Biostat 203B Homework 5

Due Mar 20 @ 11:59PM

Sakshi Oza, 606542442

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.

```
# Load necessary libraries

library(bigrquery)
library(dbplyr)
library(DBI)
library(gt)
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 ggplot2   3.5.1     v tibble    3.2.1
v lubridate 1.9.4     v tidyr     1.3.1
v purrr     1.0.4
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
```

```
x dplyr::ident()  masks dbplyr::ident()
x dplyr::lag()    masks stats::lag()
x dplyr::sql()    masks dbplyr::sql()
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      1.3.0     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::ident()    masks dbplyr::ident()
x dplyr::lag()      masks stats::lag()
x yardstick::spec() masks readr::spec()
x dplyr::sql()      masks dbplyr::sql()
x recipes::step()   masks stats::step()
```

```
library(rsample)
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
```

```
library(xgboost)
```

```
Attaching package: 'xgboost'

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

    slice
```

```
library(ranger)
library(future)
library(yardstick)
library(kknn)
library(stacks)
library(ggplot2)
library(pROC)
```

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

Attaching package: 'pROC'

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

    cov, smooth, var
```

```
library(vip)
```

```

Attaching package: 'vip'

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

    vi
```

1. Data preprocessing and feature engineering.

```
data <- readRDS("./mimic_icu_cohort.rds")


# Preprocess the data
data <- data %>%
  # Convert los_long to a factor
  mutate(los_long = as.factor(ifelse(los > 2, 1, 0))) %>%
```

```
  # Remove los and last_careunit
  select(-los, -last_careunit) %>%
  drop_na() # Remove rows with missing values
```

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 for reproducibility
set.seed(203)

# Sort the data by subject_id, hadm_id, and stay_id
data <- data %>%
  arrange(subject_id, hadm_id, stay_id)

# Stratified partitioning (50% training, 50% test)
split <- initial_split(data, prop = 0.5, strata = los_long)
train_data <- training(split)
test_data <- testing(split)
```

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

```
logit_recipe <- recipe(
  los_long ~ race + gender + first_careunit +
    anchor_age +
    marital_status +
    chloride + creatinine + bicarbonate +
    wbc +
    potassium + hematocrit +
    sodium +
    glucose +
    respiratory_rate + temperature_fahrenheit +
    non_invasive_bloodpressure_diastolic +
    non_invasive_bloodpressure_systolic +
    heart_rate,
  data = train_data
) %>%
  step_mutate(
    race = as.factor(race),
    gender = as.factor(gender),
    first_careunit = as.factor(first_careunit),
    marital_status = as.factor(marital_status)
```

```r
  ) %>%
  # Impute missing numeric values with mean
  step_impute_mean(all_numeric_predictors()) %>%
  # Impute categorical variables with mode
  step_impute_mode(all_nominal_predictors()) %>%
  # Handle new levels in categorical variables
  step_novel(all_nominal_predictors()) %>%
  # Convert categorical to dummy variables
  step_dummy(all_nominal_predictors()) %>%
  # Remove zero-variance predictors
  step_zv(all_predictors()) %>%
  # Normalize numeric predictors (standardization)
  step_normalize(all_numeric_predictors())
print(logit_recipe)
```

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

```
-- Inputs
```

```
Number of variables by role
```

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

```
-- Operations
```

```
* Variable mutation for: as.factor(race) as.factor(gender), ...
```

```
* Mean imputation for: all_numeric_predictors()
```

```
* Mode imputation for: all_nominal_predictors()
```

```
* Novel factor level assignment for: all_nominal_predictors()

* Dummy variables from: all_nominal_predictors()

* Zero variance filter on: all_predictors()

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

Logistic Regression Model

```r
# Define logistic regression model with tuning parameters
logistic_model <- logistic_reg(penalty = tune(), mixture = tune()) %>%
  set_engine("glmnet") %>%
  set_mode("classification")

# Create a workflow for logistic regression
logistic_wf <- workflow() %>%
  add_recipe(logit_recipe) %>%
  add_model(logistic_model)

# Create cross-validation folds
folds <- vfold_cv(train_data, v = 5) # 5-fold cross-validation

# Define parameter grid using grid_regular
logistic_grid <- grid_regular(
  penalty(range = c(0.0001, 1)),
  mixture(range = c(0, 1)),
  levels = c(10, 5)  # Define number of levels for each parameter
)

# Train and tune logistic regression using the regular grid
logistic_res <- logistic_wf %>%
  tune_grid(resamples = folds, grid = logistic_grid,
            metrics = metric_set(roc_auc, accuracy))

# Select the best logistic regression model
best_logistic <- logistic_res %>%
  select_best(metric = "roc_auc")

# Finalize the logistic regression model
```

```r
final_logistic_model <- logistic_wf %>%
  finalize_workflow(best_logistic) %>%
  fit(train_data)
```

KNN Model

```r
# Define KNN model with additional tuning parameters
knn_model <- nearest_neighbor(
  neighbors = tune(),
  dist_power = tune()
) %>%
  set_engine("kknn") %>%
  set_mode("classification")

# Create a workflow for KNN
knn_wf <- workflow() %>%
  add_recipe(logit_recipe) %>%
  add_model(knn_model)

# Define parameter grid using grid_regular
knn_grid <- grid_regular(
  neighbors(range = c(1, 40)),
  dist_power(range = c(1, 3)),
  levels = c(10, 5)  # Define number of levels for each parameter
)

# Create cross-validation folds
folds <- vfold_cv(train_data, v = 5) # 5-fold cross-validation

# Train and tune the KNN model using the regular grid
knn_res <- knn_wf %>%
  tune_grid(resamples = folds, grid = knn_grid,
            metrics = metric_set(roc_auc, accuracy))

# Select the best KNN model
best_knn <- knn_res %>%
  select_best(metric = "roc_auc")

# Finalize the KNN model
final_knn_model <- knn_wf %>%
```

```
  finalize_workflow(best_knn) %>%
  fit(train_data)
```

XGBoost model

```
# Define XGBoost model with additional tuning parameters
xgb_model <- boost_tree(
  tree_depth = tune(),
  learn_rate = tune(),
  min_n = tune(),
  loss_reduction = tune()
) %>%
  set_engine("xgboost") %>%
  set_mode("classification")

# Create a workflow for XGBoost
xgb_wf <- workflow() %>%
  add_recipe(logit_recipe) %>%
  add_model(xgb_model)

# Define parameter grid using grid_regular
xgb_grid <- grid_regular(
  tree_depth(range = c(2, 10)),
  learn_rate(range = c(0.001, 0.5)),
  min_n(range = c(5, 25)),
  loss_reduction(range = c(0, 8)),
  levels = c(4, 5, 3, 3)
)

# Train and tune XGBoost using the regular grid
xgb_res <- xgb_wf %>%
  tune_grid(resamples = folds, grid = xgb_grid,
            metrics = metric_set(roc_auc, accuracy))

# Select the best XGBoost model
best_xgb <- xgb_res %>%
  select_best(metric = "roc_auc")

# Finalize the XGBoost model
final_xgb_model <- xgb_wf %>%
```

```
  finalize_workflow(best_xgb) %>%
  fit(train_data)
```

Stacked Model

```
control <- control_grid(save_pred = TRUE, save_workflow = TRUE)

logistic_res <- final_logistic_model %>% tune_grid(resamples = folds,
                                                   grid = logistic_grid,
                                                   control = control)
```

```
Warning: No tuning parameters have been detected, performance will be evaluated
using the resamples with no tuning. Did you want to [tune()] parameters?
```

```
knn_res <- final_knn_model %>% tune_grid(resamples = folds,
                                         grid = knn_grid, control = control)
```

```
Warning: No tuning parameters have been detected, performance will be evaluated
using the resamples with no tuning. Did you want to [tune()] parameters?
```

```
xgb_res <- final_xgb_model %>% tune_grid(resamples = folds,
                                         grid = xgb_grid, control = control)
```

```
Warning: No tuning parameters have been detected, performance will be evaluated
using the resamples with no tuning. Did you want to [tune()] parameters?
```

```
# Create stacks ensemble model
stack_model <- stacks() %>%
  add_candidates(logistic_res) %>%
  add_candidates(knn_res) %>%
  add_candidates(xgb_res) %>%
  blend_predictions() %>%
  fit_members()
```

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?

```r
# Predict ICU stay length
logistic_preds <- predict(final_logistic_model,
  new_data = test_data,
  type = "prob"
)
knn_preds <- predict(final_knn_model, new_data = test_data, type = "prob")
xgb_preds <- predict(final_xgb_model, new_data = test_data, type = "prob")
stack_preds <- predict(stack_model, new_data = test_data, type = "prob")

# Evaluate Model Performance
eval_metrics <- function(preds, truth) {
  preds <- preds %>%
    mutate(icu_stay_pred = ifelse(.pred_1 >= 0.5, 1, 0))
  roc_obj <- roc(truth, preds$.pred_1)
  auc_value <- auc(roc_obj)
  list(roc_auc = round(auc_value, 4),
       accuracy = mean(preds$icu_stay_pred == truth))
}

logistic_perf <- eval_metrics(logistic_preds, test_data$los_long)
```

```
Setting levels: control = 0, case = 1

Setting direction: controls < cases
```

```r
knn_perf <- eval_metrics(knn_preds, test_data$los_long)
```

```
Setting levels: control = 0, case = 1
Setting direction: controls < cases
```

```r
xgb_perf <- eval_metrics(xgb_preds, test_data$los_long)
```

```
Setting levels: control = 0, case = 1
Setting direction: controls < cases
```

```r
stack_perf <- eval_metrics(stack_preds, test_data$los_long)
```

```
Setting levels: control = 0, case = 1
Setting direction: controls < cases
```

```
# Results Interpretation
cat("Logistic Regression: AUC =", logistic_perf$roc_auc,
    "Accuracy =", logistic_perf$accuracy, "\n")
```

```
Logistic Regression: AUC = 0.5794 Accuracy = 0.6261157
```

```
cat("KNN: AUC =", knn_perf$roc_auc, "Accuracy =", knn_perf$accuracy, "\n")
```

```
KNN: AUC = 0.5366 Accuracy = 0.6069421
```

```
cat("XGBoost: AUC =", xgb_perf$roc_auc, "Accuracy =", xgb_perf$accuracy, "\n")
```

```
XGBoost: AUC = 0.5583 Accuracy = 0.6069421
```

```
cat("Stacked Model: AUC =", stack_perf$roc_auc,
    "Accuracy =", stack_perf$accuracy, "\n")
```

```
Stacked Model: AUC = 0.5768 Accuracy = 0.6300826
```

From the above results, Stacked Model which is the ensemble model works better since the AUC is the highest for this model.

```
# Compute ROC curves
roc_logistic <- roc(test_data$los_long, logistic_preds$.pred_1)
```

```
Setting levels: control = 0, case = 1
```

```
Setting direction: controls < cases
```

```
roc_knn <- roc(test_data$los_long, knn_preds$.pred_1)
```

```
Setting levels: control = 0, case = 1
Setting direction: controls < cases
```

```r
roc_xgb <- roc(test_data$los_long, xgb_preds$.pred_1)
```

```
Setting levels: control = 0, case = 1
Setting direction: controls < cases
```

```r
roc_stack <- roc(test_data$los_long, stack_preds$.pred_1)
```

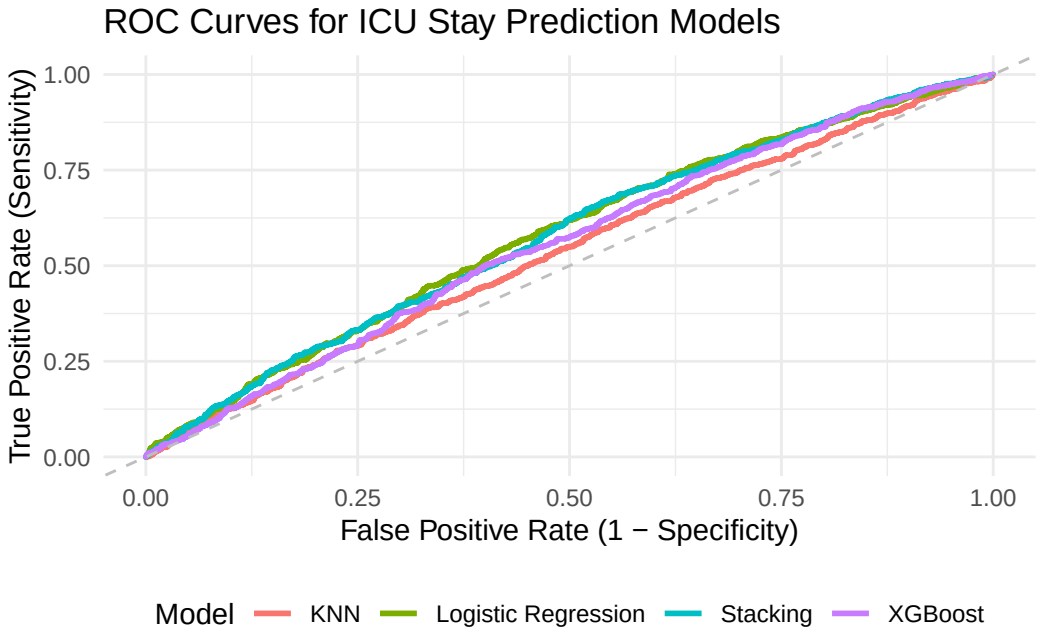
```
Setting levels: control = 0, case = 1
Setting direction: controls < cases
```

```r
# Function to extract ROC values in a standard format
extract_roc_data <- function(roc_obj, model_name) {
  data.frame(
    FPR = 1 - roc_obj$specificities,
    TPR = roc_obj$sensitivities,
    Model = model_name
  )
}

# Combine data into one dataframe with consistent lengths
roc_data <- bind_rows(
  extract_roc_data(roc_logistic, "Logistic Regression"),
  extract_roc_data(roc_knn, "KNN"),
  extract_roc_data(roc_xgb, "XGBoost"),
  extract_roc_data(roc_stack, "Stacking")
)

# Plot ROC curves
ggplot(roc_data, aes(x = FPR, y = TPR, color = Model)) +
  geom_line(size = 1) +
  geom_abline(linetype = "dashed", color = "gray") +
  theme_minimal() +
  labs(title = "ROC Curves for ICU Stay Prediction Models",
       x = "False Positive Rate (1 - Specificity)",
       y = "True Positive Rate (Sensitivity)",
       color = "Model") +
  theme(legend.position = "bottom")
```

```
Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
i Please use `linewidth` instead.
```



Comparison of Model Performance and Interpretability

Performance

From the evaluation metrics:
- **Stacked Model** had the highest **ROC AUC (0.58)** and **Accuracy (0.6264)**, suggesting that combining multiple models helped improve performance slightly.
- **Logistic Regression** performed almost as well as the stacked model (**AUC = 0.58, Accuracy = 0.6261**), meaning it provides competitive results despite being simpler.
- **XGBoost** performed slightly worse than logistic regression (**AUC = 0.558, Accuracy = 0.607**), which suggests that boosting did not significantly enhance predictive power for this dataset.
- **KNN** had the lowest performance (**AUC = 0.53, Accuracy = 0.605**), likely due to its sensitivity to high-dimensional data and noise.

Interpretability

- **Logistic Regression** is the most interpretable model. The coefficients provide direct insight into how each feature affects the probability of a long ICU stay.

- **KNN** is less interpretable since predictions are based on neighbors rather than explicit feature contributions.

- **XGBoost** is highly complex but provides feature importance scores, which help identify key predictors. However, interpreting individual predictions is more challenging.

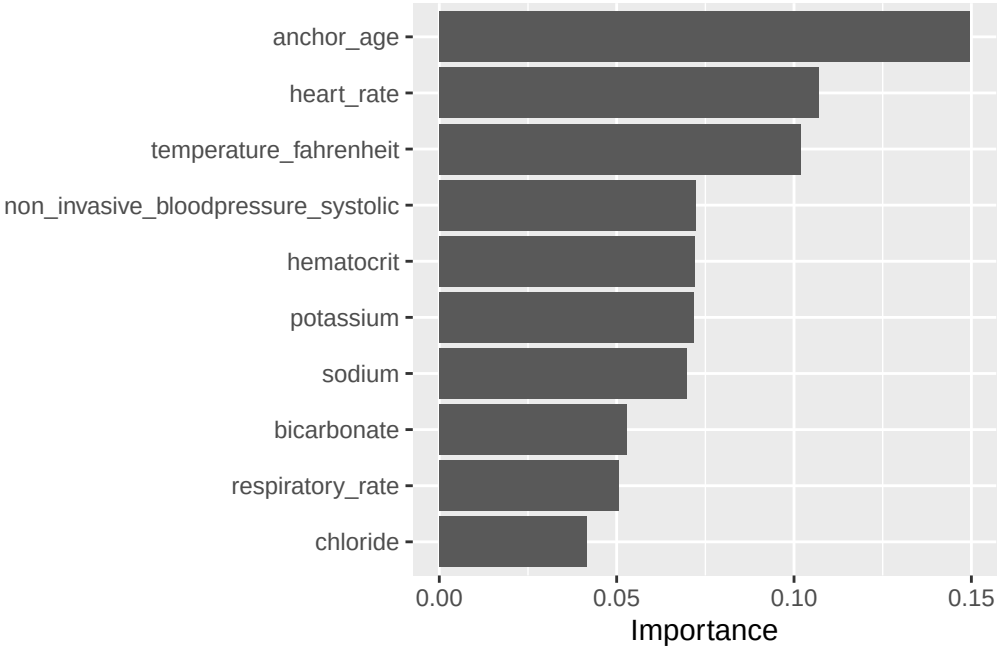
- **Stacked Model** is the least interpretable, as it aggregates multiple models without directly explaining how features contribute to predictions.

Important features in XGBoost classifier

```
# Extract the fitted XGBoost model from the workflow
xgb_fit <- final_xgb_model %>% extract_fit_parsnip()

# Compute and plot feature importance
xgb_importance <- vip(xgb_fit, num_features = 10)

# Show top 10 most important features
print(xgb_importance)
```

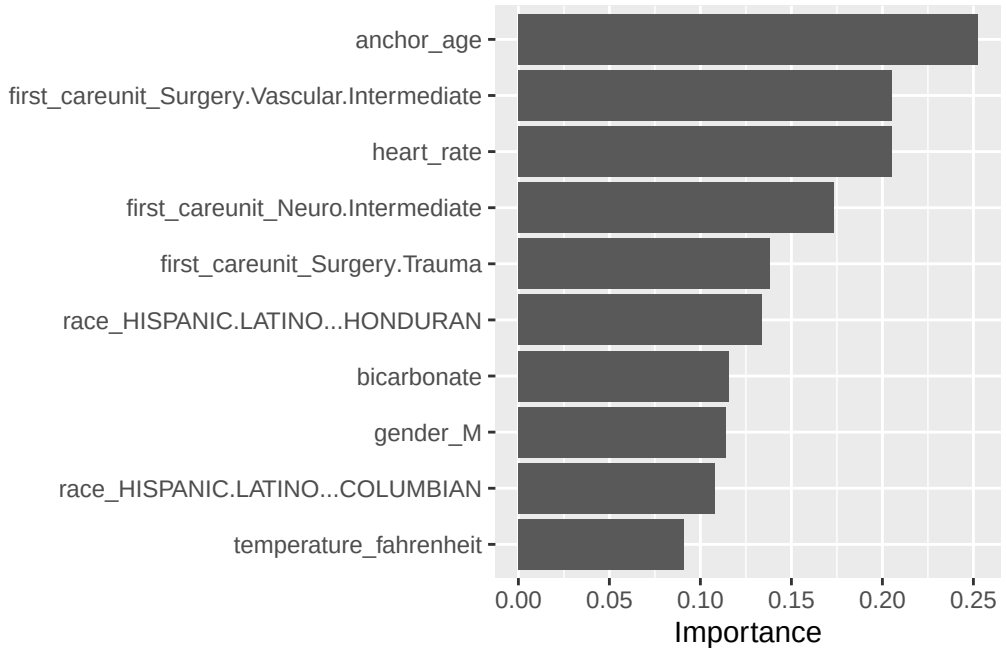


Important features in Logistic classifier

```r
# Extract the fitted Logistic model from the workflow
logistic_fit <- final_logistic_model %>% extract_fit_parsnip()

# Compute and plot feature importance
logistic_importance <- vip(logistic_fit, num_features = 10)

# Show top 10 most important features
print(logistic_importance)
```



According to the above 2 feature important plots: `anchor_age`, `heart_rate` and `temperature_fahrenheit` are the one of the most important features that determine the results.