ML to knit

Andreas Methling

30/11/2021

```r
library(tidyverse)
library(lubridate)
library(magrittr)
library(FactoMineR)
library(factoextra)
library(uwot)
library(GGally)
library(rsample)
library(ggridges)
library(xgboost)
library(recipes)
library(parsnip)
library(glmnet)
library(tidymodels)
library(skimr)
library(VIM)
library(visdat)
library(ggmap)
library(ranger)
library(vip)
library(SnowballC)
library(tokenizers)
library(formatR)
```

```
## Warning: pakke 'formatR' blev bygget under R version 4.1.2
```

Data

```r
library(readr)
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.
```

```
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.
```

Artist data

```
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, Songs)) %>%
  distinct()
```

Data Rock or Pop

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



data_pop_rock=data_genre %>%
  mutate(genre = ifelse(Pop==1 & Rock == 1, "pop/rock", ifelse(Rock==1 & Pop==0, "Rock", ifelse(Rock ==
  select(-c(Pop, Rock))

data_pop_rock_labels= data_pop_rock %>%
  select(name, genre)
```

Data Rock and Pop

```
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)
```

Preprocessing / EDA

First we tokenize the data.

```
library(tidytext)
text_genre_tidy = data_pop_rock %>% unnest_tokens(word, text, token = "words")

head(text_genre_tidy)
```

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

We remove short words and stopwords.

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

We use the hunspell package, which seems to produce the best stemming for our data. Reducing a word to its "root" word.

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

```
## # A tibble: 21,941 x 2
##    stem       n
##    <chr>  <int>
```

```
##  1 love  138187
##  2 time   70143
##  3 baby   62667
##  4 feel   62082
##  5 yeah   59708
##  6 gonna  44342
##  7 wanna  42810
##  8 girl   41029
##  9 day    39207
## 10 heart  39135
## # ... with 21,931 more rows
```

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

We weight the data using tf-idf (Term-frequency Inverse document frequency).

```
# TFIDF weights
text_tf_idf= text_genre_tidy %>%
group_by(name) %>%
  count(word, sort = TRUE) %>%
  ungroup() %>%
  bind_tf_idf(word, name, n) %>%
  arrange(desc(tf_idf))


text_genre_tf_idf = text_tf_idf %>%
  left_join(data_pop_rock_labels)
```

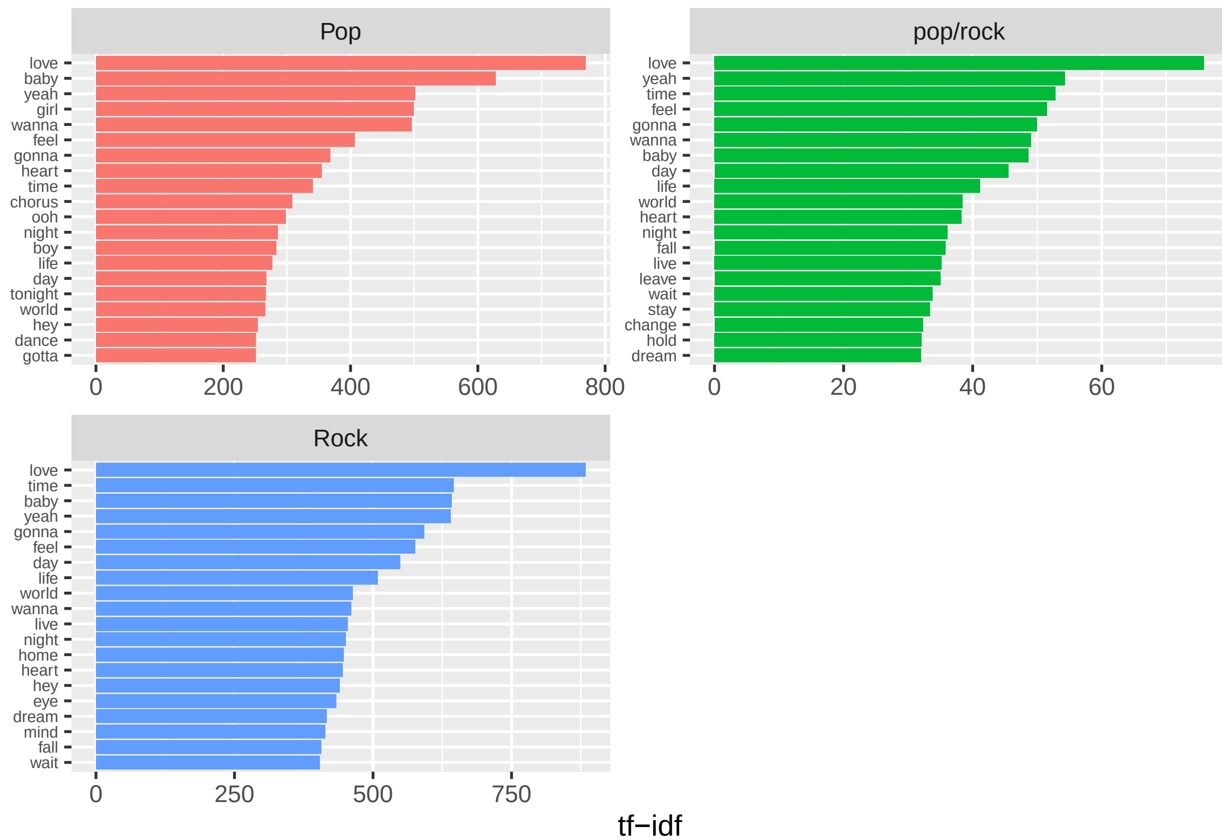
```
## Joining, by = "name"
```

We show the 25 most common words.

```
# TFIDF topwords
text_genre_tidy_rock= text_genre_tf_idf %>%
  filter(genre == "Rock")%>%
count(word, wt = tf_idf, sort = TRUE) %>% #remove
head(25)


text_genre_tidy_rock_pop= text_genre_tf_idf %>%
  filter(genre == "pop/rock")%>%
count(word, wt = tf_idf, sort = TRUE) %>% #remove
head(25)

text_genre_tidy_pop= text_genre_tf_idf %>%
  filter(genre == "Pop")%>%
count(word, wt = tf_idf, sort = TRUE) %>% #remove
head(25)
```

```
labels_words <- text_genre_tf_idf %>%
group_by(genre) %>%
count(word, wt = tf_idf, sort = TRUE, name = "tf_idf") %>%
dplyr::slice(1:20) %>% #slice
ungroup()
```

```
labels_words %>%
mutate(word = reorder_within(word, by = tf_idf, within = genre)) %>% #Pop & Rock
ggplot(aes(x = word, y = tf_idf, fill = genre)) +
geom_col(show.legend = FALSE) +
labs(x = NULL, y = "tf-idf") +
facet_wrap(~genre, ncol = 2, scales = "free") +
coord_flip() +
scale_x_reordered() +
theme(axis.text.y = element_text(size = 6))
```



Machine learning model

Making labels

```
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')
```

We use stemming

```
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"))
```

Bing

```
sentiment_bing= data_top_10000 %>%
  inner_join(get_sentiments("bing")) %>%
  mutate(sentiment= ifelse(sentiment == "positive", 1,0))
```

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

```
sentiment_bing %<>%
  group_by(name) %>%
  summarise(mean= mean(sentiment))%>%
  mutate(label= ifelse(mean>=0.5, 1,0))
```

Afinn

```
sentiment_afinn= data_top_10000 %>%
  inner_join(get_sentiments("afinn"))
```

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

```
sentiment_afinn %<>%
  group_by(name) %>%
  summarise(mean= mean(value))%>%
  mutate(label= ifelse(mean>=0, 1,0))
```

Data

```
data_bing= sentiment_bing %>%
  inner_join(data)%>%
  select(text, label, name)
```

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

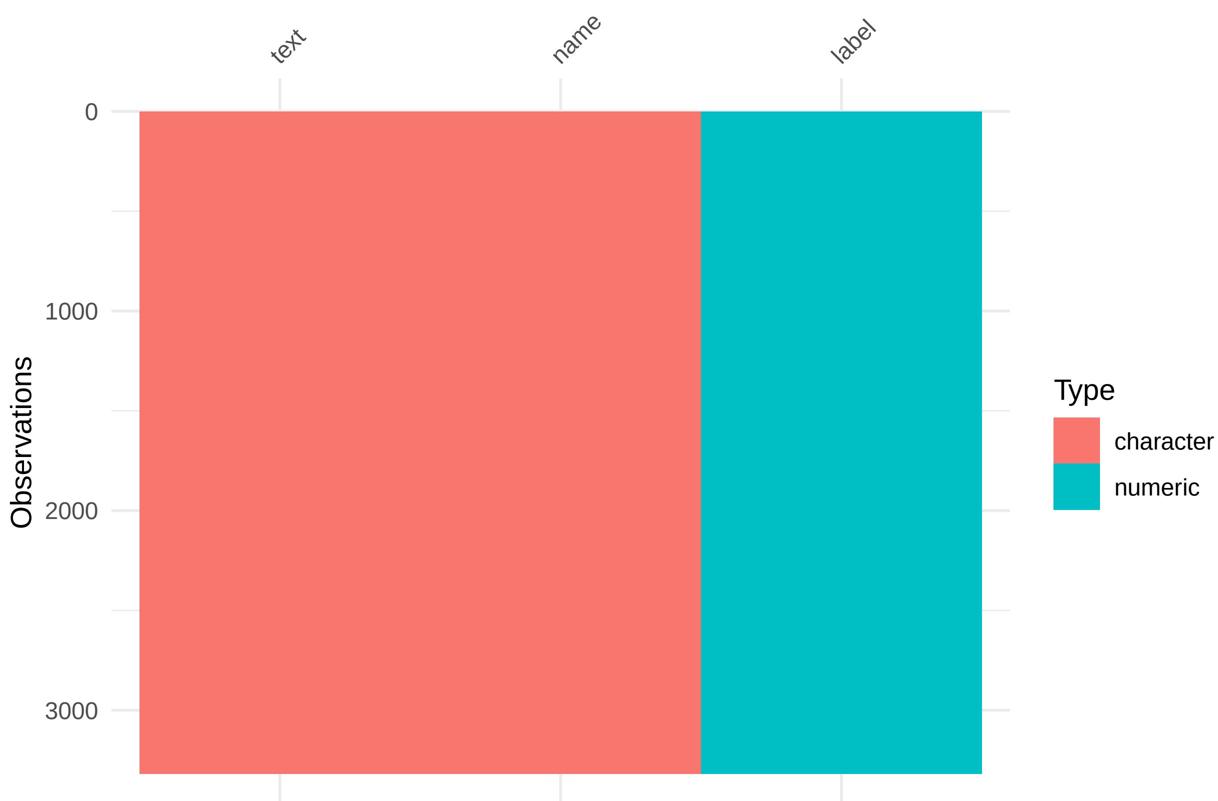
```
data_afinn= sentiment_afinn %>%
  inner_join(data)%>%
  select(text, label, name)
```

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

We now want to create a multiclass supervised machinelearning model to predict the sentiment of a song based on its lyrics.

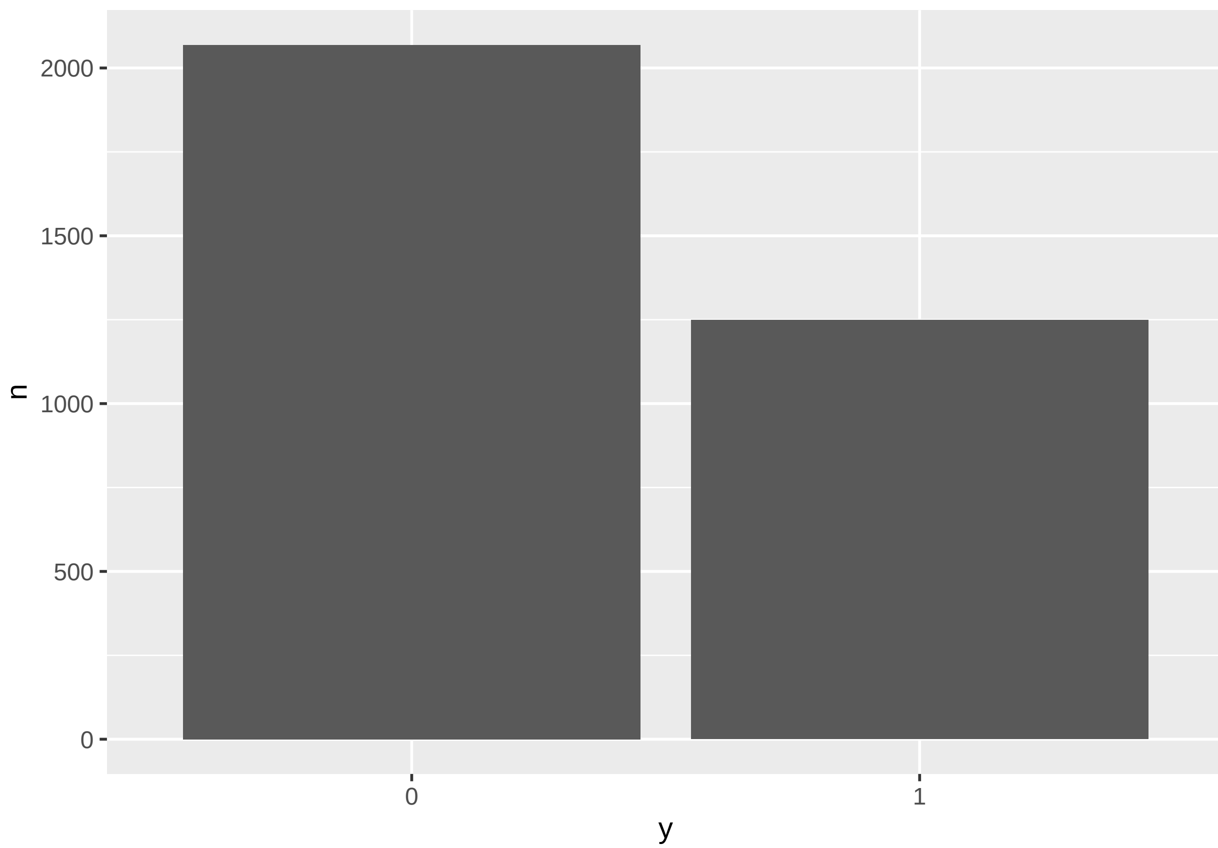
First we look for missing values.

```
library(visdat)
vis_dat(data_bing)
```



We remove NA's and look at the distribution between the two classes.

```
data_bing %<>%
  drop_na() %>%
  rename(y = label) %>%
  select(-name)
data_bing$y <- as.factor(data_bing$y)
data_bing %>%
  count(y) %>%
  ggplot(aes(x = y, y = n)) +
  geom_col()
```



We can see that negative sentiment is much more represented than positive sentiment, so we have to do some down or upsampling.

We will create three different receipes: one using embedding, one using tf-idf and one using Hash.

So we load the embeddings using the "textdata" package.

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

We create a training and test dataset using strata=y to get the same ratio between the classes in both the training and test dataset.

```
library(rsample)
set.seed(19)
tidy_split <- initial_split(data_bing, strata = y)
train_data <- training(tidy_split)
test_data <- testing(tidy_split)
```

We use downsampling only on the training data to better fit the model

```
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: 2 x 2
##   y         n
##   <fct> <int>
## 1 0       936
## 2 1       936
```

And can now see that the classes are evenly distributed.

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:

We set some of the parameters for tuning.

Logistic model

```r
#model_lg <- logistic_reg(mode = 'classification', penalty = tune(), mixture = 0.5) %>%
  #set_engine('glm', family = binomial)
model_lg <- logistic_reg(mode = 'classification') %>%
  set_engine('glm', family = binomial)
```

KNN model

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

Random Forrest

```r
model_rf <-
  rand_forest(trees = NULL, mtry = NULL, min_n = NULL) %>%
  set_engine("ranger", importance = "impurity") %>%
  set_mode("classification")
```

Decision tree

```r
model_dt <- decision_tree(mode = 'classification',
                          cost_complexity = tune(),
                          tree_depth = tune(),
                          min_n = tune()
                          ) %>%
  set_engine('rpart')
```

Workflow

We create workflows for each recipe.

tf_idf

```r
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)
workflow_dt_tf <- workflow_general_tf %>%
  add_model(model_dt)
```

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)
workflow_dt_emb <- workflow_general_emb %>%
  add_model(model_dt)
```

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)
workflow_dt_hash <- workflow_general_hash %>%
  add_model(model_dt)
```

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)
logistic_grid <- 5
# knn_grid <- grid_regular(parameters(model_knn), levels = 5, filter = c(neighbors > 1))
knn_grid <- 5
dt_grid <- 5
rf_grid <- 5
```

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

```
library(text2vec)
```

```
##
## Vedhæfter pakke: 'text2vec'

## Det følgende objekt er maskeret fra 'package:infer':
##
##     fit

## Det følgende objekt er maskeret fra 'package:parsnip':
##
##     fit
```

```
# Tune hash models

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

#rf_hash_res <- tune_grid(
  #model_rf,
  #hash_rec,
  #grid = rf_grid,
  #control = model_control,
  #metrics = model_metrics,
  #resamples = k_folds_data
#)

dt_hash_res <- tune_grid(
  model_dt,
  hash_rec,
  grid = dt_grid,
  control = model_control,
  metrics = model_metrics,
  resamples = k_folds_data
)
```

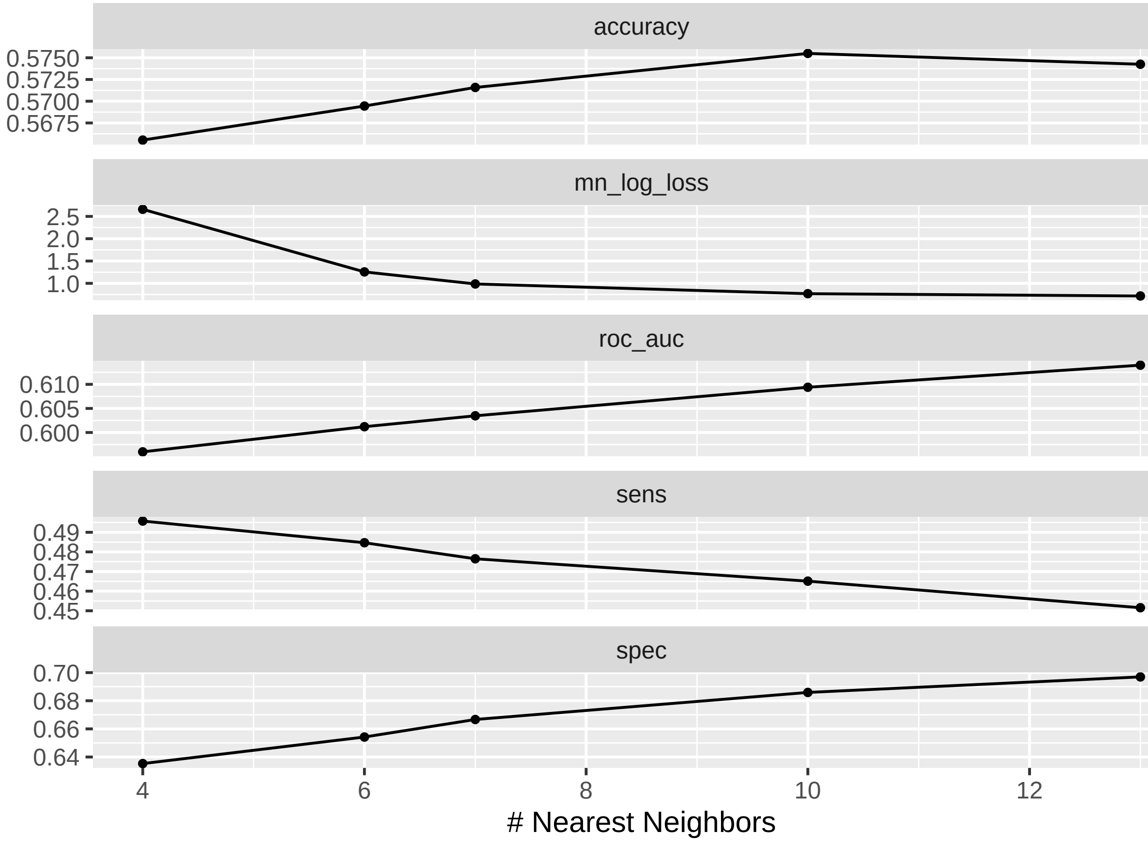
```r
# Tune embed models

knn_embed_res <- tune_grid(
  model_knn,
  embeddings_rec,
  grid = knn_grid,
  control = model_control,
  metrics = model_metrics,
  resamples = k_folds_data
)
#rf_embed_res <- tune_grid(
  #model_rf,
  #embeddings_rec,
  #grid = rf_grid,
  #control = model_control,
  #metrics = model_metrics,
  #resamples = k_folds_data
#)
dt_embed_res <- tune_grid(
  model_dt,
  embeddings_rec,
  grid = dt_grid,
  control = model_control,
  metrics = model_metrics,
  resamples = k_folds_data
)
```

```r
# Tune tf-idf models
knn_tf_res <- tune_grid(
  model_knn,
  tf_idf_rec,
  grid = knn_grid,
  control = model_control,
  metrics = model_metrics,
  resamples = k_folds_data
)
#rf_tf_res <- tune_grid(
  #model_rf,
  #tf_idf_rec,
  #grid = rf_grid,
  #control = model_control,
  #metrics = model_metrics,
  #resamples = k_folds_data
#)
dt_tf_res <- tune_grid(
  model_dt,
  tf_idf_rec,
  grid = dt_grid,
  control = model_control,
  metrics = model_metrics,
  resamples = k_folds_data
)
```

Best parameters

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

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

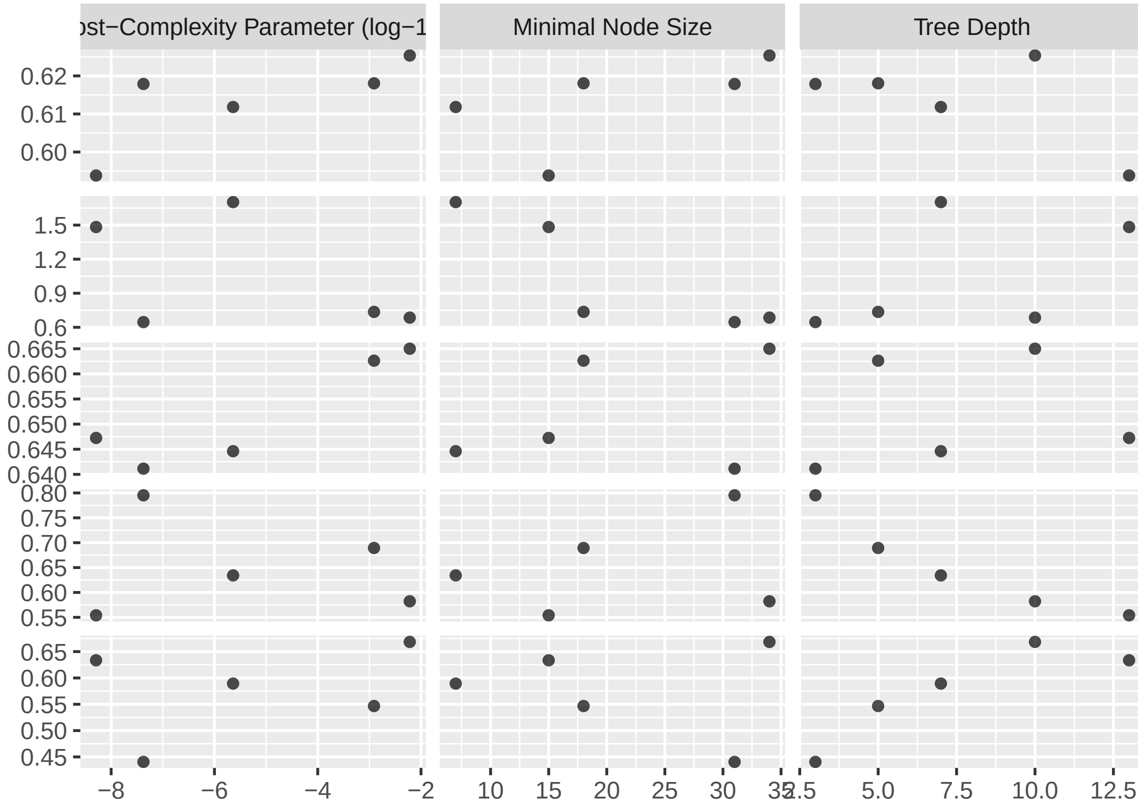


knn_embed_res

```
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        10 Preprocessor1_Model4
```

```
dt_hash_res %>%
    autoplot()
```

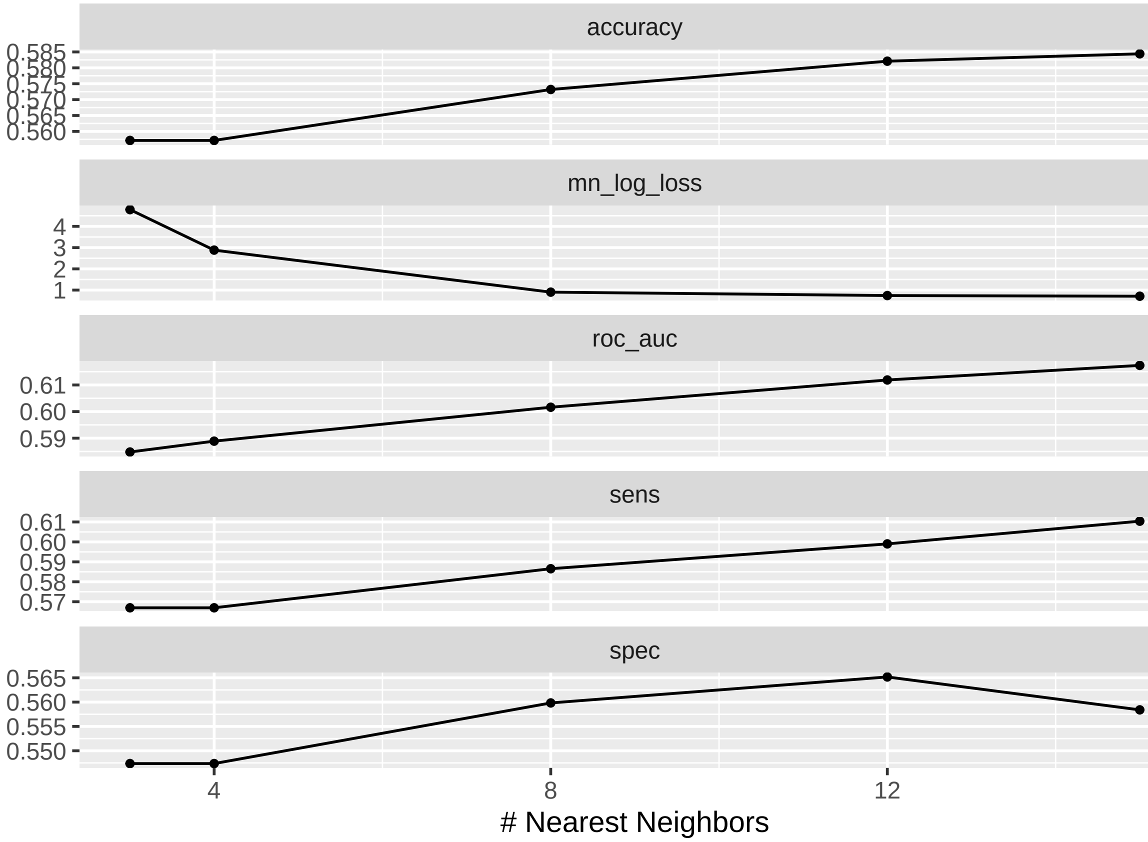


decision tree hash

```
best_param_dt_hash_res <- dt_hash_res %>% select_best(metric = 'accuracy')
best_param_dt_hash_res
```

```
## # A tibble: 1 x 4
##   cost_complexity tree_depth min_n .config
##             <dbl>      <int> <int> <chr>
## 1         0.00606         10    34 Preprocessor1_Model5
```

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

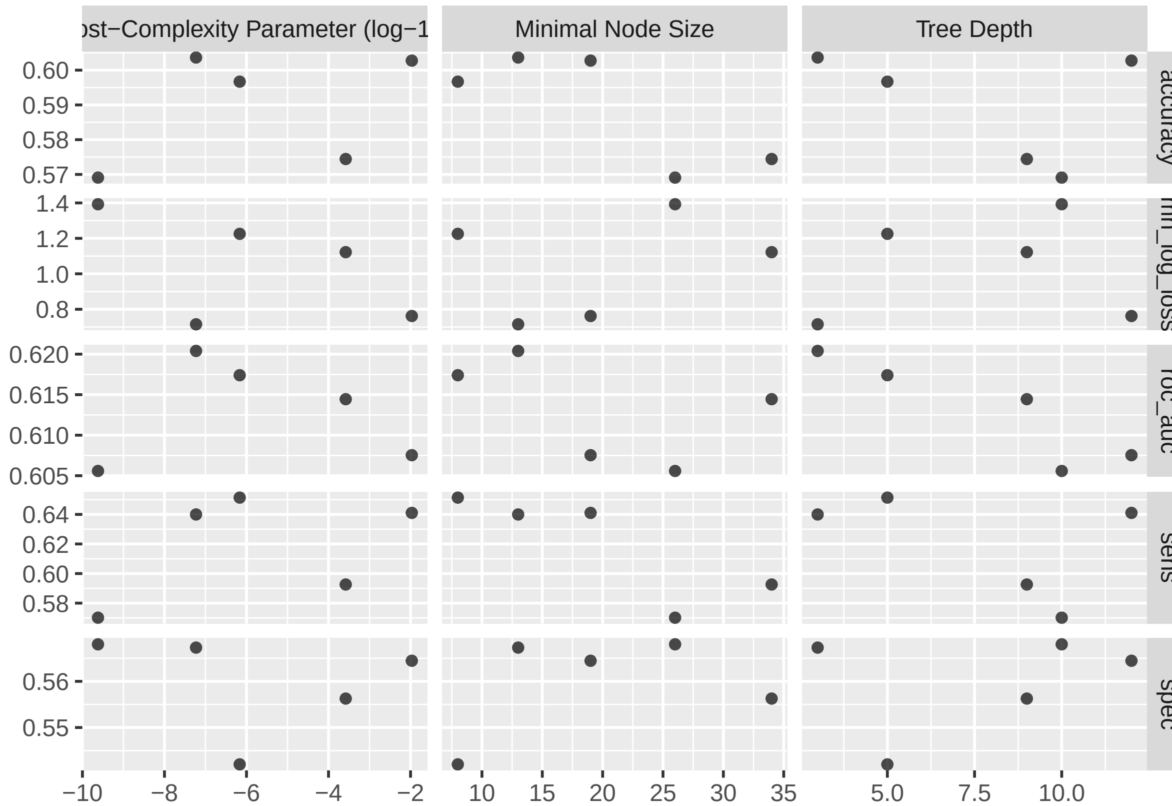


knn_embed_res

```
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_Model5
```

```
dt_embed_res %>%
    autoplot()
```

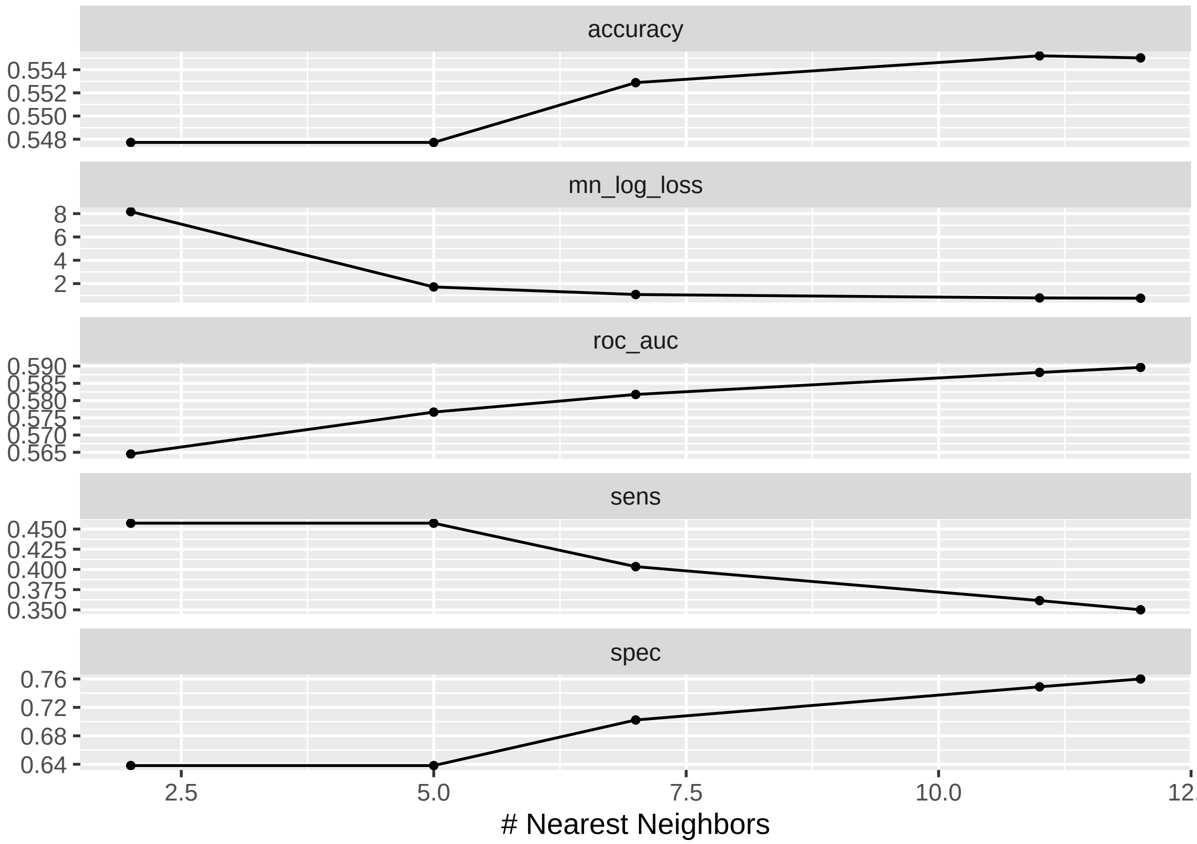


dt_embed_res

```
best_param_dt_embed_res <- dt_embed_res %>% select_best(metric = 'accuracy')
best_param_dt_embed_res
```

```
## # A tibble: 1 x 4
##   cost_complexity tree_depth min_n .config
##             <dbl>      <int> <int> <chr>
## 1    0.0000000589          3    13 Preprocessor1_Model4
```

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

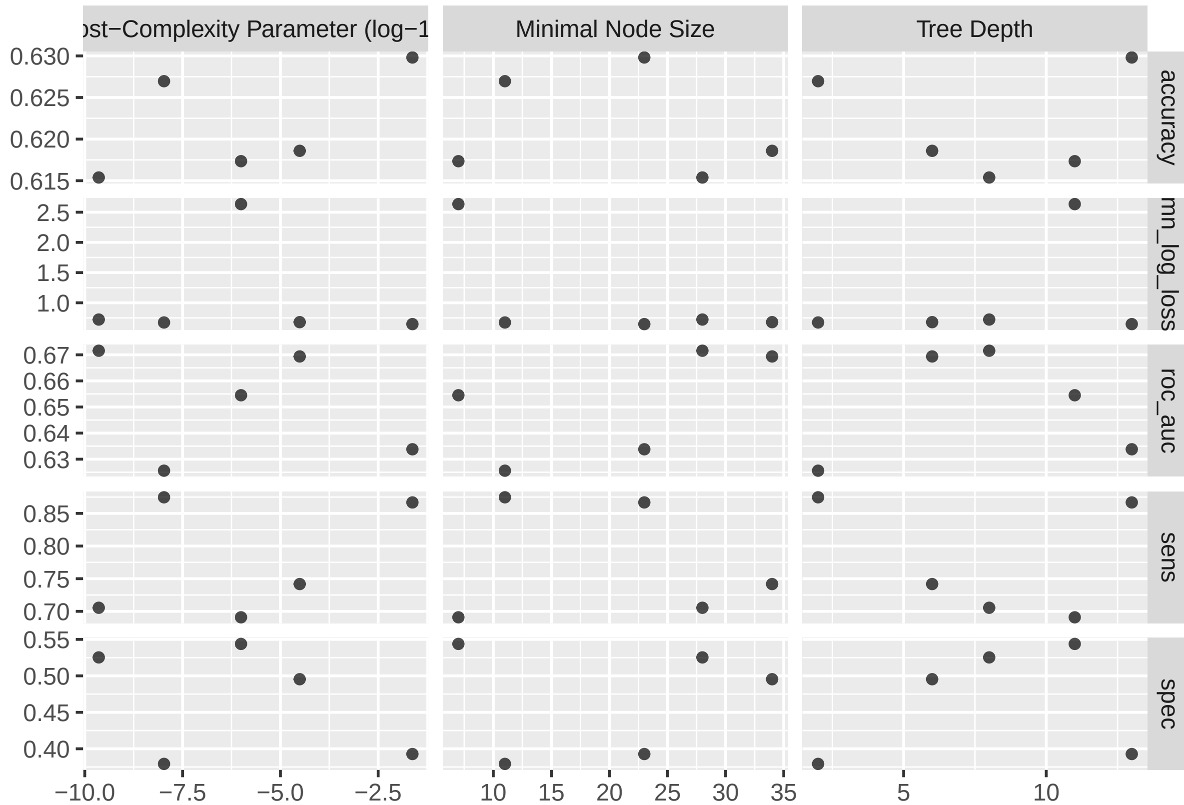


knn_tf_res

```
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        11 Preprocessor1_Model4
```

```
dt_tf_res %>%
    autoplot()
```



dt_tf_res

```
best_param_dt_tf_res <- dt_tf_res %>% select_best(metric = 'accuracy')
best_param_dt_tf_res
```

```
## # A tibble: 1 x 4
##   cost_complexity tree_depth min_n .config
##             <dbl>      <int> <int> <chr>
## 1          0.0238         13    23 Preprocessor1_Model1
```

Finalize workflows

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

Hash

```
workflow_final_knn_hash <- workflow_knn_hash %>%
  finalize_workflow(parameters = best_param_knn_hash_res)
#workflow_final_rf_hash <- workflow_rf_hash %>%
 # finalize_workflow(parameters = best_param_rf_hash_res)
workflow_final_dt_hash <- workflow_dt_hash %>%
  finalize_workflow(parameters = best_param_dt_hash_res)
```

Tf-idf

```
workflow_final_knn_tf <- workflow_knn_tf %>%
  finalize_workflow(parameters = best_param_knn_tf_res)
#workflow_final_rf_tf <- workflow_rf_tf %>%
 # finalize_workflow(parameters = best_param_rf_tf_res)
workflow_final_dt_tf <- workflow_dt_tf %>%
  finalize_workflow(parameters = best_param_dt_tf_res)
```

Embedings

```
workflow_final_knn_emb <- workflow_knn_emb %>%
  finalize_workflow(parameters = best_param_knn_embed_res)
#workflow_final_rf_emb <- workflow_rf_emb %>%
 # finalize_workflow(parameters = best_param_rf_embed_res)
workflow_final_dt_emb <- workflow_dt_emb %>%
  finalize_workflow(parameters = best_param_dt_embed_res)
```

Evaluate models

here we us the resampled data to evaluate the models.

Logistic regression

```


log_res_hash <-
  workflow_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)
    )
```

hash

```
## ! Fold1, Repeat1: preprocessor 1/1, model 1/1: glm.fit: fitted probabilities numerically 0...

## ! Fold2, Repeat1: preprocessor 1/1, model 1/1: glm.fit: fitted probabilities numerically 0...

## ! Fold1, Repeat2: preprocessor 1/1, model 1/1: glm.fit: fitted probabilities numerically 0...

## ! Fold2, Repeat2: preprocessor 1/1, model 1/1: glm.fit: fitted probabilities numerically 0...
```

```
## ! Fold2, Repeat3: preprocessor 1/1, model 1/1: glm.fit: fitted probabilities numerically 0...

## ! Fold3, Repeat3: preprocessor 1/1, model 1/1: glm.fit: fitted probabilities numerically 0...
```

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

```
## # A tibble: 8 x 6
##   .metric   .estimator  mean     n std_err .config
##   <chr>     <chr>      <dbl> <int>   <dbl> <chr>
## 1 accuracy  binary     0.661     9 0.0103  Preprocessor1_Model1
## 2 f_meas    binary     0.670     9 0.0106  Preprocessor1_Model1
## 3 kap       binary     0.322     9 0.0205  Preprocessor1_Model1
## 4 precision binary     0.652     9 0.00919 Preprocessor1_Model1
## 5 recall    binary     0.689     9 0.0131  Preprocessor1_Model1
## 6 roc_auc   binary     0.719     9 0.0107  Preprocessor1_Model1
## 7 sens      binary     0.689     9 0.0131  Preprocessor1_Model1
## 8 spec      binary     0.632     9 0.00988 Preprocessor1_Model1
```

```
log_res_tf <-
  workflow_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)
    )
```

Tf_idf

```
## ! Fold1, Repeat1: preprocessor 1/1, model 1/1: glm.fit: algorithm did not converge, glm.fi...

## ! Fold1, Repeat1: preprocessor 1/1, model 1/1 (predictions): prediction from a rank-defici...

## ! Fold2, Repeat1: preprocessor 1/1, model 1/1: glm.fit: algorithm did not converge, glm.fi...

## ! Fold2, Repeat1: preprocessor 1/1, model 1/1 (predictions): prediction from a rank-defici...

## ! Fold3, Repeat1: preprocessor 1/1, model 1/1: glm.fit: algorithm did not converge, glm.fi...

## ! Fold3, Repeat1: preprocessor 1/1, model 1/1 (predictions): prediction from a rank-defici...

## ! Fold1, Repeat2: preprocessor 1/1, model 1/1: glm.fit: algorithm did not converge, glm.fi...

## ! Fold1, Repeat2: preprocessor 1/1, model 1/1 (predictions): prediction from a rank-defici...
```

```
## ! Fold2, Repeat2: preprocessor 1/1, model 1/1: glm.fit: algorithm did not converge, glm.fi...

## ! Fold2, Repeat2: preprocessor 1/1, model 1/1 (predictions): prediction from a rank-defici...

## ! Fold3, Repeat2: preprocessor 1/1, model 1/1: glm.fit: algorithm did not converge, glm.fi...

## ! Fold3, Repeat2: preprocessor 1/1, model 1/1 (predictions): prediction from a rank-defici...

## ! Fold1, Repeat3: preprocessor 1/1, model 1/1: glm.fit: algorithm did not converge, glm.fi...

## ! Fold1, Repeat3: preprocessor 1/1, model 1/1 (predictions): prediction from a rank-defici...

## ! Fold2, Repeat3: preprocessor 1/1, model 1/1: glm.fit: algorithm did not converge, glm.fi...

## ! Fold2, Repeat3: preprocessor 1/1, model 1/1 (predictions): prediction from a rank-defici...

## ! Fold3, Repeat3: preprocessor 1/1, model 1/1: glm.fit: algorithm did not converge, glm.fi...

## ! Fold3, Repeat3: preprocessor 1/1, model 1/1 (predictions): prediction from a rank-defici...
```

```r
log_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  binary     0.616     9 0.00435 Preprocessor1_Model1
## 2 f_meas    binary     0.617     9 0.00405 Preprocessor1_Model1
## 3 kap       binary     0.233     9 0.00870 Preprocessor1_Model1
## 4 precision binary     0.617     9 0.00496 Preprocessor1_Model1
## 5 recall    binary     0.617     9 0.00585 Preprocessor1_Model1
## 6 roc_auc   binary     0.640     9 0.00591 Preprocessor1_Model1
## 7 sens      binary     0.617     9 0.00585 Preprocessor1_Model1
## 8 spec      binary     0.616     9 0.00849 Preprocessor1_Model1
```

```r
log_res_emb <-
  workflow_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  binary     0.665     9 0.00561 Preprocessor1_Model1
## 2 f_meas    binary     0.665     9 0.00631 Preprocessor1_Model1
## 3 kap       binary     0.329     9 0.0112  Preprocessor1_Model1
## 4 precision binary     0.664     9 0.00599 Preprocessor1_Model1
## 5 recall    binary     0.667     9 0.00942 Preprocessor1_Model1
## 6 roc_auc   binary     0.727     9 0.00559 Preprocessor1_Model1
## 7 sens      binary     0.667     9 0.00942 Preprocessor1_Model1
## 8 spec      binary     0.662     9 0.00875 Preprocessor1_Model1
```

KNN model

```r
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  binary     0.575     9 0.00609 Preprocessor1_Model1
## 2 f_meas    binary     0.523     9 0.00739 Preprocessor1_Model1
## 3 kap       binary     0.151     9 0.0122  Preprocessor1_Model1
## 4 precision binary     0.597     9 0.00800 Preprocessor1_Model1
## 5 recall    binary     0.465     9 0.00835 Preprocessor1_Model1
## 6 roc_auc   binary     0.609     9 0.00501 Preprocessor1_Model1
## 7 sens      binary     0.465     9 0.00835 Preprocessor1_Model1
## 8 spec      binary     0.686     9 0.00865 Preprocessor1_Model1
```

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

```r
      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  binary     0.555     9 0.00589 Preprocessor1_Model1
## 2 f_meas    binary     0.441     9 0.0240  Preprocessor1_Model1
## 3 kap       binary     0.110     9 0.0118  Preprocessor1_Model1
## 4 precision binary     0.595     9 0.0115  Preprocessor1_Model1
## 5 recall    binary     0.361     9 0.0326  Preprocessor1_Model1
## 6 roc_auc   binary     0.588     9 0.00680 Preprocessor1_Model1
## 7 sens      binary     0.361     9 0.0326  Preprocessor1_Model1
## 8 spec      binary     0.749     9 0.0299  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  binary     0.584     9 0.00882 Preprocessor1_Model1
## 2 f_meas    binary     0.595     9 0.00591 Preprocessor1_Model1
## 3 kap       binary     0.169     9 0.0176  Preprocessor1_Model1
## 4 precision binary     0.581     9 0.00931 Preprocessor1_Model1
## 5 recall    binary     0.610     9 0.00478 Preprocessor1_Model1
## 6 roc_auc   binary     0.617     9 0.00704 Preprocessor1_Model1
## 7 sens      binary     0.610     9 0.00478 Preprocessor1_Model1
## 8 spec      binary     0.558     9 0.0170  Preprocessor1_Model1
```

Random forest model

```r
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  binary     0.687     9 0.00660 Preprocessor1_Model1
## 2 f_meas    binary     0.706     9 0.00742 Preprocessor1_Model1
## 3 kap       binary     0.374     9 0.0132  Preprocessor1_Model1
## 4 precision binary     0.666     9 0.00513 Preprocessor1_Model1
## 5 recall    binary     0.751     9 0.0124  Preprocessor1_Model1
## 6 roc_auc   binary     0.758     9 0.00913 Preprocessor1_Model1
## 7 sens      binary     0.751     9 0.0124  Preprocessor1_Model1
## 8 spec      binary     0.623     9 0.00733 Preprocessor1_Model1
```

```r
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  binary     0.733     9 0.00700 Preprocessor1_Model1
## 2 f_meas    binary     0.744     9 0.00740 Preprocessor1_Model1
## 3 kap       binary     0.467     9 0.0140  Preprocessor1_Model1
```

```
## 4 precision binary     0.715     9 0.00595 Preprocessor1_Model1
## 5 recall    binary     0.775     9 0.0105  Preprocessor1_Model1
## 6 roc_auc   binary     0.818     9 0.00599 Preprocessor1_Model1
## 7 sens      binary     0.775     9 0.0105  Preprocessor1_Model1
## 8 spec      binary     0.691     9 0.00692 Preprocessor1_Model1
```

```
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  binary     0.653     9 0.00658 Preprocessor1_Model1
## 2 f_meas    binary     0.657     9 0.00554 Preprocessor1_Model1
## 3 kap       binary     0.306     9 0.0132  Preprocessor1_Model1
## 4 precision binary     0.649     9 0.00796 Preprocessor1_Model1
## 5 recall    binary     0.666     9 0.00597 Preprocessor1_Model1
## 6 roc_auc   binary     0.705     9 0.00485 Preprocessor1_Model1
## 7 sens      binary     0.666     9 0.00597 Preprocessor1_Model1
## 8 spec      binary     0.640     9 0.0117  Preprocessor1_Model1
```

Decision tree

```
dt_res_hash <-
  workflow_final_dt_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)
    )
dt_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  binary     0.625     9 0.00965 Preprocessor1_Model1
## 2 f_meas    binary     0.607     9 0.0152  Preprocessor1_Model1
## 3 kap       binary     0.251     9 0.0193  Preprocessor1_Model1
## 4 precision binary     0.636     9 0.00802 Preprocessor1_Model1
## 5 recall    binary     0.582     9 0.0227  Preprocessor1_Model1
## 6 roc_auc   binary     0.665     9 0.00712 Preprocessor1_Model1
## 7 sens      binary     0.582     9 0.0227  Preprocessor1_Model1
## 8 spec      binary     0.668     9 0.0106  Preprocessor1_Model1
```

```
dt_res_tf <-
  workflow_final_dt_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)
    )
dt_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  binary     0.630     9 0.00400 Preprocessor1_Model1
## 2 f_meas    binary     0.700     9 0.00431 Preprocessor1_Model1
## 3 kap       binary     0.260     9 0.00799 Preprocessor1_Model1
## 4 precision binary     0.589     9 0.00416 Preprocessor1_Model1
## 5 recall    binary     0.867     9 0.0166  Preprocessor1_Model1
## 6 roc_auc   binary     0.634     9 0.00530 Preprocessor1_Model1
## 7 sens      binary     0.867     9 0.0166  Preprocessor1_Model1
## 8 spec      binary     0.393     9 0.0191  Preprocessor1_Model1
```

```
dt_res_emb <-
  workflow_final_dt_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)
    )
dt_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  binary     0.604     9 0.00355 Preprocessor1_Model1
## 2 f_meas    binary     0.612     9 0.0181  Preprocessor1_Model1
## 3 kap       binary     0.207     9 0.00711 Preprocessor1_Model1
## 4 precision binary     0.600     9 0.00659 Preprocessor1_Model1
## 5 recall    binary     0.640     9 0.0407  Preprocessor1_Model1
## 6 roc_auc   binary     0.620     9 0.00714 Preprocessor1_Model1
## 7 sens      binary     0.640     9 0.0407  Preprocessor1_Model1
## 8 spec      binary     0.567     9 0.0357  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.

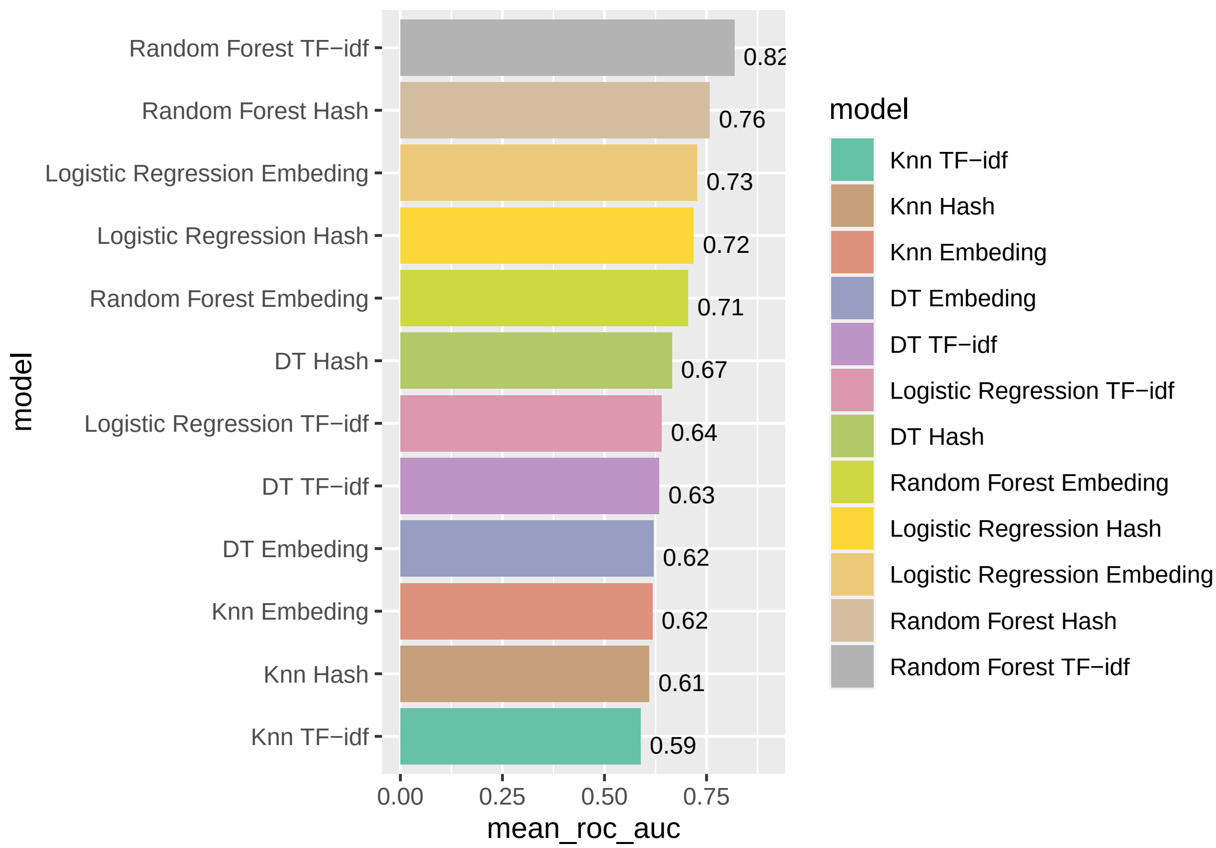
```
log_metrics_tf <-
  log_res_tf %>%
  collect_metrics(summarise = TRUE) %>%
  mutate(model = "Logistic Regression TF-idf")
log_metrics_emb <-
  log_res_emb %>%
  collect_metrics(summarise = TRUE) %>%
  mutate(model = "Logistic Regression Embeding")
log_metrics_hash <-
  log_res_hash %>%
  collect_metrics(summarise = 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")
dt_metrics_tf <-
  dt_res_tf %>%
  collect_metrics(summarise = TRUE) %>%
  mutate(model = "DT TF-idf")
dt_metrics_emb <-
  dt_res_emb %>%
  collect_metrics(summarise = TRUE) %>%
  mutate(model = "DT Embeding")
dt_metrics_hash <-
  dt_res_hash %>%
  collect_metrics(summarise = TRUE) %>%
  mutate(model = "DT Hash")
```

```
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,
                          dt_metrics_tf,
                          dt_metrics_emb,
                          dt_metrics_hash
                           )
model_comp <-
  model_compare %>%
  select(model, .metric, mean, std_err) %>%
  pivot_wider(names_from = .metric, values_from = c(mean, std_err))
library(RColorBrewer)
nb.cols <- 12
mycolors <- colorRampPalette(brewer.pal(8, "Set2"))(nb.cols)
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_manual(values = mycolors) +
  #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_manual(values = mycolors) +
  #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 random forest using hash.

So we only continue with the two best ones.

Random forest model with TF IDF

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  binary     0.733     9 0.00700 Preprocessor1_Model1
## 2 f_meas    binary     0.744     9 0.00740 Preprocessor1_Model1
## 3 kap       binary     0.467     9 0.0140  Preprocessor1_Model1
## 4 precision binary     0.715     9 0.00595 Preprocessor1_Model1
## 5 recall    binary     0.775     9 0.0105  Preprocessor1_Model1
## 6 roc_auc   binary     0.818     9 0.00599 Preprocessor1_Model1
## 7 sens      binary     0.775     9 0.0105  Preprocessor1_Model1
## 8 spec      binary     0.691     9 0.00692 Preprocessor1_Model1
```

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

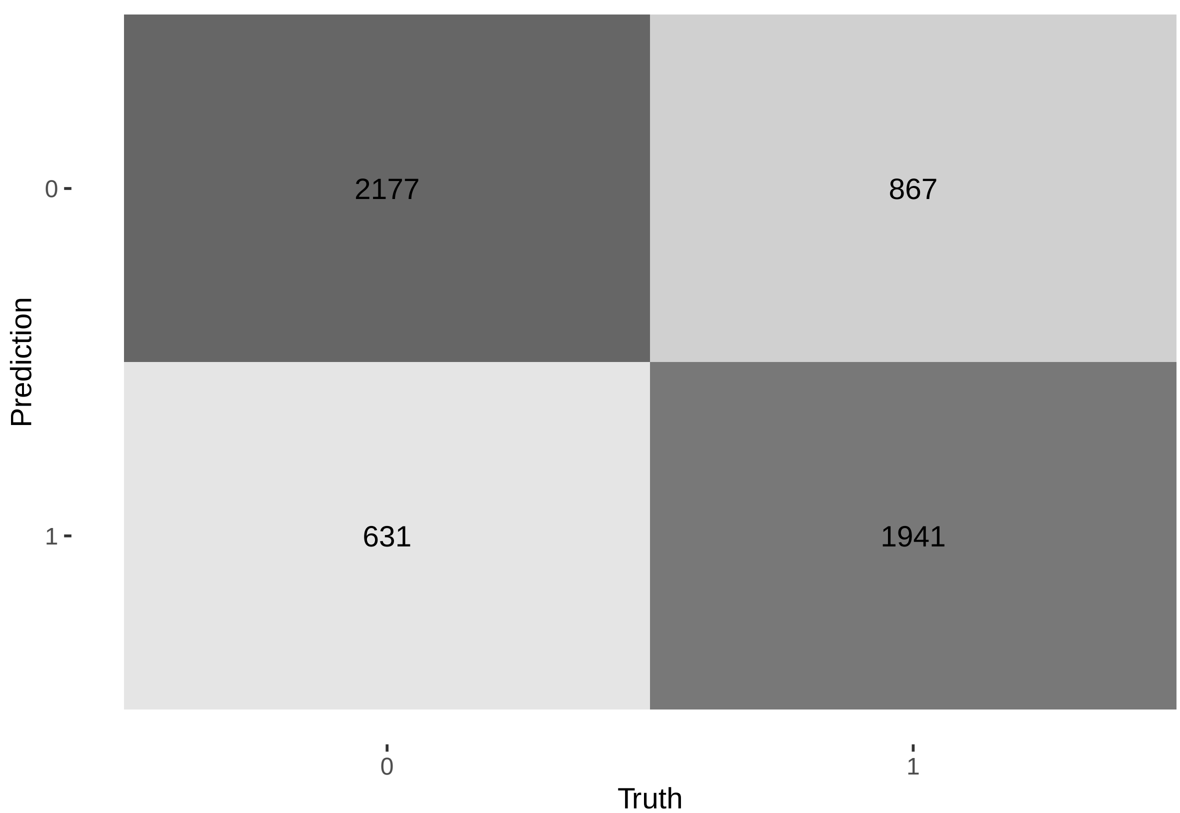
```
rf_pred_tf <-
  rf_res_tf %>%
  collect_predictions()
```

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

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

```
##           Truth
## Prediction    0    1
##          0 2177  867
##          1  631 1941
```

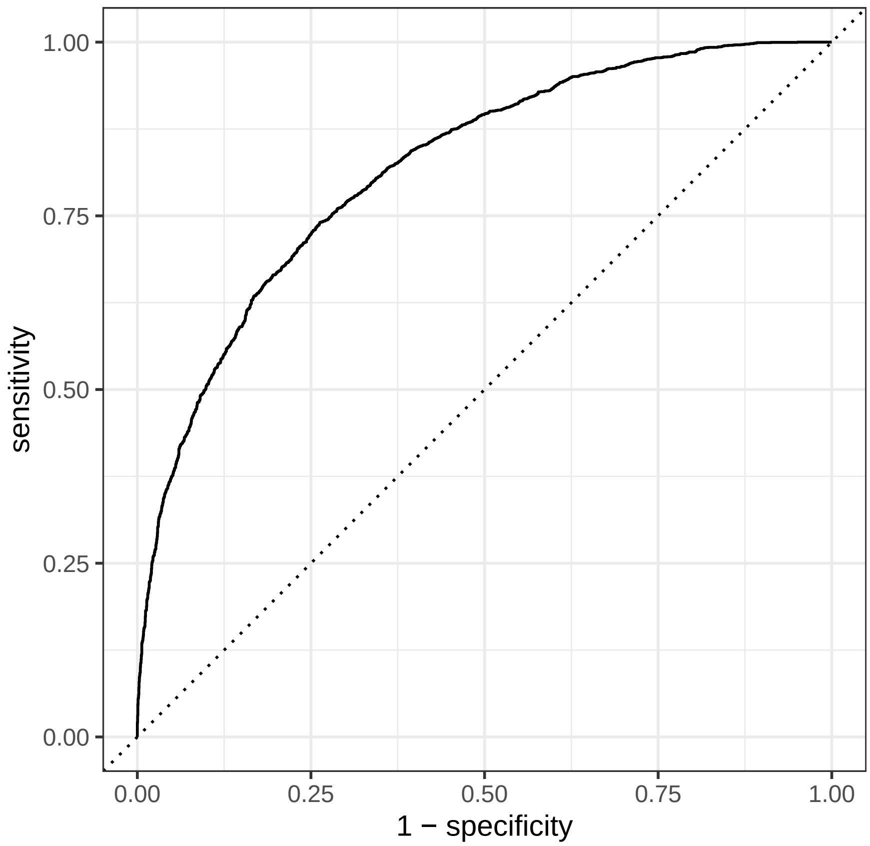
```
rf_pred_tf %>%
  conf_mat(y, .pred_class) %>%
  autoplot(type = "heatmap")
```



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

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)).

```r
rf_pred_tf %>%
  roc_curve(y,.pred_0) %>%
  autoplot()
```



Random forest model hash

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

```r
rf_pred_hash <-
  rf_res_hash %>%
  collect_predictions()
```

Performance metrics Show average performance over all folds (note that we use rf_res):

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

```
## # A tibble: 8 x 6
##   .metric   .estimator  mean     n std_err .config
##   <chr>     <chr>      <dbl> <int>   <dbl> <chr>
## 1 accuracy  binary     0.687     9 0.00660 Preprocessor1_Model1
## 2 f_meas    binary     0.706     9 0.00742 Preprocessor1_Model1
## 3 kap       binary     0.374     9 0.0132  Preprocessor1_Model1
```

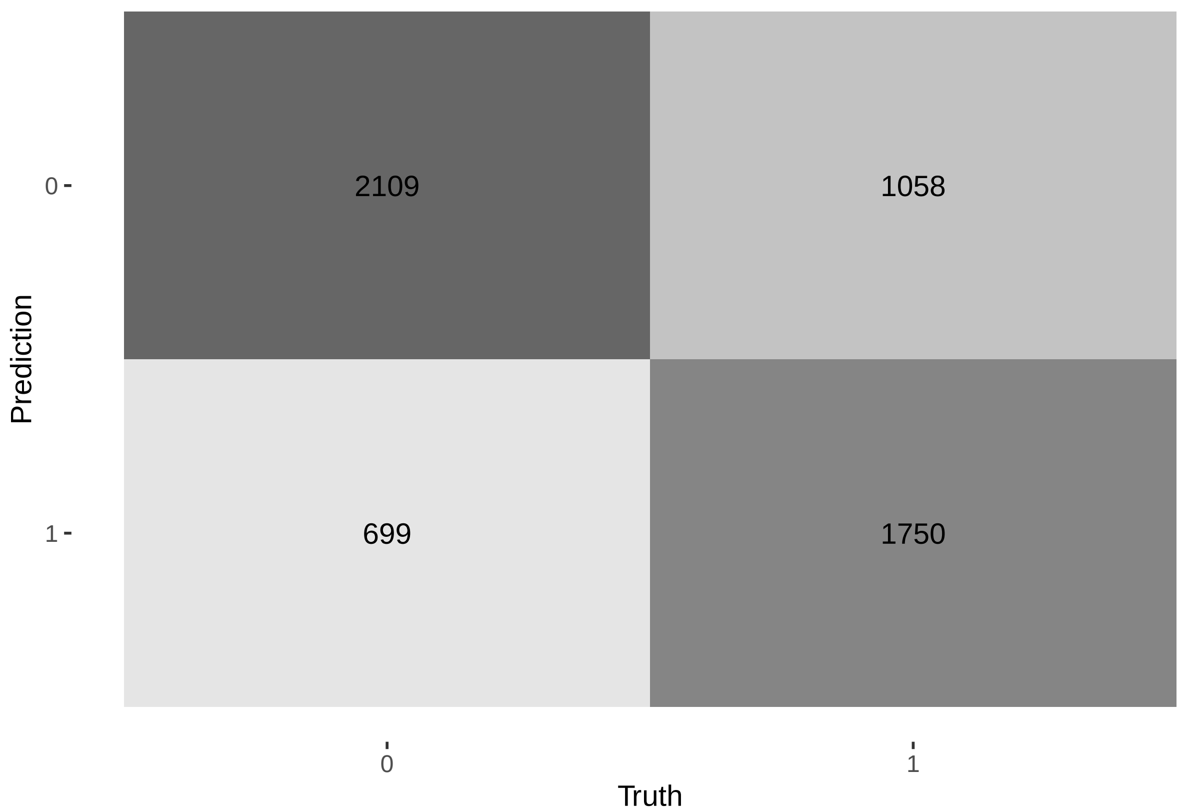
```
## 4 precision binary     0.666     9 0.00513 Preprocessor1_Model1
## 5 recall    binary     0.751     9 0.0124  Preprocessor1_Model1
## 6 roc_auc   binary     0.758     9 0.00913 Preprocessor1_Model1
## 7 sens      binary     0.751     9 0.0124  Preprocessor1_Model1
## 8 spec      binary     0.623     9 0.00733 Preprocessor1_Model1
```

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

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

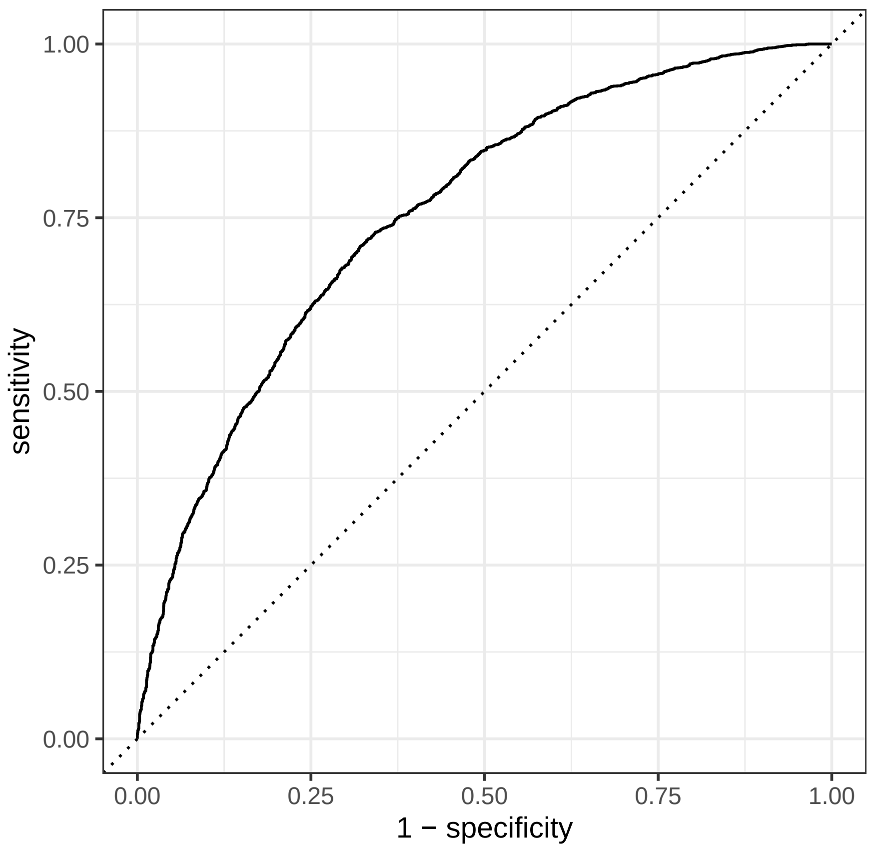
```
##           Truth
## Prediction    0    1
##          0 2109 1058
##          1  699 1750
```

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



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)).

```
rf_pred_hash %>%
  roc_curve(y, .pred_0) %>%
  autoplot()
```



Models on test data

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

Random forest model TF IDF

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

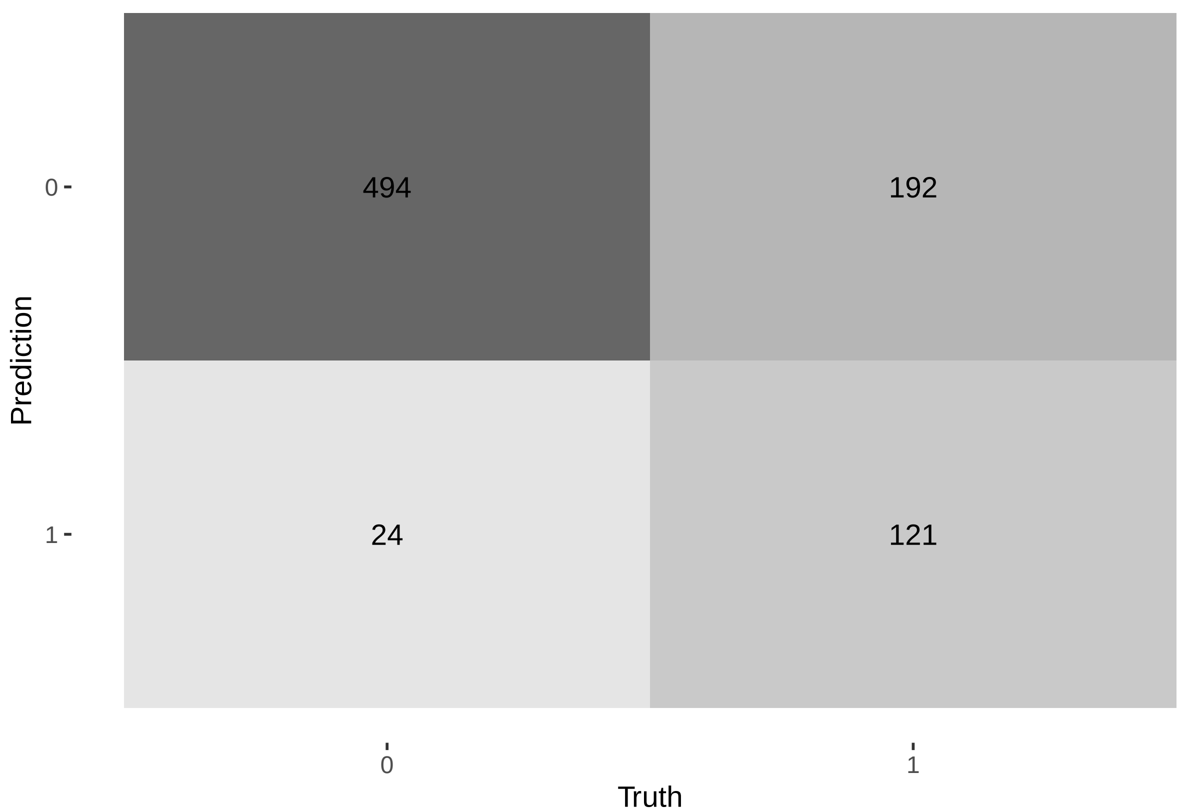
```
last_fit_rf %>%
  collect_metrics()
```

```
## # A tibble: 8 x 4
```

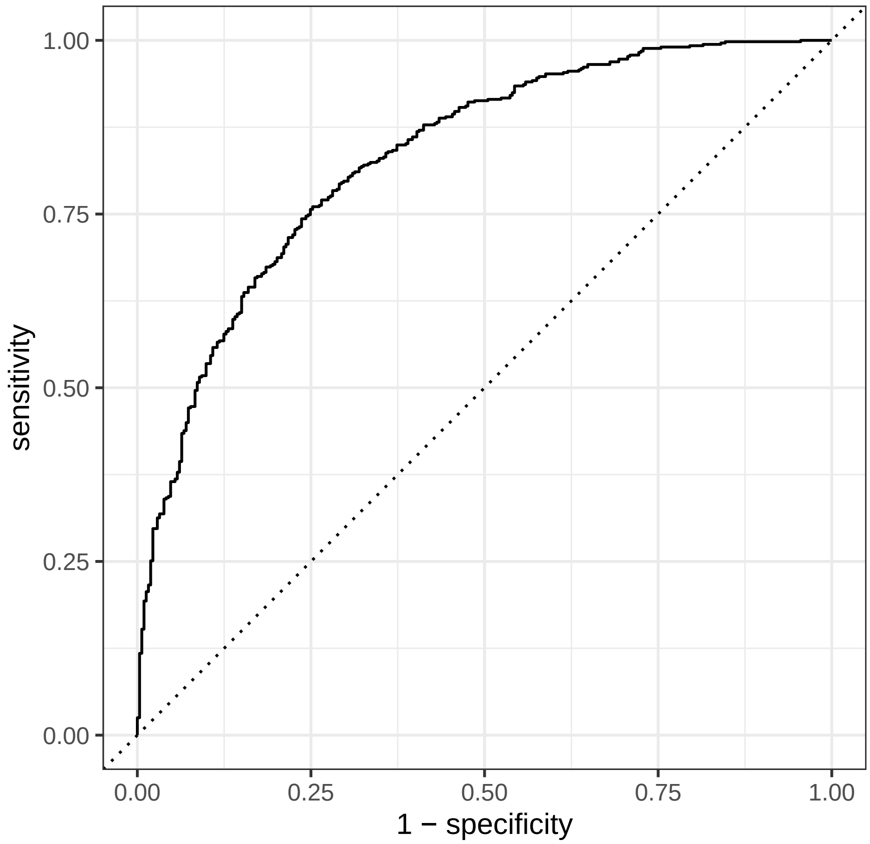
```
##   .metric   .estimator .estimate .config
##   <chr>     <chr>          <dbl> <chr>
## 1 recall    binary         0.954 Preprocessor1_Model1
## 2 precision binary         0.720 Preprocessor1_Model1
## 3 f_meas    binary         0.821 Preprocessor1_Model1
## 4 accuracy  binary         0.740 Preprocessor1_Model1
## 5 kap       binary         0.381 Preprocessor1_Model1
## 6 sens      binary         0.954 Preprocessor1_Model1
## 7 spec      binary         0.387 Preprocessor1_Model1
## 8 roc_auc   binary         0.831 Preprocessor1_Model1
```

We can again make a confusion matrix on the test data predictions

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




```
last_fit_rf %>%
  collect_predictions() %>%
  roc_curve(y, .pred_0) %>%
  autoplot()
```



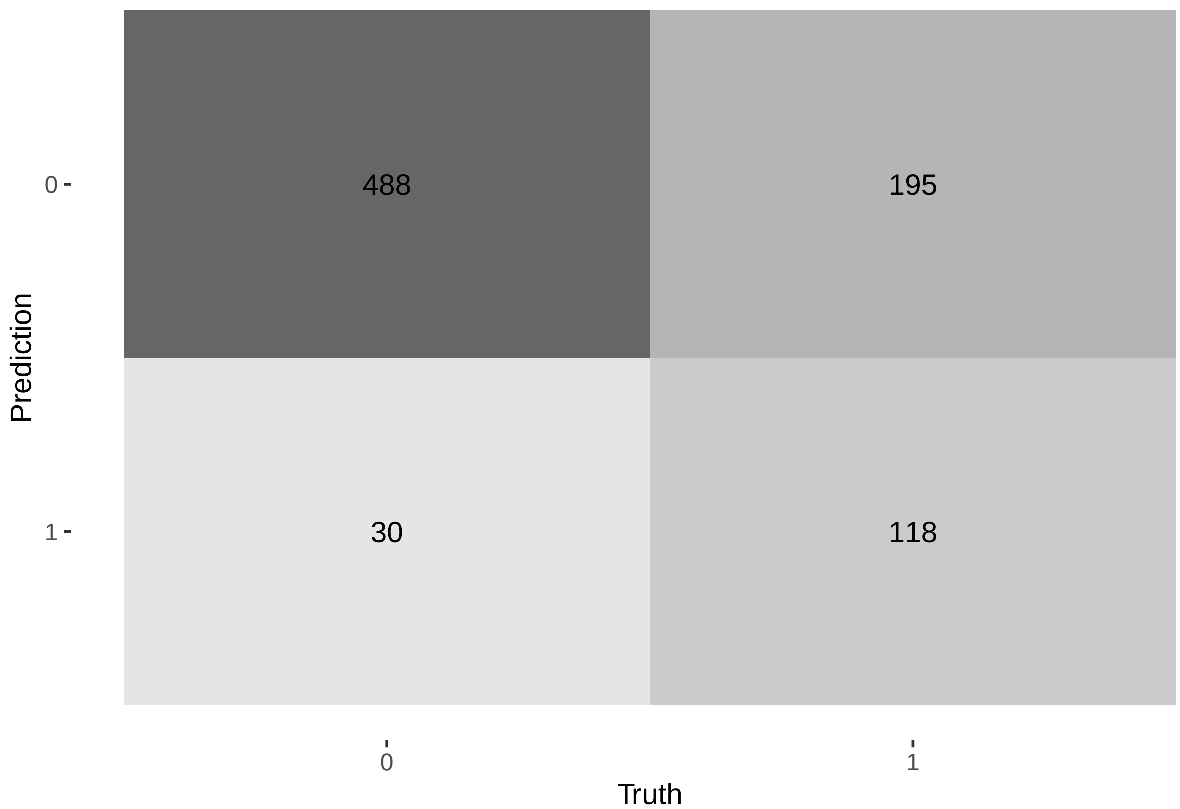
Random forest hash

```
last_fit_rf_hash <- last_fit(workflow_rf_hash,
                             split = tidy_split,
                             metrics = metric_set(
                               recall, precision, f_meas,
                               accuracy, kap,
                               roc_auc, sens, spec)
                             )
```

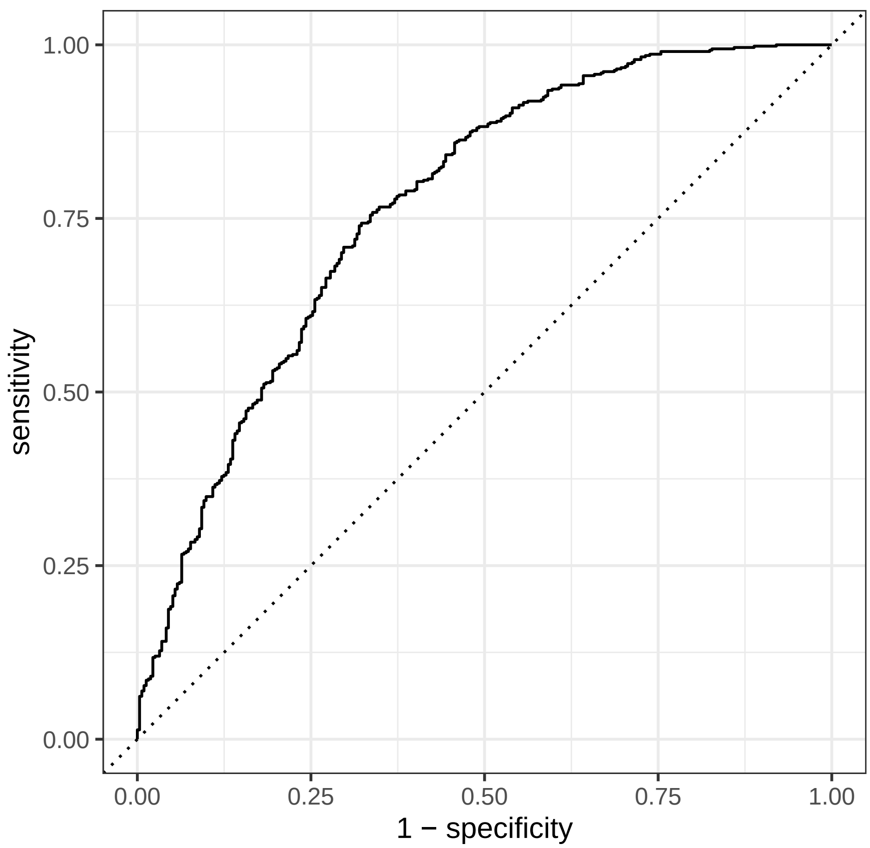
```
last_fit_rf_hash %>%
  collect_metrics()
```

```
## # A tibble: 8 x 4
##   .metric   .estimator .estimate .config
##   <chr>     <chr>          <dbl> <chr>
## 1 recall    binary         0.942 Preprocessor1_Model1
## 2 precision binary         0.714 Preprocessor1_Model1
## 3 f_meas    binary         0.813 Preprocessor1_Model1
## 4 accuracy  binary         0.729 Preprocessor1_Model1
## 5 kap       binary         0.356 Preprocessor1_Model1
## 6 sens      binary         0.942 Preprocessor1_Model1
## 7 spec      binary         0.377 Preprocessor1_Model1
## 8 roc_auc   binary         0.768 Preprocessor1_Model1
```

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




```r
last_fit_rf_hash %>%
  collect_predictions() %>%
  roc_curve(y, .pred_0) %>%
  autoplot()
```


1.00
0.75
0.50
0.25
0.00
sensitivity
0.00
0.25
0.50
0.75
1.00
1 − specificity