432 Class 24

Thomas E. Love, Ph.D.

2024-04-16

## Today’s Topic

Fitting logistic regressions using tidymodels packages

* Pre-processing activities
* Model building (with multiple fitting engines)
* Measuring model effectiveness
* Creating a model workflow

## Setup

knitr::opts\_chunk$set(comment=NA)  
options(width = 80)  
  
library(janitor)

Attaching package: 'janitor'

The following objects are masked from 'package:stats':  
  
 chisq.test, fisher.test

library(gt)  
library(naniar)  
library(rstanarm)

Loading required package: Rcpp

This is rstanarm version 2.26.1

- See https://mc-stan.org/rstanarm/articles/priors for changes to default priors!

- Default priors may change, so it's safest to specify priors, even if equivalent to the defaults.

- For execution on a local, multicore CPU with excess RAM we recommend calling

options(mc.cores = parallel::detectCores())

library(rms)

Loading required package: Hmisc

Attaching package: 'Hmisc'

The following object is masked from 'package:gt':  
  
 html

The following objects are masked from 'package:base':  
  
 format.pval, units

library(tidymodels)

── Attaching packages ────────────────────────────────────── tidymodels 1.1.1 ──

✔ broom 1.0.5 ✔ recipes 1.0.9  
✔ dials 1.2.0 ✔ rsample 1.2.0  
✔ dplyr 1.1.4 ✔ tibble 3.2.1  
✔ ggplot2 3.4.4 ✔ tidyr 1.3.0  
✔ infer 1.0.5 ✔ tune 1.1.2  
✔ modeldata 1.2.0 ✔ workflows 1.1.3  
✔ parsnip 1.1.1 ✔ workflowsets 1.0.1  
✔ purrr 1.0.2 ✔ yardstick 1.2.0

── Conflicts ───────────────────────────────────────── tidymodels\_conflicts() ──  
✖ purrr::discard() masks scales::discard()  
✖ dplyr::filter() masks stats::filter()  
✖ dplyr::lag() masks stats::lag()  
✖ rsample::populate() masks Rcpp::populate()  
✖ dplyr::src() masks Hmisc::src()  
✖ recipes::step() masks stats::step()  
✖ dplyr::summarize() masks Hmisc::summarize()  
✖ parsnip::translate() masks Hmisc::translate()  
• Search for functions across packages at https://www.tidymodels.org/find/

library(tidyverse)

── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──  
✔ forcats 1.0.0 ✔ readr 2.1.5  
✔ lubridate 1.9.3 ✔ stringr 1.5.1

── Conflicts ────────────────────────────────────────── tidyverse\_conflicts() ──  
✖ readr::col\_factor() masks scales::col\_factor()  
✖ purrr::discard() masks scales::discard()  
✖ dplyr::filter() masks stats::filter()  
✖ stringr::fixed() masks recipes::fixed()  
✖ dplyr::lag() masks stats::lag()  
✖ readr::spec() masks yardstick::spec()  
✖ dplyr::src() masks Hmisc::src()  
✖ dplyr::summarize() masks Hmisc::summarize()  
ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors

theme\_set(theme\_bw())

## Today’s Data (from Class 10)

fram\_raw <- read\_csv("c24/data/framingham.csv",  
 show\_col\_types = FALSE) |>  
 type.convert(as.is = FALSE) |>  
 clean\_names()

The variables describe n = 4238 adults examined at baseline, then followed for 10 years to see if they developed incident coronary heart disease. Our outcome (below) has no missing values.

fram\_raw |> tabyl(ten\_year\_chd)

ten\_year\_chd n percent  
 0 3594 0.8480415  
 1 644 0.1519585

## Data Cleanup

fram\_new <- fram\_raw |>  
 rename(cigs = "cigs\_per\_day",  
 stroke = "prevalent\_stroke",  
 hrate = "heart\_rate",  
 sbp = "sys\_bp",  
 chd10\_n = "ten\_year\_chd") |>  
 mutate(educ = fct\_recode(factor(education),   
 "Some HS" = "1",  
 "HS grad" = "2",  
 "Some Coll" = "3",  
 "Coll grad" = "4")) |>  
 mutate(chd10\_f = fct\_recode(factor(chd10\_n),  
 "chd" = "1", "chd\_no" = "0")) |>  
 select(subj\_id, chd10\_n, chd10\_f, age,   
 cigs, educ, hrate, sbp, stroke)

## Today’s (main) Variables

| Variable | Description |
| --- | --- |
| subj\_id | identifying code added by Dr. Love |
| chd10\_n | (numeric) 1 = coronary heart disease in next 10y |
| chd10\_f | (factor) “chd” or “chd\_no” in next ten years |
| age | in years (range is 32 to 70) |
| cigs | number of cigarettes smoked per day |
| educ | 4-level factor: educational attainment |
| hrate | heart rate in beats per minute |
| sbp | systolic blood pressure in mm Hg |
| stroke | 1 = history of stroke, else 0 |

## Steps we’ll describe today

1. Prepare our (binary) outcome.
2. Split the data into training and testing samples.
3. Build a recipe for our model.
   * Specify roles for outcome and predictors.
   * Deal with missing data in a reasonable way.
   * Complete all necessary pre-processing so we can fit models.
4. Specify a modeling engine for each fit we will create.

## Steps we’ll describe today

1. Create a workflow for each engine and fit model to the training data.
2. Compare coefficients graphically from two modeling approaches.
3. Assess performance in the models we create in the training data.
4. Compare multiple models based on their performance in test data.

Key Reference: Kuhn and Silge, [Tidy Modeling with R](https://www.tmwr.org/)

## Stage 1. Prepare our outcome.

To do logistic regression using tidymodels, we’ll want our binary outcome to be a factor variable.

str(fram\_new$chd10\_f)

Factor w/ 2 levels "chd\_no","chd": 1 1 1 2 1 1 2 1 1 1 ...

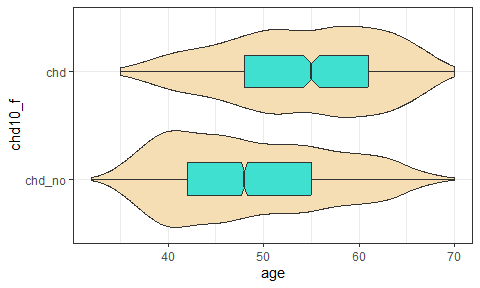
fram\_new |> tabyl(chd10\_f, chd10\_n)

chd10\_f 0 1  
 chd\_no 3594 0  
 chd 0 644

## Working with Binary Outcome Models

Does Pr(CHD in next ten years) look higher for *older* or *younger* people?

ggplot(fram\_new, aes(x = age, y = chd10\_f)) +   
 geom\_violin(fill = "wheat") +  
 geom\_boxplot(fill = "turquoise", width = 0.3, notch = TRUE)



fram\_new |> group\_by(chd10\_f) |>   
 summarize(n = n(), mean(age), sd(age), median(age)) |>  
 gt() |> fmt\_number(decimals = 2) |> tab\_options(table.font.size = 20)

| chd10\_f | n | mean(age) | sd(age) | median(age) |
| --- | --- | --- | --- | --- |
| chd\_no | 3,594.00 | 48.77 | 8.41 | 48.00 |
| chd | 644.00 | 54.15 | 8.01 | 55.00 |

## So what do we expect in this model?

Pr(CHD in next ten years) looks higher for *older* people?

If we predict log(odds(CHD in next ten years)), we want to ensure that value will be **rising** with increased age.

So, for the mage\_1 model below, what sign do we expect for the slope of age?

mage\_1 <- glm(chd10\_f ~ age, family = binomial,   
 data = fram\_new)

## Results for mage\_1

tidy(mage\_1) |>   
 gt() |> fmt\_number(decimals = 3) |> tab\_options(table.font.size = 20)

| term | estimate | std.error | statistic | p.value |
| --- | --- | --- | --- | --- |
| (Intercept) | -5.558 | 0.284 | -19.585 | 0.000 |
| age | 0.075 | 0.005 | 14.166 | 0.000 |

tidy(mage\_1, exponentiate = TRUE) |>   
 gt() |> fmt\_number(decimals = 3) |> tab\_options(table.font.size = 20)

| term | estimate | std.error | statistic | p.value |
| --- | --- | --- | --- | --- |
| (Intercept) | 0.004 | 0.284 | -19.585 | 0.000 |
| age | 1.077 | 0.005 | 14.166 | 0.000 |

## Stage 2. Split the data into training/test samples.

set.seed(2022432)  
  
fram\_splits <-   
 initial\_split(fram\_new, prop = 3/4, strata = chd10\_f)  
  
fram\_train <- training(fram\_splits)  
fram\_test <- testing(fram\_splits)

### Did the stratification work?

fram\_train |> tabyl(chd10\_f)

chd10\_f n percent  
 chd\_no 2695 0.8480176  
 chd 483 0.1519824

fram\_test |> tabyl(chd10\_f)

chd10\_f n percent  
 chd\_no 899 0.8481132  
 chd 161 0.1518868

## Stage 3. Build a recipe for our model.

fram\_rec <-   
 recipe(chd10\_f ~ age + cigs + educ + hrate +   
 sbp + stroke, data = fram\_new) |>  
 step\_impute\_bag(all\_predictors()) |>  
 step\_dummy(all\_nominal(), -all\_outcomes()) |>  
 step\_normalize(all\_predictors())

1. Specify the roles for the outcome and the predictors.
2. Use bagged trees to impute missing values in predictors.
3. Form dummy variables to represent all categorical variables.
   * Forgetting the -all\_outcomes() wasted a half hour of my life, so learn from my mistake.
4. Normalize (subtract mean and divide by SD) all quantitative predictors.

## Stage 4. Specify engines for our fit(s).

fram\_glm\_model <-   
 logistic\_reg() |>   
 set\_engine("glm")  
  
prior\_dist <- rstanarm::normal(0, 3)  
  
fram\_stan\_model <- logistic\_reg() |>  
 set\_engine("stan",  
 prior\_intercept = prior\_dist,  
 prior = prior\_dist)

## Working with rstanarm

* I recommend How To Use the rstanarm Package at <http://mc-stan.org/rstanarm/articles/rstanarm.html>
* rstanarm models have default prior distributions for their parameters. These are discussed at <http://mc-stan.org/rstanarm/articles/priors.html>

In general, the default priors are *weakly informative* rather than flat. They are designed to help stabilize computation.

## Stage 5. Create a workflow and fit model(s).

fram\_glm\_wf <- workflow() |>  
 add\_model(fram\_glm\_model) |>  
 add\_recipe(fram\_rec)  
  
fram\_stan\_wf <- workflow() |>  
 add\_model(fram\_stan\_model) |>  
 add\_recipe(fram\_rec)

Ready to fit the models?

## Fit the glm and stan models

fit\_A <- fit(fram\_glm\_wf, fram\_train)  
  
set.seed(432)  
fit\_B <- fit(fram\_stan\_wf, fram\_train)

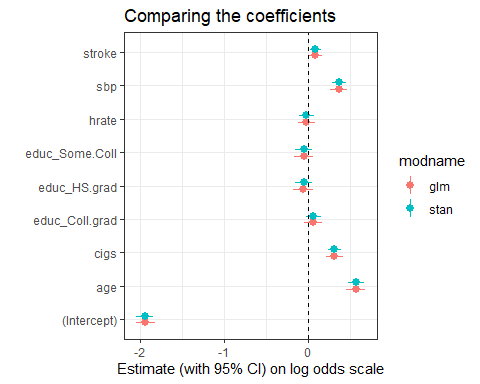
## Produce tidied coefficients (log odds scale)

A\_tidy <- tidy(fit\_A, conf.int = T) |>  
 mutate(modname = "glm")  
  
B\_tidy <- broom.mixed::tidy(fit\_B, conf.int = T) |>  
 mutate(modname = "stan")  
  
coefs\_comp <- bind\_rows(A\_tidy, B\_tidy)

That’s set us up for some plotting.

## Stage 6. Compare fit coefficients.

ggplot(coefs\_comp, aes(x = term, y = estimate, col = modname,  
 ymin = conf.low, ymax = conf.high)) +  
 geom\_point(position = position\_dodge2(width = 0.4)) +  
 geom\_pointrange(position = position\_dodge2(width = 0.4)) +  
 geom\_hline(yintercept = 0, lty = "dashed") +  
 coord\_flip() +  
 labs(x = "",   
 y = "Estimate (with 95% CI) on log odds scale",  
 title = "Comparing the coefficients")



## Can we compare coefficients as odds ratios?

A\_odds <- A\_tidy |>   
 mutate(odds = exp(estimate),  
 odds\_low = exp(conf.low),  
 odds\_high = exp(conf.high)) |>  
 filter(term != "(Intercept)") |>  
 select(modname, term, odds, odds\_low, odds\_high)  
  
head(A\_odds, 2)

# A tibble: 2 × 5  
 modname term odds odds\_low odds\_high  
 <chr> <chr> <dbl> <dbl> <dbl>  
1 glm age 1.76 1.57 1.99  
2 glm cigs 1.37 1.24 1.51

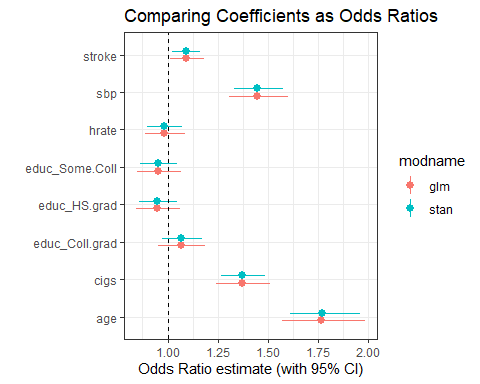
Then repeat to create B\_odds (see next slide)

## Creating B\_odds

B\_odds <- B\_tidy |>   
 mutate(odds = exp(estimate),  
 odds\_low = exp(conf.low),  
 odds\_high = exp(conf.high)) |>  
 filter(term != "(Intercept)") |>  
 select(modname, term, odds, odds\_low, odds\_high)

## Combined Results (Odds Ratios)

odds\_comp <- bind\_rows(A\_odds, B\_odds)  
  
ggplot(odds\_comp, aes(x = term, y = odds, col = modname,  
 ymin = odds\_low, ymax = odds\_high)) +  
 geom\_point(position = position\_dodge2(width = 0.4)) +  
 geom\_pointrange(position = position\_dodge2(width = 0.4)) +  
 geom\_hline(yintercept = 1, lty = "dashed") +  
 coord\_flip() +  
 labs(x = "", y = "Odds Ratio estimate (with 95% CI)",  
 title = "Comparing Coefficients as Odds Ratios")



## Stage 7. Assess training sample performance.

1. We’ll make predictions for the training sample using each model, and use them to find the C statistic and plot the ROC curve.
2. We’ll show some other summaries of performance in the training sample.

## Make Predictions with fit\_A

We’ll start by using the glm model fit\_A to make predictions.

glm\_probs <-   
 predict(fit\_A, fram\_train, type = "prob") |>  
 bind\_cols(fram\_train |> select(chd10\_f))  
  
head(glm\_probs, 4)

# A tibble: 4 × 3  
 .pred\_chd\_no .pred\_chd chd10\_f  
 <dbl> <dbl> <fct>   
1 0.759 0.241 chd   
2 0.917 0.0826 chd   
3 0.911 0.0892 chd   
4 0.889 0.111 chd

## Obtain C statistic for fit\_A

Next, we’ll use roc\_auc from yardstick. This assumes that the first level of chd10\_f is the thing we’re trying to predict. Is that true in our case?

fram\_train |> tabyl(chd10\_f)

chd10\_f n percent  
 chd\_no 2695 0.8480176  
 chd 483 0.1519824

## Do we want to predict the first level of chd\_f?

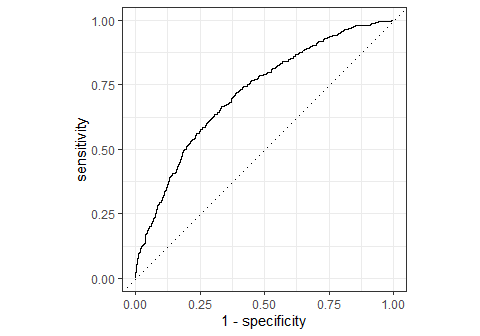
No. We want to predict the second level: chd. So we need to switch the event\_level to “second”, like this.

glm\_probs |> roc\_auc(chd10\_f, .pred\_chd,   
 event\_level = "second") |>  
 gt() |> fmt\_number(decimals = 5) |> tab\_options(table.font.size = 20)

| .metric | .estimator | .estimate |
| --- | --- | --- |
| roc\_auc | binary | 0.71860 |

## Can we plot the ROC curve for fit\_A?

glm\_roc <- glm\_probs |>  
 roc\_curve(chd10\_f, .pred\_chd, event\_level = "second")  
autoplot(glm\_roc)



* We saw on the prior slide that our C statistic for the glm fit is 0.719.

## Make Predictions with fit\_B

We’ll use the stan model fit\_B to make predictions.

stan\_probs <-   
 predict(fit\_B, fram\_train, type = "prob") |>  
 bind\_cols(fram\_train |> select(chd10\_f))

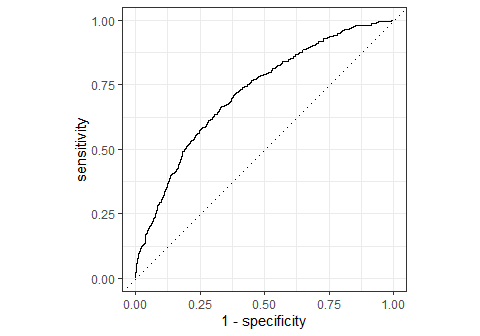
Now, we’ll obtain the C statistic for fit\_B

stan\_probs |>   
 roc\_auc(chd10\_f, .pred\_chd,   
 event\_level = "second") |>  
 gt() |> fmt\_number(decimals = 5) |> tab\_options(table.font.size = 20)

| .metric | .estimator | .estimate |
| --- | --- | --- |
| roc\_auc | binary | 0.71861 |

## Plotting the ROC curve for fit\_B?

stan\_roc <- stan\_probs |>  
 roc\_curve(chd10\_f, .pred\_chd, event\_level = "second")  
autoplot(stan\_roc)



* Our C statistic for the stan fit is also 0.719.

## Other available summaries from yardstick

For a logistic regression where we’re willing to specify a decision rule, we can consider:

* Conf\_mat which produces a confusion matrix if we specify a decision rule.
  + There is a way to tidy a confusion matrix, summarize it with summary() and autoplot it with either a mosaic or a heatmap.

## Other yardstick summaries

* accuracy = proportion of the data that are predicted correctly
* kap is very similar to accuracy but is normalized by the accuracy that would be expected by chance alone and is most useful when one or more classes dominate the distribution - attributed to Cohen (1960)
* sens = sensitivity and spec specificity
* ppv positive predictive value and npv negative predictive value

## Establishing a decision rule for the glm fit

Let’s use .pred\_chd > 0.2 for now to indicate a prediction of chd.

glm\_probs <-   
 predict(fit\_A, fram\_train, type = "prob") |>  
 bind\_cols(fram\_train |> select(chd10\_f)) |>  
 mutate(chd10\_pre =   
 ifelse(.pred\_chd > 0.2, "chd", "chd\_no")) |>  
 mutate(chd10\_pre = fct\_relevel(factor(chd10\_pre),  
 "chd\_no"))  
  
glm\_probs |> tabyl(chd10\_pre, chd10\_f)

chd10\_pre chd\_no chd  
 chd\_no 2144 234  
 chd 551 249

## Why didn’t I use .pred\_chd > 0.5?

glm\_probs5 <-   
 predict(fit\_A, fram\_train, type = "prob") |>  
 bind\_cols(fram\_train |> select(chd10\_f)) |>  
 mutate(chd10\_pre =   
 ifelse(.pred\_chd > 0.5, "chd", "chd\_no")) |>  
 mutate(chd10\_pre = fct\_relevel(factor(chd10\_pre),  
 "chd\_no"))  
  
glm\_probs5 |> tabyl(chd10\_pre)

chd10\_pre n percent  
 chd\_no 3138 0.98741347  
 chd 40 0.01258653

## What can we run now?

conf\_mat(glm\_probs, truth = chd10\_f, estimate = chd10\_pre)

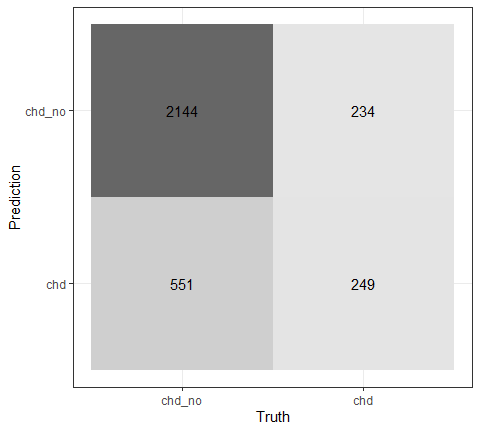
Truth  
Prediction chd\_no chd  
 chd\_no 2144 234  
 chd 551 249

metrics(glm\_probs, truth = chd10\_f, estimate = chd10\_pre)

# A tibble: 2 × 3  
 .metric .estimator .estimate  
 <chr> <chr> <dbl>  
1 accuracy binary 0.753  
2 kap binary 0.245

## Plot confusion matrix for glm fit?

conf\_mat(glm\_probs,   
 truth = chd10\_f, estimate = chd10\_pre) |>   
 autoplot(type = "heatmap")



## More Confusion Matrix Summaries?

Other available metrics include:

* sensitivity, specificity, positive predictive value, negative predictive value, and the statistics below.

conf\_mat(glm\_probs, truth = chd10\_f, estimate = chd10\_pre) |>   
 summary() |> slice(7:13)

# A tibble: 7 × 3  
 .metric .estimator .estimate  
 <chr> <chr> <dbl>  
1 mcc binary 0.257  
2 j\_index binary 0.311  
3 bal\_accuracy binary 0.656  
4 detection\_prevalence binary 0.748  
5 precision binary 0.902  
6 recall binary 0.796  
7 f\_meas binary 0.845

## Establishing a decision rule for the stan fit

Let’s also use .pred\_chd > 0.2 to indicate a prediction of chd.

stan\_probs <-   
 predict(fit\_B, fram\_train, type = "prob") |>  
 bind\_cols(fram\_train |> select(chd10\_f)) |>  
 mutate(chd10\_pre =   
 ifelse(.pred\_chd > 0.2, "chd", "chd\_no")) |>  
 mutate(chd10\_pre = fct\_relevel(factor(chd10\_pre),  
 "chd\_no"))

## Confusion Matrix and Basic Metrics

conf\_mat(stan\_probs, truth = chd10\_f, estimate = chd10\_pre)

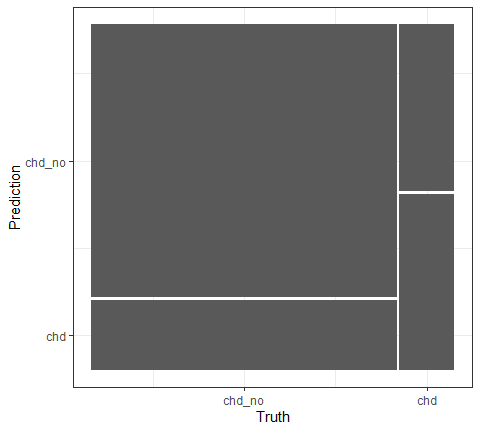
Truth  
Prediction chd\_no chd  
 chd\_no 2150 235  
 chd 545 248

metrics(stan\_probs, truth = chd10\_f, estimate = chd10\_pre)

# A tibble: 2 × 3  
 .metric .estimator .estimate  
 <chr> <chr> <dbl>  
1 accuracy binary 0.755  
2 kap binary 0.246

## Plot a confusion matrix for stan fit

conf\_mat(stan\_probs,   
 truth = chd10\_f, estimate = chd10\_pre) |>   
 autoplot(type = "mosaic")



## More Confusion Matrix Summaries?

conf\_mat(stan\_probs,   
 truth = chd10\_f, estimate = chd10\_pre) |>   
 summary()

# A tibble: 13 × 3  
 .metric .estimator .estimate  
 <chr> <chr> <dbl>  
 1 accuracy binary 0.755  
 2 kap binary 0.246  
 3 sens binary 0.798  
 4 spec binary 0.513  
 5 ppv binary 0.901  
 6 npv binary 0.313  
 7 mcc binary 0.258  
 8 j\_index binary 0.311  
 9 bal\_accuracy binary 0.656  
10 detection\_prevalence binary 0.750  
11 precision binary 0.901  
12 recall binary 0.798  
13 f\_meas binary 0.846

## Stage 8. Assess test sample performance.

glm\_test <-   
 predict(fit\_A, fram\_test, type = "prob") |>  
 bind\_cols(fram\_test |> select(chd10\_f))  
  
stan\_test <-   
 predict(fit\_B, fram\_test, type = "prob") |>  
 bind\_cols(fram\_test |> select(chd10\_f))

## Test Sample C statistic comparison?

glm\_test |> roc\_auc(chd10\_f, .pred\_chd,   
 event\_level = "second") |>  
 gt() |> fmt\_number(decimals = 4) |> tab\_options(table.font.size = 20)

| .metric | .estimator | .estimate |
| --- | --- | --- |
| roc\_auc | binary | 0.7231 |

stan\_test |> roc\_auc(chd10\_f, .pred\_chd,   
 event\_level = "second") |>  
 gt() |> fmt\_number(decimals = 4) |> tab\_options(table.font.size = 20)

| .metric | .estimator | .estimate |
| --- | --- | --- |
| roc\_auc | binary | 0.7229 |

## What’s Next?

A little bit of K-Means Clustering and Principal Components Analysis