Biostat 203B Homework 5

Due Mar 20 @ 11:59PM

Emma Mo 906542365

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.
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.

```
set.seed(203)

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

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

3. Train and tune the models using the training set.

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?

```
# Load necessary libraries
library(ggplot2)
library(dplyr)
```

```
Attaching package: 'dplyr'

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

    filter, lag

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

    intersect, setdiff, setequal, union
```

```
library(caret)
```

```
Loading required package: lattice
```

```
library(tidymodels)
```

```
-- Attaching packages ------------------------------------ tidymodels 1.3.0 --

v broom        1.0.7     v rsample      1.2.1
v dials        1.4.0     v tibble       3.2.1
v infer        1.0.7     v tidyr        1.3.1
v modeldata    1.4.0     v tune         1.3.0
v parsnip      1.3.0     v workflows    1.2.0
v purrr        1.0.4     v workflowsets 1.1.0
v recipes      1.1.1     v yardstick    1.3.2
```

```
-- Conflicts --------------------------------------- tidymodels_conflicts() --
x purrr::discard()         masks scales::discard()
x dplyr::filter()          masks stats::filter()
x dplyr::lag()             masks stats::lag()
x purrr::lift()            masks caret::lift()
x yardstick::precision()   masks caret::precision()
x yardstick::recall()      masks caret::recall()
x yardstick::sensitivity() masks caret::sensitivity()
x yardstick::specificity() masks caret::specificity()
x recipes::step()          masks stats::step()
```

```r
library(ranger)
library(tidyr)

# Load dataset
mimic_data <- readRDS("../hw4/mimiciv_shiny/mimic_icu_cohort.rds")

# Select only the features available at ICU admission
icu_data <- mimic_data |>
  arrange(subject_id, hadm_id, stay_id) |>
  select(gender, age_intime, marital_status, race, first_careunit,
         glucose, potassium, sodium, chloride, creatinine, wbc,
         bicarbonate, hematocrit, heart_rate, temperature_fahrenheit,
         non_invasive_blood_pressure_diastolic,
         respiratory_rate, non_invasive_blood_pressure_systolic, los_long)
```

```r
set.seed(203)

icu_data_sorted <- icu_data |>
  filter(!is.na(los_long)) |>  # drop rows where los_long is NA
  mutate(los_long = as.factor(los_long)) # ensure it's a factor

# Create a 50/50 split stratified by los_long
data_split <- initial_split(icu_data_sorted,
                            strata = "los_long",
                            prop = 0.5)
data_split
```

```
<Training/Testing/Total>
<47221/47223/94444>
```

```r
train_data <- training(data_split)
test_data  <- testing(data_split)
dim(train_data)
```

```
[1] 47221    19
```

```r
dim(test_data)
```

```
[1] 47223    19
```

Random Forest

```r
# Convert categorical variables to factors
rf_recipe <- recipe(los_long ~ ., data = train_data) |>
  # Convert categorical variables to factors
  step_mutate_at(c("gender", "marital_status", "race", "first_careunit"),
                 fn = as.factor) |>
  # Impute missing numeric predictors with the mean
  step_impute_mean(all_numeric_predictors()) |>
  # Impute missing categorical predictors with the mode
  step_impute_mode(all_nominal_predictors()) |>
  # Remove predictors with zero variance
  step_zv(all_predictors()) |>
  print()
```

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

```
-- Inputs
```

```
Number of variables by role
```

```
outcome:    1
predictor: 18
```

```
-- Operations

* Variable mutation for: c("gender", "marital_status", "race",
  "first_careunit")

* Mean imputation for: all_numeric_predictors()

* Mode imputation for: all_nominal_predictors()

* Zero variance filter on: all_predictors()
```

```
rf_spec <- rand_forest(
  mtry  = tune(),   # number of variables randomly sampled at each split
  trees = tune(),   # number of trees in the ensemble
  min_n = tune()    # minimum number of observations in terminal nodes
) |>
  set_engine("ranger", importance = "impurity") |>
  set_mode("classification")
rf_spec
```

```
Random Forest Model Specification (classification)

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

Engine-Specific Arguments:
  importance = impurity

Computational engine: ranger
```

```
# Create the workflow combining the recipe and the model
rf_workflow <- workflow() |>
  add_recipe(rf_recipe) |>
  add_model(rf_spec)
rf_workflow
```

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

-- Preprocessor ----------------------------------------------------------------
4 Recipe Steps

* step_mutate_at()
* step_impute_mean()
* step_impute_mode()
* step_zv()

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

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

Engine-Specific Arguments:
  importance = impurity

Computational engine: ranger
```

```r
coarse_rf_grid <- grid_regular(
  mtry(range = c(3, 3)),
  trees(range = c(110, 130)),
  min_n(range = c(4, 4)),
  levels = 3
)
```

```r
# Create 5-fold cross-validation, stratified by los_long
set.seed(203)
icu_folds <- vfold_cv(train_data, v = 5, strata = los_long)
icu_folds
```

```
#  5-fold cross-validation using stratification
# A tibble: 5 x 2
  splits               id
  <list>               <chr>
1 <split [37776/9445]> Fold1
```

```
2 <split [37776/9445]> Fold2
3 <split [37776/9445]> Fold3
4 <split [37778/9443]> Fold4
5 <split [37778/9443]> Fold5
```

```
rf_tune_results <- tune_grid(
  rf_workflow,
  resamples = icu_folds,
  grid = coarse_rf_grid,
  metrics = metric_set(roc_auc, accuracy),
  control = control_grid(
    save_pred = TRUE,
    save_workflow = TRUE
  )
)
```

```
i The workflow being saved contains a recipe, which is 6.2 Mb in i memory. If
this was not intentional, please set the control setting i `save_workflow =
FALSE`.
```

```
rf_tune_results
```

```
# Tuning results
# 5-fold cross-validation using stratification
# A tibble: 5 x 5
  splits               id    .metrics          .notes           .predictions
  <list>               <chr> <list>            <list>           <list>
1 <split [37776/9445]> Fold1 <tibble [6 x 7]> <tibble [0 x 3]> <tibble>
2 <split [37776/9445]> Fold2 <tibble [6 x 7]> <tibble [0 x 3]> <tibble>
3 <split [37776/9445]> Fold3 <tibble [6 x 7]> <tibble [0 x 3]> <tibble>
4 <split [37778/9443]> Fold4 <tibble [6 x 7]> <tibble [0 x 3]> <tibble>
5 <split [37778/9443]> Fold5 <tibble [6 x 7]> <tibble [0 x 3]> <tibble>
```

```
# show top models
rf_tune_results |> show_best(metric = "roc_auc")
```

```
# A tibble: 3 x 9
   mtry trees min_n .metric .estimator  mean     n std_err .config
  <int> <int> <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
1     3   120     4 roc_auc binary     0.625     5 0.00103 Preprocessor1_Model2
2     3   130     4 roc_auc binary     0.625     5 0.00199 Preprocessor1_Model3
3     3   110     4 roc_auc binary     0.624     5 0.00176 Preprocessor1_Model1
```

```r
# select the best model
best_rf_params <- select_best(rf_tune_results, metric = "roc_auc")
best_rf_params
```

```
# A tibble: 1 x 4
   mtry trees min_n .config
  <int> <int> <int> <chr>
1     3   120     4 Preprocessor1_Model2
```

```r
# Final workflow
final_rf_workflow <- finalize_workflow(rf_workflow, best_rf_params)
final_rf_workflow
```

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

-- Preprocessor ----------------------------------------------------------------
4 Recipe Steps

* step_mutate_at()
* step_impute_mean()
* step_impute_mode()
* step_zv()

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

Main Arguments:
  mtry = 3
  trees = 120
  min_n = 4

Engine-Specific Arguments:
  importance = impurity

Computational engine: ranger
```

```r
# Fit the final model on the training data
final_rf_fit <- final_rf_workflow |> last_fit(data_split)
rf_metrics <- final_rf_fit |>
```

```
  collect_metrics() |>
  filter(.metric %in% c("accuracy", "roc_auc")) |>
  select(.metric, .estimate) |>
  pivot_wider(names_from = .metric, values_from = .estimate) |>
  mutate(model = "Random Forest")
rf_metrics
```

```
# A tibble: 1 x 3
  accuracy roc_auc model
     <dbl>   <dbl> <chr>
1    0.586   0.620 Random Forest
```

Logit

```
library(tidymodels)
library(GGally)
```

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

```
library(gtsummary)
library(tidyverse)
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v forcats   1.0.0     v readr     2.1.5
v lubridate 1.9.3     v stringr   1.5.1

-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x readr::col_factor() masks scales::col_factor()
x purrr::discard()    masks scales::discard()
x dplyr::filter()     masks stats::filter()
x stringr::fixed()    masks recipes::fixed()
x dplyr::lag()        masks stats::lag()
x purrr::lift()       masks caret::lift()
x readr::spec()       masks yardstick::spec()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to becor
```

```r
log_reg_recipe <- recipe(los_long ~ ., data = train_data) |>
  # Converts to factors
  step_mutate_at(c("gender", "marital_status", "race", "first_careunit"),
                 fn = as.factor) |>
  step_impute_mean(all_numeric_predictors()) |>
  step_impute_mode(all_nominal_predictors()) |>
  # zero-variance filter
  step_zv(all_numeric_predictors()) |>
  # create traditional dummy variables (necessary for svm)
  step_dummy(all_nominal_predictors()) |>
  # center and scale numeric data
  step_normalize(all_numeric_predictors()) |>
  print()
```

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

-- Inputs
Number of variables by role
outcome:    1
predictor: 18

-- Operations
* Variable mutation for: c("gender", "marital_status", "race",
  "first_careunit")
* Mean imputation for: all_numeric_predictors()
* Mode imputation for: all_nominal_predictors()
* Zero variance filter on: all_numeric_predictors()
* Dummy variables from: all_nominal_predictors()
* Centering and scaling for: all_numeric_predictors()
```

```r
# Define a logistic regression model with elastic net regularization.
# We tune 'penalty' and 'mixture'.
log_reg_spec <- logistic_reg(
  penalty = tune(),
  mixture = tune()
) |>
  set_engine("glmnet") |>
  set_mode("classification")
log_reg_spec
```

```
Logistic Regression Model Specification (classification)
```

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

Computational engine: glmnet
```

```
# Build the workflow by combining the recipe and the model
log_reg_workflow <- workflow() |>
  add_recipe(log_reg_recipe) |>
  add_model(log_reg_spec)
log_reg_workflow
```

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

-- Preprocessor ----------------------------------------------------------------
6 Recipe Steps

* step_mutate_at()
* step_impute_mean()
* step_impute_mode()
* step_zv()
* step_dummy()
* step_normalize()

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

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

Computational engine: glmnet
```

```
# Tuning
logit_grid <- grid_regular(
  penalty(),
  mixture()
)
logit_grid
```

```
# A tibble: 9 x 2
       penalty mixture
         <dbl>   <dbl>
1 0.0000000001     0
2 0.00001          0
3 1                0
4 0.0000000001     0.5
5 0.00001          0.5
6 1                0.5
7 0.0000000001     1
8 0.00001          1
9 1                1
```

```
# Create 5-fold cross-validation, stratified by los_long
set.seed(203)
icu_folds <- vfold_cv(train_data, v = 5, strata = los_long)
```

```
# Tune the SVM model using the coarse grid
set.seed(203)
log_reg_tune <- tune_grid(
  log_reg_workflow,
  resamples = icu_folds,
  grid = logit_grid,
  metrics = metric_set(roc_auc, accuracy),
  control = control_grid(
    save_pred = TRUE,
    save_workflow = TRUE
  )
)
```

```
i The workflow being saved contains a recipe, which is 6.23 Mb in i memory. If
this was not intentional, please set the control setting i `save_workflow =
FALSE`.
```

```
log_reg_tune
```

```
# Tuning results
# 5-fold cross-validation using stratification
# A tibble: 5 x 5
  splits               id    .metrics          .notes           .predictions
  <list>               <chr> <list>            <list>           <list>
```

```
1 <split [37776/9445]> Fold1 <tibble [18 x 6]> <tibble [0 x 3]> <tibble>
2 <split [37776/9445]> Fold2 <tibble [18 x 6]> <tibble [0 x 3]> <tibble>
3 <split [37776/9445]> Fold3 <tibble [18 x 6]> <tibble [0 x 3]> <tibble>
4 <split [37778/9443]> Fold4 <tibble [18 x 6]> <tibble [0 x 3]> <tibble>
5 <split [37778/9443]> Fold5 <tibble [18 x 6]> <tibble [0 x 3]> <tibble>
```

```
log_reg_tune |> 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 0.0000000001     0   roc_auc binary     0.596     5 0.00136 Preprocessor1_Mod~
2 0.00001          0   roc_auc binary     0.596     5 0.00136 Preprocessor1_Mod~
3 0.0000000001     0.5 roc_auc binary     0.596     5 0.00132 Preprocessor1_Mod~
4 0.00001          0.5 roc_auc binary     0.596     5 0.00132 Preprocessor1_Mod~
5 0.0000000001     1   roc_auc binary     0.596     5 0.00131 Preprocessor1_Mod~
```

```
best_logit <- log_reg_tune |>
  select_best(metric = "roc_auc")
best_logit
```

```
# A tibble: 1 x 3
       penalty mixture .config
         <dbl>   <dbl> <chr>
1 0.0000000001       0 Preprocessor1_Model1
```

```
final_log_reg_workflow <- log_reg_workflow |>
  finalize_workflow(best_logit)
final_log_reg_workflow
```

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

-- Preprocessor ----------------------------------------------------------------
6 Recipe Steps

* step_mutate_at()
* step_impute_mean()
* step_impute_mode()
```

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

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

Main Arguments:
  penalty = 1e-10
  mixture = 0

Computational engine: glmnet
```

```
# Fit the whole training set, then predict the test cases
final_log_reg_fit <- final_log_reg_workflow |> last_fit(data_split)

# Collect and view performance metrics on the test set
logit_metrics <- final_log_reg_fit |>
  collect_metrics() |>
  filter(.metric %in% c("accuracy", "roc_auc")) |>
  select(.metric, .estimate) |>
  pivot_wider(names_from = .metric, values_from = .estimate) |>
  mutate(model = "Logistic Regression")
logit_metrics
```

```
# A tibble: 1 x 3
  accuracy roc_auc model
     <dbl>   <dbl> <chr>
1    0.565   0.592 Logistic Regression
```

Boosting (XGboost)

```
library(xgboost)
```

```
Attaching package: 'xgboost'

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

    slice
```

```r
xgb_recipe <- recipe(los_long ~ ., data = train_data) |>
  step_mutate_at(c("gender", "marital_status", "race", "first_careunit"),
                 fn = as.factor) |>
  step_impute_mean(all_numeric_predictors()) |>
  step_impute_mode(all_nominal_predictors()) |>
  step_zv(all_predictors()) |>
  step_dummy(all_nominal_predictors()) |>
  print()
```

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



-- Inputs

Number of variables by role

outcome:    1
predictor: 18



-- Operations

* Variable mutation for: c("gender", "marital_status", "race",
  "first_careunit")

* Mean imputation for: all_numeric_predictors()

* Mode imputation for: all_nominal_predictors()

* Zero variance filter on: all_predictors()

* Dummy variables from: all_nominal_predictors()
```

```
# tune key hyperparameters:
#   - trees: number of trees (boosting rounds)
#   - learn_rate: learning rate (shrinkage)
#   - tree_depth: maximum depth of trees
xgb_spec <- boost_tree(
  trees = tune(),
  learn_rate = tune(),
  tree_depth = tune()
) |>
  set_engine("xgboost") |>
  set_mode("classification")
xgb_spec
```

```
Boosted Tree Model Specification (classification)

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

Computational engine: xgboost
```

```
xgb_workflow <- workflow() |>
  add_recipe(xgb_recipe) |>
  add_model(xgb_spec)
xgb_workflow
```

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

-- Preprocessor ----------------------------------------------------------------
5 Recipe Steps

* step_mutate_at()
* step_impute_mean()
* step_impute_mode()
* step_zv()
* step_dummy()

-- Model -----------------------------------------------------------------------
```

```
Boosted Tree Model Specification (classification)

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

Computational engine: xgboost
```

```
narrow_xgb_grid <- grid_regular(
  trees(range = c(110, 130)),
  learn_rate(range = c(0.05, 0.3)),
  tree_depth(range = c(3, 3)),
  levels = 3
)
narrow_xgb_grid
```

```
# A tibble: 9 x 3
  trees learn_rate tree_depth
  <int>      <dbl>      <int>
1   110       1.12          3
2   120       1.12          3
3   130       1.12          3
4   110       1.50          3
5   120       1.50          3
6   130       1.50          3
7   110       2.00          3
8   120       2.00          3
9   130       2.00          3
```

```
# Create 5-fold cross-validation, stratified by los_long
set.seed(203)
icu_folds <- vfold_cv(train_data, v = 5, strata = los_long)
```

```
set.seed(203)
xgb_tune_results <- tune_grid(
  xgb_workflow,
  resamples = icu_folds,
  grid = narrow_xgb_grid,
  metrics = metric_set(roc_auc, accuracy),
  control = control_grid(
```

```
    save_pred = TRUE,
    save_workflow = TRUE
  )
)
```

```
i The workflow being saved contains a recipe, which is 6.23 Mb in i memory. If
this was not intentional, please set the control setting i `save_workflow =
FALSE`.
```

```
xgb_tune_results
```

```
# Tuning results
# 5-fold cross-validation using stratification
# A tibble: 5 x 5
  splits               id    .metrics          .notes           .predictions
  <list>               <chr> <list>            <list>           <list>
1 <split [37776/9445]> Fold1 <tibble [18 x 7]> <tibble [0 x 3]> <tibble>
2 <split [37776/9445]> Fold2 <tibble [18 x 7]> <tibble [0 x 3]> <tibble>
3 <split [37776/9445]> Fold3 <tibble [18 x 7]> <tibble [0 x 3]> <tibble>
4 <split [37778/9443]> Fold4 <tibble [18 x 7]> <tibble [0 x 3]> <tibble>
5 <split [37778/9443]> Fold5 <tibble [18 x 7]> <tibble [0 x 3]> <tibble>
```

```
# Show the top 5 models
xgb_tune_results |> show_best(metric = "roc_auc")
```

```
# A tibble: 5 x 9
  trees tree_depth learn_rate .metric .estimator  mean     n std_err .config
  <int>      <int>      <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
1   110          3       1.12 roc_auc binary     0.611     5 0.00126 Preprocess~
2   120          3       1.12 roc_auc binary     0.610     5 0.00107 Preprocess~
3   130          3       1.12 roc_auc binary     0.609     5 0.00168 Preprocess~
4   110          3       1.50 roc_auc binary     0.597     5 0.00356 Preprocess~
5   120          3       1.50 roc_auc binary     0.596     5 0.00360 Preprocess~
```

```
# select the best one
best_xgb_params <- xgb_tune_results |>
  select_best(metric = "roc_auc")
best_xgb_params
```

```
# A tibble: 1 x 4
  trees tree_depth learn_rate .config
  <int>      <int>      <dbl> <chr>
1   110          3       1.12 Preprocessor1_Model1
```

```
final_xgb_workflow <- xgb_workflow |>
  finalize_workflow(best_xgb_params)
final_xgb_workflow
```

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

-- Preprocessor ----------------------------------------------------------------
5 Recipe Steps

* step_mutate_at()
* step_impute_mean()
* step_impute_mode()
* step_zv()
* step_dummy()

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

Main Arguments:
  trees = 110
  tree_depth = 3
  learn_rate = 1.12201845430196

Computational engine: xgboost
```

```
final_xgb_fit <- final_xgb_workflow |>
  last_fit(data_split)
final_xgb_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
xgb_metrics <- final_xgb_fit |>
  collect_metrics() |>
  filter(.metric %in% c("accuracy", "roc_auc")) |>
  select(.metric, .estimate) |>
  pivot_wider(names_from = .metric, values_from = .estimate) |>
  mutate(model = "XGBoost")
xgb_metrics
```

```
# A tibble: 1 x 3
  accuracy roc_auc model
     <dbl>   <dbl> <chr>
1    0.580   0.612 XGBoost
```

Stacking

```r
library(stacks)
library(tidymodels)
library(dplyr)
library(yardstick)
library(pROC)
```

```
Type 'citation("pROC")' for a citation.


Attaching package: 'pROC'

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

    cov, smooth, var
```

```r
model_stack <- stacks() |>
  add_candidates(rf_tune_results, top_n = 3) |>
  add_candidates(log_reg_tune, top_n = 3) |>
  add_candidates(xgb_tune_results, top_n = 3) |>
  # Blend predictions from all base learners
  blend_predictions() |>
  # fit the candidates with nonzero stacking coefficients
  fit_members()
```

```
Warning: The `...` are not used in this function but one or more arguments were passed: 'top
The `...` are not used in this function but one or more arguments were passed: 'top_n'
The `...` are not used in this function but one or more arguments were passed: 'top_n'

Warning: Predictions from 8 candidates were identical to those from existing candidates
and were removed from the data stack.

Warning: Predictions from 4 candidates were identical to those from existing candidates
and were removed from the data stack.
```

```
model_stack
```

```
-- A stacked ensemble model -------------------------------------



Out of 15 possible candidate members, the ensemble retained 9.

Penalty: 0.001.

Mixture: 1.


The 9 highest weighted member classes are:

# A tibble: 9 x 3
  member                          type         weight
  <chr>                           <chr>         <dbl>
1 .pred_TRUE_rf_tune_results_1_2  rand_forest  1.15
2 .pred_TRUE_rf_tune_results_1_3  rand_forest  0.997
3 .pred_TRUE_rf_tune_results_1_1  rand_forest  0.958
4 .pred_TRUE_log_reg_tune_1_4     logistic_reg 0.854
5 .pred_TRUE_xgb_tune_results_1_1 boost_tree   0.449
6 .pred_TRUE_xgb_tune_results_1_3 boost_tree   0.259
7 .pred_TRUE_xgb_tune_results_1_4 boost_tree   0.227
8 .pred_TRUE_log_reg_tune_1_7     logistic_reg 0.113
9 .pred_TRUE_xgb_tune_results_1_6 boost_tree   0.0878
```

```
test_results <- test_data |>
  bind_cols(predict(model_stack, new_data = test_data, type = "prob"))
```

```
stack_preds <- predict(model_stack, new_data = test_data, type = "prob")
stack_class_preds <- predict(model_stack, new_data = test_data, type = "class")
results <- test_data |>
  bind_cols(stack_preds) |>
  bind_cols(stack_class_preds)
```

```
library(dplyr)
library(pROC)
library(yardstick)

results <- results |> rename(.pred_1 = .pred_TRUE)

# Accuracy
stack_acc_value <- results |>
  accuracy(truth = los_long, estimate = .pred_class)

# ROC AUC
stack_roc_obj <- roc(
  response = results$los_long,
  predictor = results$.pred_1,  # or .pred_TRUE, after renaming
  levels = rev(levels(results$los_long))
)
```

```
Setting direction: controls > cases
```

```
auc_value <- auc(stack_roc_obj)

# Combine into a df
stack_metrics_df <- data.frame(
  metric = c("stack_accuracy", "stack_auc"),
  value = c(stack_acc_value$.estimate, auc_value)
)

stack_metrics_df
```

```
          metric     value
1 stack_accuracy 0.5953243
2      stack_auc 0.6361625
```

```
stack_metrics_wide <- stack_metrics_df |>
  pivot_wider(names_from = metric, values_from = value) |>
  rename(accuracy = stack_accuracy, roc_auc = stack_auc) |>
  mutate(model = "Stacking")

stack_metrics_wide
```

```
# A tibble: 1 x 3
  accuracy roc_auc model
     <dbl>   <dbl> <chr>
1    0.595   0.636 Stacking
```

Final comparison

Compare the accuracy and AUC

```
model_comparison <- bind_rows(
  rf_metrics,
  logit_metrics,
  xgb_metrics,
  stack_metrics_wide
)

model_comparison
```

```
# A tibble: 4 x 3
  accuracy roc_auc model
     <dbl>   <dbl> <chr>
1    0.586   0.620 Random Forest
2    0.565   0.592 Logistic Regression
3    0.580   0.612 XGBoost
4    0.595   0.636 Stacking
```

- Stacking has the highest AUC and accuracy, meaning that the ensemble improves performance.
- Random forest and XGBoost have moderate AUC and accuracy
- Logistic regression has the lowest accuracy and AUC, though the difference is slight.

Find the most important features of each model

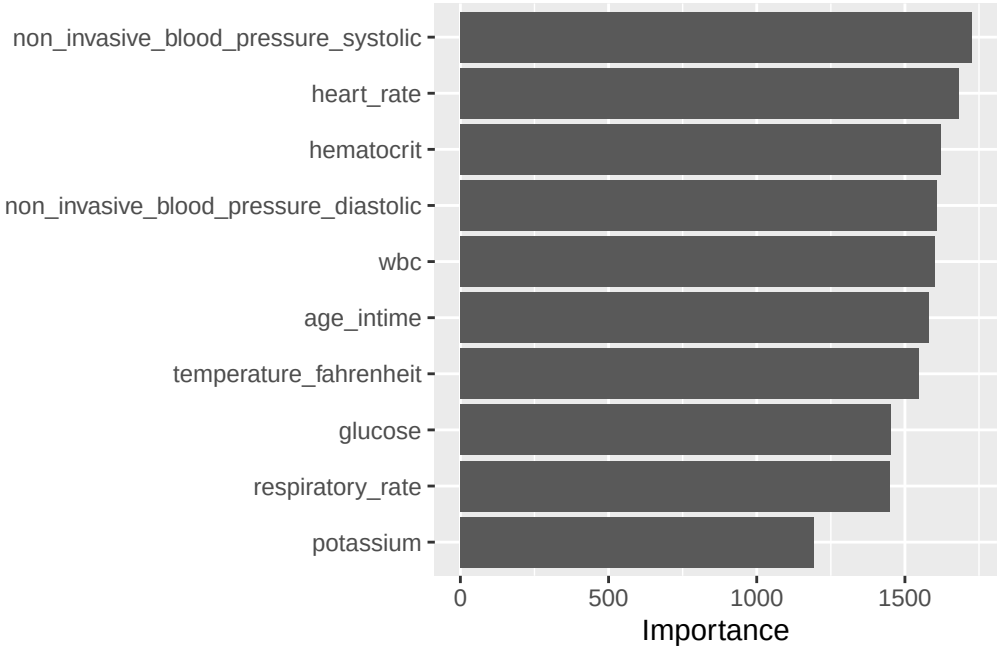
```
## Random forest
library(tidymodels)
library(vip)
```

```
Attaching package: 'vip'
```

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

    vi
```

```
# Extract the underlying parsnip model, then plot VIP
final_rf_fit |>
  extract_fit_parsnip() |>
  vip::vip(num_features = 10)
```



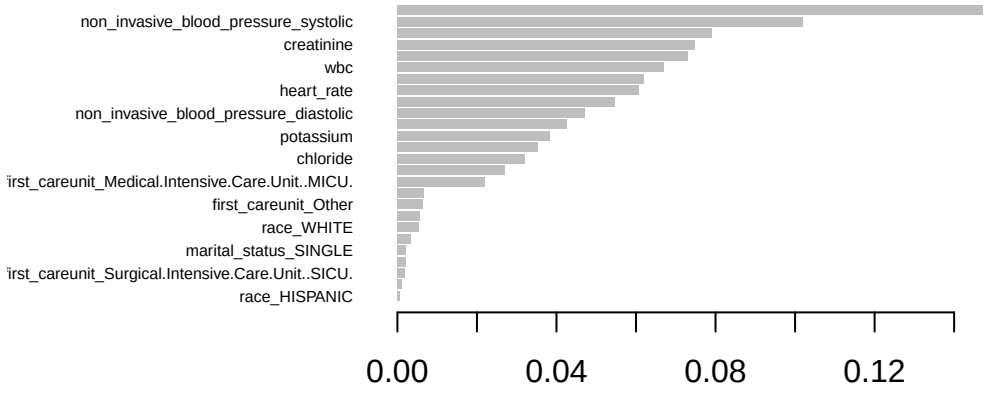
- Random forest
  - From the Variable Importance plot, the top 3 contributors are:
    - `non_invasive_blood_pressure_systolic`, `heart_rate`, and `hematocrit`
  - They most frequently help differentiate between longer vs shorter stays

  - These 3 variables contribute slightly more than the rest of the top 10 variables.

```
## XGboost
library(xgboost)
library(tidymodels)
library(vip)

# Extract the underlying xgboost model
xgb_model <- final_xgb_fit |>
  extract_fit_parsnip()
xgb_model <- xgb_model$fit

# Use xgb.importance() + xgb.plot.importance()
importance <- xgb.importance(model = xgb_model)
xgb.plot.importance(importance)
```



- XGboost
  - The most important feature is `temperature_fahrenheit`.
  - It contributes dramatically more than the second variable (`non_invasive_blood_pressure_systolic`

```r
## Logit
library(broom)

log_reg_model <- final_log_reg_fit |>
  extract_fit_parsnip()

coefs <- tidy(log_reg_model) |>
  filter(term != "(Intercept)") |>
  mutate(abs_estimate = abs(estimate)) |>
  arrange(desc(abs_estimate))

head(coefs, 10)  # top 10 largest absolute coefficients
```

```
# A tibble: 10 x 4
   term                                            estimate penalty abs_estimate
   <chr>                                              <dbl>   <dbl>        <dbl>
 1 first_careunit_Medical.Surgical.Intensive.Care~  -0.159    1e-10       0.159
 2 heart_rate                                        0.130    1e-10       0.130
 3 respiratory_rate                                  0.129    1e-10       0.129
 4 non_invasive_blood_pressure_systolic             -0.102    1e-10       0.102
 5 hematocrit                                       -0.101    1e-10       0.101
 6 first_careunit_Medical.Intensive.Care.Unit..MI~  -0.0956   1e-10       0.0956
 7 chloride                                         -0.0909   1e-10       0.0909
 8 wbc                                               0.0838   1e-10       0.0838
 9 age_intime                                        0.0831   1e-10       0.0831
10 race_WHITE                                       -0.0822   1e-10       0.0822
```

- Logistic regression
  - The top 3 contributors:
    - `first_careunit` (negative), `heart_rate` (positive), and `respiratory_rate` (positive)
- Interpretability
  - Stacking provides improvement in AUC and accuracy
  - Logistic Regression is good at providing coefficient signs and magnitudes of predictors
  - Random Forest and XGBoost are more complex but provide variable importance
  - The most critical predictors across models:
    - `heart_rate`, `respiratory_rate` and `non_invasive_blood_pressure_systolic`.