INFSCI 2595: Homework 10

Assigned: November 17, 2019, Due: November 26, 2019

David Ayodele

Submission time: 11/26/2019 at 9:00PM

Collaborators

Include the names of your collaborators here.

Overview

In this assignment you will focus on tuning machine learning algorithms using the `caret` package. You will practice most of the models we have discussed in this course. You will also get experience working with and interpreting ROC curves. At the conclusion of this assignment you will have completed a small binary classification modeling project from start to finish.

You will notice that when you run the code chunks warnings might be displayed. Usually the warning messages are disabled, but they are allowed to appear to help with installing packages required by `caret`.

Load packages

The code chunk below loads in the following packages. If you do not have `caret` or `plotROC` installed, please do so before starting this assignment. `caret` will you prompt you to download and install all other necessary packages, as required.

```
#install.packages("caret")
#install.packages("plotROC")
```

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(ggplot2)

library(caret)
```

```
## Loading required package: lattice
```

```
library(plotROC)
```

Read in training data

All problems in this assignment use the same training data set. That data set is loaded in the code chunk below. As shown by the `glimpse()` call, there are 1500 rows with 8 variables. The first 7 variables, `x1` through `x7` are inputs. Five of the 7 inputs are continuous. One input, `x3` is a categorical variable with 5 levels, while `x7` is a binary variable. The last variable, `output`, is the binary response consisting of two levels `"event"` and `"non_event"`. In this assignment you will try and predict the `"event"` class.

```
train_df <- readr::read_csv("https://raw.githubusercontent.com/jyurko/INFSCI_2595_Fall_2019/master/hw_da
                            col_types = list(readr::col_double(),
                                             readr::col_double(),
                                             readr::col_factor(levels = c("A", "B", "C", "D", "E")),
                                             readr::col_double(),
                                             readr::col_double(),
                                             readr::col_double(),
                                             readr::col_factor(levels = c("aa", "bb")),
                                             readr::col_factor(levels = c("event", "non_event"))))

train_df %>% glimpse()
```

```
## Observations: 1,500
## Variables: 8
## $ x1     <dbl> -3.0000000, 2.6666667, -4.0000000, -3.3333333, 3.333333...
## $ x2     <dbl> -3.0000000, -0.6666667, -0.6666667, 1.6666667, -0.66666...
## $ x3     <fct> D, A, B, D, E, A, D, B, E, E, D, B, E, E, A, B, D, A, A...
## $ x4     <dbl> 3.3333333, -1.3333333, 0.0000000, 1.6666667, 2.3333333,...
## $ x5     <dbl> 0.6666667, -1.6666667, 4.0000000, -3.6666667, -3.000000...
## $ x6     <dbl> 2.0000000, -2.0000000, 2.6666667, -3.0000000, -3.666666...
## $ x7     <fct> bb, aa, aa, aa, aa, aa, aa, aa, aa, aa, aa, bb, aa, bb,...
## $ output <fct> non_event, non_event, non_event, non_event, non_event, ...
```

Problem 1

As with any data analysis project, it's best to start out exploring the data before jumping into building models. You will begin by getting a high level overview of summary statistics through a simple exploratory data analysis.

1a)

A quick way to get important summary statistics is to call the `summary()` function. For continuous variables, `summary()` displays important statistics about the central tendancy with the mean and median. `summary()` also gives an idea about the spread in the data by displaying the first and third quartiles, as well as the absolute min and max values observed in the data set. For discrete factor variables, `summary()` provides the number of observations for each unique level.

PROBLEM

Call the `summary()` function on the training data set, `train_df`. Based on the counts for the discrete variables `x3` and `x7`, are there particular levels that dominate the observations? Based

on the counts alone, would you say the binary response, `output`, is a balanced or imbalanced data set?

SOLUTION

```
### your code here
summary(train_df)
```

```
##        x1                  x2                x3            x4
##  Min.   :-4.00000   Min.   :-4.00000   A:297   Min.   :-4.00000
##  1st Qu.:-2.00000   1st Qu.:-2.33333   B:305   1st Qu.:-2.00000
##  Median : 0.00000   Median : 0.00000   C:291   Median : 0.00000
##  Mean   : 0.04844   Mean   :-0.05333   D:303   Mean   :-0.04756
##  3rd Qu.: 2.00000   3rd Qu.: 2.00000   E:304   3rd Qu.: 2.00000
##  Max.   : 4.00000   Max.   : 4.00000           Max.   : 4.00000
##        x5                x6          x7            output
##  Min.   :-4.0000   Min.   :-4.000   aa:753   event    :585
##  1st Qu.:-2.3333   1st Qu.:-2.000   bb:747   non_event:915
##  Median :-0.3333   Median : 0.000
##  Mean   :-0.1320   Mean   :-0.006
##  3rd Qu.: 2.0000   3rd Qu.: 2.000
##  Max.   : 4.0000   Max.   : 4.000
```

There appear to be no particular levels for x3 and x7 that dominate as the levels are all within 5% of each other. Based on the counts, the response appears to be an imbalanced set.

1b)

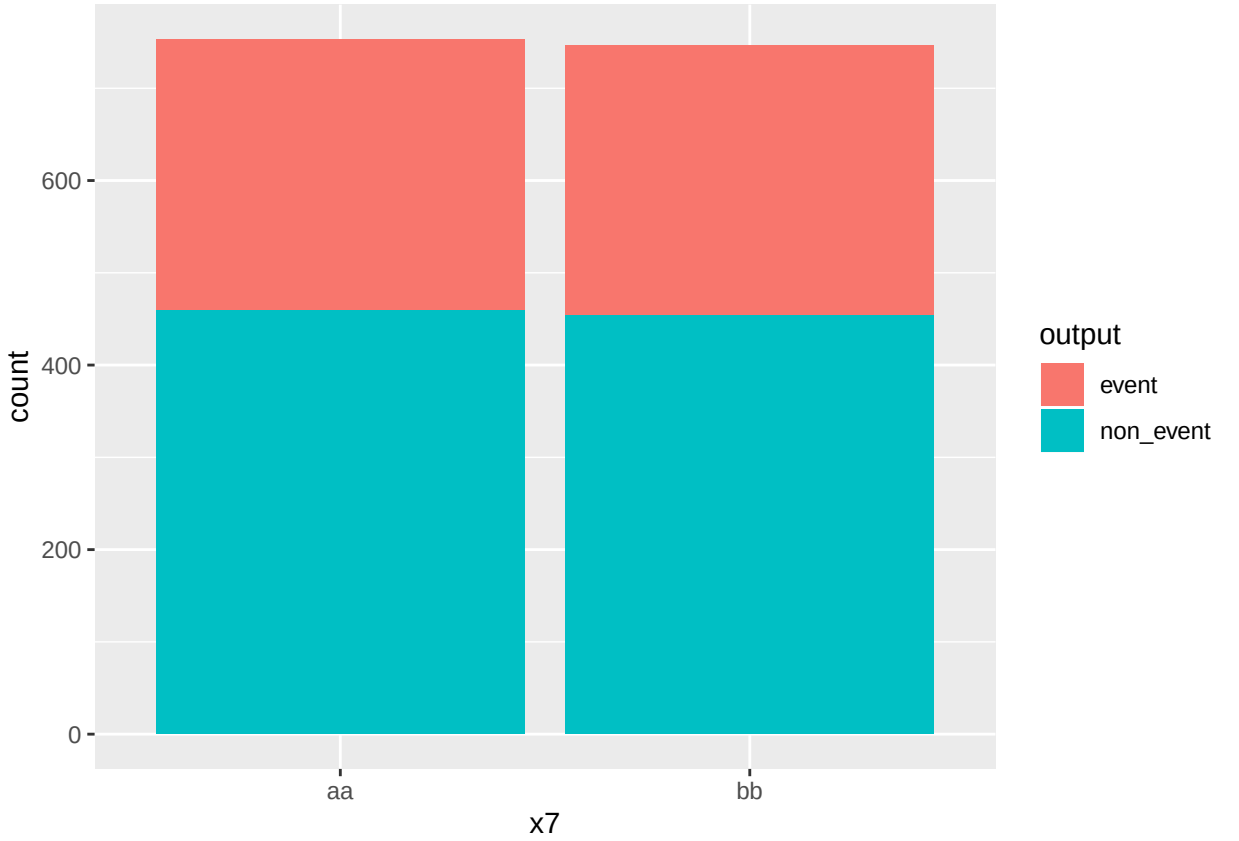
Let's now visualize the binary response for each of the two discrete factors, starting with the binary variable x7.

PROBLEM

Use `ggplot2` to create a bar chart with `geom_bar()`. Set the `x` aesthetic equal to `x7` in the parent `ggplot()` call. In the `geom_bar()` call set the `fill` aesthetic equal to `output`. Can you tell if the `"event"` occurs more frequently for either of the levels of `x7`?

SOLUTION

```
### your code here

ggplot(data=train_df, mapping=aes(x=x7)) + geom_bar(mapping = aes(fill=output))
```



Yes, for both levels, the event appears to occur less frequently than the non-event.

1c)

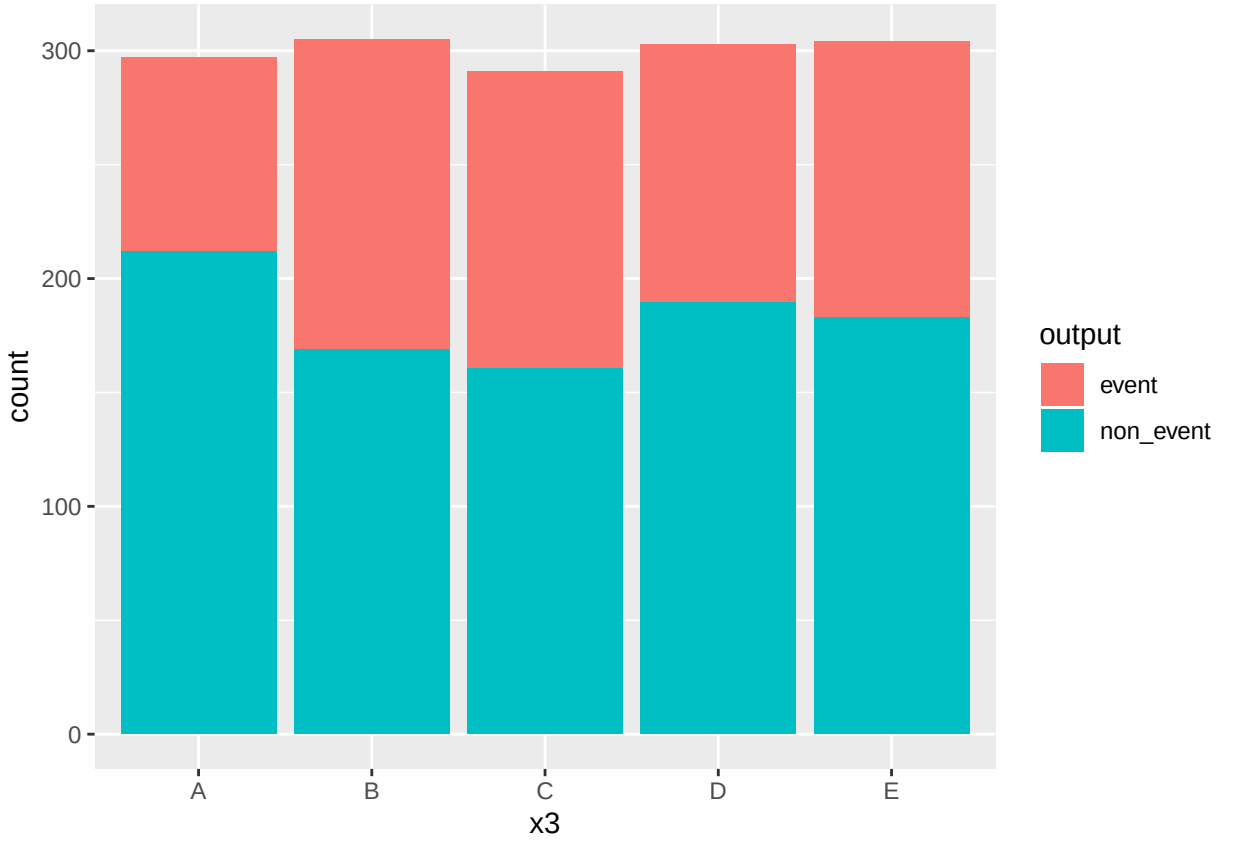
Now consider the `x3` factor which has 5 unique levels.

PROBLEM

Use `ggplot2` to create a bar chart with `geom_bar()`. Set the `x` aesthetic equal to `x3` in the parent `ggplot()` call. In the `geom_bar()` call set the `fill` aesthetic equal to `output`. Can you tell if the `"event"` occurs more frequently for either of the levels of `x3`?

SOLUTION

```
### your code here
ggplot(data=train_df, mapping=aes(x=x3)) + geom_bar(mapping = aes(fill=output))
```

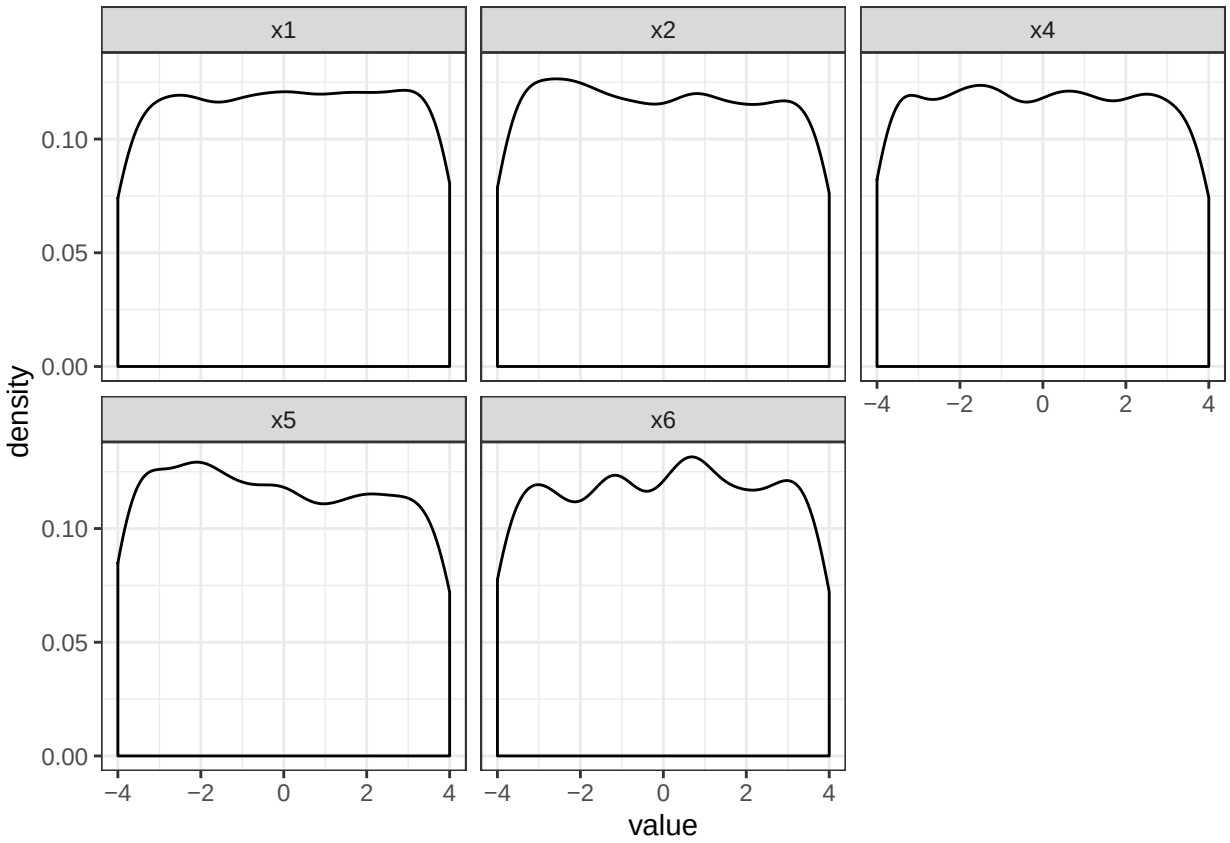


In all cases for x3, the non-event appears to occur more frequently with level A showing a relatively lower occurence of the event and levels B, and C showing a relatively higher occurence of the event.

1d)

Let's now look at the distributions associated with each continuous input. We will first consider the distributions irrespective of the response and the other discrete inputs. The code chunk below reshapes `train_df` to allow visualizing the individual continuous inputs within separate facets. The *marginal* distributions are visualized using the `geom_density()` function. As shown by the figure below, the distributions are essentially uniform.

```
train_df %>%
  tibble::rowid_to_column("obs_id") %>%
  tidyr::gather(key = "input_name", value = "value",
                -obs_id, -x3, -x7, -output) %>%
  ggplot(mapping = aes(x = value)) +
  geom_density() +
  facet_wrap(~input_name) +
  theme_bw()
```



You will now break up each distribution based on the the discrete variable `x3` and the response, `output`. To help understand the structure of the reshaped data set, the code chunk below prints the first 9 rows to the screen.

Map the values of the continuous inputs with respect to 'x3', 'x7' and 'output' ?

```
train_df %>%
  tibble::rowid_to_column("obs_id") %>%
  tidyr::gather(key = "input_name", value = "value",
                -obs_id, -x3, -x7, -output) %>%
  head(100)
```

```
## # A tibble: 100 x 6
##    obs_id x3    x7    output    input_name  value
##     <int> <fct> <fct> <fct>     <chr>       <dbl>
##  1      1 D     bb    non_event x1         -3
##  2      2 A     aa    non_event x1          2.67
##  3      3 B     aa    non_event x1         -4
##  4      4 D     aa    non_event x1         -3.33
##  5      5 E     aa    non_event x1          3.33
##  6      6 A     aa    non_event x1         -3.33
##  7      7 D     aa    event     x1          2.67
##  8      8 B     aa    event     x1          3.67
##  9      9 E     aa    non_event x1          0.667
## 10     10 E     aa    event     x1         -3
## # ... with 90 more rows
```

PROBLEM

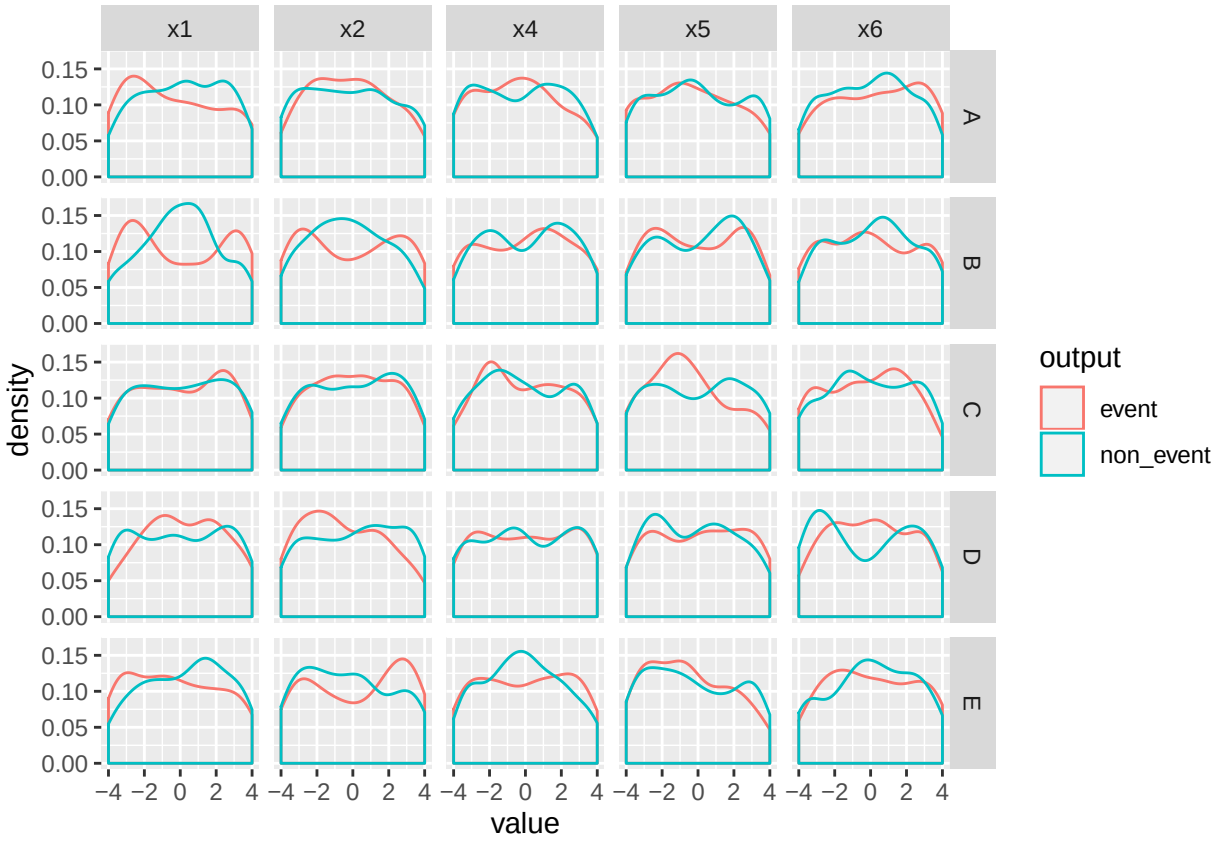
You must modify the code which creates the separate distributions by setting the `color` aesthetic in the `geom_density()` call to be equal to `output`. You must also modify the `facet_wrap()` call to be `facet_grid()`. The horizontal facets should be based on the `input_name` variable while `x3` controls the vertical facets.

Based on the visualizations, are there any values of the continuous inputs that seem to be more associated with the `"event"` occuring? Does the `x3` discrete factor influence behavior?

HINT: The R for Data Science book discusses how to create facets if you need a refresher.

SOLUTION

```
### place your modification of the code from
### the show_density_marg_train code chunk here
train_df %>%
  tibble::rowid_to_column("obs_id") %>%
  tidyr::gather(key = "input_name", value = "value",
                -obs_id, -x3, -x7, -output) %>%
  ggplot(mapping = aes(x = value)) +
  geom_density(mapping=aes(color=output)) +
  facet_grid(x3 ~ input_name)
```



Based on the visualizations, are there any values of the continuous inputs that seem to be more associated with the `"event"` occuring? Does the `x3` discrete factor influence behavior?

x1 with respect to A, x4 with respect to E, and x5 with respect to C appear to show more non-uniform behavior but, overall, none of the inputs appear to be strongly correlated with the event. Yes, the discrete factor x3 appears to skew the behavior of the event and non-event in a seemingly random manner.

1e)

Exploratory Data Analysis (EDA) helps us get famaliar with a data set and observe high level trends. EDA can help us make an informed decision about what modeling practices are appropriate to consider.

PROBLEM

Based on your visualizations, would you feel it is safe to say interactions are not relevant for this problem?

SOLUTION

No, all possible interactions of the inputs have not been explored.

Problem 2

You will now begin to model the binary outcome, `output`, based on the inputs. You will use `caret` to manage the cross-validation data splits and calculate the performance metrics. In this assignment, you will focus on the area under the ROC curve (AUC) as the primary metric of interest for assess model performance. In `caret`, the area under the ROC curve is referred to as ROC.

2a)

As discussed in lecture, the `trainControl()` function is used to tell `caret` how to manage the training and performance assessment. You will specify the arguments of the `trainControl()` function and also specify the primary metric of interest.

PROBLEM

Specify the `trainControl()` to use 5-fold cross validation. You must specify the `summaryFunction`, `classProbs`, and `savePredictions` appropriately in order to allow creating the ROC curves. You must also set the variable `metric_use` equal to `"ROC"`.

SOLUTION

```
ctrl_k05 <- trainControl(method="repeatedcv",
                         number = 5,
                         repeats = 3,
                         summaryFunction=twoClassSummary,
                         classProbs=TRUE,
                         savePredictions = TRUE)

metric_used <- "ROC"
```

2b)

As we have discussed multiple times this semester, it is always important to consider a simple model before building complex ones. So you will begin by fitting a logistic regression model. Your job will be to complete the `train()` function call to fit the logistic regression model.

The `set.seed()` function is set for you, and is set before all models in this assignment. Using the same random seed forces the random data splits to be the same across all models.

PROBLEM

Complete the code chunk below by specifying the formula to correctly model the binary outcome, `output`, as a function of all other variables in the data set. Set the `data` argument equal to `train_df`. Set the `method` argument equal to `"glm"`, and the `metric` argument equal to the `metric_use` variable defined in Problem 2a). Finally, specify the `trControl` argument equal to the `ctrl_k05` variable you defined in Problem 2a).

Once the code chunk below completes, print the `caret` object `fit_glm` to the screen. What is the area under the ROC curve value (which `caret` calls ROC) for your logistic regression model? Is this value associated with the training set or holdout set error?

SOLUTION

```
set.seed(98131)
fit_glm <- train(output ~ ., data = train_df,
                 method = "glm",
                 metric = metric_used,
                 trControl = ctrl_k05)
```

```
fit_glm
```

```
## Generalized Linear Model
##
## 1500 samples
##    7 predictor
##    2 classes: 'event', 'non_event'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 3 times)
## Summary of sample sizes: 1200, 1200, 1200, 1200, 1200, 1200, ...
## Resampling results:
##
##   ROC        Sens        Spec
##   0.5288808  0.01937322  0.9763206
```

What is the area under the ROC curve value (which `caret` calls ROC) for your logistic regression model? Is this value associated with the training set or holdout set error?** ?

The area is approx. 0.529. Since we used the train_df set, the area appears to be associated with the training set.

2c)

This problem consists of 7 inputs. But, how many parameters were learned in the logistic regression model? A simple way to find out is to use the `coef()` function, which extracts the coefficients (parameter) values associated with the model object. You can access the logistic regression model within the `caret` object by calling `fit_glm$finalModel`.

PROBLEM

Call the `coef()` function on the logistic regression model. How many parameters were learned? Why are the parameters named the way they are?

SOLUTION

```
### your code here
coef(fit_glm$finalModel)
```

```
##   (Intercept)            x1            x2           x3B           x3C
##  0.9257183054  0.0269253125  0.0013738964 -0.7015426126 -0.7053128092
##           x3D           x3E            x4            x5            x6
## -0.4006518286 -0.4962845183 -0.0006240123  0.0278721689 -0.0100927137
##          x7bb
## -0.0119654125
```

How many parameters were learned? Why are the parameters named the way they are?* ?
11 parameters appear to have been learned. They are named, it seems, after the inputs they represent, e.g. "x3C" for input x3 level C.

2d)

You will now use regularization with elastic net through the `glmnet` package. We discussed how the penalty or regularization factor for lasso and ridge regression can be tuned via cross-validation. The elastic net uses the same regularization parameter, `lambda`, but also includes a second tuning parameter, `alpha`. The additional parameter is a weighting or mixing parameter which controls which of the two penalty terms, lasso or ridge, is more prevelant.

You will use the elastic net to a fit a model accounting for interactions between the inputs. Before fitting the model you must determine how many parameters are in a model with all pair-wise and all 3-way interactions.

PROBLEM

How many parameters must be learned in a generalized linear model with all pair-wise interactions? How many parameters must be learned in a generalized linear model with all 3-way interactions? Why would a model like elastic net be an appropriate choice over a conventional generalized model when considering the interactions in this case?

SOLUTION

```
### your code here
choose(7, 2)
```

```
## [1] 21
```

```
choose(7, 3)
```

```
## [1] 35
```

How many parameters must be learned in a generalized linear model with all pair-wise interactions? How many parameters must be learned in a generalized linear model with all 3-way interactions? Why would a model like elastic net be an appropriate choice over a conventional generalized model when considering the interactions in this case? ?
Since there 7 inputs, there would be 21 pairwise interactions, and 35 3-way interactions. Because elastic net considers the combined penalties of lasso and ridge regresion, it would likely better suited for multi-interaction parameters.

2e)

You will now fit the elastic net model with all pair-wise interactions. You will use the default `caret` search grid for the `alpha` and `lambda` tuning parameters, so you do not need to additional arguments in the call to the `train()` function just yet.

PROBLEM

Fit an elastic net model with all pair-wise interactions. You must specify the formula correctly and set the `method` argument equal to `"glmnet"`. All other arguments can be the same as the call in Problem 2b). After the code chunk completes, print the `caret` object to the screen.

SOLUTION

```
#install.packages("glmnet")
```

```
set.seed(98131)
fit_glmnet_2 <- train(output ~ .^2, data = train_df,
                      method = "glmnet",
                      metric = metric_used,
                      trControl = ctrl_k05)
```

```
fit_glmnet_2
```

```
## glmnet
##
## 1500 samples
##    7 predictor
##    2 classes: 'event', 'non_event'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 3 times)
## Summary of sample sizes: 1200, 1200, 1200, 1200, 1200, 1200, ...
## Resampling results across tuning parameters:
##
##   alpha  lambda        ROC        Sens        Spec
```

```
##   0.10   0.0001359713  0.5619292  0.23361823  0.8298725
##   0.10   0.0013597126  0.5618732  0.23247863  0.8320583
##   0.10   0.0135971264  0.5590771  0.20113960  0.8684882
##   0.55   0.0001359713  0.5622095  0.23304843  0.8320583
##   0.55   0.0013597126  0.5617829  0.22450142  0.8364299
##   0.55   0.0135971264  0.5572276  0.15441595  0.9234973
##   1.00   0.0001359713  0.5620880  0.23247863  0.8309654
##   1.00   0.0013597126  0.5617113  0.21880342  0.8466302
##   1.00   0.0135971264  0.5607367  0.09059829  0.9511840
##
## ROC was used to select the optimal model using the largest value.
## The final values used for the model were alpha = 0.55 and lambda
##  = 0.0001359713.
```

2f)

PROBLEM

Fit another elastic net model using the `caret` default tuning grid. This time however, fit the model allowing for all 3-way interactions. How does the best tuned elastic net with 3-way interactions compare to the best tuned elastic net with pair-wise interactions, based on the area under the ROC curve? Which model type appears to be better?

SOLUTION

```
set.seed(98131)
fit_glmnet_3 <- train(output ~ .^3, data = train_df,
                      method = "glmnet",
                      metric = metric_used,
                      trControl = ctrl_k05)
```

```
fit_glmnet_3
```

```
## glmnet
##
## 1500 samples
##    7 predictor
##    2 classes: 'event', 'non_event'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 3 times)
## Summary of sample sizes: 1200, 1200, 1200, 1200, 1200, 1200, ...
## Resampling results across tuning parameters:
##
##   alpha  lambda        ROC        Sens       Spec
##   0.10   0.0002665207  0.6162782  0.3783476  0.7814208
##   0.10   0.0026652066  0.6177541  0.3669516  0.7934426
##   0.10   0.0266520660  0.6262513  0.3002849  0.8615665
##   0.55   0.0002665207  0.6169041  0.3772080  0.7828780
##   0.55   0.0026652066  0.6241745  0.3458689  0.8123862
##   0.55   0.0266520660  0.6426510  0.1829060  0.9570128
##   1.00   0.0002665207  0.6173898  0.3772080  0.7854281
```

```
##   1.00   0.0026652066  0.6289695  0.3327635  0.8327869
##   1.00   0.0266520660  0.6319960  0.1441595  0.9795993
##
## ROC was used to select the optimal model using the largest value.
## The final values used for the model were alpha = 0.55 and lambda
##  = 0.02665207.
```

How does the best tuned elastic net with 3-way interactions compare to the best tuned elastic net with pair-wise interactions, based on the area under the ROC curve? Which model type appears to be better?** ?
The elastic net with 3-way interactions appears to be better with a best tuned ROC value of appox. 0.64 while the pairwise interaction case had a best ROC value of approx. 0.562.

2g)

The `caret` object print outs display the default grid search values used to try and identify the "best" values of `alpha` and `lambda`. This default grid is a great starting point, but let's try and refine it further. You will use the `expand.grid()` function to define a custom grid search over the `alpha` and `lambda` tuning parameters.

PROBLEM

Complete the code chunk below which creates the grid search `glmnet_grid` over `alpha` and `lambda`, and then executes the training. You must specify `alpha` to be a vector of 4 values: 0.25, 0.55, 0.85, and 1.0. You must specify `lambda` to be a vector from -8 and 0.5 in log-lambda space. Each point is spaced by 0.25 log-lambda units.

WHY NOT JUST GIVE THE PARAMETERS OF THE SEQENCE? WHAT DOES SOLVING THIS (IN GARBAGE R) HAVE TO DO WITH MACHINE LEARNING??

You must then perform the training on the elastic net model with all 3-way interactions. In order to use your custom search grid, you must specify `tuneGrid` to be equal to `glmnet_grid`.

After the code chunk completes, rather than printing the results to screen, you will plot the results by calling the `plot()` function on the `caret` object `fit_glmnet_3_b`. Call the `plot()` function two times. In the first call, just plot the results. In the second call set the `xTrans` argument equal to `log` in the `plot()` call. This converts the x-axis to be log-lambda, rather than `lambda` directly.

Which values of the regularization parameter, `lambda`, appear to yield the highest area under the ROC curve? Are there `lambda` values that clearly yield worse results? What did you use to make that judgement?

HINT: `caret` requires the tuning parameter to correspond to `lambda` **not** log-lambda. By setting the values to be evenly spaced in log-lambda, you must transform the vector appropriately in order to create the `lambda` value required by `caret`.

SOLUTION

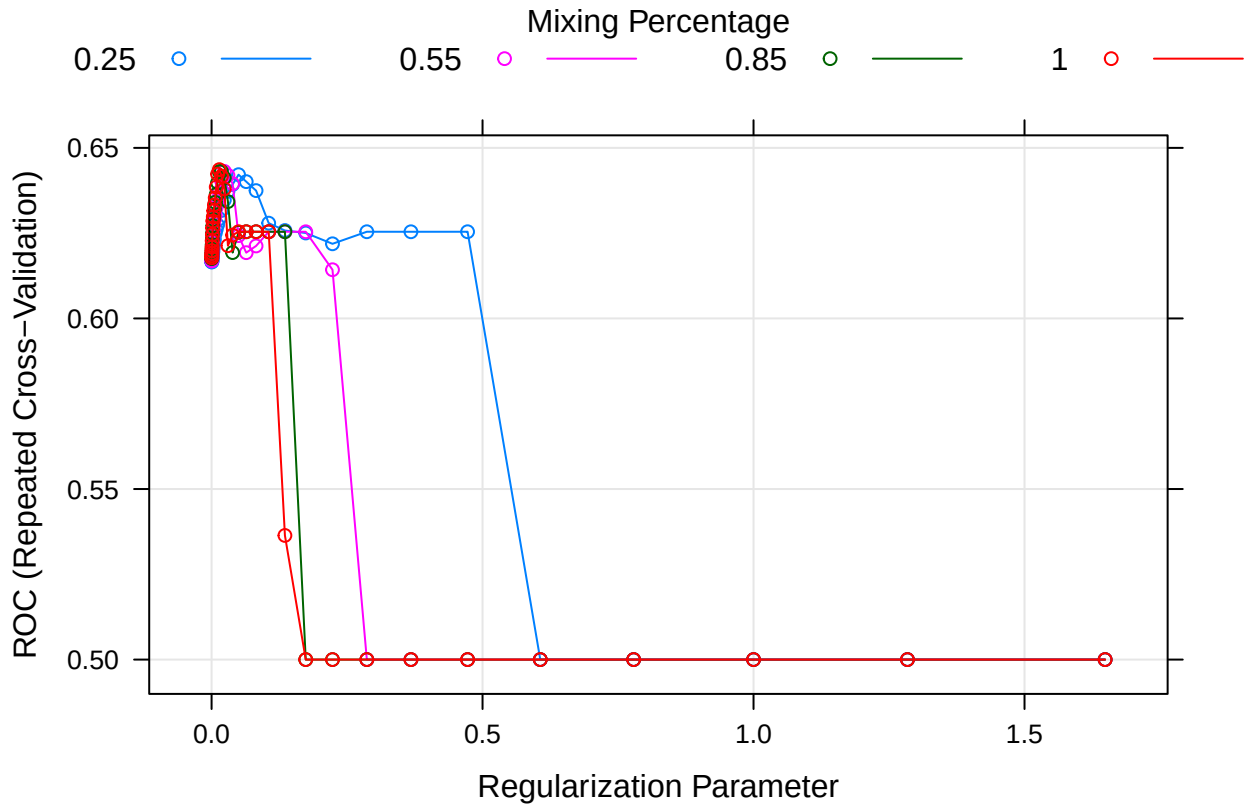
```
alpha = c(0.25, 0.55, 0.85, 1)
lambda = exp(seq(-8, 0.5, by=0.25))
```

```
glmnet_grid <- expand.grid(
                alpha = c(0.25, 0.55, 0.85, 1),
                lambda = exp(seq(-8, 0.5, by=0.25))
```

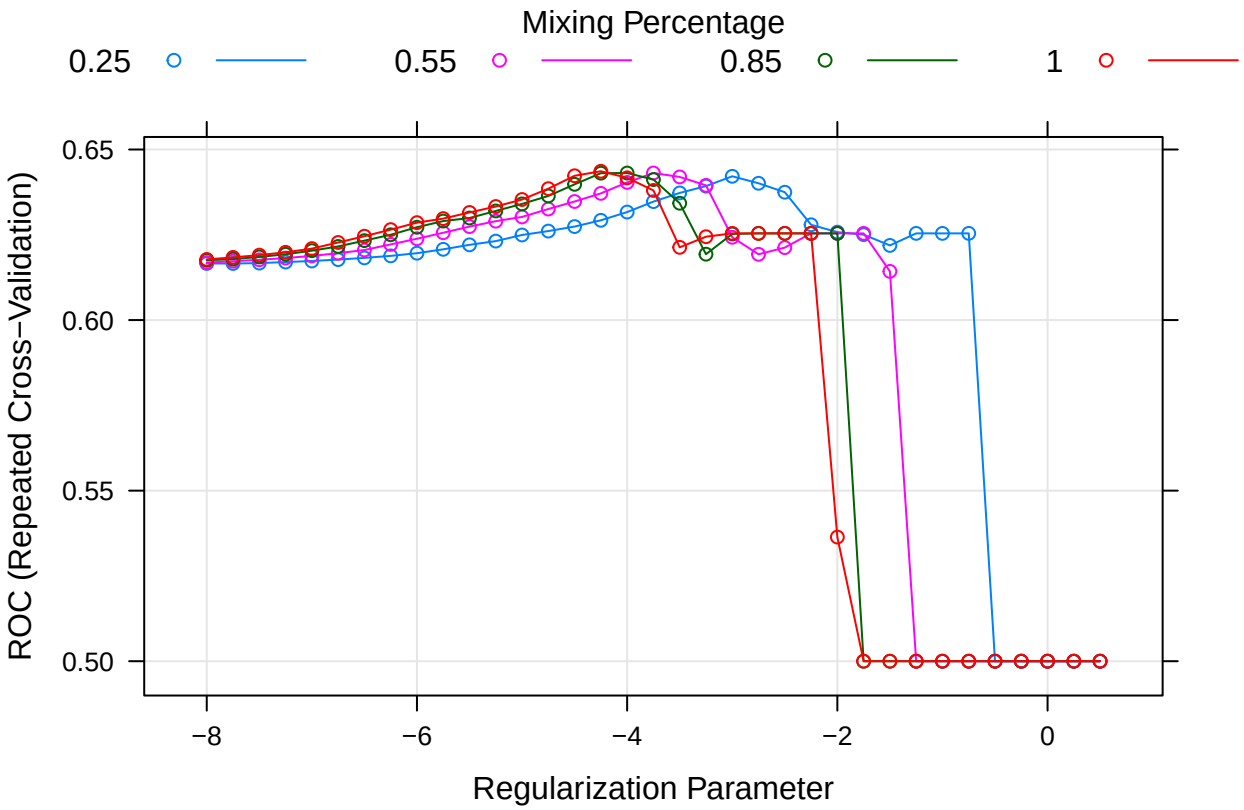
```
                )

set.seed(98131)
fit_glmnet_3_b <- train(output ~ .^3, data = train_df,
                        method = "glmnet",
                        metric = metric_used,
                        tuneGrid = glmnet_grid,
                        trControl = ctrl_k05)
```

Plot the results.

```
### your code here
plot(fit_glmnet_3_b)
```




```
### your code here
plot(fit_glmnet_3_b, xTrans=log)
```



Which values of the regularization parameter, `lambda`, appear to yield the highest area under the ROC curve? Are there `lambda` values that clearly yield worse results? What did you use to make that judgement?

?

Values of lambda near 0 appear to yield the highest area. Yes, lambda values greater than approx. 0.55 appear to yield the worst area. The area under the ROC curve was used.

2h)

You could print the `fit_glmnet_3_b` object to screen, but the print out will be very large. After all, you just trained elastic net models for 140 combinations of `lambda` and `alpha`! So rather than reading through the complete print out, you can access the tuned parameter values directly with the `$bestTune` variable within the `caret` model object. Furthermore, you can the access the different tuning parameters with the `$` operator. For example, to access the best tuned value for `alpha`, you would use `$bestTune$alpha`.

PROBLEM

Print the best tuning parameters values to screen and calculate the log of the best tuned `lambda` value and print it to screen. Are these values in line with your intepretation of the figures in the previous problem?

SOLUTION

```
### your code here
fit_glmnet_3_b$bestTune
```

```
##     alpha     lambda
## 121     1 0.01426423
```

```
log(fit_glmnet_3_b$bestTune$lambda)
```

```
## [1] -4.25
```

Are these values in line with your intepretation of the figures in the previous problem?**
?
Yes. The curve corresponding to alpha=1 near lambda=0.014 appears to have the highest area under the curve.

2i)

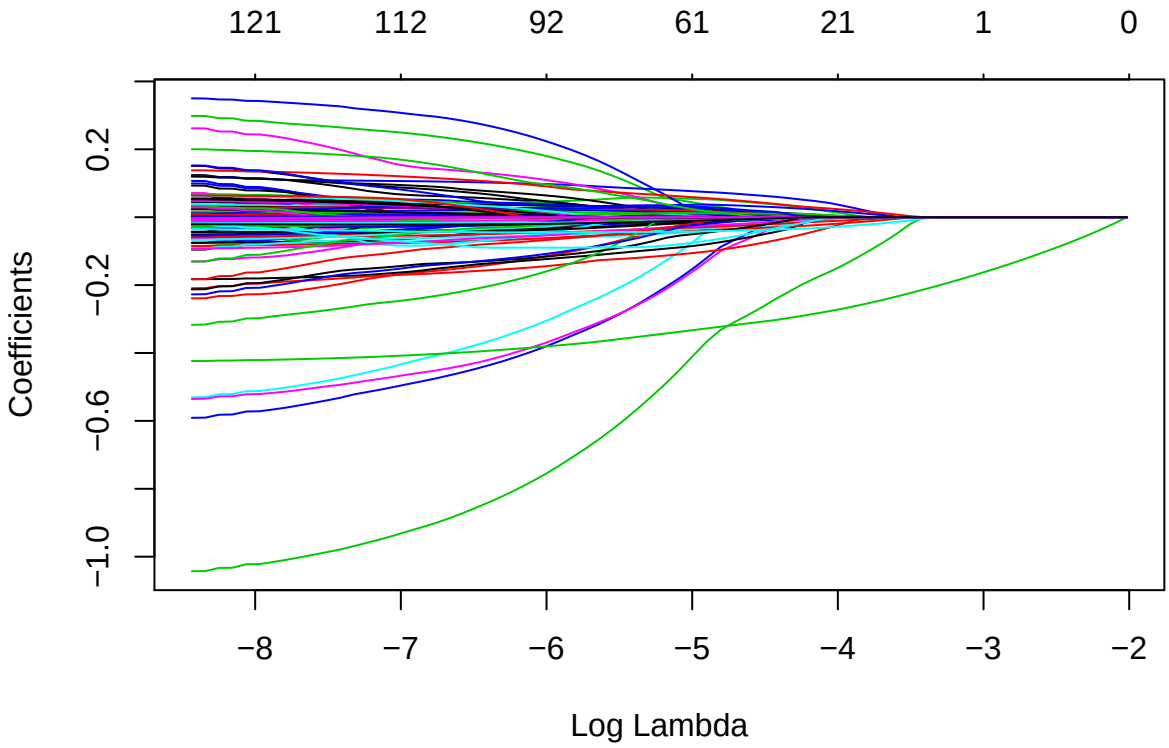
Just as with the logistic regression model, you can access the final elastic net model object by `$finalModel`. Let's check that the behavior of the elastic net model directly, by calling `plot()` on the final model object. This model object is the result from the `glmnet` package directly, and so provides the same functionality as when we discussed interpreting `glmnet` for lasso regression in lecture.

PROBLEM

Call `plot()` on the final model object associated with the `fit_glmnet_3_b caret` object. Set the `xvar` argument to `"lambda"`. How many parameters are "active" around the `lambda` value you identified as optimal for the 3-way interaction model?

SOLUTION

```
### your code here
plot(fit_glmnet_3_b$finalModel, xvar="lambda")
```



How many parameters are "active" around the `lambda` value you identified as optimal for the 3-way interaction model?** ?
At least 2 parameters apprear to be active around log(lambda)=-4.25.

2j)

By turning off in-active or unimportant inputs, the elastic net is giving us an idea about what variables matter. We discussed in lecture how the `varImp()` function can be used to display the variable importance rankings for the random forest model. The function can also be used for `glmnet`.

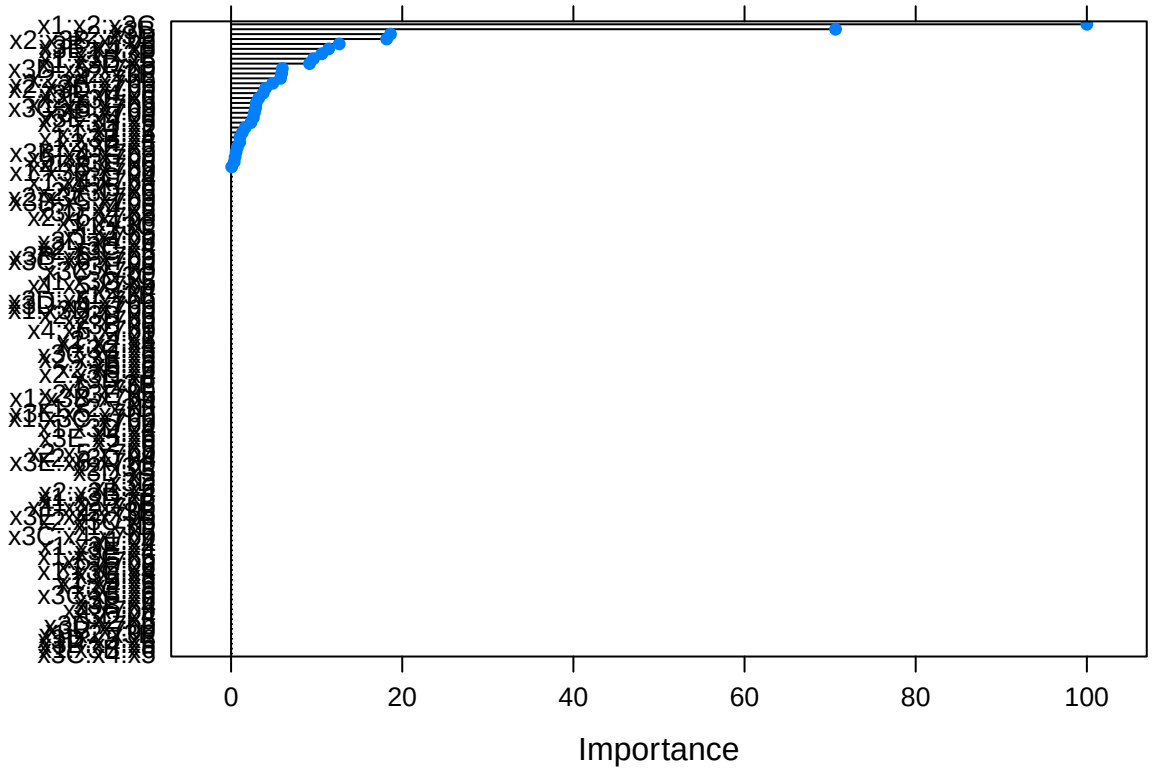
PROBLEM

Call `plot()` on the `varImp()` function applied to the elastic net model with all 3-way interactions. Then call it again with the `top` argument in the `plot()` call set to 25.

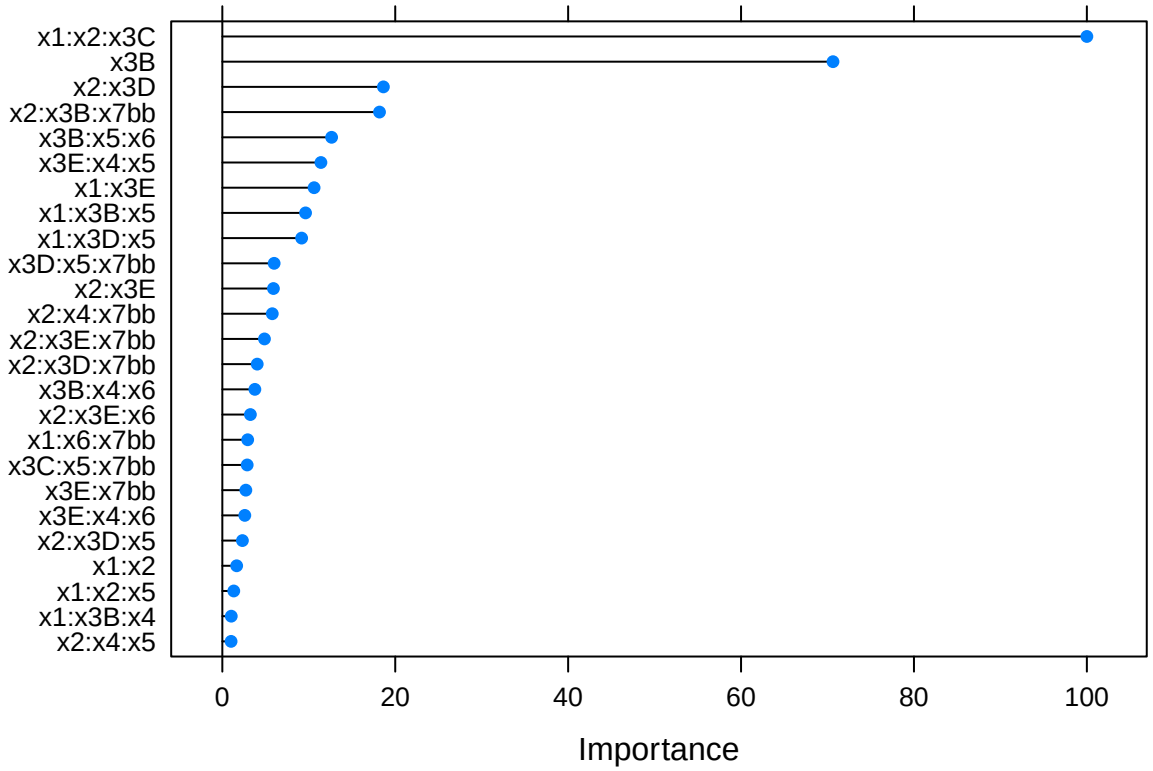
Why does the first plot, which includes all variables, look the way it does? What are the top two important variables according to the elastic net?

SOLUTION

```
### your code here
plot(varImp(fit_glmnet_3))
```




```
### your code here
plot(varImp(fit_glmnet_3), top=25)
```



Why does the first plot, which includes all variables, look the way it does? What are the top two important variables according to the elastic net? ?
The first plot looks congested because it contains all interactions. The two most important variables appear to be x1:x2:x3C and x3B.

Problem 3

Let's start to consider more complex models. You will fit a Support Vector Machine (SVM) with a linear kernel. If you have not used a SVM in `R` before, `caret` will if it can download and install the necessary packages. Follow the instructions within the `R` console and the packages will be installed correctly.

3a)

The linear kernel SVM has a single tuning parameter, the cost `C`. We discussed this parameter in lecture.

PROBLEM

What does the `C` parameter control within the SVM? Also, how many unknown parameters are there for the SVM in this specific problem?

SOLUTION

The C parameter controls the size of the margin surrounding the hyperplane. There are appear to be 7 unknown parameters.

3b)

You will fit the SVM with a linear kernel. You will use a custom tuning grid for the `C` parameter. Even though it's a single variable you must still "wrap" the grid values within the `expand.grid()` call. There are several different SVM's with linear kernals available within the `R` ecosystem. However, you will use the library associated with setting the `method` argument equal to `svmLinear`.

PROBLEM

Specify the grid of possible `C` values to be a regular vector containing 0.25, 0.5, 1.0, and 2.0. Train the SVM with a linear kernel. Set the formula to be the response, `output`, as a function of all other variables in the data set. Set the `method` argument equal to `"svmLinear"`. Set the `tuneGrid` argument equal to the `svm_lin_grid` object. The other arguments should be the same as you used for the logistic regression and elastic net models.

Once the code chunk completes, and it MIGHT TAKE A FEW MINUTES, print the `caret` object to screen. How does the SVM with a linear kernel compare to the best tuned elastic net with all 3-way interactions?

SOLUTION

```
#install.packages("e1071")
```

```
#ctrl <- trainControl(method="repeatedcv",
#                      number = 10,
#                      repeats = 2,
#                      classProbs = TRUE
#                      )

svm_lin_grid <- expand.grid(
                C= c(0.25, 0.5, 1.0, 2.0)
                )

set.seed(98131)
fit_svm_lin <- train(output ~ ., data = train_df,
                 method = "svmLinear",
                 metric = metric_used,
                 tuneGrid = svm_lin_grid,
                 trControl = ctrl_k05)
```

```
## maximum number of iterations reached 0.0005448217 -0.0005439533maximum number of iterations reached
```

```
fit_svm_lin
```

```
## Support Vector Machines with Linear Kernel
##
## 1500 samples
##    7 predictor
##    2 classes: 'event', 'non_event'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 3 times)
```

```
## Summary of sample sizes: 1200, 1200, 1200, 1200, 1200, 1200, ...
## Resampling results across tuning parameters:
##
##   C     ROC        Sens  Spec
##   0.25  0.4907664  0     1
##   0.50  0.4860835  0     1
##   1.00  0.5041490  0     1
##   2.00  0.5038874  0     1
##
## ROC was used to select the optimal model using the largest value.
## The final value used for the model was C = 1.
```

How does the SVM with a linear kernel compare to the best tuned elastic net with all 3-way interactions?** The SVM appears to yield worse ROC values than the elastic net.

3c)

To move beyond linearity, we will use the SVM with a radial basis function. As with the linear kernel, there are multiple libraries to choose from to build this model. You will however use the library associated with the `method` argument equal to `"svmRadial"`. Compared with the linear kernel, the radial basis function kernel has an additional tuning parameter, `sigma`, which controls the behavior of the radial basis or Gaussian kernel.

You will first use the default `caret` search grid. This will allow the underlying package, `kernlab`, to estimate `sigma`, which serves as a useful starting point for refining the grid later.

PROBLEM

Train a SVM with a radial basis function. Use the default `caret` search grid by NOT including the `tuneGrid` argument in the `train()` call. Set the remaining arguments to be the same as what you used in Problem 3b), except set the `method` equal to `"svmRadial"`.

After the training completes, print the `caret` object to the screen. What value of `sigma` was used and what was the optimally identified `C` value?

SOLUTION

```
set.seed(98131)
fit_svm_rbf <- train(output ~ ., data = train_df,
                     method = "svmRadial",
                     metric = metric_used,
                     trControl = ctrl_k05)
```

```
fit_svm_rbf
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 1500 samples
##    7 predictor
##    2 classes: 'event', 'non_event'
##
## No pre-processing
```

```
## Resampling: Cross-Validated (5 fold, repeated 3 times)
## Summary of sample sizes: 1200, 1200, 1200, 1200, 1200, 1200, ...
## Resampling results across tuning parameters:
##
##   C     ROC        Sens       Spec
##   0.25  0.5949714  0.1339031  0.9409836
##   0.50  0.5951832  0.1396011  0.9406193
##   1.00  0.6017997  0.1527066  0.9329690
##
## Tuning parameter 'sigma' was held constant at a value of 0.06695495
## ROC was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 0.06695495 and C = 1.
```

What value of `sigma` was used and what was the optimally identified `C` value?**
?
A sigmna value of approx. 0.067 was used along with a C value of 1.

3d)

You will now use a refined grid based on the results from Problem 3c). You will create a search grid that varies `sigma` as being equal to, double that, four times that, and 8 times the value identified within `fit_svm_rbf`. Can you think of how you can access the best tuned value of `sigma` from the `fit_svm_rbf` object?

PROBLEM

Define the search grid with with the `expand.grid()` function. Set `sigma` to be a vector equal to 1x, 2x, 4x, and 8x times the value identified in Problem 3c). Set the `C` parameter to a vector with elements equal to 0.5, 1, 2, 4, 8, 16, and 32.

Train a new SVM with radial basis function but now use your custom grid search defined in `rbf_grid`. The other arguments should be the same as what you used in Problem 3c).

Once the model training completes, WHICH MIGHT TAKE A FEW MINUTES, print the results to screen and plot the results.

Would continuing to increase the `sigma` or `C` parameters relative to the values from Problem 3c) continue to improve the area under the ROC curve?

SOLUTION

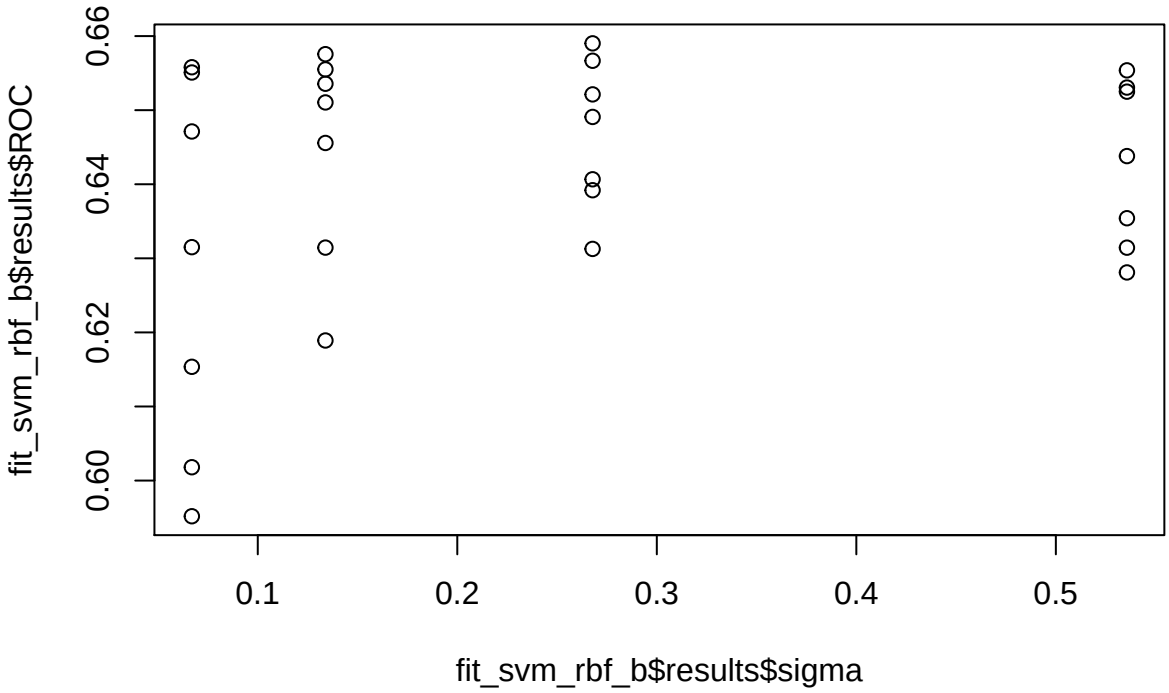
```
rbf_grid <- expand.grid(sigma=c(1*fit_svm_rbf$bestTune$sigma,
                                2*fit_svm_rbf$bestTune$sigma,
                                4*fit_svm_rbf$bestTune$sigma,
                                8*fit_svm_rbf$bestTune$sigma),
                                C=c(0.5, 1, 2, 4, 8, 16, 32) )
```

```
set.seed(98131)
fit_svm_rbf_b <- train(output ~ ., data = train_df,
                  method = "svmRadial",
                  metric = "ROC",
                  tuneGrid = rbf_grid,
                  trControl = ctrl_k05)
```

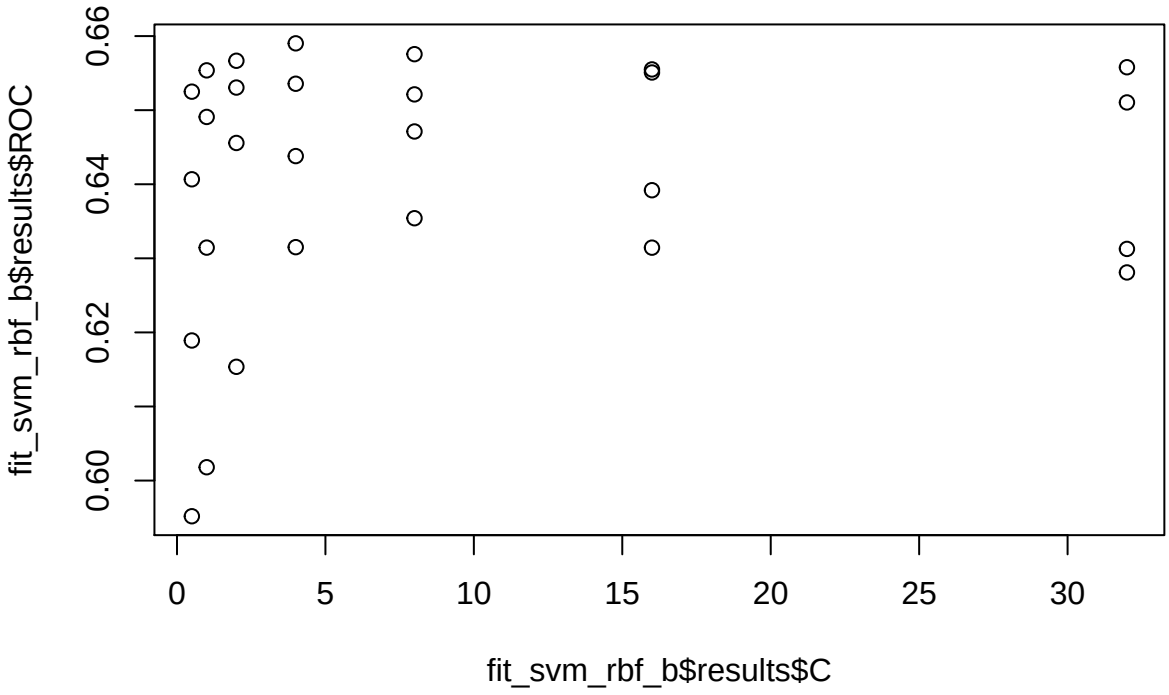
```
fit_svm_rbf_b
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 1500 samples
##    7 predictor
##    2 classes: 'event', 'non_event'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 3 times)
## Summary of sample sizes: 1200, 1200, 1200, 1200, 1200, 1200, ...
## Resampling results across tuning parameters:
##
##   sigma       C     ROC        Sens       Spec
##   0.06695495   0.5  0.5951832  0.1247863  0.9482696
##   0.06695495   1.0  0.6017997  0.1447293  0.9369763
##   0.06695495   2.0  0.6153535  0.1886040  0.9183971
##   0.06695495   4.0  0.6315072  0.1965812  0.9183971
##   0.06695495   8.0  0.6471222  0.2438746  0.9063752
##   0.06695495  16.0  0.6550869  0.2626781  0.9016393
##   0.06695495  32.0  0.6557938  0.2461538  0.9027322
##   0.13390990   0.5  0.6189093  0.1903134  0.9074681
##   0.13390990   1.0  0.6314480  0.2102564  0.9030965
##   0.13390990   2.0  0.6455716  0.2621083  0.8881603
##   0.13390990   4.0  0.6535675  0.2843305  0.8856102
##   0.13390990   8.0  0.6575592  0.2672365  0.8892532
##   0.13390990  16.0  0.6555011  0.2398860  0.9060109
##   0.13390990  32.0  0.6510423  0.2279202  0.9067395
##   0.26781981   0.5  0.6406707  0.2455840  0.8867031
##   0.26781981   1.0  0.6490838  0.2729345  0.8746812
##   0.26781981   2.0  0.6566687  0.3190883  0.8717668
##   0.26781981   4.0  0.6590195  0.2923077  0.8816029
##   0.26781981   8.0  0.6521290  0.2581197  0.8932605
##   0.26781981  16.0  0.6391917  0.2558405  0.8888889
##   0.26781981  32.0  0.6312705  0.2609687  0.8837887
##   0.53563961   0.5  0.6524995  0.3082621  0.8648452
##   0.53563961   1.0  0.6553734  0.3099715  0.8772313
##   0.53563961   2.0  0.6530506  0.3213675  0.8641166
##   0.53563961   4.0  0.6437999  0.2792023  0.8743169
##   0.53563961   8.0  0.6354148  0.2655271  0.8768670
##   0.53563961  16.0  0.6314387  0.2478632  0.8797814
##   0.53563961  32.0  0.6280946  0.2666667  0.8703097
##
## ROC was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 0.2678198 and C = 4.
```

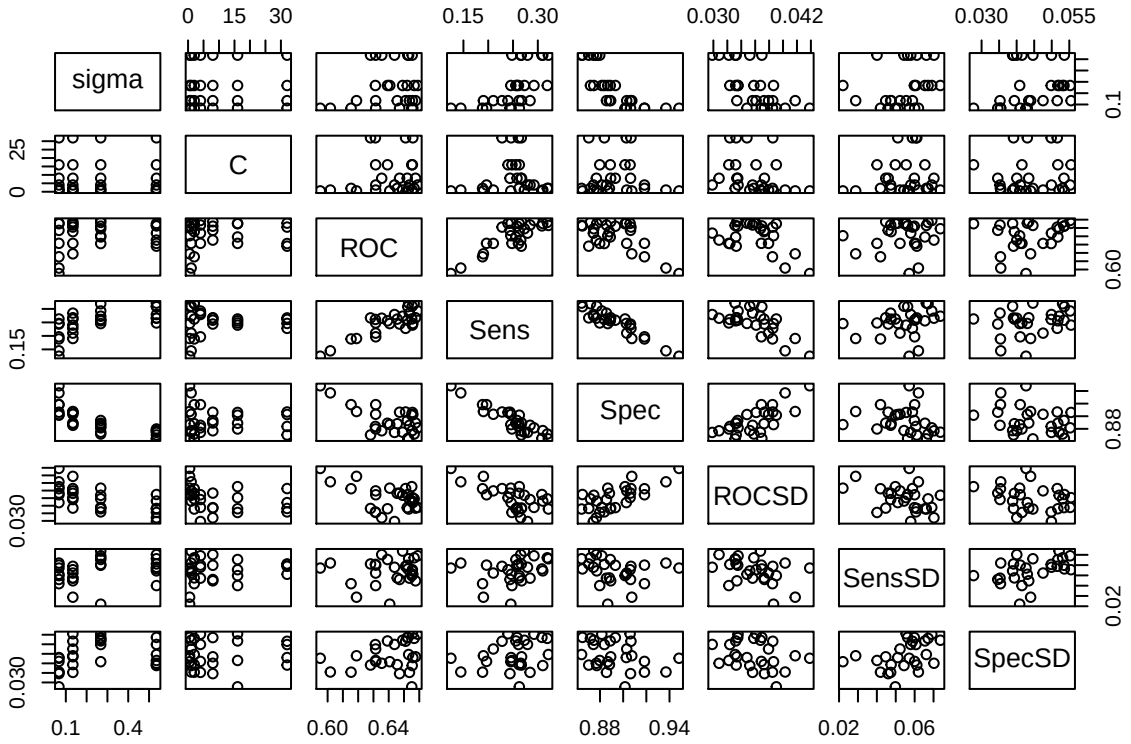
```
### your code here
plot(x=fit_svm_rbf_b$results$sigma, y=fit_svm_rbf_b$results$ROC)
```




```
plot(x=fit_svm_rbf_b$results$C, y=fit_svm_rbf_b$results$ROC)
```




```
plot(fit_svm_rbf_b$results)
```



*Would continuing to increase the `sigma` or `C` parameters relative to the values from Problem 3c) continue to improve the area under the ROC curve? ?
It appears that increasing the sigma parameter would improve the area under the ROC curve more than increasing the C parameter. ## Problem 4

4a)

Up to this point, we have only focused on the area under the ROC curve. However, there are multiple other metrics to consider. You can easily see the confusion matrix performance assessed on the hold-out sets by calling the `confusionMatrix.train()` function on a `caret` model object.

PROBLEM

Call `confusionMatrix.train()` on the custom search grid elastic net model with all 3-way interactions and the custom search grid SVM with radial basis function. Which model has the higher hold-out average accuracy? Which model is better at predicting the `"event"` when the `"event"` actually occurs?

SOLUTION

```
### your code here
confusionMatrix.train(fit_glmnet_3)
```

```
## Cross-Validated (5 fold, repeated 3 times) Confusion Matrix
##
```

```
## (entries are percentual average cell counts across resamples)
##
##            Reference
## Prediction  event non_event
##   event       7.1       2.6
##   non_event  31.9      58.4
##
##  Accuracy (average) : 0.6551
```

```
### your code here
confusionMatrix.train(fit_svm_rbf)
```

```
## Cross-Validated (5 fold, repeated 3 times) Confusion Matrix
##
## (entries are percentual average cell counts across resamples)
##
##            Reference
## Prediction  event non_event
##   event       6.0       4.1
##   non_event  33.0      56.9
##
##  Accuracy (average) : 0.6287
```

Which model has the higher hold-out average accuracy? Which model is better at predicting the `"event"` when the `"event"` actually occurs? ** ?
The elastic-net model appears to have higher average accuracy than the svm-rbf. The elastic-net model appears to also be better at predicting the event when it occurs (corresponding to higher sensitivity).

4b)

The previous question was alluding to the Sensitivity or True Positive (TP) rate of the models. You might have noticed that in the `caret` model object print outs, there are two other columns next to ROC: Sens, and Spec. Sens is the Sensitivity, or TP rate, and Spec is the Specificity which is related to the False Positive (FP) rate. We will now go through understanding the Sensitivity and Specificity in relationship to the ROC curve itself.

You will need to work with the hold-out set predictions directly to accomplish this. As discussed in lecture, the predictions are contained within the `$pred` data.frame within the`caret` model object. The first 11 rows associated with the hold-out set predictions for the `fit_svm_rbf_b` object are printed to screen for you in the code chunk below.

```
fit_svm_rbf_b$pred %>% tbl_df() %>% head(11)
```

```
## # A tibble: 11 x 8
##    pred      obs       event non_event rowIndex  sigma     C Resample
##    <fct>     <fct>     <dbl>     <dbl>    <int>  <dbl> <dbl> <chr>
##  1 non_event non_event 0.345     0.655        4 0.0670   0.5 Fold1.Rep1
##  2 non_event event     0.360     0.640        7 0.0670   0.5 Fold1.Rep1
##  3 non_event event     0.383     0.617       10 0.0670   0.5 Fold1.Rep1
##  4 non_event non_event 0.352     0.648       11 0.0670   0.5 Fold1.Rep1
##  5 non_event event     0.418     0.582       23 0.0670   0.5 Fold1.Rep1
##  6 non_event non_event 0.356     0.644       27 0.0670   0.5 Fold1.Rep1
```

```
##  7 non_event non_event 0.381     0.619       29 0.0670   0.5 Fold1.Rep1
##  8 non_event event     0.334     0.666       31 0.0670   0.5 Fold1.Rep1
##  9 non_event event     0.393     0.607       33 0.0670   0.5 Fold1.Rep1
## 10 non_event event     0.422     0.578       34 0.0670   0.5 Fold1.Rep1
## 11 non_event event     0.389     0.611       41 0.0670   0.5 Fold1.Rep1
```

The last column `Resample` is an identifer for the resample Fold ID (in this case). The code chunk below counts the number of rows in the hold-out prediction object associated with each Fold.

```
fit_svm_rbf_b$pred %>% tbl_df() %>%
  count(Resample)
```

```
## # A tibble: 15 x 2
##    Resample       n
##    <chr>      <int>
##  1 Fold1.Rep1  8400
##  2 Fold1.Rep2  8400
##  3 Fold1.Rep3  8400
##  4 Fold2.Rep1  8400
##  5 Fold2.Rep2  8400
##  6 Fold2.Rep3  8400
##  7 Fold3.Rep1  8400
##  8 Fold3.Rep2  8400
##  9 Fold3.Rep3  8400
## 10 Fold4.Rep1  8400
## 11 Fold4.Rep2  8400
## 12 Fold4.Rep3  8400
## 13 Fold5.Rep1  8400
## 14 Fold5.Rep2  8400
## 15 Fold5.Rep3  8400
```

The reason why it seems like there are so many predictions, is because of all of the `C` and `sigma` combinations that were tried! To simplify things, you will focus just on the best tuned case when creating the ROC curves.

PROBLEM

Complete the code chunk below which filters the hold-out prediction object for the SVM radial basis kernel to focus just on the best tuned `C` and `sigma` values. In addition, you must focus just on `"Fold1"`. The result is stored to the `best_svm_rbf_F1_pred` object.

How can you check if the resulting object has the correct number of rows?

HINT: How can you access the best tuned `C` and `sigma` values?

SOLUTION

```
best_svm_rbf_F1_pred <- fit_svm_rbf_b$pred %>% tbl_df() %>%
  filter(C == fit_svm_rbf$bestTune$C,
         sigma == fit_svm_rbf$bestTune$sigma,
         Resample == "Fold1.Rep1")

best_svm_rbf_F1_pred
```

```
## # A tibble: 300 x 8
##    pred      obs       event non_event rowIndex  sigma     C Resample
##    <fct>     <fct>     <dbl>     <dbl>    <int>  <dbl> <dbl> <chr>
##  1 non_event non_event 0.319     0.681        4 0.0670     1 Fold1.Rep1
##  2 non_event event     0.343     0.657        7 0.0670     1 Fold1.Rep1
##  3 non_event event     0.378     0.622       10 0.0670     1 Fold1.Rep1
##  4 non_event non_event 0.330     0.670       11 0.0670     1 Fold1.Rep1
##  5 non_event event     0.436     0.564       23 0.0670     1 Fold1.Rep1
##  6 non_event non_event 0.336     0.664       27 0.0670     1 Fold1.Rep1
##  7 non_event non_event 0.376     0.624       29 0.0670     1 Fold1.Rep1
##  8 non_event event     0.304     0.696       31 0.0670     1 Fold1.Rep1
##  9 non_event event     0.394     0.606       33 0.0670     1 Fold1.Rep1
## 10 non_event event     0.441     0.559       34 0.0670     1 Fold1.Rep1
## # ... with 290 more rows
```

```
count(tbl_df(best_svm_rbf_F1_pred$Resample))
```

```
## # A tibble: 1 x 1
##       n
##   <int>
## 1   300
```

```
best_svm_rbf_F1_pred %>% tbl_df() %>%
  count(Resample)
```

```
## # A tibble: 1 x 2
##   Resample       n
##   <chr>      <int>
## 1 Fold1.Rep1   300
```

How can you check if the resulting object has the correct number of rows?
?
By counting the Resample rows and comparing with the filter results. Since there are 1500 observations and we used 5 folds, each should contain 300 observations.

4c)

The `confusionMatrix.train()` function is specific to `caret` trained model objects. However, `caret` includes a more general function `confusionMatrix()` which requires at minimum two input arguments, the predicted class and the observed/target/reference class. The generic syntax is shown below:

`confusionMatrix(<predicted class vector>, <observed/target/refernce class vector>)`

This call produces a confusion matrix that returns counts rather than percentages. Substantial more information is printed to screen as well, including the 95% confidence interval on the Accuracy, the Senstivity, and the Specificity. The funciton also tells you which class label (`"event"` or `"non_event"` in this problem) is considered to be the "Positive" class within the confusion matrix.

PROBLEM

Call the `confusionMatrix()` function on the radial basis kernel SVM by passing in the predicted class and observed class correctly. How many False-Positive counts were observed in this fold?

After calling the function and printing the results to screen, call the `confusionMatrix()` function a second time, but this time save the results to the variable `best_svm_rbf_F1_cm`. Doing this allows you to access the confusion matrix results programmatically. To access the Sensitivity, you would use `best_svm_rbf_F1_cm$byClass["Sensitivity"]`. Print the Sensitivity to the screen and check it is the same as what was displayed when you called the `confusionMatrix()` function directly.

SOLUTION

```
### your code here
confusionMatrix(best_svm_rbf_F1_pred$pred, best_svm_rbf_F1_pred$obs)
```

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction  event non_event
##   event        20         7
##   non_event    97       176
##
##                Accuracy : 0.6533
##                  95% CI : (0.5965, 0.7071)
##     No Information Rate : 0.61
##     P-Value [Acc > NIR] : 0.06872
##
##                   Kappa : 0.1541
##
##  Mcnemar's Test P-Value : < 2e-16
##
##             Sensitivity : 0.17094
##             Specificity : 0.96175
##          Pos Pred Value : 0.74074
##          Neg Pred Value : 0.64469
##              Prevalence : 0.39000
##          Detection Rate : 0.06667
##    Detection Prevalence : 0.09000
##       Balanced Accuracy : 0.56634
##
##        'Positive' Class : event
##
```

```
best_svm_rbf_F1_pred
```

```
## # A tibble: 300 x 8
##    pred      obs       event non_event rowIndex  sigma     C Resample
##    <fct>     <fct>     <dbl>     <dbl>    <int>  <dbl> <dbl> <chr>
##  1 non_event non_event 0.319     0.681        4 0.0670     1 Fold1.Rep1
##  2 non_event event     0.343     0.657        7 0.0670     1 Fold1.Rep1
##  3 non_event event     0.378     0.622       10 0.0670     1 Fold1.Rep1
##  4 non_event non_event 0.330     0.670       11 0.0670     1 Fold1.Rep1
##  5 non_event event     0.436     0.564       23 0.0670     1 Fold1.Rep1
##  6 non_event non_event 0.336     0.664       27 0.0670     1 Fold1.Rep1
##  7 non_event non_event 0.376     0.624       29 0.0670     1 Fold1.Rep1
```

```
##  8 non_event event     0.304     0.696       31 0.0670     1 Fold1.Rep1
##  9 non_event event     0.394     0.606       33 0.0670     1 Fold1.Rep1
## 10 non_event event     0.441     0.559       34 0.0670     1 Fold1.Rep1
## # ... with 290 more rows
```

```
best_svm_rbf_F1_cm = confusionMatrix(best_svm_rbf_F1_pred$pred, best_svm_rbf_F1_pred$obs)
best_svm_rbf_F1_cm$byClass["Sensitivity"]
```

```
## Sensitivity
##   0.1709402
```

How many False-Positive counts were observed in this fold?** ?
There were 7 false positives.

4d)

You will now plot the ROC curve associated with Fold 1 and the best tuned radial basis function SVM. You will use the `plotROC` package set of functions discussed in lecture to generate the ROC curve. Remember that within the `plotROC` syntax, you must set the probability of the EVENT of interest to the `m` aesthetic within the parent `ggplot()` call. You must also set the **observed** EVENT of interest as a 1 (if it occurred) and a 0 if it did not occur to the `d` aesthetic within the parent `ggplot()` call. The ROC curve itself is created by the `geom_roc()` function, where you can specify other grouping variables (such as through the `color` aesthetic) and the number of "cuts" or specific "cut" values to display. The "cut" corresponds to the probability threshold value used to classify the EVENT.

PROBLEM

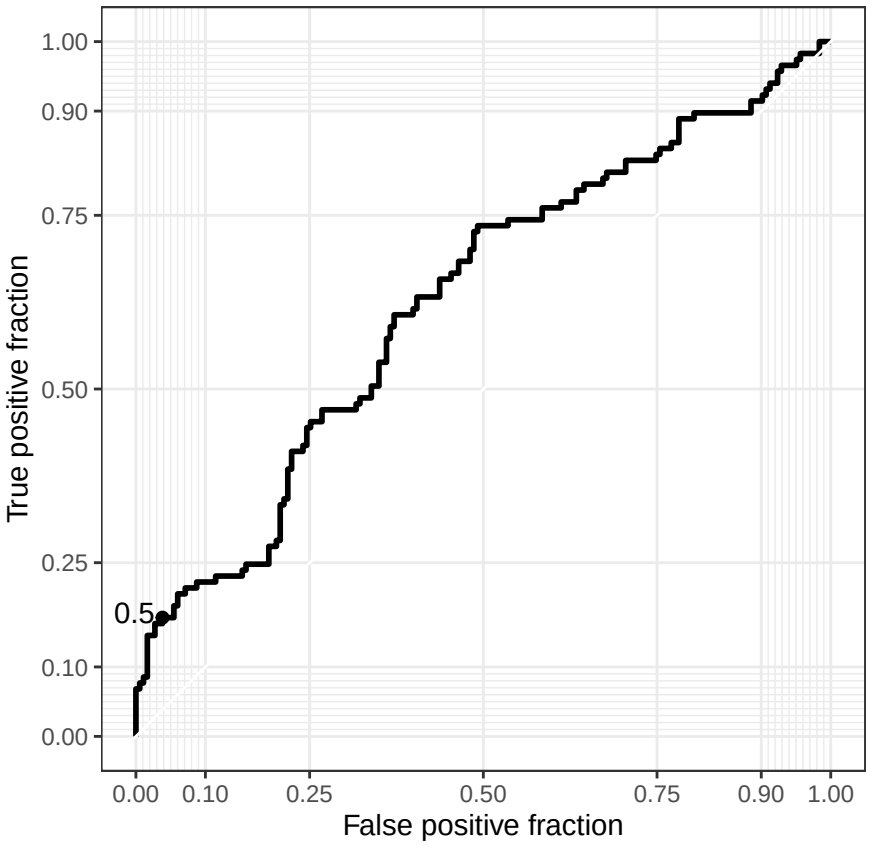
Create the ROC curve using the variables contained within `best_svm_rbf_F1_pred` object. Specify the `m` and `d` aesthetics within the parent `ggplot()` call correctly. Specify the `cutoffs.at` argument within the `geom_roc()` call to be equal to 0.5. After the `geom_roc()` call, use the `coord_equal()` and `style_roc()` function calls.

SOLUTION

```
best_svm_rbf_F1_pred %>% tbl_df()
```

```
## # A tibble: 300 x 8
##    pred      obs       event non_event rowIndex  sigma     C Resample
##    <fct>     <fct>     <dbl>     <dbl>    <int>  <dbl> <dbl> <chr>
##  1 non_event non_event 0.319     0.681        4 0.0670     1 Fold1.Rep1
##  2 non_event event     0.343     0.657        7 0.0670     1 Fold1.Rep1
##  3 non_event event     0.378     0.622       10 0.0670     1 Fold1.Rep1
##  4 non_event non_event 0.330     0.670       11 0.0670     1 Fold1.Rep1
##  5 non_event event     0.436     0.564       23 0.0670     1 Fold1.Rep1
##  6 non_event non_event 0.336     0.664       27 0.0670     1 Fold1.Rep1
##  7 non_event non_event 0.376     0.624       29 0.0670     1 Fold1.Rep1
##  8 non_event event     0.304     0.696       31 0.0670     1 Fold1.Rep1
##  9 non_event event     0.394     0.606       33 0.0670     1 Fold1.Rep1
## 10 non_event event     0.441     0.559       34 0.0670     1 Fold1.Rep1
## # ... with 290 more rows
```

```
### the lecture slides from week 12 help with
### creating ROC curves from the plotROC package
best_svm_rbf_F1_pred %>%
ggplot(mapping=aes(m = event,
                   d=ifelse(obs=="event",
                            1,
                            0))) +
geom_roc(cutoffs.at = 0.5) + coord_equal() + style_roc()
```



4e)

In Problem 4c) you saved the confusion matrix calculations to an object, `best_svm_rbf_F1_cm`. This allows you to programmatically access the Sensitivity and Specificity associated with the results. To help intepret the ROC curve, you will now plot the Sensitivity as a red dashed horizontal line, and 1 - Specificity as a red dashed vertical line.

PROBLEM

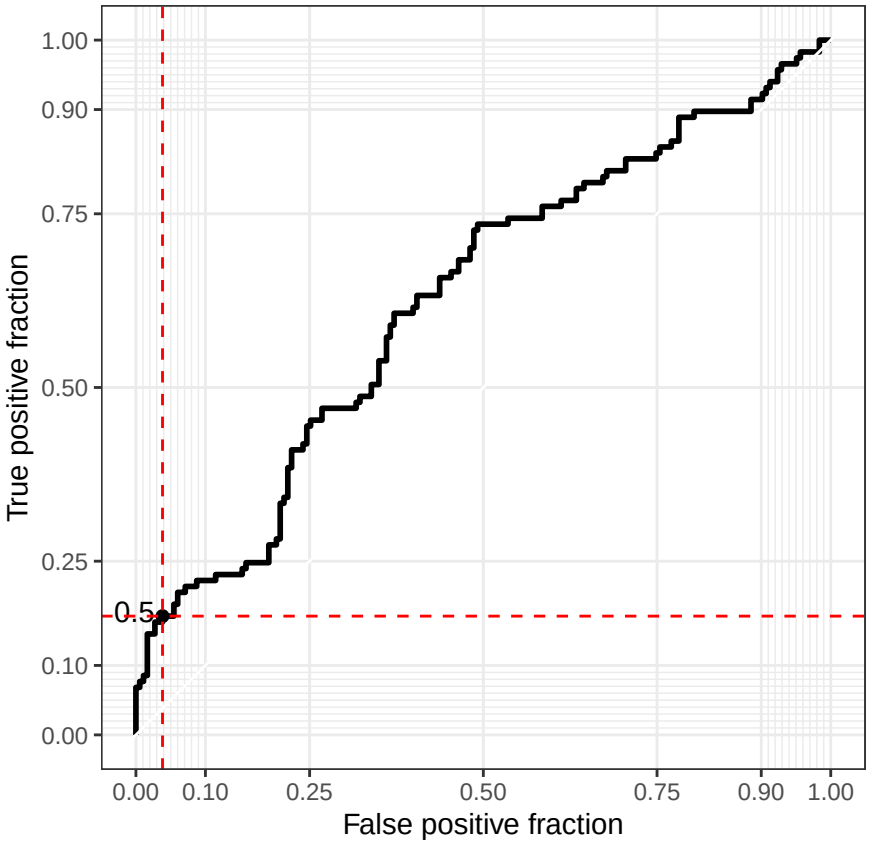
Remake the ROC curve from Problem 4d). This time include the `geom_hline()` and `geom_vline()` before the `coord_equal()` "layer". Within `geom_hline()` set the `yintercept` argument equal to the Sensitivity. Within `geom_vline()` set the `xintercept` argument equal to 1 - Specificity. Within both `geom_hlin()` and `geom_vline()` set the `color` argument to `"red"` and the `linetype` argument to `"dashed"`.

Based on the resulting figure, what threshold value do you think is used to calculate the reported Sensitivity and Specificity values in the `confusionMatrix()` function?

HINT: Look back at how to access the Sensitivity from the saved confusion matrix object.

SOLUTION

```
### your code here
best_svm_rbf_F1_pred %>%
ggplot(mapping=aes(m = event,
                   d=ifelse(obs=="event",
                            1,
                            0))) +
                   geom_roc(cutoffs.at = 0.5) +
                   geom_hline(yintercept = best_svm_rbf_F1_cm$byClass["Sensitivity"], color = "red", li
                   geom_vline(xintercept = 1 - best_svm_rbf_F1_cm$byClass["Specificity"], color = "red"
                   coord_equal() + style_roc()
```



Based on the resulting figure, what threshold value do you think is used to calculate the reported Sensitivity and Specificity values in the `confusionMatrix()` function? ?
0.5 appears to be the threshold value used to calculate the Sensitivity and Specificity.

4f)

The ROC curve you created in the previous problems corresponds to just a single Fold. In this problem, you will repeat the previous set of actions, but this time for fold 4.

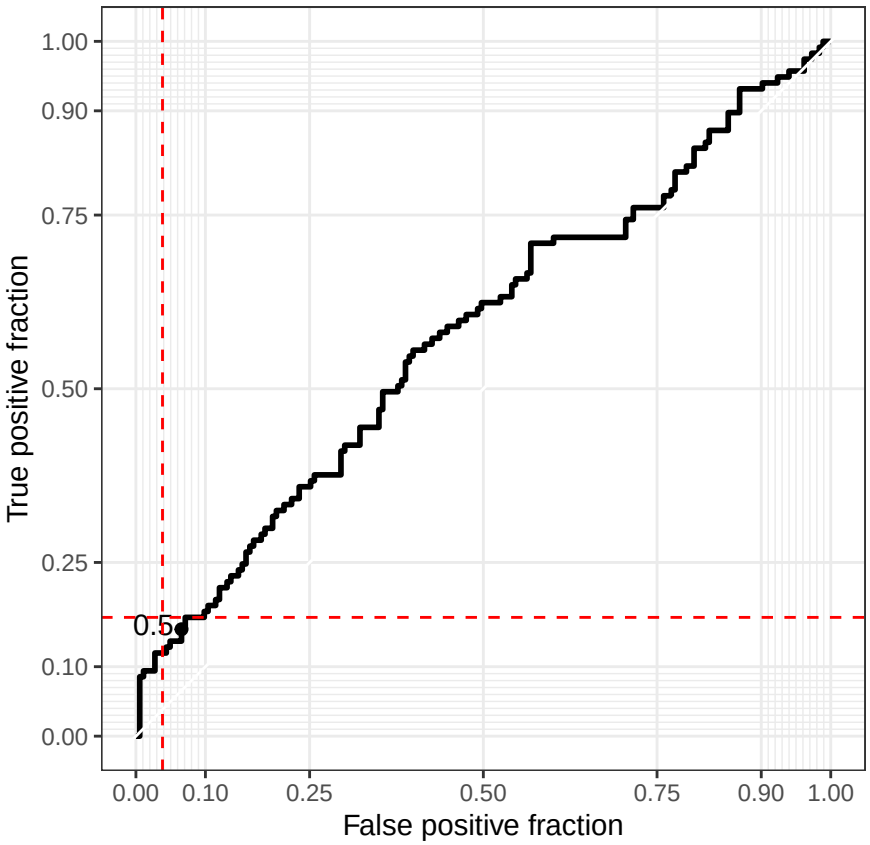
PROBLEM

Create the `best_svm_rbf_F4_pred` object by filtering to just the fourth fold associated with the best tuning parameter values. Calculate the confusion matrix and store to `best_svm_rbf_F4_cm`. Generate the ROC curve and include the Sensitivity and 1 - Specificity as horizontal and vertical dashed lines, respectively.

How does the fourth fold compare to the performance of the first fold you visualized previously?

SOLUTION

```
best_svm_rbf_F4_pred <- fit_svm_rbf_b$pred %>% tbl_df() %>%
  filter(C == fit_svm_rbf$bestTune$C,
         sigma == fit_svm_rbf$bestTune$sigma,
         Resample == "Fold4.Rep1")

best_svm_rbf_F4_cm <- confusionMatrix(best_svm_rbf_F4_pred$pred, best_svm_rbf_F4_pred$obs)

### create the ROC curve for the 4-th fold
best_svm_rbf_F4_pred %>%
ggplot(mapping=aes(m = event,
                   d=ifelse(obs=="event",
                            1,
                            0))) +
                   geom_roc(cutoffs.at = 0.5) +
                   geom_hline(yintercept = best_svm_rbf_F1_cm$byClass["Sensitivity"], color = "red", li
                   geom_vline(xintercept = 1 - best_svm_rbf_F1_cm$byClass["Specificity"], color = "red"
                   coord_equal() + style_roc()
```



?
The 4th fold appears to be straighter than the previous.

4g)

Let's now compare the variability in the ROC curves across the folds to the overall averaged ROC curve for the best tuning parameter values. To create this figure, you will need to make use of the `color` aesthetic within the `geom_roc()` call.

PROBLEM

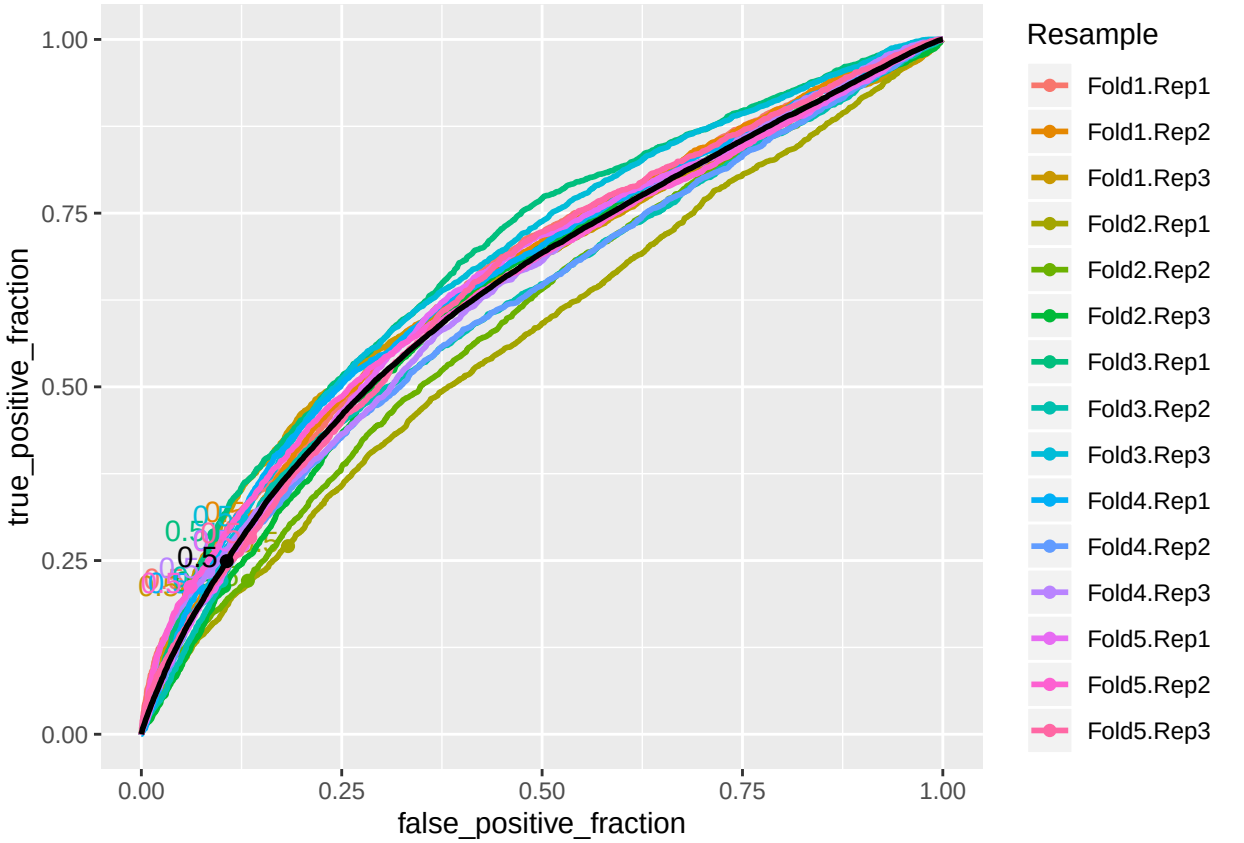
Create the `best_svm_rbf_pred` object which filters the tuning parameters to equal their best identified values. You will not filter any of the Resample folds. Use `best_svm_rbf_pred` with two different `geom_roc()` calls. In the first call, set the `color` aesthetic equal to `Resample`. In the second `geom_roc()` do not use any additional aesthetics. In both `geom_roc()` calls set the `cutoff.at` argument equal to 0.5. Do not include any additional reference lines.

The colored ROC curves correspond to the separate Resample folds. How does the overall average ROC curve compare to the fold-to-fold variability? Which fold appears to have the worse performance?

SOLUTION

```
best_svm_rbf_pred <- fit_svm_rbf_b$pred %>% tbl_df() %>%
  filter()
```

```
### create the ROC curves comparing the separate resample
### folds with the resample AVERAGED ROC curve
best_svm_rbf_pred %>%
ggplot(mapping=aes(m = event,
                   d=ifelse(obs=="event",
                            1,
                            0))) +
                   geom_roc(mapping=aes(color = Resample), cutoffs.at = 0.5) +
                   geom_roc(cutoffs.at = 0.5)
```



?
How does the overall average ROC curve compare to the fold-to-fold variability? Which fold appears to have the worse performance?**

The overall ROC curve appears to be thicker than those with fold-to-fold variability. The green colored fold (2) appears to have the best performance since it's area is greatest.

4h)

Let's now compare the ROC curves between logistic regression, elastic net, SVM with linear kernel, and SVM with radial basis kernel. You have the required information contained within `best_svm_rbf_pred` object to create the figure, but not all `caret $pred` objects will be the same. The number of columns will vary based upon the number of tuning parameters. Therefore you need to take a few additional steps to enable merging together the results.

The code chunk below demonstrates how to do that with the results from the logistic regression model. The logistic regression model does not include any tuning parameters, and so it coincides with the "best" tuned model without requiring additional filtering steps. The code chunk uses the `dplyr::select()` function to grab the minimum set of columns needed to make the ROC curve. It then uses the `mutate()` function to create a new column `model_name` which is set to `"GLM"`. The result is saved to the `best_glm_cv_pred`.

```
best_glm_cv_pred <- fit_glm$pred %>% tbl_df() %>%
  dplyr::select(pred, obs, event, non_event, rowIndex, Resample) %>%
  mutate(model_name = "GLM")
```

PROBLEM

You will follow the logistic regression example and isolate the best tuning parameter values and put the results into the correct format for the elastic net with all 3-way interactions, the SVM with linear kernel, and the SVM with radial basis kernel.

Name the elastic net model `"GLMNET-3"`. Name the SVM with linear kernel `"SVM-LIN"`. Name the SVM with radial basis kernel `"SVM-RBF"`.

SOLUTION

```
best_glmnet_3_cv_pred <- fit_glmnet_3_b$pred %>% tbl_df() %>%
  filter(alpha==fit_glmnet_3_b$bestTune$alpha, lambda==fit_glmnet_3_b$bestTune$lambda)%>% mutate(model_

best_svm_lin_cv_pred <- fit_svm_lin$pred %>% tbl_df() %>%
  filter(C == fit_svm_lin$bestTune$C)%>% mutate(model_name = "SVM-LIN")

best_svm_rbf_cv_pred <- fit_svm_rbf_b$pred %>% tbl_df() %>%
  filter(sigma == fit_svm_rbf_b$bestTune$sigma,
           C == fit_svm_rbf_b$bestTune$C)%>% mutate(model_name = "SVM-RBF")
```

4i)

The code chunk below combines all of the best tuning parameter hold-out set predictions together.

```
best_mod_cv_pred_a <- best_glm_cv_pred %>%
  bind_rows(best_glmnet_3_cv_pred) %>%
  bind_rows(best_svm_lin_cv_pred) %>%
  bind_rows(best_svm_rbf_cv_pred)
```

The `best_mod_cv_pred_a` object is now in a format suitable for generating separate resample averaged ROC curves for the different models.

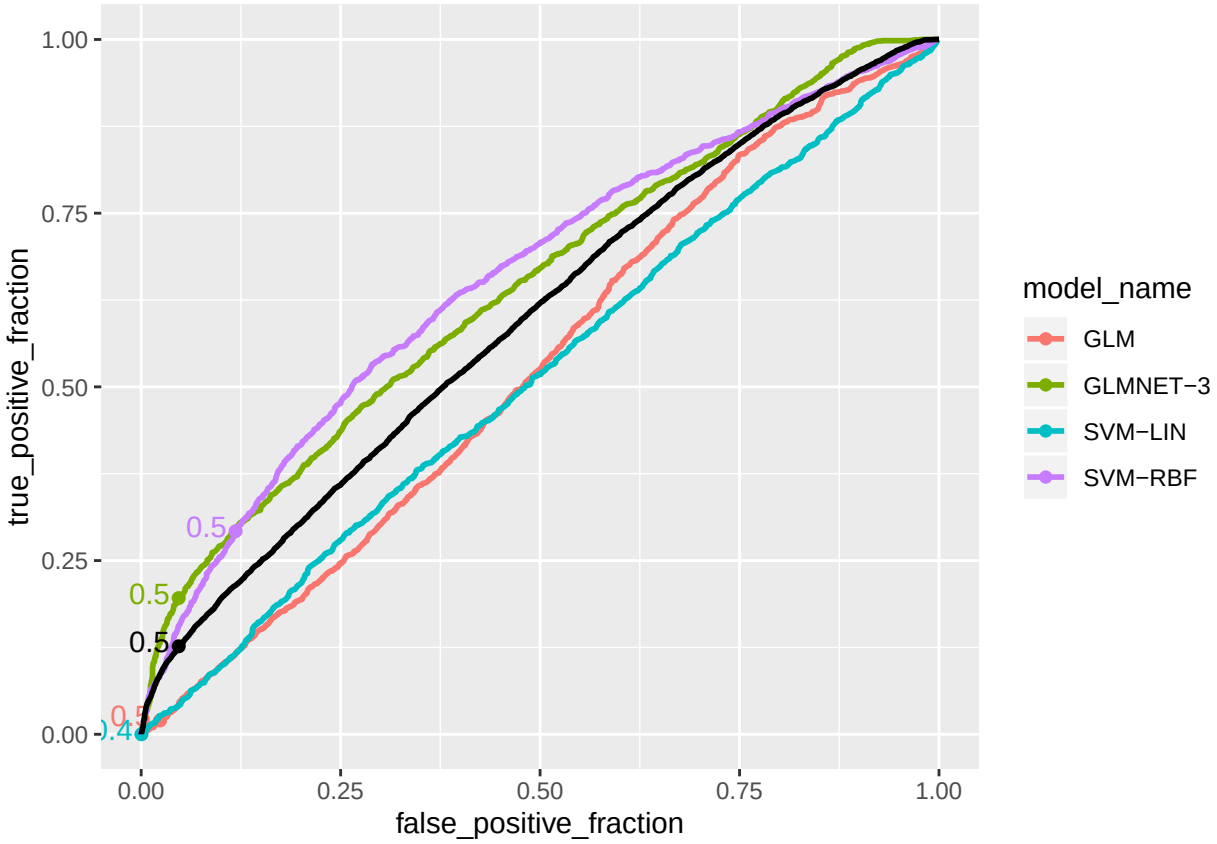
PROBLEM

Create the ROC curves for each model by correctly specifying the `m` and `d` aesthetics within the parent `ggplot()` call. Set the `color` aesthetic within `geom_roc()` equal to `model_name`, and set the `cutoff.at` argument equal to 0.5.

Which model appears the best, based on the Resample averaged ROC curves?

SOLUTION

```
### compare the models by comparing their resample
### averaged ROC curves!!!
best_mod_cv_pred_a %>%
    ggplot(mapping = aes(m = event, d = ifelse(obs == 'event',1,0))) +
    geom_roc(mapping = aes(color = model_name), cutoffs.at = 0.5) +
    geom_roc(cutoffs.at = 0.5)
```



Which model appears the best, based on the Resample averaged ROC curves?

?
The SVM-RBF model appears to be best.

Problem 5

You will now train several more complex models. Let's first start with a neural network to predict the binary outcome. You will set the `method` argument equal to `"nnet"` in the call to `train()`. If this is the first you have used `nnet caret` will ask if the necessary packages can be downloaded. Check the `R` console for the prompting questions.

5a)

PROBLEM

Use the `caret` default grid search to tune a neural network model. Besides changing the `method` argument to `"nnet"`, the other arguments should be consistent with the other problems.

However, it is recommended that you set the `trace` argument to `FALSE`. If you do not all of the iterations will be printed to screen and therefore to your rendered PDF file. SET `trace` to `FALSE`.

After training is complete, WHICH MIGHT TAKE SEVERAL MINUTES, print the `caret` object to screen. Which set of tuning parameters produce the best performing model?

SOLUTION

```
set.seed(98131)
fit_nnet_a <- train(output ~ . ,
                    method ="nnet",
                    data = train_df,
                    metric = metric_used,
                    trControl = ctrl_k05,
                    trace = FALSE
                    )
```

```
fit_nnet_a
```

```
## Neural Network
##
## 1500 samples
##    7 predictor
##    2 classes: 'event', 'non_event'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 3 times)
## Summary of sample sizes: 1200, 1200, 1200, 1200, 1200, 1200, ...
## Resampling results across tuning parameters:
##
##   size  decay  ROC        Sens        Spec
##   1     0e+00  0.5364890  0.22336182  0.8313297
##   1     1e-04  0.5477512  0.17834758  0.8732240
##   1     1e-01  0.5137515  0.08205128  0.9362477
##   3     0e+00  0.5772500  0.33105413  0.7715847
##   3     1e-04  0.5961468  0.33675214  0.7970856
##   3     1e-01  0.6823471  0.41595442  0.8269581
##   5     0e+00  0.6298289  0.40740741  0.7730419
##   5     1e-04  0.6251989  0.42621083  0.7519126
##   5     1e-01  0.7146451  0.47806268  0.8069217
##
## ROC was used to select the optimal model using the largest value.
## The final values used for the model were size = 5 and decay = 0.1.
```

?
The parameters with size = 5 appear to produce the best result with ROC = 0.715.

5b)

As you should see from Problem 5a), the tuning parameters to a neural network from `nnet` are `size` and `decay`. `size` is the number of hidden units within the hidden layer, and `decay` is the weight decay of

the parameters. Weight decay provides regularization effects to the parameters. The larger the `decay` the stronger a regularizing effect.

You will now try a custom grid search to study what happens if more hidden units are used within the model.

PROBLEM

Create a grid, `nnet_grid`, with the `expand.grid()` function. Set the `size` variable to a vector with values of 3, 5, 7, 9, 11, 13, 15. Set the `decay` variable within the grid to be a vector with values 0.1, 0.25, 0.5, and 0.75 Train the neural network model by setting the `tuneGrid` argument equal to `nnet_grid`. BE SURE TO SET `trace` equal to `FALSE`.

Once the training completes, WHICH MIGHT TAKE SEVERAL MINUTES, plot the results using the `plot()` function and print the best tuning values to screen.

SOLUTION

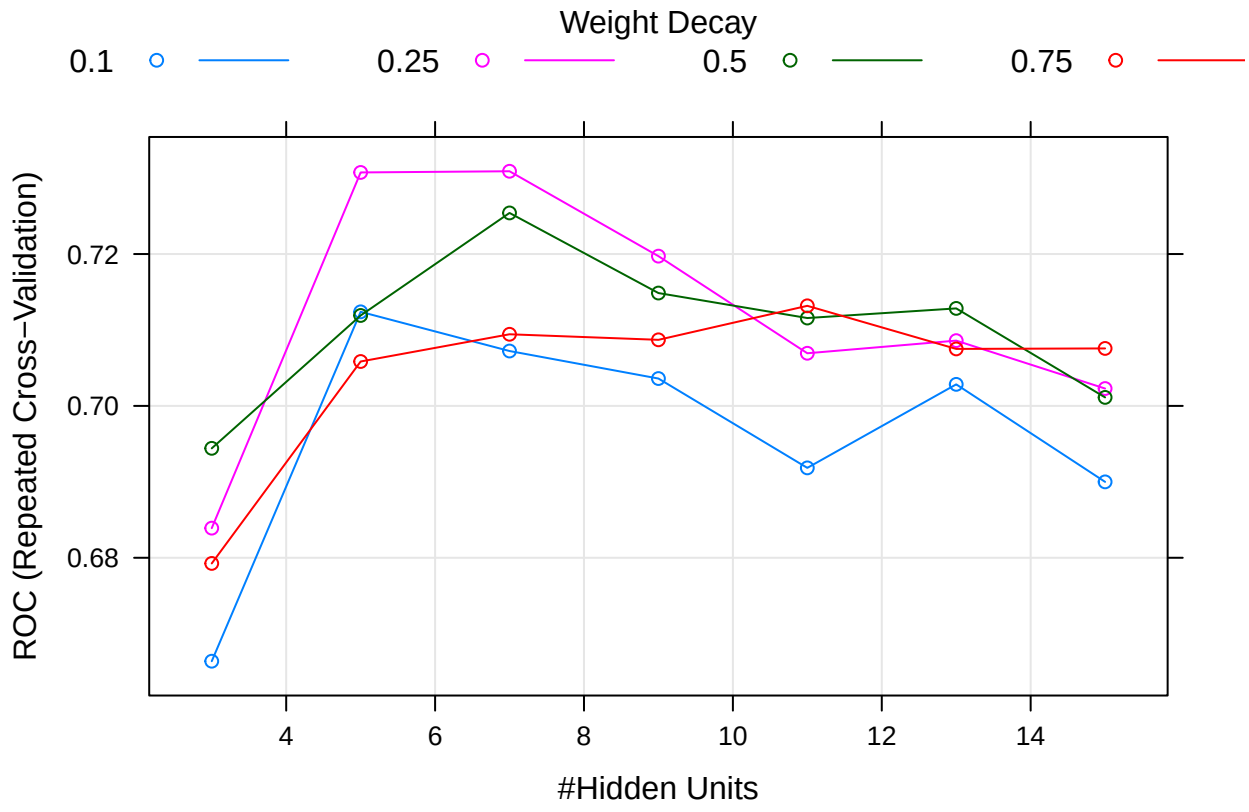
```
nnet_grid <- expand.grid(size = c(3, 5, 7, 9, 11, 13, 15),
                         decay = c(0.1, 0.25, 0.5, 0.75))
```

```
set.seed(98131)
fit_nnet_b <- train(output ~ . ,
                    method ="nnet",
                    data = train_df,
                    metric = metric_used,
                    trControl = ctrl_k05,
                    trace = FALSE,
                    tuneGrid = nnet_grid
                    )
```

```
### print out the best tuning parameters
### and plot the results

fit_nnet_b$bestTune
```

```
##    size decay
## 10    7  0.25
```

```
plot(fit_nnet_b)
```



5c)

PROBLEM

Display the confusion matrix percentages associated with the training results. How does the neural network model compare in accuracy with the other models you have built so far?

SOLUTION

```
### your code here
confusionMatrix(fit_nnet_b)
```

```
## Cross-Validated (5 fold, repeated 3 times) Confusion Matrix
##
## (entries are percentual average cell counts across resamples)
##
##            Reference
## Prediction  event non_event
##   event      19.6      11.6
##   non_event  19.4      49.4
##
##  Accuracy (average) : 0.6902
```

How does the neural network model compare in accuracy with the other models you have built so far?** ? The neural net model appears to be slightly better in accuracy than the rest but, in general, as good the other models.

5d)

Next, you will visualize the ROC curve associated with the best neural network model.

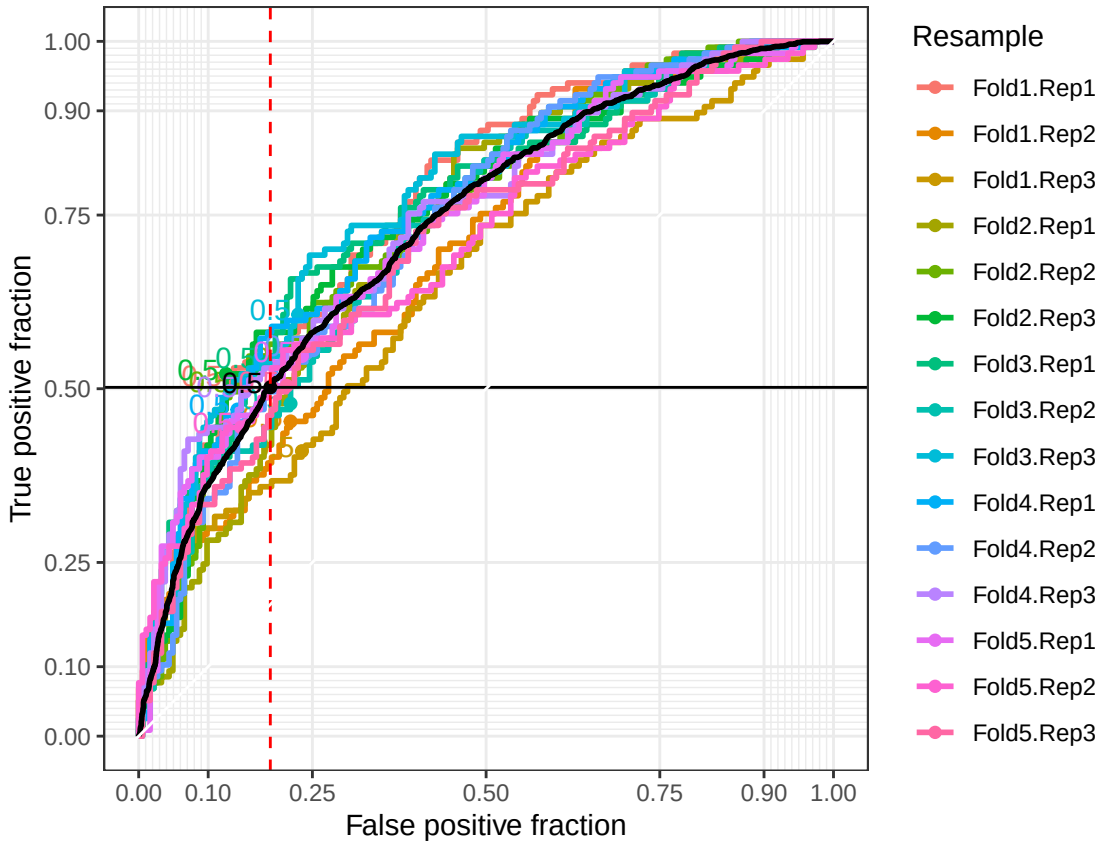
PROBLEM

Filter the hold-out set predictions for the best neural network model. Do not filter based on the Resample. Store the results to `best_nnet_cv_pred`. Follow the same format as the `best_glm_cv_pred` object, by selecting a specific set of the variables and create a new column, `model_name = "NNET"`. Use the variables within `best_nnet_cv_pred` with the `confusionMatrix()` function and store the results to `best_nnet_cm`. Create the Resample averaged ROC curve and compare to the individual Resample Folds ROC curves. For reference, plot the Sensitivity and 1 - Specificity as horizontal and vertical red dashed lines.

How does the Resample averaged ROC curve relate to the individual Fold ROC curves for the neural network model?
How does the location of the 50% threshold compare to that for the other models?

SOLUTION

```
best_nnet_cv_pred <- fit_nnet_b$pred %>% tbl_df() %>%
  filter(size == fit_nnet_b$bestTune$size,
         decay == fit_nnet_b$bestTune$decay)%>% mutate(model_name = "NNET")

best_nnet_cm <- confusionMatrix(best_nnet_cv_pred$pred, best_nnet_cv_pred$obs)

### ROC curves for the NNET model
best_nnet_cv_pred %>%
     ggplot(mapping = aes(m = event,
                          d = ifelse(obs == 'event',1,0))) +
    geom_roc(mapping = aes(color = Resample), cutoffs.at = 0.5) +
    geom_roc(cutoffs.at = 0.5) +
  geom_hline(yintercept = best_nnet_cm$byClass["Sensitivity"]) +
  geom_vline(xintercept = 1 - best_nnet_cm$byClass["Specificity"],
             color ='red',
             linetype ="dashed")+
    coord_equal() +
    style_roc()
```



How does the Resample averaged ROC curve relate to the individual Fold ROC curves for the neural network model?
How does the location of the 50% threshold compare to that for the other models? ?
For the nnet model, the resample avg ROC curve appears to be in the middle of the individual curves. The location of the 50% threshold appears higher than that for other models.

5e)

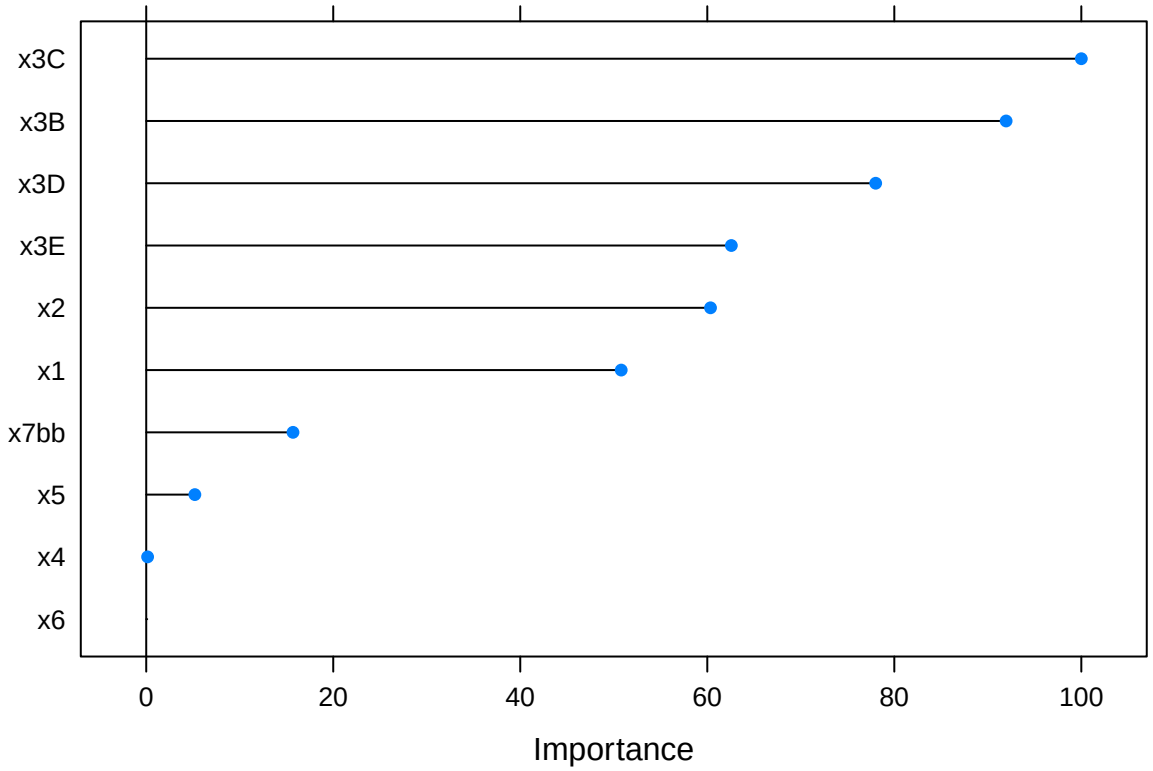
We discussed how individual hidden unit parameters cannot be used to intepret input importance with neural networks. We discussed that all of the hidden units and output layer parameters would need to be considered. Although this sounds challenging, methods exist for analyzing such variable importances with neural networks. In `caret`, all you have to do is call the `varImp()` function on the `nnet` associated `caret` model object.

PROBLEM

Use the `plot()` function to plot the variable importances as viewed by the neural network model. Which inputs are considered to be 20% or less of the importance of the top predictor (feature)?

SOLUTION

```
### your code here
plot(varImp(fit_nnet_b))
```



Which inputs are considered to be 20% or less of the importance of the top predictor (feature)?** ? Inputs x7bb, x5, and x6 appear to be of 20% or less importance than the top.

Save memory

NOTE: To save RAM the code chunk below deletes objects that we will no longer use.

```
rm(fit_glmnet_2, fit_glmnet_3, fit_nnet_a, fit_svm_rbf)

rm(best_svm_rbf_F1_cm, best_svm_rbf_F4_cm, best_svm_rbf_F1_pred, best_svm_rbf_F4_pred)

rm(best_nnet_cm)

rm(best_glm_cv_pred, best_glmnet_3_cv_pred, best_svm_lin_cv_pred, best_svm_rbf_cv_pred)
```

Problem 6

You will now build a random forest model using `caret`. The `method` argument within `train()` will be set to `"rf"`. If this is your first time building a random forest model in `R`, `caret` will ask if it can download and install the necessary packages. Please look at the `R` console for the prompt. There are other random forest implementations in `R`, but the one associated with the `"rf"` method is the original.

6a)

As discussed in lecture, the main tuning parameter for a random forest is the number of randomly selected predictors (features) to try at a split, `mtry`. You will use the default number of trees.

PROBLEM

In the code chunk below create a grid of possible `mtry` values to consider. Set the `mtry` variable within the `expand.grid()` call to be a vector with elements equal to 2, 3, 5, 7. Train the model using arguments consistent with the previous training calls, but set the `rf_grid` equal to the `tuneGrid` argument, and set `method` equal to `"rf"`. Also, include the `importance = TRUE` within the `train()` call.

After training completes, WHICH MIGHT TAKE A FEW MINUTES, print the result to screen. Which value of `mtry` provides the best performing model? Which `mtry` corresponds to a bagged tree model?

SOLUTION

```
rf_grid <- expand.grid(mtry=c(2,3,5,7))
```

```
set.seed(98131)
fit_rf <- train(output ~ . ,
                method ="rf",
                data = train_df,
                metric = metric_used,
                trControl = ctrl_k05,
                trace = FALSE,
                tuneGrid = rf_grid,
                importance = TRUE)
```

```
fit_rf
```

```
## Random Forest
##
## 1500 samples
##    7 predictor
##    2 classes: 'event', 'non_event'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 3 times)
## Summary of sample sizes: 1200, 1200, 1200, 1200, 1200, 1200, ...
## Resampling results across tuning parameters:
##
##   mtry  ROC        Sens       Spec
##   2     0.6356779  0.1680912  0.9486339
##   3     0.6638005  0.3168091  0.8899818
##   5     0.6776672  0.3612536  0.8695811
##   7     0.6858593  0.3908832  0.8663024
##
## ROC was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 7.
```

Which value of `mtry` provides the best performing model? Which `mtry` corresponds to a bagged tree model?** ?
mtry = 7 appars to provide the best performance. The final mtry value of 7 corresponds to a bagged tree. ### 6b)

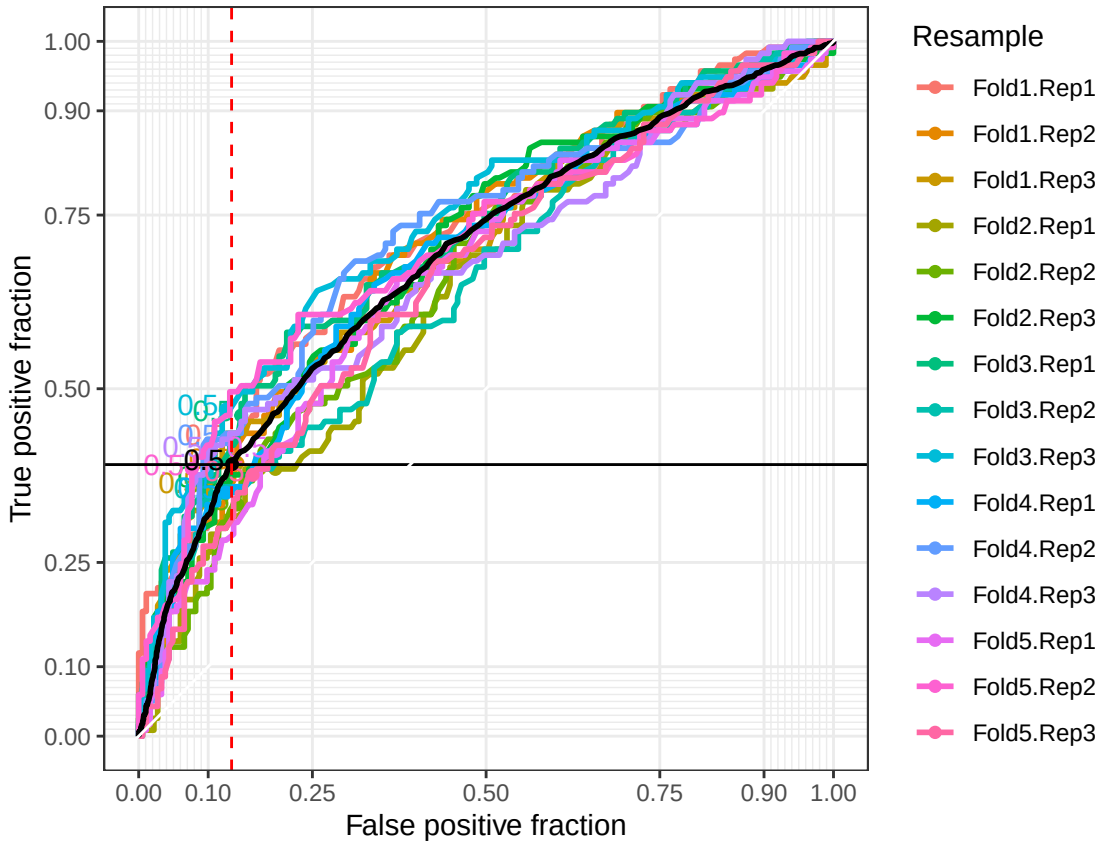
Let's now look at the ROC curve for the best performing random forest model.

PROBLEM

Extract the hold-out set predictions for the best tuned random forest model and store the result to `best_rf_cv_pred`. Do not filter by Resample. Follow the example with `best_glm_cv_pred` by selecting a specific subset of the variables. Create a new column with `mutate()` with `model_name = "RF"`. Use the variables within `best_rf_cv_pred` and the `confusionMatrix()` function to calculate the confusion matrix, and store the result to `best_rf_cm`. Create the ROC curve similar to that for the neural network model where you compared the resample averaged ROC curve to the ROC curves associated with each resample fold. Include the Sensitivity and 1 - Specificity as reference lines.

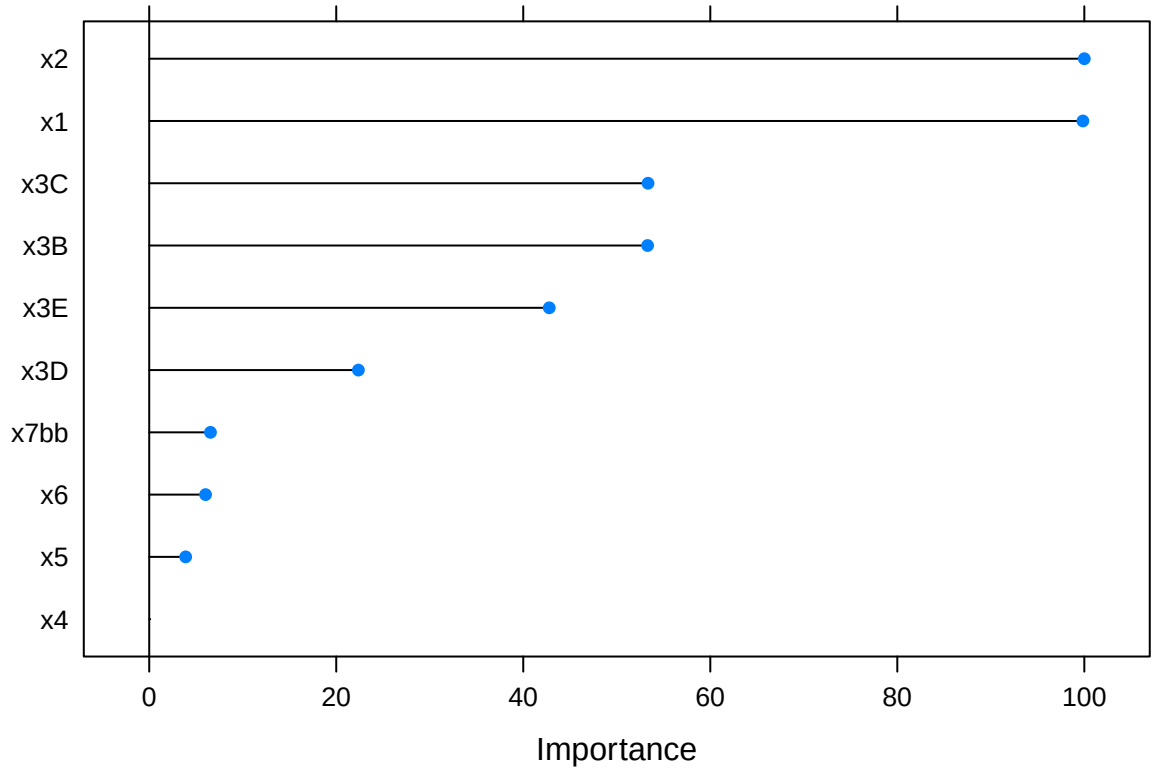
SOLUTION

```
best_rf_cv_pred <- fit_rf$pred %>% tbl_df() %>%
  filter(mtry == fit_rf$bestTune$mtry)%>%
  mutate(model_name = "RF")

best_rf_cm <- confusionMatrix(best_rf_cv_pred$pred, best_rf_cv_pred$obs)

### create the ROC curve for the random forest model
best_rf_cv_pred %>%
     ggplot(mapping = aes(m = event,
                          d = ifelse(obs == 'event',1,0))) +
    geom_roc(mapping = aes(color = Resample), cutoffs.at = 0.5) +
    geom_roc(cutoffs.at = 0.5) +
    geom_hline(yintercept = best_rf_cm$byClass["Sensitivity"]) +
    geom_vline(xintercept = 1 - best_rf_cm$byClass["Specificity"],
             color ='red',
             linetype ="dashed")+
    coord_equal() +
    style_roc()
```



6c

```
plot(varImp(fit_rf))
```



PROBLEM

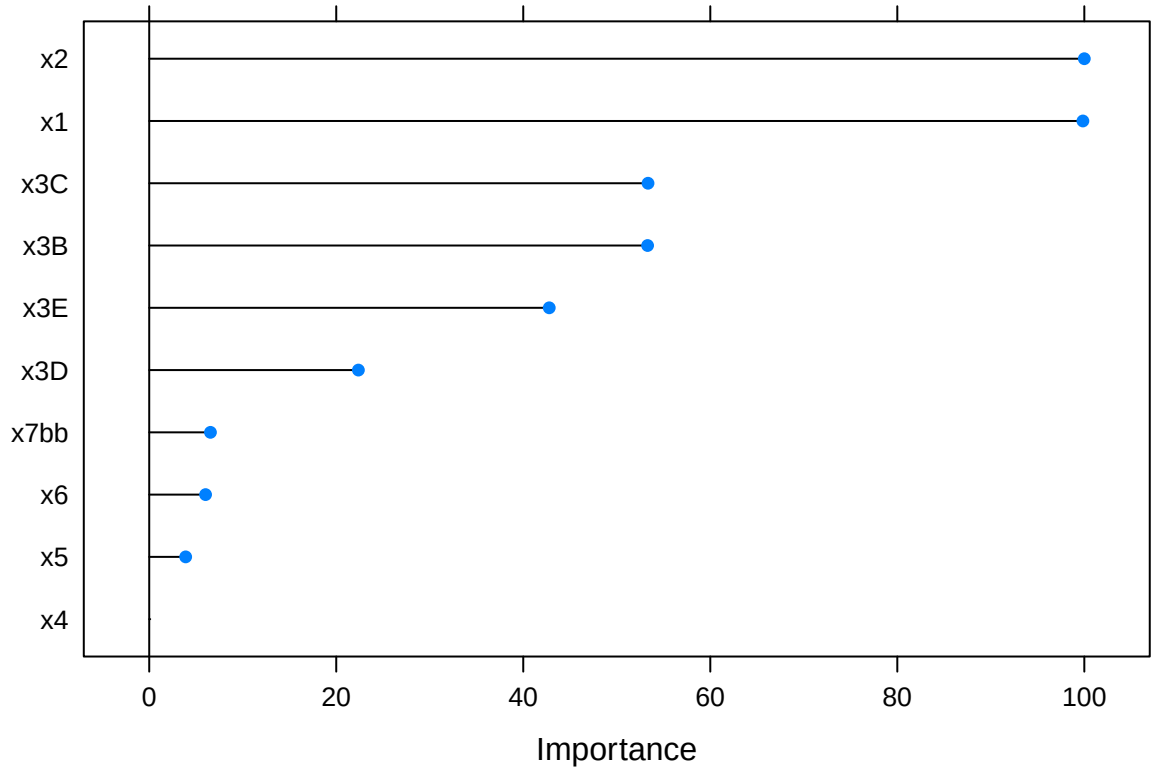
Plot the variable importances as viewed by the random forest model. Do any inputs have a relative importance of less than 20% of the top input? If so, which ones?

How do the variable importances compare between the random forest and neural network models?

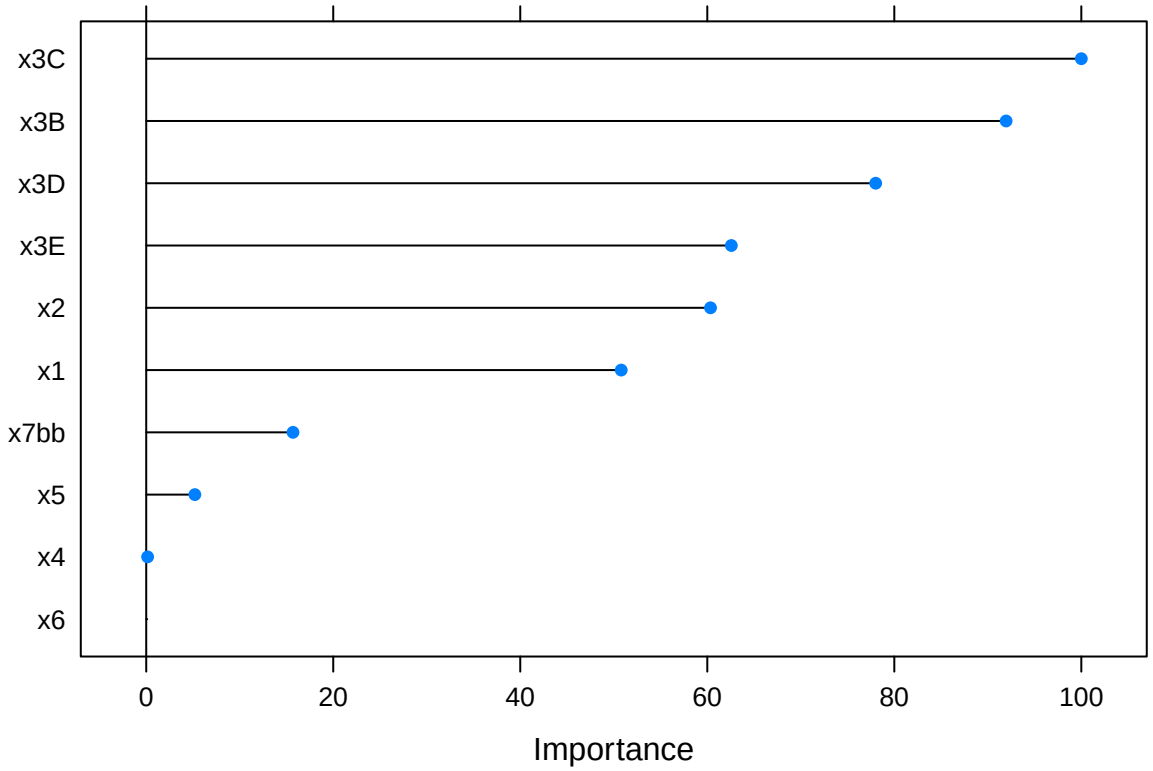
Yes. x7bb, x6, and x5 have relative importance less than 20%. The variable importances are similar to that of the neural network in terms of the unimportant variables but different in terms of the important ones.

SOLUTION

```
### your code here
plot(varImp(fit_rf))
```




```
plot(varImp(fit_nnet_b))
```



Do any inputs have a relative importance of less than 20% of the top input? If so, which ones?**

How do the variable importances compare between the random forest and neural network models?
?
Yes. x7bb, x6, and x5 have relative importance less than 20%. The variable importances are similar to that of the neural network in terms of the unimportant variables but different in terms of the important ones.

Problem 7

You will now build a boosted tree model using the `method` equal to `"gbm"`. If this is the first time using a `"gbm"` in `R` follow the `R` console prompts to install and download the package.

7a)

As discussed in lecture boosted trees have more tuning parameters compared to the random forest. You will start out considering a default value for the learning rate (the `shrinkage`) of 0.1, while varying the number of trees, `n.trees`, and the interaction depth, `interaction.depth`. You try out two different `n.minobsinnode` values as well.

PROBLEM

Create a grid of tuning parameter values, `gbm_grid_a`, with the `expand.grid()` function. Set the variable `n.trees` to a vector from 50 to 1050 by 100. Set `interaction.depth` to a vector with values of 1, 9, and 18. Set `n.minobsinnode` to be 5 and 10. Use the default `shrinkage`.

Train the boosted model using arguments consistent with the previous models, except set the `method` to `"gbm"` and `tuneGrid` to `gbm_grid_a`. Set `verbose` to `FALSE` to prevent each iteration from being printed to the screen.

After training is complete, WHICH WILL TAKE A FEW MINUTES, use the `plot()` function to plot the results and print the best tuning parameter values to the screen.

SOLUTION

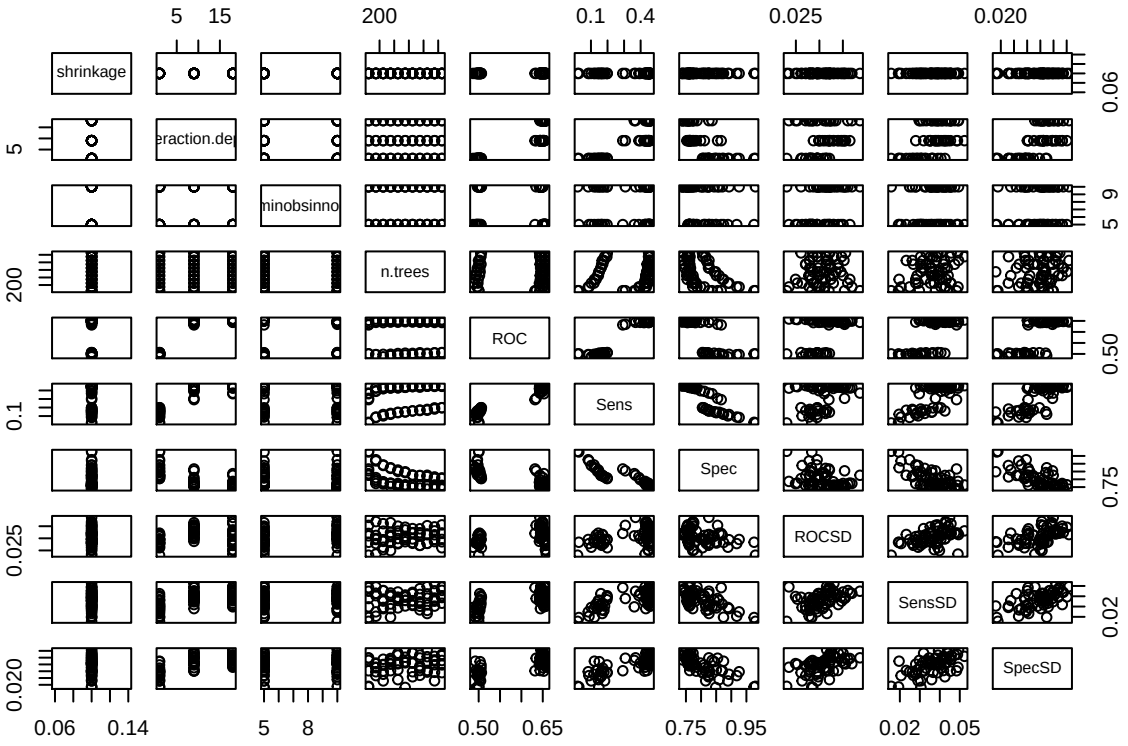
```
gbm_grid_a <- expand.grid(n.trees=seq(50, 1050, by=100),
                          interaction.depth = c(1,9,18),
                          shrinkage=0.1,
                          n.minobsinnode=c(5,10))
```

```
set.seed(98131)
fit_gbm_a <- train(output ~ . ,
                   method ="gbm",
                   data = train_df,
                   metric = metric_used,
                   trControl = ctrl_k05,
                   #trace = FALSE,
                   tuneGrid = gbm_grid_a,
                   verbose = FALSE)
                   #importance = TRUE)
```

```
### print out the best tuning parameters and plot
### the results
fit_gbm_a$bestTune
```

```
##    n.trees interaction.depth shrinkage n.minobsinnode
## 46     150                18       0.1              5
```

```
plot(fit_gbm_a$results)
```



7b)

PROBLEM

Which `interaction.depth` yielded the best results? What do you think that represents about the model behavior?

SOLUTION

?
An interaction depth of 18 appears to have yielded the best result. This suggests the model converges relatively quickly.

7c)

Now let's try lowering the learning rate. You will keep the `interaction.depth` fixed to 18 and the `n.minobsinnode` fixed at 5 for this problem.

PROBLEM

Specify a new grid with `expand.grid()`. Set the variable `n.trees` to be a vector from 250 to 2250 by increments of 500. Set the `shrinkage` to 0.005 and 0.01. Set the `interaction.depth` to 18 and the `n.minobsinnode` equal to 5.

Train the boosted tree, using arguments similar to that in Problem 7a). Be sure to set `verbose` to `FALSE`. After training is complete, print results to screen and plot the results.

THE MODEL WILL TAKE SEVERAL MINUTES TO TRAIN.

SOLUTION

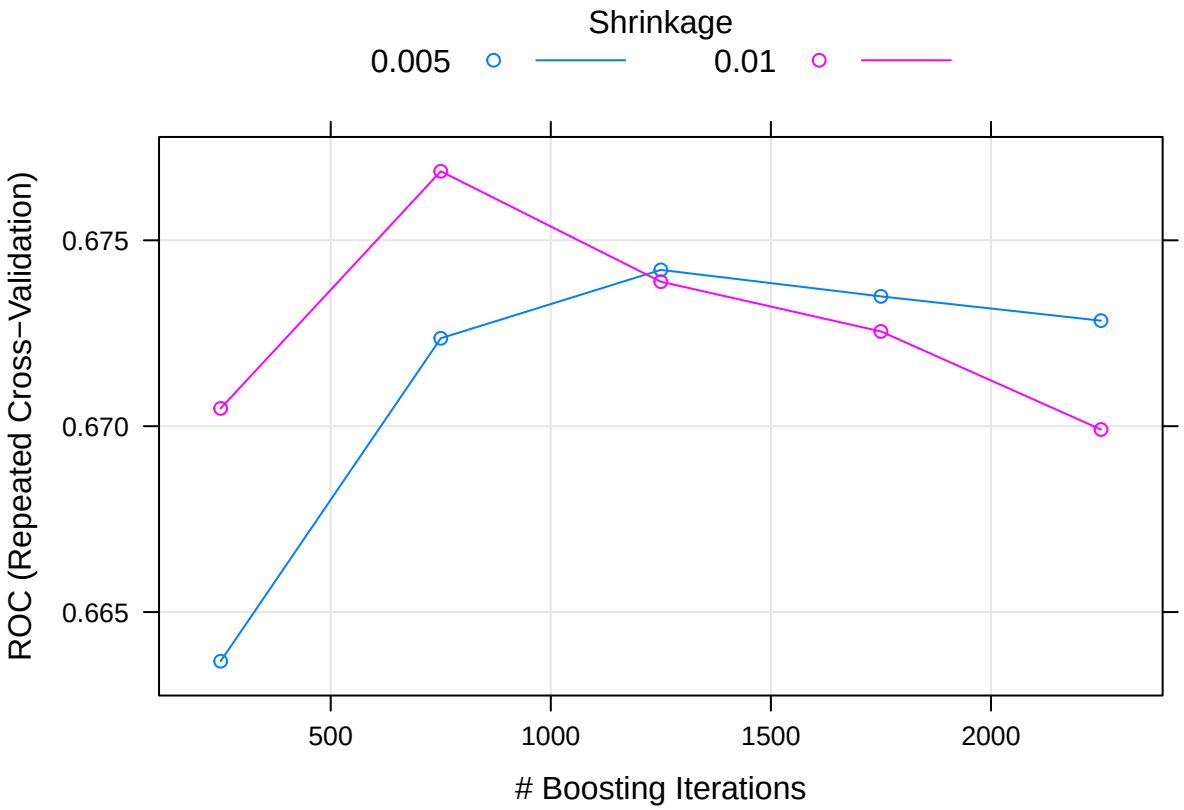
```
gbm_grid_b <- expand.grid(n.trees=seq(250, 2250, by=500),
                          interaction.depth = 18,
                          shrinkage=c(0.005,0.01),
                          n.minobsinnode=5)
```

```
set.seed(98131)
fit_gbm_b <- train(output ~ . ,
                method ="gbm",
                data = train_df,
                metric = metric_used,
                trControl = ctrl_k05,
                tuneGrid = gbm_grid_b,
                verbose = FALSE)
```

```
fit_gbm_b
```

```
## Stochastic Gradient Boosting
##
## 1500 samples
##    7 predictor
##    2 classes: 'event', 'non_event'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 3 times)
## Summary of sample sizes: 1200, 1200, 1200, 1200, 1200, 1200, ...
## Resampling results across tuning parameters:
##
##   shrinkage  n.trees  ROC        Sens       Spec
##   0.005       250     0.6636775  0.1487179  0.9650273
##   0.005       750     0.6723647  0.3213675  0.8950820
##   0.005      1250     0.6742048  0.3658120  0.8586521
##   0.005      1750     0.6734918  0.3903134  0.8397086
##   0.005      2250     0.6728379  0.4091168  0.8240437
##   0.010       250     0.6704778  0.2638177  0.9198543
##   0.010       750     0.6768577  0.3874644  0.8444444
##   0.010      1250     0.6738873  0.4113960  0.8142077
##   0.010      1750     0.6725484  0.4273504  0.7996357
##   0.010      2250     0.6699111  0.4358974  0.7938069
##
## Tuning parameter 'interaction.depth' was held constant at a value of
##  18
## Tuning parameter 'n.minobsinnode' was held constant at a value of 5
## ROC was used to select the optimal model using the largest value.
## The final values used for the model were n.trees = 750,
##  interaction.depth = 18, shrinkage = 0.01 and n.minobsinnode = 5.
```

```
### plot the results
plot(fit_gbm_b)
```



7d)

Let's now assemble the two different boosted tree model results into the format to create the ROC curve. You must filter the hold-out set predictions to just the best tuned models, but you do not need to filter by Resample.

PROBLEM

Create the `best_gbm_cv_pred_a` and `best_gbm_cv_pred_b` objects by filtering the best tuned parameter values. Do not filter by the Resample. Follow the `best_glm_cv_pred` format and select a specific subset of the variables, and `mutate()` a new column. Set the `model_name = "GBM-A"` for `best_gbm_cv_pred_a` and `model_name = "GBM-B"` for `best_gbm_cv_pred_b`.

The second code chunk is completed for you. It merges the two GBM results together into a single object. You must use this new object, `best_gbm_cv_preds`, to compare the two different boosted tree models through ROC curves.

Create the ROC curve by specifying the `m` and `d` aesthetics in the parent `ggplot()` call correctly. Use two `geom_roc()` calls, where the first one sets the `color` aesthetic equal to `Resample` and the second `geom_roc()` call does not specify other additional aesthetics. In each `geom_roc()` call set `cutoff.at` equal to 0.5. This time you must use the `facet_wrap()` function to create a facet based on `model_name`.

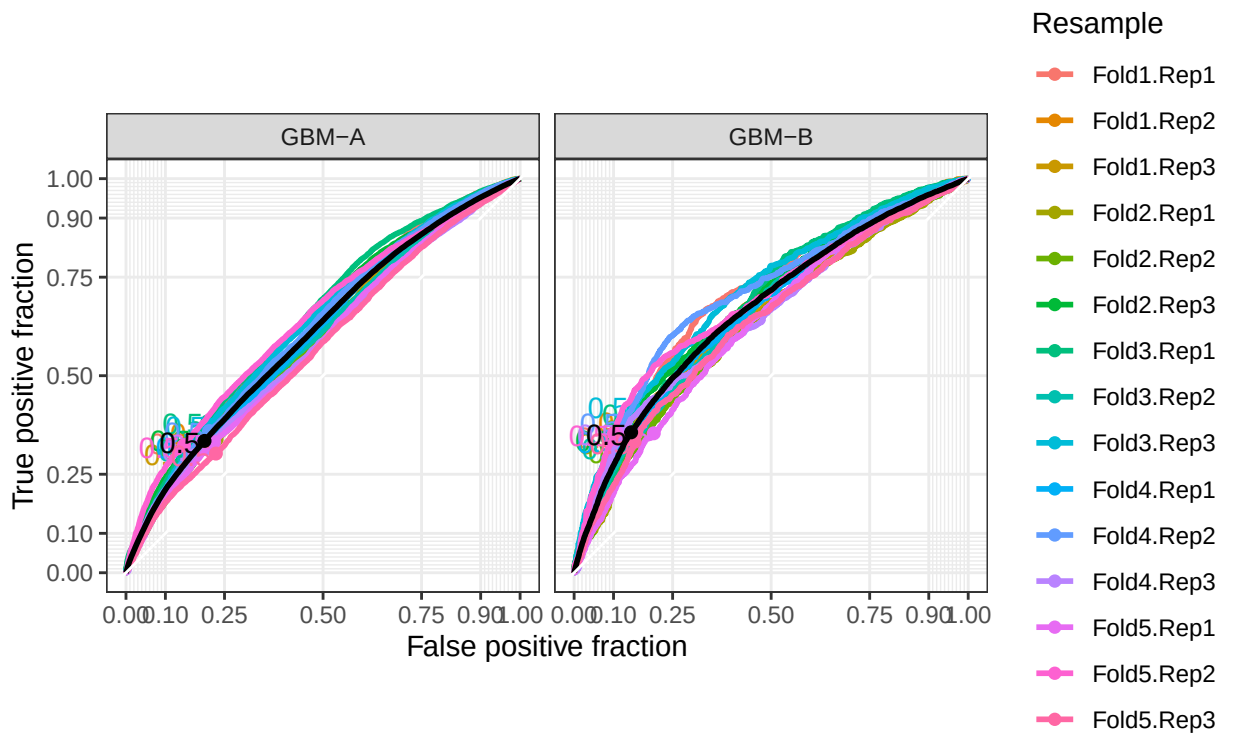
SOLUTION

```
best_gbm_cv_pred_a <- fit_gbm_a$pred %>% tbl_df() %>%
  filter(n.trees, interaction.depth, shrinkage, n.minobsinnode)%>%
  mutate(model_name = "GBM-A")

best_gbm_cv_pred_b <- fit_gbm_b$pred %>% tbl_df() %>%
  filter(n.trees, interaction.depth, shrinkage, n.minobsinnode)%>%
  mutate(model_name = "GBM-B")
```

```
### requires the code chunk above, solution_07d, to
### be completed properly
best_gbm_cv_preds <- best_gbm_cv_pred_a %>%
  bind_rows(best_gbm_cv_pred_b)
```

Compare the two boosted trees through their ROC curves.

```
### compare the boosted tree models through their
### ROC curves
best_gbm_cv_preds %>%
     ggplot(mapping = aes(m = event,
                          d = ifelse(obs == 'event',1,0))) +
    geom_roc(mapping = aes(color = Resample), cutoffs.at = 0.5) +
    geom_roc(cutoffs.at = 0.5) +
    facet_wrap(~model_name) +
    coord_equal() +
    style_roc()
```



7e

It's time to compare the different models based on their ROC curves. The code chunk below merges the neural net, random forest, and the boosted tree model hold-out predictions with the previously compiled model results `best_mod_cv_pred_a`. The new object is named `best_mod_cv_pred_b`, and as a check the the `count()` function is applied to `model_name` to show all of the models of interest are assembled.

```
best_mod_cv_pred_b <- best_mod_cv_pred_a %>%
  bind_rows(best_nnet_cv_pred) %>%
  bind_rows(best_rf_cv_pred) %>%
  bind_rows(best_gbm_cv_preds)

best_mod_cv_pred_b %>%
  count(model_name)
```

```
## # A tibble: 8 x 2
##   model_name      n
##   <chr>       <int>
## 1 GBM-A      297000
## 2 GBM-B       45000
## 3 GLM          4500
## 4 GLMNET-3     4500
## 5 NNET         4500
## 6 RF           4500
## 7 SVM-LIN      4500
## 8 SVM-RBF      4500
```

PROBLEM

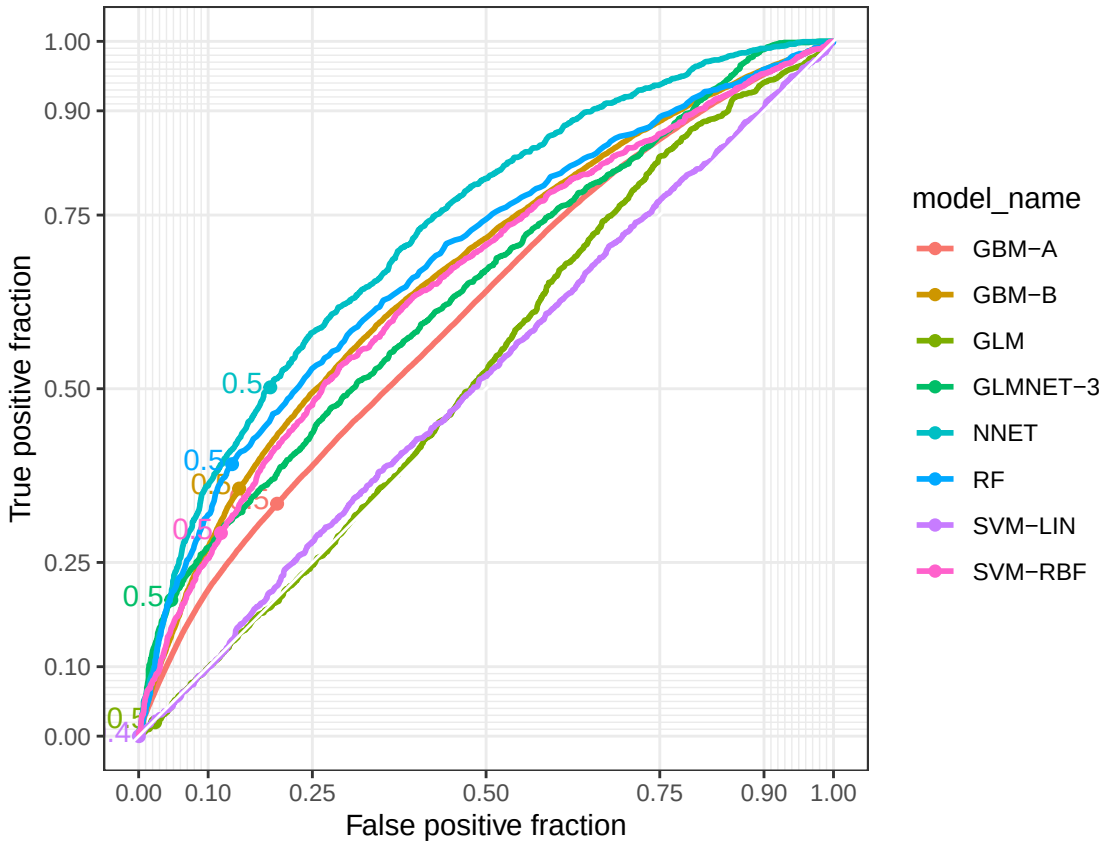
Create the ROC curves for each model by correctly specifying the `m` and `d` aesthetics within the parent `ggplot()` call. Set the `color` aesthetic within `geom_roc()` equal to `model_name`, and set the `cutoff.at` argument equal to 0.5.

Which model appears the best, based on the Resample averaged ROC curves?

SOLUTION

Based on the resampled avg ROC curves, the GBM-B model appears to be best.

```
### compare the models by comparing their resample averaged
### ROC curves
best_mod_cv_pred_b %>%
     ggplot(mapping = aes(m = event,
                          d = ifelse(obs == 'event',1,0))) +
    geom_roc(mapping = aes(color = model_name), cutoffs.at = 0.5) +
    coord_equal() +
    style_roc()
```



7f)

PROBLEM

Based on the ROC curves in Problem 7d), which is the best performing model?

Which model has the highest Sensitivity at the 50% threshold value?

We have discussed several times in lecture the shape of the "ideal" ROC curve. Based on all of the ROC curves you have generated, can you describe why the "ideal" ROC curve looks the way it does?

SOLUTION

?

Based on the ROC curves in 7d, the NNET model appears to be the best performing. The NNET model appears to have the highest Sensitivity at 50% threshold.

7g)

The `resamples()` function in `caret` allows compiling all of the resample fold results and extracting the best tuned parameter values. It essentially provides a short cut to comparing models, without having to resort to working with the hold-out predictions directly. We needed those predictions in the assignment in order to generate the ROC curves. However, the `resamples()` function allows comparing models even when the predictions were not saved.

To use the `resamples()` function, each `caret` model object must be assigned to a variable in a list. The code chunk below is started for you for the logistic regression model fit. You will complete the list by assigning the models as specified by the names in the list.

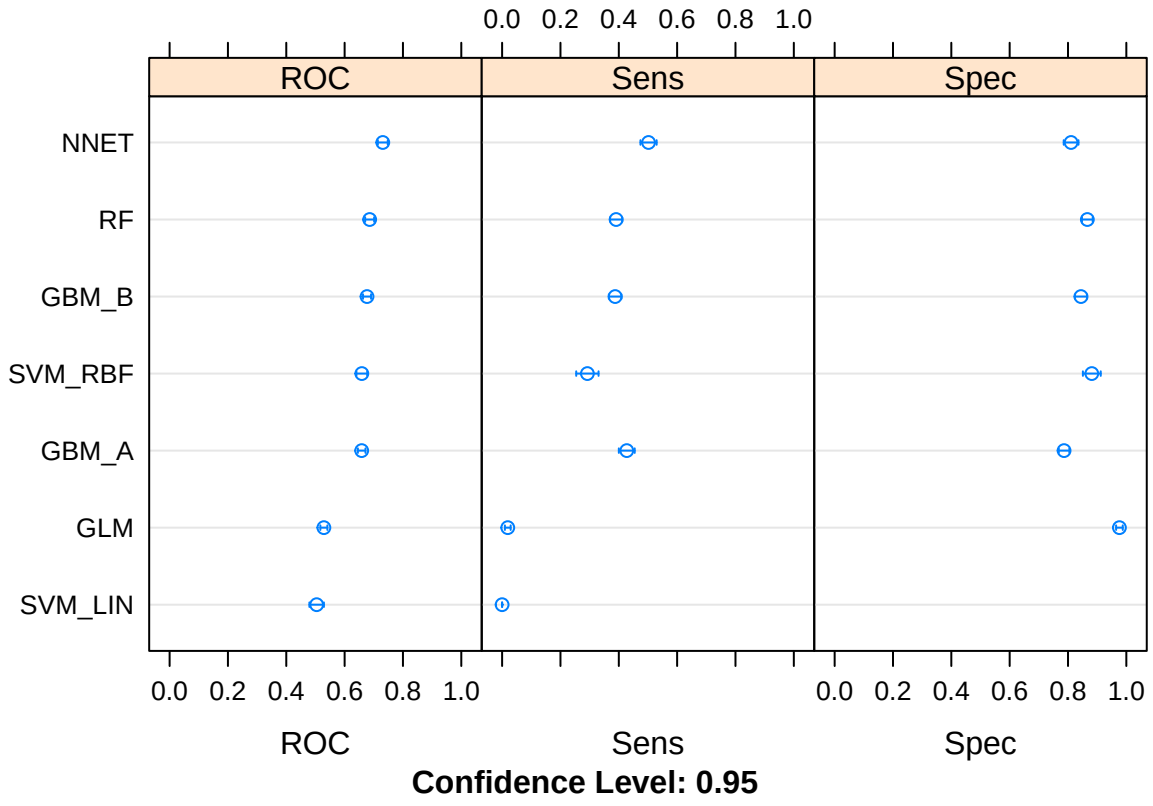
PROBLEM

Complete the list in the first code chunk below. Once completed, use the `dotplot()` function to summarize the performance metrics across the resample folds for each model. Are the results consistent with your conclusions based on the ROC curves directly?

SOLUTION

```
mod_results <- resamples(list(GLM = fit_glm,
                              SVM_LIN = fit_svm_lin,
                              SVM_RBF = fit_svm_rbf_b,
                              NNET = fit_nnet_b,
                              RF = fit_rf,
                              GBM_A = fit_gbm_a,
                              GBM_B = fit_gbm_b))
```

```
### call the dotplot function to compare the models
dotplot(mod_results)
```



Are the results consistent with your conclusions based on the ROC curves directly?**
?
Yes, the results are consistent with conclusions.