Biostat 203B Homework 5

Due Mar 20 @ 11:59PM

Wenjing Zhou and 806542441

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.

Loading necessary libraries

```r
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
```

```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 tibble       3.2.1
v ggplot2      3.5.1     v tidyr        1.3.1
v infer        1.0.7     v tune         1.3.0
v modeldata    1.4.0     v workflows    1.2.0
v parsnip      1.3.0     v workflowsets 1.1.0
v purrr        1.0.4     v yardstick    1.3.2
v recipes      1.1.1

-- Conflicts ----------------------------------------- tidymodels_conflicts() --
x purrr::discard() masks scales::discard()
x dplyr::filter()  masks stats::filter()
x dplyr::lag()     masks stats::lag()
x recipes::step()  masks stats::step()
```

```r
library(vip)  # For variable importance
```

```
Attaching package: 'vip'

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

    vi
```

```r
library(xgboost)
```

```
Attaching package: 'xgboost'

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

    slice
```

```r
library(doParallel) # For parallel processing
```

```
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(pROC)
```

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


Attaching package: 'pROC'

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

    cov, smooth, var
```

Loading dataset

```r
icu_data <- readRDS("mimic_icu_cohort.rds") %>%
    mutate(los_long = los > 2)
icu_data
```

```
# A tibble: 94,458 x 41
   subject_id  hadm_id  stay_id first_careunit last_careunit intime
        <int>    <int>    <int> <chr>          <chr>         <dttm>
 1   10000032 29079034 39553978 Medical Inten~ Medical Inte~ 2180-07-23 14:00:00
 2   10000690 25860671 37081114 Medical Inten~ Medical Inte~ 2150-11-02 19:37:00
 3   10000980 26913865 39765666 Medical Inten~ Medical Inte~ 2189-06-27 08:42:00
 4   10001217 24597018 37067082 Surgical Inte~ Surgical Int~ 2157-11-20 19:18:02
 5   10001217 27703517 34592300 Surgical Inte~ Surgical Int~ 2157-12-19 15:42:24
 6   10001725 25563031 31205490 Medical/Surgi~ Medical/Surg~ 2110-04-11 15:52:22
 7   10001843 26133978 39698942 Medical/Surgi~ Medical/Surg~ 2134-12-05 18:50:03
 8   10001884 26184834 37510196 Medical Inten~ Medical Inte~ 2131-01-11 04:20:05
 9   10002013 23581541 39060235 Cardiac Vascu~ Cardiac Vasc~ 2160-05-18 10:00:53
10   10002114 27793700 34672098 Coronary Care~ Coronary Car~ 2162-02-17 23:30:00
# i 94,448 more rows
# i 35 more variables: outtime <dttm>, los <dbl>, admittime <dttm>,
#   dischtime <dttm>, deathtime <dttm>, admission_type <chr>,
#   admit_provider_id <chr>, admission_location <chr>,
#   discharge_location <chr>, insurance <chr>, language <chr>,
#   marital_status <chr>, race <chr>, edregtime <dttm>, edouttime <dttm>,
#   hospital_expire_flag <int>, gender <chr>, anchor_age <int>, ...
```

Data cleaning

```r
icu_clean <- icu_data %>%
  select(-subject_id, -hadm_id, -stay_id, -last_careunit, -intime, -outtime,
         -dischtime,-admittime, -admit_provider_id, -deathtime, -edregtime,
         -edouttime, -dod) %>%
  select(-c(discharge_location, hospital_expire_flag, los)) %>%
  select(-c(anchor_year, anchor_year_group)) %>% # Remove future or unnecessary info
  mutate(
    los_long = factor(los_long,
                      levels = c(FALSE, TRUE),
                      labels = c("Yes", "No")
                      ),
    gender = factor(gender),
    marital_status = factor(marital_status),
    race = factor(race),
    first_careunit = factor(first_careunit)
  ) %>%
  drop_na()
```

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.

Train-Test Split

```
#Train-Test Split
set.seed(203)

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

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

Cross-validation folds

```
set.seed(203)

folds <- vfold_cv(train_data, v = 5, strata = los_long)

icu_folds <- folds
```

Recipe

```
icu_recipe <- recipe(los_long ~ ., data = train_data) %>%
  step_novel(all_nominal_predictors()) %>%
  step_other(all_nominal_predictors(), threshold = 0.01) %>%
  step_dummy(all_nominal_predictors()) %>%
  step_zv(all_predictors()) %>%    #
  step_normalize(all_numeric_predictors())  # Standardize numeric features
```

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

Model 1: Logistic Regression (Elastic Net)

```r
log_reg_spec <- logistic_reg(
  penalty = tune(),
  mixture = tune()
) %>%
  set_engine("glmnet")

log_reg_workflow <- workflow() %>%
  add_model(log_reg_spec) %>%
  add_recipe(icu_recipe)

set.seed(123)

log_reg_grid_fine <- grid_regular(
  penalty(range =10^c(-6, -1)),
  mixture(range = c(0, 1)),
  levels = 5
)

log_reg_res_fine <- tune_grid(
  log_reg_workflow,
  resamples = vfold_cv(train_data, v = 5, strata = los_long),
  grid = log_reg_grid_fine,
  metrics = metric_set(roc_auc)
)

# show 25 models
log_reg_res_fine %>% collect_metrics()
```

```
# A tibble: 25 x 8
   penalty mixture .metric .estimator  mean     n std_err .config
     <dbl>   <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
 1    1.00    0    roc_auc binary     0.598     5 0.00187 Preprocessor1_Model01
 2    1.06    0    roc_auc binary     0.598     5 0.00189 Preprocessor1_Model02
 3    1.12    0    roc_auc binary     0.598     5 0.00189 Preprocessor1_Model03
 4    1.19    0    roc_auc binary     0.598     5 0.00191 Preprocessor1_Model04
 5    1.26    0    roc_auc binary     0.598     5 0.00191 Preprocessor1_Model05
 6    1.00    0.25 roc_auc binary     0.5       5 0       Preprocessor1_Model06
 7    1.06    0.25 roc_auc binary     0.5       5 0       Preprocessor1_Model07
 8    1.12    0.25 roc_auc binary     0.5       5 0       Preprocessor1_Model08
 9    1.19    0.25 roc_auc binary     0.5       5 0       Preprocessor1_Model09
10    1.26    0.25 roc_auc binary     0.5       5 0       Preprocessor1_Model10
# i 15 more rows
```

```r
# Show the best roc
log_reg_res_fine %>% 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    1.00       0 roc_auc binary     0.598     5 0.00187 Preprocessor1_Model01
2    1.06       0 roc_auc binary     0.598     5 0.00189 Preprocessor1_Model02
3    1.12       0 roc_auc binary     0.598     5 0.00189 Preprocessor1_Model03
4    1.19       0 roc_auc binary     0.598     5 0.00191 Preprocessor1_Model04
5    1.26       0 roc_auc binary     0.598     5 0.00191 Preprocessor1_Model05
```

```r
# choose the best model
best_fine <- log_reg_res_fine %>%
  select_best(metric = "roc_auc")

best_fine
```

```
# A tibble: 1 x 3
  penalty mixture .config
    <dbl>   <dbl> <chr>
1    1.00       0 Preprocessor1_Model01
```

Make sure the final model is trained on the entire training set and evaluated on the test set.

```r
final_workflow_fine <- log_reg_workflow %>%
  finalize_workflow(best_fine)

# train the final model on the full training set
final_fit_fine <- final_workflow_fine %>%
  last_fit(split = data_split)

# evaluat on the test set
final_fit_fine %>% collect_metrics()
```

```
# A tibble: 3 x 4
  .metric     .estimator .estimate .config
  <chr>       <chr>          <dbl> <chr>
1 accuracy    binary         0.558 Preprocessor1_Model1
2 roc_auc     binary         0.596 Preprocessor1_Model1
3 brier_class binary         0.246 Preprocessor1_Model1
```

Model 2: Random Forest

```r
set.seed(203)

# define the RF model
rf_spec <- rand_forest(
  mtry = tune(),
  min_n = tune(),
  trees = 500
) %>%
  set_mode("classification") %>%
  set_engine("ranger", importance = "impurity")

# set workflow
rf_workflow <- workflow() %>%
  add_model(rf_spec) %>%
  add_recipe(icu_recipe)

# grid search
rf_grid <- grid_random(
  mtry(range = c(2, 10)),
  min_n(range = c(5, 20)),
  size = 10
)

# tune grid
rf_res <- tune_grid(
  rf_workflow,
  resamples = folds,
  grid = rf_grid,
  metrics = metric_set(roc_auc)
)
```

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

```
# A tibble: 5 x 8
   mtry min_n .metric .estimator  mean     n std_err .config
  <int> <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
1     5    16 roc_auc binary     0.654     5 0.00456 Preprocessor1_Model7
2     4     7 roc_auc binary     0.654     5 0.00394 Preprocessor1_Model4
3     5    13 roc_auc binary     0.654     5 0.00384 Preprocessor1_Model6
4     6    12 roc_auc binary     0.653     5 0.00420 Preprocessor1_Model2
```

```
5     3     7 roc_auc binary     0.653     5 0.00522 Preprocessor1_Model8
```

```
# choose the best parameters
best_rf <- rf_res %>% select_best(metric = "roc_auc")
best_rf
```

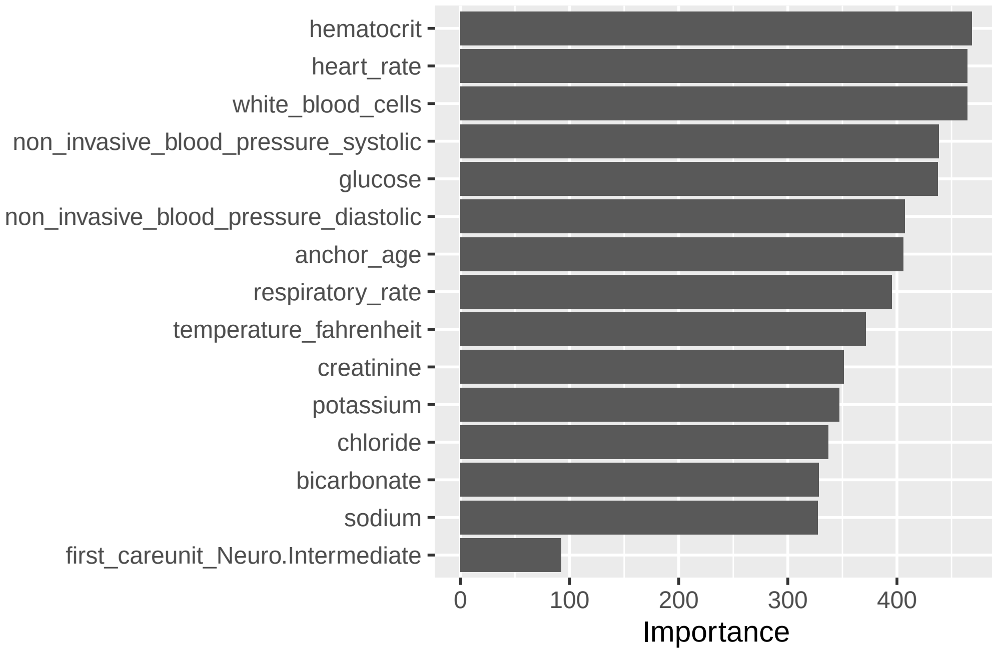
```
# A tibble: 1 x 3
   mtry min_n .config
  <int> <int> <chr>
1     5    16 Preprocessor1_Model7
```

```
# final workflow
final_rf_workflow <- rf_workflow %>% finalize_workflow(best_rf)

# fit the final model
final_rf_fit <- final_rf_workflow %>% last_fit(data_split)

# evaluate on the test set
final_rf_fit %>% collect_metrics()
```

```
# A tibble: 3 x 4
  .metric     .estimator .estimate .config
  <chr>       <chr>          <dbl> <chr>
1 accuracy    binary         0.609 Preprocessor1_Model1
2 roc_auc     binary         0.651 Preprocessor1_Model1
3 brier_class binary         0.233 Preprocessor1_Model1
```

Importance of features (vip)

```
# fit the final model to whole training set
final_rf_fit <- final_rf_workflow %>%
  last_fit(data_split)

# feature importance
final_rf_fit %>%
  extract_fit_parsnip() %>%
  vip(num_features = 15)
```



Model 3: XGBoost

```
xgb_spec <- boost_tree(
  trees = 500,
  tree_depth = tune(),
  learn_rate = tune(),
  min_n = tune(),
  #loss_reduction = tune()
) %>%
  set_engine("xgboost") %>%
  set_mode("classification")
```

```
xgb_workflow <- workflow() %>%
  add_recipe(icu_recipe) %>%
  add_model(xgb_spec)
```

```
set.seed(123)

xgb_grid <- grid_random(
  learn_rate(range = c(-3, -1)),
  tree_depth(range = c(3, 8)),
  min_n(range = c(5, 20)),
```

```
  size = 10
)
```

Perform cross-validation for hyperparameter tuning (enable parallel acceleration)

```
library(doFuture)
```

```
Loading required package: future
```

```
library(doParallel)

cores <- parallel::detectCores() - 1
registerDoFuture()
plan(multisession, workers = cores)

set.seed(123)
xgb_res <- tune_grid(
  xgb_workflow,
  resamples = vfold_cv(train_data, v = 5, strata = los_long),
  grid = xgb_grid,
  metrics = metric_set(roc_auc)
)

plan(sequential)
```

```
xgb_res %>% show_best(metric ="roc_auc")
```

```
# A tibble: 5 x 9
  min_n tree_depth learn_rate .metric .estimator  mean     n std_err .config
  <int>      <int>      <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
1    17          3     0.0583 roc_auc binary     0.650     5 0.00206 Preprocess~
2    14          7     0.0114 roc_auc binary     0.650     5 0.00189 Preprocess~
3     7          4     0.0760 roc_auc binary     0.648     5 0.00301 Preprocess~
4    11          5     0.0609 roc_auc binary     0.648     5 0.00210 Preprocess~
5    14          5     0.0127 roc_auc binary     0.647     5 0.00204 Preprocess~
```

Review tuning results and select the best model.

```r
# review tuning results
xgb_res %>% show_best( metric = "roc_auc")
```

```
# A tibble: 5 x 9
  min_n tree_depth learn_rate .metric .estimator  mean     n std_err .config
  <int>      <int>      <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
1    17          3     0.0583 roc_auc binary     0.650     5 0.00206 Preprocess~
2    14          7     0.0114 roc_auc binary     0.650     5 0.00189 Preprocess~
3     7          4     0.0760 roc_auc binary     0.648     5 0.00301 Preprocess~
4    11          5     0.0609 roc_auc binary     0.648     5 0.00210 Preprocess~
5    14          5     0.0127 roc_auc binary     0.647     5 0.00204 Preprocess~
```

```r
# select the best model
best_xgb <- xgb_res %>% select_best( metric = "roc_auc")
best_xgb
```

```
# A tibble: 1 x 4
  min_n tree_depth learn_rate .config
  <int>      <int>      <dbl> <chr>
1    17          3     0.0583 Preprocessor1_Model04
```

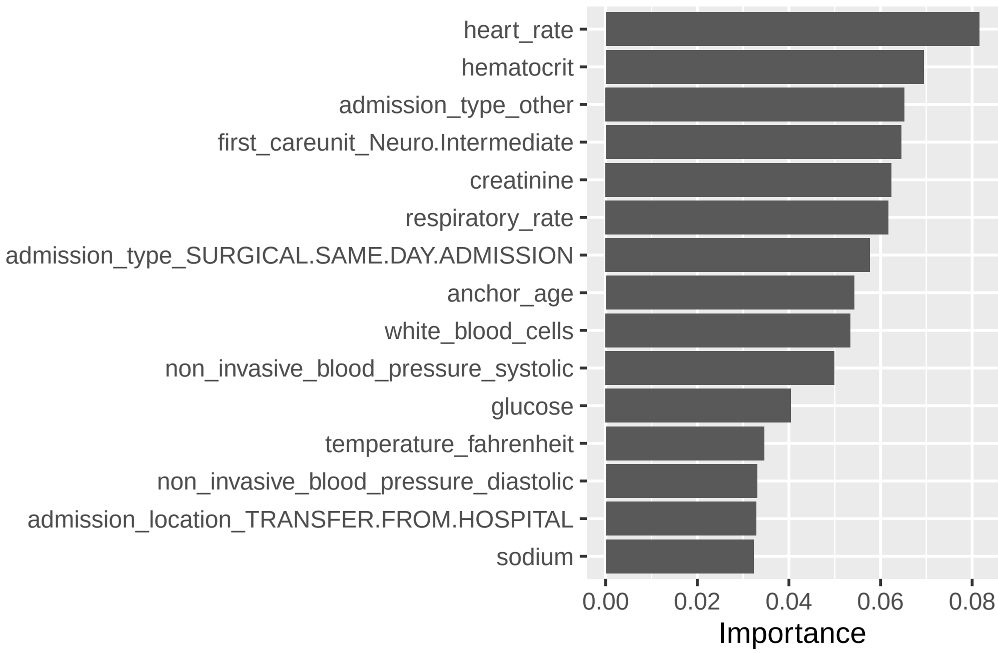
Evaluate the results by best parameters.

```r
#  workflow
final_xgb_workflow <- xgb_workflow %>% finalize_workflow(best_xgb)

#
final_xgb_fit <- final_xgb_workflow %>% last_fit(data_split)

#
final_xgb_fit %>% collect_metrics()
```

```
# A tibble: 3 x 4
  .metric     .estimator .estimate .config
  <chr>       <chr>          <dbl> <chr>
1 accuracy    binary         0.610 Preprocessor1_Model1
2 roc_auc     binary         0.649 Preprocessor1_Model1
3 brier_class binary         0.232 Preprocessor1_Model1
```

Feature Importance Analysis for the Model

```
final_xgb_fit %>%
  extract_fit_parsnip() %>%
  vip(num_features = 15)
```



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?

Model Performance Evaluation on the Test Set

Evaluating and Comparing Model Performance

```
# Extract the predictions of each model on the test set
lr_preds <- collect_predictions(final_fit_fine)
rf_preds <- collect_predictions(final_rf_fit)
xgb_preds <- collect_predictions(final_xgb_fit)

# calculate ROC AUC & Accuracy
model_metrics <- bind_rows(
  lr_preds %>% mutate(model = "Logistic Regression"),
```

```r
  rf_preds %>% mutate(model = "Random Forest"),
  xgb_preds %>% mutate(model = "XGBoost")
) %>%
  group_by(model) %>%
  summarise(
    accuracy = accuracy_vec(truth = los_long, estimate = .pred_class),
    roc_auc = roc_auc_vec(truth = los_long, estimate = .pred_Yes)
  )

# show the results
model_metrics
```

```
# A tibble: 3 x 3
  model               accuracy roc_auc
  <chr>                  <dbl>   <dbl>
1 Logistic Regression    0.558   0.596
2 Random Forest          0.609   0.650
3 XGBoost                0.610   0.649
```

```r
# Define all final workflow parameters
final_rf_workflow <- rf_workflow %>% finalize_workflow(best_rf)
final_xgb_workflow <- xgb_workflow %>% finalize_workflow(best_xgb)
final_log_workflow <- log_reg_workflow %>% finalize_workflow(best_fine)

# run the final models
final_rf_fit <- final_rf_workflow %>% last_fit(data_split)
final_xgb_fit <- final_xgb_workflow %>% last_fit(data_split)
final_log_fit <- final_log_workflow %>% last_fit(data_split)

# Perform the standard evaluation of the results
final_rf_fit %>% collect_metrics()
```

```
# A tibble: 3 x 4
  .metric     .estimator .estimate .config
  <chr>       <chr>          <dbl> <chr>
1 accuracy    binary         0.606 Preprocessor1_Model1
2 roc_auc     binary         0.649 Preprocessor1_Model1
3 brier_class binary         0.233 Preprocessor1_Model1
```

```r
final_xgb_fit %>% collect_metrics()
```

```
# A tibble: 3 x 4
  .metric     .estimator .estimate .config
  <chr>       <chr>          <dbl> <chr>
1 accuracy    binary         0.610 Preprocessor1_Model1
2 roc_auc     binary         0.649 Preprocessor1_Model1
3 brier_class binary         0.232 Preprocessor1_Model1
```

```r
final_log_fit %>% collect_metrics()
```

```
# A tibble: 3 x 4
  .metric     .estimator .estimate .config
  <chr>       <chr>          <dbl> <chr>
1 accuracy    binary         0.558 Preprocessor1_Model1
2 roc_auc     binary         0.596 Preprocessor1_Model1
3 brier_class binary         0.246 Preprocessor1_Model1
```

Model Stacking Code

```r
set.seed(123)

# define the stacking model
log_reg_res <- final_log_workflow %>%
  fit_resamples(
    resamples = folds,
    control = control_stack_resamples()
  )

rf_res <- final_rf_workflow %>%
  fit_resamples(
    resamples = folds,
    control = control_stack_resamples()
  )

xgb_res <- final_xgb_workflow %>%
  fit_resamples(
    resamples = folds,
    control = control_stack_resamples()
  )
```

```r
# Develop stacking models
model_stack <- stacks() %>%
  add_candidates(log_reg_res) %>%
  add_candidates(rf_res) %>%
  add_candidates(xgb_res) %>%
  blend_predictions() %>%
  fit_members()

stack_preds <- predict(model_stack, test_data, type = "prob") %>%
  bind_cols(test_data) %>%
  mutate(.pred_class = factor(ifelse(.pred_Yes > 0.5, "Yes", "No"),
                              levels = c("Yes", "No")))

roc_auc(stack_preds, truth = los_long, .pred_Yes)
```

```
# A tibble: 1 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 roc_auc binary         0.654
```

```r
accuracy(stack_preds, truth = los_long, .pred_class)
```

```
# A tibble: 1 x 3
  .metric  .estimator .estimate
  <chr>    <chr>          <dbl>
1 accuracy binary         0.611
```

Model Performance Evaluation (ROC & Accuracy)

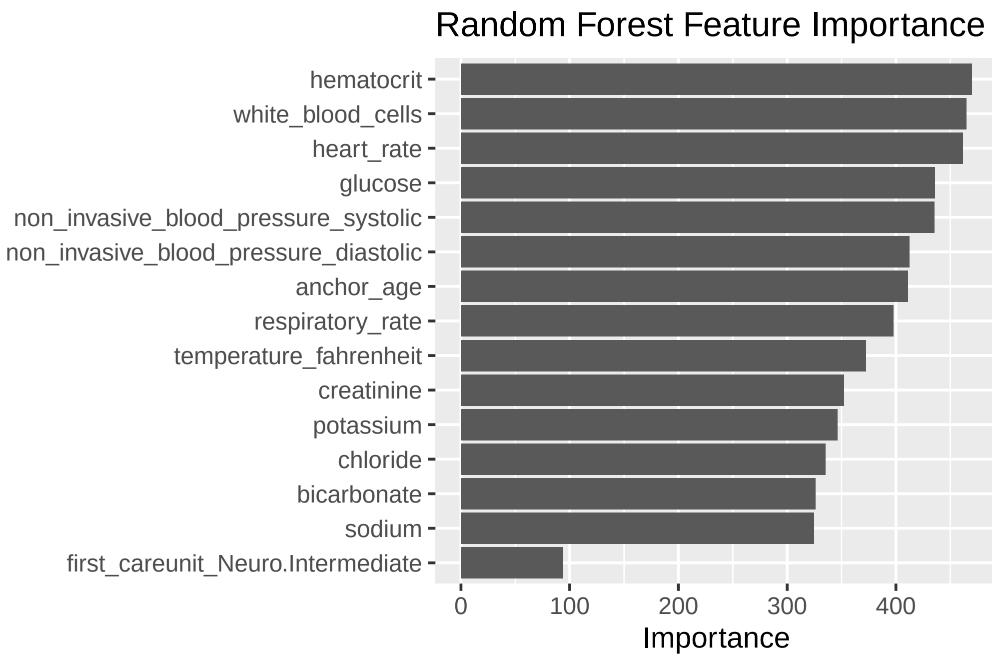
```r
# Extract Model Performance
log_metrics <- final_log_fit %>%
  collect_metrics() %>%
  select(.metric, .estimate)

rf_metrics  <- final_rf_fit  %>%
  collect_metrics() %>%
  select(.metric, .estimate)

xgb_metrics <- final_xgb_fit %>%
  collect_metrics() %>%
```

```r
  select(.metric, .estimate)

# Stacking Model Performance
stack_roc <- roc_auc(stack_preds, truth = los_long, .pred_Yes)
stack_acc <- accuracy(stack_preds, truth = los_long, .pred_class)

# Show Combined Results
results <- tibble(
  Model = c("Logistic Regression", "Random Forest", "XGBoost", "Stacking"),
  ROC_AUC = c(log_metrics$.estimate[log_metrics$.metric=="roc_auc"],
              rf_metrics$.estimate[rf_metrics$.metric=="roc_auc"],
              xgb_metrics$.estimate[xgb_metrics$.metric=="roc_auc"],
              stack_roc$.estimate),
  Accuracy = c(log_metrics$.estimate[log_metrics$.metric=="accuracy"],
               rf_metrics$.estimate[rf_metrics$.metric=="accuracy"],
               xgb_metrics$.estimate[xgb_metrics$.metric=="accuracy"],
               stack_acc$.estimate)
)

results
```

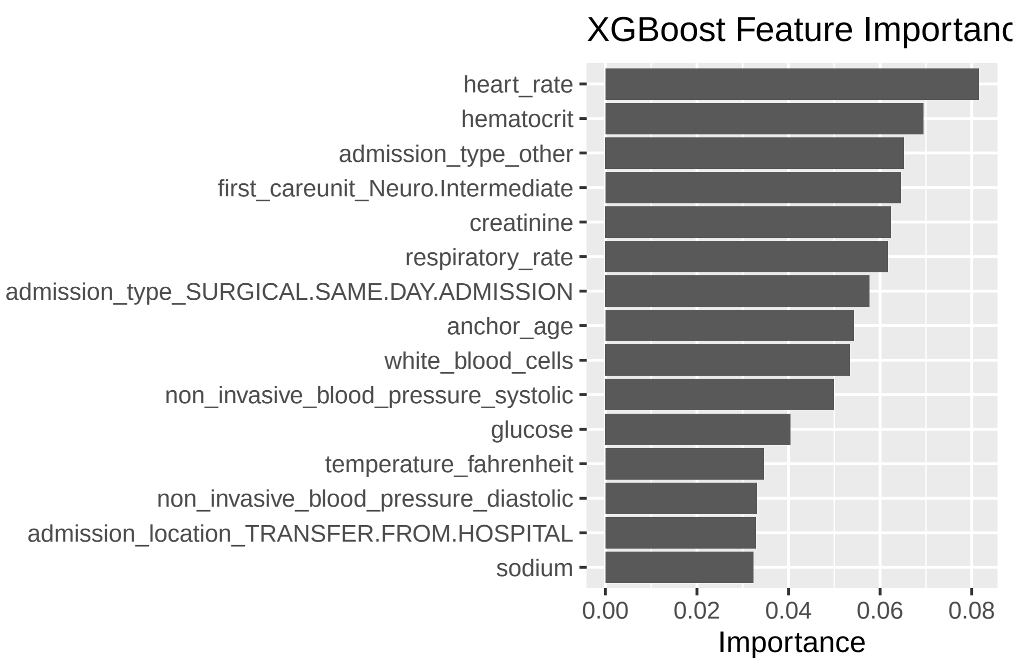
```
# A tibble: 4 x 3
  Model               ROC_AUC Accuracy
  <chr>                 <dbl>    <dbl>
1 Logistic Regression   0.596    0.558
2 Random Forest         0.649    0.606
3 XGBoost               0.649    0.610
4 Stacking              0.654    0.611
```

Model Feature Importance Analysis

```r
# Random Forest Feature Importance
final_rf_fit %>%
  extract_fit_parsnip() %>%
  vip(num_features = 15) +
  ggtitle("Random Forest Feature Importance")
```




```r
# XGBoost Feature Importance
final_xgb_fit %>%
  extract_fit_parsnip() %>%
  vip(num_features = 15) +
  ggtitle("XGBoost Feature Importance")
```



Model Performance Comparison (ROC Curve Plot)

```
# Extract Prediction Probabilities
log_preds <- final_log_fit %>% collect_predictions()
rf_preds  <- final_rf_fit %>% collect_predictions()
xgb_preds <- final_xgb_fit %>% collect_predictions()

# ROC Curve
log_roc <- roc(log_preds$los_long, log_preds$.pred_Yes)
```

```
Setting levels: control = Yes, case = No

Setting direction: controls > cases
```

```
rf_roc  <- roc(rf_preds$los_long, rf_preds$.pred_Yes)
```

```
Setting levels: control = Yes, case = No
Setting direction: controls > cases
```

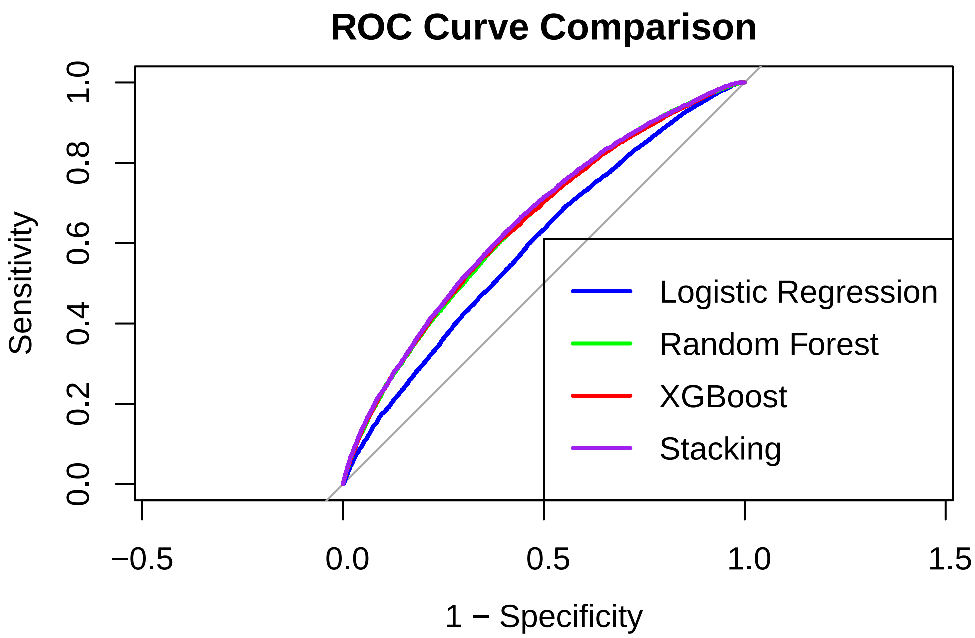
```
xgb_roc <- roc(xgb_preds$los_long, xgb_preds$.pred_Yes)
```

```
Setting levels: control = Yes, case = No
Setting direction: controls > cases
```

```
stack_roc_curve <- roc(stack_preds$los_long, stack_preds$.pred_Yes)
```

```
Setting levels: control = Yes, case = No
Setting direction: controls > cases
```

```
plot(log_roc, col = "blue", legacy.axes=TRUE, main="ROC Curve Comparison")
plot(rf_roc, col = "green", add = TRUE)
plot(xgb_roc, col = "red", add = TRUE)
plot(stack_roc_curve, col = "purple", add = TRUE)

legend("bottomright",
       legend=c("Logistic Regression","Random Forest","XGBoost","Stacking"),
       col=c("blue","green","red","purple"), lwd=2)
```



Mode	Roc_Auc	Accuracy
Logistic Regression	0.5958	0.5576
Random Forest	0.6489	0.6062
XGBoost	0.6488	0.6099
Stacking	0.6538	0.6107

The stacking model slightly outperforms the individual models in terms of ROC-AUC (0.6538), followed closely by XGBoost (0.6488) and Random Forest (0.6489). Logistic regression has significantly lower performance (0.5958), indicating its limited predictive capability for this dataset.

Similar trends emerge in accuracy, with stacking achieving the highest accuracy (0.6107). However, accuracy differences between stacking, XGBoost, and Random Forest are relatively small, indicating that the stacking method's improvement, although statistically measurable, is practically modest.