Biostat 203B Homework 5

Due Mar 20 @ 11:59PM

Wenqiang Ge UID:106371961

Table of contents

0.1 Predicting ICU duration

Using the ICU cohort `mimiciv_icu_cohort.rds` you built in Homework 4, develop at least three machine learning approaches (logistic regression with enet regularization, random forest, boosting, SVM, MLP, etc) plus a model stacking approach for predicting whether a patient's ICU stay will be longer than 2 days. You should use the `los_long` variable as the outcome. You algorithms can use patient demographic information (gender, age at ICU `intime`, marital status, race), ICU admission information (first care unit), the last lab measurements before the ICU stay, and first vital measurements during ICU stay as features. You are welcome to use any feature engineering techniques you think are appropriate; but make sure to not use features that are not available at an ICU stay's `intime`. For instance, `last_careunit` cannot be used in your algorithms.

1. Data preprocessing and feature engineering.

```r
# Load libraries
library(GGally)
```

```
Loading required package: ggplot2
```

```
Registered S3 method overwritten by 'GGally':
  method from
  +.gg   ggplot2
```

```r
library(gtsummary)
library(ranger)
library(tidyverse)
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.5
v forcats   1.0.0     v stringr   1.5.1
```

```
v lubridate 1.9.4     v tibble    3.2.1
v purrr     1.0.4     v tidyr     1.3.1
```

```
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to beco
```

```r
library(tidymodels)
```

```
-- Attaching packages -------------------------------------- tidymodels 1.3.0 --
v broom        1.0.7     v rsample      1.2.1
v dials        1.4.0     v tune         1.3.0
v infer        1.0.7     v workflows    1.2.0
v modeldata    1.4.0     v workflowsets 1.1.0
v parsnip      1.3.0     v yardstick    1.3.2
v recipes      1.1.1
-- Conflicts ----------------------------------------- tidymodels_conflicts() --
x scales::discard() masks purrr::discard()
x dplyr::filter()   masks stats::filter()
x recipes::fixed()  masks stringr::fixed()
x dplyr::lag()      masks stats::lag()
x yardstick::spec() masks readr::spec()
x recipes::step()   masks stats::step()
```

```r
library(xgboost)
```

```

Attaching package: 'xgboost'

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

    slice
```

```r
library(doParallel)
```

```
Loading required package: foreach

Attaching package: 'foreach'

The following objects are masked from 'package:purrr':

    accumulate, when

Loading required package: iterators
Loading required package: parallel
```

```r
library(stacks)
library(keras)
```

```
Attaching package: 'keras'

The following object is masked from 'package:yardstick':

    get_weights
```

```
library(dplyr)
library(e1071)
```

```
Attaching package: 'e1071'

The following object is masked from 'package:tune':

    tune

The following object is masked from 'package:rsample':

    permutations

The following object is masked from 'package:parsnip':

    tune
```

```
library(rsample)
library(glmnet)
```

```
Loading required package: Matrix

Attaching package: 'Matrix'

The following objects are masked from 'package:tidyr':

    expand, pack, unpack

Loaded glmnet 4.1-8
```

```
library(vip)
```

```
Attaching package: 'vip'

The following object is masked from 'package:utils':

    vi
```

```r
#loading data
mimic_icu_cohort <- readRDS("../hw4/mimiciv_shiny/mimic_icu_cohort.rds")
lab <- c("creatinine", "potassium", "sodium", "chloride", "bicarbonate",
              "hematocrit", "glucose", "wbc")

vital <- c("heart_rate", "non-invasive_blood_pressure_systolic",
                "non-invasive_blood_pressure_diastolic",
                "temperature_fahrenheit", "respiratory_rate")

#Select the variables
mimic_icu_cohort <- mimic_icu_cohort |>
  select(c("subject_id", "hadm_id", "stay_id",
           "gender","first_careunit",
            "insurance","marital_status",
           "race","age_intime",
            all_of(lab), all_of(vital),
           "los_long")) |>
  arrange("subject_id", "hadm_id", "stay_id")
```

```r
#Convert gender to factor variables
mimic_icu_cohort$gender <- ifelse(mimic_icu_cohort$gender == "M",0,1)
mimic_icu_cohort$gender <- factor(mimic_icu_cohort$gender)

# For character or factor variables, replace NA with 0, and label other unique values
mimic_icu_cohort[] <- lapply(mimic_icu_cohort, function(x) {
  # Only modify character or factor columns, but skip 'gender'
  if (is.factor(x) || is.character(x)) {
    # Skip the 'gender' column
    if (!"gender" %in% names(mimic_icu_cohort)[
      which(sapply(mimic_icu_cohort, identical, x))
      ]) {
      # Convert to character type if it is not gender
      x <- as.character(x)

      # Replace NA with "0"
      x[is.na(x)] <- "0"

      # Convert back to factor and then numeric
      x <- factor(x)
      return(as.numeric(x))  # Convert factor levels to numeric
    }
  }
  return(x)  # Leave other columns unchanged
})
```

```r
# Check for outliers in numeric variables
mimic_icu_cohort[] <- lapply(mimic_icu_cohort, function(x) {
  if (is.numeric(x)) {
    # Calculate the first (Q1) and third (Q3) quartiles, and the IQR (Interquartile Range)
    Q1 <- quantile(x, 0.25, na.rm = TRUE)
    Q3 <- quantile(x, 0.75, na.rm = TRUE)
    IQR_value <- Q3 - Q1

    # Define the lower and upper limits for outliers using the IQR method
    lower_limit <- Q1 - 1.5 * IQR_value
    upper_limit <- Q3 + 1.5 * IQR_value

    # Replace values outside the defined limits with NA (outliers)
    x[x < lower_limit | x > upper_limit] <- NA
  }
  return(x)
})
```

```r
#Replace NA values with the mean of the column
mimic_icu_cohort[] <- lapply(mimic_icu_cohort, function(x) {
  if (is.numeric(x)) {
    x[is.na(x)] <- mean(x, na.rm = TRUE)  # Replace NA with the mean value
  }
  return(x)
})
```

```r
# Convert los_long to a factor
mimic_icu_cohort$los_long <- factor(mimic_icu_cohort$los_long, levels = c(FALSE, TRUE))

# Remove rows where los_long is NA
mimic_icu_cohort <- mimic_icu_cohort %>%
  drop_na(los_long)

#Convert other variables to factors
mimic_icu_cohort$marital_status <- factor(mimic_icu_cohort$marital_status)
mimic_icu_cohort$race <- factor(mimic_icu_cohort$race)
mimic_icu_cohort$first_careunit <- factor(mimic_icu_cohort$first_careunit)
mimic_icu_cohort$insurance <- factor(mimic_icu_cohort$insurance)

head(mimic_icu_cohort)
```

```
# A tibble: 6 x 23
  subject_id  hadm_id  stay_id gender first_careunit insurance marital_status
       <dbl>    <dbl>    <dbl> <fct>  <fct>          <fct>     <fct>
1   10000032 29079034 39553978 1      2              2         5
2   10000690 25860671 37081114 1      2              3         5
```

```
3   10000980 26913865 39765666 1      2               3         3
4   10001217 24597018 37067082 1      5               6         3
5   10001217 27703517 34592300 1      5               6         3
6   10001725 25563031 31205490 1      3               6         3
# i 16 more variables: race <fct>, age_intime <dbl>, creatinine <dbl>,
#   potassium <dbl>, sodium <dbl>, chloride <dbl>, bicarbonate <dbl>,
#   hematocrit <dbl>, glucose <dbl>, wbc <dbl>, heart_rate <dbl>,
#   `non-invasive_blood_pressure_systolic` <dbl>,
#   `non-invasive_blood_pressure_diastolic` <dbl>,
#   temperature_fahrenheit <dbl>, respiratory_rate <dbl>, los_long <fct>
```

2. Partition data into 50% training set and 50% test set. Stratify partitioning according to `los_long`. For grading purpose, sort the data by `subject_id`, `hadm_id`, and `stay_id` and use the seed `203` for the initial data split. Below is the sample code.

0.1.0.1 Initial split into test and non-test sets

```
set.seed(203)

# sort
mimic_icu_cohort <- mimic_icu_cohort |>
  arrange(subject_id, hadm_id, stay_id)

data_split <- initial_split(
  mimic_icu_cohort,
  # stratify by los_long
  strata = "los_long",
  prop = 0.5
  )

mimic_train <- training(data_split) |>
  select(-subject_id, -hadm_id, -stay_id)
dim(mimic_train )
```

```
[1] 47221    20
```

```
mimic_test <- testing(data_split) |>
  select(-subject_id, -hadm_id, -stay_id)
dim(mimic_test)
```

```
[1] 47223    20
```

0.1.0.2 Recipe

```
data_recipe <-
  recipe(
    los_long ~ .,
    data = mimic_train
    ) |>
  step_dummy(all_nominal_predictors()) |>
  step_zv(all_numeric_predictors()) |>
  step_normalize(all_numeric_predictors())
data_recipe
```

```
-- Recipe ----------------------------------------------------------------------


-- Inputs

Number of variables by role

outcome:    1
predictor: 19


-- Operations

* Dummy variables from: all_nominal_predictors()

* Zero variance filter on: all_numeric_predictors()

* Centering and scaling for: all_numeric_predictors()
```

```
folds <- vfold_cv(mimic_train, v = 5)
```

0.1.1 logistic regression with enet regularization

0.1.1.1 Model & Workflow

```
# set up the logistic regression model
logit_model <- logistic_reg(
  penalty = tune(),
  mixture = tune()) |>
  set_engine("glmnet",
             standardize = TRUE)

logit_wf <- workflow() |>
  add_recipe(data_recipe) |>
  add_model(logit_model) |>
  print()
```

```
== Workflow ====================================================================
Preprocessor: Recipe
Model: logistic_reg()
```

```
-- Preprocessor ----------------------------------------------------------------
3 Recipe Steps

* step_dummy()
* step_zv()
* step_normalize()

-- Model -----------------------------------------------------------------------
Logistic Regression Model Specification (classification)

Main Arguments:
  penalty = tune()
  mixture = tune()

Engine-Specific Arguments:
  standardize = TRUE

Computational engine: glmnet
```

0.1.1.2 Tuning grid

```r
logit_grid <- grid_regular(
  penalty(range = c(-6, 4)),
  mixture(),
  levels = c(40, 4)
)
```

0.1.1.3 Cross-validation

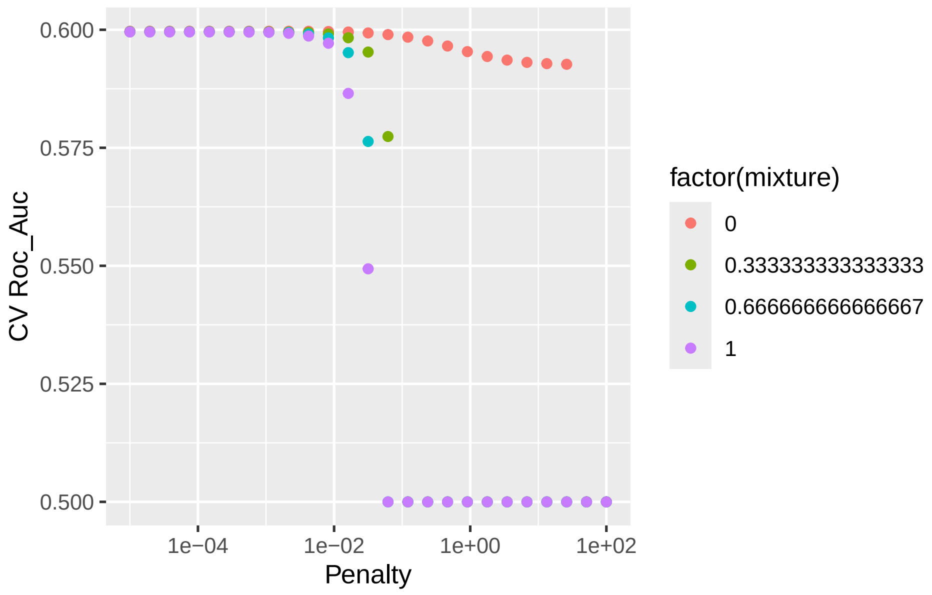
```r
set.seed(203)

if (file.exists("logit_res.rds")) {
  logit_res <- read_rds("logit_res.rds")
} else {
  logit_res <-
    tune_grid(
      object = logit_wf,
      resamples = folds,
      grid = logit_grid,
      metrics = metric_set(roc_auc, accuracy),
      control = control_stack_grid()
    )
  write_rds(logit_res, "logit_res.rds")
}
logit_res
```

```
# Tuning results
# 5-fold cross-validation
# A tibble: 5 x 5
  splits               id    .metrics           .notes           .predictions
  <list>               <chr> <list>             <list>           <list>
1 <split [37776/9445]> Fold1 <tibble [200 x 6]> <tibble [0 x 3]> <tibble>
2 <split [37777/9444]> Fold2 <tibble [200 x 6]> <tibble [0 x 3]> <tibble>
3 <split [37777/9444]> Fold3 <tibble [200 x 6]> <tibble [0 x 3]> <tibble>
4 <split [37777/9444]> Fold4 <tibble [200 x 6]> <tibble [0 x 3]> <tibble>
5 <split [37777/9444]> Fold5 <tibble [200 x 6]> <tibble [0 x 3]> <tibble>
```

```
# visualize the CV results
logit_res |>
  collect_metrics() |>
  print(width = Inf) |>
  filter(.metric == "roc_auc") |>
  ggplot(mapping = aes(x = penalty, y = mean,
                       color = factor(mixture))) +
  geom_point() +
  labs(x = "Penalty", y = "CV Roc_Auc") +
  scale_x_log10()
```

```
# A tibble: 200 x 8
     penalty mixture .metric  .estimator  mean     n std_err
       <dbl>   <dbl> <chr>    <chr>      <dbl> <int>   <dbl>
 1 0.00001         0 accuracy binary     0.572     5 0.00196
 2 0.00001         0 roc_auc  binary     0.600     5 0.00233
 3 0.0000196       0 accuracy binary     0.572     5 0.00196
 4 0.0000196       0 roc_auc  binary     0.600     5 0.00233
 5 0.0000383       0 accuracy binary     0.572     5 0.00196
 6 0.0000383       0 roc_auc  binary     0.600     5 0.00233
 7 0.0000750       0 accuracy binary     0.572     5 0.00196
 8 0.0000750       0 roc_auc  binary     0.600     5 0.00233
 9 0.000147        0 accuracy binary     0.572     5 0.00196
10 0.000147        0 roc_auc  binary     0.600     5 0.00233
   .config
   <chr>
 1 Preprocessor1_Model001
 2 Preprocessor1_Model001
 3 Preprocessor1_Model002
 4 Preprocessor1_Model002
 5 Preprocessor1_Model003
 6 Preprocessor1_Model003
 7 Preprocessor1_Model004
 8 Preprocessor1_Model004
 9 Preprocessor1_Model005
```

```
10 Preprocessor1_Model005
# i 190 more rows
```



0.1.1.4 Finalize the model

```r
#show best models
logit_res |>
  show_best(metric = "roc_auc")
```

```
# A tibble: 5 x 8
    penalty mixture .metric .estimator  mean     n std_err .config
      <dbl>   <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
1 0.00001         0 roc_auc binary     0.600     5 0.00233 Preprocessor1_Model0~
2 0.0000196       0 roc_auc binary     0.600     5 0.00233 Preprocessor1_Model0~
3 0.0000383       0 roc_auc binary     0.600     5 0.00233 Preprocessor1_Model0~
4 0.0000750       0 roc_auc binary     0.600     5 0.00233 Preprocessor1_Model0~
5 0.000147        0 roc_auc binary     0.600     5 0.00233 Preprocessor1_Model0~
```

```r
# select the best model
best_logit <- logit_res |>
  select_best(metric = "roc_auc")
best_logit
```

```
# A tibble: 1 x 3
  penalty mixture .config
    <dbl>   <dbl> <chr>
1 0.00001       0 Preprocessor1_Model001
```

```
#Testing
final_logit <- logit_wf |>
  finalize_workflow(best_logit)
final_logit
```

```
== Workflow ====================================================================
Preprocessor: Recipe
Model: logistic_reg()

-- Preprocessor ----------------------------------------------------------------
3 Recipe Steps

* step_dummy()
* step_zv()
* step_normalize()

-- Model -----------------------------------------------------------------------
Logistic Regression Model Specification (classification)

Main Arguments:
  penalty = 1e-05
  mixture = 0

Engine-Specific Arguments:
  standardize = TRUE

Computational engine: glmnet
```

```
# Fit the whole training set, then predict the test cases
set.seed(203)
final_logit_fit <-
  final_logit |>
  last_fit(data_split)
final_logit_fit
```

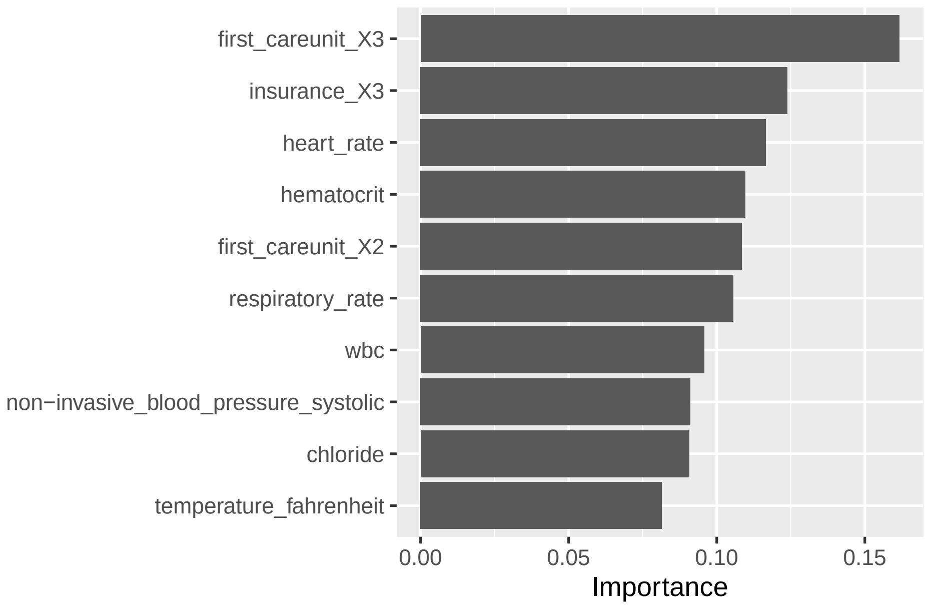
```
# Resampling results
# Manual resampling
# A tibble: 1 x 6
  splits                id              .metrics .notes   .predictions .workflow
  <list>                <chr>           <list>   <list>   <list>       <list>
1 <split [47221/47223]> train/test sp~ <tibble> <tibble> <tibble>     <workflow>
```

```
final_logit_fit |>
  collect_metrics()
```

```
# A tibble: 3 x 4
  .metric     .estimator .estimate .config
  <chr>       <chr>          <dbl> <chr>
1 accuracy    binary         0.570 Preprocessor1_Model1
2 roc_auc     binary         0.595 Preprocessor1_Model1
3 brier_class binary         0.243 Preprocessor1_Model1
```

0.1.1.5 Visualize the final model

```r
final_logic_tree <- extract_workflow(final_logit_fit)

final_logic_tree |>
  extract_fit_parsnip() |>
  vip()
```



```r
rm(final_logic_tree,
   final_logit,logit_res,logit_grid)
```

0.1.2 Boosting

0.1.2.1 Model & Workflow

```r
#Model
bt_mod <-
  boost_tree(
    mode = "classification",
    trees = 800,
    tree_depth = tune(),
    learn_rate = tune()
  ) %>%
  set_engine("xgboost")
bt_mod
```

```
Boosted Tree Model Specification (classification)

Main Arguments:
  trees = 800
  tree_depth = tune()
  learn_rate = tune()

Computational engine: xgboost
```

```
#Workflow
bt_wf <- workflow() |>
  add_recipe(data_recipe) |>
  add_model(bt_mod)
bt_wf
```

```
== Workflow ====================================================================
Preprocessor: Recipe
Model: boost_tree()

-- Preprocessor ----------------------------------------------------------------
3 Recipe Steps

* step_dummy()
* step_zv()
* step_normalize()

-- Model -----------------------------------------------------------------------
Boosted Tree Model Specification (classification)

Main Arguments:
  trees = 800
  tree_depth = tune()
  learn_rate = tune()

Computational engine: xgboost
```

0.1.2.2 Tuning grid

```
#Tuning
param_grid <- grid_regular(
  tree_depth(range = c(1L, 3L)),
  learn_rate(range = c(-5, 2), trans = log10_trans()),
  levels = c(3, 1)
  )
param_grid
```

```
# A tibble: 3 x 2
  tree_depth learn_rate
       <int>      <dbl>
1          1    0.00001
2          2    0.00001
3          3    0.00001
```

```
library(doParallel)
registerDoParallel(cores = 4)  # Set the number of cores you want to use
```

0.1.2.3 Cross-validation

```
bt_fit <- bt_wf %>%
  tune_grid(
    resamples = folds,
    grid = param_grid,
    metrics = metric_set(yardstick::roc_auc,
                         yardstick::accuracy)
    )
```

```
Warning: ! tune detected a parallel backend registered with foreach but no backend
  registered with future.
i Support for parallel processing with foreach was soft-deprecated in tune
  1.2.1.
i See ?parallelism (`?tune::parallelism()`) to learn more.
```

```
bt_fit
```

```
# Tuning results
# 5-fold cross-validation
# A tibble: 5 x 4
  splits               id    .metrics         .notes
  <list>               <chr> <list>           <list>
1 <split [37776/9445]> Fold1 <tibble [6 x 6]> <tibble [0 x 3]>
2 <split [37777/9444]> Fold2 <tibble [6 x 6]> <tibble [0 x 3]>
3 <split [37777/9444]> Fold3 <tibble [6 x 6]> <tibble [0 x 3]>
4 <split [37777/9444]> Fold4 <tibble [6 x 6]> <tibble [0 x 3]>
5 <split [37777/9444]> Fold5 <tibble [6 x 6]> <tibble [0 x 3]>
```

```
bt_fit <- bt_wf |>
  tune_grid(
    resamples = folds,
    grid = param_grid,
    metrics = metric_set(roc_auc, accuracy),
    control = control_stack_grid()
  )
```

```
Warning: ! tune detected a parallel backend registered with foreach but no backend
  registered with future.
i Support for parallel processing with foreach was soft-deprecated in tune
  1.2.1.
i See ?parallelism (`?tune::parallelism()`) to learn more.
```

```
i The workflow being saved contains a recipe, which is 6.19 Mb in i memory. If
this was not intentional, please set the control setting i `save_workflow =
FALSE`.
```

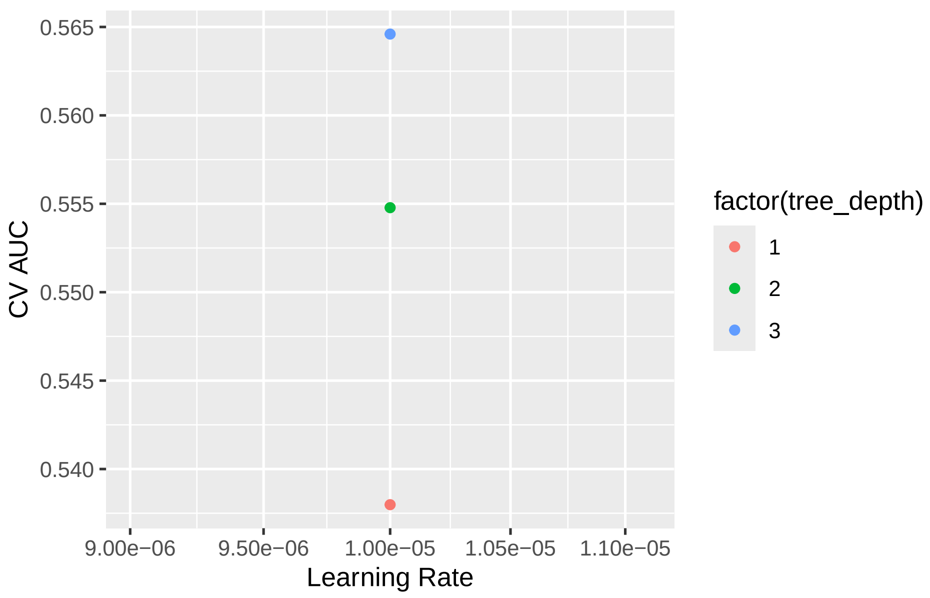
```
bt_fit |> write_rds("./boost_fit.rds")
```

```
stopImplicitCluster()
registerDoSEQ()
```

```
bt_fit <- readRDS("./boost_fit.rds")

bt_fit |>
  collect_metrics() |>
  print(width = Inf) |>
  filter(.metric == "roc_auc") |>
  ggplot(mapping =
           aes(x = learn_rate,
               y = mean,
               color = factor(tree_depth))) +
  geom_point() +
  labs(x = "Learning Rate", y = "CV AUC") +
  scale_x_log10()
```

```
# A tibble: 6 x 8
  tree_depth learn_rate .metric  .estimator  mean     n std_err
       <int>      <dbl> <chr>    <chr>      <dbl> <int>   <dbl>
1          1    0.00001 accuracy binary     0.538     5 0.00199
2          1    0.00001 roc_auc  binary     0.538     5 0.00207
3          2    0.00001 accuracy binary     0.545     5 0.00163
4          2    0.00001 roc_auc  binary     0.555     5 0.00457
5          3    0.00001 accuracy binary     0.551     5 0.00249
6          3    0.00001 roc_auc  binary     0.565     5 0.00472
  .config
  <chr>
1 Preprocessor1_Model1
2 Preprocessor1_Model1
3 Preprocessor1_Model2
```

```
4 Preprocessor1_Model2
5 Preprocessor1_Model3
6 Preprocessor1_Model3
```



0.1.2.4 Finalize the model

```
bt_fit |>
  show_best(metric = "roc_auc")
```

```
# A tibble: 3 x 8
  tree_depth learn_rate .metric .estimator  mean     n std_err .config
       <int>      <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
1          3    0.00001 roc_auc binary     0.565     5 0.00472 Preprocessor1_Mo~
2          2    0.00001 roc_auc binary     0.555     5 0.00457 Preprocessor1_Mo~
3          1    0.00001 roc_auc binary     0.538     5 0.00207 Preprocessor1_Mo~
```

```
best_boost <- bt_fit |>
  select_best(metric = "roc_auc")
best_boost
```

```
# A tibble: 1 x 3
  tree_depth learn_rate .config
       <int>      <dbl> <chr>
1          3    0.00001 Preprocessor1_Model3
```

```
final_boost_wf <- bt_wf |>
  finalize_workflow(best_boost)
final_boost_wf
```

```
== Workflow ====================================================================
Preprocessor: Recipe
Model: boost_tree()

-- Preprocessor ----------------------------------------------------------------
3 Recipe Steps

* step_dummy()
* step_zv()
* step_normalize()

-- Model -----------------------------------------------------------------------
Boosted Tree Model Specification (classification)

Main Arguments:
  trees = 800
  tree_depth = 3
  learn_rate = 1e-05

Computational engine: xgboost
```

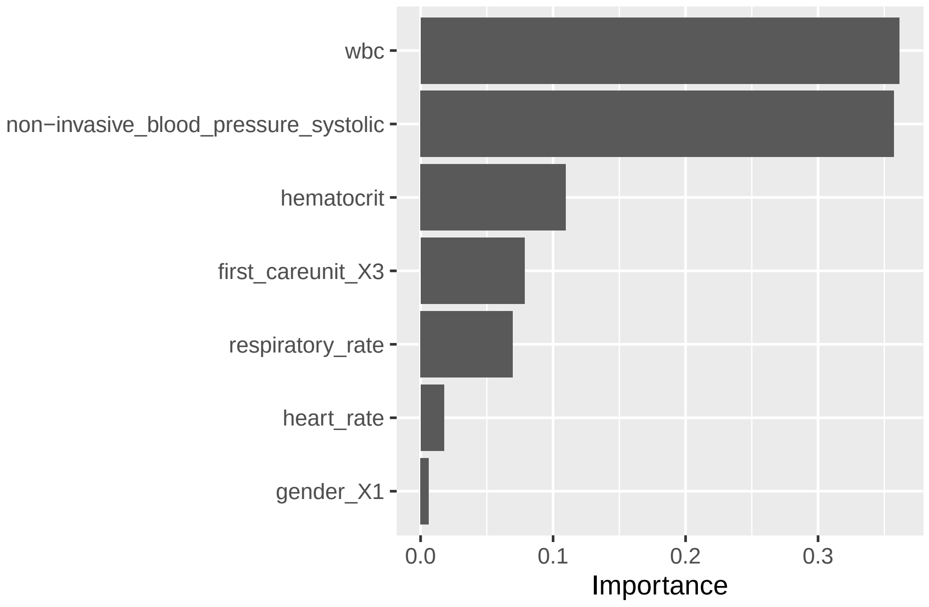
```r
set.seed(203)
final_boost_fit <-
  final_boost_wf |>
  last_fit(data_split)
```

```r
final_boost_fit |>
  collect_metrics()
```

```
# A tibble: 3 x 4
  .metric     .estimator .estimate .config
  <chr>       <chr>          <dbl> <chr>
1 accuracy    binary         0.545 Preprocessor1_Model1
2 roc_auc     binary         0.564 Preprocessor1_Model1
3 brier_class binary         0.250 Preprocessor1_Model1
```

0.1.2.5 Visualize the final model

```r
final_boost_tree <- extract_workflow(final_boost_fit)

final_boost_tree |>
  extract_fit_parsnip() |>
  vip()
```



```
rm(final_boost_tree,final_boost_fit,
   final_boost_wf,param_grid)
```

0.1.3 Random Forest

0.1.3.1 Model & Workflow

```
rf_mod <-
  rand_forest(
    mode = "classification",
    mtry = tune(),
    trees = tune()
  ) |>
  set_engine("ranger", importance = "impurity")
rf_mod
```

```
Random Forest Model Specification (classification)

Main Arguments:
  mtry = tune()
  trees = tune()

Engine-Specific Arguments:
  importance = impurity

Computational engine: ranger
```

```
rf_wf <- workflow() |>
  add_recipe(data_recipe) |>
  add_model(rf_mod)
rf_wf
```

```
== Workflow ================================================================
Preprocessor: Recipe
Model: rand_forest()

-- Preprocessor ------------------------------------------------------------
3 Recipe Steps

* step_dummy()
* step_zv()
* step_normalize()

-- Model -------------------------------------------------------------------
Random Forest Model Specification (classification)

Main Arguments:
  mtry = tune()
  trees = tune()

Engine-Specific Arguments:
  importance = impurity

Computational engine: ranger
```

0.1.3.2 Tuning grid

```
rf_grid <- grid_regular(
  trees(range = c(300L, 700L)),
  mtry(range = c(3L, 7L)),
  levels = c(2, 4)
  )
rf_grid
```

```
# A tibble: 8 x 2
  trees  mtry
  <int> <int>
1   300     3
2   700     3
3   300     4
4   700     4
5   300     5
6   700     5
```

```
7   300     7
8   700     7
```

0.1.3.3 Cross-validation

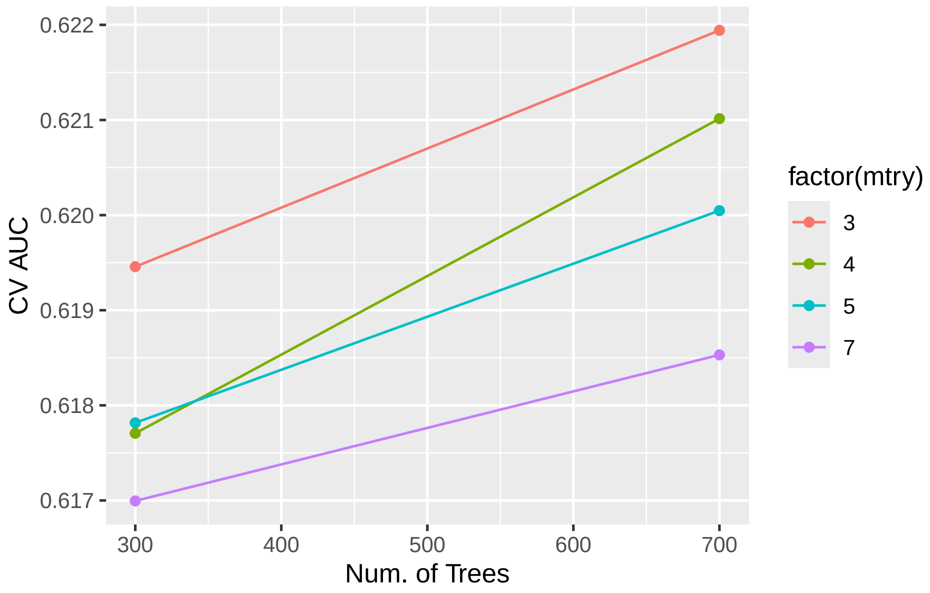
```r
set.seed(203)

folds <- vfold_cv(mimic_train, v = 3)
```

```r
if (file.exists("rf_res.rds")) {
  rf_res <- read_rds("rf_res.rds")
} else {
  rf_res <- rf_wf |>
    tune_grid(
    resamples = folds,
    grid = rf_grid,
    metrics = metric_set(roc_auc, accuracy),
    control = control_grid(verbose = TRUE)
    )
  write_rds(rf_res, "rf_res.rds")
}
rf_res
```

```
# Tuning results
# 3-fold cross-validation
# A tibble: 3 x 4
  splits                id    .metrics          .notes
  <list>                <chr> <list>            <list>
1 <split [31480/15741]> Fold1 <tibble [16 x 6]> <tibble [0 x 3]>
2 <split [31481/15740]> Fold2 <tibble [16 x 6]> <tibble [0 x 3]>
3 <split [31481/15740]> Fold3 <tibble [16 x 6]> <tibble [0 x 3]>
```

```r
rf_res |>
  collect_metrics() |>
  print(width = Inf) |>
  filter(.metric == "roc_auc") |>
  ggplot(mapping = aes(x = trees, y = mean, color = factor(mtry))) +
  geom_point() +
  geom_line() +
  labs(x = "Num. of Trees", y = "CV AUC")
```

```
# A tibble: 16 x 8
    mtry trees .metric  .estimator  mean     n std_err .config
   <int> <int> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
 1     3   300 accuracy binary     0.583     3 0.00193 Preprocessor1_Model1
 2     3   300 roc_auc  binary     0.619     3 0.00181 Preprocessor1_Model1
 3     3   700 accuracy binary     0.586     3 0.00131 Preprocessor1_Model2
 4     3   700 roc_auc  binary     0.622     3 0.00168 Preprocessor1_Model2
 5     4   300 accuracy binary     0.581     3 0.00155 Preprocessor1_Model3
 6     4   300 roc_auc  binary     0.618     3 0.00184 Preprocessor1_Model3
 7     4   700 accuracy binary     0.585     3 0.00219 Preprocessor1_Model4
 8     4   700 roc_auc  binary     0.621     3 0.00189 Preprocessor1_Model4
 9     5   300 accuracy binary     0.584     3 0.00188 Preprocessor1_Model5
10     5   300 roc_auc  binary     0.618     3 0.00174 Preprocessor1_Model5
11     5   700 accuracy binary     0.583     3 0.00215 Preprocessor1_Model6
12     5   700 roc_auc  binary     0.620     3 0.00219 Preprocessor1_Model6
13     7   300 accuracy binary     0.582     3 0.00323 Preprocessor1_Model7
14     7   300 roc_auc  binary     0.617     3 0.00261 Preprocessor1_Model7
15     7   700 accuracy binary     0.584     3 0.00344 Preprocessor1_Model8
16     7   700 roc_auc  binary     0.619     3 0.00233 Preprocessor1_Model8
```



0.1.3.4 Finalize the model

```r
rf_res |>
  show_best(metric = "roc_auc")
```

```
# A tibble: 5 x 8
   mtry trees .metric .estimator  mean     n std_err .config
  <int> <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
1     3   700 roc_auc binary     0.622     3 0.00168 Preprocessor1_Model2
2     4   700 roc_auc binary     0.621     3 0.00189 Preprocessor1_Model4
3     5   700 roc_auc binary     0.620     3 0.00219 Preprocessor1_Model6
4     3   300 roc_auc binary     0.619     3 0.00181 Preprocessor1_Model1
5     7   700 roc_auc binary     0.619     3 0.00233 Preprocessor1_Model8
```

```r
#Select best model
best_rf <- rf_res |>
  select_best(metric = "roc_auc")
best_rf
```

```
# A tibble: 1 x 3
   mtry trees .config
  <int> <int> <chr>
1     3   700 Preprocessor1_Model2
```

```r
final_rf <- finalize_workflow(rf_wf, best_rf)
```

```r
final_rf_fit <- final_rf |>
  last_fit(data_split)

final_rf_fit
```

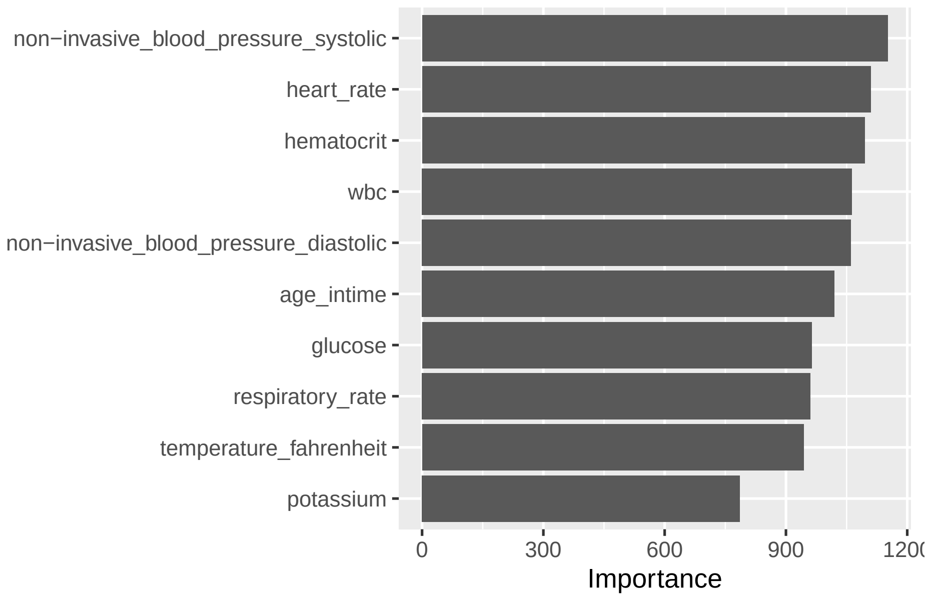
```
# Resampling results
# Manual resampling
# A tibble: 1 x 6
  splits                id              .metrics .notes   .predictions .workflow
  <list>                <chr>           <list>   <list>   <list>       <list>
1 <split [47221/47223]> train/test sp~ <tibble> <tibble> <tibble>     <workflow>
```

```r
final_rf_fit |>
  collect_metrics()
```

```
# A tibble: 3 x 4
  .metric     .estimator .estimate .config
  <chr>       <chr>          <dbl> <chr>
1 accuracy    binary         0.583 Preprocessor1_Model1
2 roc_auc     binary         0.619 Preprocessor1_Model1
3 brier_class binary         0.239 Preprocessor1_Model1
```

0.1.3.5 Visualize the final model

```r
# Extract fitted model
final_boost_tree <- extract_workflow(final_rf_fit)

final_boost_tree |>
  extract_fit_parsnip() |>
  vip()
```



4. Compare model classification performance on the test set. Report both the area under ROC curve and accuracy for each machine learning algorithm and the model stacking. Interpret the results. What are the most important features in predicting long ICU stays? How do the models compare in terms of performance and interpretability?

See Stacking.qmd file.