Homework 4: Machine Learning

GUIDO BOZZANO

2023-06-04

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", …

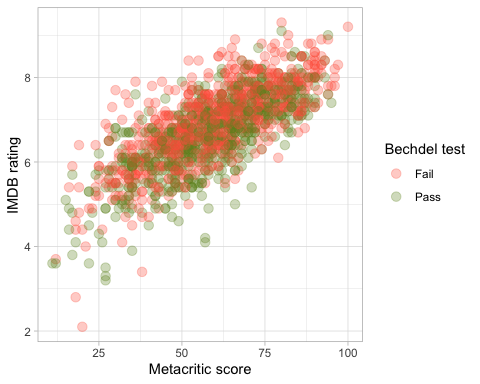
How many films fail/pass the test, both as a number and as a %?

#To answer this question I begin by using the bechdel data set  
  
bechdel %>%   
 group\_by(test) %>% #First I group\_by the information by test to understand films by this group  
 summarise(count=n()) %>% #Then I perform a count of the number of films that passed or failed the test  
 mutate(prop = round(count/sum(count)\*100,1)) #Last I mutate a new column with the proportion by the above groupin

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

## 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()



# 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.

#To check the counts and proportions I repeat the first code but I run it on the bechdel\_train and bechdel\_test datasets.  
#From the tables below I can conclude that both the train and test datasets are rightfully distributed since the proportions of pass and test are the same between the two datasets and between the entire data set (55.4% for Fail and 44.6% for Pass)  
  
bechdel\_train %>%   
 group\_by(test) %>%   
 summarise(count=n()) %>%   
 mutate(prop = round(count/sum(count)\*100,1))

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

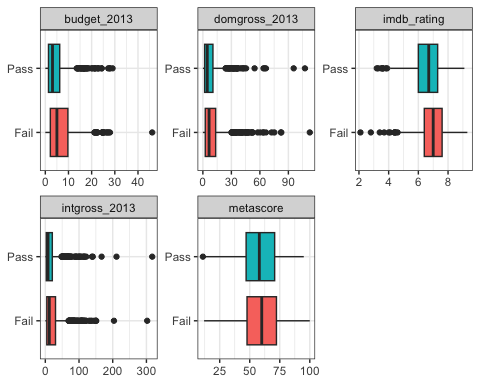
bechdel\_test %>%   
 group\_by(test) %>%   
 summarise(count=n()) %>%   
 mutate(prop = round(count/sum(count)\*100,1))

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

## 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)

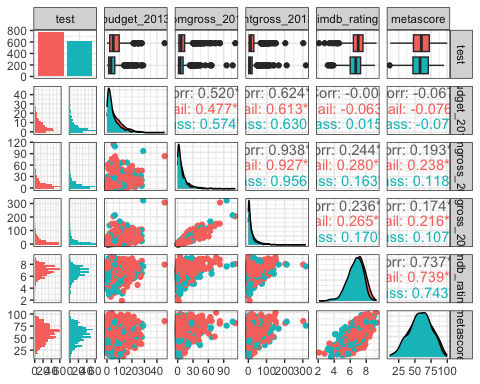


#From the boxplots below I see that there seems to be an outlier on the budget\_2013 variable that failed the test, on the domgross\_2013 variable that passed and failed, and the intgross\_2013 that passed and failed.  
#In addition there seems to be an outlier on the imdb\_rating that failed

## 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()



#To being with, there seems to be a positive correlation between imdb\_rating and metascore (0.737). This positive correlation is equally stronger for movies that passed or failed the test. As both variables are a rating it makes sense they are positively correlated because it means that the movies they rate are rated in the same direction (i.e: if it has a high score in imdb\_rating it will probably have a high score in metascore and viceversa. This also applies to bad scores)  
#Furthermore, both imdb\_rating and metascore seem to have no correlation with the budget variable (-0.004 and -0.067 respectively). Therefore, from this information it looks like having a big or small budget does not necesarily mean that the rating ot metascore will be higher or lower.  
#In addition, in terms of revenues there's a strong correlation (0.938) between domestic and international revenues which means that if the movie performs well domestically usually will perform well internationally and viceversa

## 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

#By analyzing the test pass rate by genre, I can conclude that Action movies have one of the highest fail rates (70%) given the large number of cases. There are also some particular genres that reach 100% but the sample is particularly small.  
#On the other hand, both Horror and Comedy perfom really well in terms of pass rates.  
  
#By analyzing the test pass rate by rates, I can conlude that PG-13 have the highest pass rates with 47%, followed by R movies. On the other hand, the G and PG movies seem to perform really bad in terms of test pass rates.

# 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

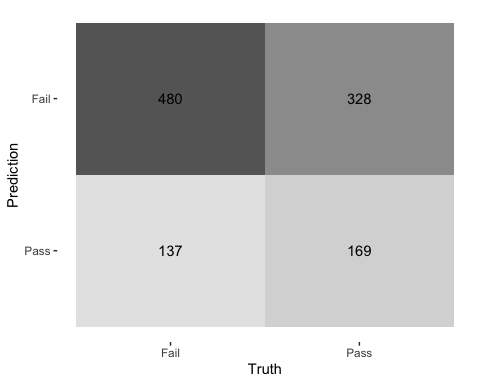
lr\_fit %>%  
 broom::tidy()

## # 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))

### Confusion matrix

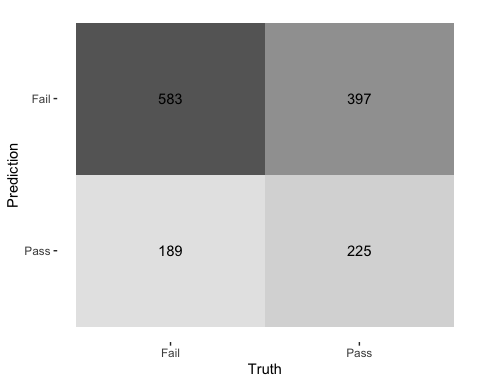
lr\_preds %>%   
 conf\_mat(truth = test, estimate = .pred\_class) %>%   
 autoplot(type = "heatmap")



## Decision Tree

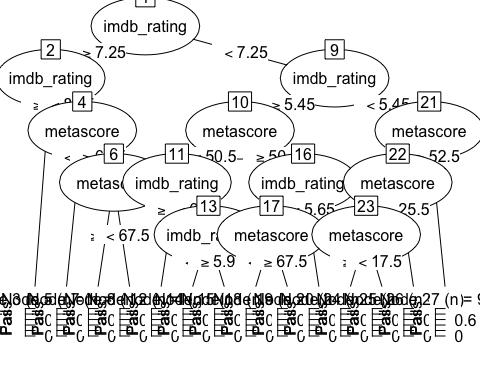
tree\_preds <- tree\_fit %>%  
 augment(new\_data = bechdel) %>%  
 mutate(.pred\_match = if\_else(test == .pred\_class, 1, 0))

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



## 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)



# Cross Validation

Run the code below. What does it return?

set.seed(123)  
bechdel\_folds <- vfold\_cv(data = bechdel\_train,   
 v = 3,   
 strata = test)  
bechdel\_folds

## # 3-fold cross-validation using stratification   
## # A tibble: 3 × 2  
## splits id   
## <list> <chr>  
## 1 <split [742/372]> Fold1  
## 2 <split [742/372]> Fold2  
## 3 <split [744/370]> Fold3

#The code below returns the number of folds we are going to train and test the data, the code creates 10 folds of roughly equal size that are used to train and then test the model  
#EDIT: I have modified the original code from 10 folds to 3 folds because the model comparison at the bottom takes too long to run

## fit\_resamples()

Trains and tests a resampled model.

lr\_fit <- lr\_mod %>%  
 fit\_resamples(  
 test ~ metascore + imdb\_rating,  
 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 0.585 3 0.0143 Preprocessor1\_Model1  
## 2 roc\_auc binary 0.602 3 0.0190 Preprocessor1\_Model1

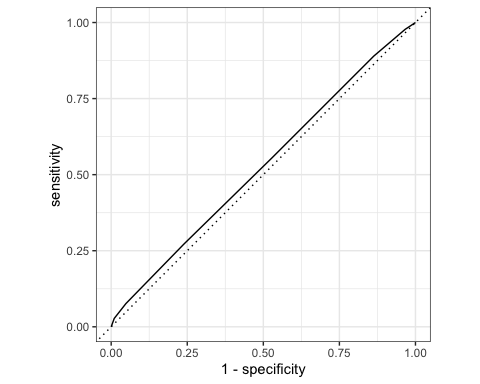
collect\_metrics(tree\_fit)

## # A tibble: 2 × 6  
## .metric .estimator mean n std\_err .config   
## <chr> <chr> <dbl> <int> <dbl> <chr>   
## 1 accuracy binary 0.554 3 0.00458 Preprocessor1\_Model1  
## 2 roc\_auc binary 0.524 3 0.0122 Preprocessor1\_Model1

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: 9 × 3  
## .threshold specificity sensitivity  
## <dbl> <dbl> <dbl>  
## 1 -Inf 0 1   
## 2 0.346 0 1   
## 3 0.430 0.0342 0.977   
## 4 0.554 0.137 0.890   
## 5 0.556 0.471 0.556   
## 6 0.569 0.761 0.272   
## 7 0.819 0.952 0.0762  
## 8 0.867 0.990 0.0276  
## 9 Inf 1 0

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



# Build a better training set with recipes

## Preprocessing options

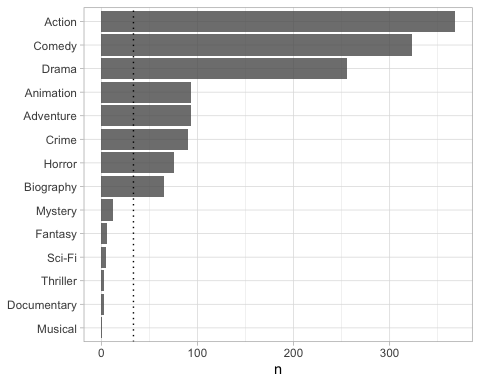
* Encode categorical predictors
* Center and scale variables
* Handle class imbalance
* Impute missing data
* Perform dimensionality reduction
* … …

## 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? No, the model wouldn’t have a coefficient for a movie that is rated NC-17 if there is no information from that genre in the training data
* What will happen if the test data includes a film with rated NC-17? If the test data includes a film with the rated NC-17, the model will consider it as an unseen category. The model will not necessarily understand how to treat films rated NC-17. However, the model will use the relationships it learned from the training data to make predictions for the film with the NC-17 rating. Having said that, the model’s prediction for the film with the NC-17 rating may be less accurate or less reliable compared to films with other ratings that are present in the training data.

## 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())  
  
  
  
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? I have specified 5 models: Logistic regression, Decision Tree, Random Forest, Boosted tree (XGBoost) and K-nearest neighbour (k-NN)
2. What’s the difference between a model specification and a workflow? A model specification refers to the definition of the specific model engine the user wants to use, including the choice of the modeling technique and the associated settings or parameters. From the examples above I can use the Logistic Regression specification as an example where I we set the model engine as “generalized linear models” and set the mode of the model to classification. On the other hand, the workflow refers to a step by step process from data preparation and transformation to model fitting and evaluation. In the example above, from the logistic regression I defined a workflow using the recipe defined above and the model specification from the logistic regression. This way of working, allows the workflow to replicate itself quite easily from one model to the other. The user needs to adjust only a few variables and then is able to run another workflow using a different model specification.
3. Do you need to add a formula (e.g., test ~ .) if you have a recipe? I believe it’s not necessary to add a formula since the recipe itself may help the user to automatically define the relationship between the variables

# 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.

## Evaluate Models  
  
## Logistic regression results{.smaller}  
  
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: x1There were issues with some computations A: x2 B: x2There were issues with some computations A: x3 B: x2There were issues with some computations A: x3 B: x3

# Show average performance over all folds (note that we use log\_res):  
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 0.434 3 0.0150 Preprocessor1\_Model1  
## 2 f\_meas binary 0.346 3 0.0772 Preprocessor1\_Model1  
## 3 kap binary -0.0916 3 0.0140 Preprocessor1\_Model1  
## 4 precision binary 0.470 3 0.0244 Preprocessor1\_Model1  
## 5 recall binary 0.290 3 0.0978 Preprocessor1\_Model1  
## 6 roc\_auc binary 0.450 3 0.0115 Preprocessor1\_Model1  
## 7 sens binary 0.290 3 0.0978 Preprocessor1\_Model1  
## 8 spec binary 0.614 3 0.0918 Preprocessor1\_Model1

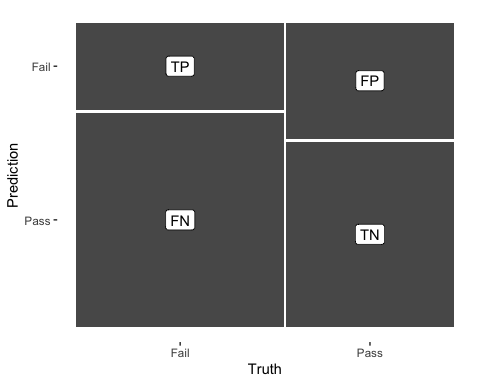
# Show performance for every single fold:  
log\_res %>% collect\_metrics(summarize = FALSE)

## # A tibble: 24 × 5  
## id .metric .estimator .estimate .config   
## <chr> <chr> <chr> <dbl> <chr>   
## 1 Fold1 recall binary 0.184 Preprocessor1\_Model1  
## 2 Fold1 precision binary 0.463 Preprocessor1\_Model1  
## 3 Fold1 f\_meas binary 0.264 Preprocessor1\_Model1  
## 4 Fold1 accuracy binary 0.430 Preprocessor1\_Model1  
## 5 Fold1 kap binary -0.0751 Preprocessor1\_Model1  
## 6 Fold1 sens binary 0.184 Preprocessor1\_Model1  
## 7 Fold1 spec binary 0.735 Preprocessor1\_Model1  
## 8 Fold1 roc\_auc binary 0.464 Preprocessor1\_Model1  
## 9 Fold2 recall binary 0.485 Preprocessor1\_Model1  
## 10 Fold2 precision binary 0.515 Preprocessor1\_Model1  
## # ℹ 14 more rows

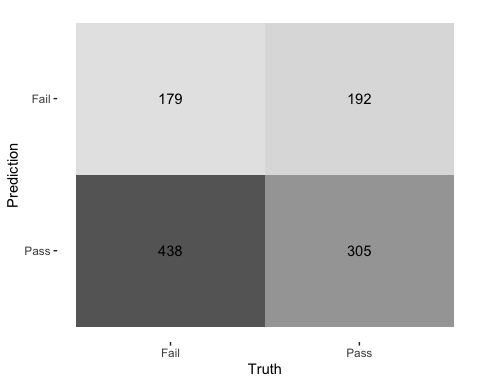
## `collect\_predictions()` and get confusion matrix{.smaller}  
  
log\_pred <- log\_res %>% collect\_predictions()  
  
log\_pred %>% conf\_mat(test, .pred\_class)

## Truth  
## Prediction Fail Pass  
## Fail 179 192  
## Pass 438 305

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")))



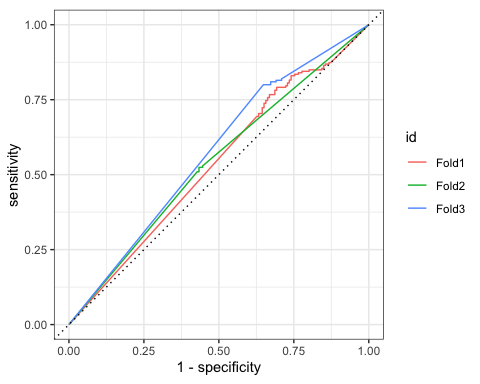
log\_pred %>%   
 conf\_mat(test, .pred\_class) %>%   
 autoplot(type = "heatmap")



log\_pred

## # A tibble: 1,114 × 7  
## id .pred\_class .row .pred\_Fail .pred\_Pass test .config   
## <chr> <fct> <int> <dbl> <dbl> <fct> <chr>   
## 1 Fold1 Pass 1 2.22e-16 1 e+ 0 Fail Preprocessor1\_Model1  
## 2 Fold1 Pass 2 2.22e-16 1 e+ 0 Fail Preprocessor1\_Model1  
## 3 Fold1 Pass 4 2.22e-16 1 e+ 0 Fail Preprocessor1\_Model1  
## 4 Fold1 Pass 6 2.22e-16 1 e+ 0 Fail Preprocessor1\_Model1  
## 5 Fold1 Pass 8 2.22e-16 1 e+ 0 Fail Preprocessor1\_Model1  
## 6 Fold1 Fail 9 1 e+ 0 2.22e-16 Fail Preprocessor1\_Model1  
## 7 Fold1 Pass 14 2.22e-16 1 e+ 0 Fail Preprocessor1\_Model1  
## 8 Fold1 Pass 16 2.22e-16 1 e+ 0 Fail Preprocessor1\_Model1  
## 9 Fold1 Pass 20 2.22e-16 1 e+ 0 Fail Preprocessor1\_Model1  
## 10 Fold1 Pass 24 2.22e-16 1 e+ 0 Fail Preprocessor1\_Model1  
## # ℹ 1,104 more rows

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



## Decision Tree results  
  
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 0.588 3 0.0147 Preprocessor1\_Model1  
## 2 f\_meas binary 0.633 3 0.0244 Preprocessor1\_Model1  
## 3 kap binary 0.163 3 0.0257 Preprocessor1\_Model1  
## 4 precision binary 0.623 3 0.0105 Preprocessor1\_Model1  
## 5 recall binary 0.647 3 0.0473 Preprocessor1\_Model1  
## 6 roc\_auc binary 0.589 3 0.0151 Preprocessor1\_Model1  
## 7 sens binary 0.647 3 0.0473 Preprocessor1\_Model1  
## 8 spec binary 0.515 3 0.0393 Preprocessor1\_Model1

## 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 0.634 3 0.00727 Preprocessor1\_Model1  
## 2 f\_meas binary 0.702 3 0.00975 Preprocessor1\_Model1  
## 3 kap binary 0.238 3 0.0177 Preprocessor1\_Model1  
## 4 precision binary 0.640 3 0.00999 Preprocessor1\_Model1  
## 5 recall binary 0.780 3 0.0332 Preprocessor1\_Model1  
## 6 roc\_auc binary 0.665 3 0.0140 Preprocessor1\_Model1  
## 7 sens binary 0.780 3 0.0332 Preprocessor1\_Model1  
## 8 spec binary 0.453 3 0.0448 Preprocessor1\_Model1

## Boosted tree - XGBoost  
  
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 0.606 3 0.00256 Preprocessor1\_Model1  
## 2 f\_meas binary 0.657 3 0.0107 Preprocessor1\_Model1  
## 3 kap binary 0.195 3 0.00120 Preprocessor1\_Model1  
## 4 precision binary 0.634 3 0.00426 Preprocessor1\_Model1  
## 5 recall binary 0.684 3 0.0288 Preprocessor1\_Model1  
## 6 roc\_auc binary 0.635 3 0.00299 Preprocessor1\_Model1  
## 7 sens binary 0.684 3 0.0288 Preprocessor1\_Model1  
## 8 spec binary 0.509 3 0.0303 Preprocessor1\_Model1

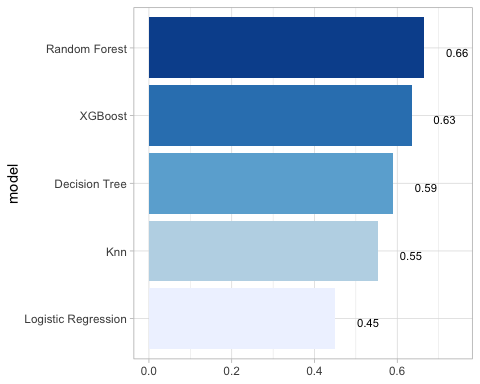
## K-nearest neighbour  
  
knn\_res <-   
 knn\_wflow %>%   
 fit\_resamples(  
 resamples = bechdel\_folds,   
 metrics = metric\_set(  
 recall, precision, f\_meas,   
 accuracy, kap,  
 roc\_auc, sens, spec),  
 control = control\_resamples(save\_pred = TRUE)  
 )

## → 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 206 true event(s) actually occured for the problematic event level, 'Fail'.  
## There were issues with some computations A: x1There 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 0.518 3 0.0359 Preprocessor1\_Model1  
## 2 f\_meas binary 0.713 2 0.000120 Preprocessor1\_Model1  
## 3 kap binary 0 3 0 Preprocessor1\_Model1  
## 4 precision binary 0.554 2 0.000145 Preprocessor1\_Model1  
## 5 recall binary 0.667 3 0.333 Preprocessor1\_Model1  
## 6 roc\_auc binary 0.553 3 0.0284 Preprocessor1\_Model1  
## 7 sens binary 0.667 3 0.333 Preprocessor1\_Model1  
## 8 spec binary 0.333 3 0.333 Preprocessor1\_Model1

## Model Comparison  
  
log\_metrics <-   
 log\_res %>%   
 collect\_metrics(summarise = TRUE) %>%  
 # add the name of the model to every row  
 mutate(model = "Logistic Regression")   
  
tree\_metrics <-   
 tree\_res %>%   
 collect\_metrics(summarise = TRUE) %>%  
 mutate(model = "Decision Tree")  
  
rf\_metrics <-   
 rf\_res %>%   
 collect\_metrics(summarise = TRUE) %>%  
 mutate(model = "Random Forest")  
  
xgb\_metrics <-   
 xgb\_res %>%   
 collect\_metrics(summarise = TRUE) %>%  
 mutate(model = "XGBoost")  
  
knn\_metrics <-   
 knn\_res %>%   
 collect\_metrics(summarise = TRUE) %>%  
 mutate(model = "Knn")  
  
# create dataframe with all models  
model\_compare <- bind\_rows(log\_metrics,  
 tree\_metrics,  
 rf\_metrics,  
 xgb\_metrics,  
 knn\_metrics)   
  
#Pivot wider to create barplot  
 model\_comp <- model\_compare %>%   
 select(model, .metric, mean, std\_err) %>%   
 pivot\_wider(names\_from = .metric, values\_from = c(mean, std\_err))   
  
# show mean are under the curve (ROC-AUC) for every model  
model\_comp %>%   
 arrange(mean\_roc\_auc) %>%   
 mutate(model = fct\_reorder(model, mean\_roc\_auc)) %>% # order results  
 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  
 )+  
 theme\_light()+  
 theme(legend.position = "none")+  
 labs(y = NULL)



## `last\_fit()` on test set  
  
# - `last\_fit()` fits a model to the whole training data and evaluates it on the test set.   
# - provide the workflow object of the best model as well as the data split object (not the training data).   
   
last\_fit\_xgb <- last\_fit(xgb\_wflow,   
 split = data\_split,  
 metrics = metric\_set(  
 accuracy, f\_meas, kap, precision,  
 recall, roc\_auc, sens, spec))  
  
last\_fit\_xgb %>% collect\_metrics(summarize = TRUE)

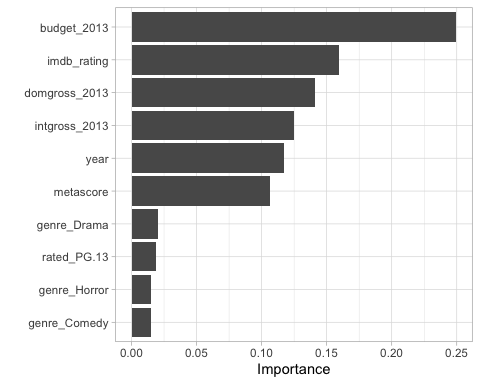
## # A tibble: 8 × 4  
## .metric .estimator .estimate .config   
## <chr> <chr> <dbl> <chr>   
## 1 accuracy binary 0.568 Preprocessor1\_Model1  
## 2 f\_meas binary 0.630 Preprocessor1\_Model1  
## 3 kap binary 0.114 Preprocessor1\_Model1  
## 4 precision binary 0.599 Preprocessor1\_Model1  
## 5 recall binary 0.665 Preprocessor1\_Model1  
## 6 sens binary 0.665 Preprocessor1\_Model1  
## 7 spec binary 0.448 Preprocessor1\_Model1  
## 8 roc\_auc binary 0.610 Preprocessor1\_Model1

#Compare to training  
xgb\_res %>% collect\_metrics(summarize = TRUE)

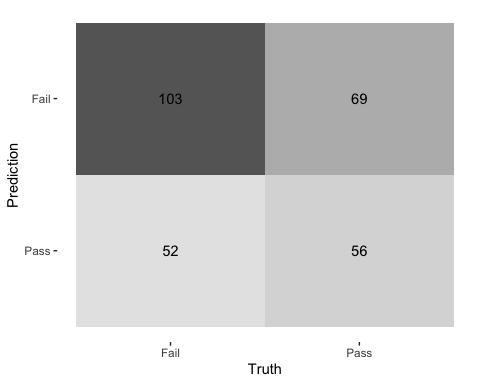
## # A tibble: 8 × 6  
## .metric .estimator mean n std\_err .config   
## <chr> <chr> <dbl> <int> <dbl> <chr>   
## 1 accuracy binary 0.606 3 0.00256 Preprocessor1\_Model1  
## 2 f\_meas binary 0.657 3 0.0107 Preprocessor1\_Model1  
## 3 kap binary 0.195 3 0.00120 Preprocessor1\_Model1  
## 4 precision binary 0.634 3 0.00426 Preprocessor1\_Model1  
## 5 recall binary 0.684 3 0.0288 Preprocessor1\_Model1  
## 6 roc\_auc binary 0.635 3 0.00299 Preprocessor1\_Model1  
## 7 sens binary 0.684 3 0.0288 Preprocessor1\_Model1  
## 8 spec binary 0.509 3 0.0303 Preprocessor1\_Model1

## Variable importance using `{vip}` package  
  
library(vip)  
  
last\_fit\_xgb %>%   
 pluck(".workflow", 1) %>%   
 pull\_workflow\_fit() %>%   
 vip(num\_features = 10) +  
 theme\_light()

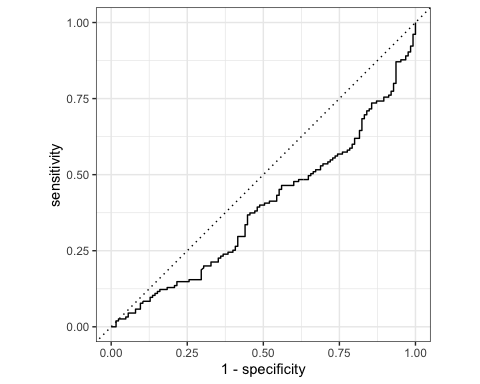
## Warning: `pull\_workflow\_fit()` was deprecated in workflows 0.2.3.  
## ℹ Please use `extract\_fit\_parsnip()` instead.  
## This warning is displayed once every 8 hours.  
## Call `lifecycle::last\_lifecycle\_warnings()` to see where this warning was  
## generated.



## Final Confusion Matrix  
  
last\_fit\_xgb %>%  
 collect\_predictions() %>%   
 conf\_mat(test, .pred\_class) %>%   
 autoplot(type = "heatmap")



## Final ROC curve  
last\_fit\_xgb %>%   
 collect\_predictions() %>%   
 roc\_curve(test, .pred\_Pass) %>%   
 autoplot()



#To conclude, the model that has the best classification is the Random Forest since it has the biggest accuracy with 66%

# 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: TYPE NAMES HERE
* Approximately how much time did you spend on this problem set: 3 hours
* What, if anything, gave you the most trouble: The last problem, adapting the code from the slides to a new data set. Even though there were only a few minor modifications, it involved interpreting the code from the slides and understanding which variables are meaningful from the new code so that it can run without any problems

**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.