Biostat 203B Homework 5

Due Mar 20 @ 11:59PM

Palash Raval and 406551574

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. Your 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.

```
library(dbplyr)
library(tidymodels)
```

```
-- Attaching packages ------------------------------------- tidymodels 1.2.0 --

v broom        1.0.7     v recipes      1.1.0
v dials        1.3.0     v rsample      1.2.1
v dplyr        1.1.4     v tibble       3.2.1
v ggplot2      3.5.1     v tidyr        1.3.1
v infer        1.0.7     v tune         1.2.1
v modeldata    1.4.0     v workflows    1.1.4
v parsnip      1.2.1     v workflowsets 1.1.0
v purrr        1.0.2     v yardstick    1.3.2

Warning: package 'broom' was built under R version 4.3.3

Warning: package 'dials' was built under R version 4.3.3
```

```
Warning: package 'modeldata' was built under R version 4.3.3

Warning: package 'recipes' was built under R version 4.3.3

Warning: package 'yardstick' was built under R version 4.3.3

-- Conflicts --------------------------------------- tidymodels_conflicts() --
x purrr::discard() masks scales::discard()
x dplyr::filter()  masks stats::filter()
x dplyr::ident()   masks dbplyr::ident()
x dplyr::lag()     masks stats::lag()
x dplyr::sql()     masks dbplyr::sql()
x recipes::step()  masks stats::step()
* Dig deeper into tidy modeling with R at https://www.tmwr.org
```

```
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::ident()      masks dbplyr::ident()
x dplyr::lag()        masks stats::lag()
x readr::spec()       masks yardstick::spec()
x dplyr::sql()        masks dbplyr::sql()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to becor
```

```
library(recipes)
library(ranger)
```

```
Warning: package 'ranger' was built under R version 4.3.3
```

```
library(stacks)
```

```
Warning: package 'stacks' was built under R version 4.3.3
```

1. Data preprocessing and feature engineering.

```
mimiciv_icu_cohort = readRDS("../hw4/mimiciv_shiny/mimic_icu_cohort.rds")
```

```
#removes the 14 rows with los_long being NA

mimiciv_icu_cohort = mimiciv_icu_cohort %>% filter(!is.na(los_long))

dim(mimiciv_icu_cohort)
```

```
[1] 94444    42
```

```
mimiciv_icu_cohort$los_long = ifelse(mimiciv_icu_cohort$los_long, "YES", "NO")
```

```
mimiciv_icu_cohort = mimiciv_icu_cohort %>%
  mutate(across(where(is.character), as.factor))
```

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)


# removing some features

mimiciv_icu_cohort = mimiciv_icu_cohort %>%
  select(c(-1, -2, -3, -5, -6, -7, -8, -9, -10, -11, -13, -15, -17, -20, -21, -22,
           -24, -25, -26, -27))


# splitting data

data_split = initial_split(
  mimiciv_icu_cohort,
```

```
  # stratify by los_long
  strata = "los_long",
  prop = 0.5
  )
```

```
training_data = training(data_split)
```

```
testing_data = testing(data_split)
```

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

```
#Checks how many NA values are in each column

#colSums(is.na(training_data))
```

Tuning and Training for Logistic Regression with Enet Regularization

```
logistic_recipe = recipe(data = training_data,
                         los_long ~ .) %>%
  step_impute_mean(potassium) %>%
  step_impute_mean(white_blood_cell_count) %>%
  step_impute_mean(glucose) %>%
  step_impute_mean(chloride) %>%
  step_impute_mean(hematocrit) %>%
  step_impute_mean(sodium) %>%
  step_impute_mean(creatinine) %>%
  step_impute_mean(bicarbonate) %>%
  step_impute_mean(heart_rate) %>%
  step_impute_mean(body_temperature) %>%
  step_impute_mean(`diastolic_non-invasive_blood_pressure`) %>%
  step_impute_mean(respiratory_rate) %>%
  step_impute_mean(`systolic_non-invasive_blood_pressure`) %>%
  step_impute_mode(insurance) %>%
  step_impute_mode(marital_status) %>%
  step_dummy(all_nominal_predictors()) %>%
  step_zv(all_numeric_predictors()) %>%
  step_normalize(all_numeric_predictors())
```

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

```r
logistic_workflow = workflow() %>%
  add_recipe(logistic_recipe) %>%
  add_model(logistic_model)
```

```r
parameter_grid = grid_regular(penalty(range = c(-7, -1)),
                      mixture(), levels = c(50, 5)) %>%
  print()
```

```
# A tibble: 250 x 2
       penalty mixture
         <dbl>   <dbl>
 1 0.0000001         0
 2 0.000000133       0
 3 0.000000176       0
 4 0.000000233       0
 5 0.000000309       0
 6 0.000000409       0
 7 0.000000543       0
 8 0.000000720       0
 9 0.000000954       0
10 0.00000126        0
# i 240 more rows
```

```r
set.seed(203)
folds = vfold_cv(training_data, v = 5)
```

```r
logistic_fit = logistic_workflow %>%
  tune_grid(resamples = folds, grid = parameter_grid,
            metrics = metric_set(roc_auc, accuracy),
            control = control_stack_grid())
```

```r
logistic_fit %>%
  collect_metrics() %>%
  filter(.metric == "roc_auc") %>%
  ggplot(mapping = aes(x = penalty, y = mean, color = factor(mixture))) +
  geom_point() +
  labs(x = "Penalty", y = "CV AUC") +
  scale_x_log10()
```

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

```
best_logistic = logistic_fit %>%
  select_best(metric = "roc_auc")

best_logistic
```

Tuning and Training of Boosting Model

```
boosting_recipe = recipe(data = training_data,
                         los_long ~ .) %>%
  step_impute_mean(potassium) %>%
  step_impute_mean(white_blood_cell_count) %>%
  step_impute_mean(glucose) %>%
  step_impute_mean(chloride) %>%
  step_impute_mean(hematocrit) %>%
  step_impute_mean(sodium) %>%
  step_impute_mean(creatinine) %>%
  step_impute_mean(bicarbonate) %>%
  step_impute_mean(heart_rate) %>%
  step_impute_mean(body_temperature) %>%
  step_impute_mean(`diastolic_non-invasive_blood_pressure`) %>%
  step_impute_mean(respiratory_rate) %>%
  step_impute_mean(`systolic_non-invasive_blood_pressure`) %>%
  step_impute_mode(insurance) %>%
  step_impute_mode(marital_status) %>%
  step_dummy(all_nominal_predictors()) %>%
  step_zv(all_numeric_predictors())
```

```
boosting_model = boost_tree(mode = "classification",
                            trees = 1000,
                            tree_depth = tune(),
                            learn_rate = tune()) %>%
  set_engine("xgboost")
```

```
boosting_workflow = workflow() %>%
  add_recipe(boosting_recipe) %>%
  add_model(boosting_model)
```

```
boosting_grid = grid_regular(tree_depth(range = c(1L, 3L)),
                             learn_rate(range = c(-4, 4),
                             trans = log10_trans()),
                             levels = c(2, 10))
```

```
boosting_fit = boosting_workflow %>%
  tune_grid(resamples = folds, grid = boosting_grid,
            metrics = metric_set(roc_auc, accuracy),
            control = control_stack_grid())
```

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

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

```
best_boosting = boosting_fit %>%
  select_best(metric = "roc_auc")

best_boosting
```

Tuning and Training for Random Forest

```
random_forest_recipe = recipe(data = training_data,
                          los_long ~ .) %>%
  step_impute_mean(potassium) %>%
  step_impute_mean(white_blood_cell_count) %>%
  step_impute_mean(glucose) %>%
  step_impute_mean(chloride) %>%
  step_impute_mean(hematocrit) %>%
  step_impute_mean(sodium) %>%
  step_impute_mean(creatinine) %>%
  step_impute_mean(bicarbonate) %>%
  step_impute_mean(heart_rate) %>%
  step_impute_mean(body_temperature) %>%
```

```
  step_impute_mean(`diastolic_non-invasive_blood_pressure`) %>%
  step_impute_mean(respiratory_rate) %>%
  step_impute_mean(`systolic_non-invasive_blood_pressure`) %>%
  step_impute_mode(insurance) %>%
  step_impute_mode(marital_status) %>%
  step_zv(all_numeric_predictors())
```

```
random_forest_model = rand_forest(mode = "classification",
                mtry = tune(),
                trees = tune()) %>%
  set_engine("ranger")
```

```
random_forest_workflow = workflow() %>%
  add_recipe(random_forest_recipe) %>%
  add_model(random_forest_model)
```

```
random_forest_grid = grid_regular(trees(range = c(450L, 650L)),
                        mtry(range = c(1L, 3L)),
                        levels = c(5, 5))
```

```
random_forest_fit = random_forest_workflow %>% tune_grid(resamples = folds,
                                    grid = random_forest_grid,
                                    metrics = metric_set(roc_auc, accuracy),
                                    control = control_stack_grid())
```

```
random_forest_fit %>%
  collect_metrics() %>%
  filter(.metric == "roc_auc") %>%
  ggplot(mapping = aes(x = trees, y = mean, color = factor(mtry))) +
  geom_point() +
  labs(x = "Number of Trees", y = "CV AUC")
```

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

```
best_random_forest = random_forest_fit %>%
  select_best(metric = "roc_auc")

best_random_forest
```

Tuning and Training for Stacking

```
stacking_model = stacks() %>%
  add_candidates(logistic_fit) %>%
  add_candidates(boosting_fit) %>%
  add_candidates(random_forest_fit) %>%
  blend_predictions(penalty = 10^(-4:4),
                    metrics = c("roc_auc", "accuracy")) %>%
  fit_members()
```

```
stacking_model
```

```
autoplot(stacking_model)
```

```
# Plot shows impact of each model in the stacking model

autoplot(stacking_model, type = "weights")
```

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?

Logistic Regression Model for Testing Data

```
logistic_final_workflow = logistic_workflow %>%
  finalize_workflow(best_logistic)
```

```
logistic_final_workflow %>%
  last_fit(data_split) %>%
  collect_metrics()
```

Boosting Model for Testing Data

```
boosting_final_workflow = boosting_workflow %>%
  finalize_workflow(best_boosting)
```

```
boosting_final_workflow %>%
  last_fit(data_split) %>%
  collect_metrics()
```

Random Forest Model for Testing Data

```
random_forest_final_workflow = random_forest_workflow %>%
  finalize_workflow(best_random_forest)
```

```
random_forest_final_workflow %>%
  last_fit(data_split) %>%
  collect_metrics()
```

Stacking Model for Testing Data

```
los_long_predictions = testing_data %>%
  bind_cols(predict(stacking_model, ., type = "prob")) %>%
  print(width = Inf)
```

```
yardstick::roc_auc(los_long_predictions, truth = los_long,
                   contains(".pred_No"))
```

```
los_long_predictions = los_long_predictions %>%
  mutate(.pred_class = as.factor(ifelse(.pred_YES > .pred_NO, "YES", "NO")))

yardstick::accuracy(los_long_predictions, truth = los_long,
                    estimate = .pred_class)
```

```
xgb.importance(model = boosting_final_workflow %>%
  last_fit(data_split) %>%
  extract_fit_engine())
```

The top 3 most important features to predict whether length of stay is long according to the boosting model are: body temperature, systolic non-invasive blood pressure, and respiratory rate.

The Logistic Model had an AUC of about 0.60, while the accuracy was around 0.578. The Boosting Model had an AUC of around 0.64, while the accuracy was 0.60. The Random Forest Model had an AUC of around 0.64, while the accuracy was 0.60. The Stacked Model had an AUC of 0.64 with an accuracy of 0.61.

The Stacked Model seemed to have the best performance in terms of both AUC and accuracy compared to each of the models individually. While the AUC value was around the same for the Stacked Model alongside the Boosting and Random Forest Model, its accuracy was better. The Logistic Model seemed to have the worst performance. Its AUC and accuracy values were the lowest compared to the values of the other models.

In terms of interpretability, I would say that none of these models seemed to be that great at predicting if the length of stay is long or not. A coin toss has a probability of 0.50, so an AUC or accuracy value of around 0.6 isn't too impressive in comparison.