HW5: Programming for Model Comparison

Mike Maccia

Task 1: Conceptual Questions

- What is the purpose of using cross-validation when fitting a random forest model?

Cross-validation is used when fitting a random forest model in an effect to prevent overfitting. Since random forest models are ensembles of decision trees, where each decision tree model should be created on different versions of the training set. We still want to only train our model using training data (preventing the model from seeing the test data), so using cross-validation allows us to create multiple folds of the data so each tree can be created on different versions of the training data.

These individual trees are combined into a random forest.

- Describe the bagged tree algorithm.

The bagged tree algorithm is a method of using bootstrapping to get multiple samples to fit on the same model. The results of these models would then be averaged together across multiple trees to create the final model. This allows for decreased variance when compared to an individual tree fit model.

Bootstrapping is a method of using resampling of data to create many samples. You would treat each sample as the population and this would then be used to fit the regression tree. The individual sample "y"s are then combined together to create the final model/prediction. In resampling of the data, normally it is done with replacement, so the same observation could be pulled into each individual sample multiple times.

- What is meant by a general linear model?

General linear models (GLM) are a form of modeling in which the response variable could be binary (success or failure, yes/no, etc). The GLM models are considered more flexible since they allow for this flexibility of binary variable. One example of a GLM is logistic regression in which the predicted probabilities of the response variable are a log function and would fall between 0 and 1. When graphing, this takes on a sigmoid shape. Even though the response variable is binary, continuous variables are able to be part of the prediction formula.

- When fitting a multiple linear regression model, what does adding an interaction term do? That is, what does it allow the model to do differently as compared to when it is not included in the model?

Adding an interaction effect allows for a more flexible surface when modeling a multiple linear regression. As opposed to a line of linear regression (plotted as x vs y), a multiple linear regression model would be plotted on more axes (in 3-dimensions). The model would instead be plotted as a "plane" and could take on a flat or saddle-like shape. An example of an interaction could be seen using this equation:

$$E[Y \mid x_1, x_2] = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 (x_1 \cdot x_2)$$

In this instance the effect of x1 on Y differs based on the value of x2.

- Why do we split our data into a training and test set?

When creating predictive models, our goal is for our model to predict well for observations the model has never seen before. If we train our model on all of our data, there is a risk of overfitting and the model would not perform well on data it has not seen before.

In order to prevent overfitting, we create a training and test set by randomly splitting the data. Normally the data is split into either 80/20 or 70/30 training / test sets. The training data set is what we use to train the model. After, we can then predict using the test set and use a metric to determine how well our model worked.

Task 2: Data Prep

Loading packages and data

First the needed packages

```
suppressPackageStartupMessages(library(tidyverse))
suppressPackageStartupMessages(library(tidymodels))
suppressPackageStartupMessages(library(caret))
suppressPackageStartupMessages(library(yardstick))
```

Now let's load in the data. Looks like we will be looking at heart disease (`HeartDisease = 1` for patients with heart disease and `0` for patients who do not) along with different measurements of someone's health.

```
heart <- read.csv("heart.csv", header = TRUE)
heart_data <- as_tibble(heart) #make the data set a tibble
```

Let's first look at the dataset using the `summary()` function

```
summary(heart_data)
```

```
      Age            Sex            ChestPainType        RestingBP
 Min.   :28.00   Length:918         Length:918         Min.   :  0.0
 1st Qu.:47.00   Class :character   Class :character   1st Qu.:120.0
 Median :54.00   Mode  :character   Mode  :character   Median :130.0
 Mean   :53.51                                         Mean   :132.4
 3rd Qu.:60.00                                         3rd Qu.:140.0
 Max.   :77.00                                         Max.   :200.0
  Cholesterol      FastingBS       RestingECG            MaxHR
 Min.   :  0.0   Min.   :0.0000   Length:918         Min.   : 60.0
 1st Qu.:173.2   1st Qu.:0.0000   Class :character   1st Qu.:120.0
 Median :223.0   Median :0.0000   Mode  :character   Median :138.0
 Mean   :198.8   Mean   :0.2331                      Mean   :136.8
 3rd Qu.:267.0   3rd Qu.:0.0000                      3rd Qu.:156.0
 Max.   :603.0   Max.   :1.0000                      Max.   :202.0
 ExerciseAngina        Oldpeak          ST_Slope          HeartDisease
 Length:918         Min.   :-2.6000   Length:918         Min.   :0.0000
 Class :character   1st Qu.: 0.0000   Class :character   1st Qu.:0.0000
 Mode  :character   Median : 0.6000   Mode  :character   Median :1.0000
                    Mean   : 0.8874                      Mean   :0.5534
                    3rd Qu.: 1.5000                      3rd Qu.:1.0000
                    Max.   : 6.2000                      Max.   :1.0000
```

What type of varible (in R) is Heart Disease? Categorical or Quantitative?

The heart disease variable is quantitative in R.

Does this make sense? Why or why not?

No it does not make sense that heart disease is quantitative, since it really is not a numeric value. We only want to determine if someone has heart disease or not. It is current stored as an integer, but it is really a binary value, where 1 = success (or patient has heart disease) while 0 = failure (or the patient does not have heart disease). There is not a possibility of the value being 0.25 or something. We do not want it to be considered numeric since the mean of the responses does not necessarily make sense when modeling in the same way we would model using say the mean heart rate value.

Adjusting the data a bit

We will adjust the `heartdisease` variable to make it binary and format is as a factor. This should help with future modeling

We will create a final tibble for this step and call it `new_hear` where the old `heartdisease` variable and also `ST_slope` will be removed.

```
new_heart <- heart_data |>
  mutate(heart_disease = factor(HeartDisease, levels = c(0,1),
                                labels = c("No", "Yes"))) |> #make heart disease a factor
  select(-HeartDisease, -ST_Slope) |> #remove original heartdisease and ST_slope
  relocate(heart_disease) #move new heart disease variable to front

new_heart
```

```
# A tibble: 918 x 11
   heart_disease   Age Sex   ChestPainType RestingBP Cholesterol FastingBS
   <fct>         <int> <chr> <chr>              <int>       <int>     <int>
 1 No               40 M     ATA                  140         289         0
 2 Yes              49 F     NAP                  160         180         0
 3 No               37 M     ATA                  130         283         0
 4 Yes              48 F     ASY                  138         214         0
 5 No               54 M     NAP                  150         195         0
 6 No               39 M     NAP                  120         339         0
 7 No               45 F     ATA                  130         237         0
 8 No               54 M     ATA                  110         208         0
 9 Yes              37 M     ASY                  140         207         0
10 No               48 F     ATA                  120         284         0
# i 908 more rows
# i 4 more variables: RestingECG <chr>, MaxHR <int>, ExerciseAngina <chr>,
#   Oldpeak <dbl>
```

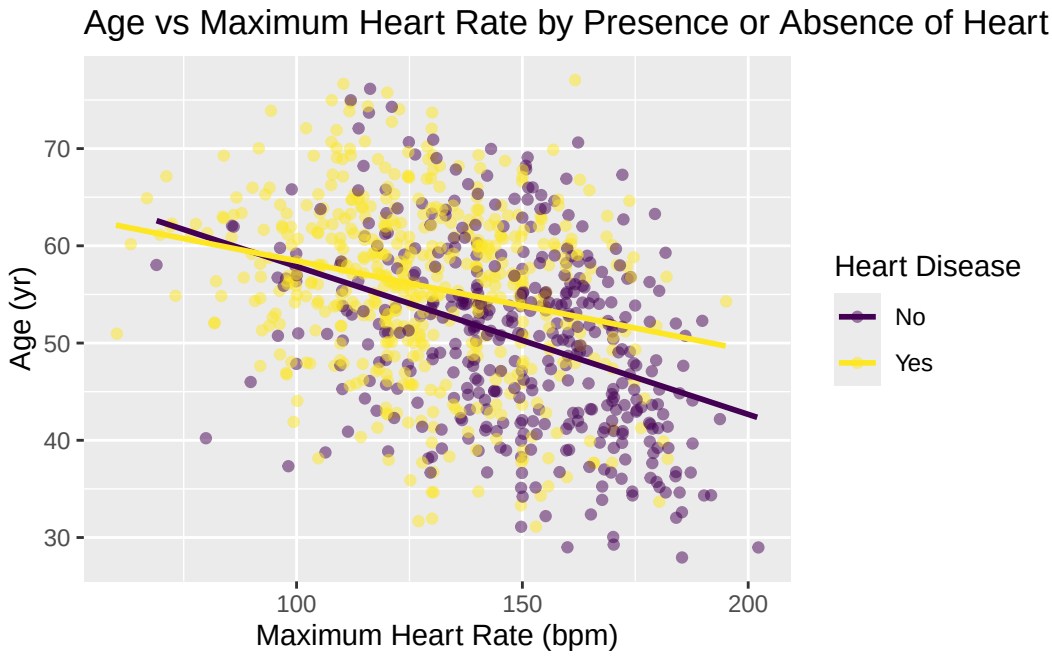
Task 3: Exploratory Data Analysis

Model Age (response variable) as a function of heart disease and their max heart rate.

We are going to first create a scatter plot representing if someone does or does not have heart disease

```
ggplot(data=new_heart, aes(x = MaxHR, y = Age, color = heart_disease)) +
  #convert heart_disease to a factor to make plotting easier
  geom_jitter(alpha = 0.5) +
  geom_smooth(method = "lm", se = FALSE) +
  scale_color_viridis_d() +
  labs(
    title = "Age vs Maximum Heart Rate by Presence or Absence of Heart Disease",
    x = "Maximum Heart Rate (bpm)",
    y = "Age (yr)",
    color = "Heart Disease"
  )
```

```
`geom_smooth()` using formula = 'y ~ x'
```



Should we use an interaction or additive model to answer our question?

Based on the plot above, it looks like people with older ages and lower maximum heart rates have more likelihood of having heart disease. It seems like there is a larger amount of people in the > 50 age range with a maximum heart rate of 100 - 125; these people also seem more likely to have heart disease. On the other hand, people who are younger and have higher maximum heart rates seem to have less heart disease.

As a result it seems like this is a situation where an interaction model would be more appropriate. The lines in this graph do not have the same slop and it seems like the effect of age and heart rate can be dependent on each other.

Task 4: Testing and Training

Let's split our data into a testing and training set. We will use a 80/20 split.

```
set.seed(101)
heart_split <- initial_split(new_heart, prop = 0.8)
train <- training(heart_split)
test <- testing(heart_split)
```

Task 5 OLS and LASSO

1: Fitting an interaction model with multiple linear regression

```
ols_mlr <- lm(Age ~ MaxHR * heart_disease, data = train)
summary(ols_mlr)
```

```
Call:
lm(formula = Age ~ MaxHR * heart_disease, data = train)

Residuals:
     Min       1Q   Median       3Q      Max
-22.7703  -5.7966   0.4516   5.7772  20.6378

Coefficients:
                       Estimate Std. Error t value Pr(>|t|)
(Intercept)            75.58896    3.07510  24.581  < 2e-16 ***
MaxHR                  -0.16992    0.02064  -8.233 8.43e-16 ***
heart_diseaseYes       -8.58502    3.83433  -2.239  0.02546 *
MaxHR:heart_diseaseYes  0.08343    0.02716   3.072  0.00221 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 8.478 on 730 degrees of freedom
Multiple R-squared:  0.1839,    Adjusted R-squared:  0.1806
F-statistic: 54.84 on 3 and 730 DF,  p-value: < 2.2e-16
```

2: Use Root Mean Square Error (RMSE) to evaluate the model's predictive performance on new data

```
#create the predictions
prediction_ols_mlr <- predict(ols_mlr, newdata = test)
#calculate the root mean square error, use yardstick
ols_results <- tibble(
  truth = test$Age,
  prediction = prediction_ols_mlr
)

ols_rmse <- rmse(ols_results, truth = truth, estimate = prediction)
ols_rmse
```

```
# A tibble: 1 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 rmse    standard        9.10
```

3: Using a LASSO model to see if it has a better predictive performance

In this model with will use cross-validation to select the best tuning parameter.

First let's create 10 fold cross-validation

```
train_folds <- vfold_cv(train)
```

Creating our LASSO recipe

```
LASSO_recipe <- recipe(Age ~ MaxHR + heart_disease, data = train) |>
  step_dummy(heart_disease) |>
  step_normalize(all_numeric_predictors()) |>
  step_interact(terms = ~ MaxHR:starts_with("heart_disease"))
```

4: Setting up our spec, workflow, and grid

Spec:

```
LASSO_spec <- linear_reg(penalty = tune(), mixture = 1) |>
  set_engine("glmnet")
```

Workflow:

```
LASSO_wf <- workflow() |>
  add_recipe(LASSO_recipe) |>
  add_model(LASSO_spec)
LASSO_wf
```

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

-- Preprocessor ----------------------------------------------------------------
3 Recipe Steps

* step_dummy()
* step_normalize()
* step_interact()

-- Model -----------------------------------------------------------------------
Linear Regression Model Specification (regression)

Main Arguments:
  penalty = tune()
  mixture = 1

Computational engine: glmnet
```

Grid:

```
LASSO_grid <- LASSO_wf |>
  tune_grid(resamples = train_folds,
            grid = grid_regular(penalty(), levels = 200))
```

Now collect the metrics:

```
LASSO_grid |>
  collect_metrics() |>
  filter(.metric == "rmse")
```

```
# A tibble: 200 x 7
    penalty .metric .estimator  mean     n std_err .config
      <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
 1 1   e-10 rmse    standard    8.50    10   0.162 Preprocessor1_Model001
 2 1.12e-10 rmse    standard    8.50    10   0.162 Preprocessor1_Model002
 3 1.26e-10 rmse    standard    8.50    10   0.162 Preprocessor1_Model003
 4 1.41e-10 rmse    standard    8.50    10   0.162 Preprocessor1_Model004
 5 1.59e-10 rmse    standard    8.50    10   0.162 Preprocessor1_Model005
 6 1.78e-10 rmse    standard    8.50    10   0.162 Preprocessor1_Model006
 7 2.00e-10 rmse    standard    8.50    10   0.162 Preprocessor1_Model007
 8 2.25e-10 rmse    standard    8.50    10   0.162 Preprocessor1_Model008
 9 2.52e-10 rmse    standard    8.50    10   0.162 Preprocessor1_Model009
10 2.83e-10 rmse    standard    8.50    10   0.162 Preprocessor1_Model010
# i 190 more rows
```

Pull out the best model:

```
best_LASSO <- LASSO_grid |>
  select_best(metric = "rmse")

best_LASSO
```

```
# A tibble: 1 x 2
  penalty .config
    <dbl> <chr>
1  0.0174 Preprocessor1_Model165
```

Now will fit that best model on the training set:

First we will make the workflow

```
LASSO_wf |>
  finalize_workflow(best_LASSO)
```

```
== Workflow ====================================================================
Preprocessor: Recipe
```

```
Model: linear_reg()

-- Preprocessor ----------------------------------------------------------------
3 Recipe Steps

* step_dummy()
* step_normalize()
* step_interact()

-- Model -----------------------------------------------------------------------
Linear Regression Model Specification (regression)

Main Arguments:
  penalty = 0.0174263338600965
  mixture = 1

Computational engine: glmnet
```

Fit the training set

```
LASSO_final <- LASSO_wf |>
  finalize_workflow(best_LASSO) |>
  fit(train)
tidy(LASSO_final)
```

```
# A tibble: 4 x 3
  term                       estimate penalty
  <chr>                         <dbl>   <dbl>
1 (Intercept)                   54.0   0.0174
2 MaxHR                         -3.08  0.0174
3 heart_disease_Yes              1.36  0.0174
4 MaxHR_x_heart_disease_Yes      1.03  0.0174
```

5: Would we expect RMSE calculations do the best same or different?

The RMSE calculations should be different since we are adding different penalty parameters as we run the model. When I set up my grid, I set `levels = 200`, so that tuned 200 values to the penalty hyperparameter and then we used the CV folds to evaluate the performance of the model.

Below are the results of the RMSE values:

```r
summary_LASSO <- LASSO_grid |>
  collect_metrics() |>
  filter(.metric == "rmse") |>
  distinct(mean) |> #only output distinct means
  arrange(mean) |>
  mutate(mean = format(mean, digits = 6)) |> #print at least 6 digits so the difference can b
  print()
```

```
# A tibble: 37 x 1
   mean
   <chr>
 1 8.49525
 2 8.49525
 3 8.49529
 4 8.49529
 5 8.49529
 6 8.49529
 7 8.49529
 8 8.49530
 9 8.49530
10 8.49531
# i 27 more rows
```

```r
min_rmse_LASSO <- min(as.numeric(summary_LASSO$mean))
max_rmse_LASSO <- max(as.numeric(summary_LASSO$mean))
```

As you can see, the RMSE values varied from the different models being evaluated.The mean RMSE varies from 8.49525 to 8.61431.

6. Compare the RMSE values between our OLS and LASSO models

RMSE for the OLS model

```r
ols_rmse #pulling from above
```

```
# A tibble: 1 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 rmse    standard        9.10
```

RMSE for Lasso model

```
LASSO_final |>
  predict(test) |>
  pull() |>
  rmse_vec(truth = test$Age)
```

```
[1] 9.095981
```

7: Why are the RMSE calculations roughtly the same if the coefficients for each model are different?

OLS and LASSO models are similar in the way they predict, but the LASSO model involves an additional tuning parameter. When fitting an OLS model, its goal is find betas that are close to the actual outcomes. LASSO models give a penalty to large coefficients to prevent overfitting and then small coefficients are shrunk to zero. While they may be created in different ways, it seems that OLS and LASSO models may typically have similar RMSE calculations.

Task 6: Logistic Regression

We will now look at 2 different logistic regression models to predict the presence of heart diease.

Model 1 will use the predictors of `Age`, `RestingBP`, and `Cholesterol`.

Model 2 will use the predictors of `RestingBP`, `MaxHR`, and `ExerciseAngina` (with an interaction between the latter 2).

We will start with our 2 model recipes:

```
LR1_recipe <- recipe(heart_disease ~ Age + RestingBP + Cholesterol, data = train) |>
  step_normalize(all_predictors()) #since only numeric variables

LR2_recipe <- recipe(heart_disease ~ RestingBP + MaxHR + ExerciseAngina, data = train) |>
  step_dummy(ExerciseAngina) |>  #categorical
  step_normalize(all_numeric_predictors()) |>
  step_interact(terms = ~ MaxHR:starts_with("ExerciseAngina")) #adding the interaction
```

Next we will set up our model and engine:

```
LR_spec <- logistic_reg() |>
  set_engine("glm")
```

Now make our 2 workflows:

```
LR1_wf <- workflow() |>
  add_recipe(LR1_recipe) |>
  add_model(LR_spec)

LR2_wf <- workflow() |>
  add_recipe(LR2_recipe) |>
  add_model(LR_spec)

LR1_wf
```

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

-- Preprocessor ----------------------------------------------------------------
1 Recipe Step

* step_normalize()

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

Computational engine: glm
```

```
LR2_wf
```

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

-- Preprocessor ----------------------------------------------------------------
3 Recipe Steps

* step_dummy()
* step_normalize()
```

```
* step_interact()

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

Computational engine: glm
```

Now let's fit to our cross validation folds:

```
LR1_fit <- LR1_wf |>
  fit_resamples(train_folds, metrics = metric_set(accuracy, mn_log_loss))

LR2_fit <- LR2_wf |>
  fit_resamples(train_folds, metrics = metric_set(accuracy, mn_log_loss))
```

Collect the metrics from both models and see which one was best:

```
rbind(LR1_fit |> collect_metrics(),
      LR2_fit |> collect_metrics()) |>
  mutate(Model = c("Model 1", "Model 1", "Model 2", "Model 2")) |>
  select(Model, everything())
```

```
# A tibble: 4 x 7
  Model   .metric     .estimator  mean     n std_err .config
  <chr>   <chr>       <chr>      <dbl> <int>   <dbl> <chr>
1 Model 1 accuracy    binary     0.665    10  0.0234 Preprocessor1_Model1
2 Model 1 mn_log_loss binary     0.622    10  0.0168 Preprocessor1_Model1
3 Model 2 accuracy    binary     0.743    10  0.0164 Preprocessor1_Model1
4 Model 2 mn_log_loss binary     0.531    10  0.0198 Preprocessor1_Model1
```

Above we have the accurary and mean log loss of both models. For some comparison, let's check the mean number of subjects with heart disease in the training data.

```
mean(train$heart_disease == "Yes")
```

```
[1] 0.5694823
```

Identifying the best model:

The best model to predict the presence of heart disease is model 2, with the predictors of `RestingBP`, `MaxHR`, and `ExerciseAngina` (with an interaction between the latter 2).

Model 2 is the best since it has higher accuracy at 0.748 and lower mean log loss at 0.527 compared to model 1. If we look at the mean of presence of heart disease in the training data set, we find a mean of 0.569. In a model, we want our accuracy to be higher than that, which Model 2 is. Additionally, the lower log loss indicates the model is more reliable in its predictions.

To provide a summary of the model, we had 3 predictors: - Resting Blood Pressure - Maximum Heart Rate - Exercise Angina (Yes or No presence during exercise)

Plus there was an interaction between maximum heart rate and presence of exercise angina. The model would show that the value of all 3 of these likely affect whether or not a patient has heart disease. Additionally, patients who experience exercise-induced angina, their relationship with their maximum heart rate and the presence of heart disease is probably different than someone who does not experience exercise-induced angina.

2: Using `confusionMatrix()` to check how well the model does on our test data:

First, let's get the confusion Matrix for our training set

```
LR_train_fit <- LR2_wf |>
  fit(train)
conf_mat(train |> mutate(estimate = LR_train_fit |>
                           predict(train) |>
                           pull()),
         heart_disease,
         estimate)
```

```
          Truth
Prediction  No Yes
       No  245 111
       Yes  71 307
```

Taking our best model and testing it on the test set

```
LR2_wf |>
  last_fit(heart_split, metrics = metric_set(accuracy, mn_log_loss)) |>
  collect_metrics()
```

```
# A tibble: 2 x 4
  .metric     .estimator .estimate .config
  <chr>       <chr>          <dbl> <chr>
1 accuracy    binary         0.728 Preprocessor1_Model1
2 mn_log_loss binary         0.516 Preprocessor1_Model1
```

Now here is the confusion matrix for our test set:

```
test_conf_mat <- conf_mat(test |> mutate(estimate = LR_train_fit |>
                          predict(test) |>
                          pull()),
         heart_disease,
         estimate)

test_conf_mat
```

```
          Truth
Prediction No Yes
       No  73  29
       Yes 21  61
```

3. Sensitivity and Specificity of the Model

```
Sensitivity <- sensitivity_vec(truth = test$heart_disease,
                               estimate = predict(LR_train_fit, test,
                                                  type = "class") |>
                                 pull())
Specificity <- specificity_vec(truth = test$heart_disease,
                               estimate = predict(LR_train_fit, test,
                                                  type = "class") |>
                                 pull())

paste("Sensitivity = ", round(Sensitivity,3))
```

```
[1] "Sensitivity =  0.777"
```

```
paste("Specificity = ", round(Specificity,3))
```

```
[1] "Specificity =  0.678"
```

With a sensitivity of 0.777, the model would correctly identify 77.7% of patients who actually have heart disease.

With a specificity of 0.678, the model would correctly identify 67.8% of patients who do not have heart disease.

In other words, our model is better at detecting people who have heart disease when they actually have it versus confirming they don't have heart disease. This would ideal in the medical world since we want to avoid missing sick patients versus find a false positive.