Biostat 203B Homework 5

Due Mar 20 @ 11:59PM

Your Name and UID

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.

Solution:

Display machine information:

```
sessionInfo()
```

```
R version 4.4.2 (2024-10-31)
Platform: x86_64-pc-linux-gnu
Running under: Ubuntu 24.04.1 LTS

Matrix products: default
BLAS:   /usr/lib/x86_64-linux-gnu/blas/libblas.so.3.12.0
LAPACK: /usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.12.0

locale:
 [1] LC_CTYPE=C.UTF-8       LC_NUMERIC=C           LC_TIME=C.UTF-8
```

```
 [4] LC_COLLATE=C.UTF-8     LC_MONETARY=C.UTF-8    LC_MESSAGES=C.UTF-8
 [7] LC_PAPER=C.UTF-8       LC_NAME=C              LC_ADDRESS=C
[10] LC_TELEPHONE=C         LC_MEASUREMENT=C.UTF-8 LC_IDENTIFICATION=C

time zone: America/Los_Angeles
tzcode source: system (glibc)

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

loaded via a namespace (and not attached):
 [1] compiler_4.4.2    fastmap_1.2.0     cli_3.6.4         tools_4.4.2
 [5] htmltools_0.5.8.1 rstudioapi_0.17.1 yaml_2.3.10       rmarkdown_2.29
 [9] knitr_1.49        jsonlite_1.9.1    xfun_0.51         digest_0.6.37
[13] rlang_1.1.5       evaluate_1.0.3
```

Display my machine memory.

```
memuse::Sys.meminfo()
```

```
Totalram:  11.686 GiB
Freeram:    5.050 GiB
```

Load database libraries and the tidyverse frontend:

```
library(GGally)
```

```
Loading required package: ggplot2

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

```
library(gtsummary)
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 becor
```

```
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      1.3.1     v yardstick    1.3.2
v recipes      1.2.0
-- 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()
```

```
library(stacks)
library(vip)
```

```
Attaching package: 'vip'

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

    vi
```

To predict ICU stay duration, we use the last lab measurements before the ICU stay as features.

```
#Reading in the RDS object
mimiciv_icu_cohort <- readRDS("../hw4/mimiciv_shiny/mimic_icu_cohort.rds")

#We want to use the last lab measurements before the ICU stay as features
#(chart events) to predict whether an ICU stay will be long or short
```

```r
#We also want to keep subject_id, hadm_id, and stay_id to sort the values

non_predicting_features <- c("subject_id",
                             "hadm_id",
                             "stay_id")

predicting_features <- c("hematocrit",
                             "sodium",
                             "bicarbonate",
                             "chloride",
                             "wbc",
                             "creatinine",
                             "glucose",
                             "potassium")

outcome_feature <- c("los_long")


#First, we create our los_long variable, as in homework 4
#Then, we select the dataframe only for our variables of interest

#For preprocessing:
#We remove patients with an NA los_long variable, since we cannot predict with
#no outcome variable
#We saw in homework 4 that some last lab measurements had some biologically
#impossiblely large measurments.
#We remove these outliers from that data set.
#Once the outliers are removed, our imputation is more robust, allowing us to
#imputate with the mean in our workflow

mimiciv_icu_cohort <- mimiciv_icu_cohort %>%
  mutate(los_long = (los >= 2)) %>%
  #Convert los_long to a factor
  mutate(los_long = as.factor(los_long)) %>%
  select(all_of(c(non_predicting_features,
           predicting_features,
           outcome_feature))) %>%
  filter(!is.na(los_long)) %>%
  #To filter out outliers, we use boxplot.stats()
  #The function returns a dataframe with an out (outliers) column
  #This column contains values considered as outliers
  #We filter out values determined as an outlier
```

```
  filter(!hematocrit %in% boxplot.stats(hematocrit)$out) %>%
  filter(!sodium %in% boxplot.stats(sodium)$out) %>%
  filter(!bicarbonate %in% boxplot.stats(bicarbonate)$out) %>%
  filter(!chloride %in% boxplot.stats(chloride)$out) %>%
  filter(!wbc %in% boxplot.stats(wbc)$out) %>%
  filter(!creatinine %in% boxplot.stats(creatinine)$out) %>%
  filter(!glucose %in% boxplot.stats(glucose)$out) %>%
  filter(!potassium %in% boxplot.stats(potassium)$out)

#Ensuring the outcome feature is balanced
table(mimiciv_icu_cohort$los_long)
```

```
FALSE  TRUE
35926 32679
```

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
  )
```

Solution:

```
set.seed(203)

# sort
mimiciv_icu_cohort <- mimiciv_icu_cohort %>%
  arrange(subject_id, hadm_id, stay_id) %>%
  #After sorting, we can remove the features we sort by to simplify the
```

```
  #dataframe
  select(all_of(c(predicting_features,
           outcome_feature)))

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

#Creating dataframes of the training and test splits
los_other <- training(data_split)
los_test <- testing(data_split)
```

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

Solution:

```
#Logistic Regression
#Using code from the lecture notes, defining the logistic regression recipe
logit_recipe <-
  recipe(
    los_long ~ .,
    data = mimiciv_icu_cohort
  ) %>%
  # Mean imputation for all lab events
  step_impute_mean(hematocrit) %>%
  step_impute_mean(sodium) %>%
  step_impute_mean(bicarbonate) %>%
  step_impute_mean(chloride) %>%
  step_impute_mean(wbc) %>%
  step_impute_mean(creatinine) %>%
  step_impute_mean(glucose) %>%
  step_impute_mean(potassium) %>%
  # zero-variance filter
  step_zv(all_numeric_predictors()) %>%
  # center and scale numeric data
  step_normalize(all_numeric_predictors())

#Creating the logistic regression model with the parameters we want to tune
logit_mod <-
```

```
  logistic_reg(
    penalty = tune(),
    mixture = tune()
  ) |>
  set_engine("glmnet", standardize = FALSE)

#Creating the logistic regression workflow with the recipe and model
logit_wf <- workflow() |>
  add_recipe(logit_recipe) |>
  add_model(logit_mod)

#Defining the parameter grid we want to try in 5-fold CV
#Before simplifying the parameter grid, the original parameter grid is as follows:
# param_grid <- grid_regular(
#   penalty(range = c(-8, 3)),
#   mixture(),
#   levels = c(100, 5)
#   )
param_grid_logit <- grid_regular(
  penalty(range = c(-8, -8)),
  mixture(range = c(0, 0)),
  levels = c(1, 1)
  )

#Setting seed for 5-fold CV
set.seed(203)

#Creating the 5-folds used for all 3 models
folds <- vfold_cv(mimiciv_icu_cohort, v = 5)

#Fitting the logistic regression model with the tunable parameters with the
#metrics of roc/auc and accuracy
#Setting the seed to ensure reproducibility of the logistic regression model.
set.seed(203)
logit_fit <- logit_wf |>
  tune_grid(
    resamples = folds,
    control = control_resamples(save_pred = TRUE, save_workflow = TRUE),
    grid = param_grid_logit,
    metrics = metric_set(roc_auc, accuracy)
    )
#Selecting the best model as the one with the highest roc/auc
```

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

#Creating the final workflow with the best logistic regression model
final_wf_logit <- logit_wf |>
  finalize_workflow(best_logit)

#Fit the best model with the training data, then test it on the test data
final_fit_logit <-
  final_wf_logit |>
  last_fit(data_split)

#Collecting the metrics of the best logistic regression model.
final_fit_logit |>
  collect_metrics()
```

```
# A tibble: 3 x 4
  .metric     .estimator .estimate .config
  <chr>       <chr>          <dbl> <chr>
1 accuracy    binary         0.531 Preprocessor1_Model1
2 roc_auc     binary         0.550 Preprocessor1_Model1
3 brier_class binary         0.248 Preprocessor1_Model1
```

```r
#XGBoost
#Using code from the lecture notes, defining the XGBoost recipe
gb_recipe <-
  recipe(
    los_long ~ .,
    data = mimiciv_icu_cohort
  ) %>%
  # Mean imputation for all lab events
  step_impute_mean(hematocrit) %>%
  step_impute_mean(sodium) %>%
  step_impute_mean(bicarbonate) %>%
  step_impute_mean(chloride) %>%
  step_impute_mean(wbc) %>%
  step_impute_mean(creatinine) %>%
  step_impute_mean(glucose) %>%
  step_impute_mean(potassium) %>%
  # zero-variance filter
  step_zv(all_numeric_predictors()) %>%
```

```
  # center and scale numeric data
  step_normalize(all_numeric_predictors())

#Creating the XGBoost model with the parameters we want to tune
gb_mod <-
  boost_tree(
    mode = "classification",
    trees = 1000,
    tree_depth = tune(),
    learn_rate = tune()
  ) |>
  set_engine("xgboost")

#Creating the XGBoost workflow with the recipe and model
gb_wf <- workflow() |>
  add_recipe(gb_recipe) |>
  add_model(gb_mod)

#Defining the parameter grid we want to try in 5-fold CV
#Before simplifying the parameter grid, the original parameter grid is as follows:
#param_grid <- grid_regular(
#   tree_depth(range = c(1L, 5L)),
#   learn_rate(range = c(-5, 2), trans = log10_trans()),
#   levels = c(3, 10)
#   )

param_grid_gb <- grid_regular(
  tree_depth(range = c(4L, 4L)),
  learn_rate(range = c(-2,-2), trans = log10_trans()),
  levels = c(1, 1)
  )

#Fitting the XGBoost model with the tunable parameters with the
#metrics of roc/auc and accuracy
#Setting the seed to ensure reproducibility of the XGBoost model.
set.seed(203)
gb_fit <- gb_wf |>
  tune_grid(
    resamples = folds,
    control = control_resamples(save_pred = TRUE, save_workflow = TRUE),
    grid = param_grid_gb,
    metrics = metric_set(roc_auc, accuracy)
```

```
    )

#Selecting the best model as the one with the highest roc/auc
best_gb <- gb_fit |>
  select_best(metric = "roc_auc")

#Creating the final workflow with the best XGBoost model
final_wf_gb <- gb_wf |>
  finalize_workflow(best_gb)

#Fit the best model with the training data, then test it on the test data
final_fit_gb <-
  final_wf_gb |>
  last_fit(data_split)

#Collecting the metrics of the best XGBoost model.
final_fit_gb |>
  collect_metrics()
```

```
# A tibble: 3 x 4
  .metric     .estimator .estimate .config
  <chr>       <chr>          <dbl> <chr>
1 accuracy    binary         0.553 Preprocessor1_Model1
2 roc_auc     binary         0.571 Preprocessor1_Model1
3 brier_class binary         0.246 Preprocessor1_Model1
```

```
#RF
#Using code from the lecture notes, defining the RF recipe
rf_recipe <-
  recipe(
    los_long ~ .,
    data = mimiciv_icu_cohort
  ) %>%
  # Mean imputation for all lab events
  step_impute_mean(hematocrit) %>%
  step_impute_mean(sodium) %>%
  step_impute_mean(bicarbonate) %>%
  step_impute_mean(chloride) %>%
  step_impute_mean(wbc) %>%
  step_impute_mean(creatinine) %>%
  step_impute_mean(glucose) %>%
```

```
  step_impute_mean(potassium) %>%
  # zero-variance filter
  step_zv(all_numeric_predictors()) %>%
  # center and scale numeric data
  step_normalize(all_numeric_predictors())

#Creating the RF model with the parameters we want to tune
rf_mod <-
  rand_forest(
    mode = "classification",
    # Number of predictors randomly sampled in each split
    mtry = tune(),
    # Number of trees in ensemble
    trees = tune()
  ) |>
  set_engine("ranger", importance = "impurity")

#Creating the rf workflow with the recipe and model
rf_wf <- workflow() |>
  add_recipe(rf_recipe) |>
  add_model(rf_mod)

#Defining the parameter grid we want to try in 5-fold CV
#Before simplifying the parameter grid, the original parameter grid is as follows:
# param_grid_rf <- grid_regular(
#   trees(range = c(100L, 500L)),
#   mtry(range = c(1L, 5L)),
#   levels = c(5, 5)
#   )
param_grid_rf <- grid_regular(
  trees(range = c(500L, 500L)),
  mtry(range = c(1L, 1L)),
  levels = c(1, 1)
  )

#Fitting the RF model with the tunable parameters with the
#metrics of roc/auc and accuracy
#Setting the seed to ensure reproducibility of the RF model.
set.seed(203)
rf_fit <- rf_wf |>
  tune_grid(
    resamples = folds,
```

```
    control = control_resamples(save_pred = TRUE, save_workflow = TRUE),
    grid = param_grid_rf,
    metrics = metric_set(roc_auc, accuracy)
    )

#Selecting the best model as the one with the highest roc/auc
best_rf <- rf_fit |>
  select_best(metric = "roc_auc")

#Creating the final workflow with the best RF model
final_wf_rf <- rf_wf |>
  finalize_workflow(best_rf)

#Fit the best model with the training data, then test it on the test data
final_fit_rf <-
  final_wf_rf |>
  last_fit(data_split)

#Collecting the metrics of the best RF model.
final_fit_rf |>
  collect_metrics()
```

```
# A tibble: 3 x 4
  .metric     .estimator .estimate .config
  <chr>       <chr>          <dbl> <chr>
1 accuracy    binary         0.548 Preprocessor1_Model1
2 roc_auc     binary         0.563 Preprocessor1_Model1
3 brier_class binary         0.247 Preprocessor1_Model1
```

```
#Model Stacking

#Using code from the lecture notes, defining the stack
los_model_st <-
  # initialize the stack
  stacks() |>
  # add candidate members, we use the best fitting models for computational
  # efficiency
  add_candidates(logit_fit) |>
  add_candidates(gb_fit) |>
  add_candidates(rf_fit) |>
  # determine how to combine their predictions
```

```
  blend_predictions(
    penalty = 10^(-6:2),
    metrics = c("roc_auc")
    ) |>
  # fit the candidates with nonzero stacking coefficients
  fit_members()
```

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

```
#Predicting los_long with the stacked models
#Setting the seed to ensure reproducibility of the stacked model.
set.seed(203)

#Outputting the predictions of the stacked model as a probability
los_pred <- los_test %>%
  bind_cols(predict(los_model_st, ., type = "prob"))

#Calculating ROC of model stack
yardstick::roc_auc(
  los_pred,
  truth = los_long,
  contains(".pred_FALSE")
  )
```

```
# A tibble: 1 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 roc_auc binary         0.820
```

```
#Generating predictions from the ensemble members
#Setting the seed to ensure reproducibility of the stacked model.
set.seed(203)
los_pred <-
  los_test |>
  select(los_long) |>
  bind_cols(
    predict(
      los_model_st,
      los_test,
      type = "class",
```

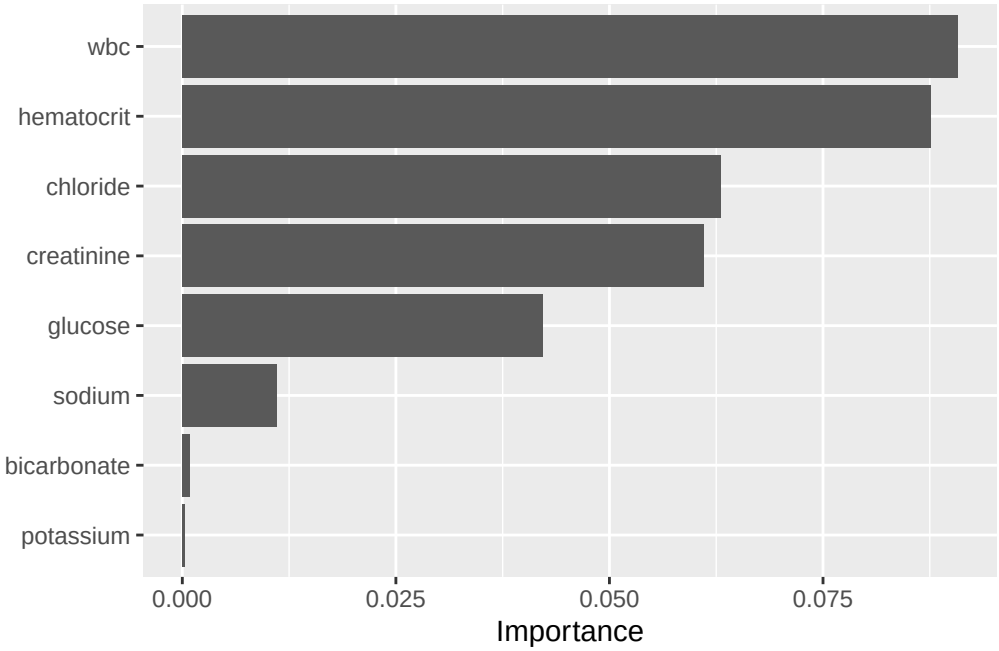
```
      members = TRUE
      )
    )

#Calculating accuracy of model stack
#We compute the proportion of predictions from the ensemble members that
#agree with the actual labels
map(
  colnames(los_pred),
  ~mean(los_pred$los_long  == pull(los_pred, .x))
  ) |>
  set_names(colnames(los_pred)) |>
  as_tibble() |>
  pivot_longer(c(everything(), -los_long))
```

```
# A tibble: 3 x 3
  los_long name                   value
     <dbl> <chr>                  <dbl>
1        1 .pred_class            0.751
2        1 .pred_class_gb_fit_1_1 0.576
3        1 .pred_class_rf_fit_1_1 0.927
```

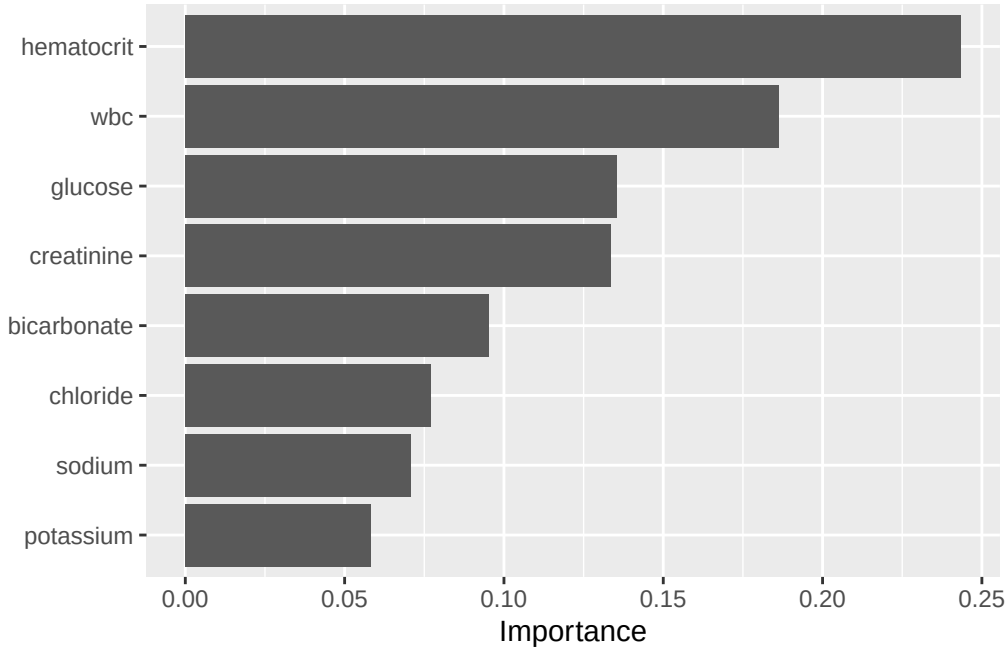
4. Compare model classification performance on the test set. Report both the area under ROC curve and accuracy for each machine learning algorithm and the model stacking. Interpret the results. What are the most important features in predicting long ICU stays? How do the models compare in terms of performance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

Solution:

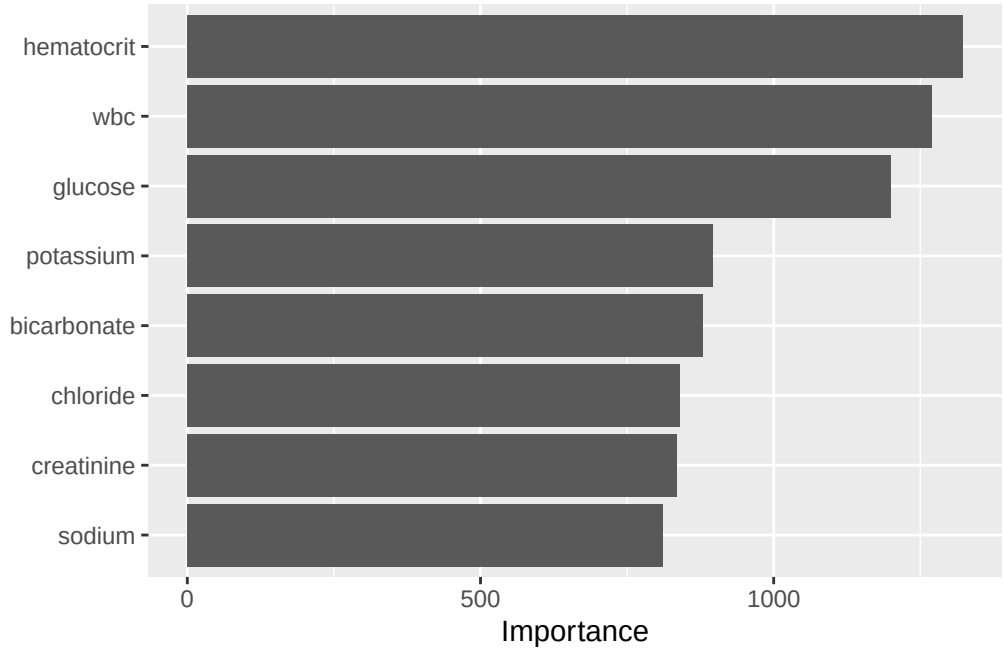
```
#Determining the most important features for predicting long ICU stays

#Logistic Regression
final_fit_logit %>%
  extract_fit_parsnip() %>%
  vip(num_features = 10)
```




```
#XGBoost
final_fit_gb %>%
  extract_fit_parsnip() %>%
  vip(num_features = 10)
```




```
#RF
final_fit_rf %>%
  extract_fit_parsnip() %>%
  vip(num_features = 10)
```



The logistic regression model's area under the curve (AUC) and accuracy are 0.550 and 0.531, respectively. The XGBoost model's area under the curve (AUC) and accuracy are 0.571 and 0.553, respectively. The RF model's area under the curve (AUC) and accuracy are 0.563 and 0.548, respectively. The stacked model's area under the curve (AUC) and accuracy are 0.820 and 0.751, respectively.

Based on these results, the best-performing individual model is the gradient-boosted model, while the best-performing model overall is the stacked model. The individual models did not perform very well because all three models failed to exceed AUCs and accuracies of 0.60. The gradient-boosted model most likely performed the best because it is a more complex model, using gradient descent to optimize the sequential building and corrections of weak learners in the model. On the other hand, logistic regression is a relatively simple model and can not handle nonlinear relationships. Similarly, random forests rely on the chance of independent tree ensembles and do not build on the mistakes of others.

The stacked model performed much better and saw increases of up to 0.27 in AUC and 22% in accuracy. This performance is expected of stacked models since we can utilize and draw on the strengths of different model architectures instead of relegating ourselves to the weakness of using a singular model. However, for making medical predictions, we still would want to aim for higher AUCs and accuracy, preferably far exceeding 0.90, since the ramifications of making incorrect medical predictions could cost lives. To do this, we want to try to use more advanced models such as neural networks (MLP, RNN, LSTM, Transformers, etc.), perform a more extensive grid search of acceptable parameters, or add/use a different set of features such as demographic information/chart events.

Using `vip()` to determine important features in predicting long ICU stays, we see that hematocrit and white blood cell levels are consistently the most important features across the three models (logistic regression, XGBoost, and random forest). A medical explanation could be that more serious diseases generally affect the blood, such as leukemia, HIV, etc., and require prolonged ICU stays to treat. The logistic model seems the most interpretable, as only five features have significant importance, while the XGBoost and RF models designate significant importance to all features.