Multiclass exam

Andreas, Kristoffer, Mikkel og Simon

31/10/2021

packages

```
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)
```

#load data

```
#titles <- read_csv("https://raw.githubusercontent.com/simonmig10/M2-sds/main/netflix_titles.csv")
data_imdb <- read_csv("https://raw.githubusercontent.com/simonmig10/M2-sds/Simon/IMDb%20movies.csv")
```

Data

Here we create data for multiclass supervised machinelearning

```
data_multi= data_imdb %>%
  filter(genre %in% c("Drama", "Comedy", "Horror", "Thriller"), year >= 2000) %>%
  filter(language == "English") %>%
  select(genre, description) %>%
  rename(text = description) %>%
  rename(y= genre)
```

```
#data_netflix= titles %>%
  #inner_join(data_imdb, by= "title")
```

Here we create data for later embeding, where “data_emb_genre” and “data_emb_title” are created for joining in genres and titles later. And “data_network_emb” used for network analysis

```
data_emb_genre = data_imdb %>%
  select(imdb_title_id, genre)

data_emb_title = data_imdb %>%
  select(imdb_title_id, title)

data_network_emb = data_imdb %>%
  filter(genre %in% c("Drama", "Comedy", "Horror", "Thriller"), year >= 2000) %>%
  filter(language == "English") %>%
  drop_na() %>%
  rename(text = description) %>%
  rename(id= imdb_title_id) %>%
  arrange(desc(avg_vote)) %>%
  select(avg_vote, everything()) %>%
  head(100) %>%
  select(id, text)

data_emb = data_imdb %>%
  filter(genre %in% c("Drama", "Comedy", "Horror", "Thriller"), year >= 2000) %>%
  filter(language == "English") %>%
```

```
  rename(text = description) %>%
  rename(id= imdb_title_id) %>%
  select(id, text)
```

Here we create data for visualization

```
data_genre <- data_imdb %>%
  filter(genre %in% c("Drama", "Comedy", "Horror", "Thriller"), year >= 2000) %>%
  filter(language == "English")

data_genre <- tibble(ID = data_genre[[1]],
                     text = data_genre[[14]] %>% as.character(),
                     labels = data_genre[[6]] %>% as.character())
```

Preprocessing / EDA

Words by genre

We now want to extract the most common words in the 4 genres: Drama, Thriller, Comedy, Horror.

First we tokenize the data

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

We remove short words and stopwords.

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

Stemming: We use the hunspell package which seems to produce the best stemming for our data. The only word which seems to create problems is “broth” so we remove this.

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

```
## # A tibble: 7,484 x 2
##    stem       n
##    <chr>  <int>
##  1 life     581
##  2 friend   461
##  3 family   420
##  4 live     417
##  5 story    332
```

```
##  6 love     297
##  7 woman    290
##  8 town     264
##  9 world    254
## 10 girl     248
## # ... with 7,474 more rows
```

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

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

```
# TFIDF weights
text_genre_tidy %<>%
  add_count(ID, word) %>%
  bind_tf_idf(term = word,
              document = ID,
              n = n)
```

We show the 25 most common words

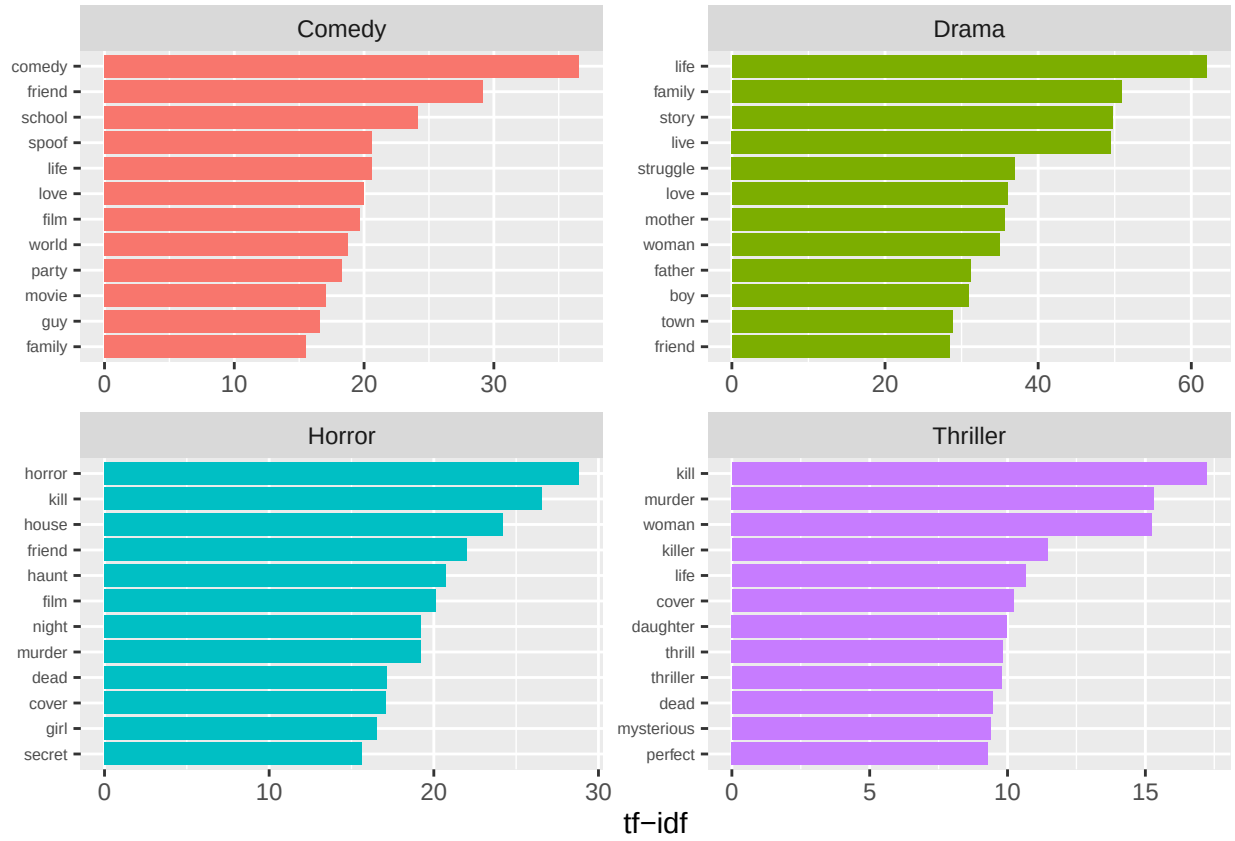
```
# TFIDF topwords
text_genre_tidy %>%
  count(word, wt = tf_idf, sort = TRUE) %>%
  head(25)
```

```
## # A tibble: 25 x 2
##    word       n
##    <chr>  <dbl>
##  1 life   101.
##  2 friend  87.6
##  3 family  87.1
##  4 live    84.2
##  5 story   77.1
##  6 woman   70.7
##  7 love    67.4
##  8 town    61.7
##  9 film    61.7
## 10 girl    60.3
## # ... with 15 more rows
```

We now plot the 12 most used words within each genre

```
labels_words <- text_genre_tidy %>%
  group_by(labels) %>%
  count(word, wt = tf_idf, sort = TRUE, name = "tf_idf") %>%
  dplyr::slice(1:12) %>%
  ungroup()
```

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



Comedy wordcloud

We now want to look at some nice EDA within the Drama and Comedy genre.

```
text_tidy_comedy = text_genre_tidy %>%
  filter(labels == "Comedy")
```

```
library(wordcloud)
```

```
## Loading required package: RColorBrewer
```

```
text_tidy_comedy %>%
  count(word) %>%
  with(wordcloud(word, n,
                 max.words = 50,
                 color = "blue"))
```

Drama wordcloud

```
text_tidy_drama = text_genre_tidy %>%
  filter(labels == "Drama")
```

```
# People love wordclouds
library(wordcloud)

text_tidy_drama %>%
  count(word) %>%
  with(wordcloud(word, n,
                 max.words = 50,
                 color = "blue"))
```

Sentiment Analysis

We do a sentiment analysis based on the different genres.

```
library(textdata)

text_tidy_drama_index= text_tidy_drama %>%
  mutate(index= 1:n())
```

We use the lexicons “bing” and “afinn” to get a measure for positivity and negativity for each word.

We use inner_join to only get the words we use from the lexicon.

```
#Bing
sentiment_bing <- text_tidy_drama_index %>%
  inner_join(get_sentiments("bing")) %>%
  count(word, index = index %/% 100, sentiment) %>%
  mutate(lexicon = 'Bing')
```

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

```
# Afinn
sentiment_afinn <- text_tidy_drama_index %>%
  inner_join(get_sentiments("afinn")) %>%
```

```
  group_by(index = index %/% 100)  %>%
  summarise(sentiment = sum(value, na.rm = TRUE)) %>%
  mutate(lexicon = 'AFINN')
```

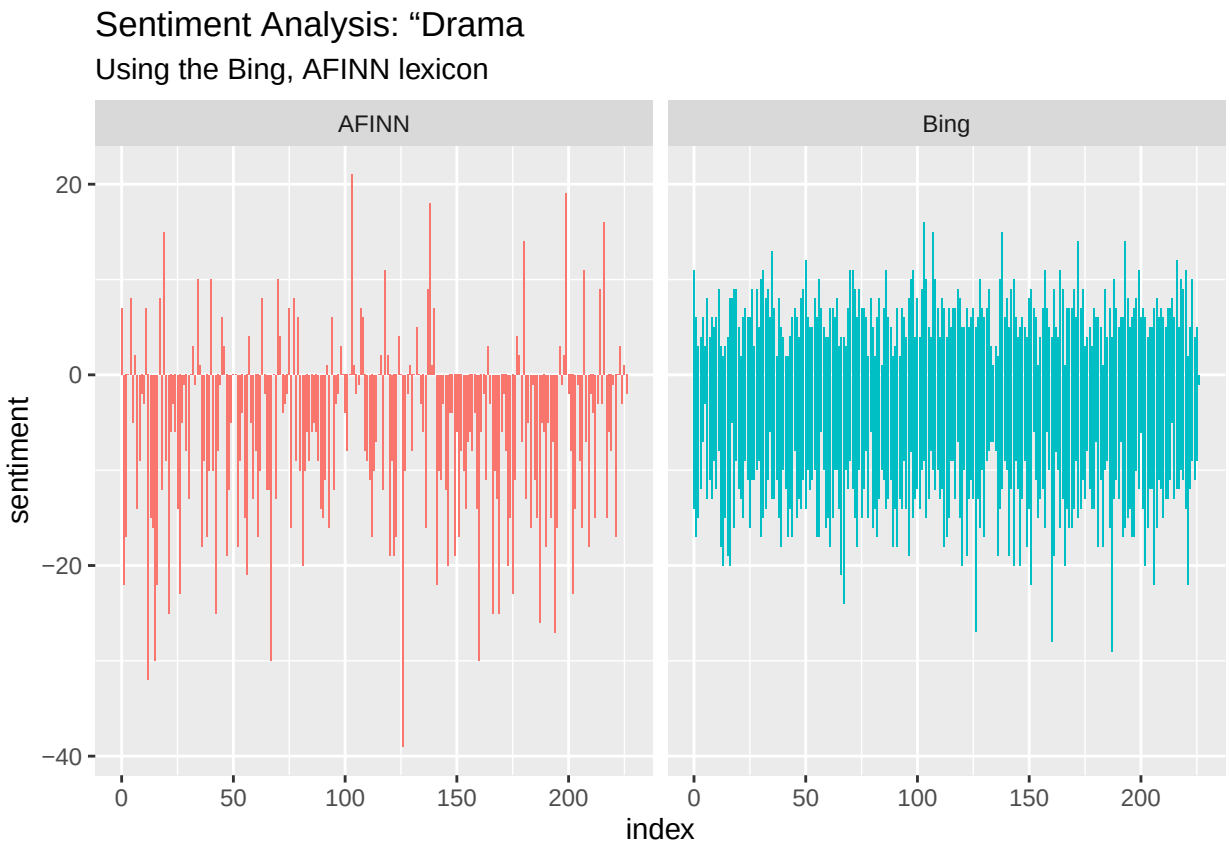
```
## Joining, by = "word"
```

We join the measures from both lexicons

```
# Lets join them all together for plotting
sentiment_all <- sentiment_afinn %>%
  bind_rows(sentiment_bing %>%
              pivot_wider(names_from = sentiment, values_from = n, values_fill = 0) %>%
              mutate(sentiment = positive - negative) %>%
              select(index, sentiment, lexicon))
```

We create a plot for the distribution between negative and positive words within the data genre.

```
sentiment_all %>%
  ggplot(aes(x = index, y = sentiment, fill = lexicon)) +
  geom_col(show.legend = FALSE) +
  facet_wrap(~ lexicon) +
  labs(title = "Sentiment Analysis: "Drama",
       subtitle = 'Using the Bing, AFINN lexicon')
```

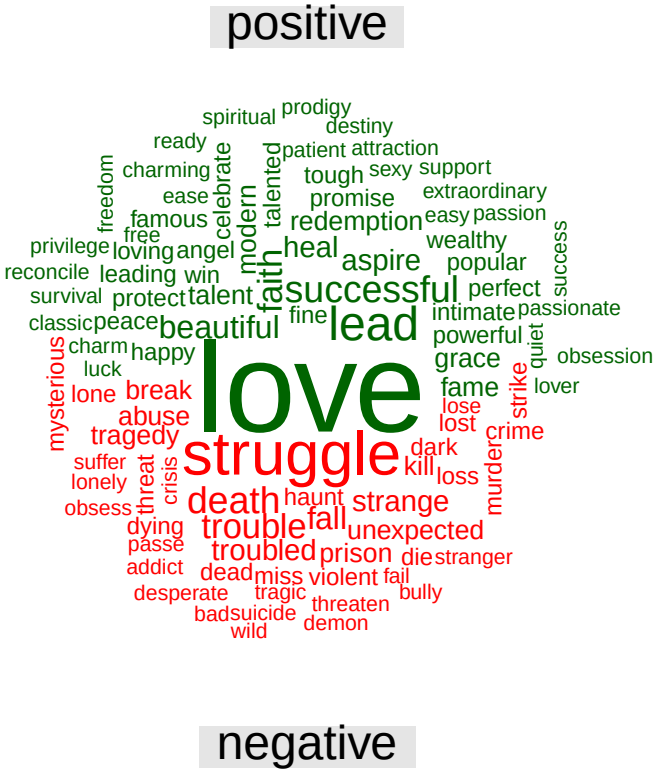


Senteminet wordcloud

We can now create a wordcloud looking at the positive and negative words in the Drama genre.

```
text_tidy_drama %>%
  inner_join(get_sentiments("bing")) %>%
  count(word, sentiment, sort = TRUE) %>%
  filter(sentiment %in% c("positive", "negative")) %>%
  pivot_wider(names_from = sentiment, values_from = n, values_fill = 0) %>%
  as.data.frame() %>%
  remove_rownames() %>%
  column_to_rownames("word") %>%
  comparison.cloud(colors = c("darkgreen", "red"),
                   max.words = 100,
                   title.size = 1.5)
```

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



Network analysis

Co-Word Network

We want to create a network which shows words linked together in a description within the drama genre.

```
library(widyr)
el_words <- text_tidy_drama_index %>%
  pairwise_count(word, ID, sort = TRUE) %>%
  rename(from = item1, to = item2, weight = n)
```

```
## Warning: `distinct_()` was deprecated in dplyr 0.7.0.
## Please use `distinct()` instead.
## See vignette('programming') for more help
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

```
library(tidygraph)
```

```
##
## Attaching package: 'tidygraph'
```

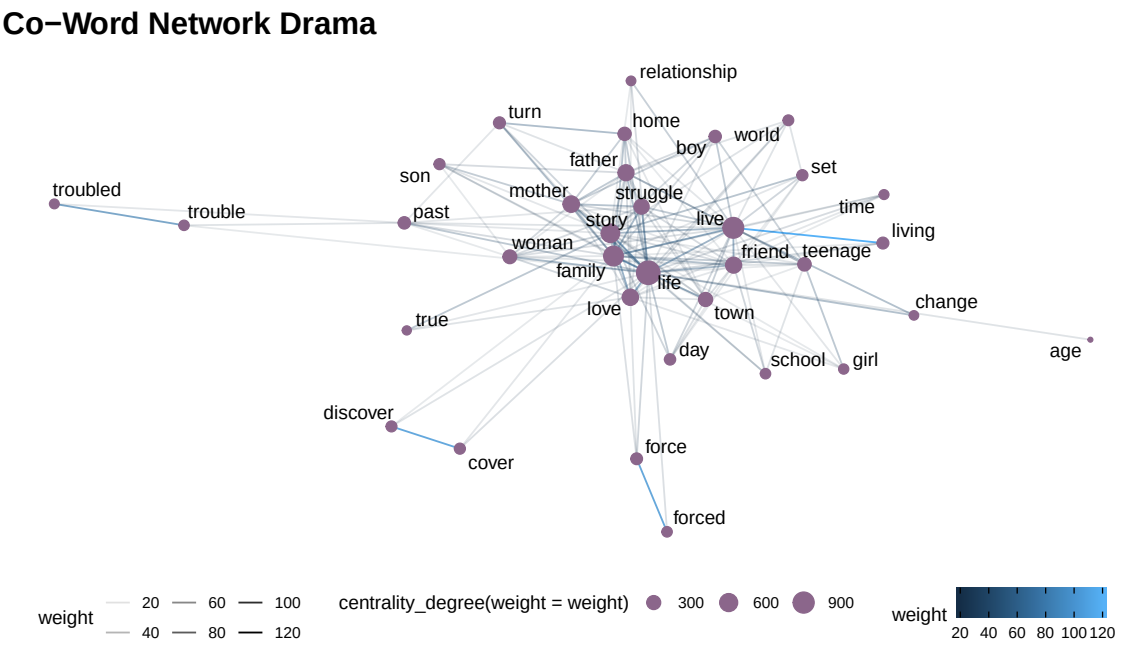
```
## The following object is masked from 'package:xgboost':
##
##     slice
```

```
## The following object is masked from 'package:stats':
##
##     filter
```

```
library(ggraph)
```

```
g <- el_words %>%
  filter(weight >= 9) %>%
  as_tbl_graph(directed = FALSE) %>%
  igraph::simplify() %>% as_tbl_graph()
```

```
set.seed(1337)
g %N>%
  filter(centrality_degree(weight = weight) > 100) %>%
  ggraph(layout = "fr") +
  geom_edge_link(aes(color = weight, edge_alpha = weight)) +
  geom_node_point(aes(size = centrality_degree(weight = weight)), color = "plum4") +
  geom_node_text(aes(label = name,), repel = TRUE) +
  theme_graph(base_family="sans") +
  theme(legend.position = 'bottom') +
  labs(title = 'Co-Word Network Drama')
```



We can see that life seems to be the most central word in Drama descriptions

Bigrams

We now want to create the same network just using bigrams.

```
data_multi2= data_imdb %>%
  filter(genre %in% c("Drama", "Comedy", "Horror", "Thriller"), year >= 2000) %>%
  filter(language == "English") %>%
  select(imdb_title_id, description) %>%
  rename(text = description) %>%
  rename(id= imdb_title_id)

text_tidy_ngrams = data_multi2 %>%
  unnest_tokens(bigram, text, token = "ngrams", n = 2) %>%
  na.omit()
```

Here we get the top 100 used bigrams within all four genres.

```
text_tidy_ngrams %>%
  count(bigram, sort = TRUE) %>%
  head(100)
```

```
## # A tibble: 100 x 2
##    bigram       n
##    <chr>    <int>
##  1 in the     609
```

```
##  2 of a       449
##  3 in a       445
##  4 of the     404
##  5 a young    311
##  6 on the     292
##  7 is a       264
##  8 on a       244
##  9 group of   236
## 10 with a     236
## # ... with 90 more rows
```

a lot of the most common bigrams are stopwords, so these will be removed

```
text_tidy_ngrams %<>%
  separate(bigram, c("word1", "word2"), sep = " ") %>%
  anti_join(stop_words, by = c('word1' = 'word')) %>%
  anti_join(stop_words, by = c('word2' = 'word')) %>%
  unite(bigram, word1, word2, sep = " ") %>%
  count(id, bigram)
text_tidy_ngrams %>%
  count(bigram, wt = n, sort = TRUE) %>%
  head(50)
```

```
## # A tibble: 50 x 2
##    bigram               n
##    <chr>            <int>
##  1 los angeles         61
##  2 serial killer       46
##  3 york city           40
##  4 road trip           38
##  5 college students    22
##  6 single mother       22
##  7 returns home        21
##  8 teenage girl        20
##  9 true story          18
## 10 real life           16
## # ... with 40 more rows
```

These are the most used bigrams through the four genres analyzed.

```
bi = text_tidy_ngrams %>% left_join(text_tidy_ngrams %>% select(id, bigram), by = "id")
bi = bi %>% select(-id) %>%
  rename(from = bigram.x, to = bigram.y) %>%
  filter(from != to); head(bi)
```

```
## # A tibble: 6 x 3
##   from                     n to
##   <chr>                <int> <chr>
## 1 bewildering unnerving     1 blackly comic
## 2 bewildering unnerving     1 comic film
## 3 bewildering unnerving     1 crispin glover
## 4 bewildering unnerving     1 facing villains
## 5 bewildering unnerving     1 multiple planes
## 6 bewildering unnerving     1 outer struggles
```

```
bi = bi %>% count(from, to, name = 'weight')
bi %>%
  count(weight)
```

```
## # A tibble: 6 x 2
##   weight      n
##    <int>  <int>
## 1      1 142358
## 2      2    202
## 3      3     12
## 4      4      6
## 5      5      2
## 6      6      2
```

a lot of bigram combination only happend once, which is not very important so those are removed.

```
bi = bi %>% filter(weight >= 2)
bi %>% arrange(desc(weight))
```

```
## # A tibble: 224 x 3
##    from         to           weight
##    <chr>        <chr>         <int>
##  1 life crisis  mid life          6
##  2 mid life     life crisis       6
##  3 breaks loose hell breaks       5
##  4 hell breaks  breaks loose      5
##  5 estate agent real estate       4
##  6 real estate  estate agent      4
##  7 trade center world trade       4
##  8 war ii       world war         4
##  9 world trade  trade center      4
## 10 world war    war ii            4
## # ... with 214 more rows
```

all combinations of bigrams are present twice as of now, which can be seen by the first two observations above. So the next thing to do is to remove duplicates.

```
bi = bi[!duplicated(t(apply(bi[c("from", "to")], 1, sort))), ]
bi %>%
  arrange(desc(weight))
```

```
## # A tibble: 112 x 3
##    from            to              weight
##    <chr>           <chr>            <int>
##  1 life crisis     mid life             6
##  2 breaks loose    hell breaks          5
##  3 estate agent    real estate          4
##  4 trade center    world trade          4
##  5 war ii          world war            4
##  6 bachelor party  las vegas            3
##  7 car crash       fatal car            3
```

```
##  8 college friends friends reunite      3
##  9 footage horror  found footage        3
## 10 horror films    short horror         3
## # ... with 102 more rows
```

That helped a lot.

Then the network is created using table_graph

```
g <- bi %>% as_tbl_graph(directed = FALSE) ; g
```

```
## # A tbl_graph: 177 nodes and 112 edges
## #
## # An undirected simple graph with 73 components
## #
## # Node Data: 177 x 1 (active)
##   name
##   <chr>
## 1 11 2001
## 2 abandoned childhood
## 3 abandoned mental
## 4 accident leaves
## 5 accidentally unleash
## 6 aids crisis
## # ... with 171 more rows
## #
## # Edge Data: 112 x 3
##    from    to weight
##   <int> <int>  <int>
## 1     1    91      2
## 2     2    99      2
## 3     3   100      2
## # ... with 109 more rows
```

Each node represent a bigram and each edge a connection bigrams who apeared in the same movie descriptions together the weight being the number of times.

Then centrality measures are calculated

```
g <- g %N>%
  mutate(cent_dgr = centrality_degree(weights = weight),
         cent_eigen = centrality_eigen(weights = weight),
         cent_between = centrality_betweenness(weights = weight))
g %N>%
  as_tibble() %>%
  arrange(desc(cent_dgr))
```

```
## # A tibble: 177 x 4
##    name                cent_dgr cent_eigen cent_between
##    <chr>                  <dbl>      <dbl>        <dbl>
##  1 life crisis               13   1.00e+ 0            5
##  2 york city                 10   0.                 14
##  3 serial killer             10   1.09e-16           30
```

```
##  4 los angeles                8   2.61e-17           5
##  5 road trip                  8   6.36e-18           5
##  6 horror film                6   1.19e-16          14
##  7 northwest town             6   3.17e-16           0
##  8 obsessions begin           6   3.35e-16           0
##  9 spiritual obsessions       6   3.17e-16           0
## 10 mid life                   6   8.02e- 1           0
## # ... with 167 more rows
```

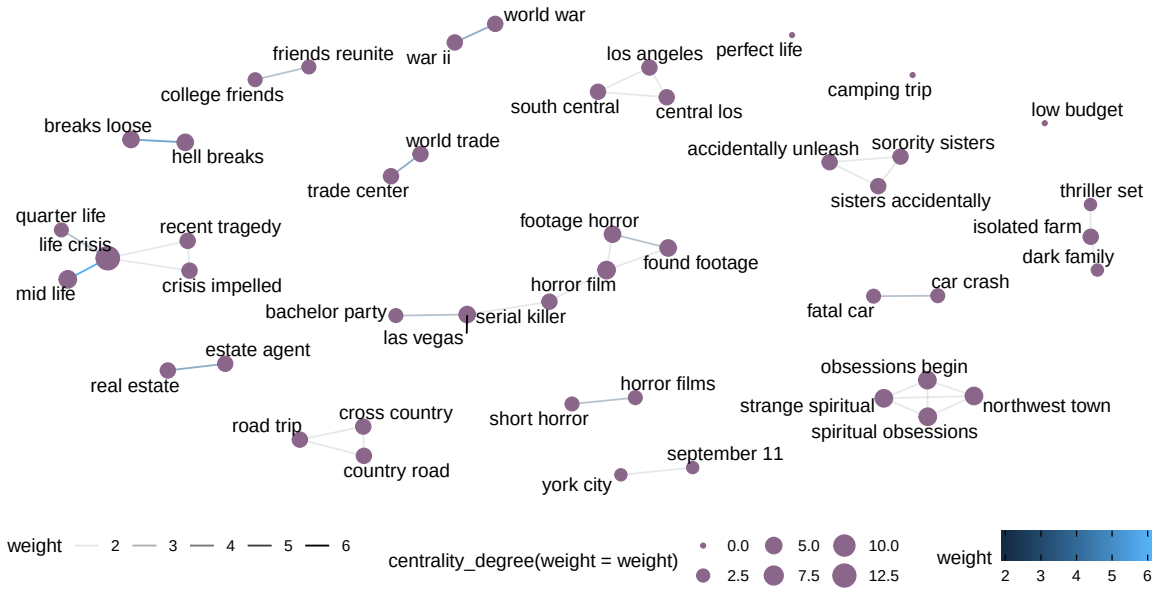
Centrality degree: How many connections a node has - what it tells: How many direct, one hop connections each node has to other nodes in the network.

Betweenness centrality: Measures the number of times a node lies on the shortest path between other nodes. - What it tells: This measure shows which nodes are bridges between nodes in a network. It does this by identifying all the shortest paths and then counting how many times each node falls on one.

Eigen centrality: Like degree centrality, EigenCentrality measures a node's influence based on the number of links it has to other nodes in the network. EigenCentrality then goes a step further by also taking into account how well connected a node is, and how many links their connections have, and so on through the network. - What it tells: By calculating the extended connections of a node, EigenCentrality can identify nodes with influence over the whole network, not just those directly connected to it.

Now the network will be ploted for nodes haveing an centrality degree of above two, to only show the most important bigrams.

```
set.seed(1337)
g %N>%
  filter(centrality_degree(weight = weight) > 2) %>%
  ggraph(layout = "fr") +
  geom_edge_link(aes(color = weight, edge_alpha = weight)) +
  geom_node_point(aes(size = centrality_degree(weight = weight)), color = "plum4") +
  geom_node_text(aes(label = name,), repel = TRUE) +
  theme_graph(base_family="sans") +
  theme(legend.position = 'bottom') +
  labs(title = 'Bigram moive network')
```



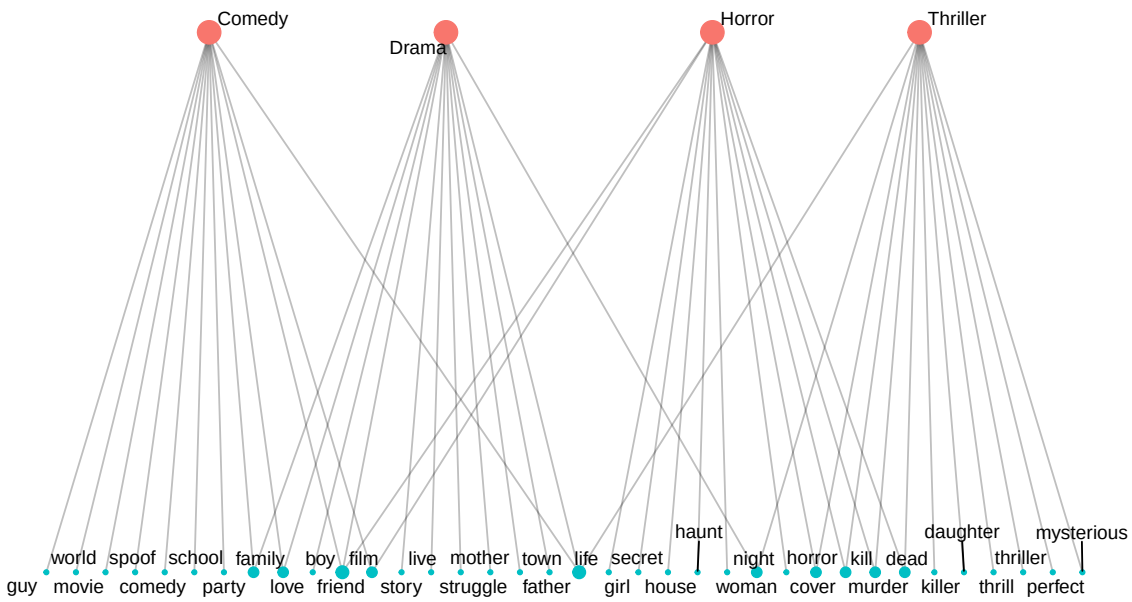
2-mode network

By extracting the words and genres a 2-mode network can be created using the as_tbl_graph function. Further manipulation is done to make node types, which is done to make sure, that nodes with the same type cant connect.

```
library(tidygraph)

edgelist = tibble(name = labels_words$labels, words = labels_words$word)
g = as_tbl_graph(edgelist)
g = g %N>% mutate(type = as.logical(c(rep(0,4), rep(1, 36))))
```

Then the 2-mode network is plotted

```
library(ggraph)

set.seed(1337)
p <- g %>% ggraph("bipartite") +
  geom_edge_link(alpha = 0.25) +
  geom_node_point(aes(col = type, size = centrality_degree(mode = 'all'))) +
  geom_node_text(aes(label = name), repel = TRUE) +
  theme_graph(base_family="sans") +
  theme(legend.position = 'none') +
  labs(title = '2-mode network Genres-words')
p
```



We can eg. see that life is used a lot within the genres Comedy, Drama and Thrillers but not within the Horror genre.

Now a 1-mode network is made both between genres and words.

```
library(igraph)
```

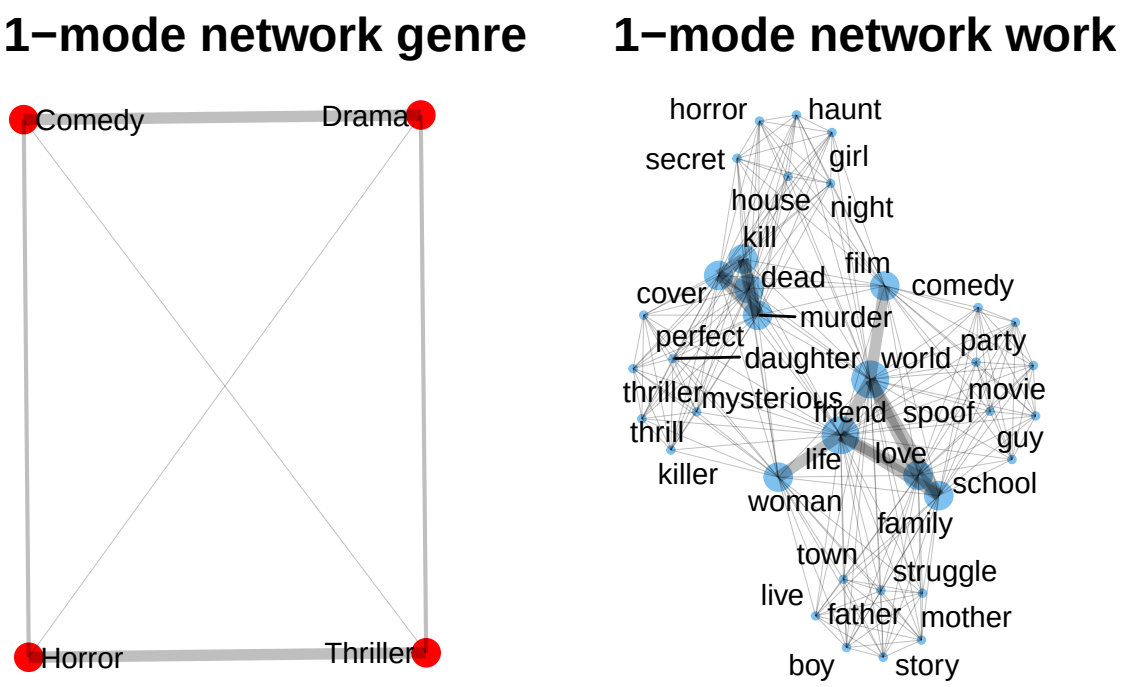
```
##
## Attaching package: 'igraph'

## The following object is masked from 'package:tidygraph':
##
##     groups

## The following objects are masked from 'package:dials':
##
##     degree, neighbors

## The following objects are masked from 'package:lubridate':
##
##     %--%, union

## The following objects are masked from 'package:dplyr':
##
##     as_data_frame, groups, union

## The following objects are masked from 'package:purrr':
##
##     compose, simplify
```

```
## The following object is masked from 'package:tidyr':
##
##     crossing

## The following object is masked from 'package:tibble':
##
##     as_data_frame

## The following objects are masked from 'package:stats':
##
##     decompose, spectrum

## The following object is masked from 'package:base':
##
##     union
```

```
g_projected <- g %>% bipartite_projection()
g_genre = g_projected[["proj1"]] %>% as_tbl_graph(directed = FALSE)
g_words = g_projected[["proj2"]] %>% as_tbl_graph(directed = FALSE)
```

Then both networks are plotted

```
set.seed(1337)
library(patchwork)
p2 <- g_genre %>% ggraph(layout = "nicely") +
  geom_node_point(aes(size = centrality_degree(weights = weight)), col = 'red') +
  geom_edge_link(aes(width = weight), alpha = 0.25) +
  scale_edge_width(range = c(0.1, 2)) +
  geom_node_text(aes(label = name), repel = TRUE) +
  theme_graph(base_family="sans") +
  theme(legend.position = 'none') +
  labs(title = '1-mode network genres')
p3 <- g_words %>% ggraph(layout = "nicely") +
  geom_node_point(aes(size = centrality_degree(weights = weight)), col = 'skyblue2') +
  geom_edge_link(aes(width = weight), alpha = 0.25) +
  scale_edge_width(range = c(0.1, 2)) +
  geom_node_text(aes(label = name), repel = TRUE) +
  theme_graph(base_family="sans") +
  theme(legend.position = 'none') +
  labs(title = '1-mode network work')
p2 + p3
```



In the left network we can see that the thicker the line the more words the two genres have in common.

In the right network we see the words that are used within the same genres, with an edge showing that the words are used together within a genre, and the width showing how many times they are used together, and the size of the node showing how many genres they are represented in.

Multiclass sml model

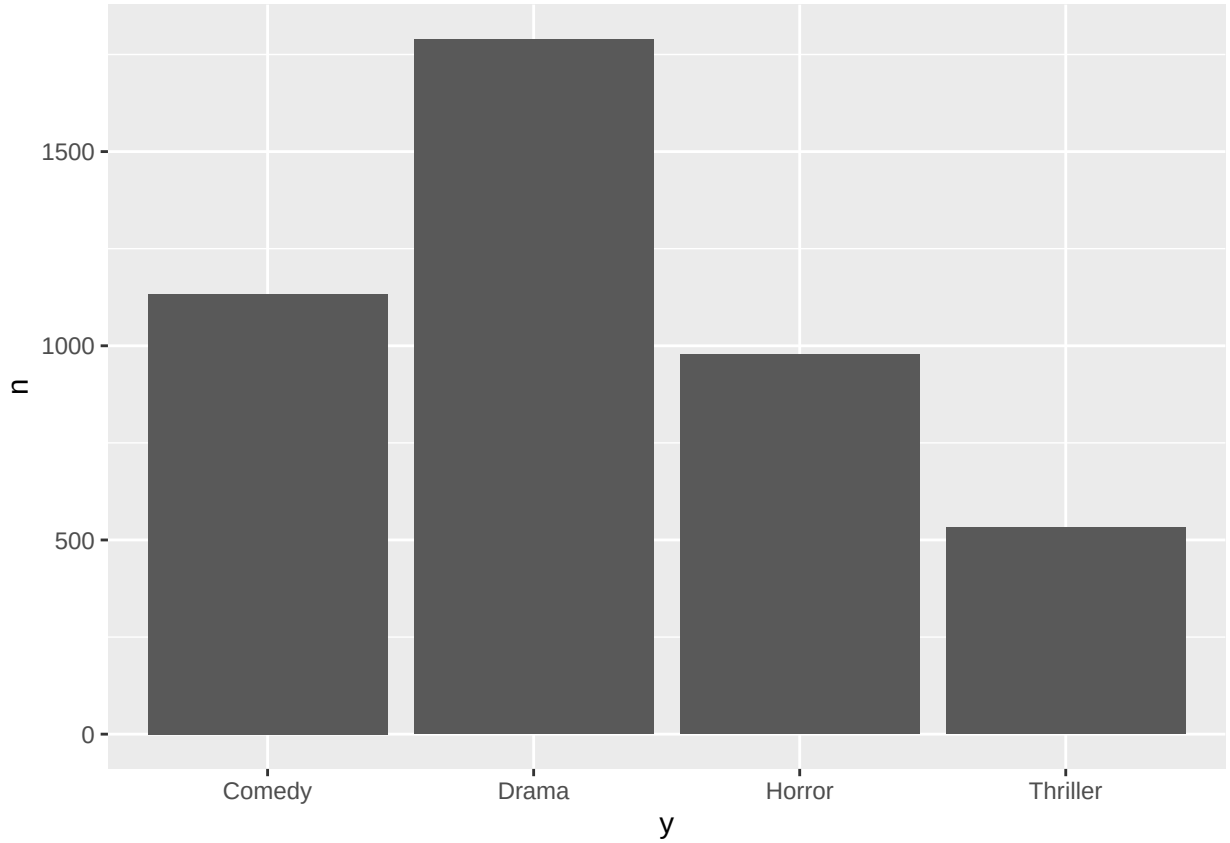
We now want to create a multiclass supervised machinelearning model to predict which genre a movie is in, based on its description.

First we look for missing values.

```
vis_dat(data_multi)
```

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

```
data_multi %<>%
  drop_na()

data_multi %>%
  count(y) %>%
  ggplot(aes(x = y, y = n)) +
  geom_col()
```



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

We will create three different recipies 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.

```
set.seed(19)

tidy_split <- initial_split(data_multi, 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: 4 x 2
##   y            n
##   <fct>    <int>
## 1 Comedy     399
## 2 Drama      399
## 3 Horror     399
## 4 Thriller   399
```

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:

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

Logistic model

```
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
)


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

```
# 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
)
```

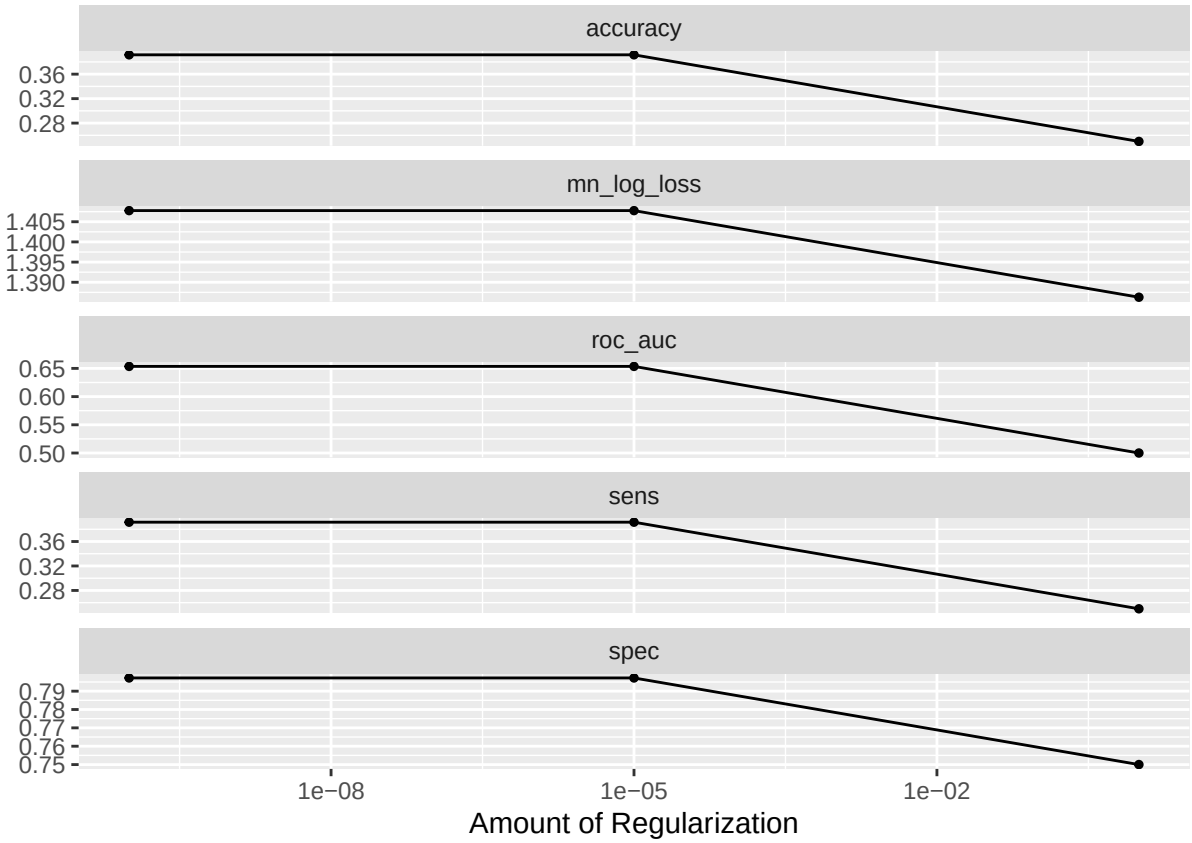
```
# Tune tf-idf models
linear_tf_res <- tune_grid(
  model_lg,
  tf_idf_rec,
  grid = logistic_grid,
  control = model_control,
  metrics = model_metrics,
  resamples = k_folds_data
)


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()
```

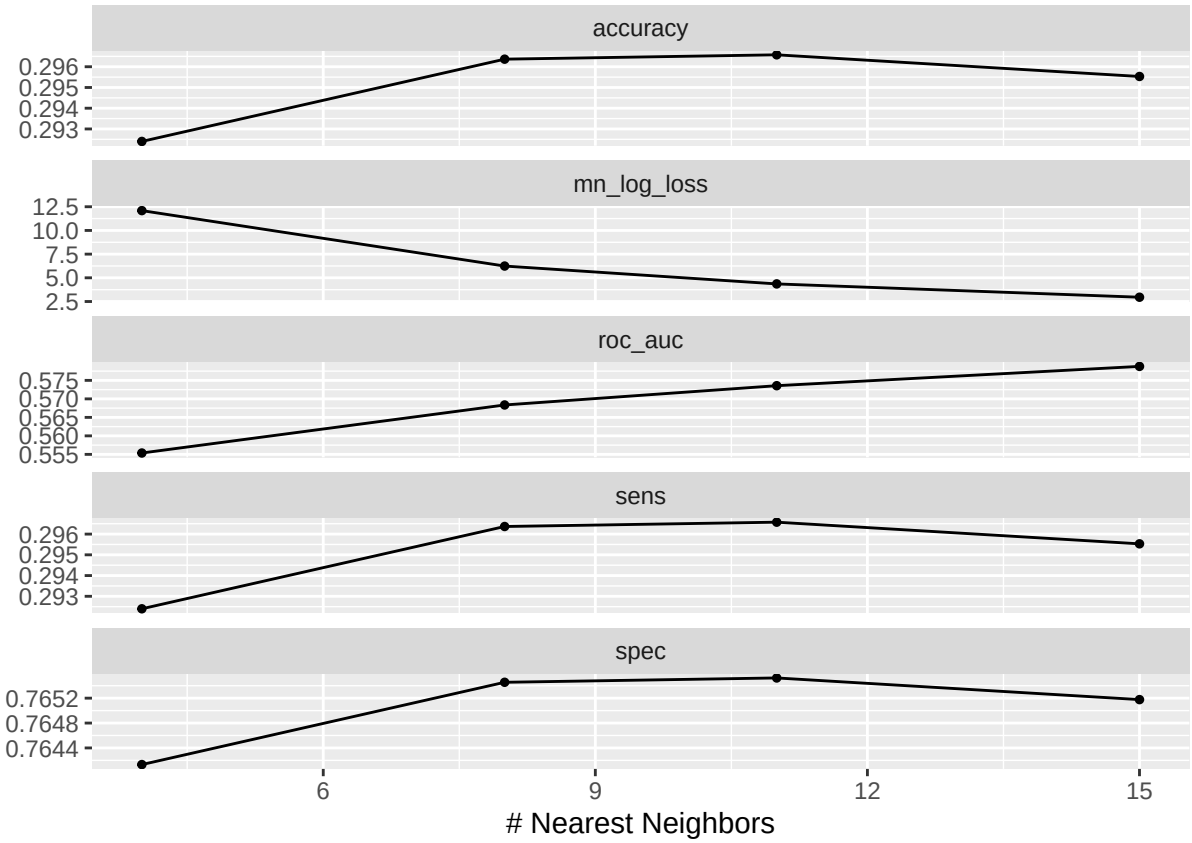



```
best_param_linear_hash_res <- linear_hash_res %>% select_best(metric = 'accuracy')
best_param_linear_hash_res
```

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

knn_embed_res We use autoplot

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

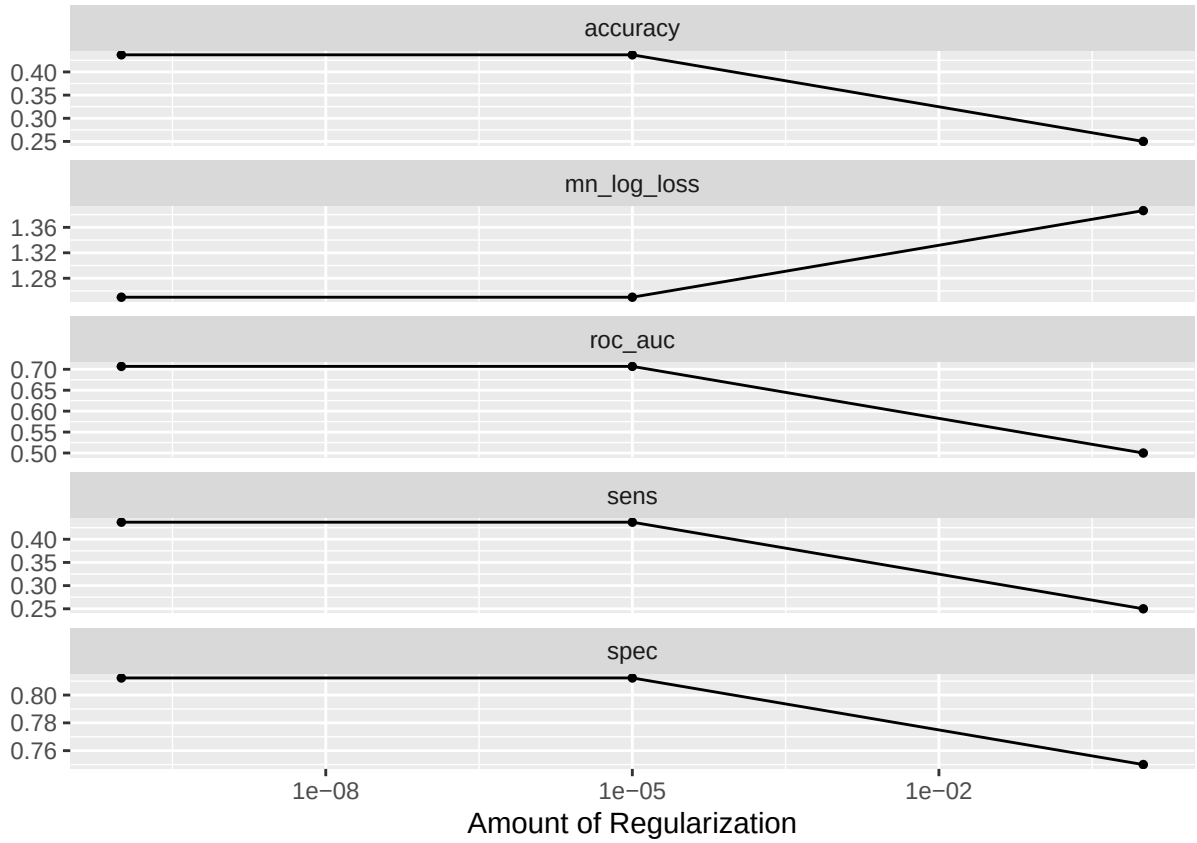



```
best_param_knn_hash_res <- knn_hash_res %>% select_best(metric = 'accuracy')
best_param_knn_hash_res
```

```
## # A tibble: 1 x 2
##   neighbors .config
##       <int> <chr>
## 1        11 Preprocessor1_Model3
```

linear_embed_res We use autoplot

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

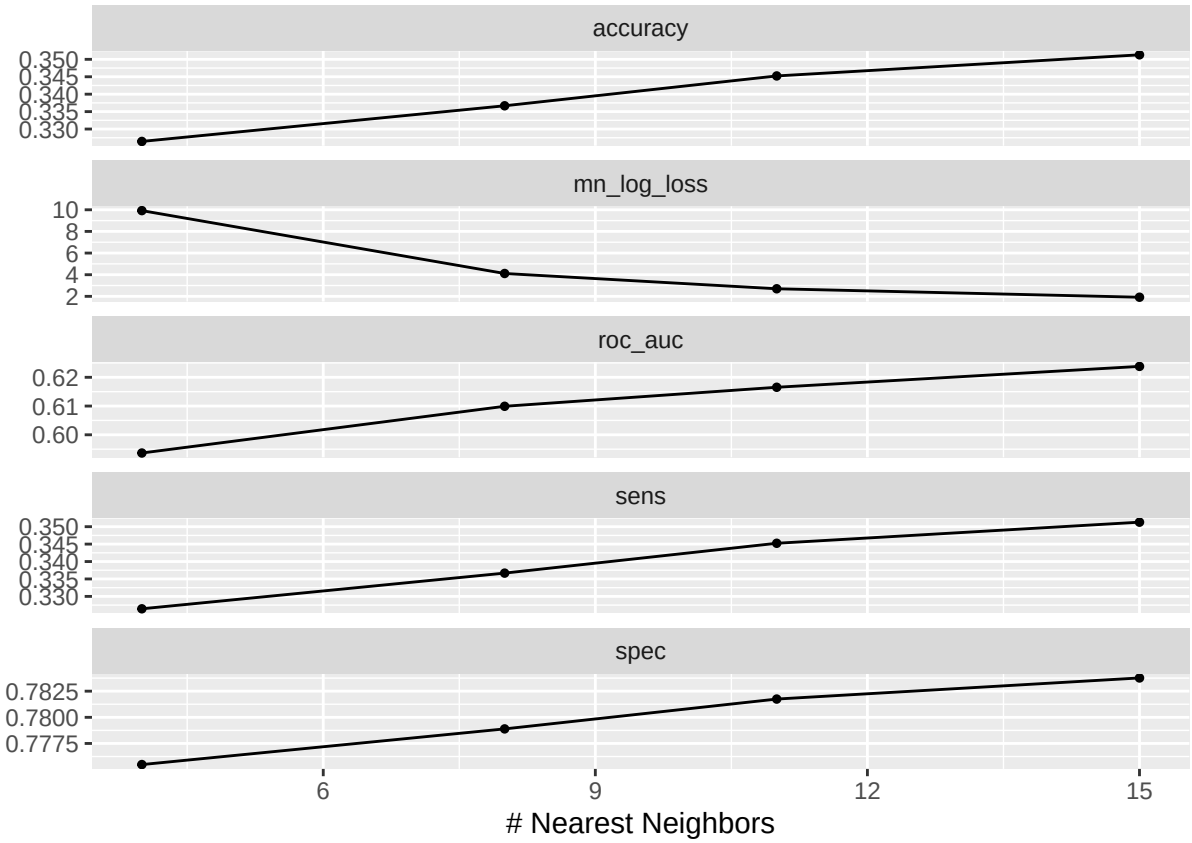



```
best_param_linear_embed_res <- linear_embed_res %>% select_best(metric = 'accuracy')
best_param_linear_embed_res
```

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

knn_embed_res We use autoplot

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

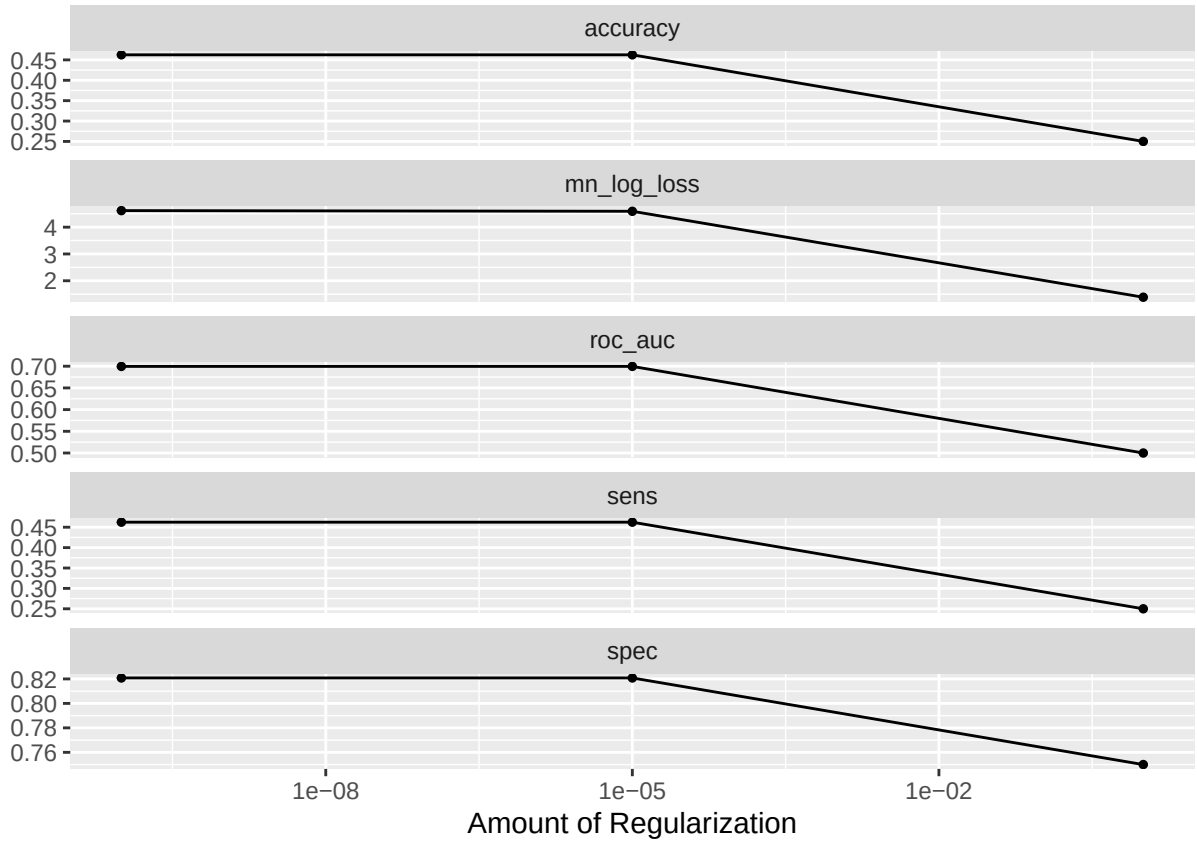



```
best_param_knn_embed_res <- knn_embed_res %>% select_best(metric = 'accuracy')
best_param_knn_embed_res
```

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

linear_tf_res We use autoplot

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

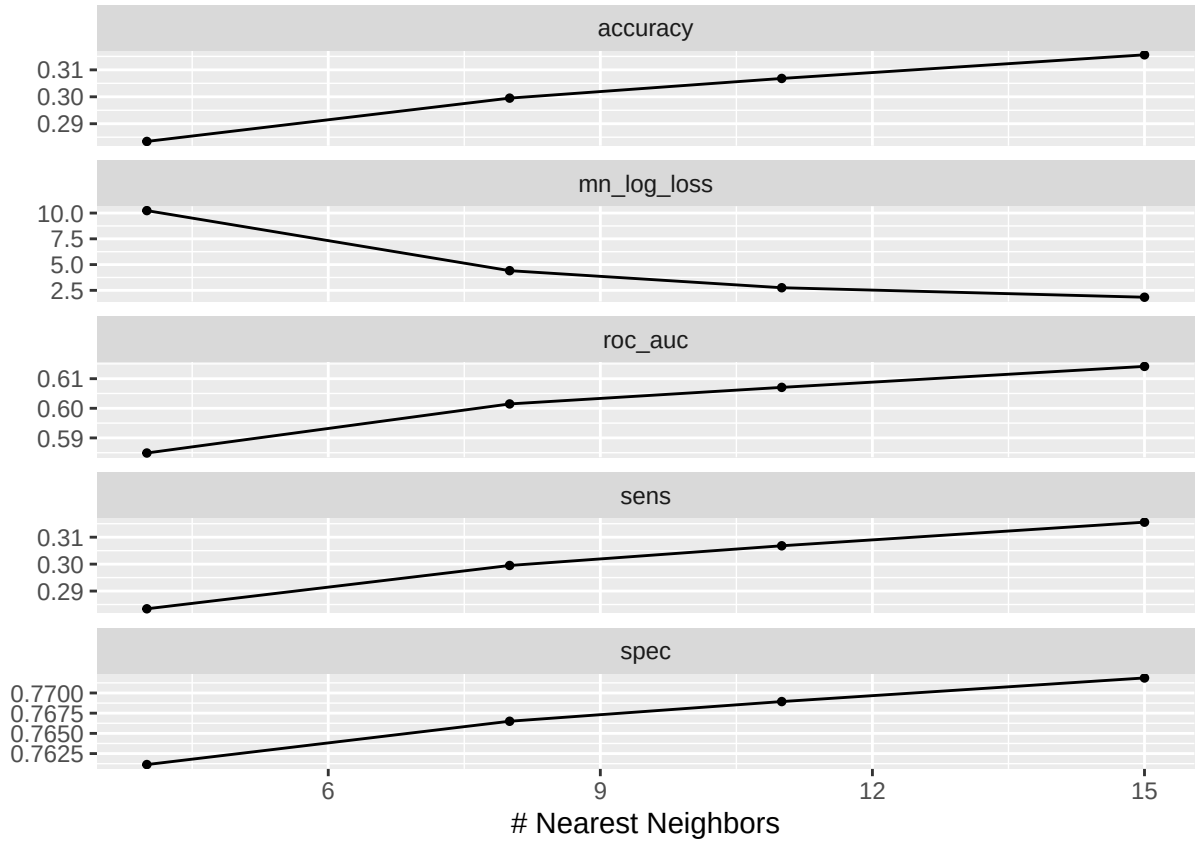



```
best_param_linear_tf_res <- linear_tf_res %>% select_best(metric = 'accuracy')
best_param_linear_tf_res
```

```
## # 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        15 Preprocessor1_Model4
```

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 0.392     9 0.00584 Preprocessor1_Model1
## 2 f_meas    macro      0.392     9 0.00583 Preprocessor1_Model1
## 3 kap       multiclass 0.189     9 0.00779 Preprocessor1_Model1
## 4 precision macro      0.393     9 0.00580 Preprocessor1_Model1
## 5 recall    macro      0.392     9 0.00584 Preprocessor1_Model1
```

```
## 6 roc_auc   hand_till  0.654     9 0.00345 Preprocessor1_Model1
## 7 sens      macro      0.392     9 0.00584 Preprocessor1_Model1
## 8 spec      macro      0.797     9 0.00195 Preprocessor1_Model1
```

```
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 0.462     9 0.00893 Preprocessor1_Model1
## 2 f_meas    macro      0.462     9 0.00897 Preprocessor1_Model1
## 3 kap       multiclass 0.283     9 0.0119  Preprocessor1_Model1
## 4 precision macro      0.463     9 0.00899 Preprocessor1_Model1
## 5 recall    macro      0.462     9 0.00893 Preprocessor1_Model1
## 6 roc_auc   hand_till  0.700     9 0.00629 Preprocessor1_Model1
## 7 sens      macro      0.462     9 0.00893 Preprocessor1_Model1
## 8 spec      macro      0.821     9 0.00298 Preprocessor1_Model1
```

```
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 0.437     9 0.00970 Preprocessor1_Model1
## 2 f_meas    macro      0.435     9 0.00931 Preprocessor1_Model1
## 3 kap       multiclass 0.249     9 0.0129  Preprocessor1_Model1
## 4 precision macro      0.438     9 0.00882 Preprocessor1_Model1
## 5 recall    macro      0.437     9 0.00970 Preprocessor1_Model1
## 6 roc_auc   hand_till  0.707     9 0.00489 Preprocessor1_Model1
## 7 sens      macro      0.437     9 0.00970 Preprocessor1_Model1
## 8 spec      macro      0.812     9 0.00323 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 0.297      9 0.00425 Preprocessor1_Model1
## 2 f_meas    macro      0.252      9 0.00690 Preprocessor1_Model1
## 3 kap       multiclass 0.0621     9 0.00566 Preprocessor1_Model1
## 4 precision macro      0.338      9 0.0104  Preprocessor1_Model1
## 5 recall    macro      0.297      9 0.00425 Preprocessor1_Model1
## 6 roc_auc   hand_till  0.574      9 0.00309 Preprocessor1_Model1
## 7 sens      macro      0.297      9 0.00425 Preprocessor1_Model1
## 8 spec      macro      0.766      9 0.00142 Preprocessor1_Model1
```

```
knn_res_tf <-
  workflow_final_knn_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)
    )

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 0.316      9 0.00885 Preprocessor1_Model1
## 2 f_meas    macro      0.279      9 0.0154  Preprocessor1_Model1
## 3 kap       multiclass 0.0874     9 0.0118  Preprocessor1_Model1
## 4 precision macro      0.375      9 0.0156  Preprocessor1_Model1
## 5 recall    macro      0.316      9 0.00885 Preprocessor1_Model1
## 6 roc_auc   hand_till  0.614      9 0.00740 Preprocessor1_Model1
## 7 sens      macro      0.316      9 0.00885 Preprocessor1_Model1
## 8 spec      macro      0.772      9 0.00295 Preprocessor1_Model1
```

```
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 0.351     9 0.0109  Preprocessor1_Model1
## 2 f_meas    macro      0.349     9 0.0115  Preprocessor1_Model1
## 3 kap       multiclass 0.135     9 0.0145  Preprocessor1_Model1
## 4 precision macro      0.356     9 0.0112  Preprocessor1_Model1
## 5 recall    macro      0.351     9 0.0109  Preprocessor1_Model1
## 6 roc_auc   hand_till  0.624     9 0.00614 Preprocessor1_Model1
## 7 sens      macro      0.351     9 0.0109  Preprocessor1_Model1
## 8 spec      macro      0.784     9 0.00363 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 0.444     9 0.00870 Preprocessor1_Model1
## 2 f_meas    macro      0.444     9 0.00875 Preprocessor1_Model1
## 3 kap       multiclass 0.259     9 0.0116  Preprocessor1_Model1
## 4 precision macro      0.445     9 0.00857 Preprocessor1_Model1
## 5 recall    macro      0.444     9 0.00870 Preprocessor1_Model1
## 6 roc_auc   hand_till  0.702     9 0.00552 Preprocessor1_Model1
## 7 sens      macro      0.444     9 0.00870 Preprocessor1_Model1
## 8 spec      macro      0.815     9 0.00290 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 0.504     9 0.00759 Preprocessor1_Model1
## 2 f_meas    macro      0.504     9 0.00780 Preprocessor1_Model1
## 3 kap       multiclass 0.339     9 0.0101  Preprocessor1_Model1
## 4 precision macro      0.511     9 0.00818 Preprocessor1_Model1
## 5 recall    macro      0.504     9 0.00759 Preprocessor1_Model1
## 6 roc_auc   hand_till  0.768     9 0.00490 Preprocessor1_Model1
## 7 sens      macro      0.504     9 0.00759 Preprocessor1_Model1
## 8 spec      macro      0.835     9 0.00253 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  multiclass 0.429     9 0.00949 Preprocessor1_Model1
## 2 f_meas    macro      0.426     9 0.00899 Preprocessor1_Model1
## 3 kap       multiclass 0.239     9 0.0126  Preprocessor1_Model1
## 4 precision macro      0.430     9 0.00932 Preprocessor1_Model1
## 5 recall    macro      0.429     9 0.00949 Preprocessor1_Model1
## 6 roc_auc   hand_till  0.703     9 0.00735 Preprocessor1_Model1
## 7 sens      macro      0.429     9 0.00949 Preprocessor1_Model1
## 8 spec      macro      0.810     9 0.00316 Preprocessor1_Model1
```

Compare performance

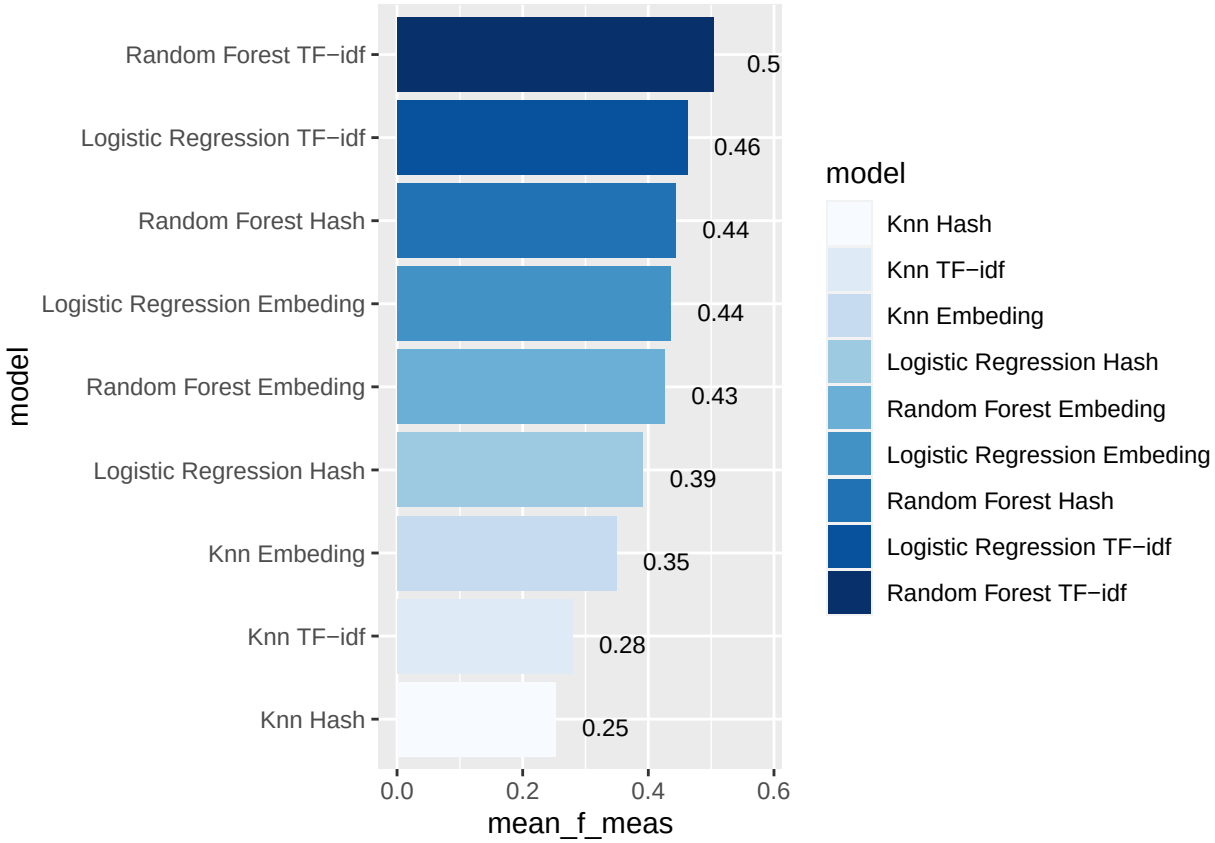
We get a summary for the performed models. We add the model name to each metric to keep the models appart from each other later on.

```
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")
```

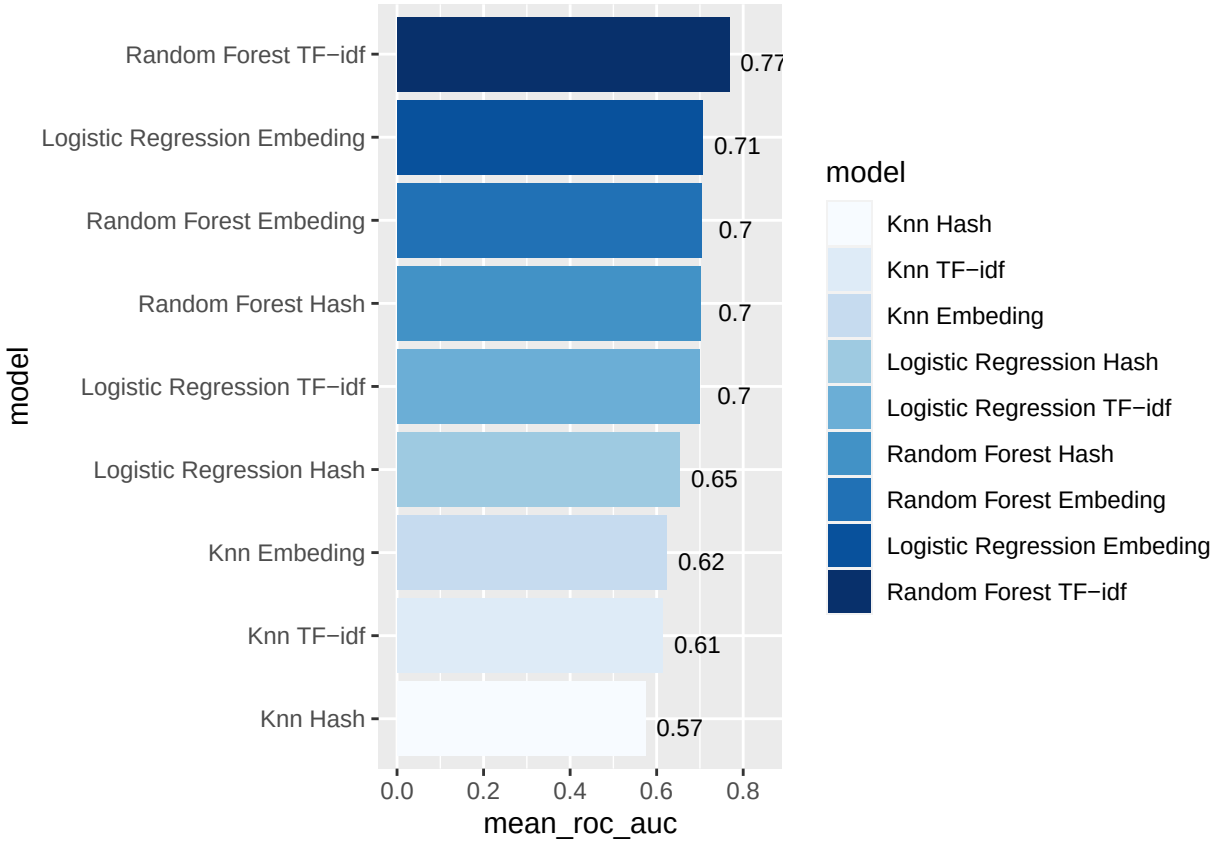
```
model_compare <- bind_rows(
                          log_metrics_tf,
                          log_metrics_emb,
                          log_metrics_hash,
                          rf_metrics_tf,
                          rf_metrics_emb,
                          rf_metrics_hash,
                          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:

```
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  multiclass 0.462     9 0.00893 Preprocessor1_Model1
## 2 f_meas    macro      0.462     9 0.00897 Preprocessor1_Model1
## 3 kap       multiclass 0.283     9 0.0119  Preprocessor1_Model1
```

```
## 4 precision macro      0.463     9 0.00899 Preprocessor1_Model1
## 5 recall    macro      0.462     9 0.00893 Preprocessor1_Model1
## 6 roc_auc   hand_till  0.700     9 0.00629 Preprocessor1_Model1
## 7 sens      macro      0.462     9 0.00893 Preprocessor1_Model1
## 8 spec      macro      0.821     9 0.00298 Preprocessor1_Model1
```

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

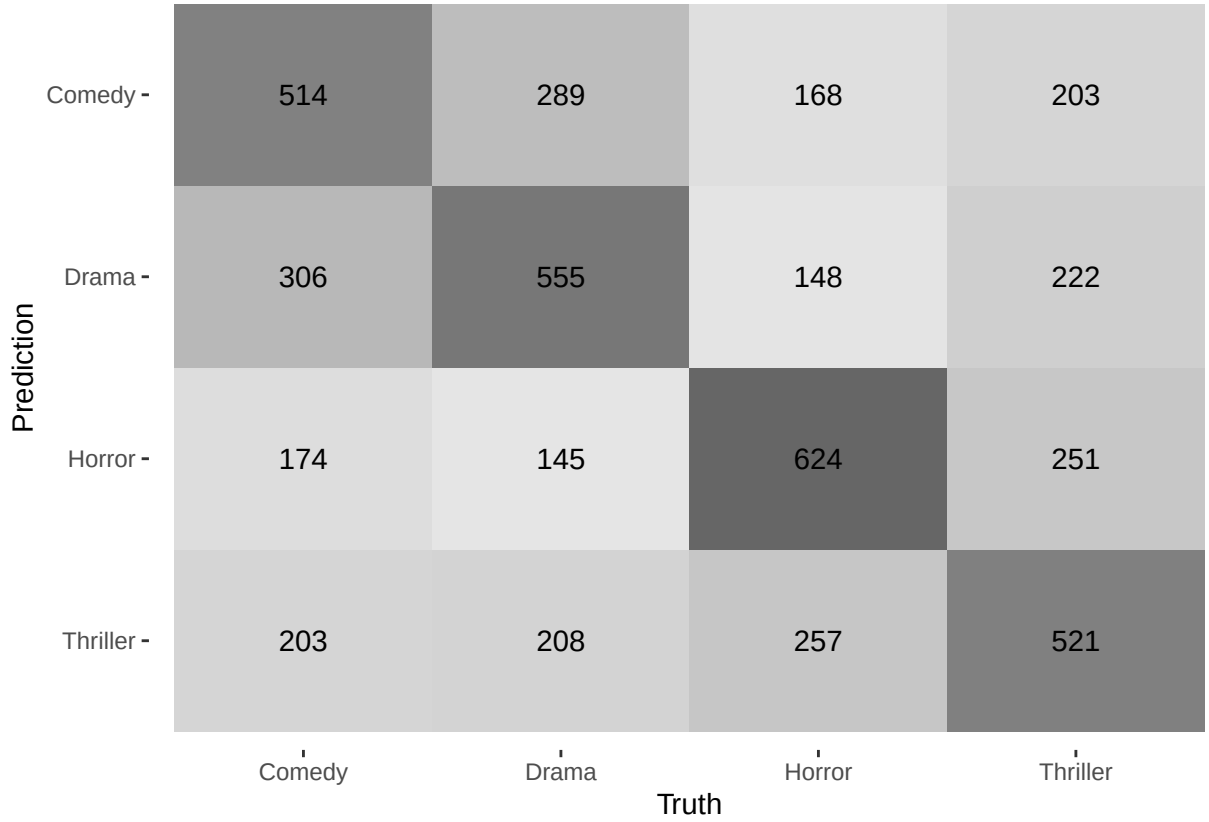
```
log_pred_tf <-
  log_res_tf %>%
  collect_predictions()
```

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

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

```
##            Truth
## Prediction Comedy Drama Horror Thriller
##   Comedy      514   289    168      203
##   Drama       306   555    148      222
##   Horror      174   145    624      251
##   Thriller    203   208    257      521
```

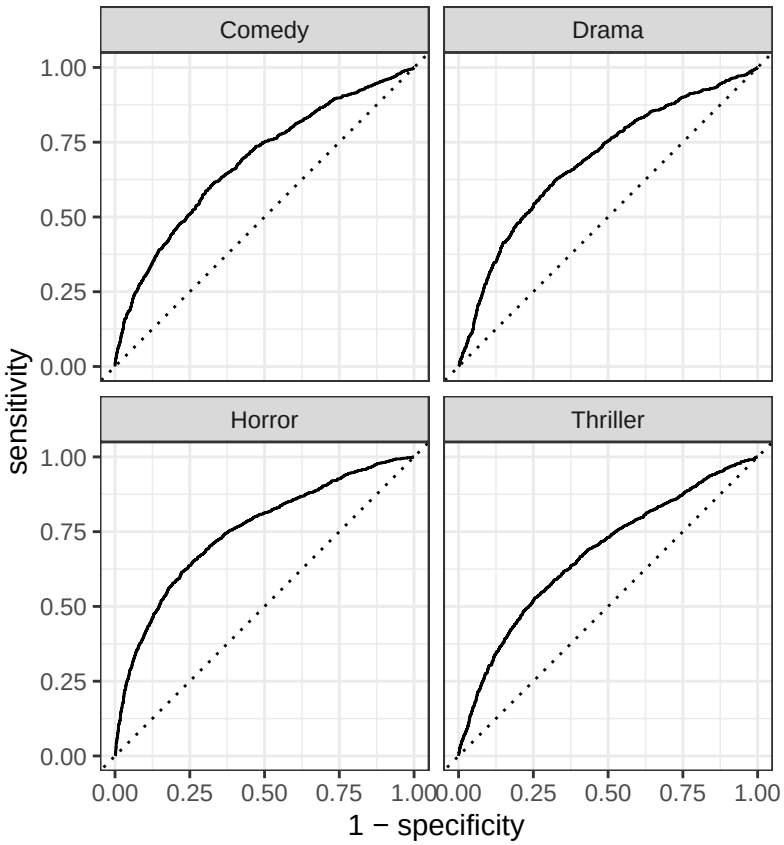
```
log_pred_tf %>%
  conf_mat(y, .pred_class) %>%
  autoplot(type = "heatmap")
```



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_Comedy:.pred_Thriller) %>%
  autoplot()
```



Random forest model

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

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

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

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

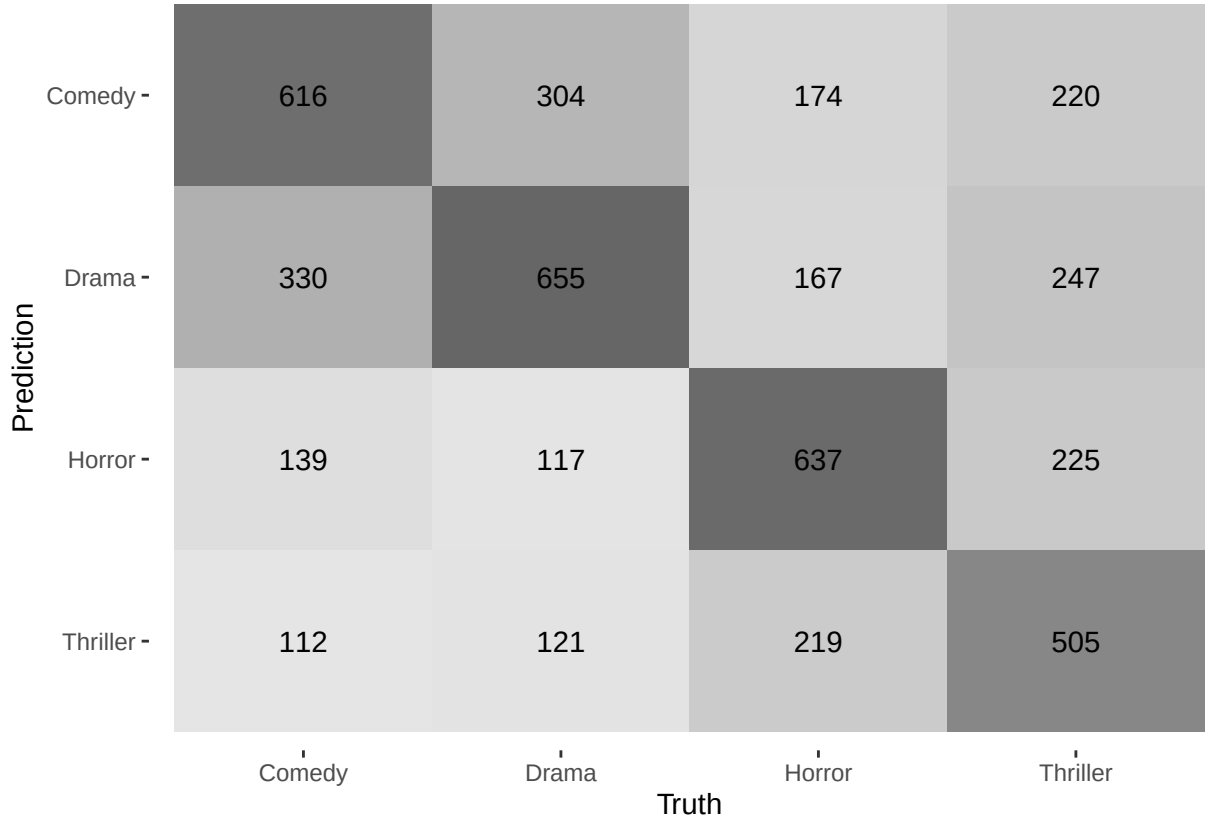
```
## # A tibble: 8 x 6
##   .metric   .estimator  mean     n std_err .config
##   <chr>     <chr>      <dbl> <int>   <dbl> <chr>
## 1 accuracy  multiclass 0.504     9 0.00759 Preprocessor1_Model1
## 2 f_meas    macro      0.504     9 0.00780 Preprocessor1_Model1
## 3 kap       multiclass 0.339     9 0.0101  Preprocessor1_Model1
## 4 precision macro      0.511     9 0.00818 Preprocessor1_Model1
## 5 recall    macro      0.504     9 0.00759 Preprocessor1_Model1
## 6 roc_auc   hand_till  0.768     9 0.00490 Preprocessor1_Model1
## 7 sens      macro      0.504     9 0.00759 Preprocessor1_Model1
## 8 spec      macro      0.835     9 0.00253 Preprocessor1_Model1
```

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

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

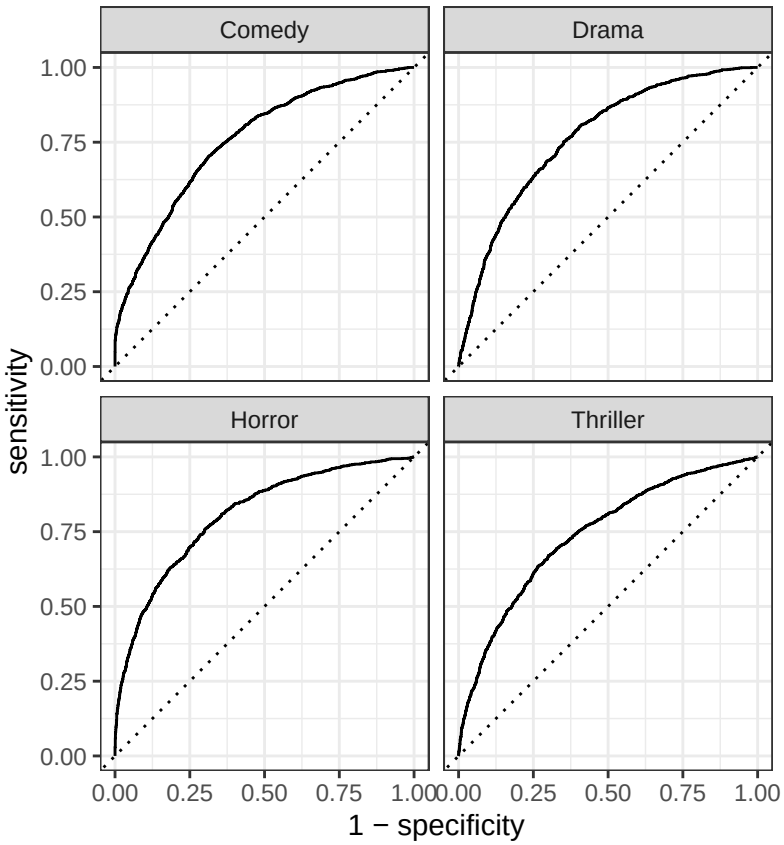
```
##            Truth
## Prediction Comedy Drama Horror Thriller
##   Comedy      616   304    174      220
##   Drama       330   655    167      247
##   Horror      139   117    637      225
##   Thriller    112   121    219      505
```

```
rf_pred_tf %>%
  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_tf %>%
  roc_curve(y, .pred_Comedy:.pred_Thriller) %>%
  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 = tidy_split,
                        metrics = metric_set(
                          recall, precision, f_meas,
                          accuracy, kap,
                          roc_auc, sens, spec)
                        )
```

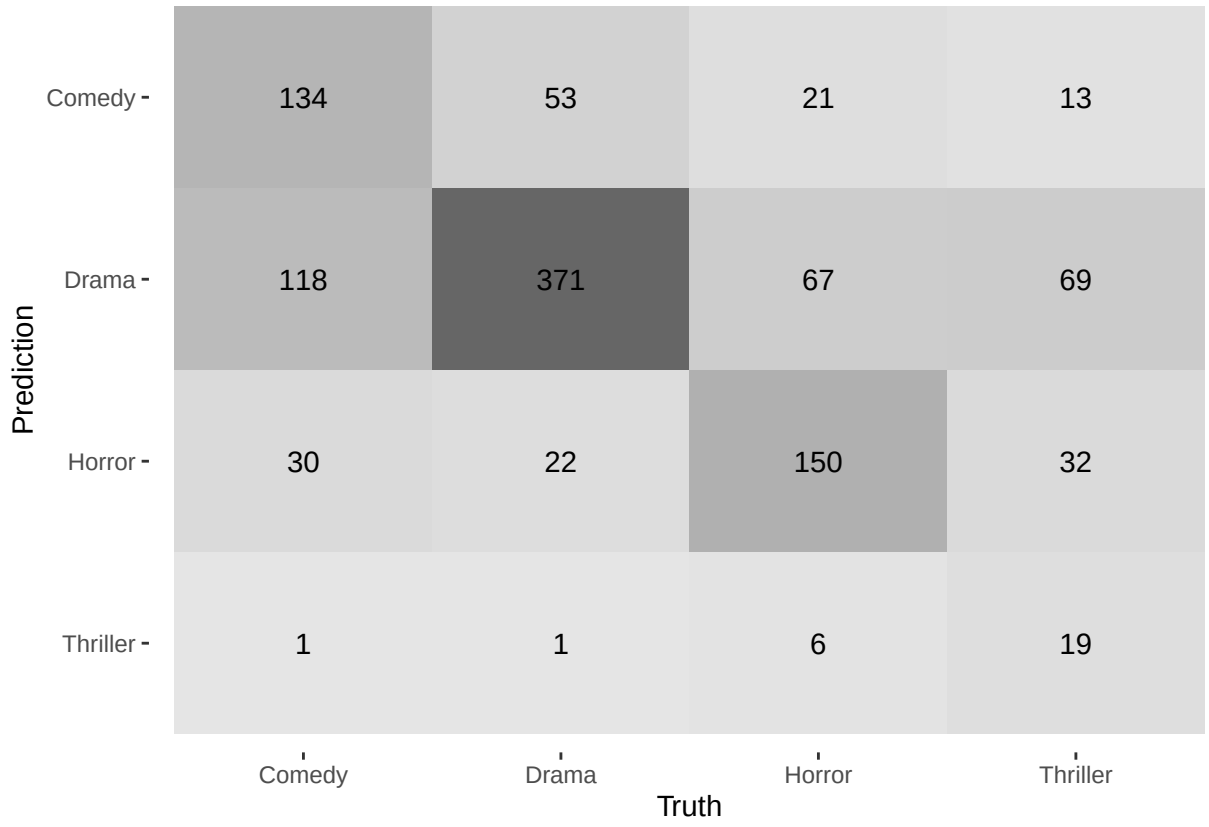
```
last_fit_rf %>%
  collect_metrics()
```

```
## # A tibble: 8 x 4
##   .metric   .estimator .estimate .config
##   <chr>     <chr>          <dbl> <chr>
## 1 recall    macro          0.515 Preprocessor1_Model1
## 2 precision macro          0.636 Preprocessor1_Model1
```

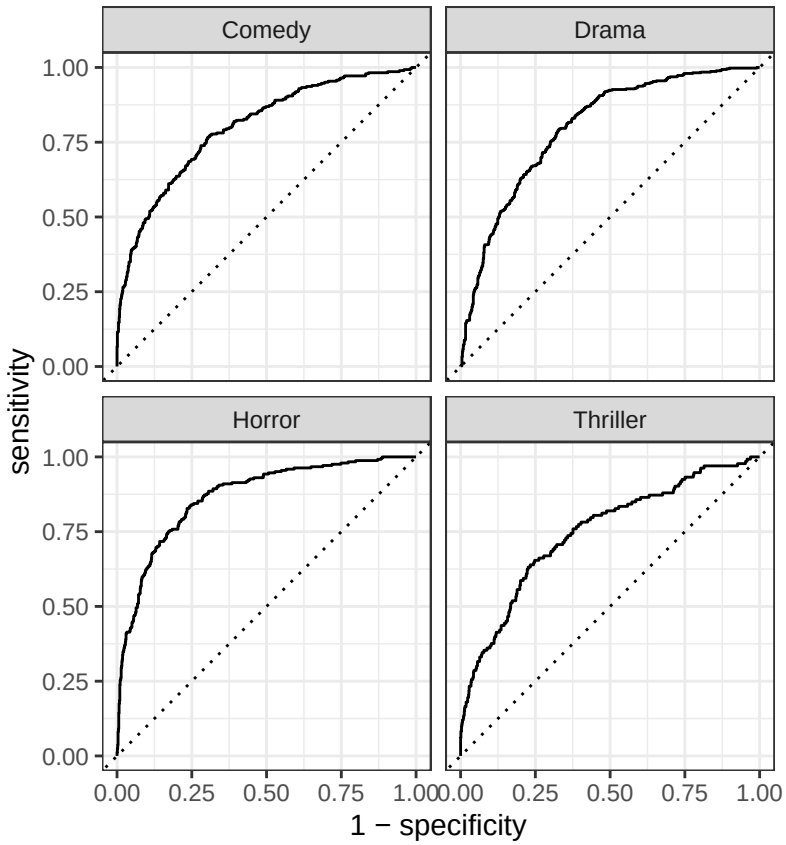
```
## 3 f_meas     macro          0.522 Preprocessor1_Model1
## 4 accuracy   multiclass     0.609 Preprocessor1_Model1
## 5 kap        multiclass     0.417 Preprocessor1_Model1
## 6 sens       macro          0.515 Preprocessor1_Model1
## 7 spec       macro          0.851 Preprocessor1_Model1
## 8 roc_auc    hand_till      0.797 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_Comedy:.pred_Thriller) %>%
  autoplot()
```



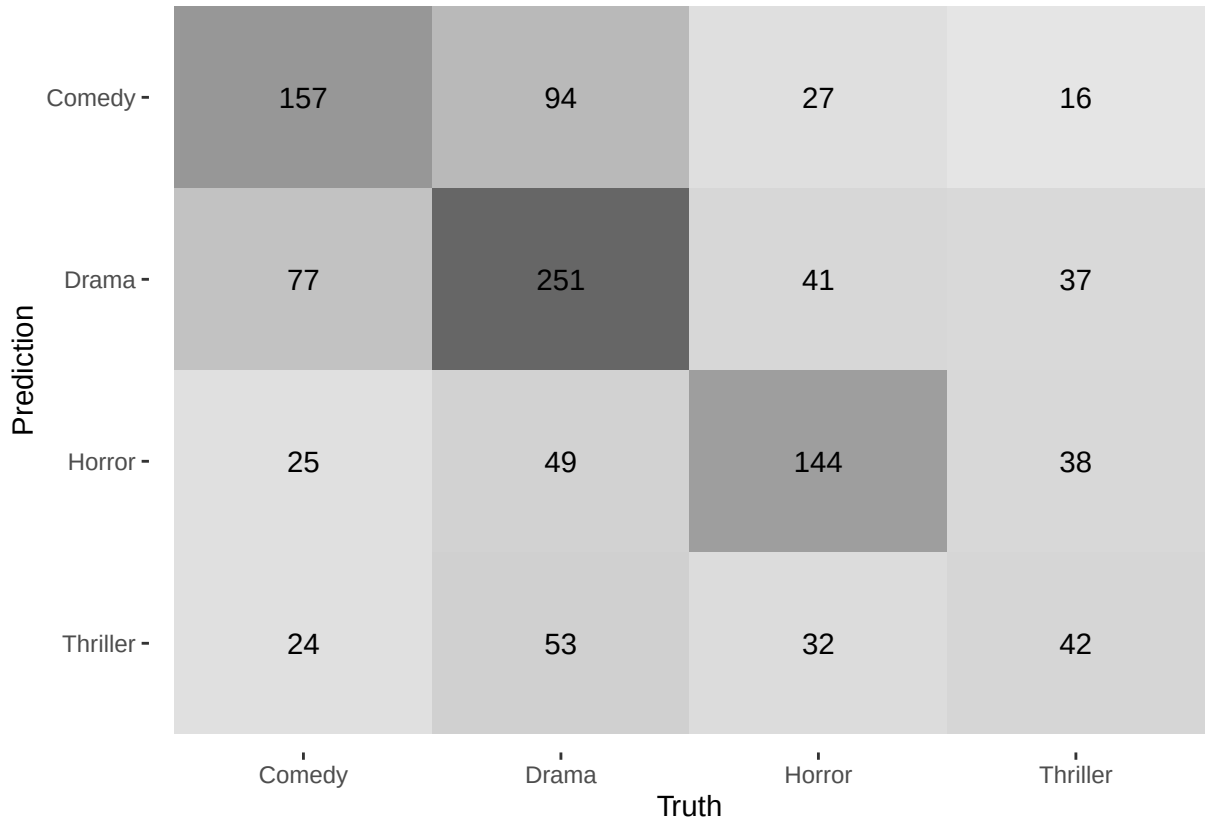
Logistic model

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

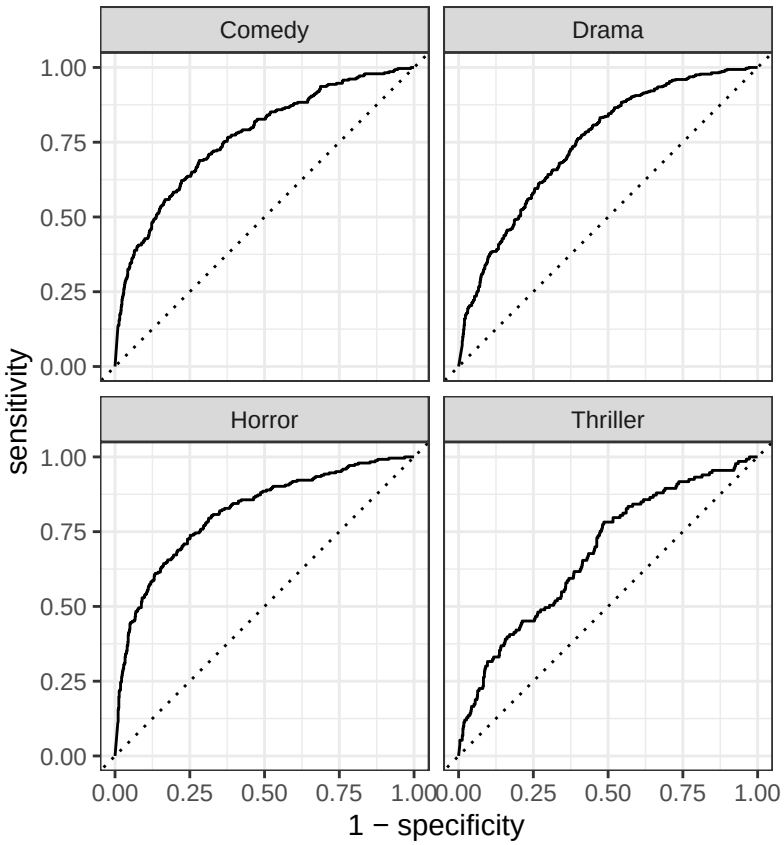
```
last_fit_log %>%
  collect_metrics()
```

```
## # A tibble: 8 x 4
##   .metric   .estimator .estimate .config
##   <chr>     <chr>          <dbl> <chr>
## 1 recall    macro          0.506 Preprocessor1_Model1
## 2 precision macro          0.498 Preprocessor1_Model1
## 3 f_meas    macro          0.501 Preprocessor1_Model1
## 4 accuracy  multiclass     0.537 Preprocessor1_Model1
## 5 kap       multiclass     0.353 Preprocessor1_Model1
## 6 sens      macro          0.506 Preprocessor1_Model1
## 7 spec      macro          0.839 Preprocessor1_Model1
## 8 roc_auc   hand_till      0.749 Preprocessor1_Model1
```

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




```
last_fit_log %>%
  collect_predictions() %>%
  roc_curve(y, .pred_Comedy:.pred_Thriller) %>%
  autoplot()
```



Embedding

We now want to use embedding data to do some machine learning and prediction using words and sentences.

Preprocessing Embedding

We start using the same preprocessing as we did on the genre data used for machine learning and networks.

```
data_emb %<>%
  drop_na()

text_tidy_emb = data_emb %>% unnest_tokens(word, text, token = "words")
```

```
library(hunspell)

text_tidy_emb %>%
  mutate(stem = hunspell_stem(word)) %>%
  unnest(stem) %>%
  count(stem, sort = TRUE)
```

```
## # A tibble: 7,860 x 2
##    stem      n
```

```
##    <chr> <int>
##  1 a      7533
##  2 the    5633
##  3 to     4185
##  4 of     3669
##  5 and    3209
##  6 in     2489
##  7 hi     1970
##  8 i      1704
##  9 h      1373
## 10 on     1245
## # ... with 7,850 more rows
```

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

We remove stopwords

```
text_tidy_emb %<>%
  anti_join(stop_words) %>%
  count(id, word)
```

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

We wheight by TF-idf

```
text_tidy_emb %<>%
  bind_tf_idf(term = word,
              document = id,
              n = n)
```

Embeding on imdb data

Joining our tidy tokenlist and the glove6b embeddings, which gives a list of mutual words within the two lists.

```
word_embeddings <- text_tidy_emb %>%
  inner_join(glove6b, by = c('word' = 'token'))%>%
  select(-c(tf, idf))
```

we can take a look at the new list.

```
word_embeddings %>% head()
```

```
## # A tibble: 6 x 104
##   id    word      n tf_idf      d1       d2      d3     d4      d5    d6      d7
##   <chr> <chr> <int>  <dbl>   <dbl>    <dbl>   <dbl>  <dbl>   <dbl> <dbl>   <dbl>
```

```
## 1 tt01~ bewi~     1  0.453 -0.263   0.167    0.0846  0.189 -0.384  0.209 -0.329
## 2 tt01~ bewi~     1  0.453 -0.520   0.577    0.629   0.367 -0.128  0.521 -0.154
## 3 tt01~ black     1  0.260 -0.0574  0.686    0.167  -0.874  0.158  0.827  0.146
## 4 tt01~ comic     1  0.307  0.0998 -0.00640 -0.293  -0.253  0.613  0.673 -0.0348
## 5 tt01~ demon     1  0.254  0.106  -0.304    0.943  -0.776 -0.0256 1.03  -0.0973
## 6 tt01~ faci~     1  0.348 -0.263   0.487    0.803   0.400  0.250  0.195 -1.05
## # ... with 93 more variables: d8 <dbl>, d9 <dbl>, 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>, d45 <dbl>, d46 <dbl>, d47 <dbl>,
## #   d48 <dbl>, d49 <dbl>, d50 <dbl>, d51 <dbl>, d52 <dbl>, d53 <dbl>,
## #   d54 <dbl>, d55 <dbl>, d56 <dbl>, d57 <dbl>, d58 <dbl>, d59 <dbl>,
## #   d60 <dbl>, d61 <dbl>, d62 <dbl>, d63 <dbl>, d64 <dbl>, d65 <dbl>,
## #   d66 <dbl>, d67 <dbl>, d68 <dbl>, d69 <dbl>, d70 <dbl>, d71 <dbl>,
## #   d72 <dbl>, d73 <dbl>, d74 <dbl>, d75 <dbl>, d76 <dbl>, d77 <dbl>,
## #   d78 <dbl>, d79 <dbl>, d80 <dbl>, d81 <dbl>, d82 <dbl>, d83 <dbl>,
## #   d84 <dbl>, d85 <dbl>, d86 <dbl>, d87 <dbl>, d88 <dbl>, d89 <dbl>,
## #   d90 <dbl>, d91 <dbl>, d92 <dbl>, d93 <dbl>, d94 <dbl>, d95 <dbl>,
## #   d96 <dbl>, d97 <dbl>, d98 <dbl>, d99 <dbl>, d100 <dbl>
```

We could also (even better) weight that by the word's tf-idf score. and group by id

```
doc_embeddings <- word_embeddings %>%
  group_by(id) %>%
  summarise(across(starts_with("d"), ~mean(.x / tf_idf, na.rm = TRUE)))
```

Umap

These embddings could now be used for instance for some clustering in relation to UML.so we want to do some unsupervised machine learning using UMAP

```
library(uwot) # for UMAP
```

We use left_join to label each movie id with the respective genre

```
doc_embeddings_genre= doc_embeddings %>%
  left_join(data_emb_genre, by= c("id"= "imdb_title_id")) %>%
  select(genre, everything())
```

We save the genres before we use UMAP to use it for collaring clusters later.

```
genre_class=doc_embeddings_genre[,1]
```

We reduce wordlist to two dimensions with UMAP.

```
embeddings_umap <- doc_embeddings  %>%
  column_to_rownames("id") %>%
  umap(n_neighbors = 15,
```

```
       metric = "cosine",
       min_dist = 0.01,
       scale = TRUE,
       verbose = TRUE,
       n_threads = 8)
```

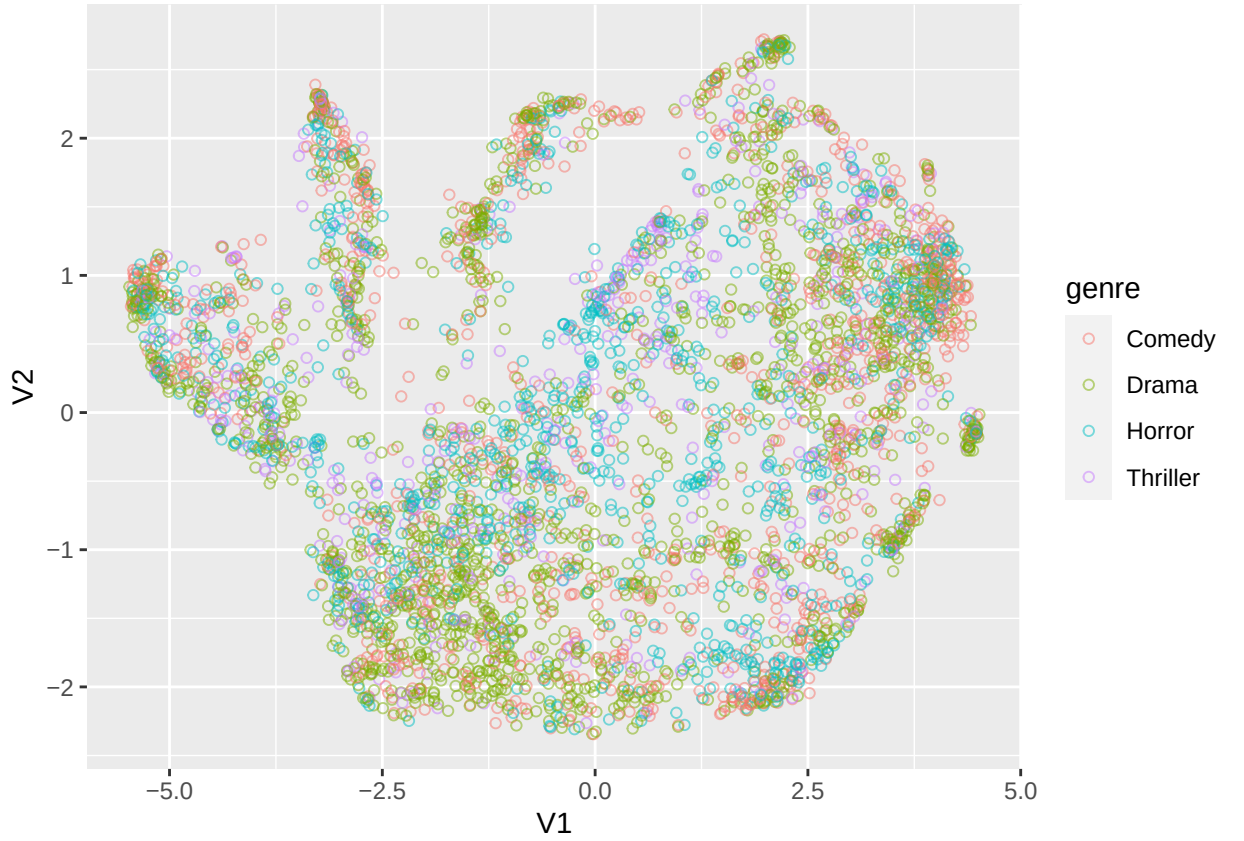
```
## 15:35:38 UMAP embedding parameters a = 1.896 b = 0.8006

## 15:35:38 Read 4429 rows and found 100 numeric columns

## 15:35:38 Scaling to zero mean and unit variance

## 15:35:38 Kept 100 non-zero-variance columns

## 15:35:38 Using Annoy for neighbor search, n_neighbors = 15

## 15:35:38 Building Annoy index with metric = cosine, n_trees = 50

## 0%   10   20   30   40   50   60   70   80   90   100%

## [----|----|----|----|----|----|----|----|----|----|

## **************************************************|
## 15:35:39 Writing NN index file to temp file C:\Users\SIMONI~1\AppData\Local\Temp\RtmpWCZX4q\file32bc
## 15:35:39 Searching Annoy index using 8 threads, search_k = 1500
## 15:35:40 Annoy recall = 100%
## 15:35:40 Commencing smooth kNN distance calibration using 8 threads
## 15:35:41 Initializing from normalized Laplacian + noise
## 15:35:41 Commencing optimization for 500 epochs, with 98610 positive edges
## 15:35:53 Optimization finished
```

We define these two dimensions as a data frame.

```
embeddings_umap %<>% as.data.frame()
```

We plot these two dimensions to see if they separate into clusters. we collor by genre to see if there is any clustering within genres.

```
embeddings_umap  %>%
  as_tibble() %>%
  bind_cols(genre_class) %>%
  ggplot(aes(x = V1, y = V2, col= genre)) +
  geom_point(shape = 21, alpha = 0.5)
```



We see that there doesnt seem to be any clustering based on the four genres.

We load the "dbscan" which is used to detect clusters within the data

```
library(dbscan)
```

```
##
## Attaching package: 'dbscan'
```

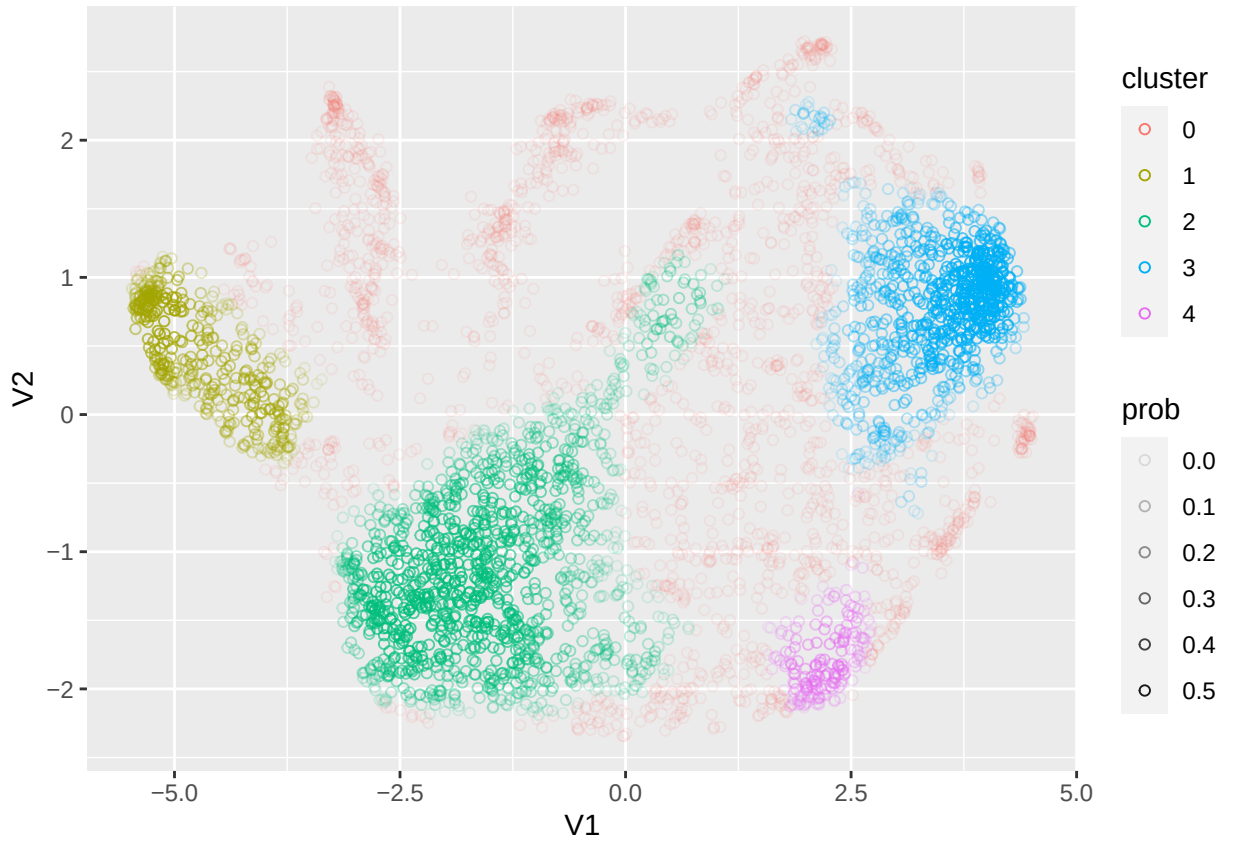
```
## The following object is masked from 'package:VIM':
##
##     kNN
```

We create clusters using hirarchical density based clustering we set minPts to 150 which means eaccch cluster should be at least 150 observations.

```
embeddings_hdbscan <- embeddings_umap %>% as.matrix() %>% hdbscan(minPts = 150)
```

We again use UMAP and color by the clusters created

```
embeddings_umap %>%
  bind_cols(cluster = embeddings_hdbscan$cluster %>% as.factor(),
            prob = embeddings_hdbscan$membership_prob) %>%
  ggplot(aes(x = V1, y = V2, col = cluster)) +
  geom_point(aes(alpha = prob), shape = 21)
```



We can see this looks way better.

Sml on embedded data

We now want to use supervised machine learning on the embedded data. we only want to predict wether the movie is within the Drama genre or Not.

```
data_class= data_imdb %>%
  select(imdb_title_id, genre)

data_class %>%
  count(genre, sort = T)
```

```
## # A tibble: 1,257 x 2
##    genre                      n
##    <chr>                  <int>
##  1 Drama                  12543
##  2 Comedy                  7693
##  3 Comedy, Drama           4039
##  4 Drama, Romance          3455
##  5 Comedy, Romance         2508
##  6 Comedy, Drama, Romance  2293
##  7 Horror                  2268
##  8 Drama, Thriller         1348
##  9 Crime, Drama            1343
```

```
## 10 Action, Crime, Drama    1310
## # ... with 1,247 more rows
```

```
library(naniar)
```

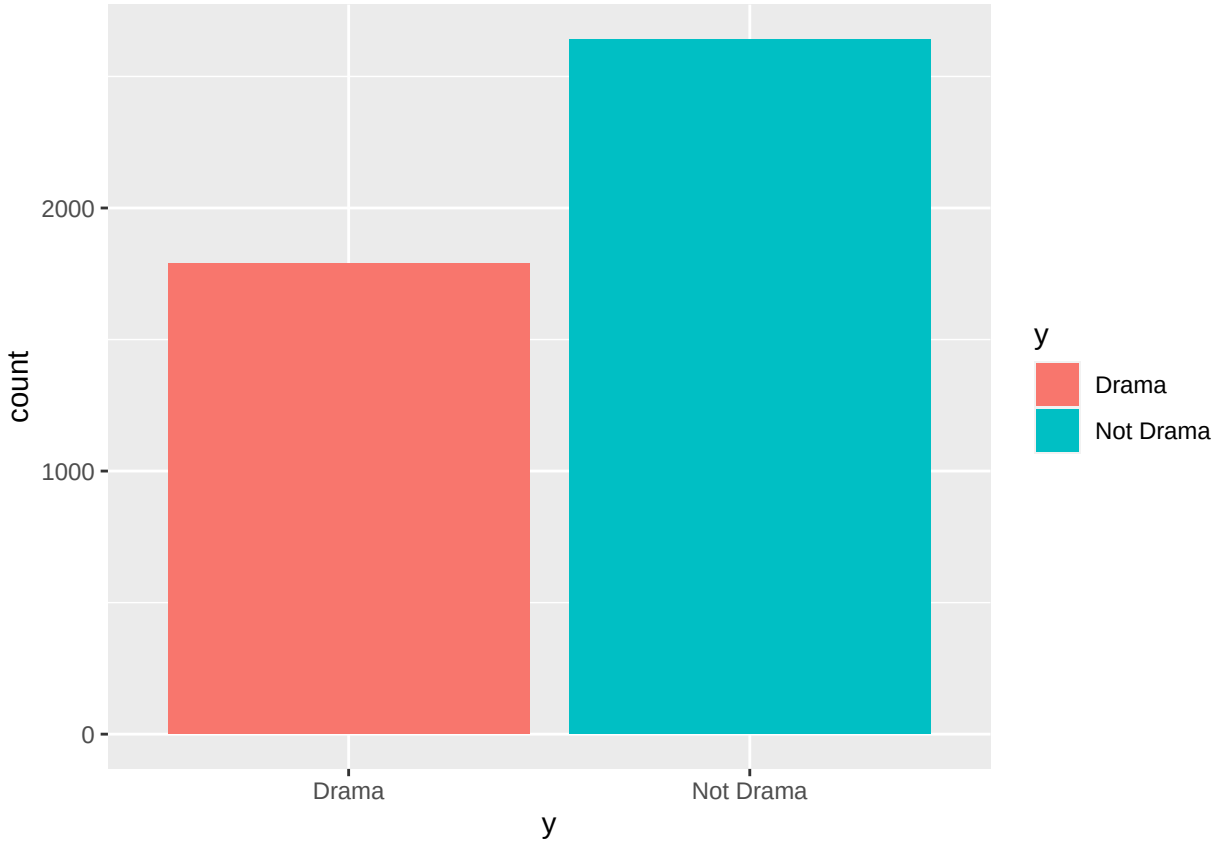
```
##
## Attaching package: 'naniar'

## The following object is masked from 'package:skimr':
##
##     n_complete
```

```
test_data=data_class %>%
  filter(genre == "Drama")
```

We can now left join with id so we get the embeddings for every id and genre, keep in mind there will be the same description (embedings) for some genre observations as they are classified with more genres.

```
new_data_sml_class=doc_embeddings %>%
  left_join(test_data, by= c("id"= "imdb_title_id"))%>%
  select(genre, everything()) %>%
  replace_na(replace = list(genre= "Not Drama"))%>%
  distinct(.keep_all = T) %>% #remove dublicans
  rename(y= genre) %>%
  select(-id)
```

```
new_data_sml_class %>%
  ggplot(aes(y, fill= y)) +
  geom_bar()
```



We can see the two classes are not evenly represented. so we can use the strata argument to keep the same ratio in the training and test dataset.

```
set.seed(123)

data_split <- initial_split(new_data_sml_class, prop = 0.75, strata = y)

data_train <- data_split  %>%  training()
data_test <- data_split %>% testing()
```

We center the data to mean = 0, and scale the data to have an Standard deviation equal to 1.

```
data_recipe <- data_train %>%
  recipe(y ~.) %>%
  step_center(all_numeric(), -all_outcomes()) %>%
  step_scale(all_numeric(), -all_outcomes()) %>%
  #step_dummy(all_nominal(), -all_outcomes()) %>%
  prep()


summary(data_recipe)
```

```
## # A tibble: 101 x 4
##    variable type    role      source
##    <chr>    <chr>   <chr>     <chr>
##  1 d1       numeric predictor original
```

```
##  2 d2       numeric predictor original
##  3 d3       numeric predictor original
##  4 d4       numeric predictor original
##  5 d5       numeric predictor original
##  6 d6       numeric predictor original
##  7 d7       numeric predictor original
##  8 d8       numeric predictor original
##  9 d9       numeric predictor original
## 10 d10      numeric predictor original
## # ... with 91 more rows
```

Defining the models

We now define the four models we end up looking at.

Logistic Regression

```
model_lg <- logistic_reg(mode = 'classification') %>%
  set_engine('glm', family = binomial)
```

Decision tree

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

K-nearest neighbor

```
model_knn <-
  nearest_neighbor(neighbors = 4) %>%
  set_engine("kknn") %>%
  set_mode("classification")
```

Random forest

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

Define workflow

We will now define the workflow of the model by adding first the recipe to a general workflow, and then using this to create a workflow for each model.

```
workflow_general <- workflow() %>%
  add_recipe(data_recipe)

workflow_lg <- workflow_general %>%
  add_model(model_lg)

workflow_dt <- workflow_general %>%
  add_model(model_dt)

workflow_knn <- workflow_general %>%
  add_model(model_knn)

workflow_rf <- workflow_general %>%
  add_model(model_rf)
```

Hyperparameter Tuning

As the parameters in the decision tree and XGBoost model are set to tune(), we will now find the optimal values for the parameters.

Validation Sampling (N-fold crossvlidation)

We use k-fold crossvalidation to build a set of 5 validation folds with the function vfold_cv. We also use stratified sampling by setting the strata argument to y. We set repeats equal to 3. We dont have to use boosttraps as we have enogh observations.

```
set.seed(100)

data_resample <- data_train %>%
  vfold_cv(strata = y,
           v = 3,
           repeats = 3)
```

Define Grids

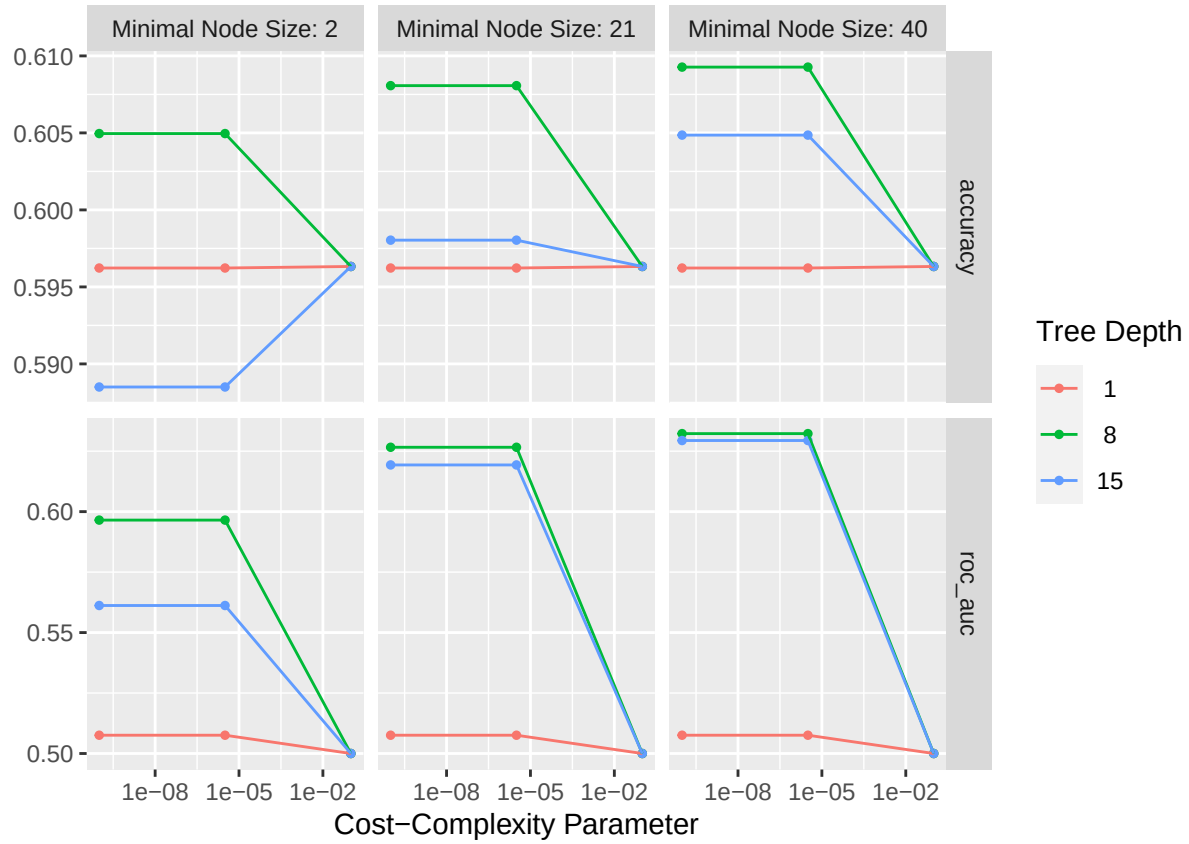
```
dt_grid <- grid_regular(parameters(model_dt), levels = 3)
```

Hyperparameter Tuning: Decision Tree

First we tune the decision tree, using the tune_grid function where we first specify the workflow, next we give it the resampled data, and last the grid is set to give different versions of every tuneable parameters.

```
tune_dt <-
  tune_grid(
    workflow_dt,
    resamples = data_resample,
    grid = dt_grid
  )
```

```
tune_dt %>% autoplot()
```




```
best_param_dt <- tune_dt %>% select_best(metric = 'roc_auc')
best_param_dt
```

```
## # A tibble: 1 x 4
##   cost_complexity tree_depth min_n .config
##             <dbl>      <int> <int> <chr>
## 1    0.0000000001          8    40 Preprocessor1_Model22
```

```
tune_dt %>% show_best(metric = 'roc_auc', n = 1)
```

```
## # A tibble: 1 x 9
##   cost_complexity tree_depth min_n .metric .estimator  mean     n std_err
##             <dbl>      <int> <int> <chr>   <chr>      <dbl> <int>   <dbl>
## 1    0.0000000001          8    40 roc_auc binary     0.632     9 0.00488
## # ... with 1 more variable: .config <chr>
```

Fit models with tuned hyperparameters

We now fit the best parameters into the workflow of the model.

```
workflow_final_dt <- workflow_dt %>%
  finalize_workflow(parameters = best_param_dt)
```

Evaluate models

here we us the resampled data to evaluate the models.

Logistic regression

We use our workflow object to perform resampling. Furthermore, we use metric_set() to choose some common classification performance metrics provided by the yardstick package. Visit yardsticks reference to see the complete list of all possible metrics.

Note that Cohen's kappa coefficient ( ) is a similar measure to accuracy, but is normalized by the accuracy that would be expected by chance alone and is very useful when one or more classes have large frequency distributions. The higher the value, the better.

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

log_res %>% 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     0.715     9 0.00527 Preprocessor1_Model1
## 2 f_meas    binary     0.627     9 0.00763 Preprocessor1_Model1
## 3 kap       binary     0.398     9 0.0115  Preprocessor1_Model1
## 4 precision binary     0.666     9 0.00700 Preprocessor1_Model1
## 5 recall    binary     0.592     9 0.00883 Preprocessor1_Model1
## 6 roc_auc   binary     0.782     9 0.00413 Preprocessor1_Model1
## 7 sens      binary     0.592     9 0.00883 Preprocessor1_Model1
## 8 spec      binary     0.799     9 0.00441 Preprocessor1_Model1
```

Decision tree

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

dt_res %>% 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     0.609     9 0.00238 Preprocessor1_Model1
## 2 f_meas    binary     0.508     9 0.0103  Preprocessor1_Model1
## 3 kap       binary     0.184     9 0.00774 Preprocessor1_Model1
## 4 precision binary     0.516     9 0.00311 Preprocessor1_Model1
## 5 recall    binary     0.502     9 0.0187  Preprocessor1_Model1
## 6 roc_auc   binary     0.632     9 0.00488 Preprocessor1_Model1
## 7 sens      binary     0.502     9 0.0187  Preprocessor1_Model1
## 8 spec      binary     0.682     9 0.0116  Preprocessor1_Model1
```

KNN

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

knn_res %>% 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     0.604     9 0.00512 Preprocessor1_Model1
## 2 f_meas    binary     0.537     9 0.00524 Preprocessor1_Model1
## 3 kap       binary     0.193     9 0.00977 Preprocessor1_Model1
## 4 precision binary     0.508     9 0.00572 Preprocessor1_Model1
## 5 recall    binary     0.570     9 0.00623 Preprocessor1_Model1
## 6 roc_auc   binary     0.661     9 0.00519 Preprocessor1_Model1
## 7 sens      binary     0.570     9 0.00623 Preprocessor1_Model1
## 8 spec      binary     0.627     9 0.00727 Preprocessor1_Model1
```

Random forrest

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

rf_res %>%  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     0.697     9 0.00407 Preprocessor1_Model1
## 2 f_meas    binary     0.538     9 0.00859 Preprocessor1_Model1
## 3 kap       binary     0.330     9 0.00967 Preprocessor1_Model1
## 4 precision binary     0.699     9 0.00754 Preprocessor1_Model1
## 5 recall    binary     0.438     9 0.0103  Preprocessor1_Model1
## 6 roc_auc   binary     0.757     9 0.00397 Preprocessor1_Model1
## 7 sens      binary     0.438     9 0.0103  Preprocessor1_Model1
## 8 spec      binary     0.872     9 0.00509 Preprocessor1_Model1
```

Compare performance

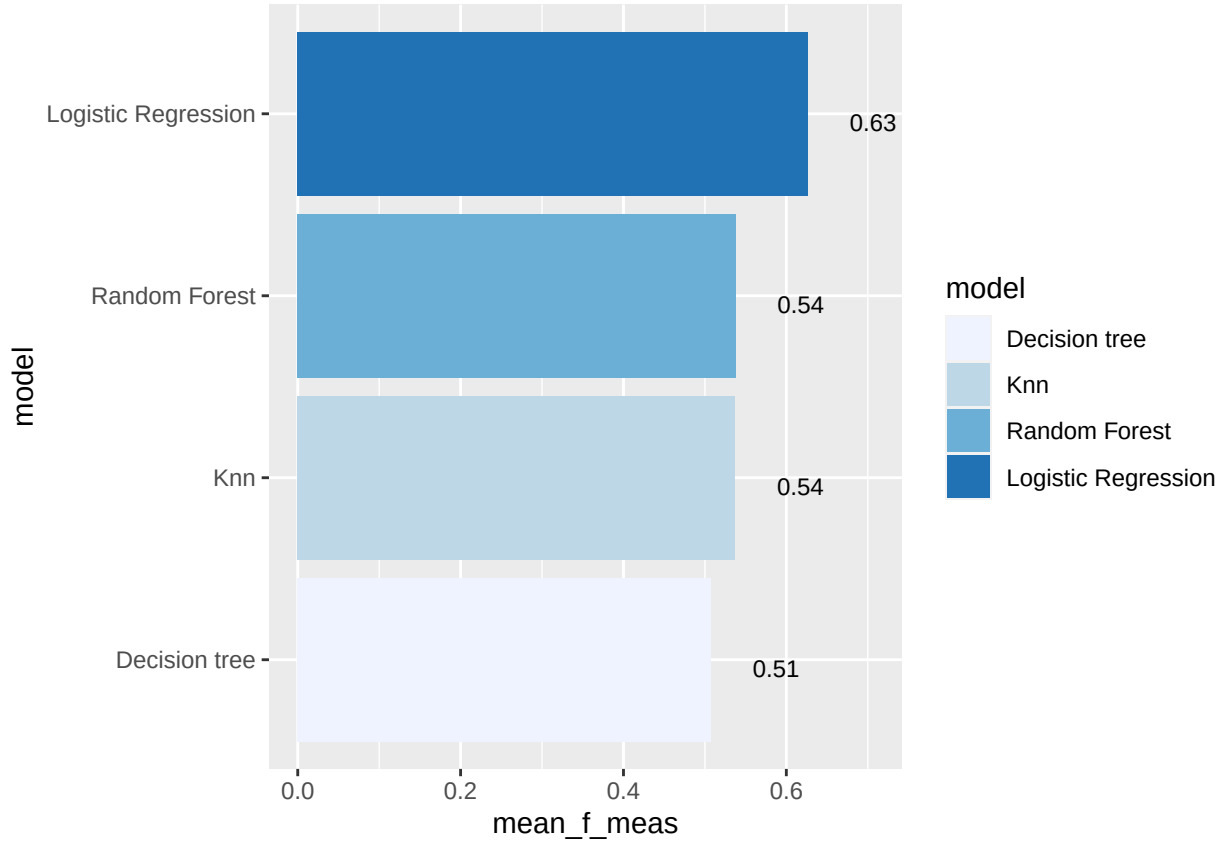
We get a summary for the performed models. We add the model name to each metric to keep the models appart from each other later on.

```
log_metrics <-
  log_res %>%
  collect_metrics(summarise = TRUE) %>%
  mutate(model = "Logistic Regression")

rf_metrics <-
  rf_res %>%
  collect_metrics(summarise = TRUE) %>%
  mutate(model = "Random Forest")

knn_metrics <-
  knn_res %>%
  collect_metrics(summarise = TRUE) %>%
  mutate(model = "Knn")

dt_metrics <-
  dt_res %>%
  collect_metrics(summarise = TRUE) %>%
  mutate(model = "Decision tree")
```

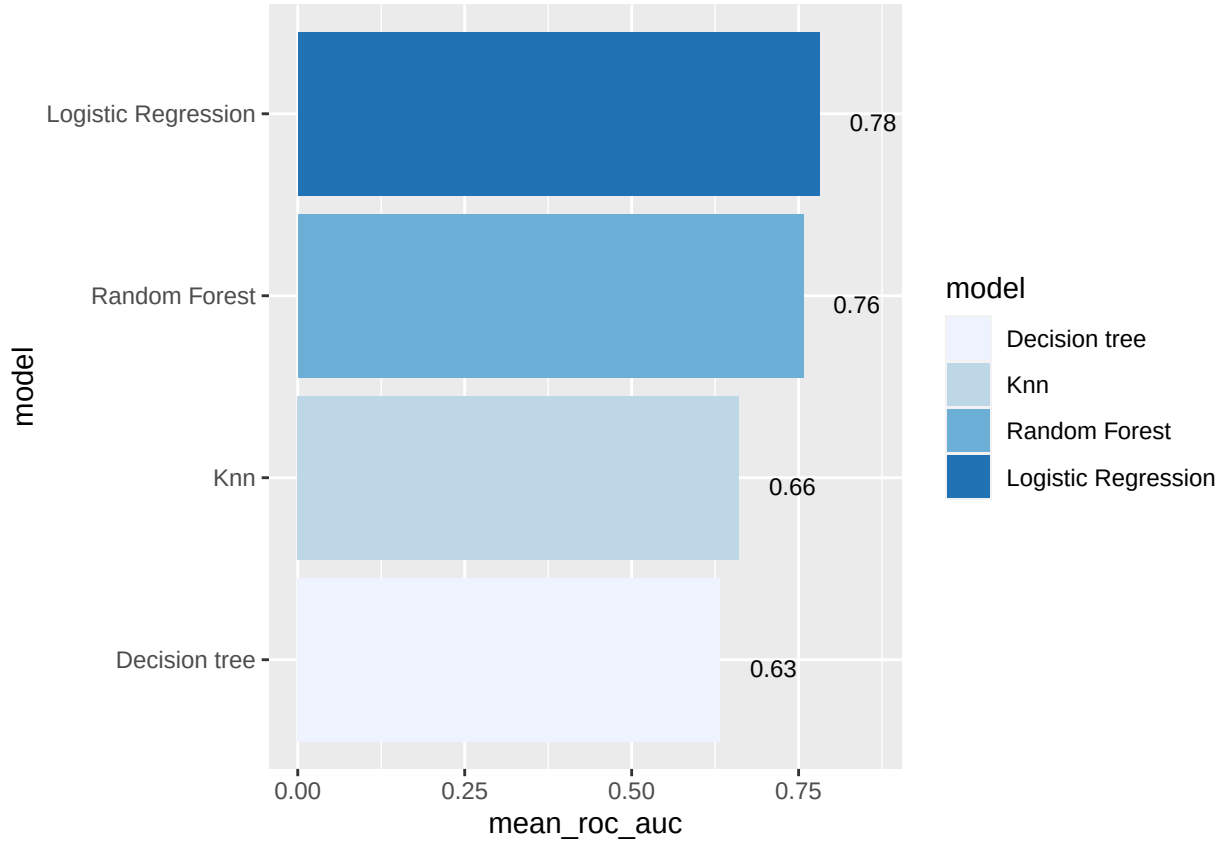
We now bind the rows for the above metricies and create a dataframe, we then change the data structure and show the mean_f_meas score for each model which include the precision and recall score.

Precision quantifies the number of positive class predictions that actually belong to the positive class. Recall quantifies the number of positive class predictions made out of all positive examples in the dataset. F-Measure provides a single score that balances both the concerns of precision and recall in one number.

```
model_compare <- bind_rows(
                          log_metrics,
                           rf_metrics,
                           knn_metrics,
                          dt_metrics,
                           )


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

We choose the Logistic regression model

Logistic regression model

Performance metrics

Show average performance over all folds.

```
log_res %>%  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     0.715     9 0.00527 Preprocessor1_Model1
## 2 f_meas    binary     0.627     9 0.00763 Preprocessor1_Model1
## 3 kap       binary     0.398     9 0.0115  Preprocessor1_Model1
## 4 precision binary     0.666     9 0.00700 Preprocessor1_Model1
## 5 recall    binary     0.592     9 0.00883 Preprocessor1_Model1
## 6 roc_auc   binary     0.782     9 0.00413 Preprocessor1_Model1
## 7 sens      binary     0.592     9 0.00883 Preprocessor1_Model1
## 8 spec      binary     0.799     9 0.00441 Preprocessor1_Model1
```

Collect model predictions

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

```
log_pred <-
  log_res %>%
  collect_predictions()
```

Confusion Matrix

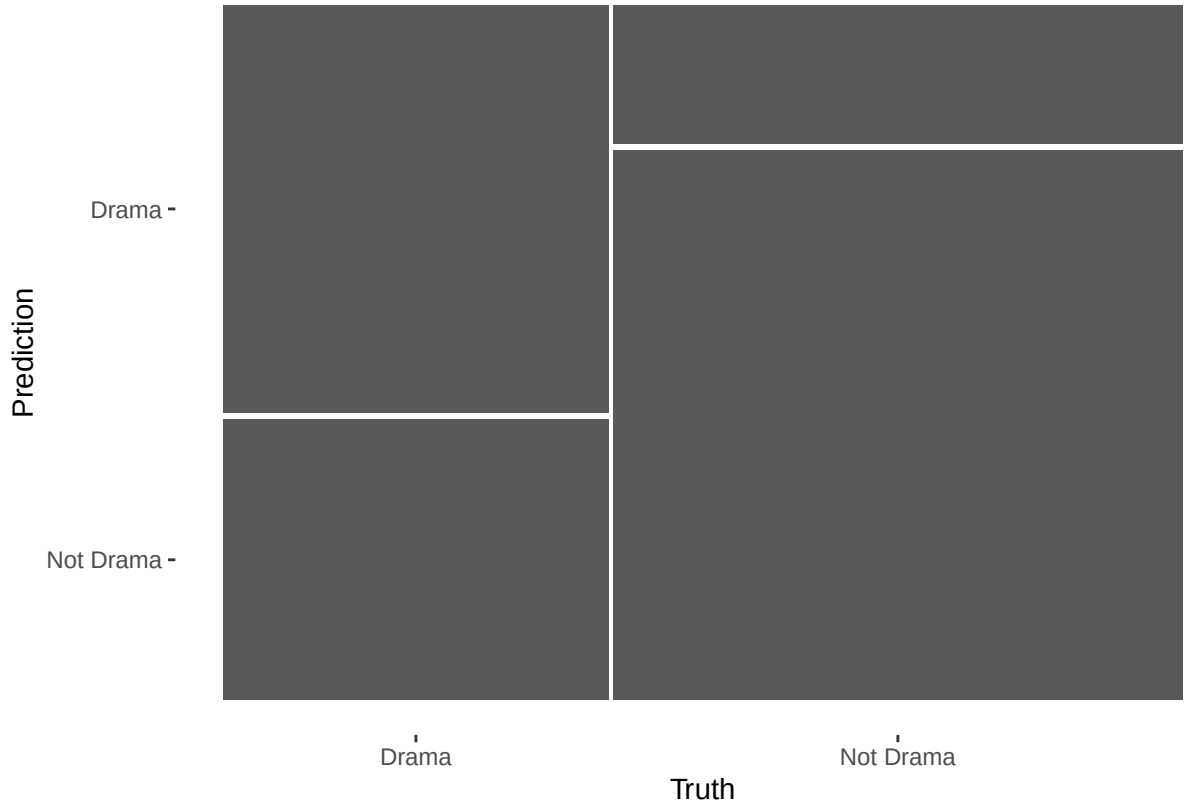
We can now use our collected predictions to make a confusion matrix

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

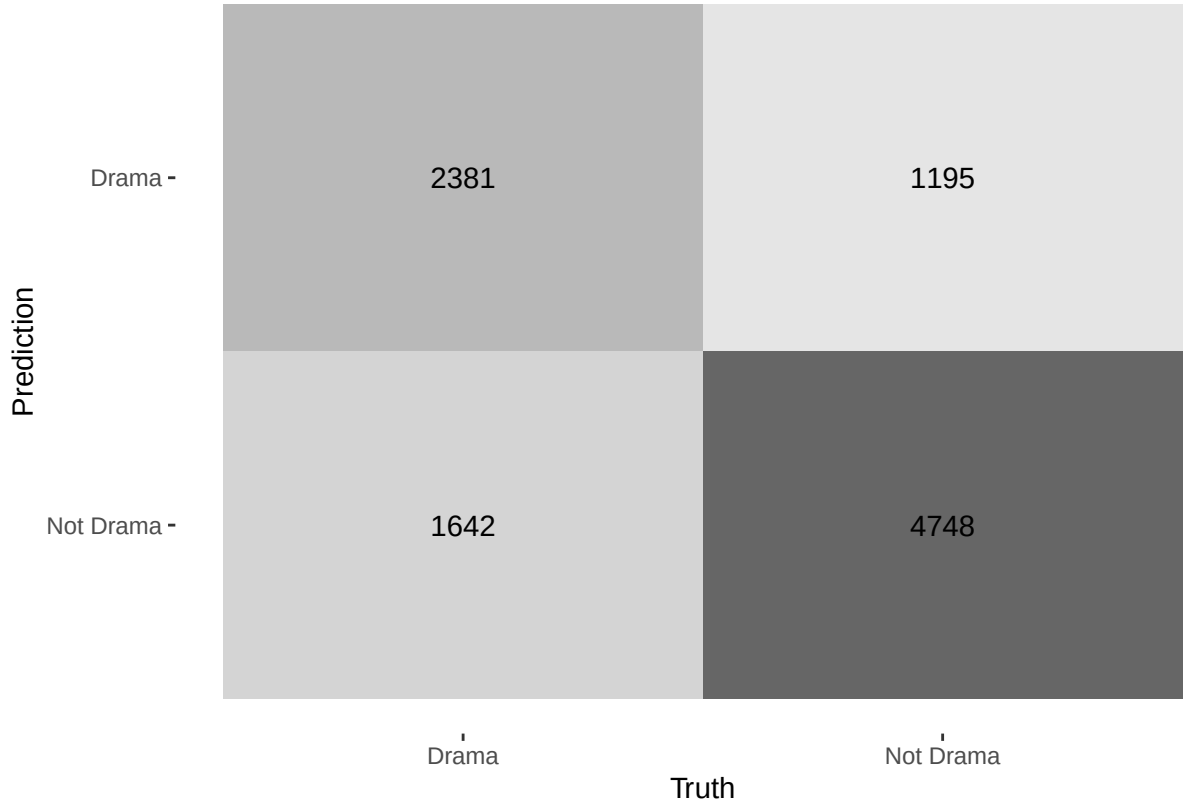
```
##            Truth
## Prediction  Drama Not Drama
##   Drama      2381      1195
##   Not Drama  1642      4748
```

And visualize it again

```
log_pred %>%
  conf_mat(y, .pred_class) %>%
  autoplot(type = "mosaic")
```



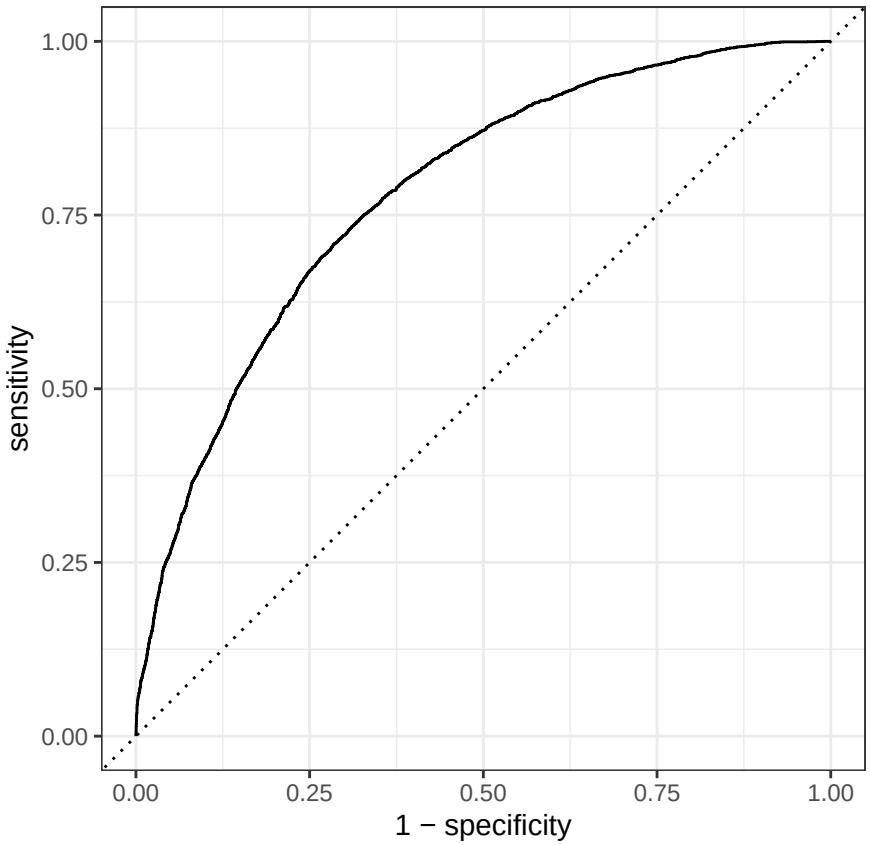
```
log_pred %>%
  conf_mat(y, .pred_class) %>%
  autoplot(type = "heatmap")
```



ROC curve

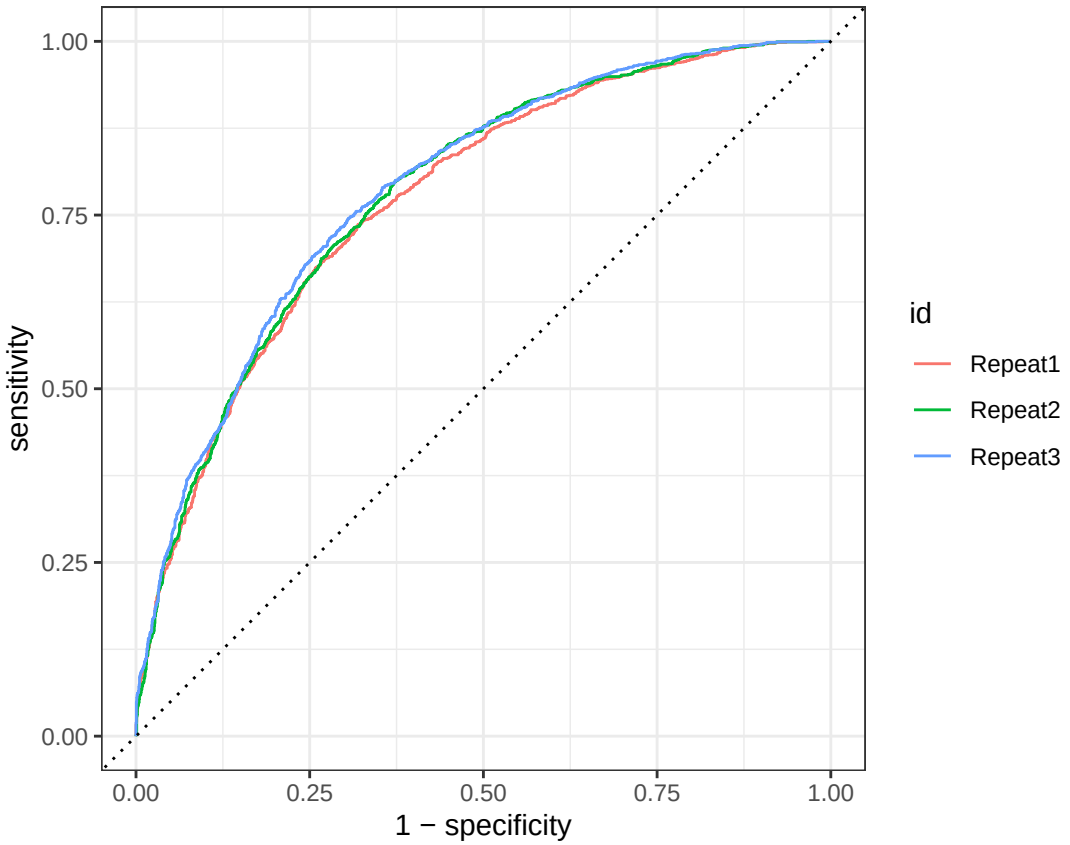
We will now create the ROC curve with 1 - specificity on the x-axis (false positive fraction = FP/(FP+TN)) and sensitivity on the y axis (true positive fraction = TP/(TP+FN)).

```
log_pred %>%
  roc_curve(y, .pred_Drama) %>%
  autoplot()
```



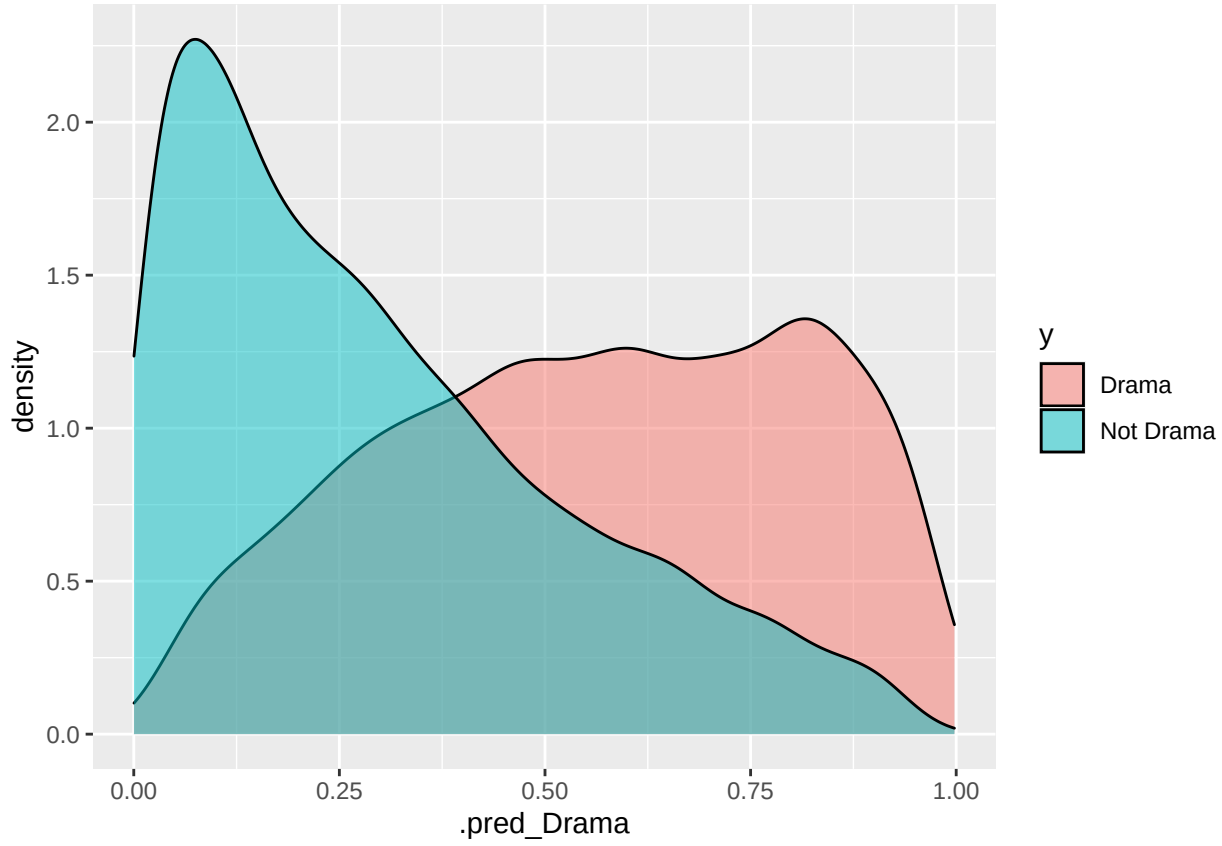
And now using the folds again.

```
log_pred %>%
  group_by(id) %>%
    roc_curve(y, .pred_Drama) %>%
  autoplot()
```



We again show the probability distributions for our two classes.

```
log_pred %>%
  ggplot() +
  geom_density(aes(x = .pred_Drama,
                   fill = y),
               alpha = 0.5)
```



On test data

We now use the test data by setting the split argument equal to data_split.

```
last_fit_log <- last_fit(workflow_lg,
                         split = data_split,
                         metrics = metric_set(
                           recall, precision, f_meas,
                           accuracy, kap,
                           roc_auc, sens, spec)
                         )
```

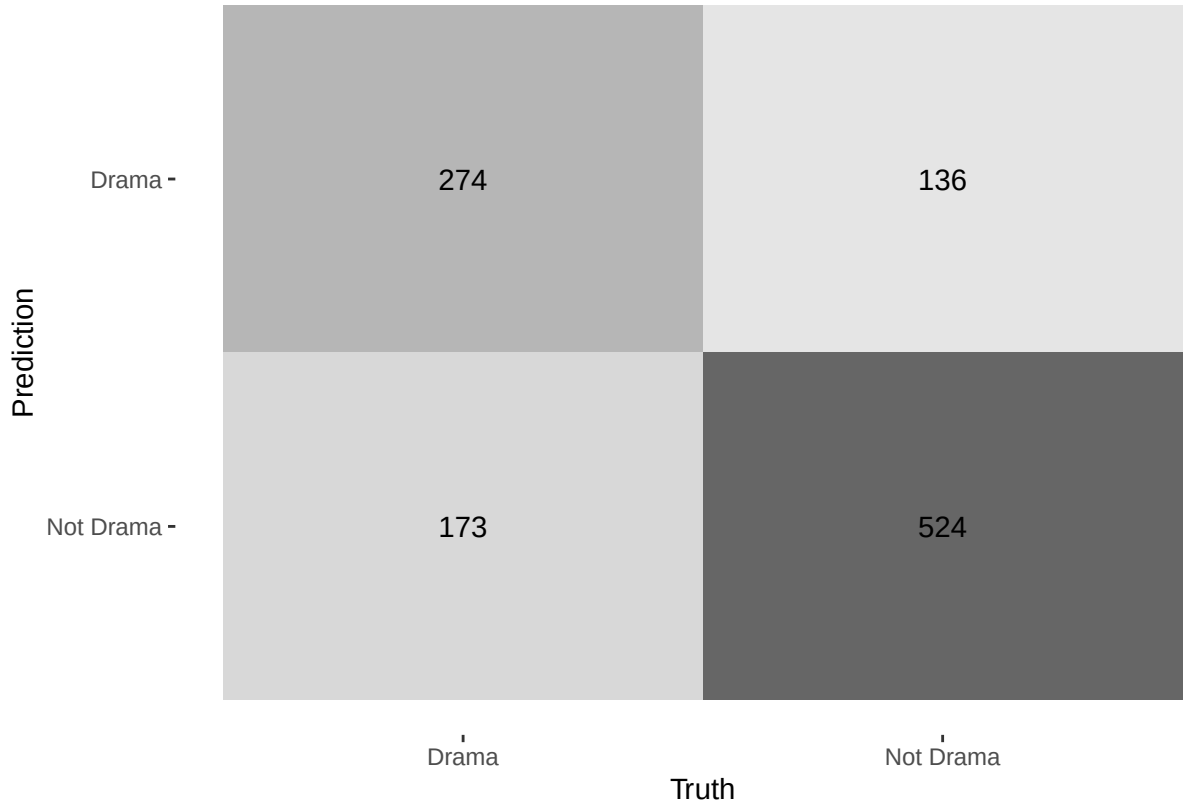
```
last_fit_log %>%
  collect_metrics()
```

```
## # A tibble: 8 x 4
##   .metric   .estimator .estimate .config
##   <chr>     <chr>          <dbl> <chr>
## 1 recall    binary         0.613 Preprocessor1_Model1
## 2 precision binary         0.668 Preprocessor1_Model1
## 3 f_meas    binary         0.639 Preprocessor1_Model1
## 4 accuracy  binary         0.721 Preprocessor1_Model1
## 5 kap       binary         0.412 Preprocessor1_Model1
## 6 sens      binary         0.613 Preprocessor1_Model1
```

```
## 7 spec      binary         0.794 Preprocessor1_Model1
## 8 roc_auc   binary         0.792 Preprocessor1_Model1
```

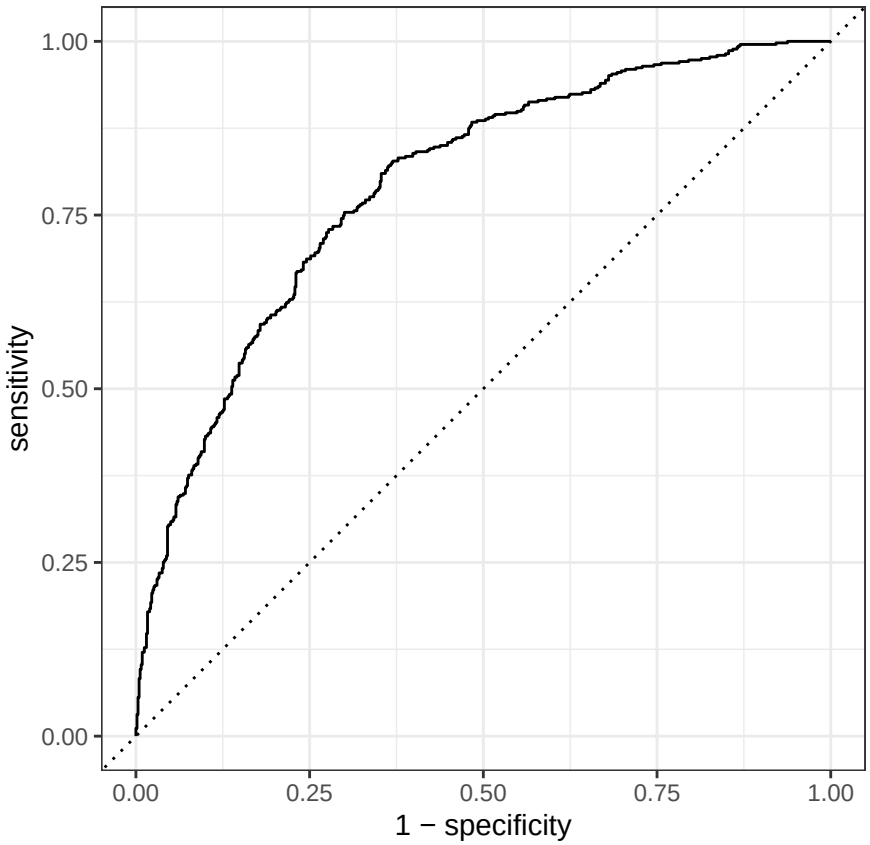
We can also make the confusion matrix using the test data

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



And lastly show the ROC curve using the test data.

```
last_fit_log %>%
  collect_predictions() %>%
  roc_curve(y, .pred_Drama) %>%
  autoplot()
```



Similarity: doc2vec

We want to use doc2vec to predict similar words/documents from a word/document/sentence.

We follow the recipe:

- Take some data and standardise it a bit.
- Make sure it has columns doc_id and text
- Make sure that each text has less than 1000 words (a word is considered separated by a single space)
- Make sure that each text does not contain newline symbols

```
data_nlp = data_imdb %>%
  rename(text = description) %>%
  filter(language == "English")%>%
  rename(id= imdb_title_id) %>%
  select(id, text)
```

We do some dat manipulation and create a subset to be used in the model

```
library(doc2vec)
library(udpipe)
x <- data.frame(doc_id = data_nlp$id,
                text   = data_nlp$text,
                stringsAsFactors = FALSE)
```

```
x$text   <- tolower(x$text)
x$text   <- gsub("[^[:alpha:]]", " ", x$text)
x$text   <- gsub("[[:space:]]+", " ", x$text)
x$text   <- trimws(x$text)
x$nwords <- txt_count(x$text, pattern = " ")
x        <- subset(x, nwords < 1000 & nchar(text) > 0)
```

Build the model

Realistic model We use the subset on the algorithm 'PV-DBOW': Distributed Bag Of Words paragraph vectors

```
model <- paragraph2vec(x = x, type = "PV-DBOW", dim = 100, iter = 20,
                       min_count = 5, lr = 0.05, threads = 4)
str(model)
```

```
## List of 3
##  $ model  :<externalptr>
##  $ data   :List of 4
##   ..$ file        : chr "C:\\Users\\SIMONI~1\\AppData\\Local\\Temp\\RtmpWCZX4q\\textspace_32bc517c13
##   ..$ n           : num 949931
##   ..$ n_vocabulary: num 11408
##   ..$ n_docs      : num 35819
##  $ control:List of 9
##   ..$ min_count: int 5
##   ..$ dim      : int 100
##   ..$ window   : int 10
##   ..$ iter     : int 20
##   ..$ lr       : num 0.05
##   ..$ skipgram : logi TRUE
##   ..$ hs       : int 0
##   ..$ negative : int 5
##   ..$ sample   : num 0.001
##  - attr(*, "class")= chr "paragraph2vec_trained"
```

Similarity prediction

Get similar documents or words when providing sentences, documents or words

```
nn <- predict(model, newdata = c("teenage", "vampire"), type = "nearest",
              which = "word2word", top_n = 5)
nn
```

```
## [[1]]
##     term1  term2 similarity rank
## 1 teenage   girl  0.6914319    1
## 2 teenage    boy  0.6416388    2
## 3 teenage  young  0.6401222    3
## 4 teenage   teen  0.6186927    4
## 5 teenage father  0.6041631    5
```

```
##
## [[2]]
##     term1    term2 similarity rank
## 1 vampire vampires  0.6297932    1
## 2 vampire    clans  0.5613207    2
## 3 vampire werewolf  0.5402659    3
## 4 vampire    human  0.5373096    4
## 5 vampire    blood  0.5284104    5
```

This suggests a movie based on the keyword. Here we use vampire. So a movie will apear that is most similar to the keyword.

```
nn1 <- predict(model, newdata = c("vampire"), type = "nearest", which = "word2doc",  top_n = 5)
nn1
```

```
## [[1]]
##     term1     term2 similarity rank
## 1 vampire tt0499464  0.5662798    1
## 2 vampire tt0068284  0.5656413    2
## 3 vampire tt0074873  0.5527760    3
## 4 vampire tt3898776  0.5527245    4
## 5 vampire tt1545106  0.5453719    5
```

```
data_imdb %>%
  filter(imdb_title_id %in% c("tt1686821", "tt0068284", "tt0074430","tt1608369", "tt3587202"))
```

```
## # A tibble: 5 x 22
##   imdb_title_id title original_title  year date_published genre duration country
##   <chr>         <chr> <chr>          <dbl> <chr>          <chr>    <dbl> <chr>
## 1 tt0068284     Blac~ Blacula         1972 1973-05-03     Fant~       93 USA
## 2 tt0074430     Dr. ~ Dr. Black, Mr~  1976 1976-01-01     Horr~       87 USA
## 3 tt1608369     The ~ The Brides of~  2013 2013-01-29     Fant~      119 USA
## 4 tt1686821     Vamp~ Vampire Acade~  2014 2014-05-01     Acti~      104 USA, UK
## 5 tt3587202     Nosf~ Nosferatu vs.~  2014 2014-03-09     Adve~      134 Canada
## # ... with 14 more variables: language <chr>, director <chr>, writer <chr>,
## #   production_company <chr>, actors <chr>, description <chr>, avg_vote <dbl>,
## #   votes <dbl>, budget <chr>, usa_gross_income <chr>,
## #   worlwide_gross_income <chr>, metascore <dbl>, reviews_from_users <dbl>,
## #   reviews_from_critics <dbl>
```

Here we can use a specific movie to suggest another similar movie. The ID that we have put in is Alice in wonderland.

```
nn2 <- predict(model, newdata = c("tt0004873"), type = "nearest", which = "doc2doc",   top_n = 5)
nn2
```

```
## [[1]]
##       term1     term2 similarity rank
## 1 tt0004873 tt0021599  0.6566673    1
## 2 tt0004873 tt1577811  0.6068298    2
## 3 tt0004873 tt0068190  0.5970679    3
## 4 tt0004873 tt0061191  0.5658745    4
## 5 tt0004873 tt0080116  0.5563391    5
```

```
data_imdb %>%
  filter(imdb_title_id %in% c("tt0021599", "tt0068190", "tt0043719","tt1577811", "tt0111276"))
```

```
## # A tibble: 5 x 22
##   imdb_title_id title original_title  year date_published genre duration country
##   <chr>         <chr> <chr>          <dbl> <chr>          <chr>    <dbl> <chr>
## 1 tt0021599     Alic~ Alice in Wond~  1931 1931-09-30     Fant~       55 USA
## 2 tt0043719     Nuda~ Lady Godiva R~  1951 1951-10-25     Come~       90 UK
## 3 tt0068190     Le a~ Alice's Adven~  1972 1973-04-22     Adve~      101 UK
## 4 tt0111276     Nell~ Stalked         1994 1994-10-01     Crim~       95 Canada~
## 5 tt1577811     Fun ~ Fun in Balloo~  1965 2009           Fami~       53 USA
## # ... with 14 more variables: language <chr>, director <chr>, writer <chr>,
## #   production_company <chr>, actors <chr>, description <chr>, avg_vote <dbl>,
## #   votes <dbl>, budget <chr>, usa_gross_income <chr>,
## #   worlwide_gross_income <chr>, metascore <dbl>, reviews_from_users <dbl>,
## #   reviews_from_critics <dbl>
```

Here we create a "sentence" (combined of keywords) with the intent of having the model search for movies similar to the sentence.

```
sentences <- list(
  sent1 = c("vampire", "werewolf", "teenager"))
nn3 <- predict(model, newdata = sentences, type = "nearest", which = "sent2doc", top_n = 5)
nn3
```

```
## $sent1
##   term1     term2 similarity rank
## 1 sent1 tt1656179  0.6364629    1
## 2 sent1 tt0053271  0.6314456    2
## 3 sent1 tt7200946  0.6206264    3
## 4 sent1 tt0050530  0.6092994    4
## 5 sent1 tt3113696  0.5739096    5
```

```
data_imdb %>%
  filter(imdb_title_id %in% c(nn3$sent1$term2))
```

```
## # A tibble: 5 x 22
##   imdb_title_id title original_title  year date_published genre duration country
##   <chr>         <chr> <chr>          <dbl> <chr>          <chr>    <dbl> <chr>
## 1 tt0050530     I Wa~ I Was a Teena~  1957 1957-06-19     Dram~       76 USA
## 2 tt0053271     Gere~ The Shaggy Dog  1959 1959-12-17     Fami~      104 USA
## 3 tt1656179     I Ki~ I Kissed a Va~  2010 2012-03-30     Musi~       91 USA
## 4 tt3113696     Don'~ Don't Kill It   2016 2017-03-03     Acti~       83 USA
## 5 tt7200946     Oh, ~ Oh, Ramona!     2019 2019-06-01     Come~      109 Romania
## # ... with 14 more variables: language <chr>, director <chr>, writer <chr>,
## #   production_company <chr>, actors <chr>, description <chr>, avg_vote <dbl>,
## #   votes <dbl>, budget <chr>, usa_gross_income <chr>,
## #   worlwide_gross_income <chr>, metascore <dbl>, reviews_from_users <dbl>,
## #   reviews_from_critics <dbl>
```

Network on embedded data

We create the embedings on the top 50 films based ont the average vote score. only taking the top 50 movies.

```
data_network_emb = data_imdb %>%
  filter(genre %in% c("Drama", "Comedy", "Horror", "Thriller"), year >= 2000) %>%
  filter(language == "English") %>%
  drop_na() %>%
  rename(text = description) %>%
  rename(id= imdb_title_id) %>%
  arrange(desc(avg_vote)) %>%
  select(avg_vote, everything()) %>%
  slice(1:50) %>%
  select(id, text)
```

We create embedings using this recipe

```
Embeding_recipe <- data_network_emb %>%
  recipe(~.) %>%
  update_role(id, new_role = "id") %>%
  step_tokenize(text, token = 'words') %>%
  step_stem(text) %>%
  step_stopwords(text, keep = FALSE) %>%
  step_word_embeddings(text, embeddings = embedding_glove6b())%>%
  prep() %>%
  juice()
```

We join with the embedings data to get the embedings for the top 50 films

```
network_dataset= Embeding_recipe %>%
  left_join(data_emb_title, by= c("id"= "imdb_title_id")) %>%
  select(title, everything(), -id)
```

To keep the names on the films we create a matrix we can join in later. as the names fall away.

```
network_dataset_movies= network_dataset[,1]
```

We use pca for dimensionality reduction

```
network_data_pca <- network_dataset[,c(2:51)] %>%
  drop_na() %>%
  prcomp(center = TRUE , scale = TRUE)
```

- Next, we could create a distance matrix (using the `dist()`) function.

```
network_data_dist <- network_data_pca$x %>% dist(method = "euclidean")
```

La voila. Such a distance matrix representas a relational structure and can be modelled as a network.

```
g <- network_data_dist %>%
  as.matrix() %>%
  as_tbl_graph(directed = FALSE)
```

we create our table graph

```
g <- g %>% simplify() %>% as_tbl_graph()
```

We add in the title

```
g = g %N>%
  mutate(title = network_dataset_movies$title)
```

show the graph

```
g
```

```
## # A tbl_graph: 50 nodes and 1225 edges
## #
## # An undirected simple graph with 1 component
## #
## # Node Data: 50 x 2 (active)
##   name  title
##   <chr> <chr>
## 1 1     Requiem for a Dream
## 2 2     The Help
## 3 3     Manchester by the Sea
## 4 4     Mi chiamo Sam
## 5 5     Una notte da leoni
## 6 6     La 25ª ora
## # ... with 44 more rows
## #
## # Edge Data: 1,225 x 3
##    from    to weight
##   <int> <int>  <dbl>
## 1     1     2  12.6
## 2     1     3   9.11
## 3     1     4  11.5
## # ... with 1,222 more rows
```

We add different measures for centrality, and look at the most central movies

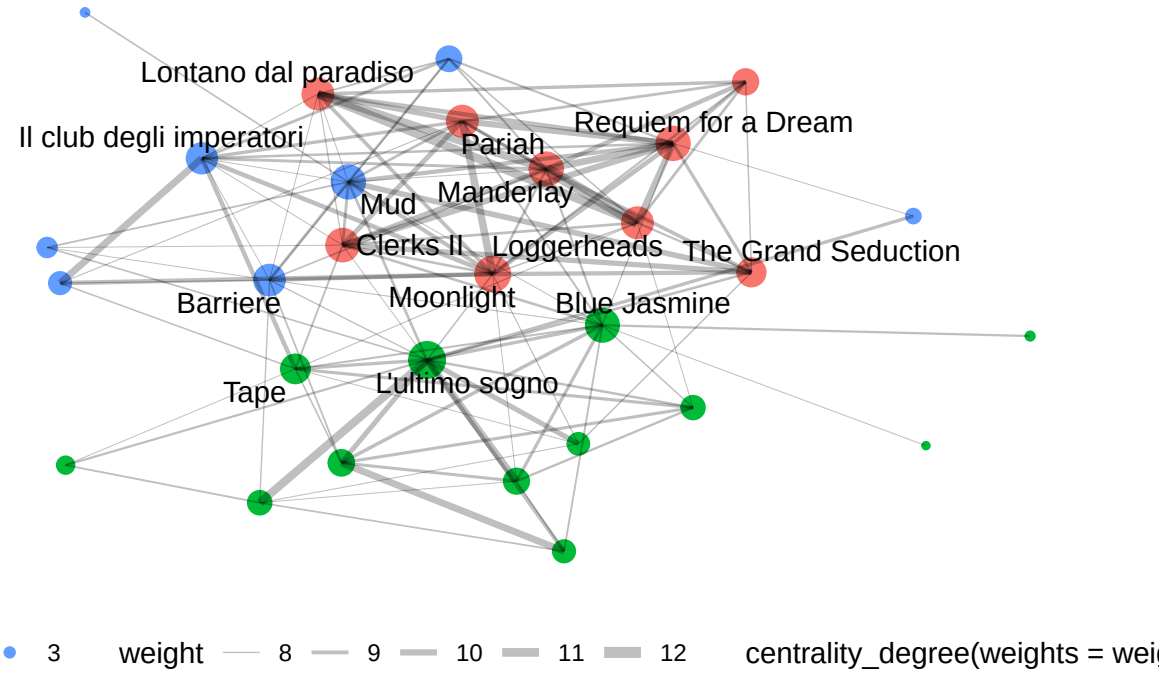
```
g <- g %N>%
  mutate(cent_dgr = centrality_degree(weights = weight),
         cent_eigen = centrality_eigen(weights = weight),
         cent_between = centrality_betweenness(weights = weight))

most_central_movies= g %N>%
  as_tibble() %>%
  arrange(desc(cent_dgr))
```

We revert the weight so it looks how similar the movies are. and only pick the top 50% We also filter away isolated nodes.

```
g <- g %E>%
  mutate(weight = max(weight) - weight) %>%
  filter(weight >= weight %>% quantile(0.50)) %N>%
  filter(!node_is_isolated()) %>%
  mutate(community = group_louvain(weights = weight) %>% factor())
```

Lets take a look!

```
set.seed(1337)
g %N>%filter(cent_dgr > 450)%>%
  ggraph(layout = "kk") +
  geom_node_point(aes(col = community, size = centrality_degree(weights = weight))) +
  geom_edge_link(aes(width = weight), alpha = 0.25) +
  scale_edge_width(range = c(0.1, 2)) +
  geom_node_text(aes(label = title, filter = percent_rank(centrality_degree(weights = weight)) > 0.5), r
  theme_graph(base_family="sans") +
  theme(legend.position = 'bottom')
```



The size of the node describing the centrality degree, and edges shows how similar the movies are. The nodes are colored by community.

Hiracial network

We can also create the network using "hclust"

```
network_data_hc <- network_data_dist %>%
  hclust(method = "ward.D2")
```

Again, this structure can be directly transfered to a graph object.

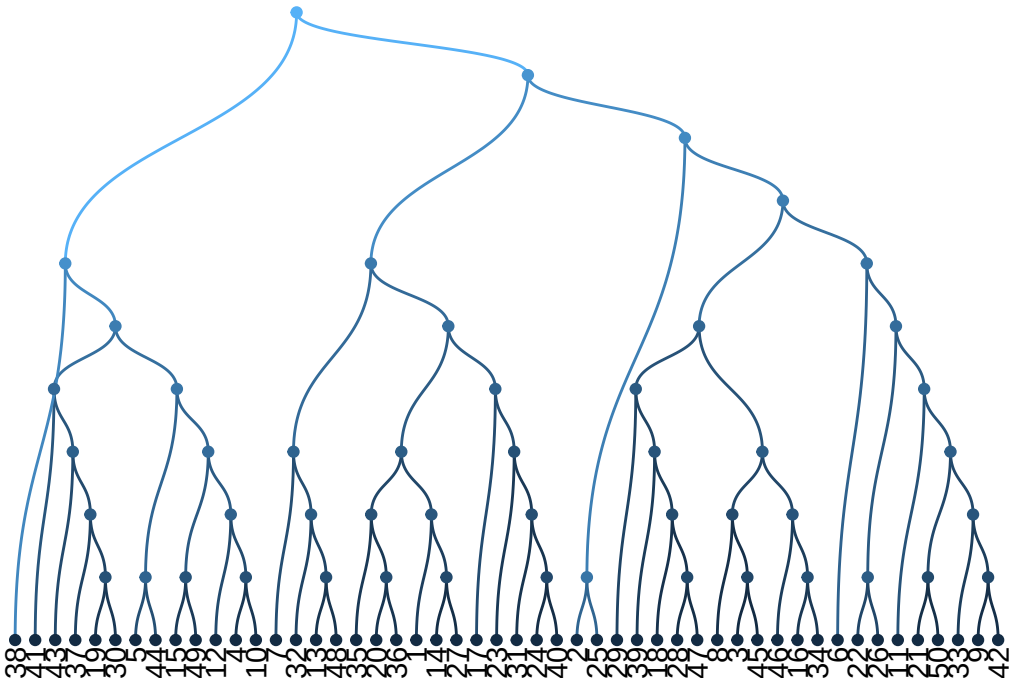
```
g_hc <- network_data_hc %>% as_tbl_graph()
```

```
g_hc
```

```
## # A tbl_graph: 99 nodes and 98 edges
## #
## # A rooted tree
## #
## # Node Data: 99 x 4 (active)
##   height leaf  label members
##    <dbl> <lgl> <chr>   <int>
## 1      0 TRUE  38          1
## 2      0 TRUE  41          1
## 3      0 TRUE  43          1
## 4      0 TRUE  37          1
## 5      0 TRUE  19          1
## 6      0 TRUE  30          1
## # ... with 93 more rows
## #
## # Edge Data: 98 x 2
##    from    to
##   <int> <int>
## 1     7     5
## 2     7     6
## 3     8     4
## # ... with 95 more rows
```

We can plot the network as a dendrogram.

```
g_hc %>% ggraph(layout = 'dendrogram') +
  geom_edge_diagonal(aes(col = .N()$height[from])) +
  geom_node_point(aes(col =height)) +
  geom_node_text(aes(filter = leaf, label = label), angle=90, hjust=1, nudge_y=-0.1) +
  theme_graph() +
  theme(legend.position = 'none')
```




```
  ylim(-0.6, NA)
```

```
## <ScaleContinuousPosition>
##  Range:
##  Limits:    0 --    1
```

We can see which number is which movie in the table under

```
network_dataset_movies %>%
  mutate(name = 1:50)
```

```
## # A tibble: 50 x 2
##    title                              name
##    <chr>                             <int>
##  1 Requiem for a Dream                   1
##  2 The Help                              2
##  3 Manchester by the Sea                 3
##  4 Mi chiamo Sam                         4
##  5 Una notte da leoni                    5
##  6 La 25ª ora                            6
##  7 Suxbad: Tre menti sopra il pelo       7
##  8 Gifted - Il dono del talento          8
##  9 Conta su di me                        9
## 10 L'ultimo sogno                       10
## # ... with 40 more rows
```