ST 558 Homework 8

Eric Warren

November 2, 2023

Contents

1 Reading in Data

The first thing we need to do is read in the data. This data is about looking at patients who might have heart disease.

```
library(tidyverse)
(heart <- read_csv("heart.csv"))
```

```
## # A tibble: 918 x 12
##      Age Sex   ChestPainType RestingBP Cholesterol FastingBS RestingECG MaxHR
##    <dbl> <chr> <chr>             <dbl>       <dbl>     <dbl> <chr>      <dbl>
##  1    40 M     ATA                 140         289         0 Normal       172
##  2    49 F     NAP                 160         180         0 Normal       156
##  3    37 M     ATA                 130         283         0 ST            98
##  4    48 F     ASY                 138         214         0 Normal       108
##  5    54 M     NAP                 150         195         0 Normal       122
##  6    39 M     NAP                 120         339         0 Normal       170
##  7    45 F     ATA                 130         237         0 Normal       170
##  8    54 M     ATA                 110         208         0 Normal       142
##  9    37 M     ASY                 140         207         0 Normal       130
## 10    48 F     ATA                 120         284         0 Normal       120
## # i 908 more rows
```

```
## # i 4 more variables: ExerciseAngina <chr>, Oldpeak <dbl>, ST_Slope <chr>,
## #   HeartDisease <dbl>
```

Now we are going to manipulate the data with how we are supposed to do it. The following is what we need to do. Create a new variable that is a factor version of the `HeartDisease` variable (if needed, this depends on how you read in your data). Remove the `ST_Slope` variable, the `ExerciseAngina` variable, and the original `HeartDisease` variable (if applicable – in our case we do not need to do this).

```
(heart <- heart %>%
    mutate(HeartDisease = as.factor(HeartDisease)) %>%
    select(-c(ST_Slope, ExerciseAngina)))
```

```
## # A tibble: 918 x 10
##      Age Sex   ChestPainType RestingBP Cholesterol FastingBS RestingECG MaxHR
##    <dbl> <chr> <chr>              <dbl>       <dbl>     <dbl> <chr>      <dbl>
##  1    40 M     ATA                  140         289         0 Normal       172
##  2    49 F     NAP                  160         180         0 Normal       156
##  3    37 M     ATA                  130         283         0 ST            98
##  4    48 F     ASY                  138         214         0 Normal       108
##  5    54 M     NAP                  150         195         0 Normal       122
##  6    39 M     NAP                  120         339         0 Normal       170
##  7    45 F     ATA                  130         237         0 Normal       170
##  8    54 M     ATA                  110         208         0 Normal       142
##  9    37 M     ASY                  140         207         0 Normal       130
## 10    48 F     ATA                  120         284         0 Normal       120
## # i 908 more rows
## # i 2 more variables: Oldpeak <dbl>, HeartDisease <fct>
```

2 KNN Modeling

We want to use kNN to predict whether or not someone has heart disease. To use kNN we generally want to have all numeric predictors (although we could try to create our own loss function to use with categorical predictors as an alternative). In this case we have some categorical predictors still in our data set: `Sex`, `ChestPainType`, and `RestingECG`.

Create dummy columns corresponding to the values of these three variables for use in our kNN fit. The caret vignette has a function to help us out here. You should use `dummyVars()` and `predict()` to create new columns. Then add these columns to our data frame and remove the original columns from which these variables were created.

```
library(caret)

# Make dummy variables
dummies <- dummyVars(HeartDisease ~ ., data = heart)
heart_dummies <- as_tibble(predict(dummies, newdata = heart))

# Make combined data frame with dummies and kept columns
heart_remove_factor_predictors <- heart %>%
    select(-c(Sex, ChestPainType, RestingECG))

heart_data <- as_tibble(merge(heart_remove_factor_predictors,
    heart_dummies, by = c("Age", "Cholesterol", "FastingBS",
```

```
        "MaxHR", "Oldpeak", "RestingBP")))

# Check for duplicates in original data
heart[duplicated(heart), ]
```

```
## # A tibble: 0 x 10
## # i 10 variables: Age <dbl>, Sex <chr>, ChestPainType <chr>, RestingBP <dbl>,
## #   Cholesterol <dbl>, FastingBS <dbl>, RestingECG <chr>, MaxHR <dbl>,
## #   Oldpeak <dbl>, HeartDisease <fct>
```

```
# Remove duplicates that might have merged for whatever
# reason (none should be present)
heart_data <- heart_data[!duplicated(heart_data), ]
```

Now split the data set you've created into a training and testing set. Use $p = 0.8$.

```
set.seed(999)
heart_index <- createDataPartition(heart_data$HeartDisease, p = 0.8,
    list = FALSE)
heart_train <- heart_data[heart_index, ]
heart_test <- heart_data[-heart_index, ]
```

Finally, train the kNN model. Use repeated 10 fold cross-validation, with the number of repeats being 3. You should also preprocess the data by centering and scaling. Lastly, set the `tuneGrid` so that you are considering values of k of 1, 2, 3, ... , 40. (Note: From the help for the `train()` function it says: `tuneGrid`: A data frame with possible tuning values. The columns are named the same as the tuning parameters. The name of the tuning parameter here is k.)

```
set.seed(999)
knn_fit <- train(HeartDisease ~ ., data = heart_train, method = "knn",
    trControl = trainControl(method = "repeatedcv", number = 10,
        repeats = 3), preProcess = c("center", "scale"), tuneGrid = data.frame(k = 1:40))
```

Check how well your model does on the test set using the `confusionMatrix()` function.

```
# Get predictions
knn_predict <- predict(knn_fit, newdata = heart_test)

# Get confusion matrix to show accuracy of model
(knn_confusion_matrix <- confusionMatrix(knn_predict, heart_test$HeartDisease))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 68 18
##          1 14 83
##
##                Accuracy : 0.8251
##                  95% CI : (0.7622, 0.8772)
##     No Information Rate : 0.5519
```

```
##     P-Value [Acc > NIR] : 0.000000000000005267
##
##                   Kappa : 0.6481
##
##  Mcnemar's Test P-Value : 0.5959
##
##             Sensitivity : 0.8293
##             Specificity : 0.8218
##          Pos Pred Value : 0.7907
##          Neg Pred Value : 0.8557
##              Prevalence : 0.4481
##          Detection Rate : 0.3716
##    Detection Prevalence : 0.4699
##       Balanced Accuracy : 0.8255
##
##        'Positive' Class : 0
##
```

```
# Accuracy
knn_confusion_matrix$overall[[1]]
```

```
## [1] 0.8251366
```

From our KNN Model we see that our model accuracy is 0.8251366. While we might wish that to be better for predictive purposes this might be a hard thing to predict. Thus, we are going to explore more models.

3 Ensemble

We'll look at predicting the same heart disease variable in this section as well, just instead of using KNN, we'll use the following methods:

- a classification tree (use `method = rpart`: tuning parameter is `cp`, use values 0, 0.001, 0.002, ..., 0.1)
- a bagged tree (use `method = treebag`: no tuning parameter)
- a random forest (use `method = rf`: tuning parameter is `mtry`, use values of 1, 2, ..., 15)
- a boosted tree (use `method = gbm`: tuning parameters are `n.trees`, `interaction.depth`, `shrinkage`, and `n.minobsinnode`, use all combinations of `n.trees` of 25, 50, 100, and 200, `interaction.depth` of 1, 2, 3, `shrinkage` = 0.1, and `n.minobsinnode` = 10; Hint: use `expand.grid()` to create your data frame for `tuneGrid`)

Using the training data you created above to fit each model (using repeated CV (3 repeats) as above but just 5 fold for computational ease). Test the model by finding the confusion matrix on the test data.

3.1 Classification Tree

First we are going to make a classification tree model.

```
# Make the model
set.seed(999)
class_tree_fit <- train(HeartDisease ~ ., data = heart_train,
    method = "rpart", trControl = trainControl(method = "repeatedcv",
```

```
        number = 5, repeats = 3), preProcess = c("center", "scale"),
    tuneGrid = data.frame(cp = seq(0, 0.1, by = 0.001)))

# Get predictions
class_tree_predict <- predict(class_tree_fit, newdata = heart_test)

# Get confusion matrix to show accuracy of model
(class_tree_confusion_matrix <- confusionMatrix(class_tree_predict,
    heart_test$HeartDisease))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 62 15
##          1 20 86
##
##                Accuracy : 0.8087
##                  95% CI : (0.7442, 0.863)
##     No Information Rate : 0.5519
##     P-Value [Acc > NIR] : 0.0000000000002522
##
##                   Kappa : 0.6111
##
##  Mcnemar's Test P-Value : 0.499
##
##             Sensitivity : 0.7561
##             Specificity : 0.8515
##          Pos Pred Value : 0.8052
##          Neg Pred Value : 0.8113
##              Prevalence : 0.4481
##          Detection Rate : 0.3388
##    Detection Prevalence : 0.4208
##       Balanced Accuracy : 0.8038
##
##        'Positive' Class : 0
##
```

```
# Accuracy
class_tree_confusion_matrix$overall[[1]]
```

```
## [1] 0.8087432
```

From our Classification Tree Model we see that our model accuracy is 0.8087432. While we might wish that to be better for predictive purposes, this model was worse than our KNN model, so we should not use it as our final model. Therefore, we are going to continue to explore more models.

3.2 Bagged Tree

Now we are going to make a bagged tree model.

```r
# Make the model
set.seed(999)
bag_tree_fit <- train(HeartDisease ~ ., data = heart_train, method = "treebag",
    trControl = trainControl(method = "repeatedcv", number = 5,
        repeats = 3), preProcess = c("center", "scale"))

# Get predictions
bag_tree_predict <- predict(bag_tree_fit, newdata = heart_test)

# Get confusion matrix to show accuracy of model
(bag_tree_confusion_matrix <- confusionMatrix(bag_tree_predict,
    heart_test$HeartDisease))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 63 15
##          1 19 86
##
##                Accuracy : 0.8142
##                  95% CI : (0.7502, 0.8678)
##     No Information Rate : 0.5519
##     P-Value [Acc > NIR] : 0.00000000000007197
##
##                   Kappa : 0.6226
##
##  Mcnemar's Test P-Value : 0.6069
##
##             Sensitivity : 0.7683
##             Specificity : 0.8515
##          Pos Pred Value : 0.8077
##          Neg Pred Value : 0.8190
##              Prevalence : 0.4481
##          Detection Rate : 0.3443
##    Detection Prevalence : 0.4262
##       Balanced Accuracy : 0.8099
##
##        'Positive' Class : 0
##
```

```r
# Accuracy
bag_tree_confusion_matrix$overall[[1]]
```

```
## [1] 0.8142077
```

From our Bag Tree Model we see that our model accuracy is 0.8142077. This model was worse than our KNN model but better than our Classification Tree model; however it is a model we will not want to use. We are going to continue to explore more models.

3.3 Random Forest

Now we are going to make a random forest model.

```r
# Make the model
set.seed(999)
rf_fit <- train(HeartDisease ~ ., data = heart_train, method = "rf",
    trControl = trainControl(method = "repeatedcv", number = 5,
        repeats = 3), preProcess = c("center", "scale"), tuneGrid = data.frame(mtry = 1:15))

# Get predictions
rf_predict <- predict(rf_fit, newdata = heart_test)

# Get confusion matrix to show accuracy of model
(rf_confusion_matrix <- confusionMatrix(rf_predict, heart_test$HeartDisease))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 64  8
##          1 18 93
##
##                Accuracy : 0.8579
##                  95% CI : (0.7988, 0.905)
##     No Information Rate : 0.5519
##     P-Value [Acc > NIR] : < 0.0000000000000002
##
##                   Kappa : 0.7094
##
##  Mcnemar's Test P-Value : 0.07756
##
##             Sensitivity : 0.7805
##             Specificity : 0.9208
##          Pos Pred Value : 0.8889
##          Neg Pred Value : 0.8378
##              Prevalence : 0.4481
##          Detection Rate : 0.3497
##    Detection Prevalence : 0.3934
##       Balanced Accuracy : 0.8506
##
##        'Positive' Class : 0
##
```

```r
# Accuracy
rf_confusion_matrix$overall[[1]]
```

```
## [1] 0.8579235
```

From our Random Forest Model we see that our model accuracy is 0.8579235. This model was better than our KNN model, so it is a model we should want to use. We are going to continue to explore one more model before our final model selection.

3.4 Boosted Tree

First we are going to make a boosted tree model.

```
# Make the model
set.seed(999)
boosted_tree_fit <- train(HeartDisease ~ ., data = heart_train,
    method = "gbm", trControl = trainControl(method = "repeatedcv",
        number = 5, repeats = 3), preProcess = c("center", "scale"),
    tuneGrid = data.frame(expand.grid(n.trees = c(25, 50, 100,
        200), interaction.depth = 1:3, shrinkage = 0.1, n.minobsinnode = 10)))
```

```
# Get predictions
boosted_tree_predict <- predict(boosted_tree_fit, newdata = heart_test)

# Get confusion matrix to show accuracy of model
(boosted_tree_confusion_matrix <- confusionMatrix(boosted_tree_predict,
    heart_test$HeartDisease))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 64 14
##          1 18 87
##
##                Accuracy : 0.8251
##                  95% CI : (0.7622, 0.8772)
##     No Information Rate : 0.5519
##     P-Value [Acc > NIR] : 0.000000000000005267
##
##                   Kappa : 0.6448
##
##  Mcnemar's Test P-Value : 0.5959
##
##             Sensitivity : 0.7805
##             Specificity : 0.8614
##          Pos Pred Value : 0.8205
##          Neg Pred Value : 0.8286
##              Prevalence : 0.4481
##          Detection Rate : 0.3497
##    Detection Prevalence : 0.4262
##       Balanced Accuracy : 0.8209
##
##        'Positive' Class : 0
##
```

```
# Accuracy
boosted_tree_confusion_matrix$overall[[1]]
```

```
## [1] 0.8251366
```

From our Boosted Tree Model we see that our model accuracy is 0.8251366. This model had worse accuracy than our random forest model. Therefore, we should select the random forest model from the models we have looked at. Note it is interesting that the KNN and Boosted tree model gave us the same accuracy rate in prediction.

4 Final Results

Here we are going to make a table showing how all the models did.

```
models <- c("KNN", "Classification Tree", "Bagged Tree", "Random Forest",
    "Boosted Tree")
accuracy <- c(knn_confusion_matrix$overall[[1]], class_tree_confusion_matrix$overall[[1]],
    bag_tree_confusion_matrix$overall[[1]], rf_confusion_matrix$overall[[1]],
    boosted_tree_confusion_matrix$overall[[1]])

# Show results
(model_results <- bind_cols(models = models, accuracy = accuracy))
```

```
## # A tibble: 5 x 2
##   models              accuracy
##   <chr>                  <dbl>
## 1 KNN                    0.825
## 2 Classification Tree    0.809
## 3 Bagged Tree            0.814
## 4 Random Forest          0.858
## 5 Boosted Tree           0.825
```

As we can see, the Random Forest model was the best ones to use for predicting heart disease. Even though we were not asked, in the future it might be interesting to see if we used logistic regression if that would have predicted things in a better way.