Homework 4: Machine Learning

Eugene Nesterenko, github YevheniiN281

2023-06-07

Table of Contents

# The Bechdel Test

<https://fivethirtyeight.com/features/the-dollar-and-cents-case-against-hollywoods-exclusion-of-women/>

The [Bechdel test](https://bechdeltest.com) is a way to assess how women are depicted in Hollywood movies. In order for a movie to pass the test:

1. It has to have at least two [named] women in it
2. Who talk to each other
3. About something besides a man

There is a nice article and analysis you can find here <https://fivethirtyeight.com/features/the-dollar-and-cents-case-against-hollywoods-exclusion-of-women/> We have a sample of 1394 movies and we want to fit a model to predict whether a film passes the test or not.

bechdel <- read\_csv(here::here("data", "bechdel.csv")) %>%   
 mutate(test = factor(test))

## Rows: 1394 Columns: 10  
## ── Column specification ────────────────────────────────────────────────────────  
## Delimiter: ","  
## chr (4): title, test, rated, genre  
## dbl (6): year, budget\_2013, domgross\_2013, intgross\_2013, metascore, imdb\_ra...  
##   
## ℹ Use `spec()` to retrieve the full column specification for this data.  
## ℹ Specify the column types or set `show\_col\_types = FALSE` to quiet this message.

glimpse(bechdel)

## Rows: 1,394  
## Columns: 10  
## $ year <dbl> 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, 20…  
## $ title <chr> "12 Years a Slave", "2 Guns", "42", "47 Ronin", "A Good …  
## $ test <fct> Fail, Fail, Fail, Fail, Fail, Pass, Pass, Fail, Pass, Pa…  
## $ budget\_2013 <dbl> 2.00, 6.10, 4.00, 22.50, 9.20, 1.20, 1.30, 13.00, 4.00, …  
## $ domgross\_2013 <dbl> 5.3107035, 7.5612460, 9.5020213, 3.8362475, 6.7349198, 1…  
## $ intgross\_2013 <dbl> 15.8607035, 13.2493015, 9.5020213, 14.5803842, 30.424919…  
## $ rated <chr> "R", "R", "PG-13", "PG-13", "R", "R", "PG-13", "PG-13", …  
## $ metascore <dbl> 97, 55, 62, 29, 28, 55, 48, 33, 90, 58, 52, 78, 83, 53, …  
## $ imdb\_rating <dbl> 8.3, 6.8, 7.6, 6.6, 5.4, 7.8, 5.7, 5.0, 7.5, 7.4, 6.2, 7…  
## $ genre <chr> "Biography", "Action", "Biography", "Action", "Action", …

How many films fail/pass the test, both as a number and as a %?

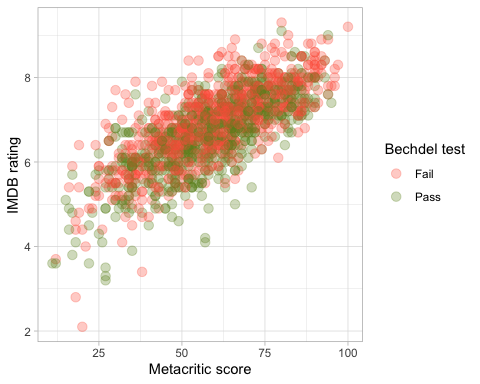
# Grouping the data by pass/fail criterion, counting occurances and percantage of each  
bechdel %>%   
 group\_by(test) %>%   
 summarize(count = n()) %>%   
 mutate(percentage = round(count/sum(count)\*100,1)) %>%   
 print()

## # A tibble: 2 × 3  
## test count percentage  
## <fct> <int> <dbl>  
## 1 Fail 772 55.4  
## 2 Pass 622 44.6

772 movies (55.4%) **fail** the Bechdel test, while only 622 (44.6%) **pass** it. The naive prediction would be that every movie fails the Bechdel test, and we expect it to be correct in 55.4% of cases.

## Movie scores

ggplot(data = bechdel, aes(  
 x = metascore,  
 y = imdb\_rating,  
 colour = test  
)) +  
 geom\_point(alpha = .3, size = 3) +  
 scale\_colour\_manual(values = c("tomato", "olivedrab")) +  
 labs(  
 x = "Metacritic score",  
 y = "IMDB rating",  
 colour = "Bechdel test"  
 ) +  
 theme\_light()



There seem to be a positive correlation of IMDB rating and Metacritic score (which is natural if both ratings assess a common factor - the “true” quality of the movie). However, it’s hard to tell from the graph whether there is correlation or causal relationship between rating and ability to pass Bechdel test.

# Split the data

# \*\*Split the data\*\*  
  
set.seed(123)  
  
data\_split <- initial\_split(bechdel, # updated data  
 prop = 0.8,   
 strata = test)  
  
bechdel\_train <- training(data\_split)   
bechdel\_test <- testing(data\_split)

Check the counts and % (proportions) of the test variable in each set.

# Grouping the data by pass/fail criterion, counting occurances and percantage of each  
# Apply separately to training sample  
bechdel\_train %>%   
 group\_by(test) %>%   
 summarize(count = n()) %>%   
 mutate(percentage = round(count/sum(count)\*100,1)) %>%   
 print()

## # A tibble: 2 × 3  
## test count percentage  
## <fct> <int> <dbl>  
## 1 Fail 617 55.4  
## 2 Pass 497 44.6

# Grouping the data by pass/fail criterion, counting occurances and percantage of each  
# Apply separately to testing sample  
bechdel\_test %>%   
 group\_by(test) %>%   
 summarize(count = n()) %>%   
 mutate(percentage = round(count/sum(count)\*100,1)) %>%   
 print()

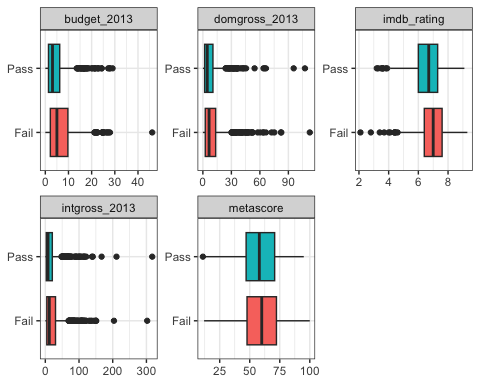
## # A tibble: 2 × 3  
## test count percentage  
## <fct> <int> <dbl>  
## 1 Fail 155 55.4  
## 2 Pass 125 44.6

Both in the **training sample** and in the **testing sample** proportion of films who pass (fail) remains the same - 55.4% (44.6%), which indicates a good split, **both samples are representative**. Counts are proportionately lower, roughly in line with 80/20 split (to be precise, 79.9/20.1). Total count in both divided samples and undivided population is 1394, which means **no observation were omitted**.

## Feature exploration

## Any outliers?

bechdel %>%   
 select(test, budget\_2013, domgross\_2013, intgross\_2013, imdb\_rating, metascore) %>%   
  
 pivot\_longer(cols = 2:6,  
 names\_to = "feature",  
 values\_to = "value") %>%   
 ggplot()+  
 aes(x=test, y = value, fill = test)+  
 coord\_flip()+  
 geom\_boxplot()+  
 facet\_wrap(~feature, scales = "free")+  
 theme\_bw()+  
 theme(legend.position = "none")+  
 labs(x=NULL,y = NULL)



Practically for any variable there is a number of outliers, most prominent for financial metrics:

* Budget: outliers start from $15-20m
* Domestic box office: outliers collect from $25-30m
* International box office: outliers collect from $50-60m

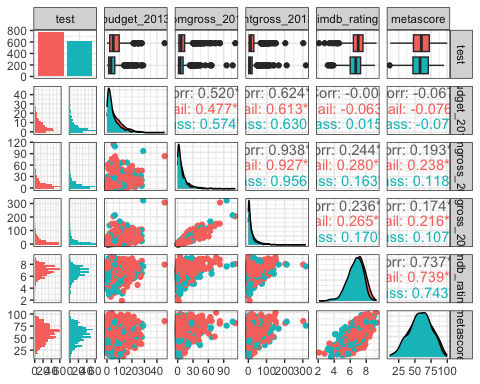
We can expect that on average, high-budget film have box office both in the U.S. and internationally, i.e. outliers in one variable coincide with outliers in others.

For ratings, there are a few outliers for the IMDB rating (below 4-5) and practically no distinct outliers for Metacritic score. However, the latter variable also has wider interquartile range (25% - about 50 score; 75% - about 75 score).

## Scatterplot - Correlation Matrix

Write a paragraph discussing the output of the following

bechdel %>%   
 select(test, budget\_2013, domgross\_2013, intgross\_2013, imdb\_rating, metascore)%>%   
 ggpairs(aes(colour=test), alpha=0.2)+  
 theme\_bw()



We would ignore the relation of independent variables to the “test” variable, as it was briefly discussed in the previous paragraph and would require more detailed analysis when applying the model.

The independent variables distribution itself is a point of interest:

* **Money variables (budget, domestic and international box gross revenues)** are highly right skewed, with the majority observations relatively close to 0, but few are blockbuster projects with high budget and high box office. The distribution is definitely not normal, and looks much like lognormal or Poisson. There is no prominent difference between movies that pass of fail the Bechdel test.
* I**MDB rating** distribution is closer to normal, but it still exhibits a little skewness (left). The outliers are movies of a bad quality (apparently, it is easy to produce a significantly worse than average movie but extremely hard to produce a significantly better one). There is no significant difference between movies that pass of fail the Bechdel test.
* **Metacritic score** is likely normally distributed, but with a large coefficient of variation (score could go either extremely high or extremely low, there is less centered expectation of movies). There is no significant difference between movies that pass of fail the Bechdel test.

Correlation analysis among independent variables provides additional insights:

* **Budget** is positively (but not perfectly) correlated with b**oth domestic and international box office**. Apparently, to produce a commercially successful product, a movie studio needs substantial investment: hire good director, scenarist, actors, allocate enough time for production. We expect a causal relationship here, as budget is spent prior to revenues.
* Obviously, **domestic and international revenues** are almost perfectly correlated, but they depend on a third factor(s) - movie quality (and probably scale, proxied by budget). If the movie is good, it would profit in both markets.
* Both ratings (**IMDB and Metacritic)** have slightly positive correlation with **revenues (domestic and international)**. We suggest that initially both variables in a pair would depend on a third factor - movie quality; but later ratings would shape viewers preferences, praising box office for good movies and hindering for bad ones.
* **Ratings themselves are highly correlated**, but we expect the relationship to be drawn from the third factor - movie quality.
* **Budget correlation with ratings is small and insignificant** - you can’t just pour money and expect to people to love the movie.

High correlation for pairs of variables (domestic revenues ; international revenues) and (IMDB rating ; Metacritic score) indicate a **multicollinearity**, which would impact the regression results, both estimates and standard errors.

At this point, we would recommend either **omitting duplicating variables** (international revenues, Metacritic score) or trying to divide factor influence by **applying PCA** (principal component analysis). However, for the sake of this assignment, we would use the whole set of variables to train models and compare them.

## Categorical variables

Write a paragraph discussing the output of the following

bechdel %>%   
 group\_by(genre, test) %>%  
 summarise(n = n()) %>%   
 mutate(prop = n/sum(n))

## `summarise()` has grouped output by 'genre'. You can override using the  
## `.groups` argument.

## # A tibble: 24 × 4  
## # Groups: genre [14]  
## genre test n prop  
## <chr> <fct> <int> <dbl>  
## 1 Action Fail 260 0.707  
## 2 Action Pass 108 0.293  
## 3 Adventure Fail 52 0.559  
## 4 Adventure Pass 41 0.441  
## 5 Animation Fail 63 0.677  
## 6 Animation Pass 30 0.323  
## 7 Biography Fail 36 0.554  
## 8 Biography Pass 29 0.446  
## 9 Comedy Fail 138 0.427  
## 10 Comedy Pass 185 0.573  
## # ℹ 14 more rows

bechdel %>%   
 group\_by(rated, test) %>%  
 summarise(n = n()) %>%   
 mutate(prop = n/sum(n))

## `summarise()` has grouped output by 'rated'. You can override using the  
## `.groups` argument.

## # A tibble: 10 × 4  
## # Groups: rated [5]  
## rated test n prop  
## <chr> <fct> <int> <dbl>  
## 1 G Fail 16 0.615  
## 2 G Pass 10 0.385  
## 3 NC-17 Fail 5 0.833  
## 4 NC-17 Pass 1 0.167  
## 5 PG Fail 115 0.561  
## 6 PG Pass 90 0.439  
## 7 PG-13 Fail 283 0.529  
## 8 PG-13 Pass 252 0.471  
## 9 R Fail 353 0.568  
## 10 R Pass 269 0.432

The probability to pass a score seems to vary significantly with genre. For instance, among comedies, as much as 57% of movies pass the Bechdel test, while among action movies, the rate is only 29%. (one explanation could be that action movies are typically high-violence plots with masculine characters).

The influence of production rating (which is, to our understanding, a category restricting which ages audience can watch the movie) is not significant. There are categories (G and NC-17) with unusually low scores. But for the categories with large enough number of observations, difference in average probability is negligible (from 43% to 47%)

# Train first models. test ~ metascore + imdb\_rating

lr\_mod <- logistic\_reg() %>%   
 set\_engine(engine = "glm") %>%   
 set\_mode("classification")  
  
lr\_mod

## Logistic Regression Model Specification (classification)  
##   
## Computational engine: glm

tree\_mod <- decision\_tree() %>%   
 set\_engine(engine = "C5.0") %>%   
 set\_mode("classification")  
  
tree\_mod

## Decision Tree Model Specification (classification)  
##   
## Computational engine: C5.0

lr\_fit <- lr\_mod %>% # parsnip model  
 fit(test ~ metascore + imdb\_rating, # a formula  
 data = bechdel\_train # dataframe  
 )  
  
tree\_fit <- tree\_mod %>% # parsnip model  
 fit(test ~ metascore + imdb\_rating, # a formula  
 data = bechdel\_train # dataframe  
 )

## Logistic regression

lr\_fit %>%  
 broom::tidy()

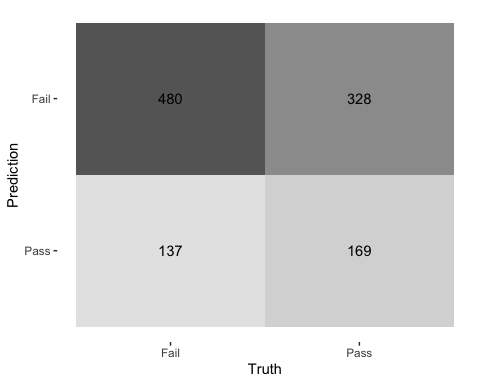
## # A tibble: 3 × 5  
## term estimate std.error statistic p.value  
## <chr> <dbl> <dbl> <dbl> <dbl>  
## 1 (Intercept) 2.80 0.494 5.68 1.35e- 8  
## 2 metascore 0.0207 0.00536 3.86 1.13e- 4  
## 3 imdb\_rating -0.625 0.100 -6.24 4.36e-10

lr\_preds <- lr\_fit %>%  
 augment(new\_data = bechdel\_train) %>%  
 mutate(.pred\_match = if\_else(test == .pred\_class, 1, 0))

Logistic regression seems to indicate positive influence of Metacritic score to pass the Bechdel test, and negative influence of IMDB rating, with both variables statistically significant at 95% confidence. However, potential multicollinearity between independent variables raises a question whether estimates are reliable. In particular, it seems counterintuitive that one rating favors the probability of passing the Bechdel test while another one hinders it.

### Confusion matrix

lr\_preds %>%   
 conf\_mat(truth = test, estimate = .pred\_class) %>%   
 autoplot(type = "heatmap")

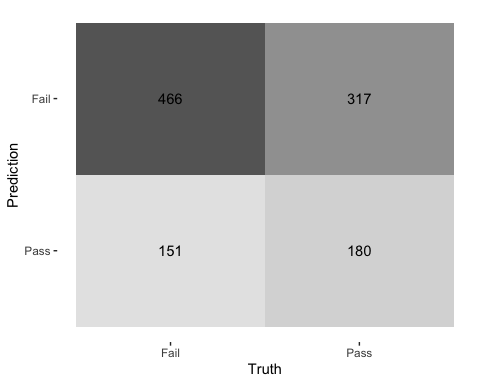


In our opinion, model quality is not very good, both in terms of prediction capability (12.3% false positives as opposed to 15.2% true positives) and test power (29.4% of false negatives compared to 43.1% of true negatives). However, it is still marginally better than the naive model predicting Fail all the time (58.3% of total correct predictions vs 55.4%)

## Decision Tree

tree\_preds <- tree\_fit %>%  
 augment(new\_data = bechdel\_train) %>%  
 mutate(.pred\_match = if\_else(test == .pred\_class, 1, 0))

tree\_preds %>%   
 conf\_mat(truth = test, estimate = .pred\_class) %>%   
 autoplot(type = "heatmap")

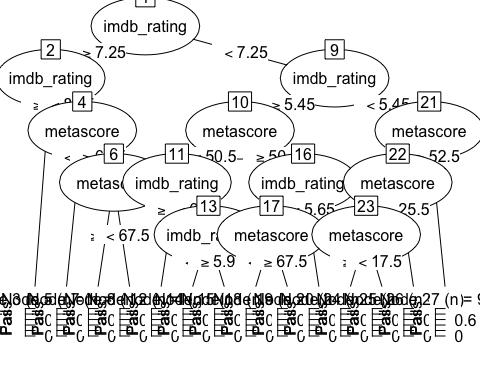


We have amended the code to use the same training sample as in case with logistic regression.

The result is more or less the same, with larger percentage of false positives (14% vs 12% in logistic regression) but lower percentage of false negatives (28% vs 29%). Overall, the choice of model would depend on what is more important to the researcher - predict the pass correctly or not making a mistake of failing to predict a pass.

## Draw the decision tree

draw\_tree <-   
 rpart::rpart(  
 test ~ metascore + imdb\_rating,  
 data = bechdel\_train, # uses data that contains both birth weight and `low`  
 control = rpart::rpart.control(maxdepth = 5, cp = 0, minsplit = 10)  
 ) %>%   
 partykit::as.party()  
plot(draw\_tree)



The decision tree seems to be too detailed, with frequently less than 10 observations in each bucket. We suspect the problem of overfitting.

# Cross Validation

Run the code below. What does it return?

set.seed(123)  
bechdel\_folds <- vfold\_cv(data = bechdel\_train,   
 v = 10,   
 strata = test)  
bechdel\_folds

## # 10-fold cross-validation using stratification   
## # A tibble: 10 × 2  
## splits id   
## <list> <chr>   
## 1 <split [1002/112]> Fold01  
## 2 <split [1002/112]> Fold02  
## 3 <split [1002/112]> Fold03  
## 4 <split [1002/112]> Fold04  
## 5 <split [1002/112]> Fold05  
## 6 <split [1002/112]> Fold06  
## 7 <split [1002/112]> Fold07  
## 8 <split [1004/110]> Fold08  
## 9 <split [1004/110]> Fold09  
## 10 <split [1004/110]> Fold10

The code returns a list of 10 folds (splits) that we are going to use it later one by one in the cross-validation stage.

## fit\_resamples()

Trains and tests a resampled model.

lr\_fit <- lr\_mod %>%  
 fit\_resamples(  
 test ~ metascore + imdb\_rating,  
   
 # Instead of data, the argument is resamples - perhaps, to use in a loop?  
 resamples = bechdel\_folds  
 )  
  
  
tree\_fit <- tree\_mod %>%  
 fit\_resamples(  
 test ~ metascore + imdb\_rating,  
 resamples = bechdel\_folds  
 )

## collect\_metrics()

Unnest the metrics column from a tidymodels fit\_resamples()

collect\_metrics(lr\_fit)

## # A tibble: 2 × 6  
## .metric .estimator mean n std\_err .config   
## <chr> <chr> <dbl> <int> <dbl> <chr>   
## 1 accuracy binary 0.575 10 0.0149 Preprocessor1\_Model1  
## 2 roc\_auc binary 0.606 10 0.0189 Preprocessor1\_Model1

collect\_metrics(tree\_fit)

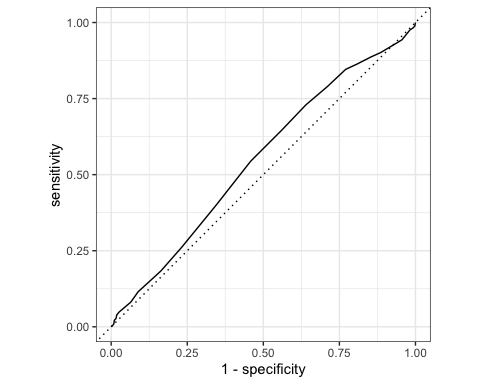
## # A tibble: 2 × 6  
## .metric .estimator mean n std\_err .config   
## <chr> <chr> <dbl> <int> <dbl> <chr>   
## 1 accuracy binary 0.571 10 0.0156 Preprocessor1\_Model1  
## 2 roc\_auc binary 0.547 10 0.0201 Preprocessor1\_Model1

Based on the area under the ROC curve criterion, the logistic regression is a better choice on average (60.6% correct predictions), whereas decision tree model actually shows **worse** results than the naive model (54.7% vs 55.4%).

tree\_preds <- tree\_mod %>%   
 fit\_resamples(  
 test ~ metascore + imdb\_rating,   
 resamples = bechdel\_folds,  
 control = control\_resamples(save\_pred = TRUE) #<<  
 )  
  
# What does the data for ROC look like?  
tree\_preds %>%   
 collect\_predictions() %>%   
 roc\_curve(truth = test, .pred\_Fail)

## # A tibble: 29 × 3  
## .threshold specificity sensitivity  
## <dbl> <dbl> <dbl>  
## 1 -Inf 0 1   
## 2 0.262 0 1   
## 3 0.317 0.00201 0.989  
## 4 0.373 0.00805 0.982  
## 5 0.440 0.0181 0.976  
## 6 0.459 0.0443 0.943  
## 7 0.460 0.0765 0.924  
## 8 0.464 0.115 0.901  
## 9 0.465 0.147 0.887  
## 10 0.465 0.191 0.864  
## # ℹ 19 more rows

# Draw the ROC  
tree\_preds %>%   
 collect\_predictions() %>%   
 roc\_curve(truth = test, .pred\_Fail) %>%   
 autoplot()



Dependent only on ratings (IMDB and Metacritic), the model seems to perform better than the naive prediction, with ROC curve surpassing 45 degrees angle line for most of the times. However, we would need to look additionally at the area under the curve.

# Build a better training set with recipes

## Preprocessing options

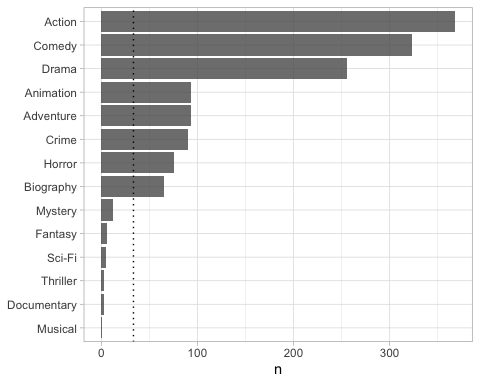
* Encode categorical predictors
* Center and scale variables
* Handle class imbalance
* Impute missing data
* Perform dimensionality reduction
* … …

## To build a recipe

1. Start the recipe()
2. Define the variables involved
3. Describe **prep**rocessing [step-by-step]

## Collapse Some Categorical Levels

Do we have any genre with few observations? Assign genres that have less than 3% to a new category ‘Other’



movie\_rec <-  
 recipe(test ~ .,  
 data = bechdel\_train) %>%  
   
 # Genres with less than 5% will be in a catewgory 'Other'  
 step\_other(genre, threshold = .03)

## Before recipe

## # A tibble: 14 × 2  
## genre n  
## <chr> <int>  
## 1 Action 293  
## 2 Comedy 254  
## 3 Drama 213  
## 4 Adventure 75  
## 5 Animation 72  
## 6 Crime 68  
## 7 Horror 68  
## 8 Biography 50  
## 9 Mystery 7  
## 10 Fantasy 5  
## 11 Sci-Fi 3  
## 12 Thriller 3  
## 13 Documentary 2  
## 14 Musical 1

## After recipe

movie\_rec %>%   
 prep() %>%   
 bake(new\_data = bechdel\_train) %>%   
 count(genre, sort = TRUE)

## # A tibble: 9 × 2  
## genre n  
## <fct> <int>  
## 1 Action 293  
## 2 Comedy 254  
## 3 Drama 213  
## 4 Adventure 75  
## 5 Animation 72  
## 6 Crime 68  
## 7 Horror 68  
## 8 Biography 50  
## 9 other 21

## step\_dummy()

Converts nominal data into numeric dummy variables

movie\_rec <- recipe(test ~ ., data = bechdel) %>%  
 step\_other(genre, threshold = .03) %>%   
 step\_dummy(all\_nominal\_predictors())   
  
movie\_rec

##

## ── Recipe ──────────────────────────────────────────────────────────────────────

##

## ── Inputs

## Number of variables by role

## outcome: 1  
## predictor: 9

##

## ── Operations

## • Collapsing factor levels for: genre

## • Dummy variables from: all\_nominal\_predictors()

## Let’s think about the modelling

What if there were no films with rated NC-17 in the training data?

* Will the model have a coefficient for rated NC-17?
* What will happen if the test data includes a film with rated NC-17?

The model will have no coefficient, because the dummy variable for the NC-17 would not be created in the first place. Subsequently, if test data includes such observations, two things could happen:

* The model breaks down as it does not know how to interpret new value of the variable (or it splits new variables in a list of dummies with 1 extra compared to the testing sample)
* The model just ignores the unknown value, applying 0 to all dummies and in fact using the implied coefficient for the base value (say, rating “G”). However, actual category (“NC-17”) is different, which would distort prediction capabilities.

## step\_novel()

Adds a catch-all level to a factor for any new values not encountered in model training, which lets R intelligently predict new levels in the test set.

movie\_rec <- recipe(test ~ ., data = bechdel) %>%  
 step\_other(genre, threshold = .03) %>%   
 step\_novel(all\_nominal\_predictors) %>% # Use \*before\* `step\_dummy()` so new level is dummified  
 step\_dummy(all\_nominal\_predictors())

## step\_zv()

Intelligently handles zero variance variables (variables that contain only a single value)

movie\_rec <- recipe(test ~ ., data = bechdel) %>%  
 step\_other(genre, threshold = .03) %>%   
 step\_novel(all\_nominal(), -all\_outcomes()) %>% # Use \*before\* `step\_dummy()` so new level is dummified  
 step\_dummy(all\_nominal(), -all\_outcomes()) %>%   
 step\_zv(all\_numeric(), -all\_outcomes())

## step\_normalize()

Centers then scales numeric variable (mean = 0, sd = 1)

movie\_rec <- recipe(test ~ ., data = bechdel) %>%  
 step\_other(genre, threshold = .03) %>%   
 step\_novel(all\_nominal(), -all\_outcomes()) %>% # Use \*before\* `step\_dummy()` so new level is dummified  
 step\_dummy(all\_nominal(), -all\_outcomes()) %>%   
 step\_zv(all\_numeric(), -all\_outcomes()) %>%   
 step\_normalize(all\_numeric())

## step\_corr()

Removes highly correlated variables

movie\_rec <- recipe(test ~ ., data = bechdel) %>%  
 step\_other(genre, threshold = .03) %>%   
 step\_novel(all\_nominal(), -all\_outcomes()) %>% # Use \*before\* `step\_dummy()` so new level is dummified  
 step\_dummy(all\_nominal(), -all\_outcomes()) %>%   
 step\_zv(all\_numeric(), -all\_outcomes()) %>%   
 step\_normalize(all\_numeric()) # Remove the last line to save processing time %>%   
 # step\_corr(all\_predictors(), threshold = 0.75, method = "spearman")   
  
  
  
movie\_rec

##

## ── Recipe ──────────────────────────────────────────────────────────────────────

##

## ── Inputs

## Number of variables by role

## outcome: 1  
## predictor: 9

##

## ── Operations

## • Collapsing factor levels for: genre

## • Novel factor level assignment for: all\_nominal(), -all\_outcomes()

## • Dummy variables from: all\_nominal(), -all\_outcomes()

## • Zero variance filter on: all\_numeric(), -all\_outcomes()

## • Centering and scaling for: all\_numeric()

# Define different models to fit

## Model Building  
  
# 1. Pick a `model type`  
# 2. set the `engine`  
# 3. Set the `mode`: regression or classification  
  
# Logistic regression  
log\_spec <- logistic\_reg() %>% # model type  
 set\_engine(engine = "glm") %>% # model engine  
 set\_mode("classification") # model mode  
  
# Show your model specification  
log\_spec

## Logistic Regression Model Specification (classification)  
##   
## Computational engine: glm

# Decision Tree  
tree\_spec <- decision\_tree() %>%  
 set\_engine(engine = "C5.0") %>%  
 set\_mode("classification")  
  
tree\_spec

## Decision Tree Model Specification (classification)  
##   
## Computational engine: C5.0

# Random Forest  
library(ranger)  
  
rf\_spec <-   
 rand\_forest() %>%   
 set\_engine("ranger", importance = "impurity") %>%   
 set\_mode("classification")  
  
  
# Boosted tree (XGBoost)  
library(xgboost)

##   
## Attaching package: 'xgboost'

## The following object is masked from 'package:dplyr':  
##   
## slice

xgb\_spec <-   
 boost\_tree() %>%   
 set\_engine("xgboost") %>%   
 set\_mode("classification")   
  
# K-nearest neighbour (k-NN)  
knn\_spec <-   
 nearest\_neighbor(neighbors = 4) %>% # we can adjust the number of neighbors   
 set\_engine("kknn") %>%   
 set\_mode("classification")

# Bundle recipe and model with workflows

log\_wflow <- # new workflow object  
 workflow() %>% # use workflow function  
 add\_recipe(movie\_rec) %>% # use the new recipe  
 add\_model(log\_spec) # add your model spec  
  
# show object  
log\_wflow

## ══ Workflow ════════════════════════════════════════════════════════════════════  
## Preprocessor: Recipe  
## Model: logistic\_reg()  
##   
## ── Preprocessor ────────────────────────────────────────────────────────────────  
## 5 Recipe Steps  
##   
## • step\_other()  
## • step\_novel()  
## • step\_dummy()  
## • step\_zv()  
## • step\_normalize()  
##   
## ── Model ───────────────────────────────────────────────────────────────────────  
## Logistic Regression Model Specification (classification)  
##   
## Computational engine: glm

## A few more workflows  
  
tree\_wflow <-  
 workflow() %>%  
 add\_recipe(movie\_rec) %>%   
 add\_model(tree\_spec)   
  
rf\_wflow <-  
 workflow() %>%  
 add\_recipe(movie\_rec) %>%   
 add\_model(rf\_spec)   
  
xgb\_wflow <-  
 workflow() %>%  
 add\_recipe(movie\_rec) %>%   
 add\_model(xgb\_spec)  
  
knn\_wflow <-  
 workflow() %>%  
 add\_recipe(movie\_rec) %>%   
 add\_model(knn\_spec)

HEADS UP

1. How many models have you specified? **5**
2. What’s the difference between a model specification and a workflow? To our understanding, model specification relates to the choice of model and setting relevant parameters, such as engine and mode (classification or regression). A workflow is a more detailed process of working with the data (including using the recipe to manipulate data, and applying the model of choice)
3. Do you need to add a formula (e.g., test ~ .) if you have a recipe? No, because in the recipe we have chosen our variables (and manipulated data) beforehand.

# Model Comparison

You now have all your models. Adapt the code from slides code-from-slides-CA-housing.R, line 400 onwards to assess which model gives you the best classification.

**LOGISTIC REGRESSION**

log\_res <- log\_wflow %>%   
 fit\_resamples(  
 resamples = bechdel\_folds,   
 metrics = metric\_set(  
 recall, precision, f\_meas, accuracy,  
 kap, roc\_auc, sens, spec),  
 control = control\_resamples(save\_pred = TRUE))

## → A | warning: glm.fit: algorithm did not converge

## There were issues with some computations A: x1

## → B | warning: prediction from a rank-deficient fit may be misleading

## There were issues with some computations A: x1There were issues with some computations A: x2 B: x1  
## There were issues with some computations A: x3 B: x2  
## There were issues with some computations A: x4 B: x3  
## There were issues with some computations A: x5 B: x4  
## There were issues with some computations A: x6 B: x5  
## There were issues with some computations A: x7 B: x6  
## There were issues with some computations A: x8 B: x7  
## There were issues with some computations A: x9 B: x8  
## There were issues with some computations A: x10 B: x9  
## There were issues with some computations A: x10 B: x10

log\_res %>% collect\_metrics(summarize = TRUE)

## # A tibble: 8 × 6  
## .metric .estimator mean n std\_err .config   
## <chr> <chr> <dbl> <int> <dbl> <chr>   
## 1 accuracy binary 0.478 10 0.0184 Preprocessor1\_Model1  
## 2 f\_meas binary 0.491 10 0.0285 Preprocessor1\_Model1  
## 3 kap binary -0.0420 10 0.0356 Preprocessor1\_Model1  
## 4 precision binary 0.531 10 0.0221 Preprocessor1\_Model1  
## 5 recall binary 0.469 10 0.0413 Preprocessor1\_Model1  
## 6 roc\_auc binary 0.473 10 0.0189 Preprocessor1\_Model1  
## 7 sens binary 0.469 10 0.0413 Preprocessor1\_Model1  
## 8 spec binary 0.489 10 0.0435 Preprocessor1\_Model1

log\_res %>% collect\_metrics(summarize = FALSE)

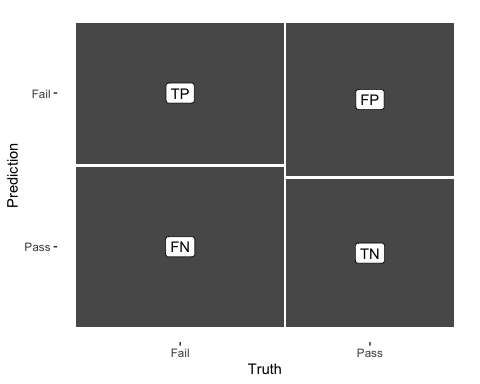
## # A tibble: 80 × 5  
## id .metric .estimator .estimate .config   
## <chr> <chr> <chr> <dbl> <chr>   
## 1 Fold01 recall binary 0.403 Preprocessor1\_Model1  
## 2 Fold01 precision binary 0.581 Preprocessor1\_Model1  
## 3 Fold01 f\_meas binary 0.476 Preprocessor1\_Model1  
## 4 Fold01 accuracy binary 0.509 Preprocessor1\_Model1  
## 5 Fold01 kap binary 0.0417 Preprocessor1\_Model1  
## 6 Fold01 sens binary 0.403 Preprocessor1\_Model1  
## 7 Fold01 spec binary 0.64 Preprocessor1\_Model1  
## 8 Fold01 roc\_auc binary 0.508 Preprocessor1\_Model1  
## 9 Fold02 recall binary 0.339 Preprocessor1\_Model1  
## 10 Fold02 precision binary 0.477 Preprocessor1\_Model1  
## # ℹ 70 more rows

**Collect results to compare further**

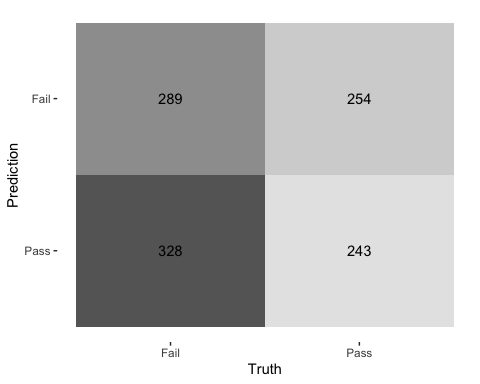
## `collect\_predictions()` and get confusion matrix{.smaller}  
  
log\_pred <- log\_res %>% collect\_predictions()  
  
log\_pred %>% conf\_mat(test, .pred\_class)

## Truth  
## Prediction Fail Pass  
## Fail 289 254  
## Pass 328 243

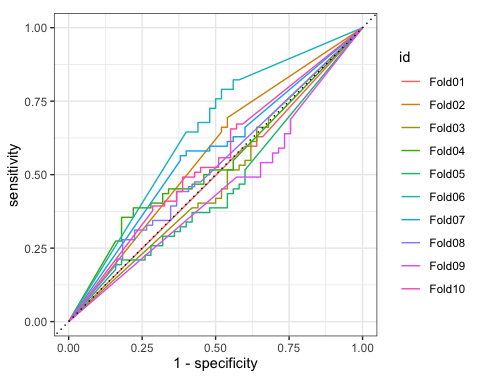
log\_pred %>%   
 conf\_mat(test, .pred\_class) %>%   
 autoplot(type = "mosaic") +  
 geom\_label(aes(  
 x = (xmax + xmin) / 2,   
 y = (ymax + ymin) / 2,   
 label = c("TP", "FN", "FP", "TN")))



log\_pred %>%   
 conf\_mat(test, .pred\_class) %>%   
 autoplot(type = "heatmap")



## ROC Curve  
  
log\_pred %>%   
 group\_by(id) %>% # id contains our folds  
 roc\_curve(test, .pred\_Pass) %>%   
 autoplot()



Logistic regression after cross-validation seems to have inferior results:

* Area under the ROC curve is just 47.3%, which is not only lower than in the naive model (55.4%), but even worse than just flipping a coin (50% chance).
* Sensitivity and specificity are below 50%, which indicates both large number of false positives and false negative alerts. The confusion matrix supports this conclusion, the number of false positives (328 is particularly troubling).
* Some ROC curves pass largely below the 50/50 line, these must be folds at which the model performs particularly poor. There is misalignment when some ROC-curves are higher than 50/50 line (good prediction quality) but some are lower (bad quality).

**DECISION TREE**

tree\_res <- tree\_wflow %>%   
 fit\_resamples(  
 resamples = bechdel\_folds,   
 metrics = metric\_set(  
 recall, precision, f\_meas, accuracy,  
 kap, roc\_auc, sens, spec),  
 control = control\_resamples(save\_pred = TRUE))   
  
  
tree\_res %>% collect\_metrics(summarize = TRUE)

## # A tibble: 8 × 6  
## .metric .estimator mean n std\_err .config   
## <chr> <chr> <dbl> <int> <dbl> <chr>   
## 1 accuracy binary 0.590 10 0.0131 Preprocessor1\_Model1  
## 2 f\_meas binary 0.632 10 0.0126 Preprocessor1\_Model1  
## 3 kap binary 0.168 10 0.0276 Preprocessor1\_Model1  
## 4 precision binary 0.629 10 0.0125 Preprocessor1\_Model1  
## 5 recall binary 0.637 10 0.0194 Preprocessor1\_Model1  
## 6 roc\_auc binary 0.591 10 0.0181 Preprocessor1\_Model1  
## 7 sens binary 0.637 10 0.0194 Preprocessor1\_Model1  
## 8 spec binary 0.530 10 0.0283 Preprocessor1\_Model1

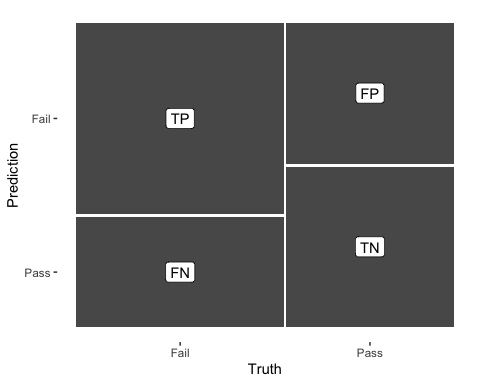
tree\_res %>% collect\_metrics(summarize = FALSE)

## # A tibble: 80 × 5  
## id .metric .estimator .estimate .config   
## <chr> <chr> <chr> <dbl> <chr>   
## 1 Fold01 recall binary 0.597 Preprocessor1\_Model1  
## 2 Fold01 precision binary 0.638 Preprocessor1\_Model1  
## 3 Fold01 f\_meas binary 0.617 Preprocessor1\_Model1  
## 4 Fold01 accuracy binary 0.589 Preprocessor1\_Model1  
## 5 Fold01 kap binary 0.175 Preprocessor1\_Model1  
## 6 Fold01 sens binary 0.597 Preprocessor1\_Model1  
## 7 Fold01 spec binary 0.58 Preprocessor1\_Model1  
## 8 Fold01 roc\_auc binary 0.577 Preprocessor1\_Model1  
## 9 Fold02 recall binary 0.677 Preprocessor1\_Model1  
## 10 Fold02 precision binary 0.646 Preprocessor1\_Model1  
## # ℹ 70 more rows

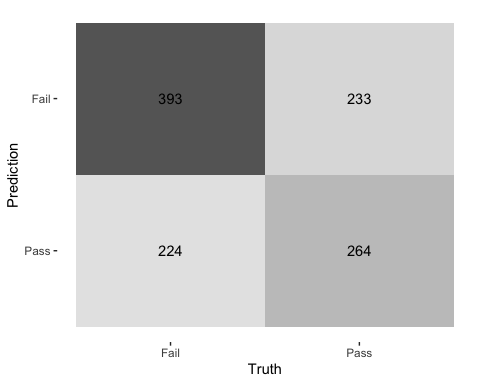
## `collect\_predictions()` and get confusion matrix{.smaller}  
  
tree\_pred <- tree\_res %>% collect\_predictions()  
  
tree\_pred %>% conf\_mat(test, .pred\_class)

## Truth  
## Prediction Fail Pass  
## Fail 393 233  
## Pass 224 264

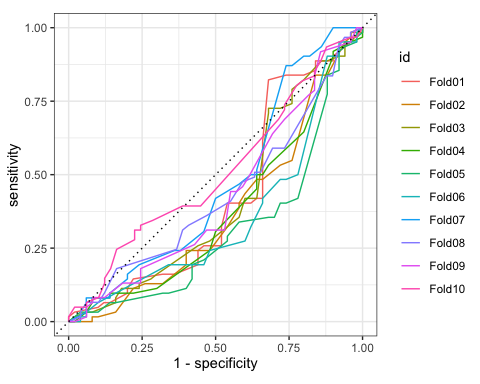
tree\_pred %>%   
 conf\_mat(test, .pred\_class) %>%   
 autoplot(type = "mosaic") +  
 geom\_label(aes(  
 x = (xmax + xmin) / 2,   
 y = (ymax + ymin) / 2,   
 label = c("TP", "FN", "FP", "TN")))



tree\_pred %>%   
 conf\_mat(test, .pred\_class) %>%   
 autoplot(type = "heatmap")



## ROC Curve  
  
tree\_pred %>%   
 group\_by(id) %>% # id contains our folds  
 roc\_curve(test, .pred\_Pass) %>%   
 autoplot()



Base decision tree model performs better and is more or less robust:

* Most ROC-curves pass below the 50/50 prediction line, but the graph seems to be inverted, so it is actually a good sign. In other way, if the model constantly predicts wrong, we could just take model results and do the opposite.  
  Area under the ROC curve is 59.1%, which is better than naive model accuracy (55.4%).
* Sensitivity, which is a measure of true positive rate, is quite high - 63.7%, which makes model a good candidate for prediction.
* Specificity, which is a measure of true negative rate, is not that good - only 53%, which indicates potential overlook of movies that would actually fail the Bechdel test.

**RANDOM FOREST**

rf\_res <- rf\_wflow %>%   
 fit\_resamples(  
 resamples = bechdel\_folds,   
 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 × 6  
## .metric .estimator mean n std\_err .config   
## <chr> <chr> <dbl> <int> <dbl> <chr>   
## 1 accuracy binary 0.637 10 0.0139 Preprocessor1\_Model1  
## 2 f\_meas binary 0.704 10 0.0115 Preprocessor1\_Model1  
## 3 kap binary 0.247 10 0.0291 Preprocessor1\_Model1  
## 4 precision binary 0.643 10 0.0112 Preprocessor1\_Model1  
## 5 recall binary 0.778 10 0.0155 Preprocessor1\_Model1  
## 6 roc\_auc binary 0.659 10 0.0216 Preprocessor1\_Model1  
## 7 sens binary 0.778 10 0.0155 Preprocessor1\_Model1  
## 8 spec binary 0.462 10 0.0221 Preprocessor1\_Model1

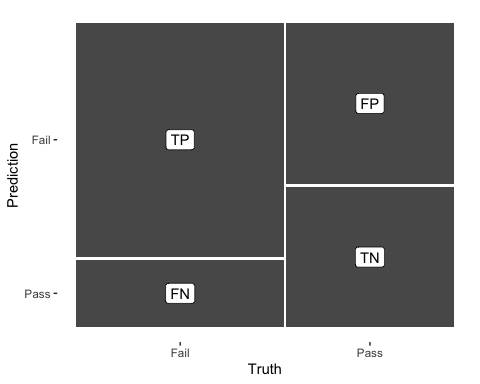
rf\_res %>% collect\_metrics(summarize = FALSE)

## # A tibble: 80 × 5  
## id .metric .estimator .estimate .config   
## <chr> <chr> <chr> <dbl> <chr>   
## 1 Fold01 recall binary 0.823 Preprocessor1\_Model1  
## 2 Fold01 precision binary 0.699 Preprocessor1\_Model1  
## 3 Fold01 f\_meas binary 0.756 Preprocessor1\_Model1  
## 4 Fold01 accuracy binary 0.705 Preprocessor1\_Model1  
## 5 Fold01 kap binary 0.391 Preprocessor1\_Model1  
## 6 Fold01 sens binary 0.823 Preprocessor1\_Model1  
## 7 Fold01 spec binary 0.56 Preprocessor1\_Model1  
## 8 Fold01 roc\_auc binary 0.812 Preprocessor1\_Model1  
## 9 Fold02 recall binary 0.726 Preprocessor1\_Model1  
## 10 Fold02 precision binary 0.6 Preprocessor1\_Model1  
## # ℹ 70 more rows

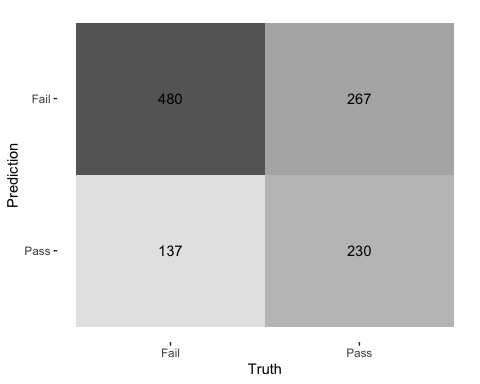
## `collect\_predictions()` and get confusion matrix{.smaller}  
  
rf\_pred <- rf\_res %>% collect\_predictions()  
  
rf\_pred %>% conf\_mat(test, .pred\_class)

## Truth  
## Prediction Fail Pass  
## Fail 480 267  
## Pass 137 230

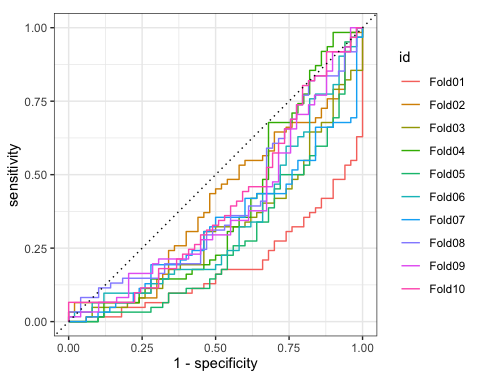
rf\_pred %>%   
 conf\_mat(test, .pred\_class) %>%   
 autoplot(type = "mosaic") +  
 geom\_label(aes(  
 x = (xmax + xmin) / 2,   
 y = (ymax + ymin) / 2,   
 label = c("TP", "FN", "FP", "TN")))



rf\_pred %>%   
 conf\_mat(test, .pred\_class) %>%   
 autoplot(type = "heatmap")



## ROC Curve  
  
rf\_pred %>%   
 group\_by(id) %>% # id contains our folds  
 roc\_curve(test, .pred\_Pass) %>%   
 autoplot()



Random forest model, making use of various decision trees, turns out to show even better results and is quite robust:

* All ROC-curves for the most part pass below the 50/50 line. Again, with all curves alignment, it is actually a good sign.  
  Area under the ROC curve is on average 65.9%, superior to all previous model (logistic regression, naive “Fail” prediction, and decision tree).
* Sensitivity is very high - 77.8%, which makes model a perfect candidate for prediction, better than in the decision tree model.
* However, specificity is below 50% (marginally worse than for the decision tree), which indicates potential overlook of movies that would actually fail the Bechdel test. A researcher needs to decide what is more important - predict pass or fail with more certainty).

**GRADIENT BOOSTING**

xgb\_res <- xgb\_wflow %>%   
 fit\_resamples(  
 resamples = bechdel\_folds,   
 metrics = metric\_set(  
 recall, precision, f\_meas, accuracy,  
 kap, roc\_auc, sens, spec),  
 control = control\_resamples(save\_pred = TRUE))   
  
  
xgb\_res %>% collect\_metrics(summarize = TRUE)

## # A tibble: 8 × 6  
## .metric .estimator mean n std\_err .config   
## <chr> <chr> <dbl> <int> <dbl> <chr>   
## 1 accuracy binary 0.634 10 0.0126 Preprocessor1\_Model1  
## 2 f\_meas binary 0.683 10 0.0105 Preprocessor1\_Model1  
## 3 kap binary 0.252 10 0.0270 Preprocessor1\_Model1  
## 4 precision binary 0.660 10 0.0136 Preprocessor1\_Model1  
## 5 recall binary 0.712 10 0.0171 Preprocessor1\_Model1  
## 6 roc\_auc binary 0.645 10 0.0169 Preprocessor1\_Model1  
## 7 sens binary 0.712 10 0.0171 Preprocessor1\_Model1  
## 8 spec binary 0.539 10 0.0295 Preprocessor1\_Model1

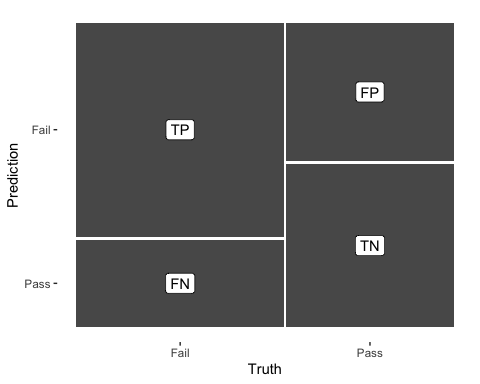
xgb\_res %>% collect\_metrics(summarize = FALSE)

## # A tibble: 80 × 5  
## id .metric .estimator .estimate .config   
## <chr> <chr> <chr> <dbl> <chr>   
## 1 Fold01 recall binary 0.694 Preprocessor1\_Model1  
## 2 Fold01 precision binary 0.729 Preprocessor1\_Model1  
## 3 Fold01 f\_meas binary 0.711 Preprocessor1\_Model1  
## 4 Fold01 accuracy binary 0.688 Preprocessor1\_Model1  
## 5 Fold01 kap binary 0.371 Preprocessor1\_Model1  
## 6 Fold01 sens binary 0.694 Preprocessor1\_Model1  
## 7 Fold01 spec binary 0.68 Preprocessor1\_Model1  
## 8 Fold01 roc\_auc binary 0.739 Preprocessor1\_Model1  
## 9 Fold02 recall binary 0.629 Preprocessor1\_Model1  
## 10 Fold02 precision binary 0.619 Preprocessor1\_Model1  
## # ℹ 70 more rows

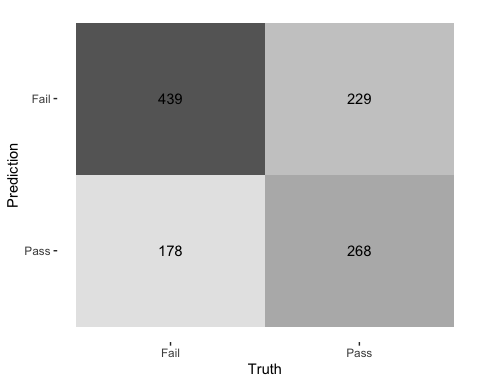
## `collect\_predictions()` and get confusion matrix{.smaller}  
  
xgb\_pred <- xgb\_res %>% collect\_predictions()  
  
xgb\_pred %>% conf\_mat(test, .pred\_class)

## Truth  
## Prediction Fail Pass  
## Fail 439 229  
## Pass 178 268

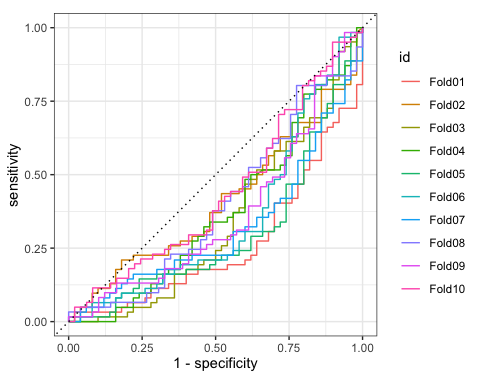
xgb\_pred %>%   
 conf\_mat(test, .pred\_class) %>%   
 autoplot(type = "mosaic") +  
 geom\_label(aes(  
 x = (xmax + xmin) / 2,   
 y = (ymax + ymin) / 2,   
 label = c("TP", "FN", "FP", "TN")))



xgb\_pred %>%   
 conf\_mat(test, .pred\_class) %>%   
 autoplot(type = "heatmap")



## ROC Curve  
  
xgb\_pred %>%   
 group\_by(id) %>% # id contains our folds  
 roc\_curve(test, .pred\_Pass) %>%   
 autoplot()



Gradient boosting model also looks promising, showing a little more modest results on average but being the most robust:

* Average area under the ROC-curve is 64.4%, marginally below that of random forest model (65.9%).
* All ROC curves lie almost entirely below the 50/50 line (good if there is alignment). Moreover, ROC-curves for different folds are not too far away, which indicates a great robustness to potential changes in data.
* Sensitivity is quite high at 71.1%, making the model a good candidate for prediction. It is lower, however, that in random forest model.
* Sensitivity is not great (53.9%), but better than in random forest model.

Overall, we would expect gradient boosting model to perform more robust and be useful in cases when we need a balanced approach in classifying positive and negative cases.

**K-NEAREST NEIGHBORS**

knn\_res <- knn\_wflow %>%   
 fit\_resamples(  
 resamples = bechdel\_folds,   
 metrics = metric\_set(  
 recall, precision, f\_meas, accuracy,  
 kap, roc\_auc, sens, spec),  
 control = control\_resamples(save\_pred = TRUE))

## → A | warning: While computing binary `precision()`, no predicted events were detected (i.e. `true\_positive + false\_positive = 0`).   
## Precision is undefined in this case, and `NA` will be returned.  
## Note that 61 true event(s) actually occured for the problematic event level, 'Fail'.

## There were issues with some computations A: x1  
## There were issues with some computations A: x1

##

knn\_res %>% collect\_metrics(summarize = TRUE)

## # A tibble: 8 × 6  
## .metric .estimator mean n std\_err .config   
## <chr> <chr> <dbl> <int> <dbl> <chr>   
## 1 accuracy binary 0.543 10 0.0110 Preprocessor1\_Model1  
## 2 f\_meas binary 0.712 9 0.00136 Preprocessor1\_Model1  
## 3 kap binary 0.000823 10 0.00424 Preprocessor1\_Model1  
## 4 precision binary 0.554 9 0.00102 Preprocessor1\_Model1  
## 5 recall binary 0.897 10 0.0997 Preprocessor1\_Model1  
## 6 roc\_auc binary 0.548 10 0.0231 Preprocessor1\_Model1  
## 7 sens binary 0.897 10 0.0997 Preprocessor1\_Model1  
## 8 spec binary 0.104 10 0.0996 Preprocessor1\_Model1

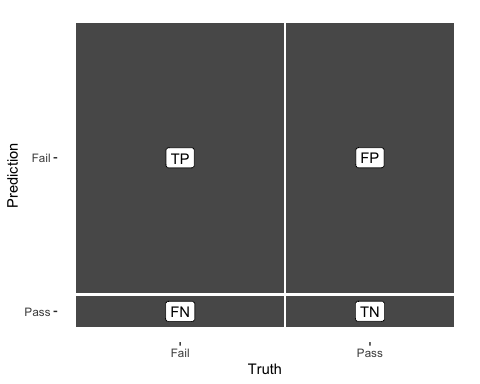
knn\_res %>% collect\_metrics(summarize = FALSE)

## # A tibble: 80 × 5  
## id .metric .estimator .estimate .config   
## <chr> <chr> <chr> <dbl> <chr>   
## 1 Fold01 recall binary 1 Preprocessor1\_Model1  
## 2 Fold01 precision binary 0.559 Preprocessor1\_Model1  
## 3 Fold01 f\_meas binary 0.717 Preprocessor1\_Model1  
## 4 Fold01 accuracy binary 0.562 Preprocessor1\_Model1  
## 5 Fold01 kap binary 0.0221 Preprocessor1\_Model1  
## 6 Fold01 sens binary 1 Preprocessor1\_Model1  
## 7 Fold01 spec binary 0.02 Preprocessor1\_Model1  
## 8 Fold01 roc\_auc binary 0.714 Preprocessor1\_Model1  
## 9 Fold02 recall binary 0.984 Preprocessor1\_Model1  
## 10 Fold02 precision binary 0.550 Preprocessor1\_Model1  
## # ℹ 70 more rows

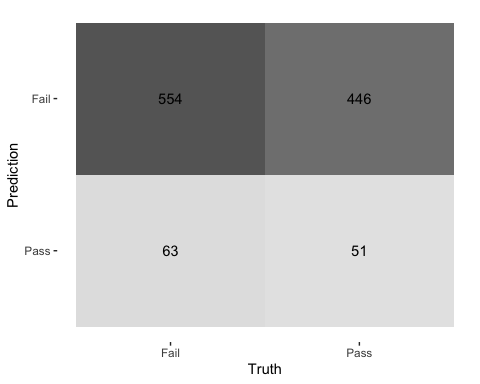
## `collect\_predictions()` and get confusion matrix{.smaller}  
  
knn\_pred <- knn\_res %>% collect\_predictions()  
  
knn\_pred %>% conf\_mat(test, .pred\_class)

## Truth  
## Prediction Fail Pass  
## Fail 554 446  
## Pass 63 51

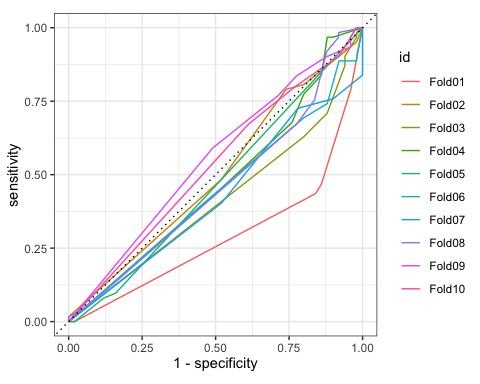
knn\_pred %>%   
 conf\_mat(test, .pred\_class) %>%   
 autoplot(type = "mosaic") +  
 geom\_label(aes(  
 x = (xmax + xmin) / 2,   
 y = (ymax + ymin) / 2,   
 label = c("TP", "FN", "FP", "TN")))



knn\_pred %>%   
 conf\_mat(test, .pred\_class) %>%   
 autoplot(type = "heatmap")



## ROC Curve  
  
knn\_pred %>%   
 group\_by(id) %>% # id contains our folds  
 roc\_curve(test, .pred\_Pass) %>%   
 autoplot()



K-nearest neighbors algorithm gives controversial results, with bad performance on average but the best in specificity:

* Average area under the ROC-curve is only 54.8%, marginally below naive “Fail prediction”
* ROC-curves are sometimes above, sometimes below 50/50 line - not a great robustness
* Sensitivity is unusually high - almost 90%, which makes pass predictions very reliable
* Specificity is unusually low - around 10%, which makes fail predictions very unreliable

Overall, model seems to predict test failure almost all the time. When it does predict a pass, it is very reliable result (hence, high sensitivity), but in the general classification task (predict both pass and fail) the model is not so useful.

**MODEL CHOICE**

To summarize, we think that two models are useful for the classification purposes: **random forest and gradient boosting**. The choice would depend on researcher’s goals:

**Random forest** gives the best predictions on average and should be used when reliability of a pass prediction (true positive rate) is more important than reliability of a fail prediction (true negative rate).

**Gradient boosting** is marginally worse on average, but is useful when a researcher wants to apply a more balanced approach between positive and negative predictions. It also seems to be more robust, which makes it safer to apply on unknown data.

# Deliverables

There is a lot of explanatory text, comments, etc. You do not need these, so delete them and produce a stand-alone document that you could share with someone. Knit the edited and completed R Markdown (Rmd) file as a Word or HTML document (use the “Knit” button at the top of the script editor window) and upload it to Canvas. You must be commiting and pushing your changes to your own Github repo as you go along.

# Details

* Who did you collaborate with: None
* Approximately how much time did you spend on this problem set: 8 hours
* What, if anything, gave you the most trouble: ANSWER HERE

**Please seek out help when you need it,** and remember the [15-minute rule](https://dsb2023.netlify.app/syllabus/#the-15-minute-rule). You know enough R (and have enough examples of code from class and your readings) to be able to do this. If you get stuck, ask for help from others, post a question on Slack– and remember that I am here to help too!

As a true test to yourself, do you understand the code you submitted and are you able to explain it to someone else?

# Rubric

13/13: Problem set is 100% completed. Every question was attempted and answered, and most answers are correct. Code is well-documented (both self-documented and with additional comments as necessary). Used tidyverse, instead of base R. Graphs and tables are properly labelled. Analysis is clear and easy to follow, either because graphs are labeled clearly or you’ve written additional text to describe how you interpret the output. Multiple Github commits. Work is exceptional. I will not assign these often.

8/13: Problem set is 60–80% complete and most answers are correct. This is the expected level of performance. Solid effort. Hits all the elements. No clear mistakes. Easy to follow (both the code and the output). A few Github commits.

5/13: Problem set is less than 60% complete and/or most answers are incorrect. This indicates that you need to improve next time. I will hopefully not assign these often. Displays minimal effort. Doesn’t complete all components. Code is poorly written and not documented. Uses the same type of plot for each graph, or doesn’t use plots appropriate for the variables being analyzed. No Github commits.