Overview

This homework begins by reviewing linear separability and why it causes issues for logistic regression models. The bulk of the assignment is dedicated to comparing binary classification models. You will train unregularized and regularized logistic regression models, neural networks, random forests, and gradient boosted trees with XGBoost.

You will use the `tidyverse`, `glmnet`, `randomForest`, `xgboost`, and `caret` packages in this assignment. The `caret` package will prompt you to install `xgboost` if you do not have it installed already.

IMPORTANT: The RMarkdown assumes you have downloaded the data sets (CSV files) to the same directory you saved the template Rmarkdown file. If you do not have the CSV files in the correct location, the data will not be loaded correctly.

IMPORTANT!!!

Certain code chunks are created for you. Each code chunk has `eval=FALSE` set in the chunk options. You **MUST** change it to be `eval=TRUE` in order for the code chunks to be evaluated when rendering the document.

You are free to add more code chunks if you would like.

Load packages

This assignment will use packages from the `tidyverse` suite.

```
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.2     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.0
## v ggplot2   3.4.3     v tibble    3.2.1
## v lubridate 1.9.2     v tidyr     1.3.0
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

This assignment also uses the `broom` package. The `broom` package is part of `tidyverse` and so you have it installed already. The `broom` package will be used via the `::` operator later in the assignment and so you do not need to load it directly.

The `caret` package will be loaded later in the assignment. The `glmnet`, `randomForest`, and `xgboost` packages will be loaded via `caret`.

Problem 01

The code chunk below loads the data you will work with in this assignment. The data consists of 3 inputs, `x1`, `x2` and `x3`, and one binary outcome, `y`. The `x1` and `x2` inputs are continuous while the `x3` input is categorical.

```
df <- readr::read_csv('hw11_binary_train.csv', col_names = TRUE)
```

```
## Rows: 300 Columns: 4
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): x3
## dbl (3): x1, x2, y
##
## i Use `spec()` to retrieve the full column specification for this data.
```

```
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

The glimpse shows the data types of the 4 variables and that there are 300 rows in the data set.

```
df %>% glimpse()
```

```
## Rows: 300
## Columns: 4
## $ x1 <dbl> -0.1737886, -0.3883115, -0.7308443, -1.3366670, -0.4435314, 0.29721~
## $ x2 <dbl> -0.750485806, -0.361116231, 0.141729009, 1.883642959, 2.134528551, ~
## $ x3 <chr> "A", "A", "A", "A", "A", "A", "A", "A", "A", "A", "A", "A", "A", "A~
## $ y  <dbl> 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0~
```

However, you will begin this assignment by working with a very small subset of the data. The small subset is created for you below. This small data set is created strictly to reinforce an important concept associated with logistic regression. You should **not** perform an operation like this in the final project. Again, this procedure is performed here for teaching purposes.

```
df_small <- df %>% slice(1:21)

df_small
```

```
## # A tibble: 21 x 4
##        x1     x2 x3        y
##     <dbl>  <dbl> <chr> <dbl>
##  1 -0.174 -0.750 A         1
##  2 -0.388 -0.361 A         1
##  3 -0.731  0.142 A         1
##  4 -1.34   1.88  A         0
##  5 -0.444  2.13  A         0
##  6  0.297 -1.24  A         0
##  7 -0.319 -0.642 A         1
##  8 -0.416 -1.85  A         0
##  9  0.336  1.38  A         0
## 10 -0.106 -0.184 A         1
## # i 11 more rows
```

The small data, `df_small`, consists of a single value for the categorical variable `x3`, as shown below.

```
df_small %>% count(x3)
```

```
## # A tibble: 1 x 2
##   x3        n
##   <chr> <int>
## 1 A        21
```

The larger data set, which you will work with later, has 3 balanced unique values (categories or *levels*):

```
df %>% count(x3)
```

```
## # A tibble: 3 x 2
##   x3        n
##   <chr> <int>
## 1 A       100
## 2 B       100
## 3 C       100
```

The binary outcome, `y`, is encoded as `y = 1` for the EVENT and `y = 0` for the NON-EVENT. The counts and proportions associated with the 2 unique value values of `y` are shown below for the small data set:

```
df_small %>% count(y) %>%
  mutate(prop = n / sum(n))
```

```
## # A tibble: 2 x 3
##       y     n  prop
##   <dbl> <int> <dbl>
## 1     0    11 0.524
## 2     1    10 0.476
```

The counts and proportions associated with `y` on the larger data set are provided below.

```
df %>% count(y) %>%
  mutate(prop = n / sum(n))
```

```
## # A tibble: 2 x 3
##       y     n  prop
##   <dbl> <int> <dbl>
## 1     0   166 0.553
## 2     1   134 0.447
```

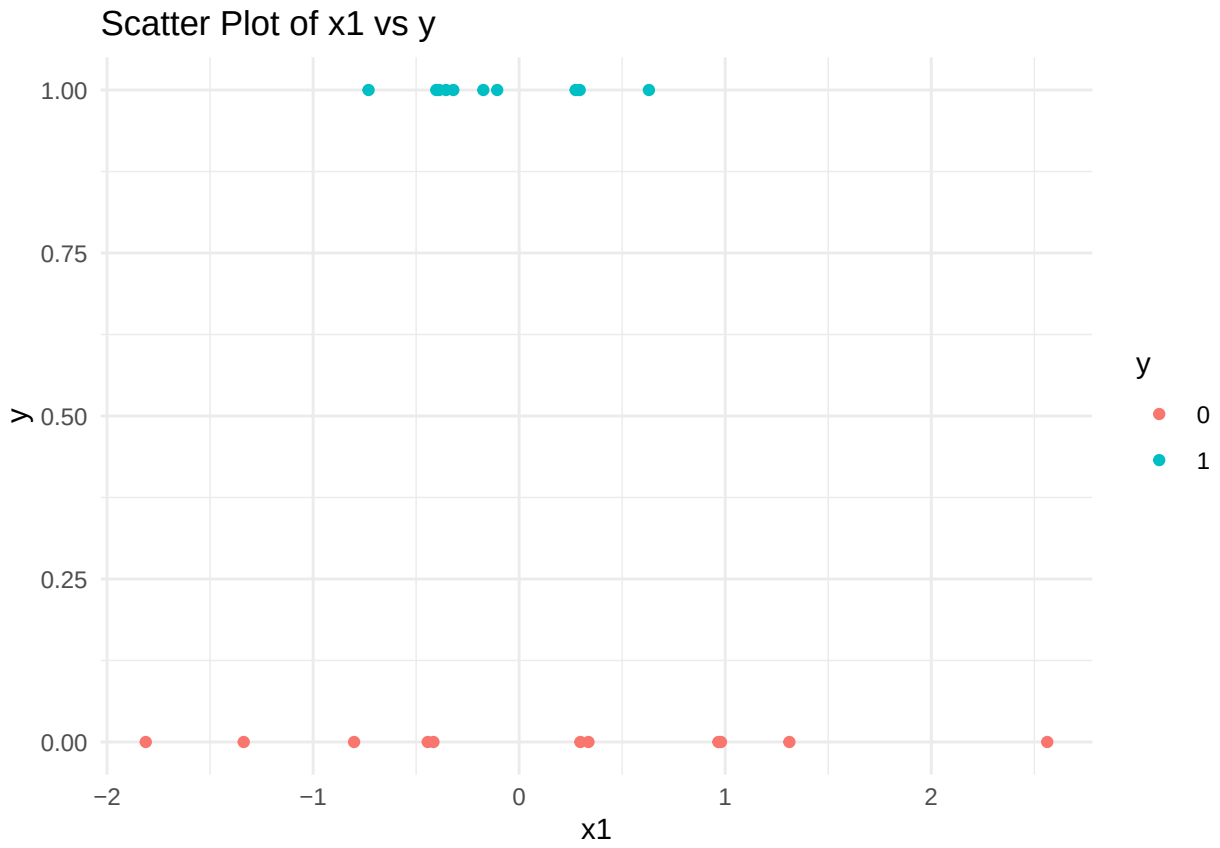
Problem 01 deals with the small data set while the other problems are focused on the original larger data. The small data, `df_small`, corresponds to a single categorical value, `x3 = 'A'`, and thus you will focus on the relationship between the binary outcome, `y`, and the continuous inputs, `x1`, and `x2`, in Problem 01.

1a)

Let's start with a simple exploration of the **small** data, `df_small`.

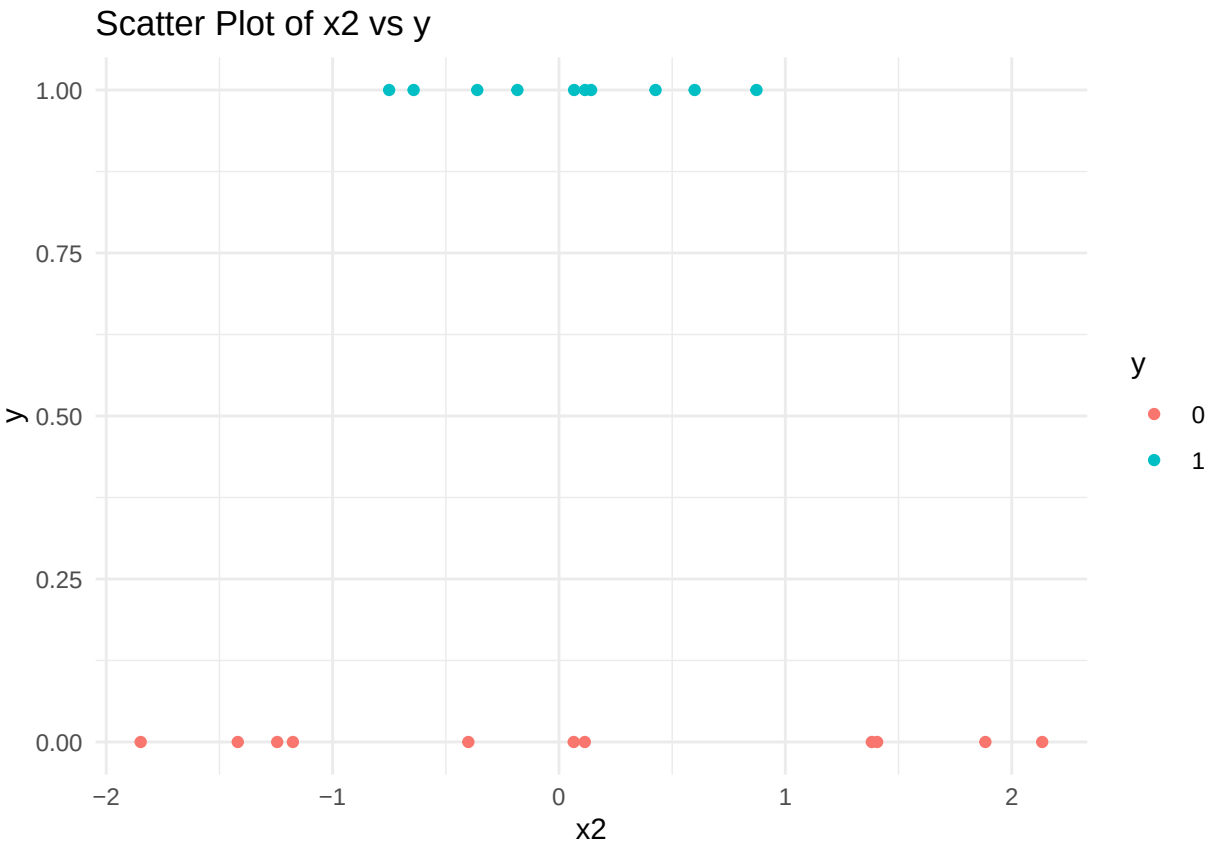
Create scatter plots to show the relationship between the binary outcome `y` and the continuous inputs `x1` and `x2`. You may create two figures or reshape the data to long-format to support creating one figure with two facets.

```
ggplot(df_small, aes(x = x1, y = y, color = as.factor(y))) +
  geom_point() +
  labs(title = "Scatter Plot of x1 vs y",
       x = "x1",
       y = "y",
       color = "y") +
  theme_minimal()
```



SOLUTION

```
ggplot(df_small, aes(x = x2, y = y, color = as.factor(y))) +
  geom_point() +
  labs(title = "Scatter Plot of x2 vs y",
       x = "x2",
       y = "y",
       color = "y") +
  theme_minimal()
```



1b)

Based on the relationships presented in 1a), do you feel you should be concerned about linear separability in this problem?

SOLUTION Since there seems to be an overlap between the data points for y = 0 and y = 1 across x1 and x2, then it's less likely that linear separability is an issue. Overlap indicates that a linear boundary would not perfectly separate the two classes.

1c)

Let's fit a non-Bayesian logistic regression model to see if your answer in 1b) was correct!

Fit a non-Bayesian logistic regression model to predict the binary outcome y based on the two continuous inputs, `x1` and `x2`. Your model must include the linear main effects and the interaction between the two continuous inputs.

Assign the model to the `mod_small_a` variable.

HINT: Be careful with the `family` argument!

```
mod_small_a <- glm(y ~ x1 + x2 + x1:x2, data = df_small, family="binomial")

summary(mod_small_a)
```

SOLUTION

##

```
## Call:
## glm(formula = y ~ x1 + x2 + x1:x2, family = "binomial", data = df_small)
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.09135    0.44314  -0.206    0.837
## x1          -0.34690    0.50343  -0.689    0.491
## x2          -0.02291    0.45185  -0.051    0.960
## x1:x2        0.09867    0.59869   0.165    0.869
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 29.065  on 20  degrees of freedom
## Residual deviance: 28.548  on 17  degrees of freedom
## AIC: 36.548
##
## Number of Fisher Scoring iterations: 4
```

1d)

Let's now fit a more complicated logistic regression model. Let's include quadratic features for both `x1` and `x2`.

Fit a non-Bayesian logistic regression model using linear main effects, interactions between the two continuous inputs, and quadratic features associated with both continuous inputs.

Assign he model to the `mod_small_b` variable.

```
mod_small_b <- glm(y ~ x1 + x2 + I(x1^2) + I(x2^2) + x1:x2, data = df_small, family="binomial")
```

SOLUTION

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

1e)

Something might seem suspicious about the result from the model in 1d)...

Display the `summary()` of the `mod_small_b` model to the screen. Do you notice anything "extreme" about about any of the coefficient MLEs?

```
summary(mod_small_b)
```

SOLUTION

```
##
## Call:
## glm(formula = y ~ x1 + x2 + I(x1^2) + I(x2^2) + x1:x2, family = "binomial",
##     data = df_small)
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)  4.543e+01  9.599e+04   0.000        1
## x1          -1.657e+01  7.303e+04   0.000        1
## x2           5.372e+00  7.124e+04   0.000        1
```

```
## I(x1^2)     -2.792e+01  5.541e+04  -0.001        1
## I(x2^2)     -3.560e+01  6.667e+04  -0.001        1
## x1:x2       -9.398e-01  2.184e+05   0.000        1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2.9065e+01  on 20  degrees of freedom
## Residual deviance: 5.2101e-10  on 15  degrees of freedom
## AIC: 12
##
## Number of Fisher Scoring iterations: 25
```

The summary of the mod_small_b model indicates some issues with the fitted model. All the coefficients have extremely large standard errors, and the z-values are close to zero, which results in p-values of 1. This suggests that the maximum likelihood estimates for the coefficients are not reliable.

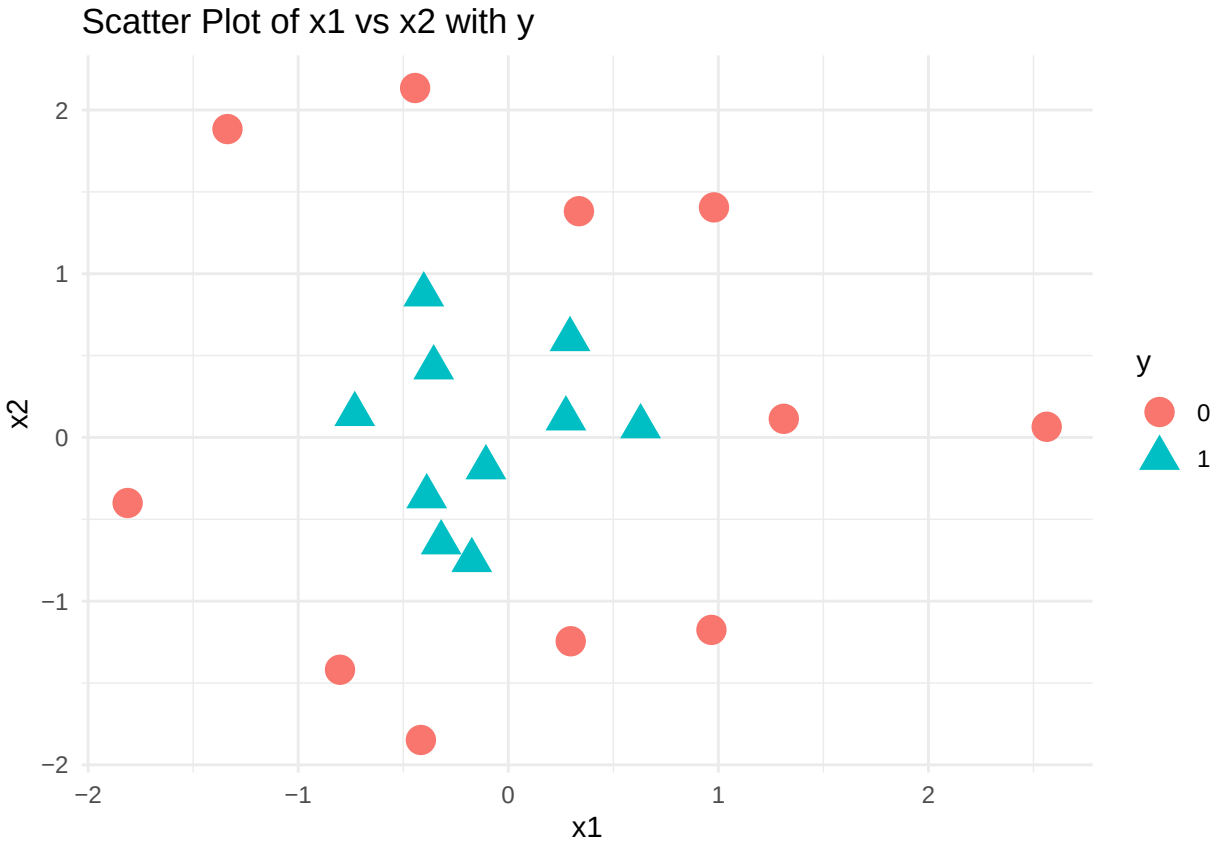
1f)

You visualized the relationship between the binary outcome, `y`, and each continuous input in 1a). However, you did not examine the behavior of `y` with respect to **both** inputs at the same time. The binary outcome can be mapped to the marker `color` and `shape` aesthetics in scatter plots between continuous inputs. Such figures allow visually identifying regions of the *joint* input space where the event occurs more frequently than other regions. Let's create such a plot to understand what happened with `mod_small_b`.

Create a scatter plot showing the relationship between the two continuous inputs. You should map the `x1` variable to the `x` aesthetic and the `x2` variable to the `y` aesthetic. Map the `color` and `shape` aesthetics to `as.factor(y)` within the appropriate geom function. Manually assign the marker `size` to be 5.

HINT: Do NOT use `geom_jitter()`.

```
ggplot(df_small, aes(x = x1, y = x2, color = as.factor(y), shape = as.factor(y))) +
  geom_point(size = 5) +
  labs(title = "Scatter Plot of x1 vs x2 with y",
       x ="x1",
       y ="x2",
       color= "y",
       shape= "y") +
  theme_minimal()
```



SOLUTION

1g)

Why does the figure in 1f) explain the behavior of `mod_small_b`?

SOLUTION The figure in 1f) likely explains the behavior of mod_small_b by showing that there is clear separation in the data, which can cause problems in fitting a logistic regression model, such as extreme coefficient estimates and potential overfitting.

Problem 02

The remainder of the assignment will use the larger `df` data set.

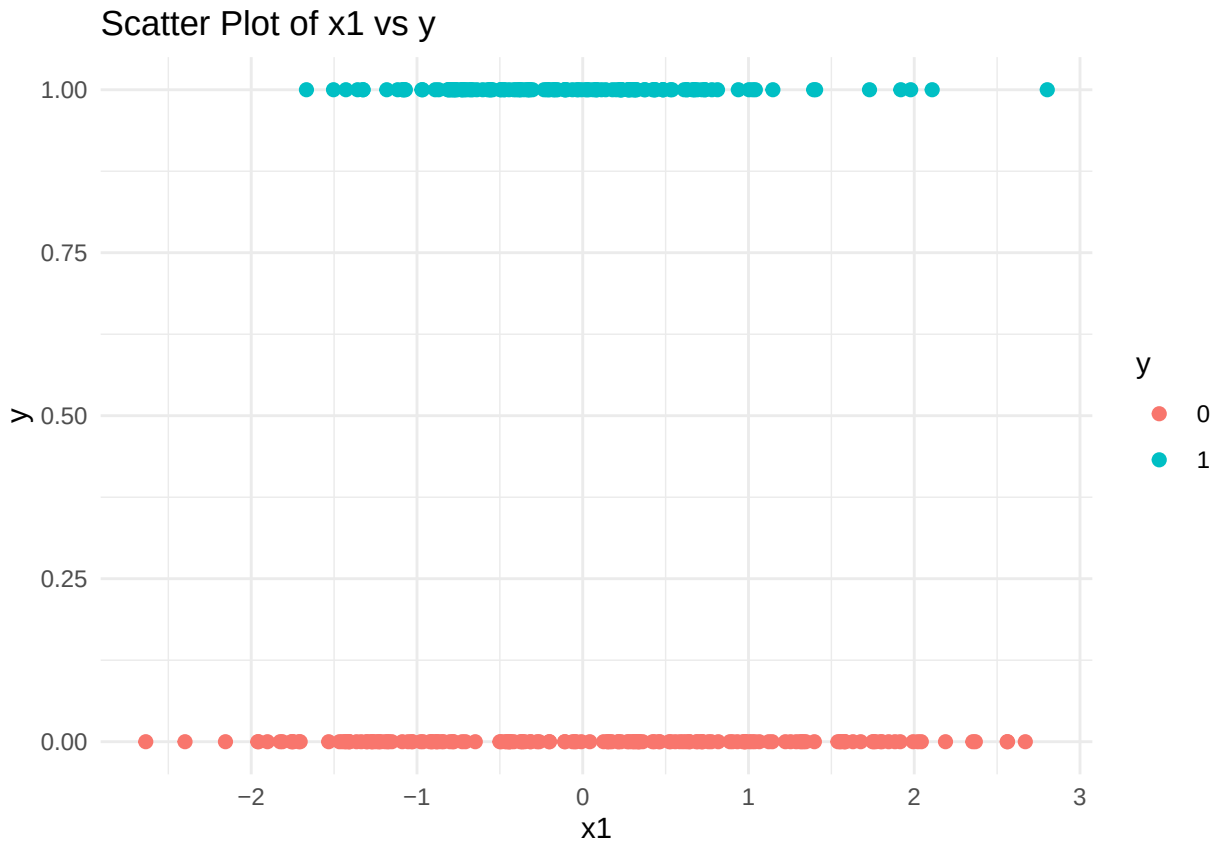
2a)

It would be best to perform a full visual exploration of the data, but you will focus on the binary outcome, `y`, to continuous input relationships in this assignment. (A full exploration includes marginal distributions and conditional distributions of the variables and also examining the relationships between the inputs.)

Create scatter plots to show the relationship between the binary outcome `y` and the continuous inputs `x1` and `x2`. You may create two figures or reshape the data to long-format to support creating one figure with two facets.

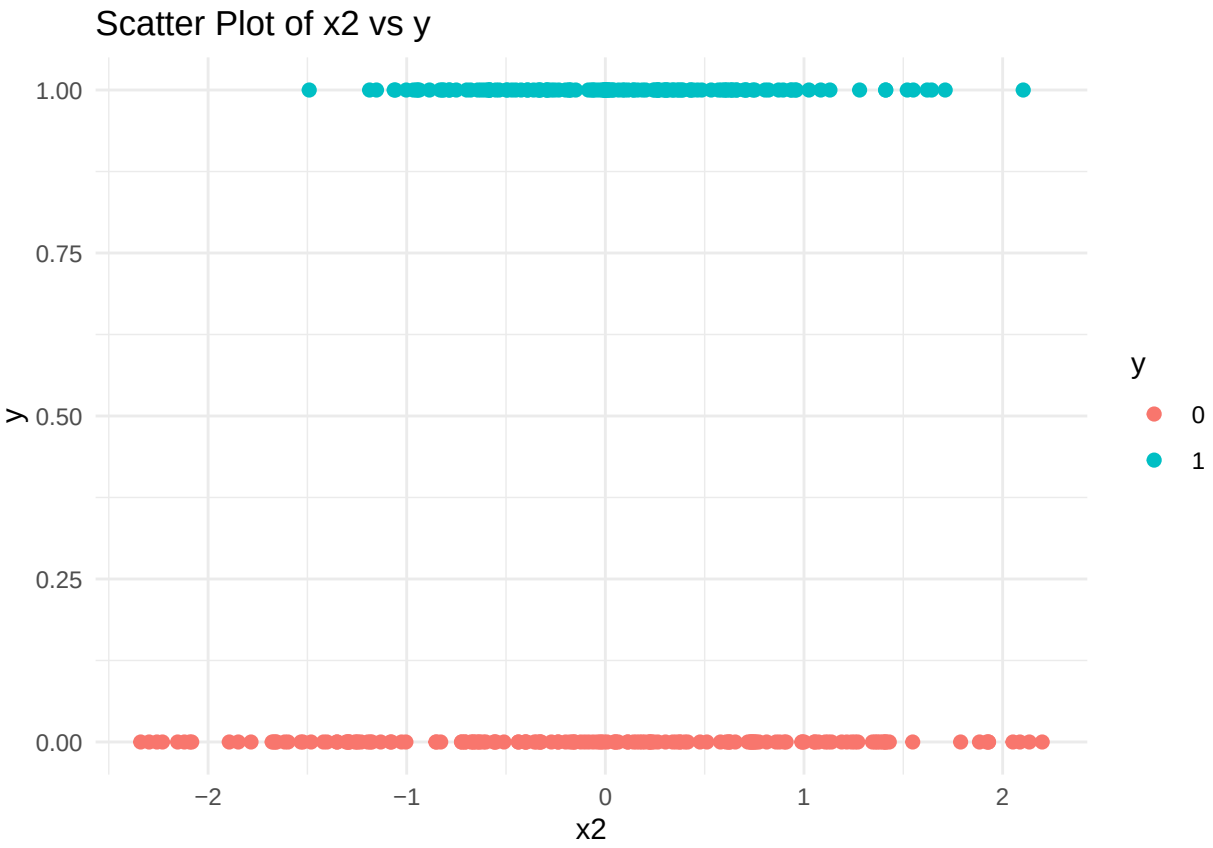
```
ggplot(df, aes(x = x1, y = y, color = as.factor(y))) +
  geom_point(size = 2) +
  labs(title = "Scatter Plot of x1 vs y",
       x = "x1",
       y = "y",
```

```
      color = "y") +
  theme_minimal()
```



SOLUTION

```
ggplot(df, aes(x = x2, y = y, color = as.factor(y))) +
  geom_point(size = 2) +
  labs(title = "Scatter Plot of x2 vs y",
       x = "x2",
       y = "y",
       color = "y") +
  theme_minimal()
```



2b)

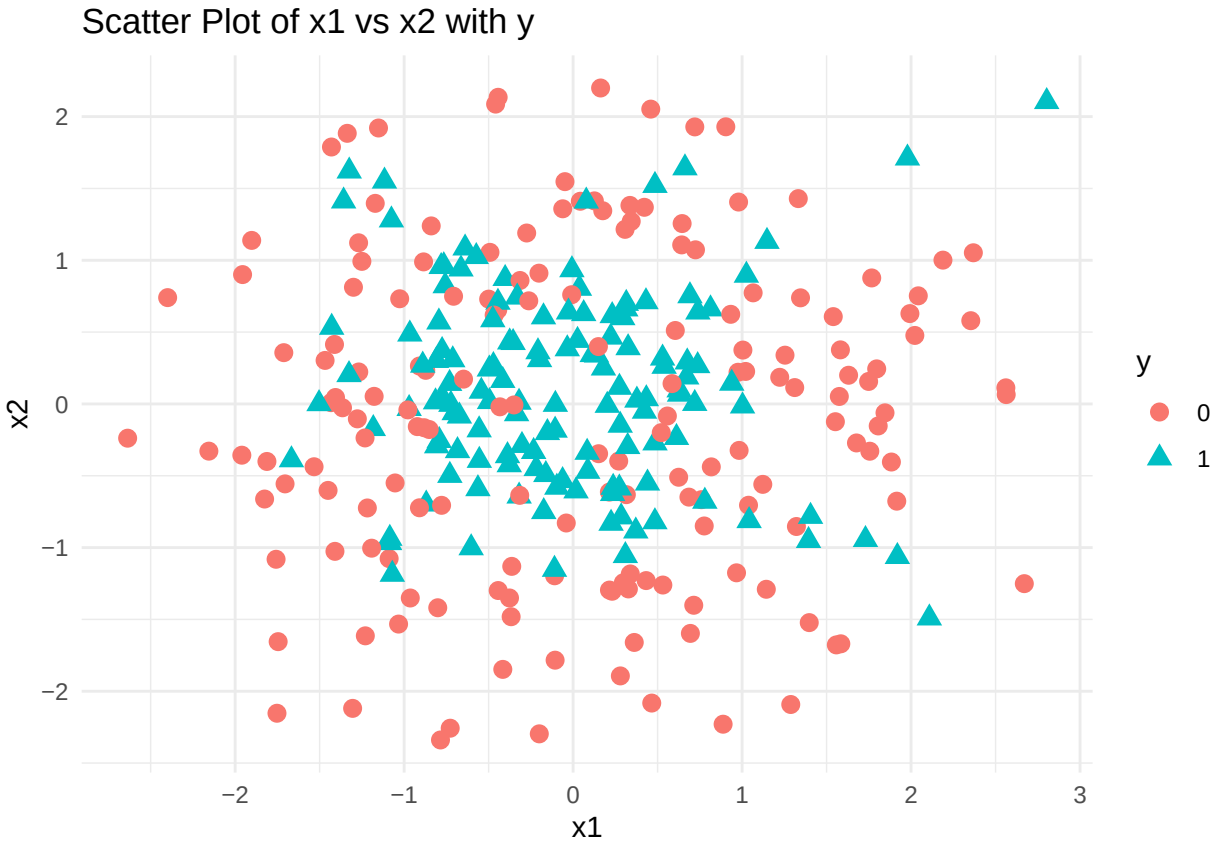
Let's now consider the relationship between the binary outcome and the *joint* behavior of the continuous inputs.

Create a scatter plot showing the relationship between the two continuous inputs. You should map the `x1` variable to the `x` aesthetic and the `x2` variable to the `y` aesthetic. Map the `color` and `shape` aesthetics to `as.factor(y)` within the appropriate geom function. Manually assign the marker `size` to be 3.

HINT: Do NOT use `geom_jitter()`.

This figure is similar to the figure in 1f) but you are using the larger data set, `df`.

```
ggplot(df, aes(x = x1, y = x2, color = as.factor(y), shape = as.factor(y))) +
  geom_point(size = 3) +
  labs(title = "Scatter Plot of x1 vs x2 with y",
       x ="x1",
       y ="x2",
       color = "y",
       shape = "y") +
  theme_minimal()
```



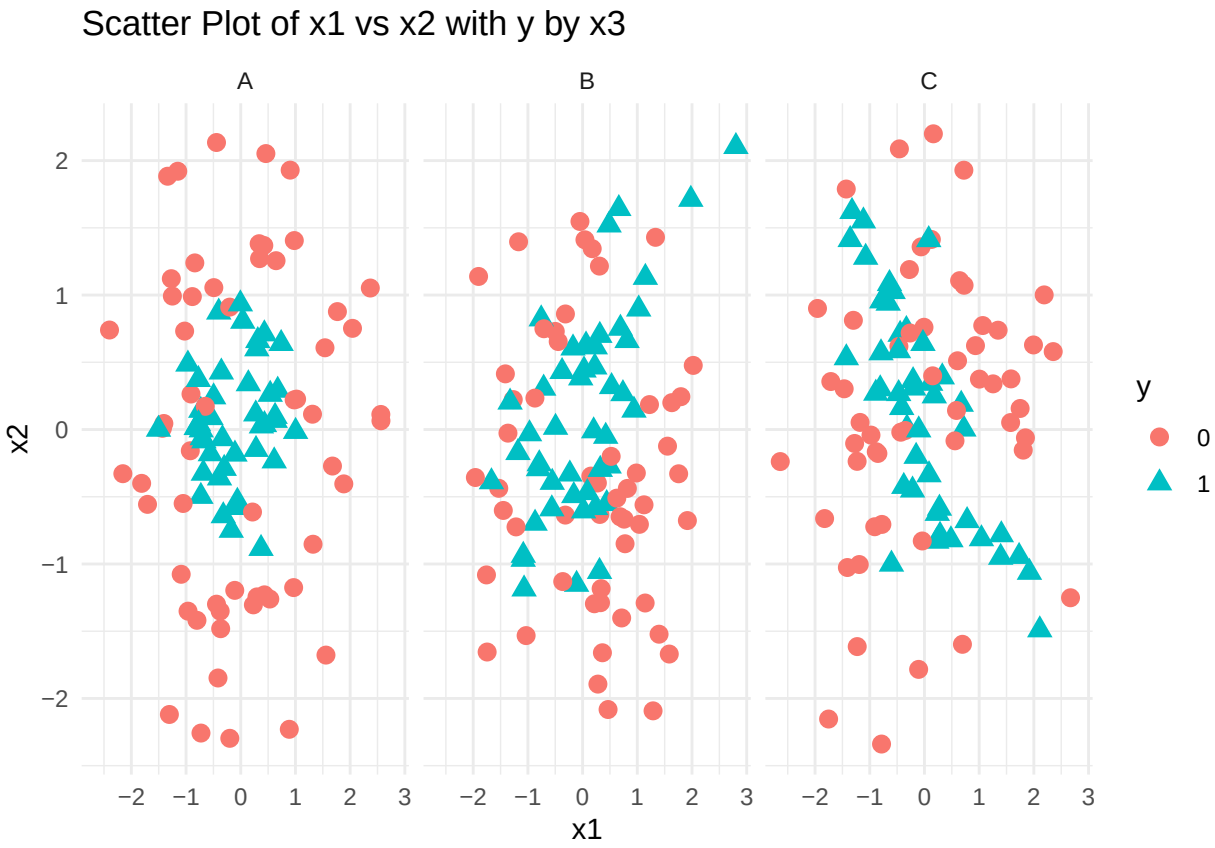
SOLUTION

2c)

Let's now include the effect of the categorical input!

Create the same scatter plot as 2b), but this time include facets based on the categorical input `x3`. You must still use the marker `color` and `shape` aesthetics to denote the binary outcome.

```
ggplot(df, aes(x = x1, y = x2, color = as.factor(y), shape = as.factor(y))) +
  geom_point(size = 3) +
  facet_wrap(~x3) +
  labs(title = "Scatter Plot of x1 vs x2 with y by x3",
       x ="x1",
       y ="x2",
       color = "y",
       shape = "y") +
  theme_minimal()
```



SOLUTION

2d)

How would you describe the behavior of the binary outcome relative to the 3 inputs, based on the figure created in 2c)?

SOLUTION

1) There is a visible difference in the distribution of data points across the three categories (A, B, C) of the categorical variable `x3`. This suggests that `x3` may have an interaction effect with `x1` and `x2` on the binary outcome `y`.
2) There is a visible overlap between the points representing y = 0 and y = 1 across all facets. This suggests that the binary outcome is not linearly separable by x1 and x2 alone.
3) The relationship between x1 and x2 with respect to y does not appear to be strictly linear, as the points do not align along a straight line. This might suggest that a logistic regression model with just linear terms may not be sufficient to model the data accurately.

Problem 03

You fit multiple logistic regression models ranging from simple to complex in the previous assignment. That is the best way to go about a modeling exercise because it allows us to learn the kinds of features required to improve model behavior. We are able to build a "story" about what it takes to predict the output!

However, we will just fit a single logistic regression model in this assignment. This model is relatively complex. We are "short cutting" the modeling workflow so we can focus on predictions. Later in the assignment, we will use the elastic net penalty to try to turn the complex model into a simpler model.

3a)

Fit a non-Bayesian logistic regression model which interacts the categorical input with all linear main effects, interactions between the two continuous inputs, and quadratic features associated with both continuous inputs. Thus, ALL features derived from the continuous inputs must interact with the categorical variable.

Assign the model to the `fit_glm` variable.

HINT: Be careful with the `family` argument!

```
fit_glm <- glm(y ~ (x1 + x2 + I(x1^2) + I(x2^2) + x1:x2) * x3, data = df, family = "binomial")
```

SOLUTION

3b)

Are the interaction features considered statistically significant according to the non-Bayesian logistic regression model?

```
summary(fit_glm)
```

SOLUTION

```
##
## Call:
## glm(formula = y ~ (x1 + x2 + I(x1^2) + I(x2^2) + x1:x2) * x3,
##     family = "binomial", data = df)
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)   3.9782     0.8496   4.682 2.84e-06 ***
## x1           -0.3150     0.5896  -0.534 0.593156
## x2            0.3181     0.6900   0.461 0.644834
## I(x1^2)      -3.3288     0.9169  -3.630 0.000283 ***
## I(x2^2)      -4.3014     1.0934  -3.934 8.36e-05 ***
## x3B          -1.9195     0.9895  -1.940 0.052405 .
## x3C          -2.2758     0.9637  -2.362 0.018198 *
## x1:x2         0.4463     1.3513   0.330 0.741205
## x1:x3B        0.1844     0.6745   0.273 0.784569
## x1:x3C        0.6564     0.7257   0.905 0.365716
## x2:x3B        0.4963     0.7999   0.620 0.534953
## x2:x3C       -0.6244     0.8683  -0.719 0.472041
## I(x1^2):x3B   1.5378     1.0122   1.519 0.128681
## I(x1^2):x3C   1.1709     1.0444   1.121 0.262241
## I(x2^2):x3B   2.0804     1.2474   1.668 0.095360 .
## I(x2^2):x3C   2.6004     1.2740   2.041 0.041236 *
## x1:x2:x3B     3.0511     1.5700   1.943 0.051961 .
## x1:x2:x3C    -4.4796     1.6633  -2.693 0.007078 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 412.47  on 299  degrees of freedom
```

```
## Residual deviance: 204.25  on 282  degrees of freedom
## AIC: 240.25
##
## Number of Fisher Scoring iterations: 8
```

The asterisks next to the p-values in the output denote the level of significance: - *** for $p < 0.001$ - ** for $p < 0.01$ - * for $p < 0.05$ - . for $p < 0.1$

This indicates that the interaction features `I(x2^2):x3B`, `I(x2^2):x3C`, `x1:x2:x3B`, and `x1:x2:x3C` are considered statistically significant.

3c)

Examining the coefficients is one way to try to interpret model behavior. However, we can also use predictions to visualize the relationships. This is especially helpful with binary classification problems because it allows us to visually identify regions of low event probability vs regions of high event probability.

The code chunk below defines an input visualization grid for you. It is a full factorial grid between the 3 inputs. We will focus on the patterns relative to the continuous inputs, which is why there are 75 unique values for each input. The 75 x 75 input grid is repeated for each unique value of the categorical input. Thus, there are 16875 input combinations in the visualization grid. You must make 16875 predictions to study the model behavior!

```
viz_grid <- expand.grid(x1 = seq(-3, 3, length.out=75),
                        x2 = seq(-3, 3, length.out=75),
                        x3 = c("A", "B", "C"),
                        KEEP.OUT.ATTRS = FALSE,
                        stringsAsFactors = FALSE) %>%
  as.data.frame() %>% tibble::as_tibble()

viz_grid %>% glimpse()
```

```
## Rows: 16,875
## Columns: 3
## $ x1 <dbl> -3.000000, -2.918919, -2.837838, -2.756757, -2.675676, -2.594595, -~
## $ x2 <dbl> -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3,~
## $ x3 <chr> "A", "A", "A", "A", "A", "A", "A", "A", "A", "A", "A", "A", "A", "A~
```

You should have fit the logistic regression model with the `glm()` function. Predictions from `glm()` fit objects are made with the `predict()` function, just like `lm()` fit models. However, *generalized* linear models make predictions for the **linear predictor** as well as the **mean output trend**. Therefore, we need to make sure we are making the predictions of the correct type of variable!

You will first make predictions with the `fit_glm` model using the default arguments to `predict()`.

Predict the input visualization grid with the `fit_glm` model. The first argument to `predict()` is the model. The second argument is `newdata` and should be named in the `predict()` function call. Assign the `viz_grid` object to the `newdata` argument to ensure the input visualization grid is predicted.

Assign the result to the `pred_viz_glm_a` variable and display the `summary()` of the `pred_viz_glm_a` to the screen.

```
pred_viz_glm_a <- predict(fit_glm, newdata = viz_grid)

summary(pred_viz_glm_a)
```

SOLUTION

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -70.6089 -22.5156  -8.4844 -13.3402  -0.9782   5.2428
```

3d)

The `predict()` function has additional arguments which control how the predictions are made. You will make predictions again, but this time you must include a third argument, `type`, in the `predict()` function call. You must set `type = 'response'` in the `predict()` call.

Predict the input visualization grid with the `fit_glm` model again. Use the same two arguments as in 3c) but this time include the third argument `type` with the value assigned to `'response'`.

Assign the result to the `pred_viz_glm_b` variable and display the `summary()` of the `pred_viz_glm_b` to the screen.

```
pred_viz_glm_b <- predict(fit_glm, newdata = viz_grid, type = 'response')

summary(pred_viz_glm_b)
```

SOLUTION

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## 0.0000000 0.0000000 0.0002066 0.1884831 0.2732392 0.9947423
```

3e)

You made two types of predictions. One corresponds to the linear predictor while one corresponds to the output mean trend. The output mean is the **event probability** in logistic regression problems.

Which of the two predictions, `pred_viz_glm_a` or `pred_viz_glm_b`, corresponds to the event probability? How do you know based on the two previous results?

SOLUTION The predictions stored in `pred_viz_glm_b` correspond to the event probability. This is because the `predict()` function was called with the argument `type = 'response'`, which for logistic regression models, gives the predicted probabilities of the response variable being 1.

3f)

Let's now visualize the event probability surface with respect to the three inputs! You will made a *raster* plot to visualize the predicted probability surface with respect to the two continuous inputs faceted by the categorical variable. Raster plots will look like *images* and are possible when we have full factorial input grids.

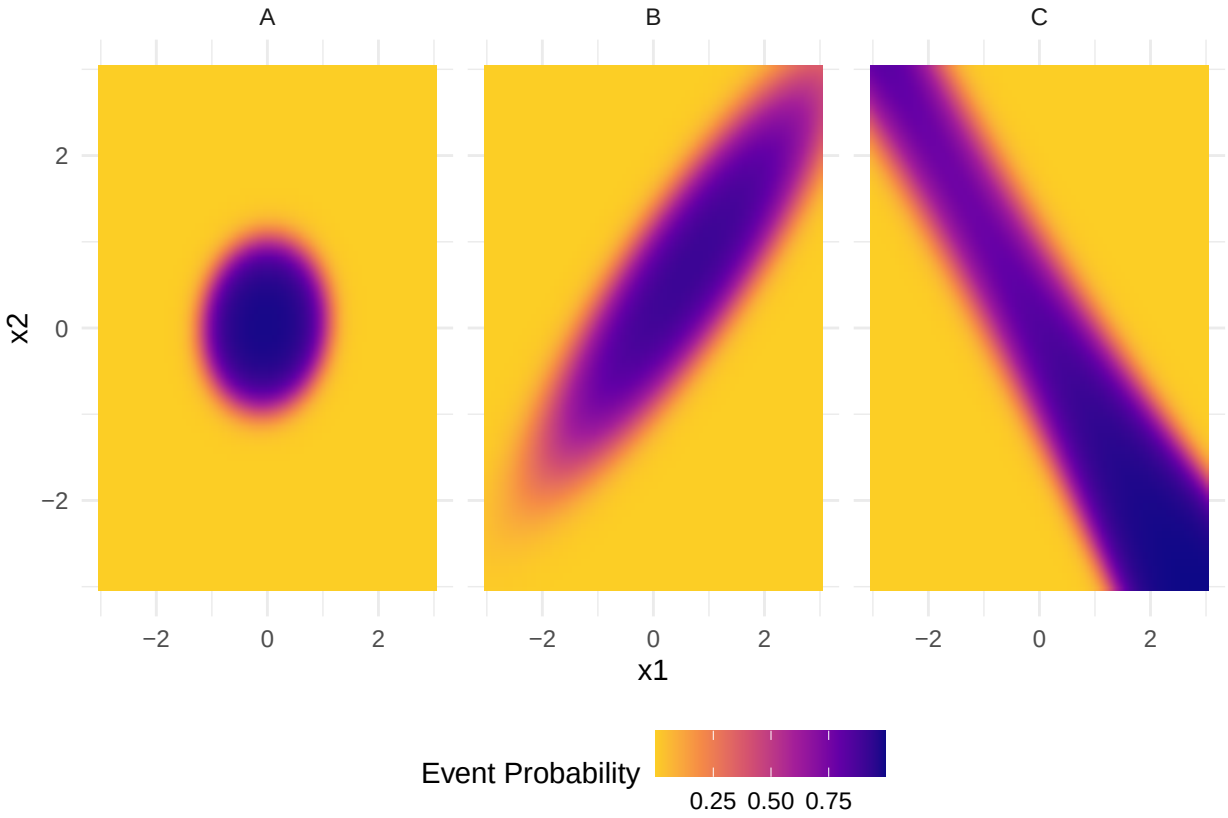
Pipe the `viz_grid` dataframe to the `mutate()` function. Include the variable `mu` to the dataframe and assign the predicted probability to the `mu` variable. Pipe the result to `ggplot()` and map the `x1` and `x2` variables to the `x` and `y` aesthetics, respectively. Add a `geom_raster()` layer and map the `fill` aesthetic to the `mu` variable. Include facets via the `facet_wrap()` function with the facets "a function of `x3`". Lastly, specify the fill color palette via the `scale_fill_viridis_c()` function.

HINT: You identified which of the predictions corresponds to the event probability in 3e)!

HINT: Don't forget about the importance of the `aes()` function!

```
viz_grid %>%
  mutate(mu = pred_viz_glm_b) %>%
```

```r
ggplot(aes(x = x1, y = x2, fill = mu)) +
geom_raster(interpolate = TRUE) +
facet_wrap(~x3) +
scale_fill_viridis_c(option = "C", direction = -1, end = 0.9) +
labs(fill = "Event Probability") +
theme_minimal() +
theme(legend.position = "bottom")
```



SOLUTION

3g)

The previous figure used a sequential color palette to represent the event probability. Low probabilities are dark blue while high probabilities are bright yellow within the viridis color palette. However, event probabilities have a natural *boundary* to focus on. We want to identify the 50% probability *decision boundary* because 0.5 is the common threshold for *classifying* events in predictions.

We can represent the 50% decision boundary by using a *diverging* palette. The diverging palette focuses on behavior away from a central value. One color represents increasing values from the midpoint while another color represents decreasing values from the midpoint. Classification problems have a natural midpoint value of 0.5 and thus the diverging colors will represent probabilities increasing away from 50% and probabilities decreasing away from 50%. For this problem, you should use the following syntax to create the diverging palette:

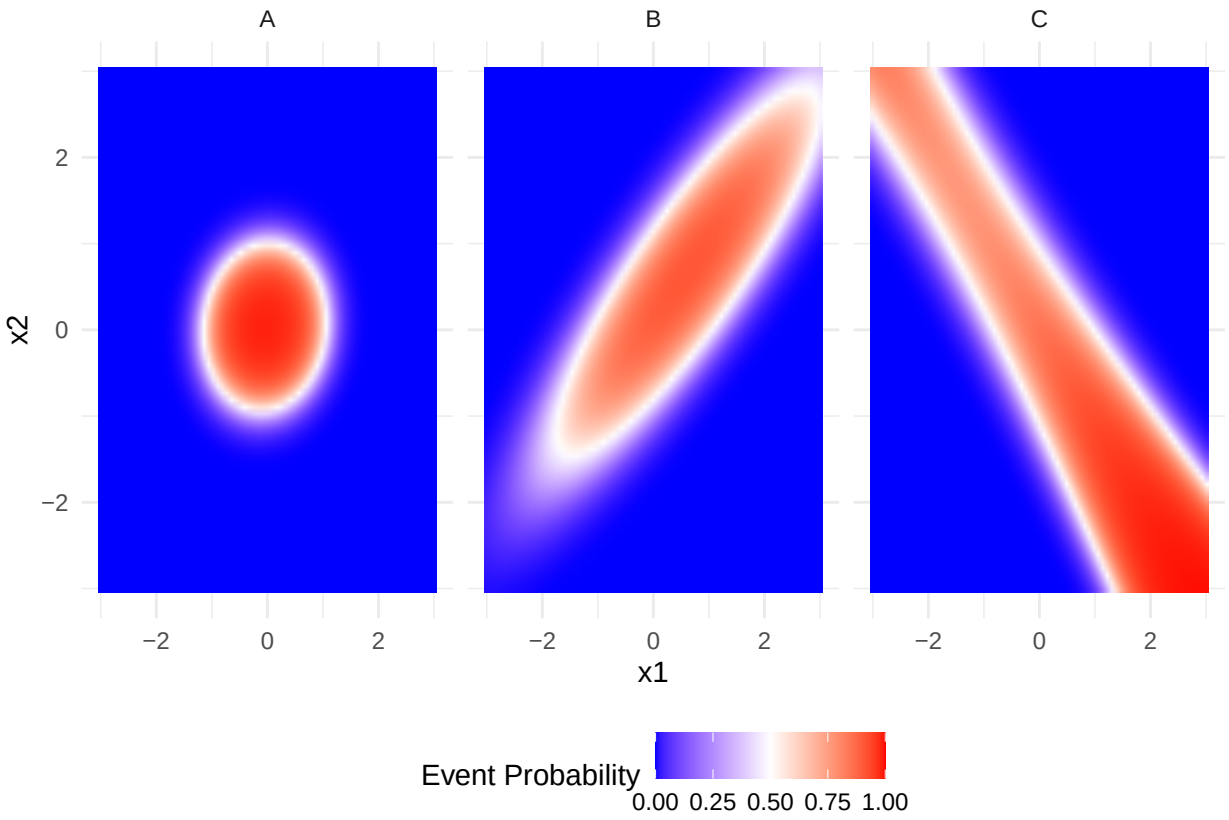
```r
# do NOT set eval=TRUE in this code chunk!!!!
scale_fill_gradient2(low = 'blue', mid = 'white', high = 'red',
                     midpoint = 0.5,
                     limits = c(0, 1))
```

The syntax specifies that the midpoint is 0.5 while the limits are 0 and 1. Values decreasing from the midpoint are represented by blue colors, while values increasing away from the midpoint are represented by

red colors. The midpoint value (0.5 in this case) corresponds to white. Thus, the 50% *decision boundary* will be represented by separating blue and red regions by a white boundary!

Create the same figure as in 3f) but this time replace `scale_fill_viridis_c()` with the diverging scale.

```
viz_grid %>%
  mutate(mu = pred_viz_glm_b) %>%
  ggplot(aes(x = x1, y = x2, fill = mu)) +
  geom_raster(interpolate = TRUE) +
  facet_wrap(~x3) +
  scale_fill_gradient2(low = 'blue', mid = 'white', high = 'red', midpoint = 0.5, limits = c(0, 1)) +
  labs(fill = "Event Probability") +
  theme_minimal() +
  theme(legend.position = "bottom")
```



SOLUTION

3h)

What are the general shapes of the 50% decision boundary for your model? Does the boundary shape depend on the categorical variable?

SOLUTION The shape of the decision boundary does depend on the categorical variable x3. Each category (A, B, and C) shows a distinct interaction effect between x1 and x2, which affects the probability of the event occurring. This interaction is visually represented by the change in the orientation and shape of the decision boundary across the facets.

These visual patterns align with the idea that the categorical variable modifies the relationship between the continuous inputs and the outcome, underlining the importance of interaction terms in the logistic regression

model.

Problem 04

We fit a relatively complex model in the previous problem. Let's use regularization to see if a simpler model will improve performance. You will use non-Bayesian regularization in this assignment via the `glmnet` package. You will tune the elastic net's mixing fraction, `alpha`, and regularization strength, `lambda`, via the `caret::train()` function rather than using `glmnet` directly.

The `caret` package is loaded for you in the code chunk below.

```
library(caret)
```

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

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:purrr':
##
##     lift
```

The `caret` package prefers the binary outcome to be organized as a factor data type compared to an integer. The data set is reformatted for you in the code chunk below. The binary outcome `y` is converted to a new variable `outcome` with values `'event'` and `'non_event'`. The first level is forced to be `'event'` to be consistent with the `caret` preferred structure.

```
df_caret <- df %>%
  mutate(outcome = ifelse(y == 1, 'event', 'non_event')) %>%
  mutate(outcome = factor(outcome, levels = c("event", "non_event"))) %>%
  select(x1, x2, x3, outcome)

df_caret %>% glimpse()
```

```
## Rows: 300
## Columns: 4
## $ x1      <dbl> -0.1737886, -0.3883115, -0.7308443, -1.3366670, -0.4435314, 0.~
## $ x2      <dbl> -0.750485806, -0.361116231, 0.141729009, 1.883642959, 2.134528~
## $ x3      <chr> "A", "A", "A", "A", "A", "A", "A", "A", "A", "A", "A", "A", "A~
## $ outcome <fct> event, event, event, non_event, non_event, non_event, event, n~
```

4a)

You must always specify the resampling scheme and primary performance metric when using `caret`. You will use the same resampling as the previous assignment, 10 fold cross-validation with 3 repeats. You will also use the Accuracy as the primary performance metric.

Specify the resampling scheme to be 10 fold with 3 repeats. Assign the result of the `trainControl()` function to the `my_ctrl` object. Specify the primary performance metric to be `'Accuracy'` and assign that to the `my_metric` object.

```
my_ctrl <- trainControl(method = "repeatedcv",
                        number = 10,
                        repeats = 3)

my_metric <- "Accuracy"
```

SOLUTION

4b)

You must train, assess, and tune an elastic net model using the default `caret` tuning grid. In the `caret::train()` function you must use the formula interface to specify a model consistent with `fit_glm`. Your model must interact the categorical input with all linear main effects, interactions between the two continuous inputs, and quadratic features associated with both continuous inputs. Thus, ALL features derived from the continuous inputs must interact with the categorical variable. **Please pay close attention to your formula.** The binary outcome is now named `outcome` and **not** `y`. Assign the method argument to `'glmnet'` and set the metric argument to `my_metric`. You **must** also instruct `caret` to standardize the features by setting the `preProcess` argument equal to `c('center', 'scale')`. This will give you practice standardizing inputs even though it is not critical for this particular application. Assign the `trControl` argument to the `my_ctrl` object.

Important: The `caret::train()` function works with the formula interface. Thus, even though you are using `glmnet` to fit the model, `caret` does not require you to organize the design matrix as required by `glmnet`! Thus, you do **NOT** need to remove the intercept when defining the formula to `caret::train()`!

Train, assess, and tune the `glmnet` elastic net model with the defined resampling scheme. Assign the result to the `enet_default` object and display the result to the screen.

Which tuning parameter combinations are considered to be the best?

Is the best set of tuning parameters more consistent with Lasso or Ridge regression?

SOLUTION The random seed is set for you for reproducibility.

```
set.seed(1234)

enet_default <- train(outcome ~ (x1 + x2 + I(x1^2) + I(x2^2) + x1:x2) * x3,
                      data = df_caret,
                      method = 'glmnet',
                      metric = my_metric,
                      preProcess = c('center', 'scale'),
                      trControl = my_ctrl)

print(enet_default)
```

```
## glmnet
##
## 300 samples
##   3 predictor
##   2 classes: 'event', 'non_event'
##
## Pre-processing: centered (17), scaled (17)
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 270, 270, 270, 269, 270, 270, ...
## Resampling results across tuning parameters:
##
##   alpha  lambda       Accuracy   Kappa
##   0.10   0.000303919  0.8379162  0.6773278
##   0.10   0.003039190  0.8434742  0.6887258
##   0.10   0.030391898  0.8345112  0.6721731
##   0.55   0.000303919  0.8379162  0.6773278
##   0.55   0.003039190  0.8445853  0.6908893
##   0.55   0.030391898  0.8325090  0.6682221
##   1.00   0.000303919  0.8368409  0.6749809
##   1.00   0.003039190  0.8457706  0.6928615
```

```
##   1.00   0.030391898  0.8281028  0.6591729
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were alpha = 1 and lambda = 0.00303919.
```

1) The final values used for the model were alpha = 1 and lambda = 0.00303919. This tuning parameter combination is considered to be the best.

2) Since alpha = 1, tuning parameters are consistent with Lasso regression.

4c)

Create a custom tuning grid to further tune the elastic net `lambda` and `alpha` tuning parameters.

Create a tuning grid with the `expand.grid()` function which has two columns named `alpha` and `lambda`. The `alpha` variable should be evenly spaced between 0.1 and 1.0 by increments of 0.1. The `lambda` variable should have 25 evenly spaced values in the log-space between the minimum and maximum `lambda` values from the caret default tuning grid. Assign the tuning grid to the `enet_grid` object.

How many tuning parameter combinations are you trying out? How many total models will be fit assuming the 10-fold with 3-repeat resampling approach?

HINT: The `seq()` function includes an argument `by` to specify the increment width.

HINT: Do not convert the `expand.grid()` result to a dataframe or tibble.

```
alpha_values <- seq(0.1, 1.0, by = 0.1)
lambda_values <- exp(seq(log(min(enet_default$results$lambda)),
                         log(max(enet_default$results$lambda)),
                         length.out = 25))

enet_grid <- expand.grid(alpha = alpha_values, lambda = lambda_values)
```

```
num_combinations <- nrow(enet_grid)
num_folds <- 10
num_repeats <- 3
total_models <- num_combinations * num_folds * num_repeats

num_combinations
```

SOLUTION

```
## [1] 250
```

```
total_models
```

```
## [1] 7500
```

1) As computed above, we are trying out 250 combinations.

2) Also, 7500 models will be fit assuming the 10-fold with 3-repeat resampling approach.

4d)

Train, assess, and tune the elastic net model with the custom tuning grid and assign the result to the `enet_tune` object. You should specify the arguments to `caret::train()` consistent

with your solution in Problem 4b), except you should also assign `enet_grid` to the `tuneGrid` argument.

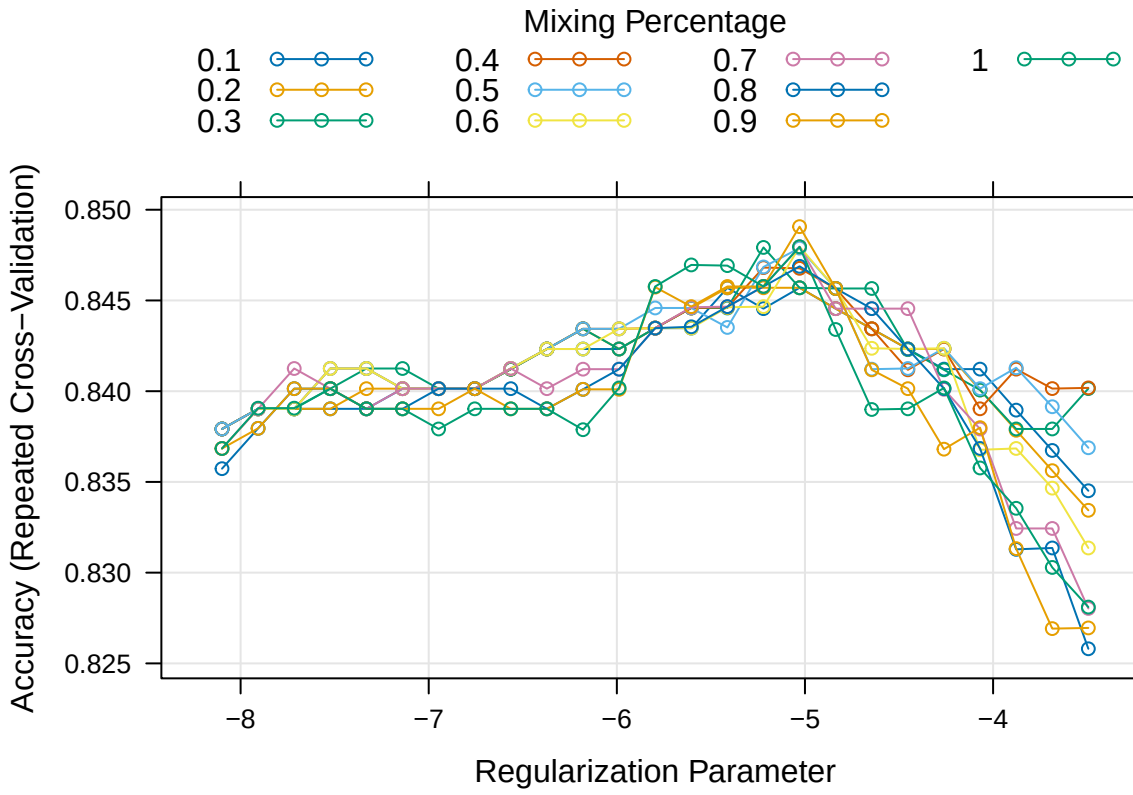
Do not print the result to the screen. Instead use the default plot method to visualize the resampling results. Assign the `xTrans` argument to `log` in the default plot method call. Use the `$bestTune` field to print the identified best tuning parameter values to the screen. Is the identified best elastic net model more similar to Lasso or Ridge regression?

SOLUTION The random seed is set for you for reproducibility. You may add more code chunks if you like.

```
set.seed(1234)

enet_tune <- train(outcome ~ (x1 + x2 + I(x1^2) + I(x2^2) + x1:x2) * x3,
                   data = df_caret,
                   method = 'glmnet',
                   metric = my_metric,
                   preProcess = c('center', 'scale'),
                   trControl = my_ctrl,
                   tuneGrid = enet_grid)

plot(enet_tune, xTrans = log)
```




```
enet_tune$bestTune
```

```
##     alpha      lambda
## 217   0.9 0.006547736
```

As it can be seen above, the best tuning parameter values are alpha = 0.9 and lambda = 0.006547736. Since alpha is closer to 1, we conclude that the identified best elastic net model is more similar to Lasso regression.

4e)

Print the coefficients to the screen for the tuned elastic net model. Which coefficients are non-zero? Has the model been converted to a simpler model?

```
coef_enet_tune <- coef(enet_tune$finalModel, s = enet_tune$bestTune$lambda, alpha = enet_tune$bestTune$a

print(coef_enet_tune)
```

SOLUTION

```
## 18 x 1 sparse Matrix of class "dgCMatrix"
##                         s1
## (Intercept)  1.348946104
## x1           .
## x2           .
## I(x1^2)      2.210073430
## I(x2^2)      2.200714730
## x3B          0.009590669
## x3C          0.148530528
## x1:x2        .
## x1:x3B       .
## x1:x3C      -0.162193989
## x2:x3B      -0.312530264
## x2:x3C       0.040287028
## I(x1^2):x3B  .
## I(x1^2):x3C  0.066845147
## I(x2^2):x3B  .
## I(x2^2):x3C -0.161467870
## x1:x2:x3B   -1.820352623
## x1:x2:x3C    1.898665489
```

The coefficients are printed above. All the coefficients except `x1`, `x2`, `x1:x2`, `x1:x3B`, `I(x1^2):x3B` and `I(x2^2):x3B` are nonzero, which shows that the complex model has been converted to a simpler model.

4f)

Let's visualize the predictive trends of the event probability from the tuned elastic net model. The `predict()` function for `caret` trained classifiers is different from the operation of `predict()` for `glm()` trained objects. You made predictions from `caret` trained binary classifiers in the previous assignment. The `type` argument is different for `caret` trained objects compared to `glm()` trained objects.

The first argument to `predict()` for `caret` trained objects is the `caret` trained model and the second object, `newdata`, is the new data set to make predictions with. Earlier in the semester in homework 03, you made predictions from `caret` trained binary classifiers. That assignment discussed that the optional third argument `type` dictated the "type" of prediction to make. Setting `type = 'prob'` instructs the `predict()` function to return the class predicted probabilities.

Complete the code chunk below. You must make predictions on the visualization grid, `viz_grid`, using the tuned elastic net model `enet_tune`. Instruct the `predict()` function to return the probabilities by setting `type = 'prob'`.

```
pred_viz_enet_probs <- predict(enet_tune, newdata = viz_grid, type = 'prob')
```

SOLUTION

4g)

The code chunk below is completed for you. The `pred_viz_enet_probs` dataframe is column binded to the `viz_grid` dataframe. The new object, `viz_enet_df`, provides the class predicted probabilities for each input combination in the visualization grid. A glimpse is printed to the screen. Please not the `eval` flag is set to `eval=FALSE` in the code chunk below. You must change `eval` to `eval=TRUE` in the chunk options to make sure the code chunk below runs when you knit the markdown.

```
viz_enet_df <- viz_grid %>% bind_cols(pred_viz_enet_probs)

viz_enet_df %>% glimpse()
```

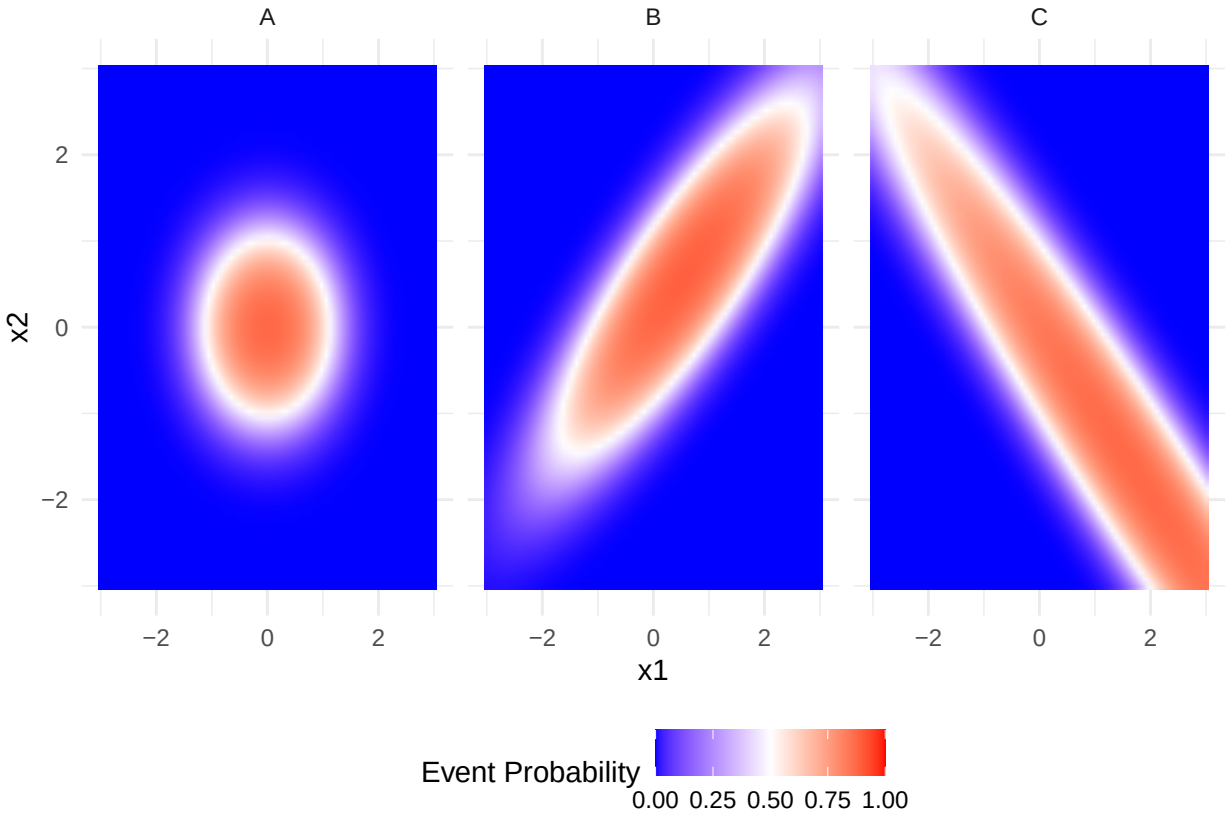
```
## Rows: 16,875
## Columns: 5
## $ x1        <dbl> -3.000000, -2.918919, -2.837838, -2.756757, -2.675676, -2.59~
## $ x2        <dbl> -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, ~
## $ x3        <chr> "A", "A", "A", "A", "A", "A", "A", "A", "A", "A", "A", "A", ~
## $ event     <dbl> 2.298162e-13, 4.891643e-13, 1.019629e-12, 2.082334e-12, 4.16~
## $ non_event <dbl> 1.0000000, 1.0000000, 1.0000000, 1.0000000, 1.0000000, 1.000~
```

The glimpse reveals that the `event` column stores the **predicted event probability**. You must visualize the predicted probability as a faceted raster plot just like you did in 3g). You must use the diverging palette to show the 50% decision boundary.

Visualize the predicted probability from the tuned elastic net model with respect to `x1` and `x2` using `geom_raster()` faceted by `x3`. You must use the same diverging palette as 3g).

HINT: The event probability was named `mu` when you made the previous figure in 3g). The event probability is now named `event`!

```
ggplot(viz_enet_df, aes(x = x1, y = x2, fill = event)) +
  geom_raster(interpolate = TRUE) +
  scale_fill_gradient2(low = 'blue', mid = 'white', high = 'red', midpoint = 0.5, limits = c(0, 1)) +
  facet_wrap(~x3) +
  labs(fill = "Event Probability") +
  theme_minimal() +
  theme(legend.position = "bottom")
```



SOLUTION

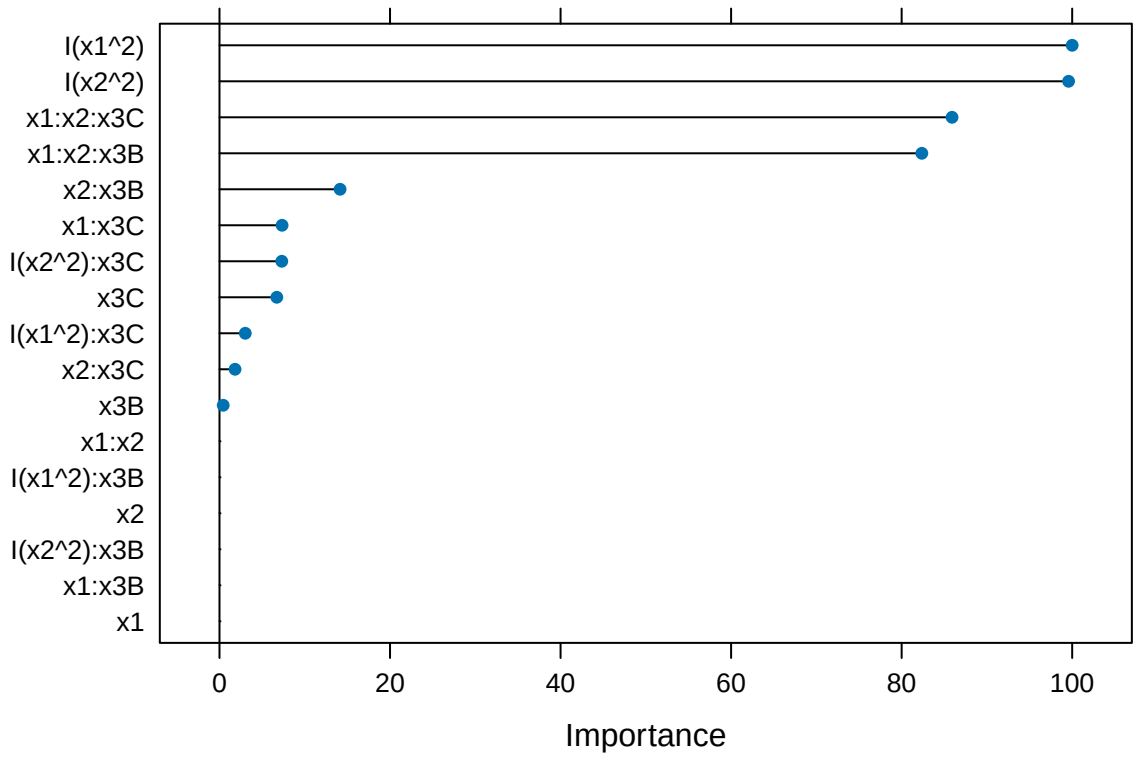
4h)

Create a plot to show the variable importance rankings associated with the tuned elastic net model. Are the importance rankings consistent with the visualization of the predicted probability?

HINT: You visualized variable importance rankings from `caret` trained models in HW09.

```
enet_tune_imp <- varImp(enet_tune, scale=TRUE)

plot(enet_tune_imp)
```



SOLUTION

1) The quadratic terms I(x1^2) and I(x2^2) are among the most important predictors, which is consistent with the non-linear decision boundaries observed in the plots.

2) Interactions involving x3C seem to be important, which aligns with the distinct decision boundary shapes in the raster plot for category C.

3) The importance of x1:x2 interactions is lower, suggesting that the simple interaction between x1 and x2 without considering x3 is less critical.

In conclusion, there appears to be a general consistency between the variable importance rankings and the visualization of the predicted probability.

Problem 05

The data in this assignment are not challenging. The point was to demonstrate that we can train **generalized linear models** with non-linear features to learn non-linear decision boundaries! We improved the performance by *manually* deriving the non-linear features.

Modern machine learning applications also involve more advanced models that do not require us to manually derive the non-linear features. Instead, the models attempt to *learn* the non-linear relationships for us. Such models can be very accurate, however these advanced models are not as easy to interpret as generalized linear models. That is why it is critical to train generalized linear models before training any advanced method like neural networks, random forests, and gradient boosted trees.

Even though this is a relatively simple application, let's train several advanced methods to compare to the tuned elastic net model. This will allow us to compare the performance and predictive trends of these more advanced models to simpler and easier to interpret models. This provides context for understanding why the more advanced models out perform the simpler ones!

Important: Each of the advanced non-linear models attempt to learn *basis functions* for us. Thus, you will not specify polynomials, or natural splines, or any other function within the formula. Instead, you should type the formula in `caret::train()` as if you are using linear additive features. The non-linear models will "create" the non-linear features for you. The `df_caret` dataframe only consists of inputs, `x1` through `x3`, and

the binary outcome, `outcome`. Thus you can use the `.` "shortcut" operator to define the formula for each non-linear model:

```
outcome ~ .
```

This will save on typing **but** can only be used because the `df_caret` dataframe only contains the inputs and output. If there were other variables within the dataframe and we did not want to use then we could not use the `.` operator like that.

5a)

You will begin by training a neural network via the `nnet` package. `caret` will prompt you to install `nnet` if you do not have it installed already. Please open the R Console to "see" the prompt messages to help with the installation.

You will first use the default `caret` tuning grid for `nnet`. You will train a neural network to classify the binary outcome, `outcome`, with respect to the continuous and categorical input. As previously stated, you may use the `.` "shortcut" operator in the formula to `caret::train()` to say the output is a function of "everything else" in the dataframe. Assign the `method` argument to `'nnet'` and set the `metric` argument to `my_metric`. You must also instruct `caret` to standardize the features by setting the `preProcess` argument equal to `c('center', 'scale')`. Assign the `trControl` argument to the `my_ctrl` object.

You are therefore using the same resampling scheme for the neural network as you did with the elastic net model! This will allow directly comparing the neural network performance to the elastic net model!

Train, assess, and tune the `nnet` neural network with the defined resampling scheme. Assign the result to the `nnet_default` object and print the result to the screen. Which tuning parameter combinations are considered to be the best?

IMPORTANT: include the argument `trace = FALSE` in the `caret::train()` function call. This will make sure the `nnet` package does **NOT** print the optimization iteration results to the screen.

SOLUTIOn The random seed is set for you for reproducibility. You may add more code chunks if you like.

```
set.seed(1234)

nnet_default <- caret::train(outcome ~ .,
                      data = df_caret,
                      method = 'nnet',
                      metric = my_metric,
                      preProcess = c('center', 'scale'),
                      trControl = my_ctrl,
                      trace = FALSE)

print(nnet_default)
```

```
## Neural Network
##
## 300 samples
##   3 predictor
##   2 classes: 'event', 'non_event'
##
## Pre-processing: centered (4), scaled (4)
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 270, 270, 270, 269, 270, 270, ...
## Resampling results across tuning parameters:
##
##   size  decay  Accuracy   Kappa
```

```
##   1     0e+00  0.5786751  0.2016377
##   1     1e-04  0.5867421  0.2194420
##   1     1e-01  0.5480324  0.1186581
##   3     0e+00  0.7075343  0.4106376
##   3     1e-04  0.6876777  0.3677065
##   3     1e-01  0.7277308  0.4539138
##   5     0e+00  0.7510691  0.5021199
##   5     1e-04  0.7380991  0.4754689
##   5     1e-01  0.8210963  0.6406258
##
## Accuracy 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 final values used for the model were size = 5 and decay = 0.1. This tuning parameter combination is considered to be the best.

5b)

The default neural network tuning grid uses a small number of **hidden units** or *neurons*. This makes sure the default training time is relatively fast. Let's try several more complicated neural networks by adding more hidden units in the hidden layer!

You must define a custom tuning grid for the neural network. This tuning grid has 2 tuning parameters, similar to the elastic net tuning grid. The first tuning parameter `size` is the number of hidden units in the hidden layer and the second tuning parameter is `decay`. The `decay` is the regularization strength of the **ridge** penalty. Thus, `nnet` is training ridge regularized neural networks!

Create a tuning grid with the `expand.grid()` function which has two columns named `size` and `decay`. The `size` variable have 4 unique values of 5, 9, 13, 17, and 21. The `decay` variable must be positive, but you will define its search grid in the log-space. You must apply the `exp()` function to 11 evenly spaced values between -6 and 0. Assign the tuning grid to the `nnet_grid` object.

How many tuning parameter combinations will you try?

```
nnet_grid <- expand.grid(size = c(5, 9, 13, 17, 21), decay = exp(seq(-6, 0, length.out = 11)))

nrow(nnet_grid)
```

SOLUTION

```
## [1] 55
```

55 tuning parameter combinations will be tried.

5c)

Train, assess, and tune the `nnet` neural network with the custom tuning grid and the defined resampling scheme. Assign the result to the `nnet_tune` object. You should specify the arguments to `caret::train()` consistent with your solution in Problem 5a), except you should also assign `nnet_grid` to the `tuneGrid` argument.

Do not print the result to the screen. Instead use the default plot method to visualize the resampling results. Assign the `xTrans` argument to `log` in the default plot method call. Use the `$bestTune` field to print the identified best tuning parameter values to the screen. How many hidden units are associated with the best model?

IMPORTANT: include the argument `trace = FALSE` in the `caret::train()` function call. This will make sure the `nnet` package does **NOT** print the optimization iteration results to the screen.

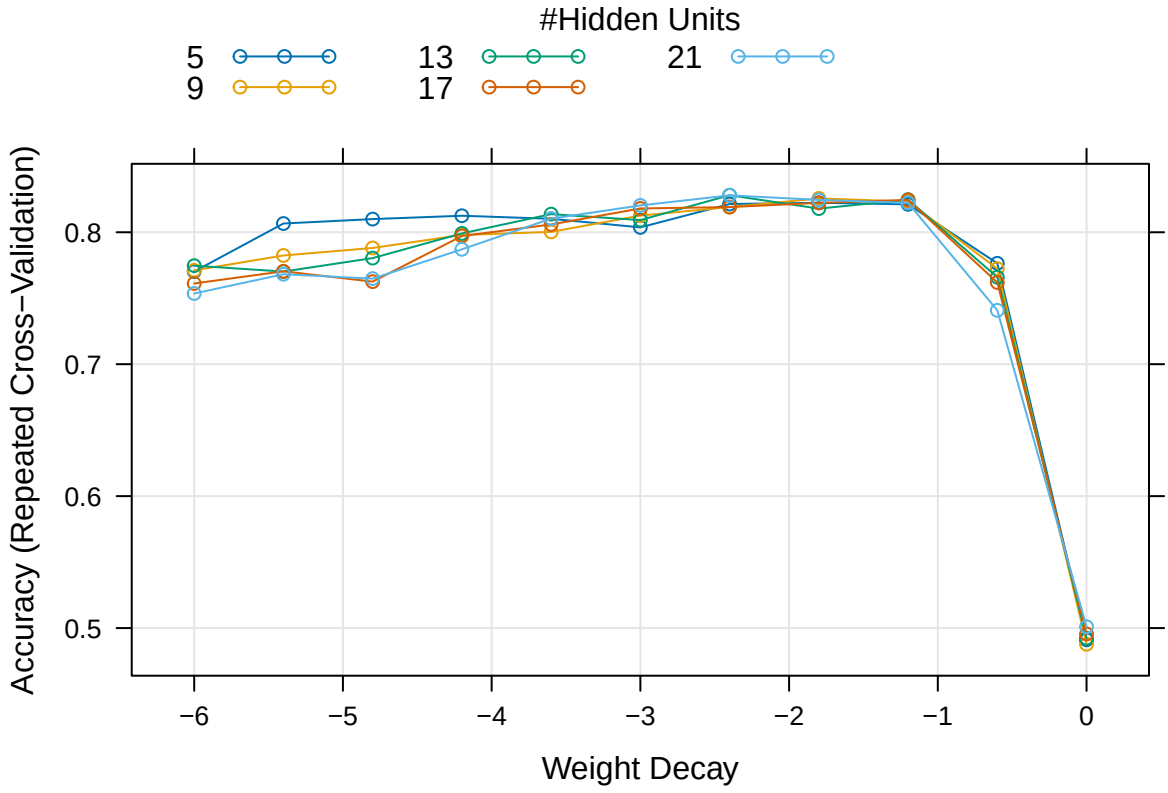
SOLUTIOn The random seed is set for you for reproducibility. You may add more code chunks if you like.

PLEASE NOTE: This code chunk may take several minutes to complete!

```
set.seed(1234)

nnet_tune <- train(outcome ~ .,
                   data = df_caret,
                   method = 'nnet',
                   tuneGrid = nnet_grid,
                   metric = my_metric,
                   preProcess = c('center', 'scale'),
                   trControl = my_ctrl,
                   trace = FALSE)

plot(nnet_tune, xTrans = log)
```




```
print(nnet_tune$bestTune)
```

```
##    size      decay
## 29   13 0.09071795
```

Number of hidden units in the best model is 13.

5d)

Let's use predictions to examine the behavior of the neural network in greater detail. You will predict the same input visualization grid, `viz_grid`, as the previous models.

Complete the code chunk below. You must make predictions on the visualization grid, `viz_grid`, using the tuned neural network model `nnet_tune`. Instruct the `predict()` function to return the probabilities by setting `type = 'prob'`.

```
pred_viz_nnet_probs <- predict(nnet_tune, newdata = viz_grid, type = 'prob')
```

SOLUTION

5e)

The code chunk below is completed for you. The `pred_viz_nnet_probs` dataframe is column binded to the `viz_grid` dataframe. The new object, `viz_nnet_df`, provides the class predicted probabilities for each input combination in the visualization grid according to the tuned neural network. A glimpse is printed to the screen. Please not the `eval` flag is set to `eval=FALSE` in the code chunk below. You must change `eval` to `eval=TRUE` in the chunk options to make sure the code chunk below runs when you knit the markdown.

```
viz_nnet_df <- viz_grid %>% bind_cols(pred_viz_nnet_probs)

viz_nnet_df %>% glimpse()
```

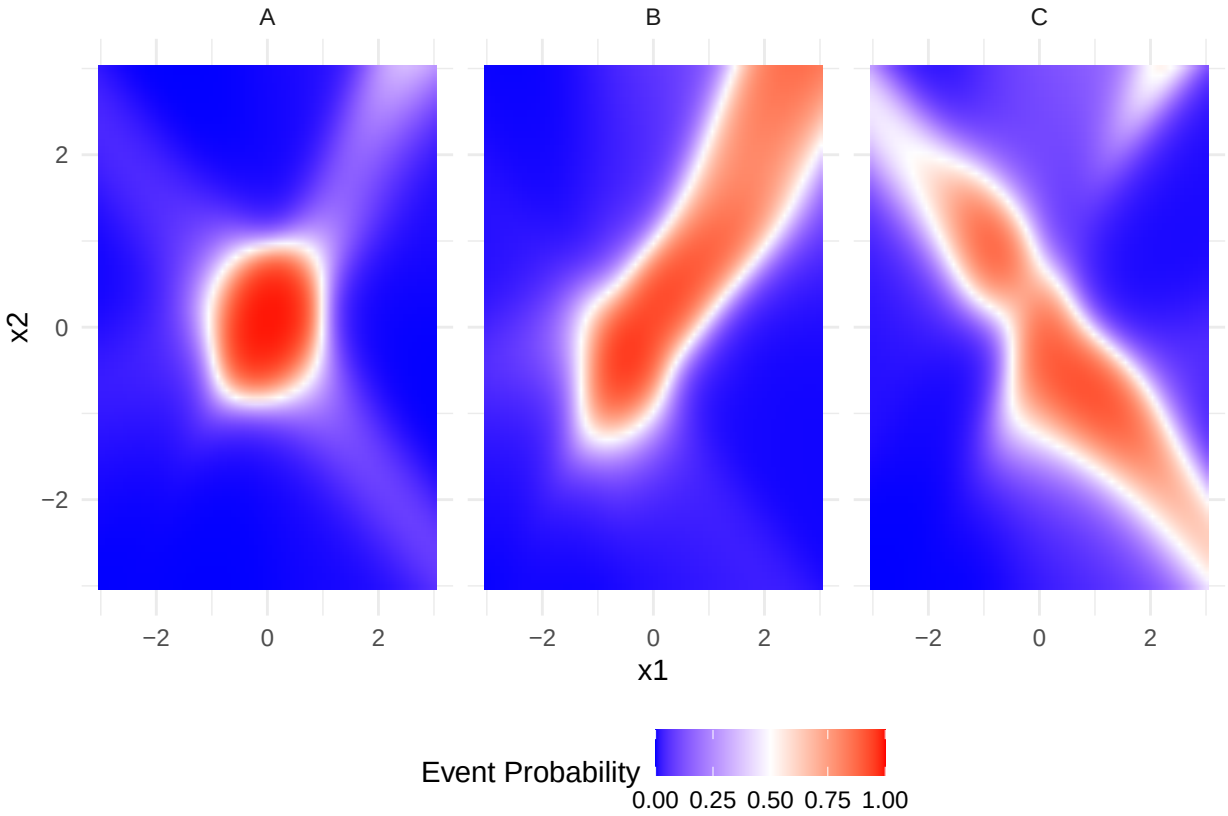
```
## Rows: 16,875
## Columns: 5
## $ x1        <dbl> -3.000000, -2.918919, -2.837838, -2.756757, -2.675676, -2.59~
## $ x2        <dbl> -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, ~
## $ x3        <chr> "A", "A", "A", "A", "A", "A", "A", "A", "A", "A", "A", "A", ~
## $ event     <dbl> 0.0004638493, 0.0004725322, 0.0004973136, 0.0005373341, 0.00~
## $ non_event <dbl> 0.9995362, 0.9995275, 0.9995027, 0.9994627, 0.9994097, 0.999~
```

The glimpse reveals that the `event` column stores the **predicted event probability**. You must visualize the predicted probability as a faceted raster plot just like you did in 3g). You must use the diverging palette to show the 50% decision boundary.

Visualize the predicted probability from the tuned neural network model with respect to `x1` and `x2` using `geom_raster()` faceted by `x3`. You must use the same diverging palette as 3g).

HINT: The event probability was named `mu` when you made the previous figure in 3g). The event probability is now named `event`!

```
ggplot(viz_nnet_df, aes(x = x1, y = x2, fill = event)) +
  geom_raster(interpolate = TRUE) +
  scale_fill_gradient2(low = 'blue', mid = 'white', high = 'red', midpoint = 0.5, limits = c(0, 1)) +
  facet_wrap(~x3) +
  labs(fill = "Event Probability") +
  theme_minimal() +
  theme(legend.position = "bottom")
```



SOLUTION

5f)

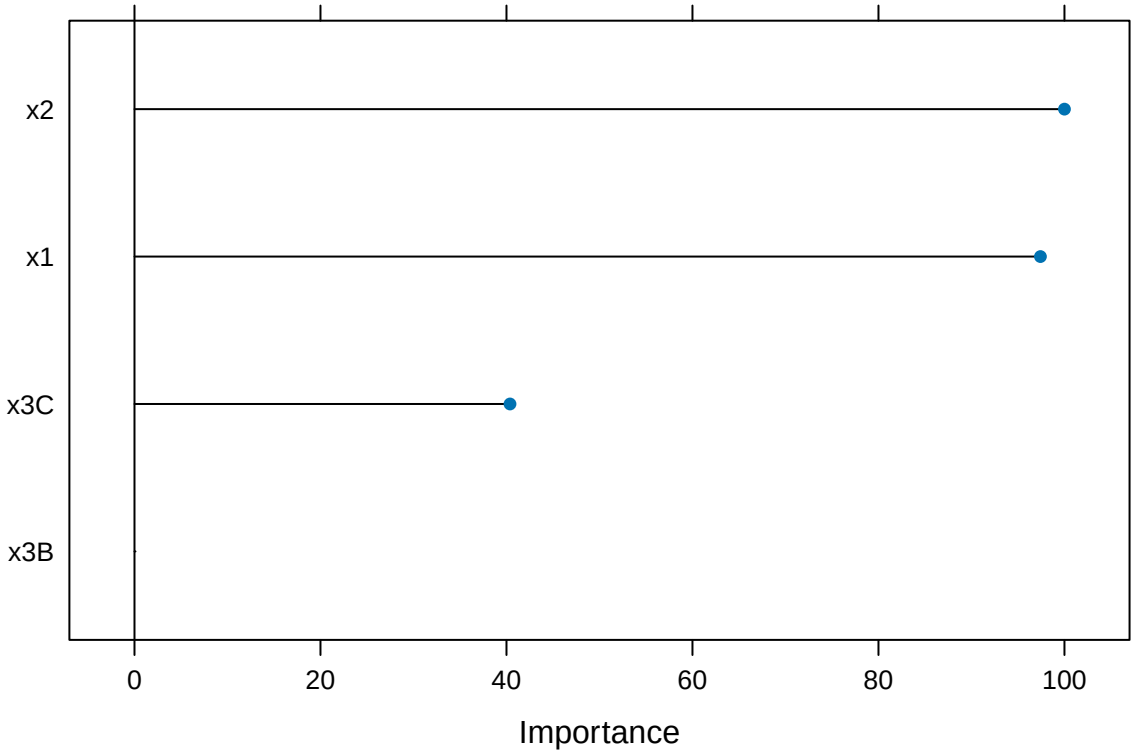
The `caret` variable importance function can be applied to neural networks. It attempts to identify the most important inputs as viewed by the neural network.

Create a plot to show the variable importance rankings associated with the tuned neural network model. Are the importance rankings consistent with the rankings from the elastic net model?

HINT: You visualized variable importance rankings from `caret` trained models in HW09.

```
nn_var_imp <- varImp(nnet_tune, scale=TRUE)

plot(nn_var_imp)
```



SOLUTION

5g)

How does the neural network relate to the tuned elastic net model? Are the event probability predictions consistent? Are the importance inputs consistent?

SOLUTION

1) Since the top-ranked variables in the neural network importance plot in 5f) are also top-ranked in the elastic net importance plot in 4h), there is a consistency in which inputs are considered most predictive.
2) The scale of importance may differ due to different modeling techniques, but the relative order of variables provides insight into the fact that the models are in agreement.
3) For the elastic net model in 4h), the non-zero coefficients correspond to the variables that the model has selected as important. The neural network does not use a coefficient-based approach, but the variable importance plot in 5f) still gives indication of which inputs the network is relying on most. Similarities are observed between the two variable importance plots.

Problem 06

Let's now fit advanced tree based models. You will start with a random forest model. You will use the default tuning grid and thus do not need to specify `tuneGrid`. Tree based models do not have the same kind of preprocessing requirements as other models. Thus, you do not need the `preProcess` argument in the `caret::train()` function call. We will discuss why that is in lecture.

6a)

Train a random forest binary classifier by setting the `method` argument equal to `"rf"`. You must set `importance = TRUE` in the `caret::train()` function call. You may use the . "shortcut" operator to define the formula. Assign the result to the `rf_default` variable. Display the `rf_default` object to the screen.

IMPORTANT: `caret` will prompt you in the R Console to install the `randomForest` package if you do not have it. Follow the instructions.

SOLUTION The random seed is set for you for reproducibility. You may add more code chunks if you like.

PLEASE NOTE: This code chunk may take several minutes to complete!

```
set.seed(1234)

rf_default <- caret::train(outcome ~ ., data = df_caret, method = "rf", metric = my_metric, trControl =

print(rf_default)
```

```
## Random Forest
##
## 300 samples
##   3 predictor
##   2 classes: 'event', 'non_event'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 270, 270, 270, 269, 270, 270, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##   2     0.7991966  0.5955484
##   3     0.7783055  0.5517902
##   4     0.7715622  0.5382123
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 2.
```

6b)

Let's examine the random forest behavior through predictions.

Complete the code chunk below. You must make predictions on the visualization grid, `viz_grid`, using the random forest model `rf_default`\`\`. Instruct the`predict()`function to return the probabilities by setting`type = 'prob'`.

```
pred_viz_rf_probs <- predict(rf_default, newdata = viz_grid, type = 'prob')
```

SOLUTION

6c)

The code chunk below is completed for you. The `pred_viz_rf_probs` dataframe is column binded to the `viz_grid` dataframe. The new object, `viz_rf_df`, provides the class predicted probabilities for each input combination in the visualization grid according to the random forest model. A glimpse is printed to the screen. Please not the `eval` flag is set to `eval=FALSE` in the code chunk below. You must change `eval` to `eval=TRUE` in the chunk options to make sure the code chunk below runs when you knit the markdown.

```
viz_rf_df <- viz_grid %>% bind_cols(pred_viz_rf_probs)

viz_rf_df %>% glimpse()
```

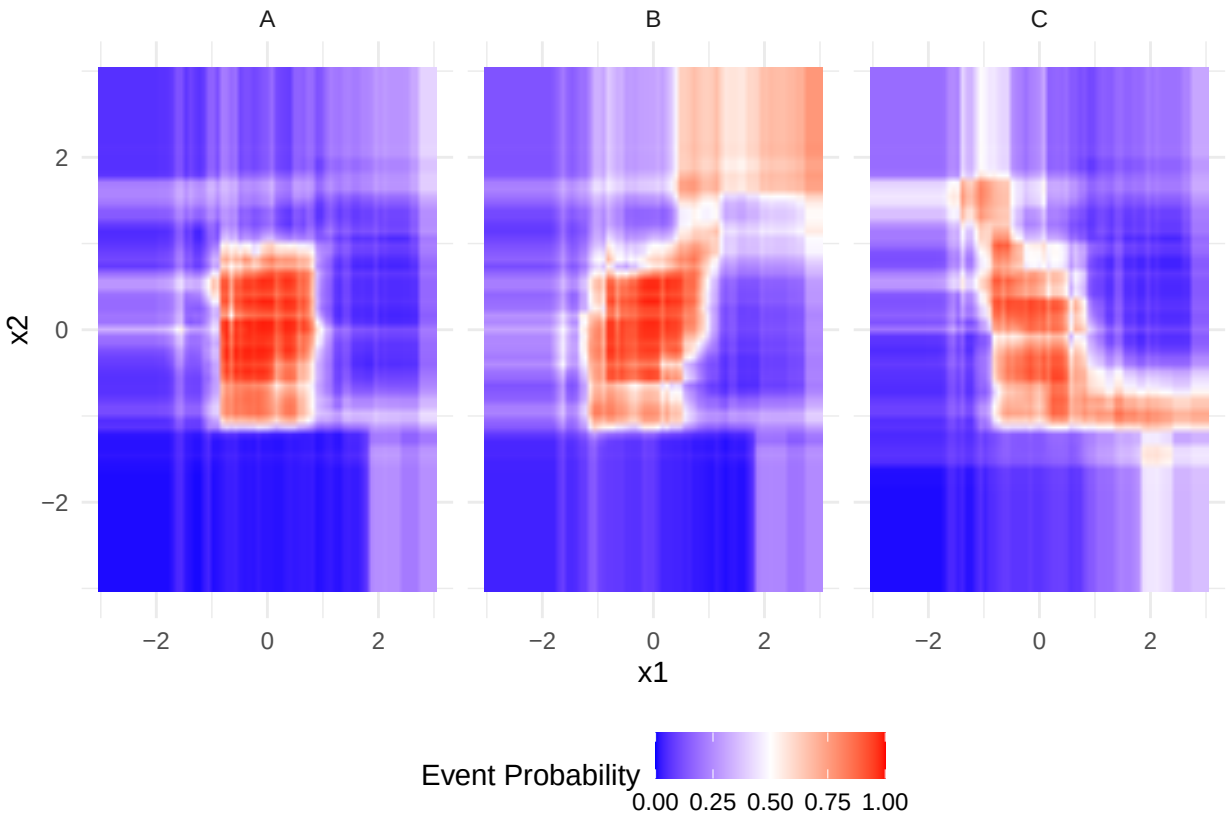
```
## Rows: 16,875
## Columns: 5
## $ x1         <dbl> -3.000000, -2.918919, -2.837838, -2.756757, -2.675676, -2.59~
## $ x2         <dbl> -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, ~
## $ x3         <chr> "A", "A", "A", "A", "A", "A", "A", "A", "A", "A", "A", "A", ~
## $ event      <dbl> 0.008, 0.008, 0.008, 0.008, 0.008, 0.008, 0.008, 0.008, 0.00~
## $ non_event <dbl> 0.992, 0.992, 0.992, 0.992, 0.992, 0.992, 0.992, 0.992, 0.99~
```

The glimpse reveals that the `event` column stores the **predicted event probability**. You must visualize the predicted probability as a faceted raster plot just like you did in 3g). You must use the diverging palette to show the 50% decision boundary.

Visualize the predicted probability from the random forest model with respect to `x1` and `x2` using `geom_raster()` faceted by `x3`. You must use the same diverging palette as 3g).

HINT: The event probability was named `mu` when you made the previous figure in 3g). The event probability is now named `event`!

```
ggplot(viz_rf_df, aes(x = x1, y = x2, fill = event)) +
  geom_raster(interpolate = TRUE) +
  scale_fill_gradient2(low = 'blue', mid = 'white', high = 'red', midpoint = 0.5, limits = c(0, 1)) +
  facet_wrap(~x3) +
  labs(fill = "Event Probability") +
  theme_minimal() +
  theme(legend.position = "bottom")
```



SOLUTION

6d)

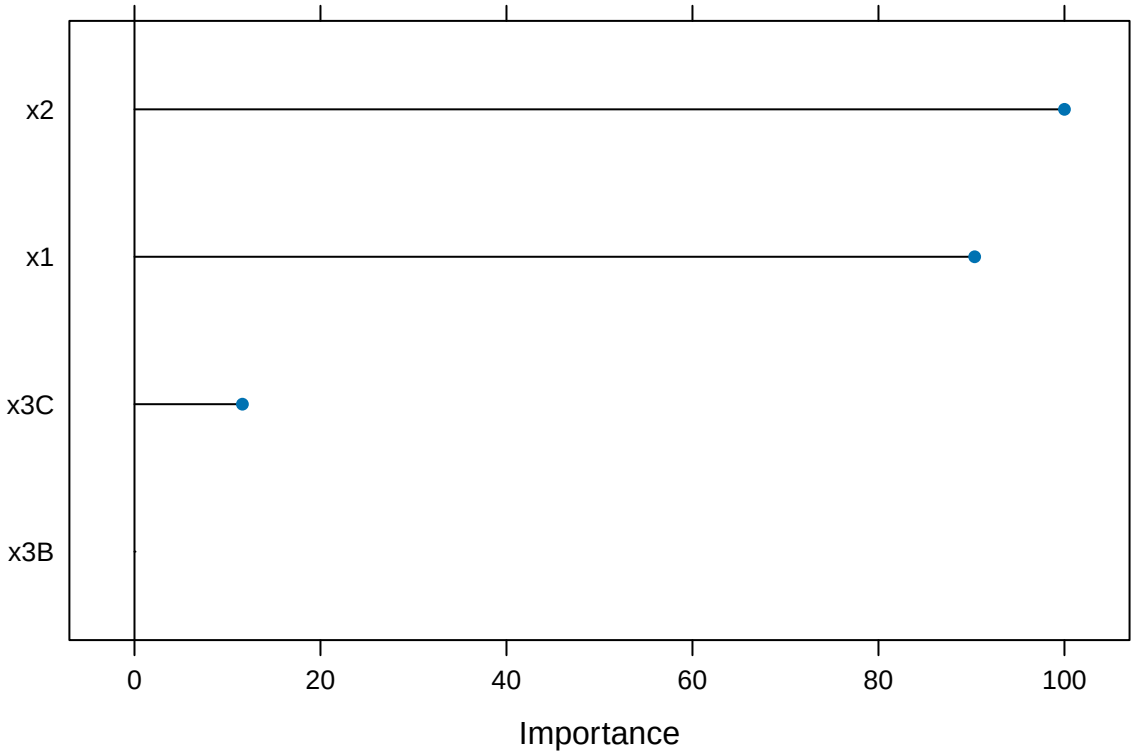
You should have included `importance = TRUE` in the `caret::train()` call in 6a). This allows the random forest specific variable importance rankings to be returned.

Create a plot to show the variable importance rankings associated with the random forest model. Are the importance rankings consistent with the rankings from the elastic net model?

HINT: You visualized variable importance rankings from `caret` trained models in HW09.

```
rf_var_imp <- varImp(rf_default, scale = TRUE)

plot(rf_var_imp)
```



SOLUTION

Yes, because of the similar facts expressed in Problem 5g), we can conclude that the importance rankings are consistent with the rankings from the elastic net model.

Problem 07

Let's wrap up this modeling exercise by training a gradient boosted tree via XGBoost.

7a)

You will begin by using the default tuning grid from `caret`.

Train a gradient boosted tree binary classifier via XGBoost by setting the `method` argument equal to `"xgbTree"`. You should NOT include `importance = TRUE` in the `caret::train()` function call. You may use the `.` "shortcut" operator to define the formula. Assign the result to the `xgb_default` variable.

Do not display `xgb_default` to the screen. Instead, use the default plot method to plot the performance. You do not need to set any additional arguments to the default plot method.

Display the best tuning parameters for the gradient boosted tree to the screen.

SOLUTION The random seed is set for you for reproducibility. You may add more code chunks if you like.

PLEASE NOTE: This code chunk may take several minutes to complete!

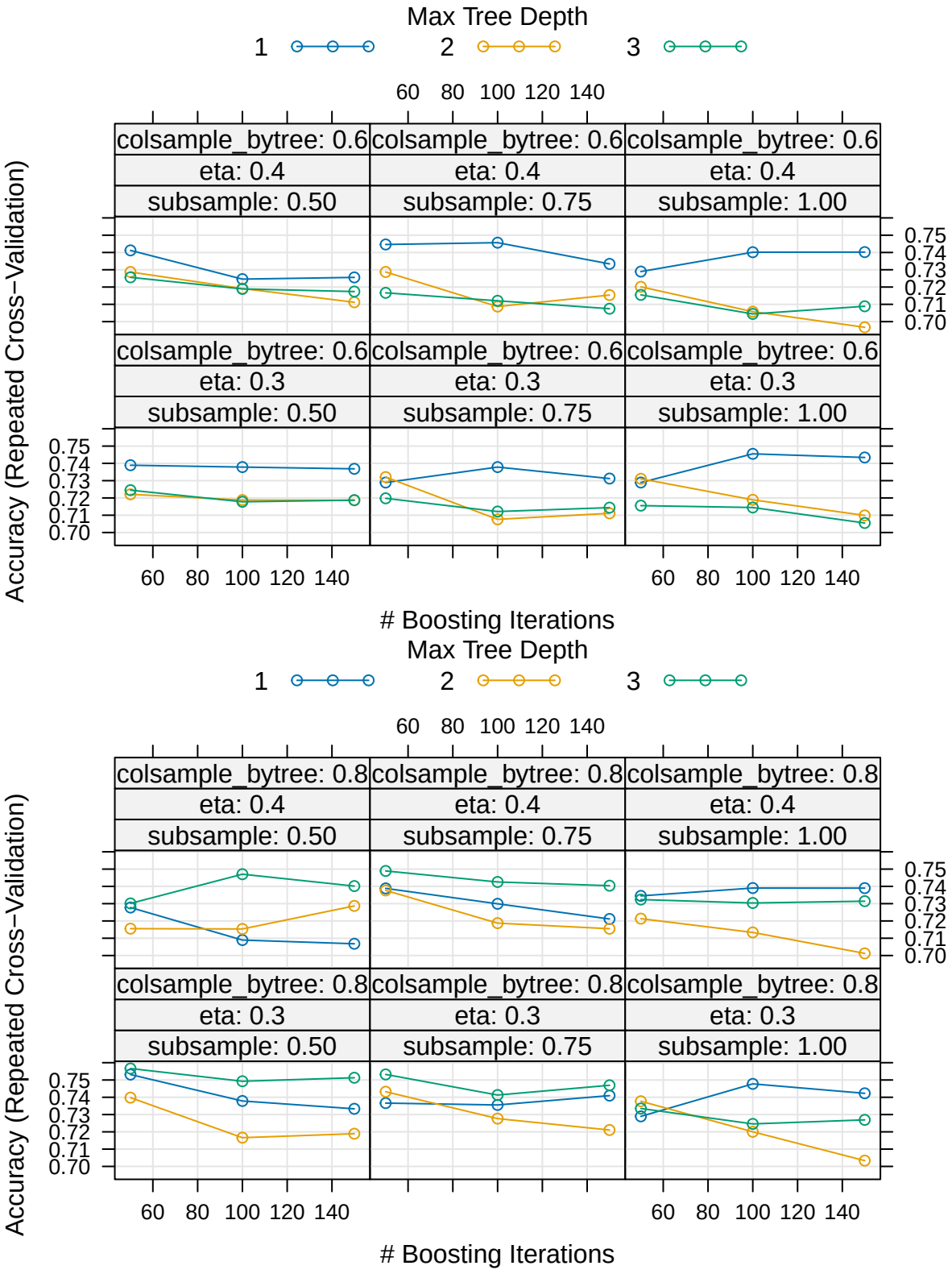
IMPORTANT: include the argument `verbosity = 0` in the `caret::train()` function call. This will make sure the `xgBoost` package does **NOT** print a lot of warning messages to the screen. The warning messages result because the XGBoost package has included syntax changes in the most recent versions. The `caret` package uses older syntax which thus causes a large number of warning messages will be displayed to the screen.

IMPORTANT: include the argument `nthread = 1` in the `caret::train()` function call. By default, XGBoost wants to use the MAXIMUM number of threads available on your machine. I do not know why this is the default, because this will cause your machine to become overloaded. XGBoost will therefore appear to take a very long time to complete. Instead, by setting `nthread = 1`, you are forcing XGBoost to run a single thread. You are thus forcing XGBoost to NOT run in parallel. Although this sounds like it could be slower, you will NOT overload your machine and therefore XGBoost will complete faster! Running in parallel is NOT always faster than running sequentially, especially if you OVERLOAD the computer!

```
set.seed(1234)

xgb_default <- train(outcome ~ .,
                     data = df_caret,
                     method = "xgbTree",
                     metric = my_metric,
                     trControl = my_ctrl,
                     verbosity = 0,
                     nthread = 1)

plot(xgb_default)
```




```
print(xgb_default$bestTune)
```

```
##    nrounds max_depth eta gamma colsample_bytree min_child_weight subsample
## 46      50         3 0.3     0              0.8                1       0.5
```

7b)

Let's make predictions with the gradient boosted tree like we did with the other models.

Complete the code chunk below. You must make predictions on the visualization grid, `viz_grid`, using the random forest model `xgb_default`. Instruct the `predict()` function to return the probabilities by setting `type = 'prob'`.

```
pred_viz_xgb_default_probs <- predict(xgb_default, newdata = viz_grid, type = 'prob')
```

SOLUTION

7c)

The code chunk below is completed for you. The `pred_viz_xgb_probs` dataframe is column binded to the `viz_grid` dataframe. The new object, `viz_xgb_df`, provides the class predicted probabilities for each input combination in the visualization grid according to the random forest model. A glimpse is printed to the screen. Please not the `eval` flag is set to `eval=FALSE` in the code chunk below. You must change `eval` to `eval=TRUE` in the chunk options to make sure the code chunk below runs when you knit the markdown.

```
viz_xgb_df <- viz_grid %>% bind_cols(pred_viz_xgb_default_probs)

viz_xgb_df %>% glimpse()
```

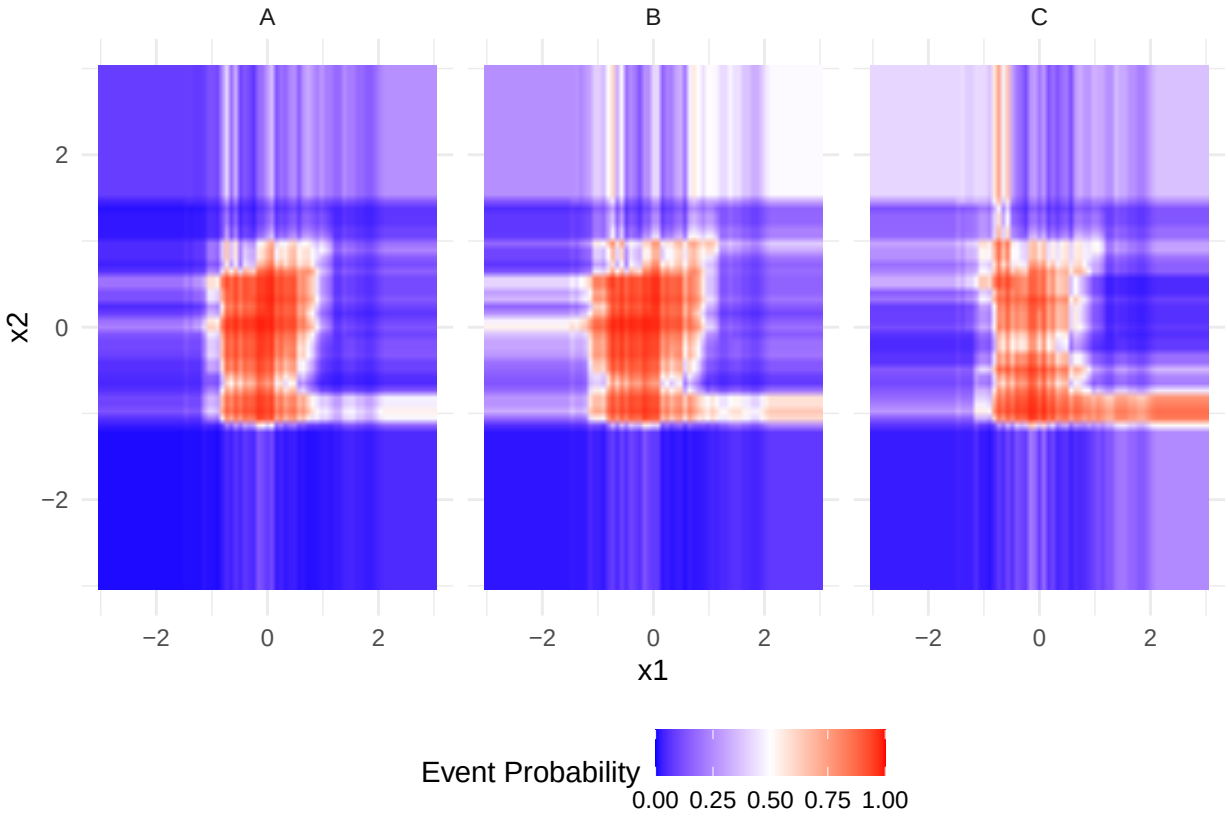
```
## Rows: 16,875
## Columns: 5
## $ x1        <dbl> -3.000000, -2.918919, -2.837838, -2.756757, -2.675676, -2.59~
## $ x2        <dbl> -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, ~
## $ x3        <chr> "A", "A", "A", "A", "A", "A", "A", "A", "A", "A", "A", "A", ~
## $ event     <dbl> 0.008535348, 0.008535348, 0.008535348, 0.008535348, 0.008535~
## $ non_event <dbl> 0.9914647, 0.9914647, 0.9914647, 0.9914647, 0.9914647, 0.991~
```

The glimpse reveals that the `event` column stores the **predicted event probability**. You must visualize the predicted probability as a faceted raster plot just like you did in 3g). You must use the diverging palette to show the 50% decision boundary.

Visualize the predicted probability from the XGBoost model with respect to `x1` and `x2` using `geom_raster()` faceted by `x3`. You must use the same diverging palette as 3g).

HINT: The event probability was named `mu` when you made the previous figure in 3g). The event probability is now named `event`!

```
ggplot(viz_xgb_df, aes(x = x1, y = x2, fill = event)) +
  geom_raster(interpolate = TRUE) +
  scale_fill_gradient2(low = 'blue', mid = 'white', high = 'red', midpoint = 0.5, limits = c(0, 1)) +
  facet_wrap(~x3) +
  labs(fill = "Event Probability") +
  theme_minimal() +
  theme(legend.position = "bottom")
```



SOLUTION

7d)

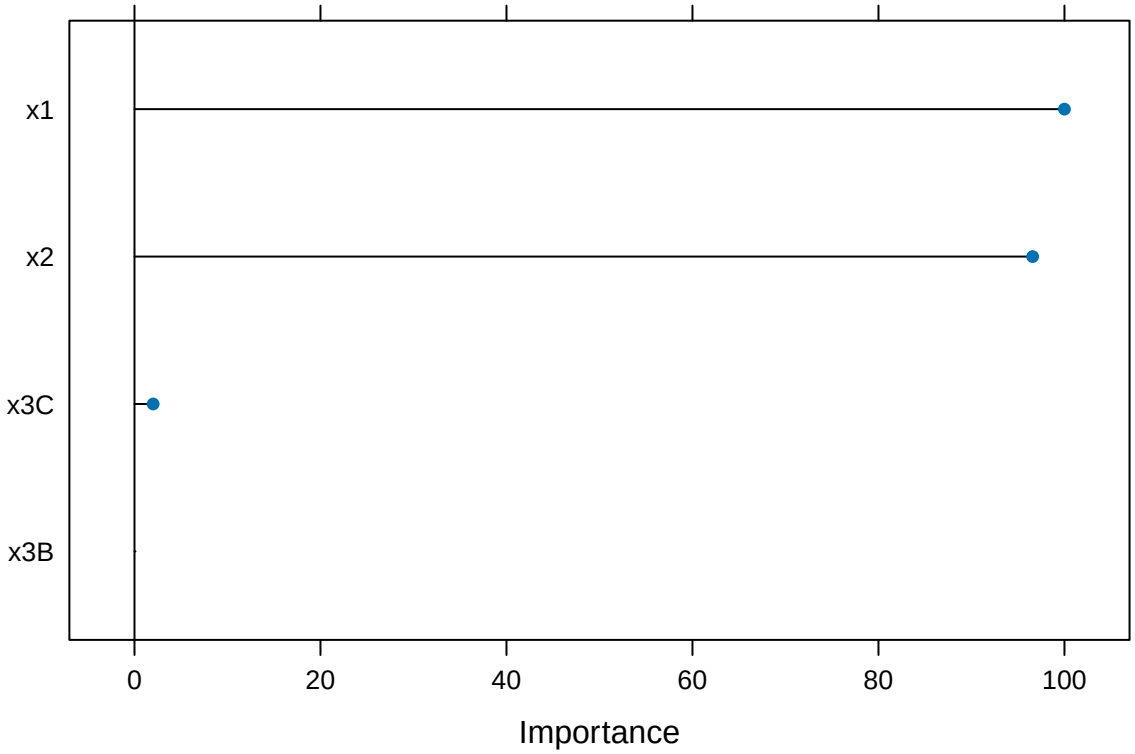
Gradient boosted trees also provide variable importance rankings.

Create a plot to show the variable importance rankings associated with the XGBoost model. Are the importance rankings consistent with the rankings from the elastic net model?

HINT: You visualized variable importance rankings from `caret` trained models in HW09.

```
xgb_var_imp <- varImp(xgb_default, scale = TRUE)

plot(xgb_var_imp)
```



SOLUTION

Yes, because of the similar facts expressed in Problem 5g), we can conclude that the importance rankings are consistent with the rankings from the elastic net model.

7e)

You used the default XGBoost tuning grid. Let's see if we can improve the model further by using a refined tuning grid! This grid will focus on the most critical tuning parameters of the XGBoost model: the number of trees, `nrounds`, the interaction depth, `max_depth`, and the learning rate, `eta`. XGBoost has even more tuning parameters, but you will keep those fixed at values identified by the default tuning grid.

The code chunk below is started for you. It defines a tuning grid via `expand.grid()` for all tuning parameters associated with XGBoost. You must complete the code chunk in order to define a tuning grid that focuses on the three most important parameters.

Complete the code chunk below. The number of trees, `nrounds`, should be a sequence from 25 to 50 in intervals of 25. The interaction depth, `max_depth`, should be a vector with values 3, 6, 9, and 129. The learning rate, `eta`, should be a vector with values equal to 0.25, 0.5, and 1 times the default grid's best `eta` value. All other tuning parameters should be set to constants equal to the identified best tuning parameter values from the default grid.

```
xgb_best_gamma <- xgb_default$bestTune$gamma
xgb_best_colsample_bytree <- xgb_default$bestTune$colsample_bytree
xgb_best_min_child_weight <- xgb_default$bestTune$min_child_weight
xgb_best_subsample <- xgb_default$bestTune$subsample
xgb_best_eta <- xgb_default$bestTune$eta

xgb_grid <- expand.grid(
  nrounds = seq(25, 50, by = 25),
  max_depth = c(3, 6, 9, 12),
  eta = xgb_best_eta * c(0.25, 0.5, 1),
```

```
  gamma = xgb_best_gamma,
  colsample_bytree = xgb_best_colsample_bytree,
  min_child_weight = xgb_best_min_child_weight,
  subsample = xgb_best_subsample
)
```

SOLUTION

7f)

Train, assess, and tune the XGBoost model with the custom tuning grid and the defined resampling scheme. Assign the result to the `xgb_tune` object. You should specify the arguments to `caret::train()` consistent with your solution in Problem 7a), except you should also assign `xgb_grid` to the `tuneGrid` argument.

Do not print the result to the screen. Instead use the default plot method to visualize the resampling results. Use the `$bestTune` field to print the identified best tuning parameter values to the screen. How many iterations or trees are associated with the best tuned model? What is the interaction depth associated with the best tuned model?

SOLUTIOn The random seed is set for you for reproducibility. You may add more code chunks if you like.

PLEASE NOTE: This code chunk may take several minutes to complete!

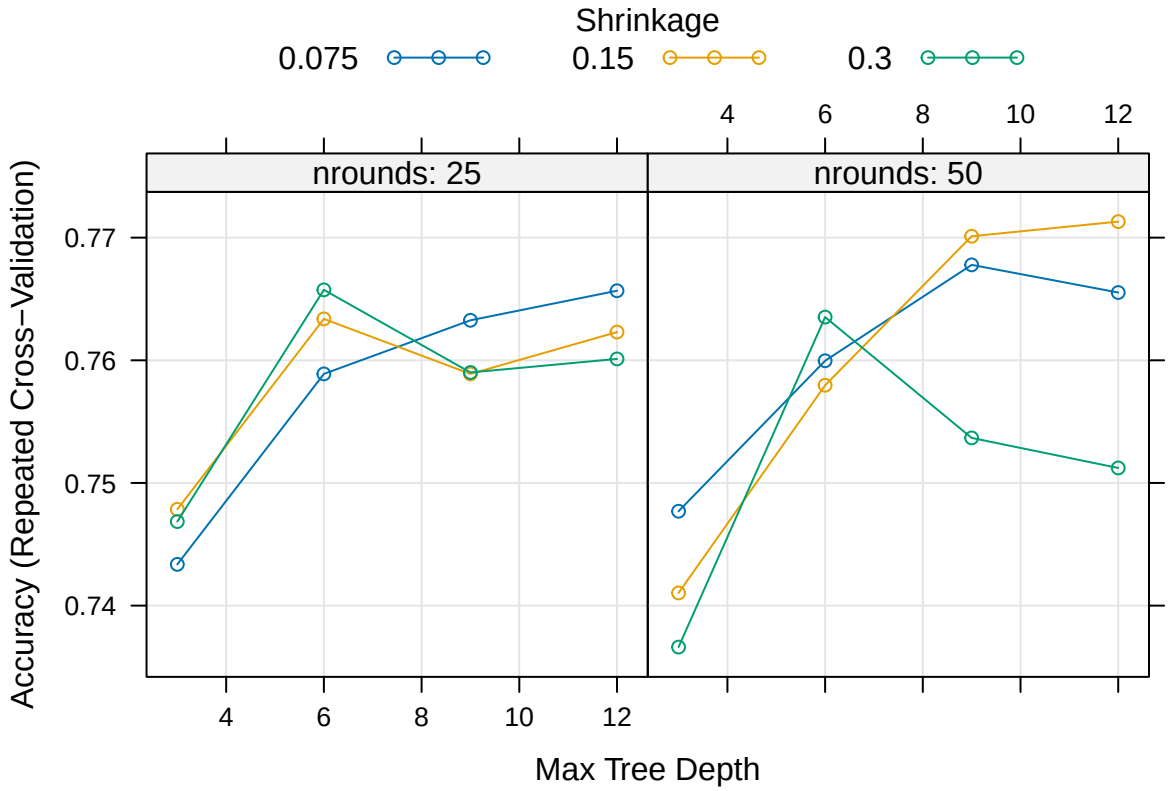
IMPORTANT: include the argument `verbosity = 0` in the `caret::train()` function call. This will make sure the `xgBoost` package does **NOT** print a lot of warning messages to the screen. The warning messages result because the XGBoost package has included syntax changes in the most recent versions. The `caret` package uses older syntax which thus causes a large number of warning messages will be displayed to the screen.

IMPORTANT: include the argument `nthread = 1` in the `caret::train()` function call. By default, XGBoost wants to use the MAXIMUM number of threads available on your machine. I do not know why this is the default, because this will cause your machine to become overloaded. XGBoost will therefore appear to take a very long time to complete. Instead, by setting `nthread = 1`, you are forcing XGBoost to run a single thread. You are thus forcing XGBoost to NOT run in parallel. Although this sounds like it could be slower, you will NOT overload your machine and therefore XGBoost will complete faster! Running in parallel is NOT always faster than running sequentially, especially if you OVERLOAD the computer!

```
set.seed(1234)

xgb_tune <- train(outcome ~ .,
                  data = df_caret,
                  method = "xgbTree",
                  tuneGrid = xgb_grid,
                  metric = my_metric,
                  trControl = my_ctrl,
                  verbosity = 0,
                  nthread = 1)

plot(xgb_tune)
```




```
print(xgb_tune$bestTune)
```

```
##    nrounds max_depth  eta gamma colsample_bytree min_child_weight subsample
## 16      50        12 0.15     0              0.8                1       0.5
```

50 iterations or trees (nrounds: 50) are associated with the best tuned model. An interaction depth of 12 (max_depth: 12) is associated with the best tuned model.

Problem 08

Lastly, let's compare the various `caret` trained models based on our resampling scheme.

8a)

Complete the first code chunk below which compiles the tuned elastic net, default neural network, tuned neural network, default random forest, and default XGBoost models together.

The field names in the list state which model should be assigned.

```
caret_acc_compare <- resamples(list(ENET_tune = enet_tune,
                                    NNET_default = nnet_default,
                                    NNET_tune = nnet_tune,
                                    RF_default = rf_default,
                                    XGB_default = xgb_default,
                                    XGB_tune = xgb_tune))
```

SOLUTION

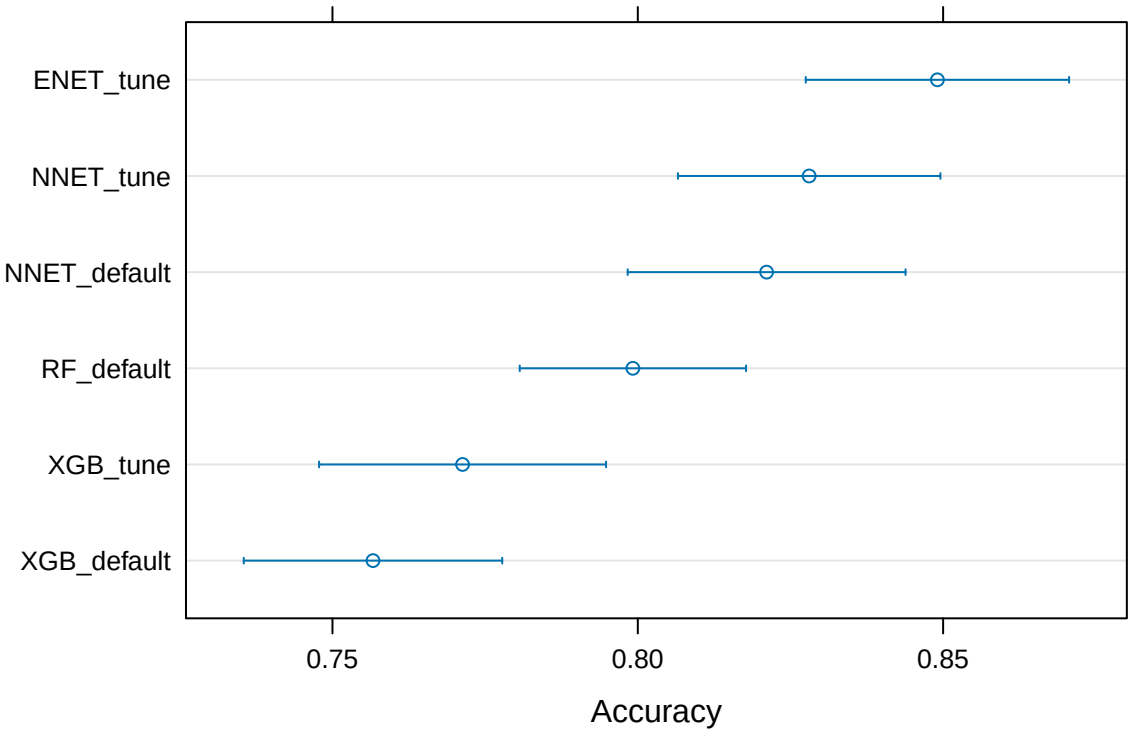
8b)

Visually compare the models based on the resampled Accuracy with a dotplot.

Use the `dotplot()` function to visualize the resampled performance summary for each model. Assign the `metric` argument to `'Accuracy'` to force the `dotplot()` function to only show Accuracy.

Which model is the best for this application?

```
dotplot(caret_acc_compare, metric = 'Accuracy')
```



SOLUTION

Based on the plot above, the tuned Elastic Net model (ENET_tune) shows the highest mean accuracy compared to the other models. So it performed best in terms of accuracy.

8c)

How would you describe the differences between the 4 types of models you trained in this application?

SOLUTION

1) ENET_tune has the highest accuracy and appears to be the most consistent across different folds of the data. It is a linear model enhanced with L1 and L2 regularization, which helps prevent overfitting.
2) NNET_tune shows improved accuracy over the default neural network (NNET_default), which highlights the importance of hyperparameter tuning. Neural networks are capable of modeling complex, non-linear relationships in the data but can require careful tuning.
3) The random forest model shows competitive accuracy and with a narrower confidence interval than the elastic net, suggesting that its performance varies less across different resamples. Random forests are

ensemble models that can capture complex structures in the data.

4) The default XGBoost model (XGB_default) shows good performance, but not quite as high as the tuned Elastic Net model. XGBoost is another ensemble technique known for its high performance on a wide range of problems. The tuned XGBoost (XGB_tune) improve upon the default as expected.