Log-reg model

Performance metrics Show average performance over all folds:

```
rf_res_tf %>%  collect_metrics(summarize = TRUE)
```

```
## # A tibble: 8 x 6
##   .metric   .estimator  mean     n std_err .config
##   <chr>     <chr>      <dbl> <int>   <dbl> <chr>
## 1 accuracy  multiclass 0.552     9 0.00585 Preprocessor1_Model1
## 2 f_meas    macro      0.551     9 0.00560 Preprocessor1_Model1
## 3 kap       multiclass 0.403     9 0.00781 Preprocessor1_Model1
```

```
## 4 precision macro      0.553     9 0.00534 Preprocessor1_Model1
## 5 recall    macro      0.552     9 0.00585 Preprocessor1_Model1
## 6 roc_auc   hand_till  0.806     9 0.00304 Preprocessor1_Model1
## 7 sens      macro      0.552     9 0.00585 Preprocessor1_Model1
## 8 spec      macro      0.851     9 0.00195 Preprocessor1_Model1
```

Collect model predictions To obtain the actual model predictions, we use the function collect_predictions and save the result as log_pred:

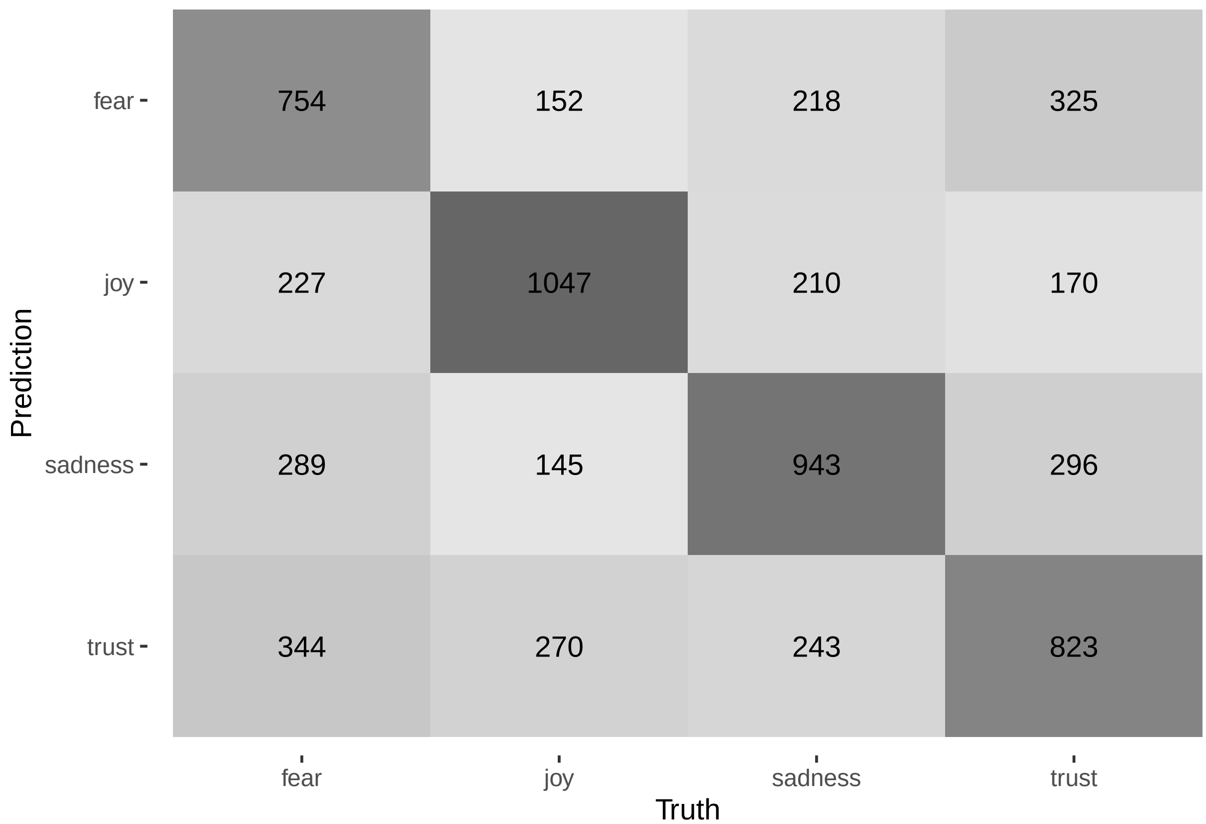
```
log_pred_tf <-
  rf_res_tf %>%
  collect_predictions()
```

Confusion Matrix We can now use our collected predictions to make a confusion matrix

```
log_pred_tf %>%
  conf_mat(y, .pred_class)
```

```
##           Truth
## Prediction fear  joy sadness trust
##    fear     754  152     218   325
##    joy      227 1047     210   170
##    sadness  289  145     943   296
##    trust    344  270     243   823
```

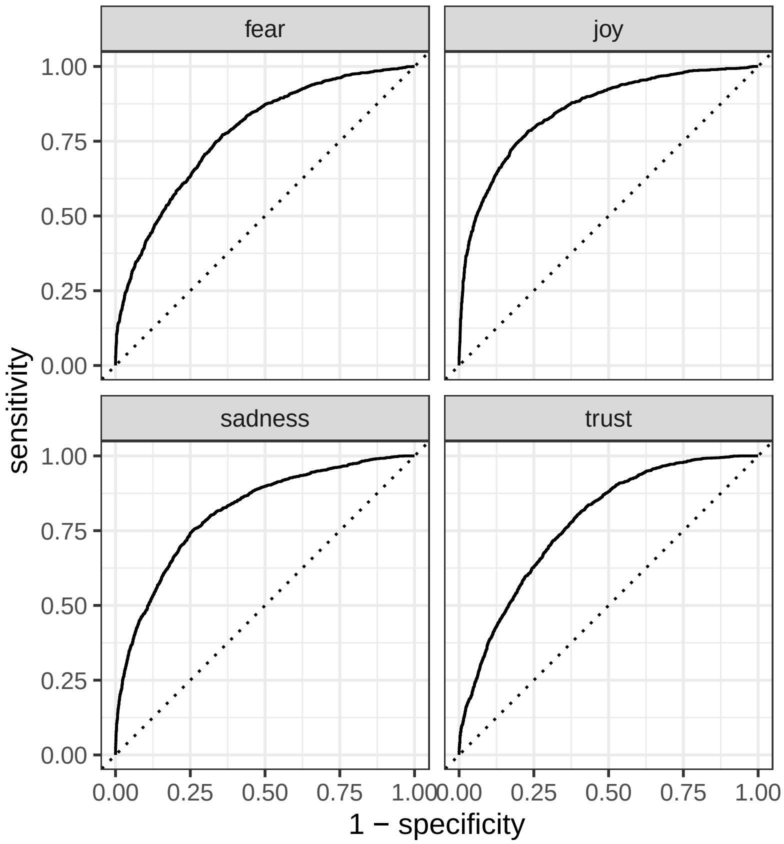
```
log_pred_tf %>%
  conf_mat(y, .pred_class) %>%
  autoplot(type = "heatmap")
```



We can see the model does okay predicting the correct genres.

ROC curve We will now create the ROC curve with 1 - specificity on the x-axis (false positive fraction = FP/(FP+TN)) and sensitivity on the y axis (true positive fraction = TP/(TP+FN)).

```
log_pred_tf %>%
  roc_curve(y, .pred_fear:.pred_trust) %>%
  autoplot()
```



Models on test data

We now want to look at how the two models perform on test data.

Random forest model

```
last_fit_rf <- last_fit(workflow_rf_tf,
                        split = split,
                        metrics = metric_set(
                          recall, precision, f_meas,
                          accuracy, kap,
                          roc_auc, sens, spec)
                        )
```

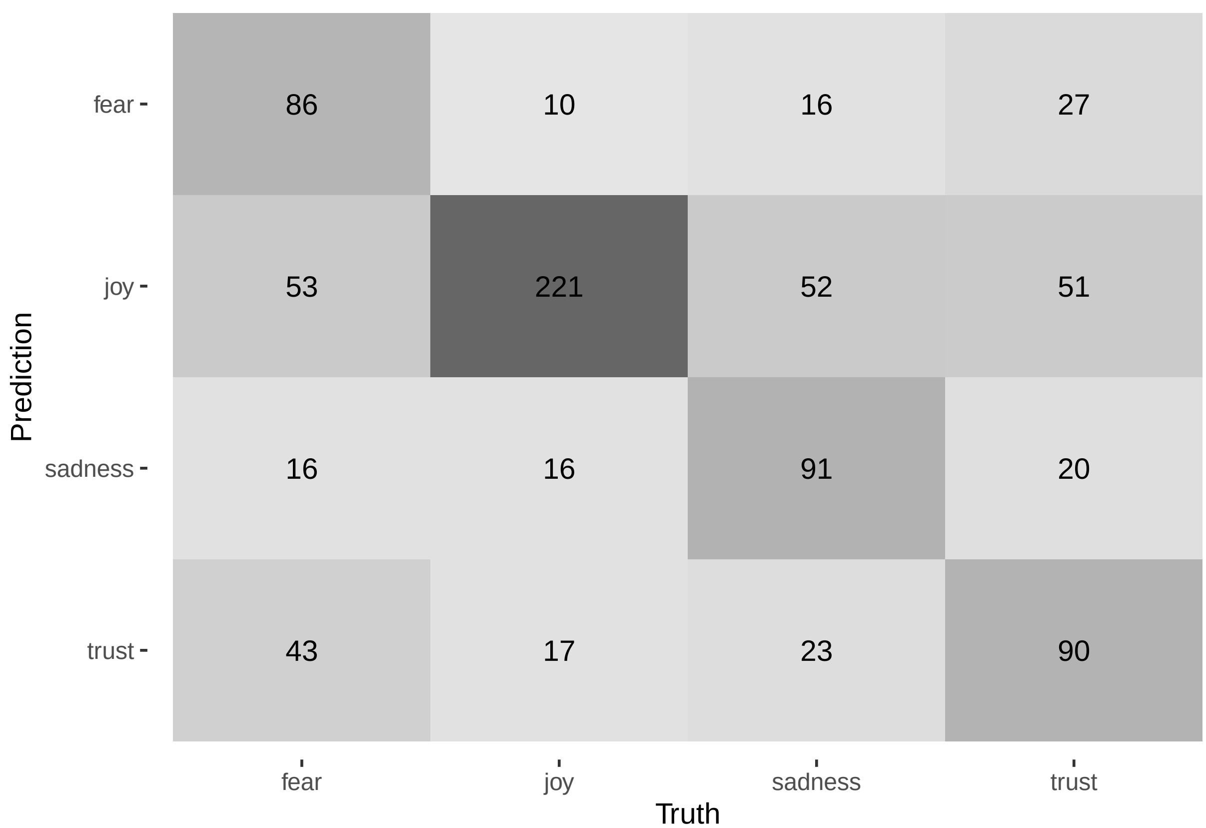
```
last_fit_rf %>%
  collect_metrics()
```

```
## # A tibble: 8 x 4
##   .metric   .estimator .estimate .config
##   <chr>     <chr>          <dbl> <chr>
## 1 recall    macro          0.563 Preprocessor1_Model1
## 2 precision macro          0.590 Preprocessor1_Model1
## 3 f_meas    macro          0.565 Preprocessor1_Model1
## 4 accuracy  multiclass     0.587 Preprocessor1_Model1
## 5 kap       multiclass     0.435 Preprocessor1_Model1
```

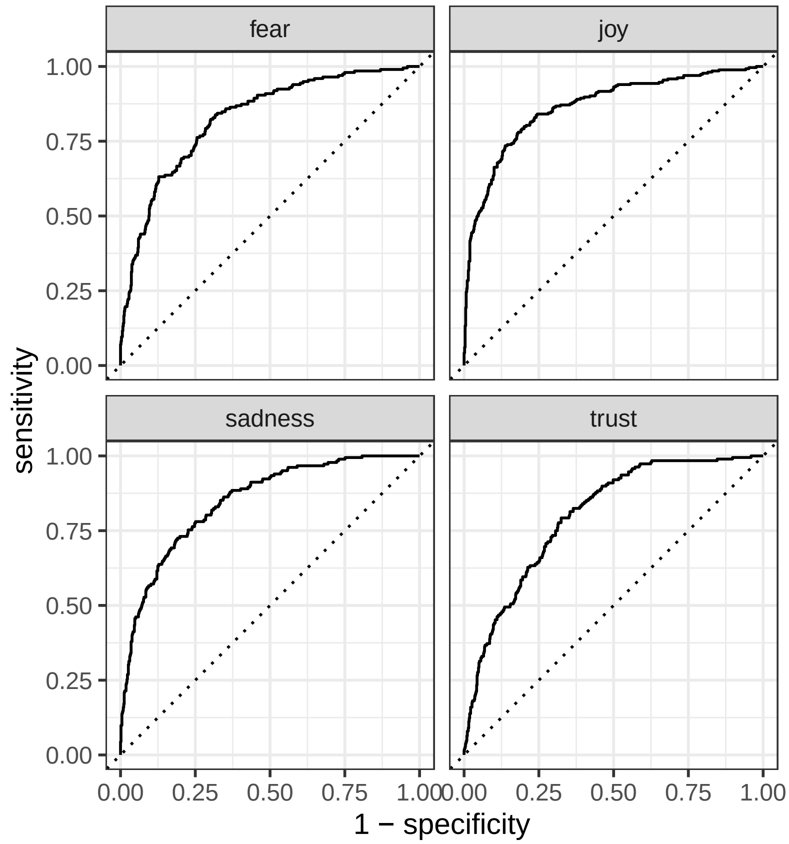
```
## 6 sens      macro          0.563 Preprocessor1_Model1
## 7 spec      macro          0.858 Preprocessor1_Model1
## 8 roc_auc   hand_till      0.831 Preprocessor1_Model1
```

WWe can again make a confusinmatrix on the testdata predictions

```
last_fit_rf %>%
  collect_predictions() %>%
  conf_mat(y, .pred_class) %>%
  autoplot(type = "heatmap")
```




```
last_fit_rf %>%
  collect_predictions() %>%
  roc_curve(y, .pred_fear:.pred_trust) %>%
  autoplot()
```


fear
joy
sadness
trust
sensitivity
1.00
0.75
0.50
0.25
0.00
0.00 0.25 0.50 0.75 1.00
1 − specificity