Overview

This homework is focused on binary classification. You will explore a binary classification application and fit non-Bayesian logistic regression models by maximizing the likelihood via the `glm()` function R. Then you will fit Bayesian logistic regression models with the Laplace Approximation by programming the log-posterior function. You will identify the best model and make predictions to study the trends in the predicted event probability. You will conclude the application by training and tuning various non-Bayesian models via resampling. First, you will tune a non-Bayesian logistic regression model with the elastic net penalty to help identify the most important features in the model. You will visualize the predicted event probability trends from the tuned elastic net model and compare the non-Bayesian regularized predictions with your Bayesian model predictions. Then you tune several advanced methods like neural networks and random forests. This last portion of the assignment introduces the basic syntax for training and tuning those advanced methods with `caret`. You will focus on assessing their performance via cross-validation and visualizing their predictive trends in order to compare their behavior to the simpler generalized linear models you previously fit! We will discuss how neural networks and random forests work later in lecture.

You will use the `tidyverse`, `coefplot`, `broom`, `MASS`, `glmnet`, `nnet`, `randomForest`, and `caret` packages in this assignment. The `caret` package will prompt you to install `nnet` and `randomForest` if you do not have them 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.

```r
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 error
```

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 and the `glmnet`, `nnet`, and `randomForest` packages will be loaded via `caret`.

Problem 01

The primary data set you will work with in this assignment is loaded for you in the code chunk below.

```
df1 <- readr::read_csv('hw10_binary_01.csv', col_names = TRUE)
```

```
## Rows: 225 Columns: 4
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): x1
## dbl (3): x2, x3, 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 data consists of 3 inputs, `x1`, `x2`, and `x3`, and one binary outcome, `y`. The glimpse below shows that `x1` is a character and thus is a categorical input. The `x2` and `x3` inputs are continuous. The binary outcome has been encoded as `y=1` for the EVENT and `y=0` for the NON-EVENT.

```
df1 %>% glimpse()
```

```
## Rows: 225
## Columns: 4
## $ x1 <chr> "C", "B", "C", "B", "A", "C", "A", "A", "C", "C", "B", "C", "A", "C~
## $ x2 <dbl> 1.682873632, -1.033648456, 0.110854156, 2.032934019, -0.225540507, ~
## $ x3 <dbl> -0.353085685, -0.778102544, 0.757536960, 0.639465847, 0.017483448, ~
## $ y  <dbl> 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0~
```

It is always best to explore data before training predictive models. This assignment does not require you to create all figures necessary to sufficiently explore the data. This assignment focuses on various ways of exploring the relationships between the binary outcome and the inputs. You will thus not consider input correlation plots in this assignment. Please note that the inputs have already been standardized for you to streamline the visualizations and modeling. Realistic applications like the final project may have inputs with vastly different scales and so you will need to execute the preprocessing as part of the model training.

The code chunk below reshapes the wide-format `df1` data into long-format, `lf1`. The continuous inputs, `x1` and `x2`, are "gathered" and "stacked" on top of each other. The long-format data supports using facets associated with the continuous inputs. You will use the long-format data in some of the visualizations below.

```
lf1 <- df1 %>%
  tibble::rowid_to_column() %>%
  pivot_longer(c(x2, x3))
```

The glimpse below shows the continuous input names are contained in the `name` column and their values are contained in the `value` column.

```
lf1 %>% glimpse()
```

```
## Rows: 450
## Columns: 5
## $ rowid <int> 1, 1, 2, 2, 3, 3, 4, 4, 5, 5, 6, 6, 7, 7, 8, 8, 9, 9, 10, 10, 11~
## $ x1    <chr> "C", "C", "B", "B", "C", "C", "B", "B", "A", "A", "C", "C", "A",~
## $ y     <dbl> 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0~
## $ name  <chr> "x2", "x3", "x2", "x3", "x2", "x3", "x2", "x3", "x2", "x3", "x2"~
## $ value <dbl> 1.682873632, -0.353085685, -1.033648456, -0.778102544, 0.1108541~
```

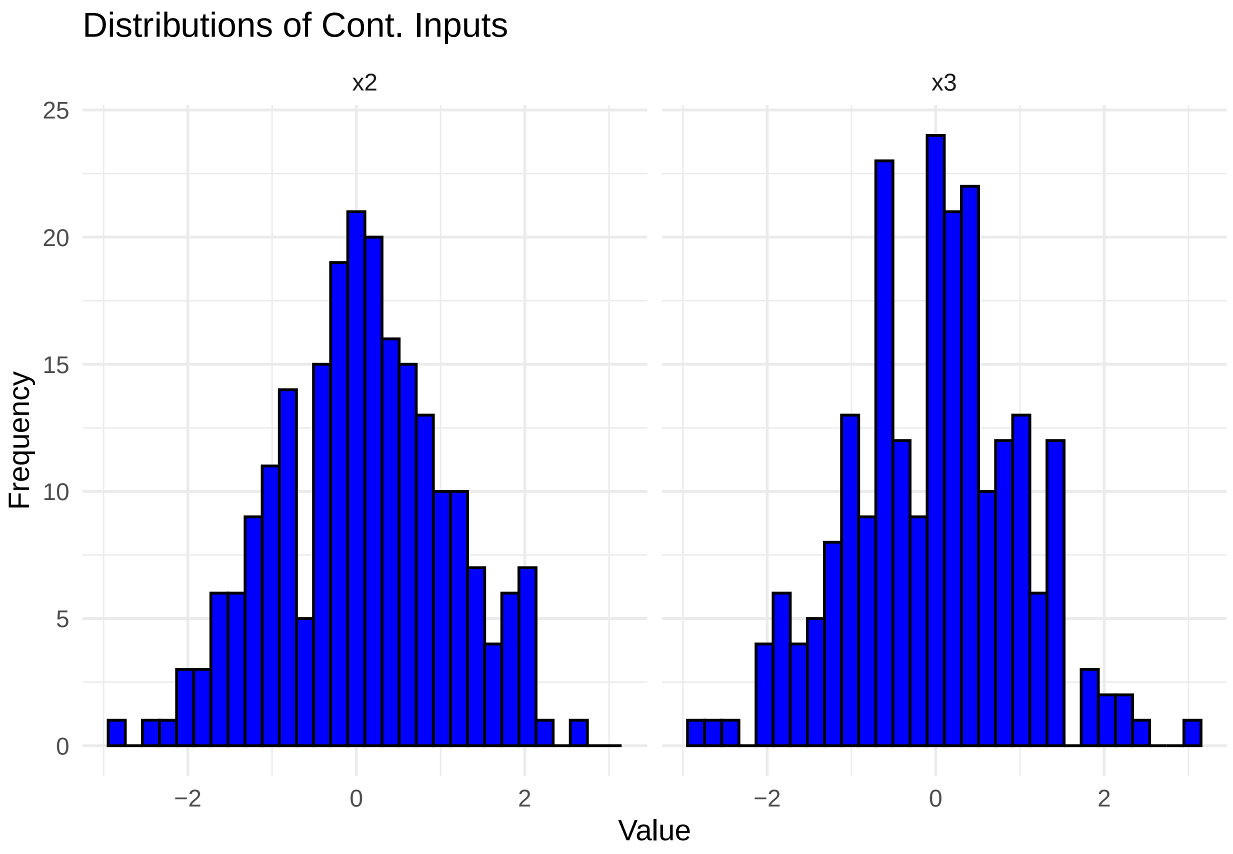
1a)

Let's start with exploring the inputs.

Visualize the distributions of the continuous inputs as histograms in `ggplot2`.

It is up to you as to whether you use the wide format or long-format data. If you use the wide-format data you should create two separate figures. If you use the long-format data you should use facets for each continuous input.

```
ggplot(lf1, aes(x = value)) +
  geom_histogram(bins = 30, fill = "blue", color = "black") +
  facet_wrap(~ name) +
  labs(x = "Value", y = "Frequency", title = "Distributions of Cont. Inputs") +
  theme_minimal()
```

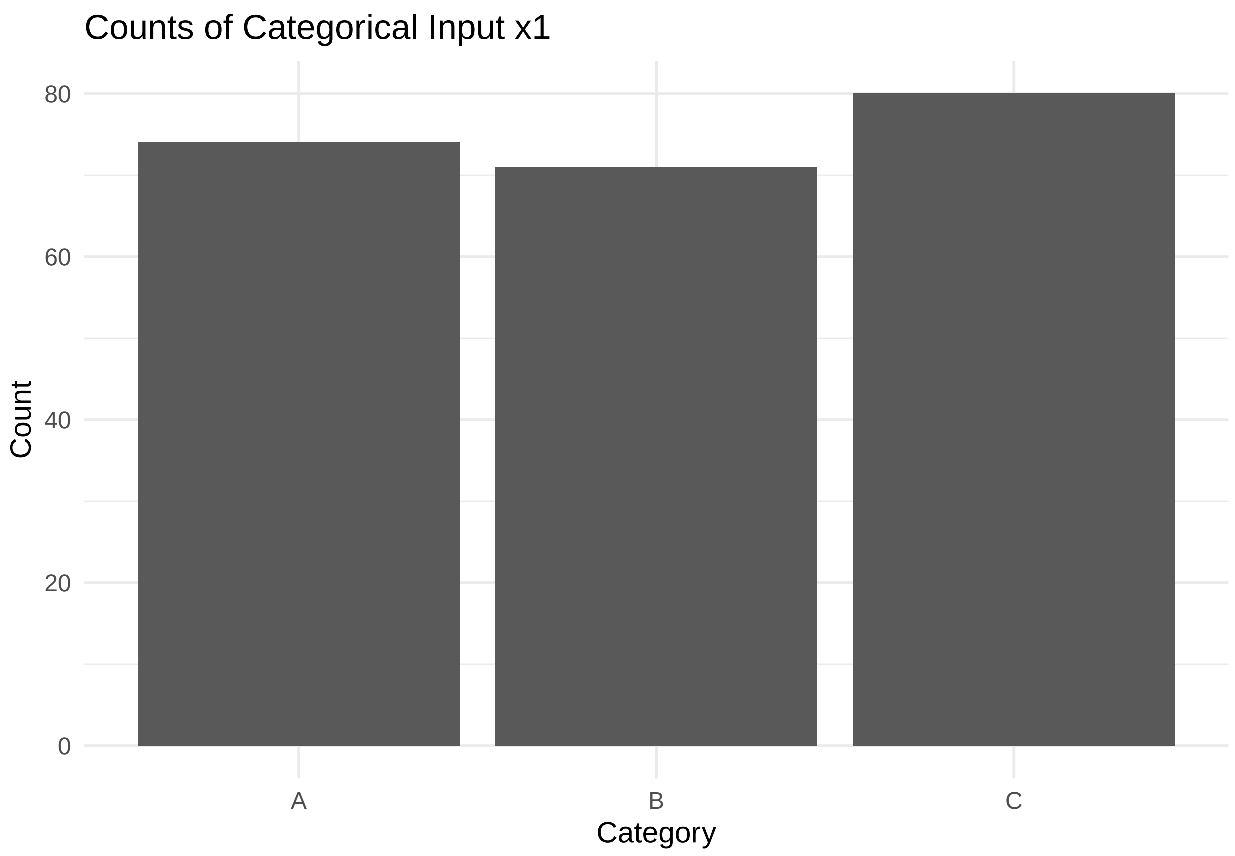


SOLUTION

1b)

Visualize the counts of the categorical input `x1` with a bar chart in `ggplot2`. You MUST use the wide-format data for this visualization.

```
ggplot(df1, aes(x = x1)) +
  geom_bar() +
  labs(x = "Category", y = "Count", title = "Counts of Categorical Input x1") +
  theme_minimal()
```



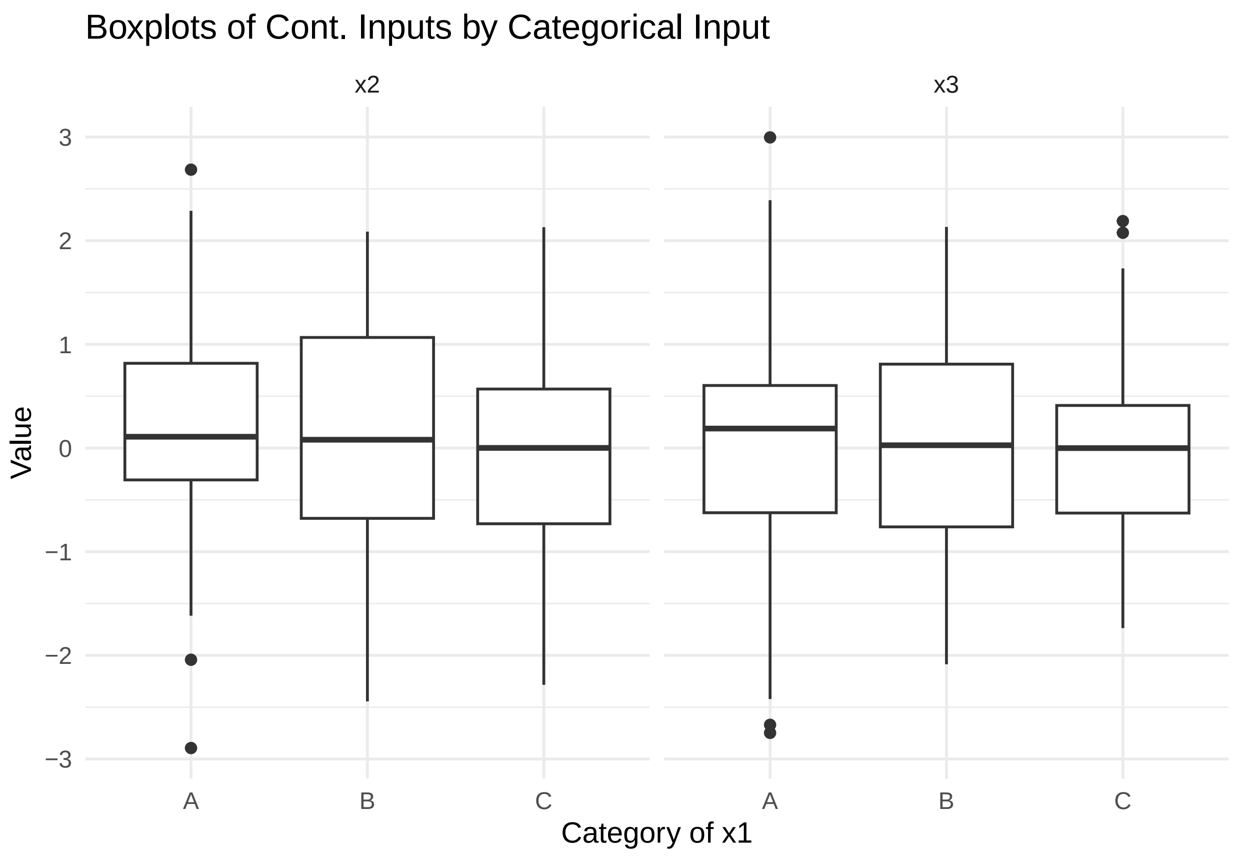
SOLUTION

1c)

Let's examine if there are differences in the continuous input distributions based on the categorical input. You will use boxplots to focus on the distribution summary statistics.

You must use the long-format data for this figure. Create a boxplot in `ggplot2` where the `x` aesthetic is the categorical input and the `y` aesthetic is the `value` column. You must include facets associated with the `name` column.

```r
ggplot(lf1, aes(x = x1, y = value)) +
  geom_boxplot() +
  facet_wrap(~ name) +
  labs(x = "Category of x1", y = "Value", title = "Boxplots of Cont. Inputs by Categorical Input") +
  theme_minimal()
```



SOLUTION

1d)

Let's now focus on the binary outcome.

Visualize the counts of the binary outcome `y` with a bar chart in `ggplot2`. You MUST use the wide-format data for this visualization.

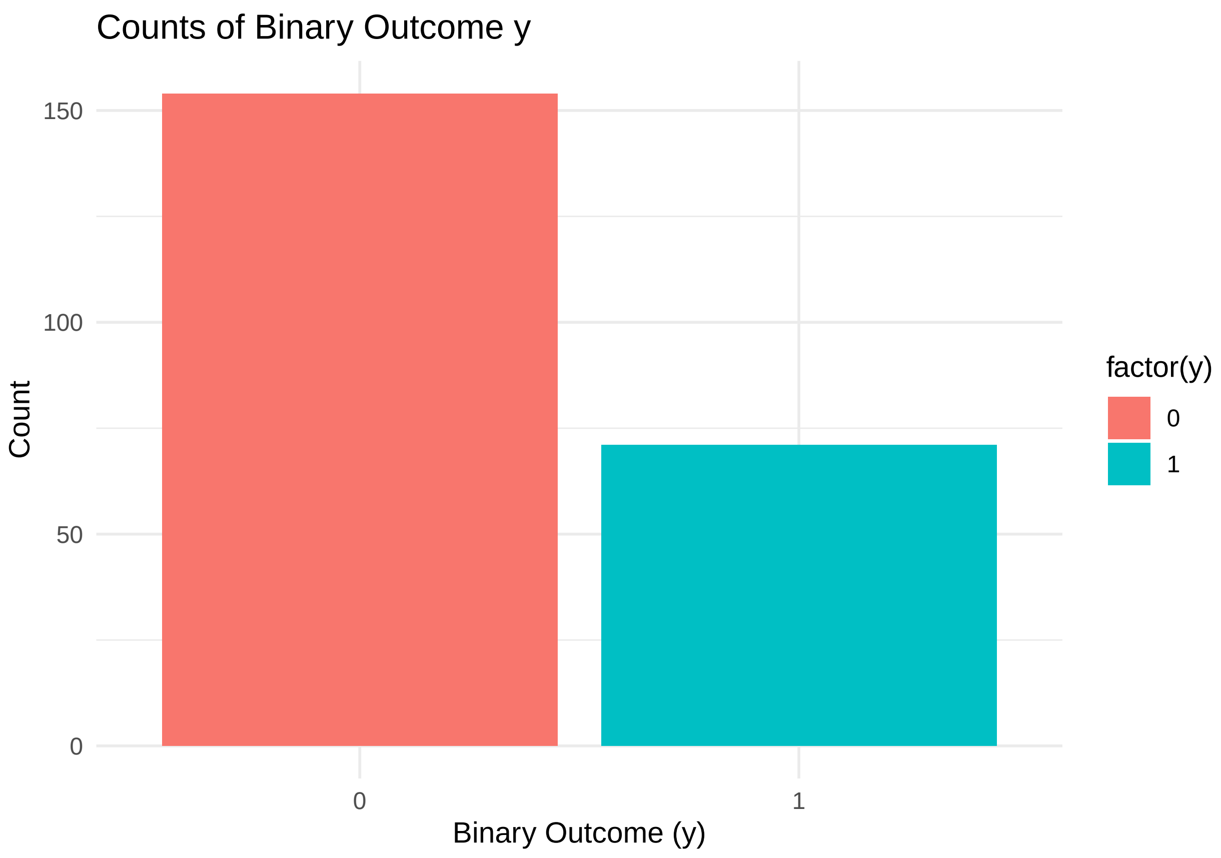
Is the binary outcome balanced?

SOLUTION Add your code chunks here.

What do you think?

The binary outcome does not seem to be balanced as it can be seen in the bar chart below.

```r
ggplot(df1, aes(x = factor(y), fill = factor(y))) +
  geom_bar() +
  labs(x = "Binary Outcome (y)", y = "Count", title = "Counts of Binary Outcome y") +
  theme_minimal()
```



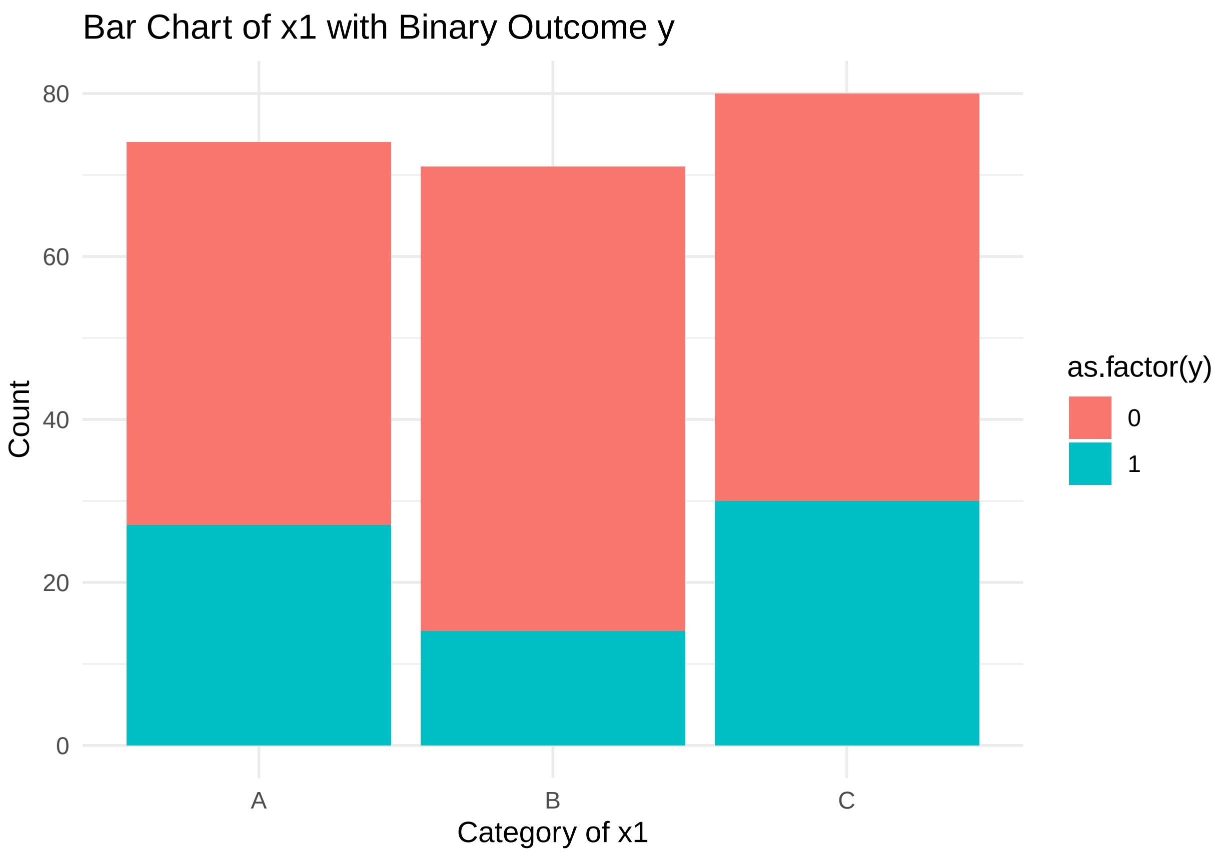
1e)

Let's see if the categorical input impacts the binary outcome.

Create a bar chart for the categorical input `x1` with `ggplot2` like you did in 1b). However, you must also map the `fill` aesthetic to `as.factor(y)`.

The data type conversion function `as.factor()` can be applied in-line. This will force a categorical fill palette to be used rather than a continuous palette.

```r
ggplot(df1, aes(x = x1, fill = as.factor(y))) +
  geom_bar() +
  labs(x = "Category of x1", y = "Count", title = "Bar Chart of x1 with Binary Outcome y") +
  theme_minimal()
```



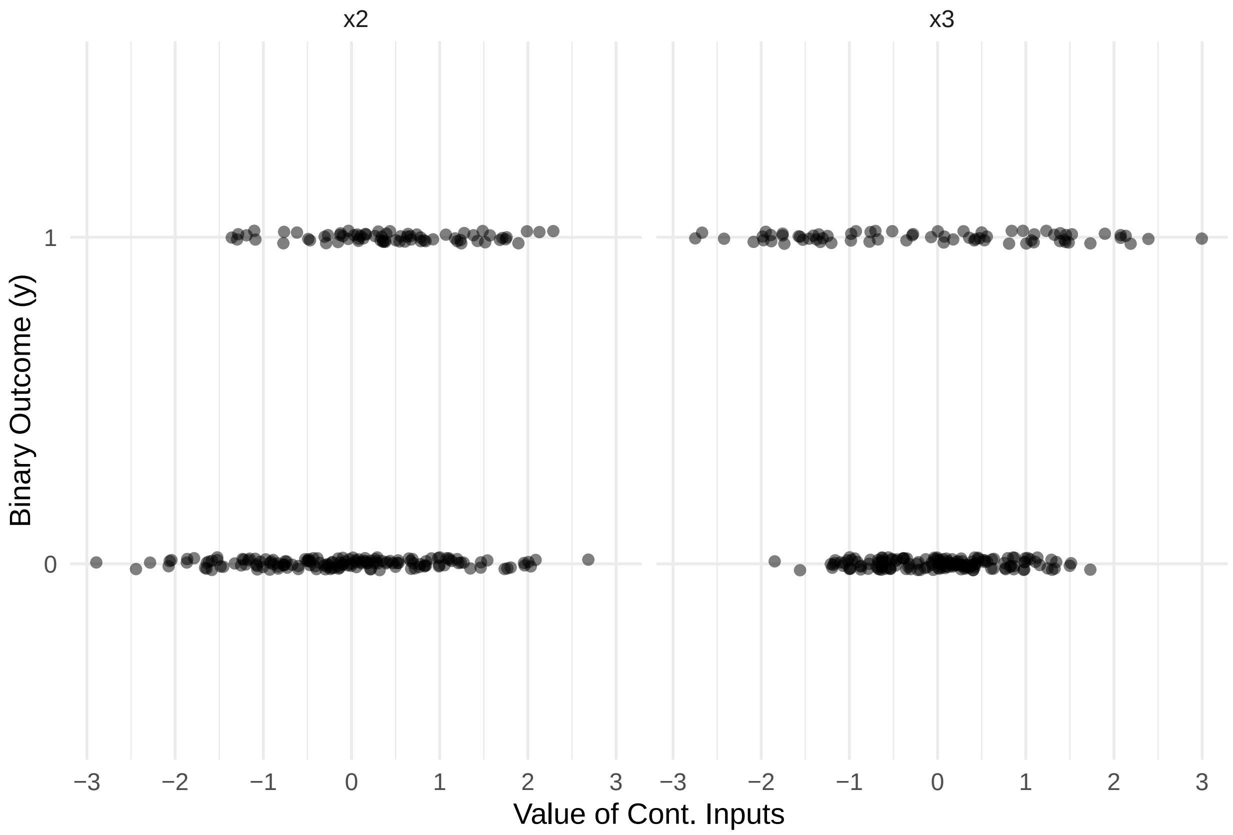
SOLUTION

1f)

Let's now visualize the binary outcome with respect to the continuous inputs. You will use the `geom_jitter()` function instead of the `geom_point()` function to create the scatter plot. The `geom_jitter()` function adds small amounts of random noise to "jitter" or perturb the locations of the points. This will make it easier to see the individual observations of the binary outcome. You **MUST** use the long-format data for this question.

Pipe the long-format data to `ggplot()` and map the `x` aesthetic to the `value` variable and the `y` aesthetic to the `y` variable. Add a `geom_jitter()` layer where you specify the `height` and `width` arguments to be 0.02 and 0, respectively. Do NOT set `height` and `width` within the `aes()` function. Facet by the continuous inputs by including a `facet_wrap()` layer with the facets "a function of" the `name` column.

```
ggplot(lf1, aes(x = value, y = as.factor(y))) +
  geom_jitter(height = 0.02, width = 0, alpha = 0.5) +
  facet_wrap(~ name) +
  labs(x = "Value of Cont. Inputs", y = "Binary Outcome (y)") +
  theme_minimal()
```



SOLUTION

1g)

We can include a logistic regression smoother to help visualize the changes in the event probability. You will use the `geom_smooth()` function to do so, but you will need to change the arguments compared to previous assignments that focused on regression.

Create the same plot as 1f) but include `geom_smooth()` as a layer between `geom_jitter()` and `facet_wrap()`. Specify the `formula` argument to `y ~ x`, the `method` argument to be `glm`, and the `method.args` argument to be `list(family = 'binomial')`.

The `formula` argument are "local" variables associated with the aesthetics. Thus the formula `y ~ x` means the `y` aesthetic is linearly related to the `x` aesthetic. However, by specifying `method = glm` and `method.args = list(family = 'binomial')` you are instructing `geom_smooth()` to fit a logistic regression model. Thus, you are actually specifying that the *linear predictor*, the log-odds ratio is linearly related to the `x` aesthetic.

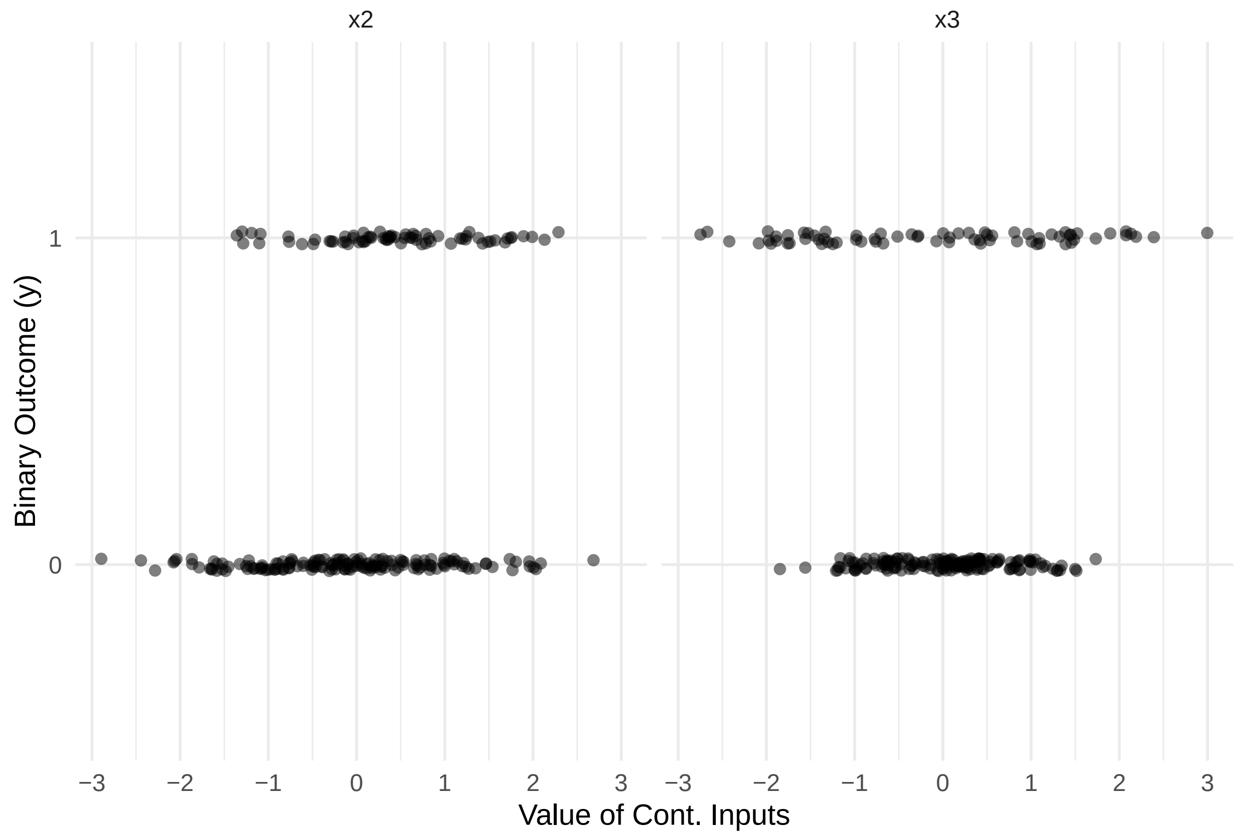
```
ggplot(lf1, aes(x = value, y = as.factor(y))) +
  geom_jitter(height = 0.02, width = 0, alpha = 0.5) +
  geom_smooth(formula = y ~ x, method = "glm", method.args = list(family = "binomial")) +
  facet_wrap(~ name) +
  labs(x = "Value of Cont. Inputs", y = "Binary Outcome (y)") +
  theme_minimal()
```

SOLUTION

```
## Warning: glm.fit: algorithm did not converge
```

```
## Warning: Computation failed in `stat_smooth()`
## Caused by error:
```

```
## ! y values must be 0 <= y <= 1

## Warning: glm.fit: algorithm did not converge

## Warning: Computation failed in `stat_smooth()`
## Caused by error:
## ! y values must be 0 <= y <= 1
```



1h)

Let's check if the categorical input influences the event probability trends with respect to the continuous inputs.

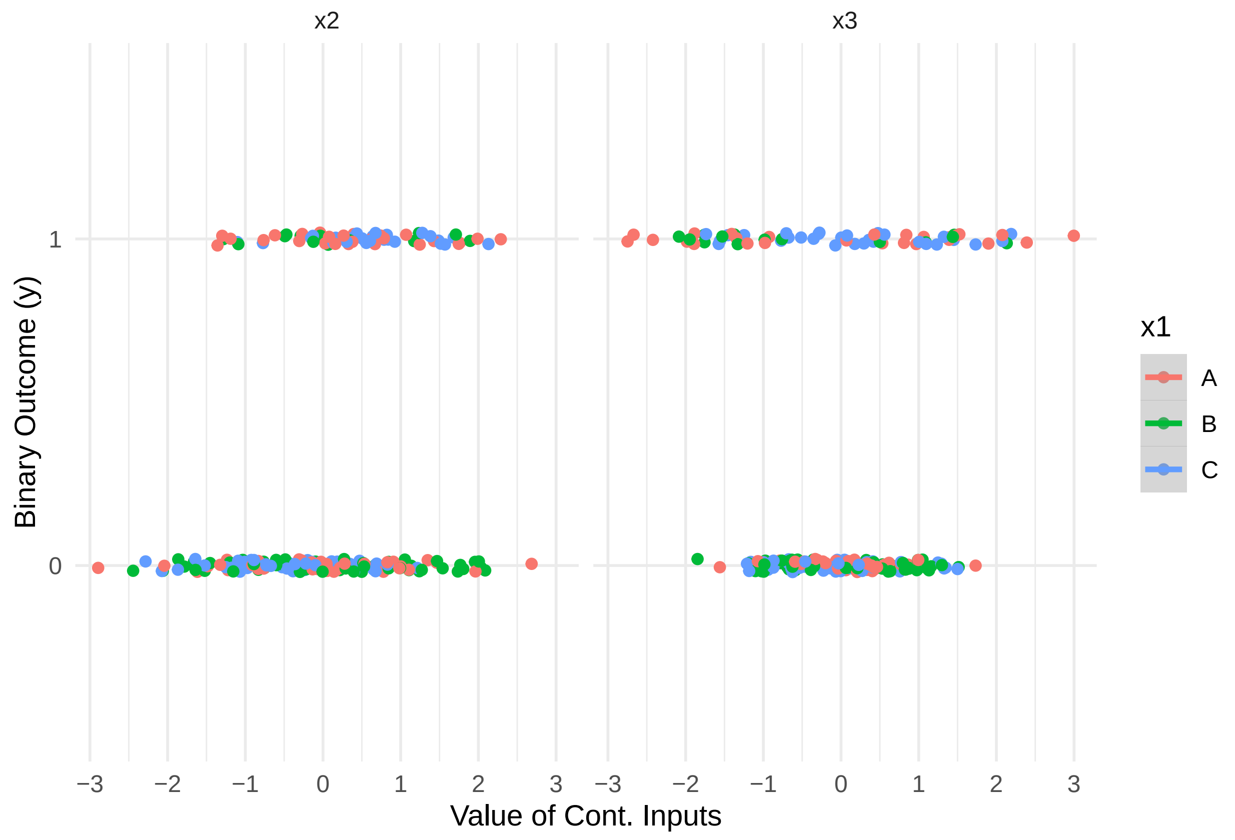
Create the same figure as in 1g), except this time map the `color` and `fill` aesthetics within the `geom_smooth()` layer to the `x1` variable. You must also map the `color` aesthetic within the `geom_jitter()` layer to the `x1` variable.

Based on your figure do the trends appear to depend on the categorical input?

SOLUTION Based on the figure below, we can conclude that the trends do not appear to depend on the categorical input `x1`.

```r
ggplot(lf1, aes(x = value, y = as.factor(y))) +
  geom_jitter(aes(color = x1), height = 0.02, width = 0) +
  geom_smooth(aes(color = x1, fill = x1), formula = y ~ x, method = "glm",
              method.args = list(family = "binomial")) +
  facet_wrap(~ name) +
  labs(x = "Value of Cont. Inputs", y = "Binary Outcome (y)") +
  theme_minimal()
```

```
## Warning: Computation failed in `stat_smooth()`
## Computation failed in `stat_smooth()`
## Caused by error:
## ! y values must be 0 <= y <= 1
```



1i)

The previous figures used the "basic" formula of `y ~ x` within `geom_smooth()`. However, we can try more complex relationships within `geom_smooth()`. For example, let's consider quadratic relationships between the log-odds ratio (linear predictor) and the continuous inputs.

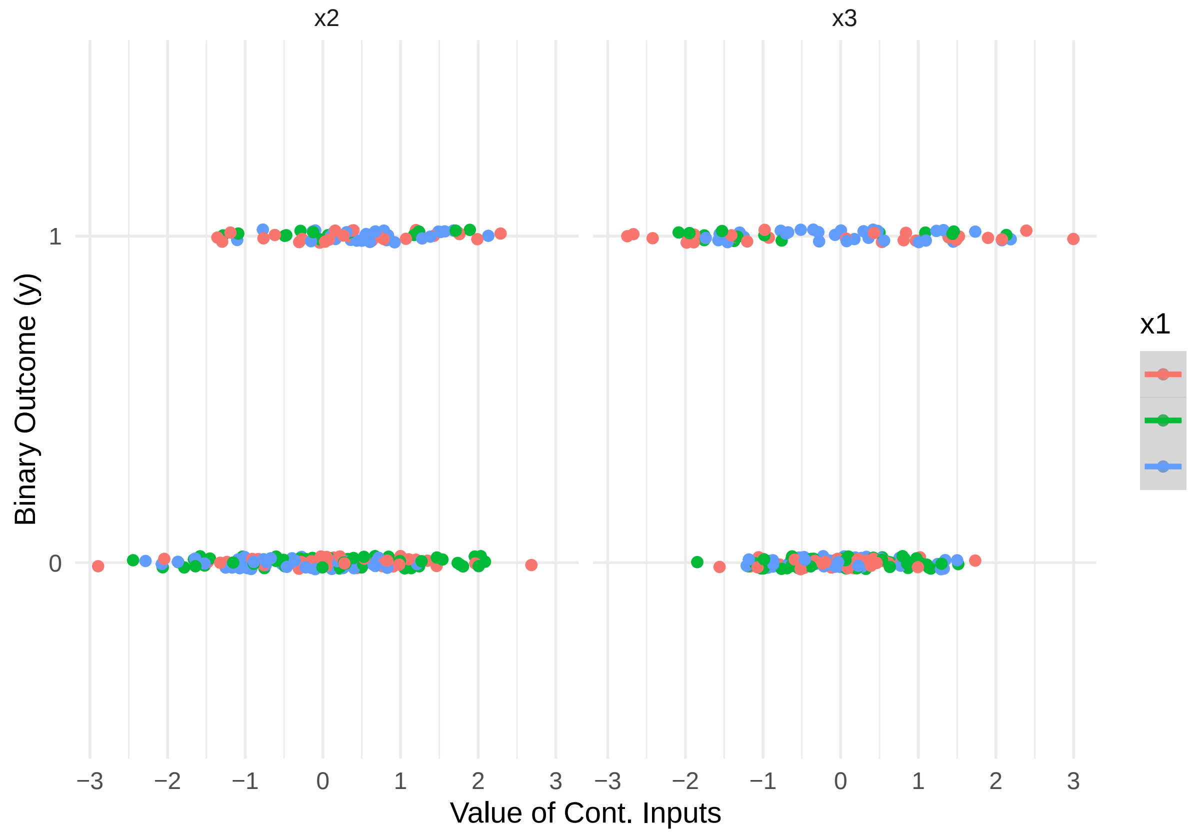
Create the same figure as 1h), except this time specify the `formula` argument to be `y ~ x + I(x^2)`. Use the same set of aesthetics as 1h) including the `color` and `fill` aesthetics.

Does the quadratic relationship appear to be consistent with the data for either of the 2 inputs?

SOLUTION As we can see in the plot below and the plot in Problem 1h), quadratic and linear relationships give similar results.

```r
ggplot(lf1, aes(x = value, y = as.factor(y))) +
  geom_jitter(aes(color = x1), height = 0.02, width = 0) +
  geom_smooth(aes(color = x1, fill = x1), formula = y ~ x + I(x^2), method = "glm",
              method.args = list(family = "binomial")) +
  facet_wrap(~ name) +
  labs(x = "Value of Cont. Inputs", y = "Binary Outcome (y)") +
  theme_minimal()
```

```
## Warning: Computation failed in `stat_smooth()`
## Computation failed in `stat_smooth()`
## Caused by error:
## ! y values must be 0 <= y <= 1
```



Problem 02

Now that you have explored the data, it's time to start modeling! You will fit multiple non-Bayesian logistic regression models using `glm()`. These models will range from simple to complex. You do not need to standardize the continuous inputs, they have already been standardized for you. You can focus on the fitting the models.

BE CAREFUL!! Make sure you set the `family` argument in `glm()` correctly!!! The `family` argument was discussed earlier in the semester.

2a)

You must fit the following models:

A: Categorical input only
B: Linear additive features using the continuous inputs only
C: Linear additive features using all inputs (categorical and continuous)
D: Interact the categorical input with the continuous inputs
E: Add the categorical input to linear main effects and interaction between the continuous inputs
F: Interact the categorical input with main effect and interaction features associated with the continuous inputs
G: Add the categorical input to the linear main continuous effects, interaction between continuous, and quadratic continuous features
H: Interact the categorical input to the linear main continuous effects, interaction between continuous, and quadratic continuous features

You must name your models `modA` through `modH`.

You do not need to fit all models in a single code chunk.

```
modA <- glm(y ~ x1, data = df1, family = "binomial")
```

```
modB <- glm(y ~ x2 + x3, data = df1, family = "binomial")
```

```
modC <- glm(y ~ x1 + x2 + x3, data = df1, family = "binomial")
```

```
modD <- glm(y ~ x1 * x2 + x1 * x3, data = df1, family = "binomial")
```

```
modE <- glm(y ~ x1 + x2 * x3, data = df1, family = "binomial")
```

```
modF <- glm(y ~ x1 * (x2 + x3 + x2:x3), data = df1, family = "binomial")
```

```
modG <- glm(y ~ x1 + x2 + x3 + x2:x3 + I(x2^2) + I(x3^2), data = df1, family = "binomial")
```

```
modH <- glm(y ~ x1 * (x2 + x3 + x2:x3 + I(x2^2) + I(x3^2)), data = df1, family = "binomial")
```

SOLUTION

2b)

Which of the 8 models is the best? Which of the 8 models is the second best?

State the performance metric you used to make your selection.

HINT: The `broom::glance()` function will help here…

SOLUTION Based on the BIC metric values expressed below, we can conclude that `modG` is the best model and `modH` is the second best model.

```
broom::glance(modA)
```

```
## # A tibble: 1 x 8
##   null.deviance df.null logLik   AIC   BIC deviance df.residual  nobs
##           <dbl>   <int>  <dbl> <dbl> <dbl>    <dbl>       <int> <int>
## 1          281.     224  -137.  279.  290.     273.         222   225
```

```
broom::glance(modB)
```

```
## # A tibble: 1 x 8
##   null.deviance df.null logLik   AIC   BIC deviance df.residual  nobs
##           <dbl>   <int>  <dbl> <dbl> <dbl>    <dbl>       <int> <int>
## 1          281.     224  -132.  270.  281.     264.         222   225
```

```
broom::glance(modC)
```

```
## # A tibble: 1 x 8
##   null.deviance df.null logLik   AIC   BIC deviance df.residual  nobs
##           <dbl>   <int>  <dbl> <dbl> <dbl>    <dbl>       <int> <int>
## 1          281.     224  -128.  265.  282.     255.         220   225
```

```r
broom::glance(modD)
```

```
## # A tibble: 1 x 8
##   null.deviance df.null logLik   AIC   BIC deviance df.residual  nobs
##           <dbl>   <int>  <dbl> <dbl> <dbl>    <dbl>       <int> <int>
## 1          281.     224  -118.  254.  285.     236.         216   225
```

```r
broom::glance(modE)
```

```
## # A tibble: 1 x 8
##   null.deviance df.null logLik   AIC   BIC deviance df.residual  nobs
##           <dbl>   <int>  <dbl> <dbl> <dbl>    <dbl>       <int> <int>
## 1          281.     224  -128.  267.  288.     255.         219   225
```

```r
broom::glance(modF)
```

```
## # A tibble: 1 x 8
##   null.deviance df.null logLik   AIC   BIC deviance df.residual  nobs
##           <dbl>   <int>  <dbl> <dbl> <dbl>    <dbl>       <int> <int>
## 1          281.     224  -115.  253.  294.     229.         213   225
```

```r
broom::glance(modG)
```

```
## # A tibble: 1 x 8
##   null.deviance df.null logLik   AIC   BIC deviance df.residual  nobs
##           <dbl>   <int>  <dbl> <dbl> <dbl>    <dbl>       <int> <int>
## 1          281.     224  -87.2  190.  218.     174.         217   225
```

```r
broom::glance(modH)
```

```
## # A tibble: 1 x 8
##   null.deviance df.null logLik   AIC   BIC deviance df.residual  nobs
##           <dbl>   <int>  <dbl> <dbl> <dbl>    <dbl>       <int> <int>
## 1          281.     224  -71.1  178.  240.     142.         207   225
```

2c)

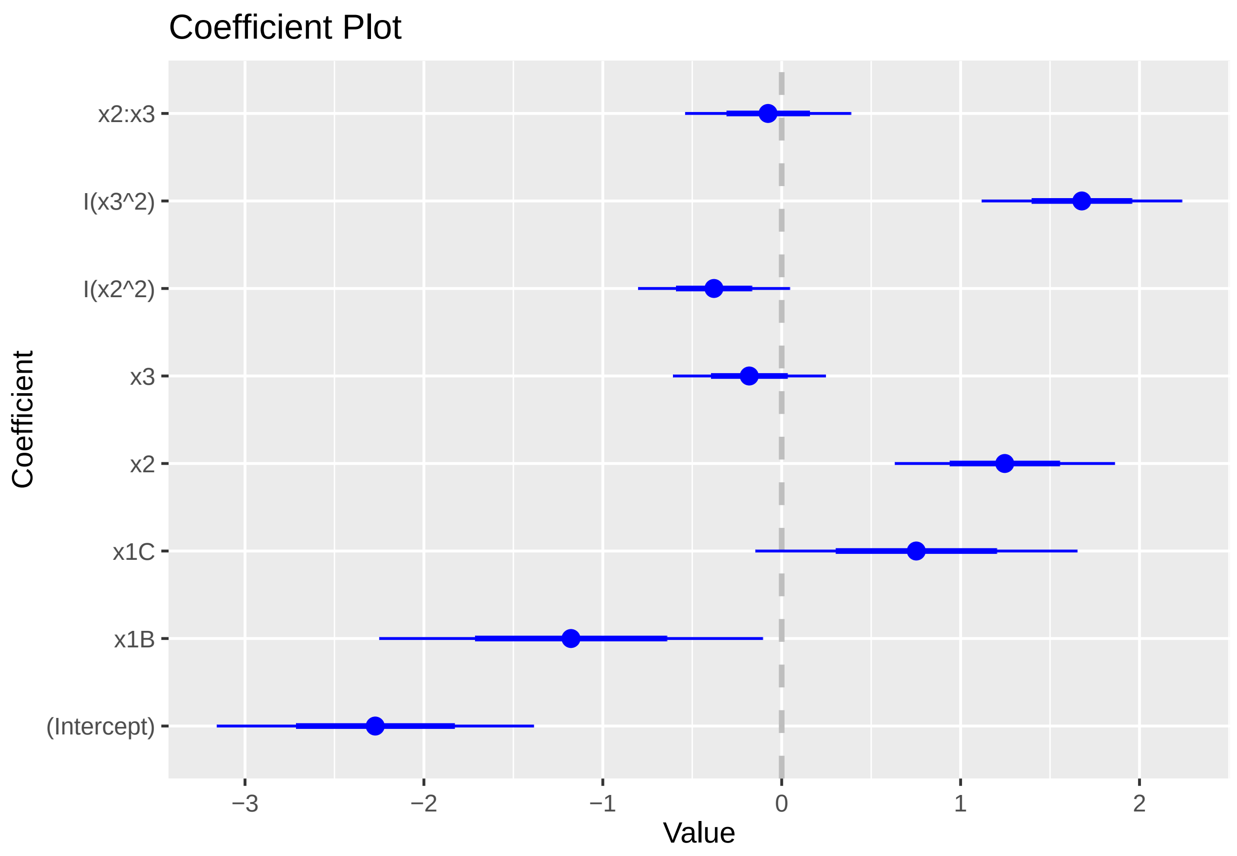
Create the coefficient plot associated with your best and second best models. How many coefficients are statistically significant in each model?

You should use the `coefplot::coefplot()` function to create the plots.

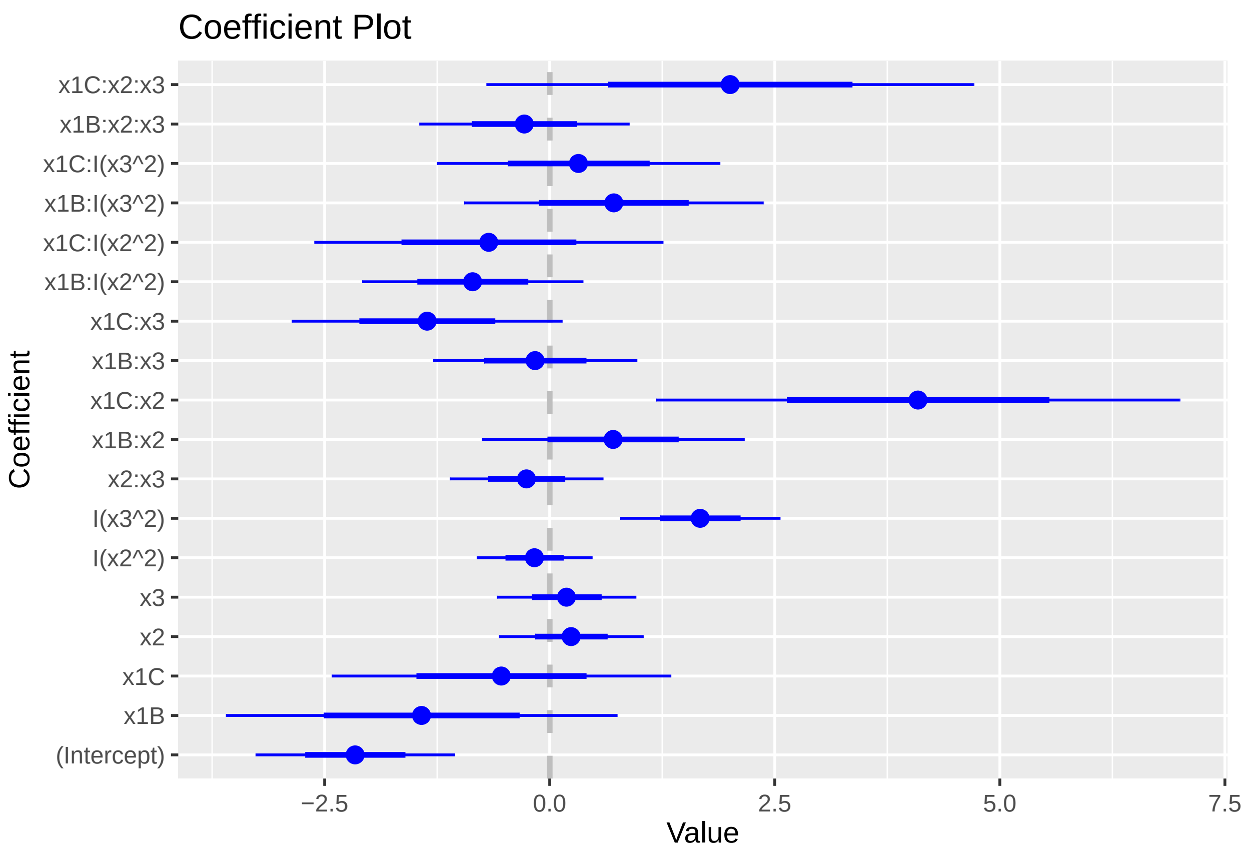
SOLUTION

1) Based on the 1st plot below, we can conclude that 4 coefficients are statistically significant in the best model `modG`.
2) Based on the 2nd plot below, we can conclude that 3 coefficients are statistically significant in the second best model `modH`.

```r
coefplot::coefplot(modG, main = "Coefficient Plot for Best Model modG")
```



```
coefplot::coefplot(modH, main = "Coefficient Plot for Second Best Model modH")
```



Problem 03

Now that you have an idea about the relationships, it's time to consider a more detailed view of the uncertainty by fitting Bayesian logistic regression models. You defined log-posterior functions for linear models in previous assignments. You worked with simple linear relationships, interactions, polynomials, and more complex spline basis features. In lecture, we discussed how the linear model framework can be generalized to handle non-continuous binary outcomes. The likelihood changed from a Gaussian to a Binomial distribution and a non-linear **link** function is required. In this way, the regression model is applied to a linear predictor which "behaves" like the trend in an ordinary linear model. In this problem, you will define the log-posterior function for logistic regression. By doing so you will be able to directly contrast what you did to define the log-posterior function for the linear model in previous assignments.

3a)

The complete probability model for logistic regression consists of the likelihood of the response y_n given the event probability μ_n, the inverse link function between the probability of the event, μ_n, and the linear predictor, η_n, and the prior on all linear predictor model coefficients $\boldsymbol{\beta}$.

As in lecture, you will assume that the $\boldsymbol{\beta}$-parameters are a-priori independent Gaussians with a shared prior mean μ_β and a shared prior standard deviation τ_β.

Write out complete probability model for logistic regression. You must write out the n-th observation's linear predictor using the inner product of the n-th row of a design matrix $\mathbf{x}_{n,:}$ and the unknown $\boldsymbol{\beta}$-parameter column vector. You can assume that the number of unknown coefficients is equal to $D+1$.

You are allowed to separate each equation into its own equation block.

HINT: The "given" sign, the vertical line, $\mid$, is created by typing `\mid` in a latex math expression. The product symbol (the giant PI) is created with `prod_{}^{}`.

SOLUTION

1. Likelihood of the Response y_n Given the Event Probability μ_n:

$$p(y_n \mid \mu_n) = \mu_n^{y_n}(1-\mu_n)^{1-y_n}$$

   This is the Bernoulli likelihood for the n-th observation, where y_n is the binary response (0 or 1), and μ_n is the probability of the event.

2. Inverse Link Function Between the Probability of the Event μ_n and the Linear Predictor η_n:

$$\mu_n = \frac{1}{1+\exp(-\eta_n)}$$

   This is the logistic function, which maps the linear predictor η_n to the event probability μ_n.

3. n-th Observation's Linear Predictor Using the Inner Product:

$$\eta_n = \mathbf{x}_{n,:}\boldsymbol{\beta}$$

   Here, $\mathbf{x}_{n,:}$ is the n-th row of the design matrix (including a 1 for the intercept), and $\boldsymbol{\beta}$ is the column vector of unknown coefficients $[\beta_0, \beta_1, \cdots, \beta_D]$.

4. The complete model is then the product of the N individual likelihoods:

$$p(\boldsymbol{y} \mid \boldsymbol{\beta}, \boldsymbol{x}) \sim \prod_{n=1}^{N} \text{Bernoulli}(y_n \mid \text{logit}^{-1}[\mathbf{x}_{n,:}\boldsymbol{\beta}])$$

3b)

You will fit 8 Bayesian logistic regression models using the same linear predictor trend expressions that you used in the non-Bayesian logistic regression models. You will program the log-posterior function in the same style as the linear model log-posterior functions. This allows you to use the same Laplace Approximation strategy to execute the Bayesian inference.

You must first define the design matrices for each of the 8 models. You must name the design matrices `Xmat_A` through `Xmat_H`. As a reminder, the 8 models are provided below.

A: Categorical input only
B: Linear additive features using the continuous inputs only
C: Linear additive features using all inputs (categorical and continuous)
D: Interact the categorical input with the continuous inputs
E: Add the categorical input to linear main effects and interaction between the continuous inputs
F: Interact the categorical input with main effect and interaction features associated with the continuous inputs
G: Add the categorical input to the linear main continuous effects, interaction between continuous, and quadratic continuous features
H: Interact the categorical input to the linear main continuous effects, interaction between continuous, and quadratic continuous features

SOLUTION Create the design matrices for the 8 models. Add your code chunks below.

```
Xmat_A <- model.matrix(~ x1, data = df1)
Xmat_B <- model.matrix(~ x2 + x3, data = df1)
Xmat_C <- model.matrix(~ x1 + x2 + x3, data = df1)
Xmat_D <- model.matrix(~ x1 * x2 + x1 * x3, data = df1)
```

```r
Xmat_E <- model.matrix(~ x1 + x2 * x3, data = df1)
Xmat_F <- model.matrix(~ x1 * (x2 + x3 + x2:x3), data = df1)
Xmat_G <- model.matrix(~ x1 + x2 + x3 + x2:x3 + I(x2^2) + I(x3^2), data = df1)
Xmat_H <- model.matrix(~ x1 * (x2 + x3 + x2:x3 + I(x2^2) + I(x3^2)), data = df1)
```

3c)

The log-posterior function you will program requires the design matrix, the observed output vector, and the prior specification. In previous assignments, you provided that information with a list. You will do the same thing in this assignment. The code chunk below is started for you. The lists follow the same naming convention as the design matrices. The `info_A` list corresponds to the information for model A, while `info_H` corresponds to the information for model H. You must assign the design matrix to the corresponding list and complete the rest of the required information. The observed binary outcome vector must be assigned to the `yobs` field. The prior mean and prior standard deviation must be assigned to the `mu_beta` and `tau_beta` fields, respectively.

Complete the code chunk below by completing the lists of required for each model. The list names are consistent with the design matrix names you defined in the previous problem. You must use a prior mean of 0 and a prior standard deviation of 4.5.

```r

info_A <- list(
  yobs = df1$y,
  design_matrix = Xmat_A,
  mu_beta = 0,
  tau_beta = 4.5
)

info_B <- list(
  yobs = df1$y,
  design_matrix = Xmat_B,
  mu_beta = 0,
  tau_beta = 4.5
)

info_C <- list(
  yobs = df1$y,
  design_matrix = Xmat_C,
  mu_beta = 0,
  tau_beta = 4.5
)

info_D <- list(
  yobs = df1$y,
  design_matrix = Xmat_D,
  mu_beta = 0,
  tau_beta = 4.5
)

info_E <- list(
  yobs = df1$y,
  design_matrix = Xmat_E,
  mu_beta = 0,
  tau_beta = 4.5
)

info_F <- list(
  yobs = df1$y,
  design_matrix = Xmat_F,
  mu_beta = 0,
  tau_beta = 4.5
)

info_G <- list(
  yobs = df1$y,
  design_matrix = Xmat_G,
  mu_beta = 0,
  tau_beta = 4.5
)

info_H <- list(
  yobs = df1$y,
  design_matrix = Xmat_H,
  mu_beta = 0,
  tau_beta = 4.5
)
```

SOLUTION

3d)

You will now define the log-posterior function for logistic regression, `logistic_logpost()`. The first argument to `logistic_logpost()` is the vector of unknowns and the second argument is the list of required information. You will assume that the variables within the `my_info` list are those contained in the `info_A` through `info_H` lists you defined previously.

Complete the code chunk to define the `logistic_logpost()` function. The comments describe what you need to fill in. Do you need to separate out the β-parameters from the vector of `unknowns`?

After you complete `logistic_logpost()`, test it by setting the `unknowns` vector to be a vector of -1's and then 2's for the model A case (the model with a only the categorical input). If you have successfully programmed the function you should get `-164.6906` and `-541.6987` for the -1 test case and +2 test case, respectively.

SOLUTION Do you need to separate the β-parameters from the `unknowns` vector?

```
logistic_logpost <- function(unknowns, my_info) {
  # extract the design matrix and assign to X
  X <- my_info$design_matrix

  # calculate the linear predictor
  eta <- X %*% unknowns

  # calculate the event probability
  mu <- boot::inv.logit(eta)

  # evaluate the log-likelihood
  log_lik <- sum(dbinom(x = my_info$yobs, size = 1, prob = mu, log = TRUE))
  # evaluate the log-prior
  log_prior <- sum(dnorm(unknowns, mean = my_info$mu_beta, sd = my_info$tau_beta, log = TRUE))

  # sum together
  log_lik + log_prior
}
```

Test out your function using the `info_A` information and setting the unknowns to a vector of -1's.

```
test_unknowns1 <- rep(-1, ncol(Xmat_A))
logistic_logpost(test_unknowns1, info_A)
```

```
## [1] -164.6906
```

Test out your function using the `info_A` information and setting the unknowns to a vector of 2's.

```
test_unknowns2 <- rep(2, ncol(Xmat_A))
logistic_logpost(test_unknowns2, info_A)
```

```
## [1] -541.6987
```

3e)

The `my_laplace()` function is provided to you in the code chunk below.

```
my_laplace <- function(start_guess, logpost_func, ...)
{
  # code adapted from the `LearnBayes`` function `laplace()`
  fit <- optim(start_guess,
               logpost_func,
               gr = NULL,
               ...,
               method = "BFGS",
               hessian = TRUE,
               control = list(fnscale = -1, maxit = 5001))

  mode <- fit$par
  post_var_matrix <- -solve(fit$hessian)
  p <- length(mode)
  int <- p/2 * log(2 * pi) + 0.5 * log(det(post_var_matrix)) + logpost_func(mode, ...)
  # package all of the results into a list
  list(mode = mode,
       var_matrix = post_var_matrix,
       log_evidence = int,
       converge = ifelse(fit$convergence == 0,
                         "YES",
                         "NO"),
       iter_counts = as.numeric(fit$counts[1]))
}
```

You will use `my_laplace()` to execute the Laplace Approximation for all 8 models. You must use an initial guess of zero for all unknowns for each model.

Perform the Laplace Approximation for all 8 models. Assign the results to the `laplace_A` through `laplace_H` objects. The names should be consistent with the design matrices and lists of required information. Thus, `laplace_A` must correspond to the `info_A` and `Xmat_A` objects.

Should you be concerned that the initial guess will impact the results?

SOLUTION Does the initial guess matter?

Yes the initial guess does matter because the models are non-linear model wrt beta (due to the logit transformation).

Below the code chunks test that the laplace transform converges in each case.

```
laplace_A <- my_laplace(rep(0, ncol(Xmat_A)), logistic_logpost, info_A)
laplace_A$converge
```

```
## [1] "YES"
```

```
laplace_B <- my_laplace(rep(0, ncol(Xmat_B)), logistic_logpost, info_B)
laplace_B$converge
```

```
## [1] "YES"
```

```
laplace_C <- my_laplace(rep(0, ncol(Xmat_C)), logistic_logpost, info_C)
laplace_C$converge
```

```
## [1] "YES"
```

```
laplace_D <- my_laplace(rep(0, ncol(Xmat_D)), logistic_logpost, info_D)
laplace_D$converge
```

```
## [1] "YES"
```

```
laplace_E <- my_laplace(rep(0, ncol(Xmat_E)), logistic_logpost, info_E)
laplace_E$converge
```

```
## [1] "YES"
```

```
laplace_F <- my_laplace(rep(0, ncol(Xmat_F)), logistic_logpost, info_F)
laplace_F$converge
```

```
## [1] "YES"
```

```
laplace_G <- my_laplace(rep(0, ncol(Xmat_G)), logistic_logpost, info_G)
laplace_G$converge
```

```
## [1] "YES"
```

```
laplace_H <- my_laplace(rep(0, ncol(Xmat_H)), logistic_logpost, info_H)
laplace_H$converge
```

```
## [1] "YES"
```

3f)

The `laplace_A` object is the Bayesian version of `modA` that you fit previously in Problem 2a). Let's compare the Bayesian result to the non-Bayesian result.

Calculate the 95% confidence interval on the regression coefficients associated with `laplace_A` and dispaly the interval bounds to the screen. Which features are statistically significant according to the Bayesian model? Are the results consistent with the non-Bayesian model, `modA`?

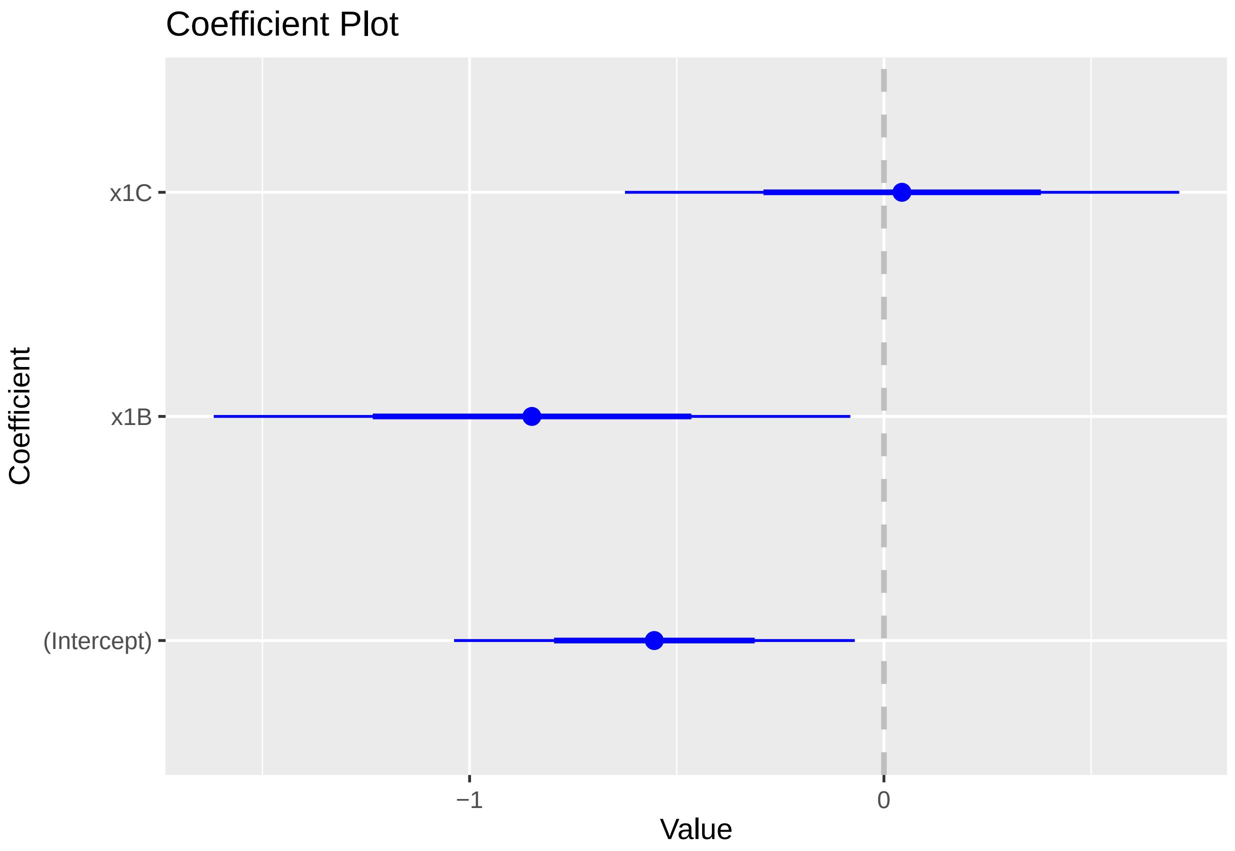
SOLUTION As it can be seen on the table below, intercept and x1B features are statistically significant according to the Bayesian model, because 0 is not in the 95% confidence interval on the regression coefficients for those features.

```r
mode_A <- laplace_A$mode
var_matrix_A <- laplace_A$var_matrix

std_dev_A <- sqrt(diag(var_matrix_A))

ci_lower_A <- mode_A - 2 * std_dev_A
ci_upper_A <- mode_A + 2 * std_dev_A

ci_A <- cbind(ci_lower_A, ci_upper_A)
print(ci_A)
```

```
##      ci_lower_A  ci_upper_A
## [1,] -1.0359508 -0.07406797
## [2,] -1.6087096 -0.08184096
## [3,] -0.6213987  0.70956971
```

Coefficient plot for the model “modA” is seen below. We can observe that the results are consistent with the non-Bayesian model, `modA`.

```r
coefplot::coefplot(modA, main = "Coefficient Plot for modA")
```

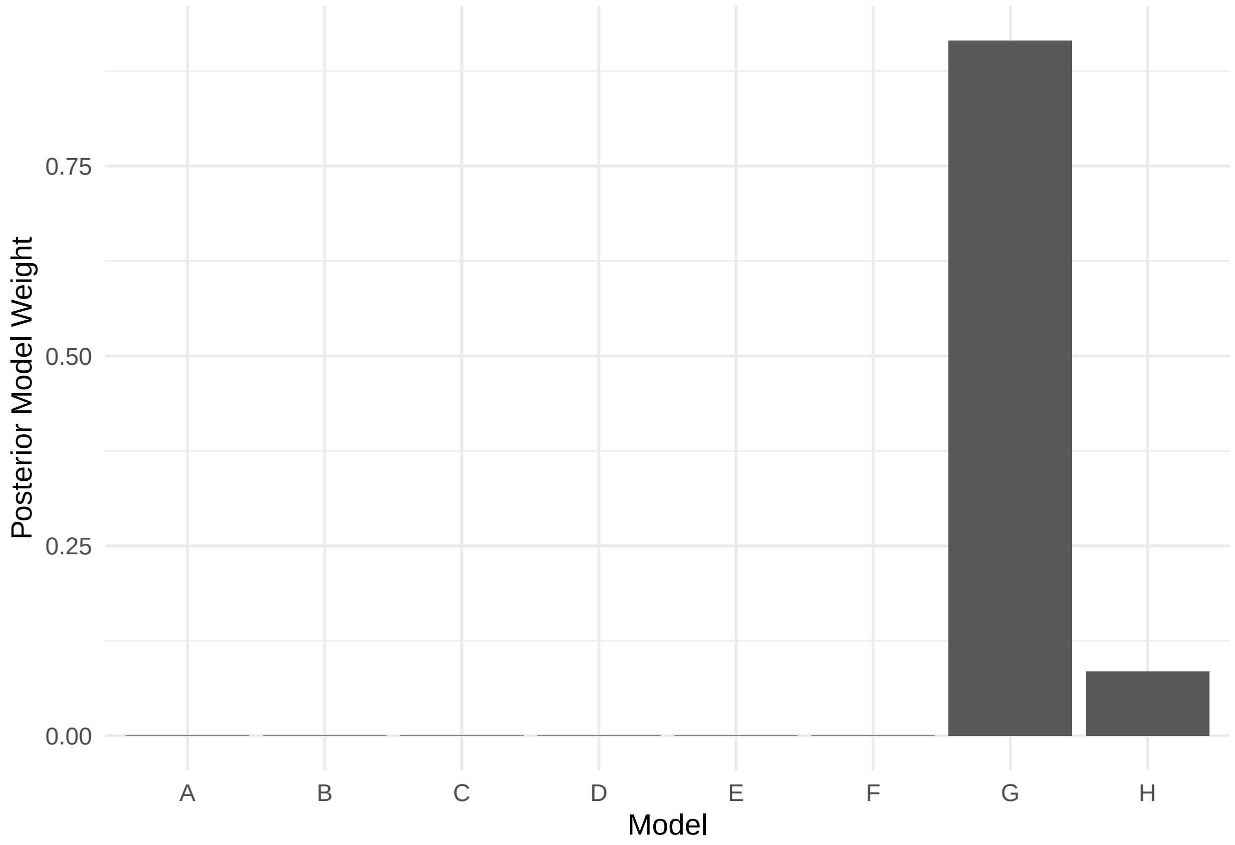


3g)

You trained 8 Bayesian logistic regression models. Let’s identify the best using the Evidence based approach!

Calculate the posterior model weight associated with each of the 8 models. Create a bar chart that shows the posterior model weight for each model. The models should be named `'A'` through `'H'`.

```r
log_evidence <- c(laplace_A$log_evidence, laplace_B$log_evidence,
                  laplace_C$log_evidence, laplace_D$log_evidence,
                  laplace_E$log_evidence, laplace_F$log_evidence,
                  laplace_G$log_evidence, laplace_H$log_evidence)

model_weights <- exp(log_evidence) / sum(exp(log_evidence))

model_names <- LETTERS[1:8]

weight_data <- data.frame(Model = model_names, Weight = model_weights)

ggplot(weight_data, aes(x = Model, y = Weight)) +
  geom_bar(stat = "identity") +
  labs(x = "Model", y = "Posterior Model Weight") +
  theme_minimal()
```



SOLUTION

3h)

Is your Bayesian identified best model consistent with the non-Bayesian identified best model?

SOLUTION Yes the Bayesian identified best model consistent with the non-Bayesian identified best model. In Problem 2c) and Problem 3g), we can see that the best model and second best model are identified as model G and model H, respectively.

Problem 04

You trained multiple models ranging from simple to complex. You identified the best model using several approaches. It is now time to examine the predictive trends of the models to better interpret their behavior. You will not predict the training set to study the trends. Instead, you will visualize the trends on a specifically designed prediction grid. The code chunk below defines that grid for you using the `expand.grid()` function. If you look closely, the `x3` variable has 51 evenly spaced points between -3 and 3. The `x1` variable has 9 unique values evenly spaced between -3 and 3. These lower and upper bounds are roughly consistent with the training set bounds. The `x1` variable consists of the 3 unique values present in the training set. The `expand.grid()` function creates the full-factorial combination of the 3 variables.

```r
viz_grid <- expand.grid(x1 = unique(df1$x1),
                        x2 = seq(-3, 3, length.out = 9),
                        x3 = seq(-3, 3, length.out = 51),
                        KEEP.OUT.ATTRS = FALSE,
                        stringsAsFactors = FALSE) %>%
  as.data.frame() %>% tibble::as_tibble()

viz_grid %>% glimpse()
```

```
## Rows: 1,377
## Columns: 3
## $ x1 <chr> "C", "B", "A", "C", "B", "A", "C", "B", "A", "C", "B", "A", "C", "B~
## $ x2 <dbl> -3.00, -3.00, -3.00, -2.25, -2.25, -2.25, -1.50, -1.50, -1.50, -0.7~
## $ x3 <dbl> -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3,~
```

The glimpse provided above shows there are 1377 combinations of the 3 inputs. You will therefore make over 1300 predictions to study the trends of the event probability!

4a)

As with previous linear model assignments, you must first generate random posterior samples of the unknown parameters from the Laplace Approximation assumed Multivariate Normal (MVN) distribution. Although you were able to apply the `my_laplace()` function to both the regression and logistic regression settings, you cannot directly apply the `generate_lm_post_samples()` function from your previous assignments. You will therefore adapt `generate_lm_post_samples()` and define `generate_glm_post_samples()`. The code chunk below starts the function for you and uses just two input arguments, `mvn_result` and `num_samples`. You must complete the function.

Why can you not directly use the `generate_lm_post_samples()` function? Since the `length_beta` argument is NOT provided to `generate_glm_post_samples()`, how can you determine the number of β-parameters? Complete the code chunk below by first assigning the number of β-parameters to the `length_beta` variable. Then generate the random samples from the MVN distribution. You do not have to name the variables, you only need to call the correct random number generator.

SOLUTION What do you think? Why do we need a new function compared to the previous assignments?

```r
generate_glm_post_samples <- function(mvn_result, num_samples)
{
  # specify the number of unknown beta parameters
  length_beta <- length(mvn_result$mode)

  # generate the random samples
  beta_samples <- MASS::mvrnorm(n = num_samples,
                                mu = mvn_result$mode,
                                Sigma = mvn_result$var_matrix)
  # change the data type and name
  beta_samples %>%
    as.data.frame() %>% tibble::as_tibble() %>%
    purrr::set_names(sprintf("beta_%02d", (1:length_beta) - 1))
}
```

In logistic regression, we only have betas as unknown (sigma is not an unknown). For that reason we cannot use `generate_lm_post_samples()` function directly in this case.

4b)

You will now define a function which calculates the posterior samples on the linear predictor and the event probability. The function, `post_logistic_pred_samples()` is started for you in the code chunk below. It consists of two input arguments `Xnew` and `Bmat`. `Xnew` is a test design matrix where rows correspond to prediction points. The matrix `Bmat` stores the posterior samples on the β-parameters, where each row is a posterior sample and each column is a parameter.

Complete the code chunk below by using matrix math to calculate the posterior samples of the linear predictor. Then, calculate the posterior smaples of the event probability.

The `eta_mat` and `mu_mat` matrices are returned within a list, similar to how the `Umat` and `Ymat` matrices were returned for the regression problems.

HINT: The `boot::inv.logit()` can take a matrix as an input. When it does, it returns a matrix as a result.

```
post_logistic_pred_samples <- function(Xnew, Bmat)
{
  # calculate the linear predictor at all prediction points and posterior samples
  eta_mat <- Xnew %*% t(Bmat)

  # calculate the event probability
  mu_mat <- boot::inv.logit(eta_mat)

  # book keeping
  list(eta_mat = eta_mat, mu_mat = mu_mat)
}
```

SOLUTION

4c)

The code chunk below defines a function `summarize_logistic_pred_from_laplace()` which manages the actions necessary to summarize posterior predictions of the event probability. The first argument, `mvn_result`, is the Laplace Approximation object. The second object is the test design matrix, `Xtest`, and the third argument, `num_samples`, is the number of posterior samples to make. You must follow the comments within the function in order to generate posterior prediction samples of the linear predictor and the event probability, and then to summarize the posterior predictions of the event probability.

The result from `summarize_logistic_pred_from_laplace()` summarizes the posterior predicted event probability with the posterior mean, as well as the 0.05 and 0.95 Quantiles. If you have completed the `post_logistic_pred_samples()` function correctly, the dimensions of the `mu_mat` matrix should be consistent with those from the `Umat` matrix from the regression problems.

The posterior summary statistics summarize the posterior samples. You must therefore choose between `colMeans()` and `rowMeans()` as to how to calculate the posterior mean event probability for each prediction point. The posterior Quantiles are calculated for you.

Follow the comments in the code chunk below to complete the definition of the `summarize_logistic_pred_from_l` function. You must generate posterior samples, make posterior predictions, and then summarize the posterior predictions of the event probability.

HINT: The result from `post_logistic_pred_samples()` is a list.

```
summarize_logistic_pred_from_laplace <- function(mvn_result, Xtest, num_samples)
{
  # generate posterior samples of the beta parameters
  betas <- generate_glm_post_samples(mvn_result, num_samples)

  # data type conversion
  betas <- as.matrix(betas)

  # make posterior predictions on the test set
  pred_test <- post_logistic_pred_samples(Xtest, betas)

  # calculate summary statistics on the posterior predicted probability
  # summarize over the posterior samples

  # posterior mean, should you summarize along rows (rowMeans) or
  # summarize down columns (colMeans) ???
  mu_avg <- rowMeans(pred_test$mu_mat)

  # posterior quantiles
  mu_q05 <- apply(pred_test$mu_mat, 1, stats::quantile, probs = 0.05)
  mu_q95 <- apply(pred_test$mu_mat, 1, stats::quantile, probs = 0.95)

  # book keeping
  tibble::tibble(
    mu_avg = mu_avg,
    mu_q05 = mu_q05,
    mu_q95 = mu_q95
  ) %>%
    tibble::rowid_to_column("pred_id")
}
```

SOLUTION

4d)

You will not make predictions from all 8 models that you previously trained. Instead, you will focus on model D, model G, and model H.

You must define the vizualization grid design matrices consistent with the model D, model G, and model H formulas. You must name the design matrices `Xviz_D`, `Xviz_G`, and `Xviz_H`. You must create the design matrices using the `viz_grid` dataframe which was defined at the start of Problem 04.

```
Xviz_D <- model.matrix(~ x1 * x2 + x1 * x3, data = viz_grid)
Xviz_G <- model.matrix(~ x1 + x2 + x3 + x2:x3 + I(x2^2) + I(x3^2), data = viz_grid)
Xviz_H <- model.matrix(~ x1 * (x2 + x3 + x2:x3 + I(x2^2) + I(x3^2)), data = viz_grid)
```

SOLUTION

4e)

Summarize the posterior predicted event probability associated with the three models on the visualization grid. After making the predictions, a code chunk is provided for you which generates a figure showing how the posterior predicted probability summaries compare with the observed binary outcomes. Which of the three models appear to better capture the trends in the binary outcome?

Call `summarize_logistic_pred_from_laplace()` for the all three models on the visualization grid. The object names specify which model you should make predictions with. For example, `post_pred_summary_D` corresponds to the predictions associated with model D. Specify the number of posterior samples to be 2500. Print the dimensions of the resulting objects to the screen. How many rows are in each data set?

SOLUTION The prediction summarizes should be executed in the code chunk below.

```
set.seed(8123)

post_pred_summary_D <- summarize_logistic_pred_from_laplace(laplace_D, Xviz_D, num_samples = 2500)

post_pred_summary_G <- summarize_logistic_pred_from_laplace(laplace_G, Xviz_G, num_samples = 2500)

post_pred_summary_H <- summarize_logistic_pred_from_laplace(laplace_H, Xviz_H, num_samples = 2500)
```

Print the dimensions of the objects to the screen.

```
dim(post_pred_summary_D)
```

```
## [1] 1377    4
```

```
dim(post_pred_summary_G)
```

```
## [1] 1377    4
```

```
dim(post_pred_summary_H)
```

```
## [1] 1377    4
```

There are 1377 rows in each dataset, corresponding to the number of rows in the viz_grid.

4f)

The code chunk below defines a function for you. The function creates a figure which visualizes the posterior predictive summary statistics of the event probability for a single model. The figure is created to focus on the trend with respect to `x3`. Facets are used to examine the influence of `x2`. The line color and ribbon aesthetics are used to denote the categorical variable `x1`. This figure is specific to the three variable names in this assignment, but it shows the basic layout required for visualizing predictive trends from Bayesian logistic regression models with 3 inputs.

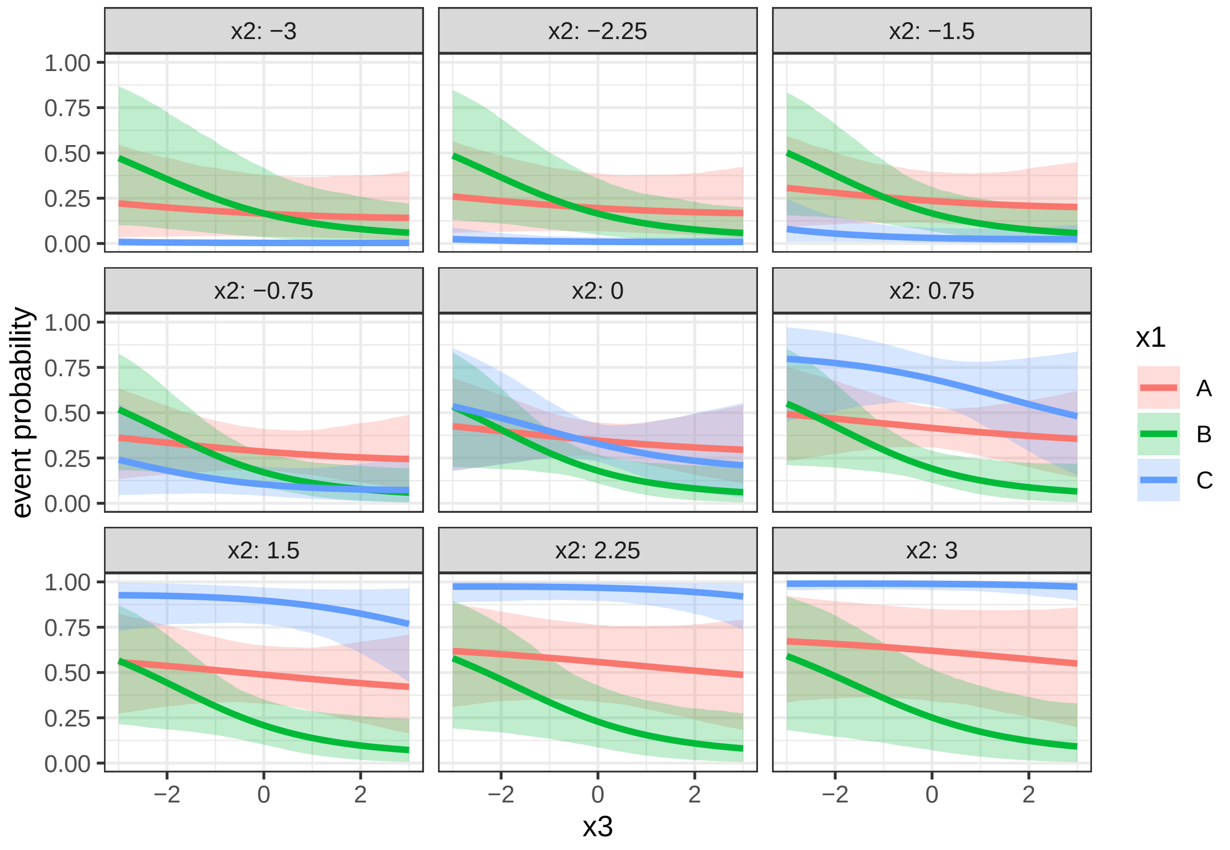
```
viz_bayes_logpost_preds <- function(post_pred_summary, input_df)
{
  post_pred_summary %>%
    left_join(input_df %>% tibble::rowid_to_column('pred_id'),
              by = 'pred_id') %>%
    ggplot(mapping = aes(x = x3)) +
    geom_ribbon(mapping = aes(ymin = mu_q05,
                              ymax = mu_q95,
                              group = interaction(x1, x2),
                              fill = x1),
                alpha = 0.25) +
    geom_line(mapping = aes(y = mu_avg,
                            group = interaction(x1, x2),
                            color = x1),
              size = 1.15) +
    facet_wrap( ~ x2, labeller = 'label_both') +
    labs(y = "event probability") +
    theme_bw()
}
```

Use the `viz_bayes_logpost_preds()` function to visualize posterior predictive trends of the event probability for the 3 models: model D, model G, and model H.

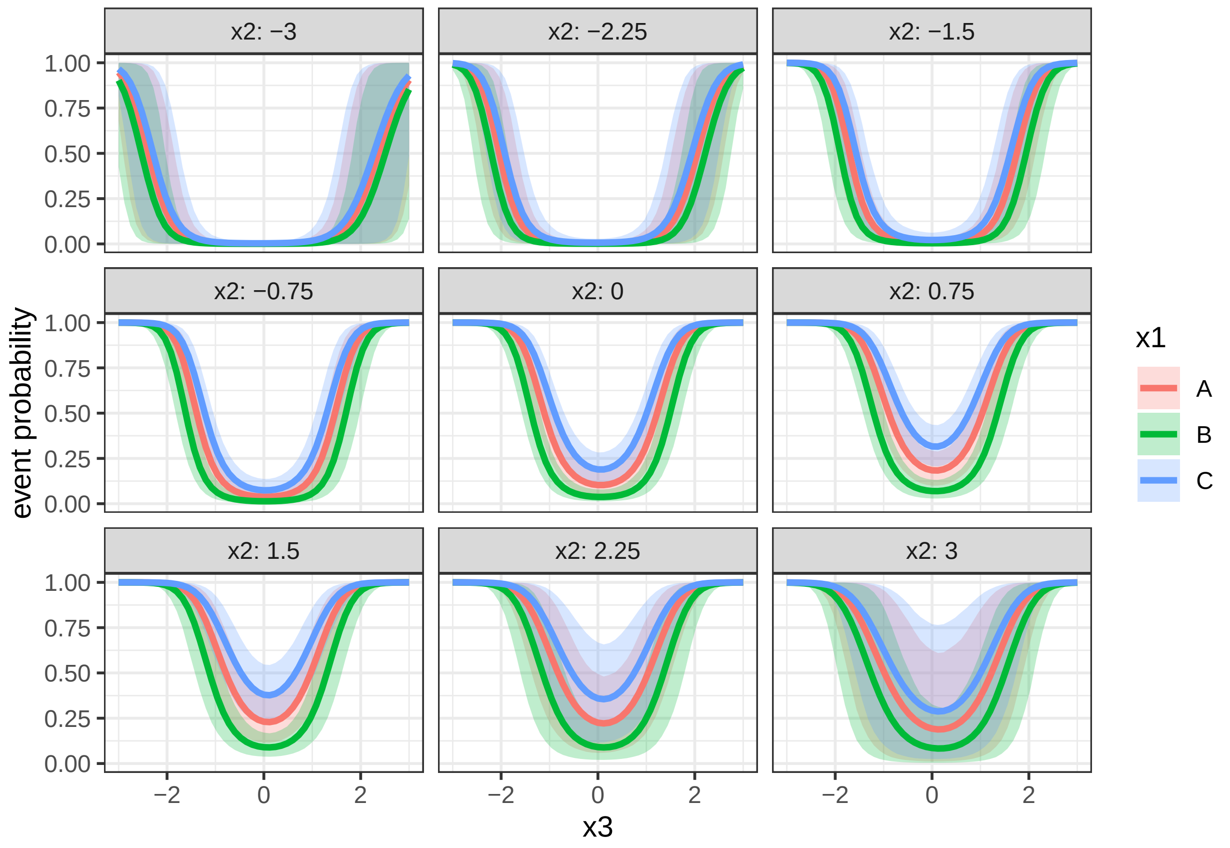
```
viz_bayes_logpost_preds(post_pred_summary_D, viz_grid)
```

SOLUTION

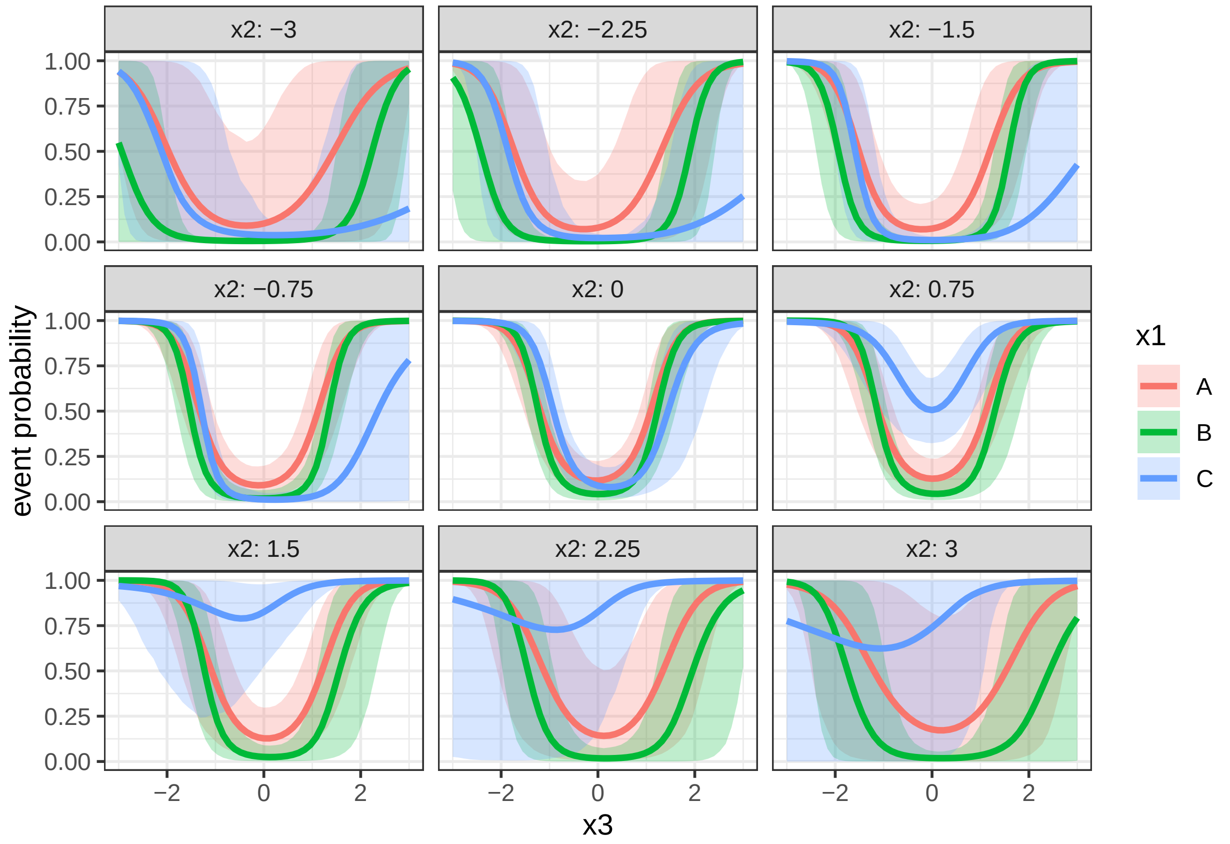
```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```



```
viz_bayes_logpost_preds(post_pred_summary_G, viz_grid)
```



```
viz_bayes_logpost_preds(post_pred_summary_H, viz_grid)
```



4g)

Describe the differences in the predictive trends between the 3 models?

SOLUTION

1) Model D primarily captures interactions between the categorical input x1 and the continuous inputs x2 and x3. The trends observed in this model's predictions will mainly reflect how the influence of x2 and x3 on the event probability changes across the different categories of x1.

2) Models G and H incorporate additional complexities. Both models include quadratic terms for x2 and x3, allowing for non-linear relationships between these continuous inputs and the event probability.

3) Model G adds non-linear terms (quadratic features) to the linear effects, making it more flexible than Model D in capturing non-linear trends. Model H is the most complex, as it includes interactions between the categorical input and both linear and quadratic continuous terms.

4) In the visualizations in Problem 4f), we see Model D showing varying slopes for x3 across different facets of x2 and colors of x1, reflecting the model's focus on interactions without non-linear terms.

5) For Models G and H, the trends show curvature in the relationship between x3 and the event probability. This curvature indicates the models' ability to capture non-linear effects.

6) Finally, Model G (which was the best model), show narrower confidence intervals compared to Model H in x1 variable.

Problem 05

You should have noticed a pattern associated with the 8 models that you previously fit. The most complex model, model H, contains all other models! It is a super set of all features from the simpler models. An alternative approach to training many models of varying complexity is to train a single complex model and use regularization to "turn off" the unimportant features. This way we can find out if the most complex model can be turned into a simpler model of just the most key features we need!

We discussed in lecture how the Lasso penalty or its Bayesian analog the Double-Exponential prior are capable of turning off the unimportant features. We focused on regression problems but Lasso can also be applied to classification problems! In this problem you will use `caret` to manage the training, assessment, and tuning of the `glmnet` elastic net penalized logistic regression model. The code chunk below imports the `caret` package for you.

```r
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.

```r
df_caret <- df1 %>%
  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: 225
## Columns: 4
## $ x1      <chr> "C", "B", "C", "B", "A", "C", "A", "A", "C", "C", "B", "C", "A~
## $ x2      <dbl> 1.682873632, -1.033648456, 0.110854156, 2.032934019, -0.225540~
## $ x3      <dbl> -0.353085685, -0.778102544, 0.757536960, 0.639465847, 0.017483~
## $ outcome <fct> event, non_event, non_event, non_event, non_event, event, even~
```

5a)

You must specify the resampling scheme that caret will use to train, assess, and tune a model. You used `caret` in the previous assignment for a regression problem. Here, you are working with a classification problem and so you cannot use the same performance metric as the previous assignment! Although there are multiple classification metrics we could consider, we will focus on Accuracy in this problem.

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.

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

SOLUTION

5b)

You must train, assess, and tune an elastic model using the default `caret` tuning grid. In the `caret::train()` function you must use the formula interface to specify a model consistent with model H. Thus, your model should interact the categorical input to the linear main continuous effects, interaction between continuous, and quadratic continuous features. However, 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`. Even though the inputs were standardized for you, 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. 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 consistent with model H 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.

```r
set.seed(1234)
enet_default <- caret::train(outcome ~ x1 * (x2 + x3 + x2:x3 + I(x2^2) + I(x3^2)),
                             data = df_caret,
                             method = "glmnet",
                             metric = my_metric,
                             trControl = my_ctrl,
                             preProcess = c("center", "scale"))
print(enet_default)
```

```
## glmnet
##
## 225 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: 202, 202, 202, 203, 202, 202, ...
## Resampling results across tuning parameters:
##
##   alpha  lambda        Accuracy   Kappa
##   0.10   0.0004769061  0.8251372  0.5805807
##   0.10   0.0047690605  0.8282993  0.5802313
##   0.10   0.0476906052  0.8072793  0.4935810
##   0.55   0.0004769061  0.8251372  0.5805807
##   0.55   0.0047690605  0.8298748  0.5824205
##   0.55   0.0476906052  0.8042545  0.4773741
##   1.00   0.0004769061  0.8207235  0.5714822
##   1.00   0.0047690605  0.8328393  0.5904347
##   1.00   0.0476906052  0.8147892  0.5081578
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were alpha = 1 and lambda = 0.004769061.
```

1) The final values used for the model were alpha = 1 and lambda = 0.04769061. This tuning parameter combination is considered to be the best.

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

5c)

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 5-fold with 3-repeat resampling approach?

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

```
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 <- 5
num_repeats <- 3
total_models <- num_combinations * num_folds * num_repeats

num_combinations
```

SOLUTION

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

```
total_models
```

```
## [1] 3750
```

How many?

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

2) Also, 3750 models will be fit assuming the 5-fold with 3-repeat resampling approach.

5d)

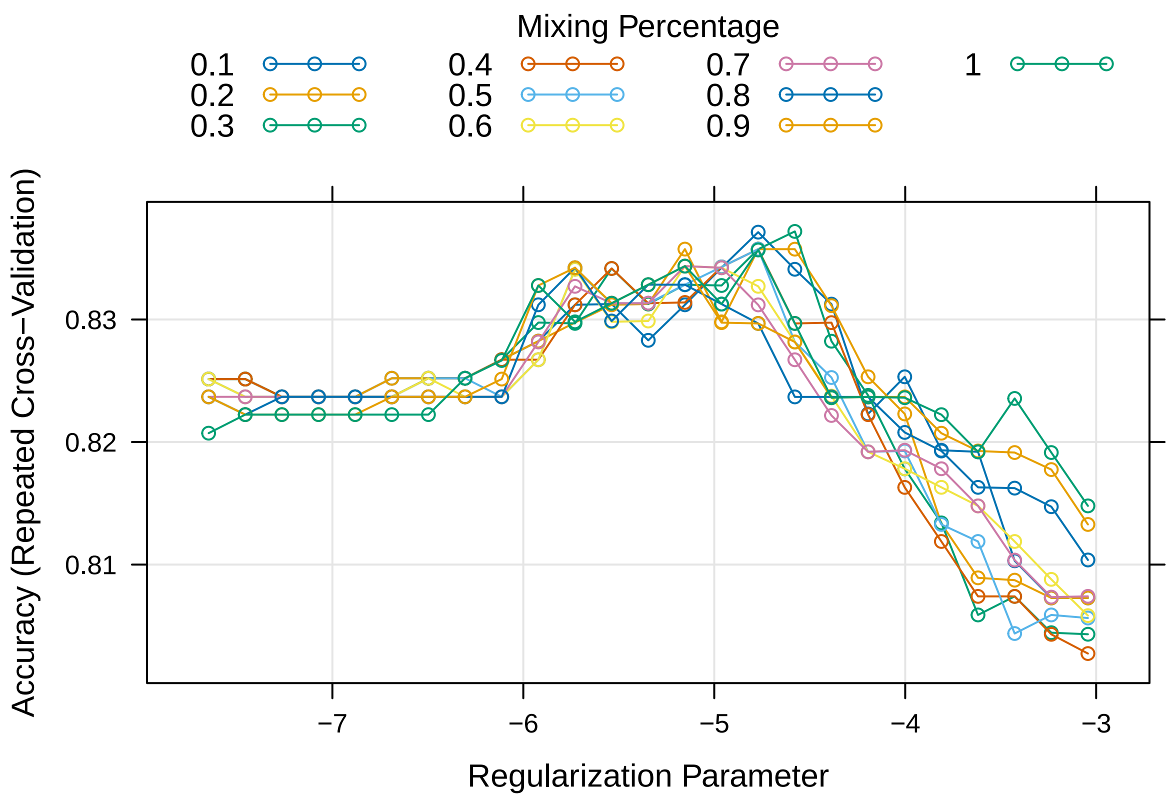
Train, assess, and tune the elastic net model with the custom tuning grid and assign the result to the `enet_tune` object. You should specify the arguments to `caret::train()` consistent with your solution in Problem 5b), except you should also assign `enet_grid` to the `tuneGrid` argument.

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



```
enet_tune$bestTune
```

```
##    alpha     lambda
## 67   0.3 0.01027463
```

As it can be seen above, the best tuning parameter values are alpha = 0.3 and lambda = 0.01027463. Since alpha is closer to 0, we conclude that the identified best elastic net model is more similar to Ridge regression.

5e)

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

```r
coef_enet_tune <- predict(enet_tune$finalModel, type = "coefficients", s = enet_tune$bestTune$lambda)
print(coef_enet_tune)
```

SOLUTION

```
## 18 x 1 sparse Matrix of class "dgCMatrix"
##                        s1
## (Intercept)  1.10820920
## x1B          0.41747015
## x1C          .
## x2          -0.34048219
## x3           .
## I(x2^2)      0.18667087
## I(x3^2)     -1.72475624
## x2:x3        0.05732699
## x1B:x2      -0.08156494
## x1C:x2      -1.18811574
## x1B:x3       0.03232367
## x1C:x3       0.29324198
## x1B:I(x2^2)  0.43908555
## x1C:I(x2^2)  .
## x1B:I(x3^2) -0.28249893
## x1C:I(x3^2) -0.13713286
## x1B:x2:x3    0.19009508
## x1C:x2:x3   -0.25635626
```

The coefficients are printed above. All the coefficients except “x1C”, “x3” and “x1C:I(x2^2)” are nonzero, which shows that the complex model H has been converted to a simpler model.

5f)

Let’s now visualize the predictions of the event probability from the tuned elastic net penalized logistic regression model. All `caret` trained models make predictions with a `predict()` function. The first argument is the `caret` trained object 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'`.

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

SOLUTION

5g)

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.

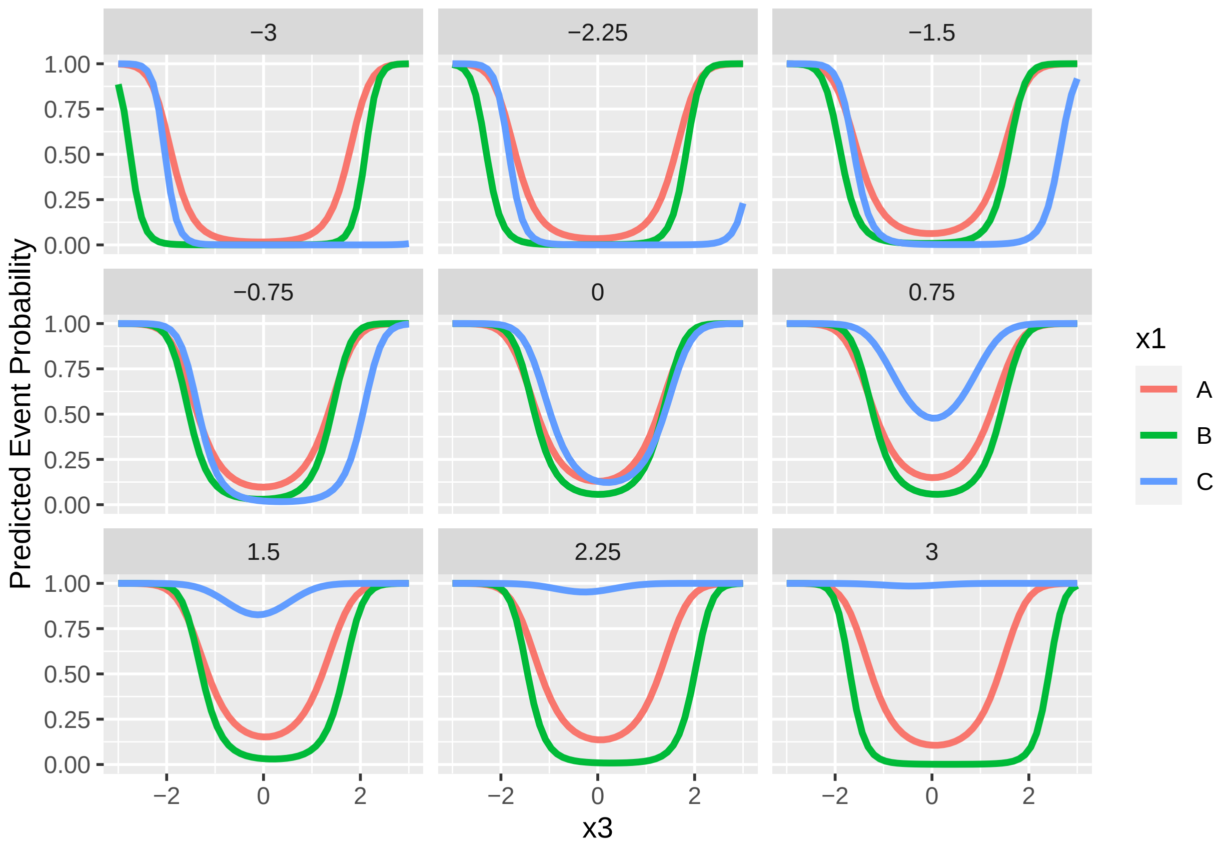
```r
viz_enet_df <- viz_grid %>% bind_cols(pred_viz_enet_probs)

viz_enet_df %>% glimpse()
```

```
## Rows: 1,377
## Columns: 5
## $ x1        <chr> "C", "B", "A", "C", "B", "A", "C", "B", "A", "C", "B", "A", ~
## $ x2        <dbl> -3.00, -3.00, -3.00, -2.25, -2.25, -2.25, -1.50, -1.50, -1.5~
## $ x3        <dbl> -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, ~
## $ event     <dbl> 0.9999022, 0.8864276, 0.9984171, 0.9999666, 0.9958424, 0.999~
## $ non_event <dbl> 9.779124e-05, 1.135724e-01, 1.582902e-03, 3.336859e-05, 4.15~
```

The glimpse reveals that the `event` column stores the **predicted event probability**. You must visualize the predicted event probability in a manner consistent with the `viz_bayes_logpost_preds()` function. The `caret` trained object does not return uncertainty estimates from the `glmnet` model and so you will not include uncertainty intervals as ribbons. You will visualize the predicted probability as a line (curve) with respect to `x3`, for each combination of `x2` and `x1`.

Pipe the `viz_enet_df` object to `ggplot()`. Map the `x` aesthetic to the `x3` variable and the `y` aesthetic to the `event` variable. Add a `geom_line()` layer and map the `color` aesthetic to the `x1` variable. Manually assign the size to 1.2. Create the facets by including the `facet_wrap()` function and specify the facets are "functions of" the `x2` input.

```r
viz_enet_df %>%
  ggplot(aes(x = x3, y = event, color = x1)) +
  geom_line(size = 1.2) +
  facet_wrap(~ x2) +
  labs(y = "Predicted Event Probability", x = "x3")
```



SOLUTION

5h)

Are the predicted trends from the tuned elastic net model consistent with the behavior visualized by the Bayesian models?

SOLUTION Yes, the predicted trends from the tuned elastic net model are consistent with the behavior visualized by the Bayesian models in Problem 4f).

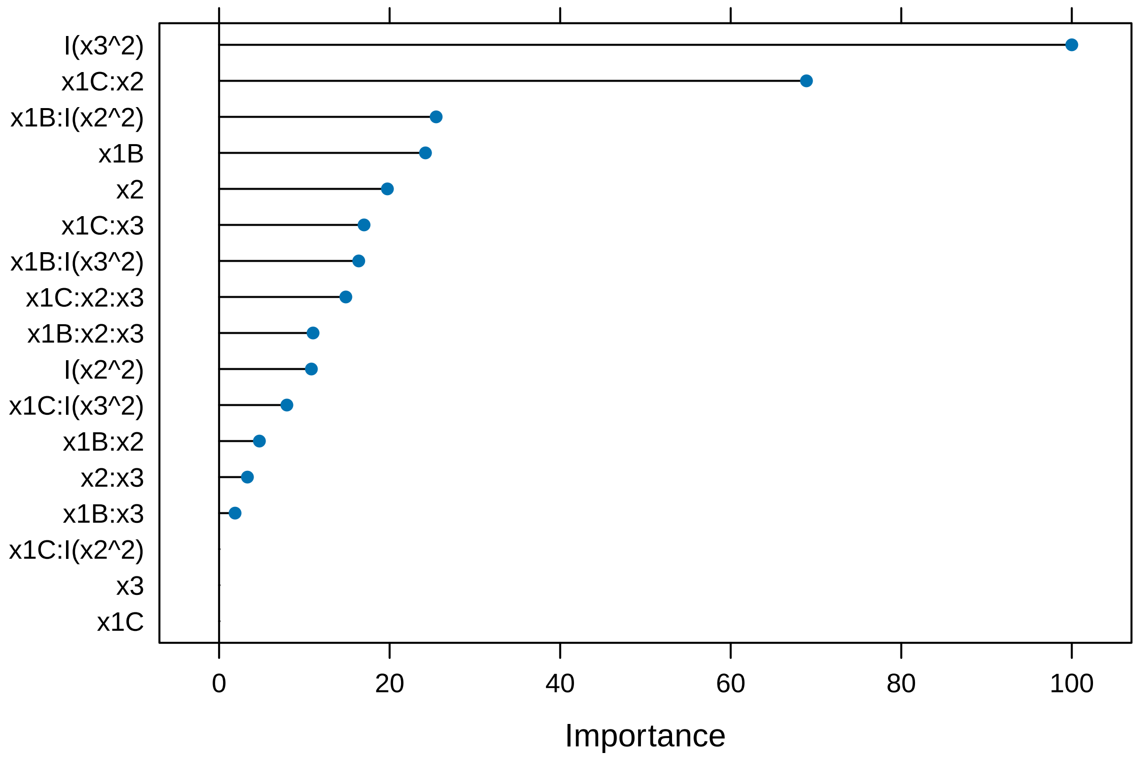
5i)

Use the caret `varImp()` function to generate the variable importances associated with the tuned elastic net model. Plot the variable importances via the default plot method.

What is the most important feature?

SOLUTION As it can be seen in the plot below, the most important feature is "(x3ˆ2)"

```r
enet_var_imp <- varImp(enet_tune)
plot(enet_var_imp)
```



Problem 06

Let's now train and tune several advanced methods. You will not program these methods from scratch in this assignment. Instead, you will use the `caret::train()` function to manage the preprocessing, training, and evaluation of the models via resampling.

You will use the default `caret` tuning grids associated with each of the models. The default tuning may not yield the best possible performance for these models. For example, small neural networks are trained in the default grid to make sure the run time is relatively fast. However, the point is for you to gain experience with the syntax associated with these models to support your work on the final project.

6a)

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 train a neural network to classify the binary outcome, `outcome`, with respect to all inputs. You should not interact inputs together. The formula should therefore "look" as if you are using linear additive features. The neural network will attempt to create non-linear relationships for you! 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 ~ x1 + x2 + x3,
                             data = df_caret,
                             method = "nnet",
                             metric = my_metric,
                             trControl = my_ctrl,
                             preProcess = c("center", "scale"),
                             trace = FALSE)
print(nnet_default)
```

```
## Neural Network
##
## 225 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: 202, 202, 202, 203, 202, 202, ...
## Resampling results across tuning parameters:
##
##   size  decay  Accuracy   Kappa
##   1     0e+00  0.6886913  0.1707863
##   1     1e-04  0.6855402  0.1692201
##   1     1e-01  0.7019379  0.2065714
##   3     0e+00  0.7585913  0.4005057
##   3     1e-04  0.7630105  0.4113777
##   3     1e-01  0.8014822  0.5105946
##   5     0e+00  0.7612374  0.4241012
##   5     1e-04  0.7746596  0.4677505
##   5     1e-01  0.8176932  0.5557965
##
## 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.
```

As indicated on the table above, the tuning parameter combinations which are considered to be the best are size = 5 and decay = 0.1.

6b)

Let's use predictions to understand the behavior of the neural network! Predictions are made consistent with the previously trained elastic net model because `caret` managed the training of the neural network. Thus, you will use syntax very similar to the syntax used to make predictions from the tuned elastic net model.

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

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

SOLUTION

6c)

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. 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.

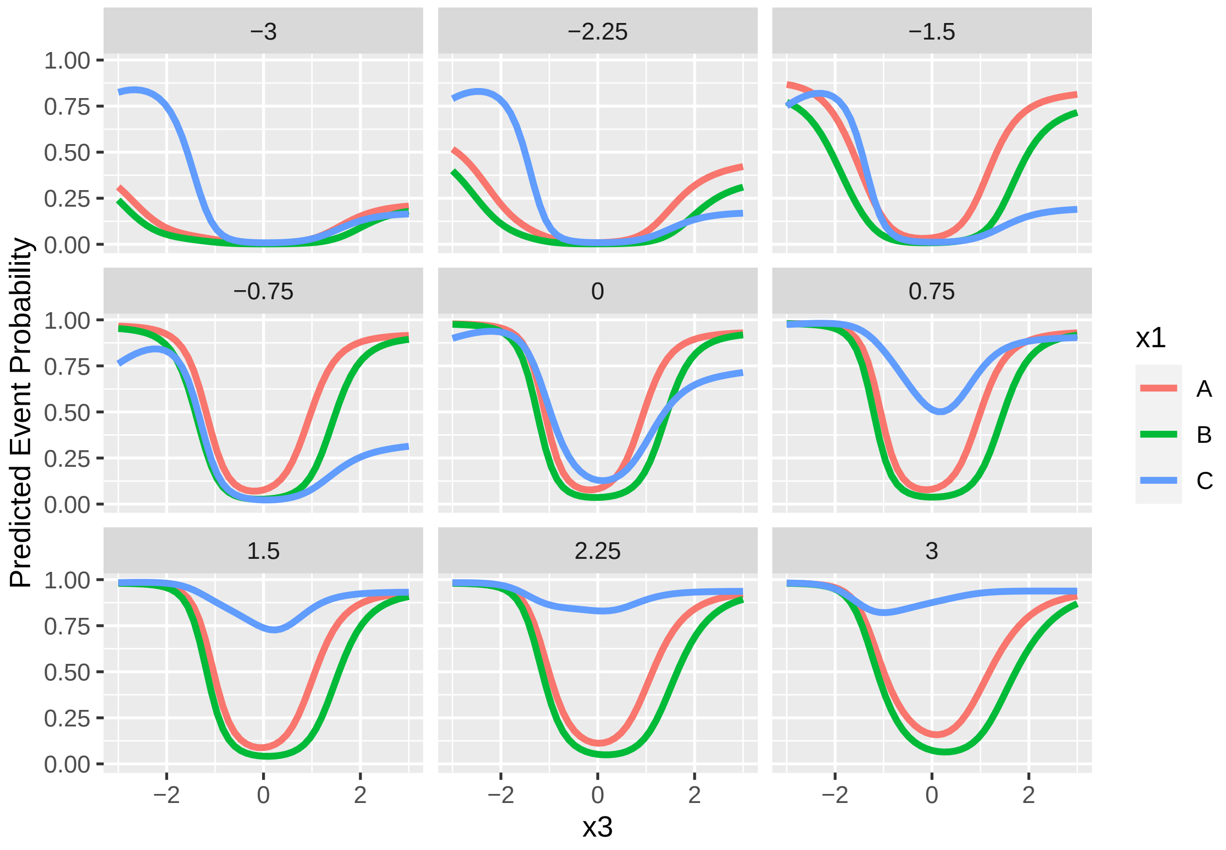
```
viz_nnet_df <- viz_grid %>% bind_cols(pred_viz_nnet_probs)

viz_nnet_df %>% glimpse()
```

```
## Rows: 1,377
## Columns: 5
## $ x1        <chr> "C", "B", "A", "C", "B", "A", "C", "B", "A", "C", "B", "A", ~
## $ x2        <dbl> -3.00, -3.00, -3.00, -2.25, -2.25, -2.25, -1.50, -1.50, -1.5~
## $ x3        <dbl> -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, ~
## $ event     <dbl> 0.8247354, 0.2403616, 0.3114790, 0.7902554, 0.3992162, 0.516~
## $ non_event <dbl> 0.17526457, 0.75963841, 0.68852102, 0.20974462, 0.60078381, ~
```

The glimpse reveals that the `event` column stores the **predicted event probability**. You must visualize the predicted event probability in a manner consistent with the `viz_bayes_logpost_preds()` function and the tuned elastic net model predictions. You will visualize the predicted probability as a line (curve) with respect to `x3`, for each combination of `x2` and `x1`.

Pipe the `viz_nnet_df` object to `ggplot()`. Map the `x` aesthetic to the `x3` variable and the `y` aesthetic to the `event` variable. Add a `geom_line()` layer and map the `color` aesthetic to the `x1` variable. Manually assign the size to 1.2. Create the facets by including the `facet_wrap()` function and specify the facets are "functions of" the `x2` input.

```
viz_nnet_df %>%
  ggplot(aes(x = x3, y = event, color = x1)) +
  geom_line(size = 1.2) +
  facet_wrap(~ x2) +
  labs(y = "Predicted Event Probability", x = "x3")
```



SOLUTION

6d)

Let's now a tree based method. 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 the case in lecture.

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 should not define any interactions or derive features from the inputs in the formula interface. The formula interface should "look" like linear additive features. 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!

```r
set.seed(1234)
rf_default <- caret::train(outcome ~ x1 + x2 + x3,
                           data = df_caret,
                           method = "rf",
                           metric = my_metric,
                           trControl = my_ctrl,
                           importance = TRUE)
print(rf_default)
```

```
## Random Forest
##
## 225 samples
##   3 predictor
##   2 classes: 'event', 'non_event'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 202, 202, 202, 203, 202, 202, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##   2     0.8074769  0.5274879
##   3     0.7925889  0.4992796
##   4     0.7927811  0.5032971
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 2.
```

6e)

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 thepredict()function to return the probabilities by settingtype = 'prob'`.

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

SOLUTION

6f)

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()
```

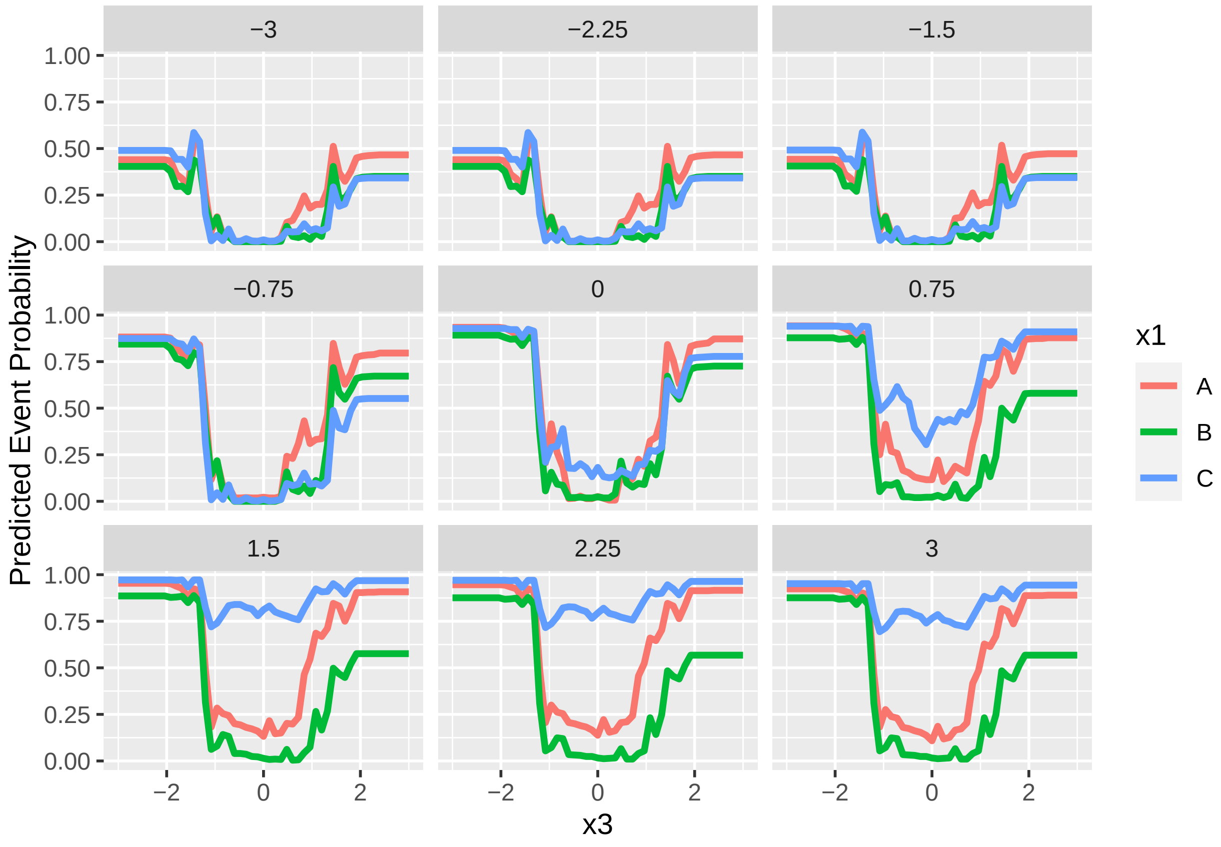
```
## Rows: 1,377
## Columns: 5
## $ x1        <chr> "C", "B", "A", "C", "B", "A", "C", "B", "A", "C", "B", "A", ~
## $ x2        <dbl> -3.00, -3.00, -3.00, -2.25, -2.25, -2.25, -1.50, -1.50, -1.5~
## $ x3        <dbl> -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, ~
## $ event     <dbl> 0.490, 0.404, 0.440, 0.490, 0.404, 0.440, 0.492, 0.406, 0.44~
## $ non_event <dbl> 0.510, 0.596, 0.560, 0.510, 0.596, 0.560, 0.508, 0.594, 0.55~
```

The glimpse reveals that the `event` column stores the **predicted event probability**. You must visualize the predicted event probability in the same fashion as you did in 6c).

Pipe the `viz_rf_df` object to `ggplot()`. Map the `x` aesthetic to the `x3` variable and the `y` aesthetic to the `event` variable. Add a `geom_line()` layer and map the `color` aesthetic to the

x1 variable. Manually assign the size to 1.2. Create the facets by including the `facet_wrap()` function and specify the facets are "functions of" the x2 input.

```r
viz_rf_df %>%
  ggplot(aes(x = x3, y = event, color = x1)) +
  geom_line(size = 1.2) +
  facet_wrap(~ x2) +
  labs(y = "Predicted Event Probability", x = "x3")
```



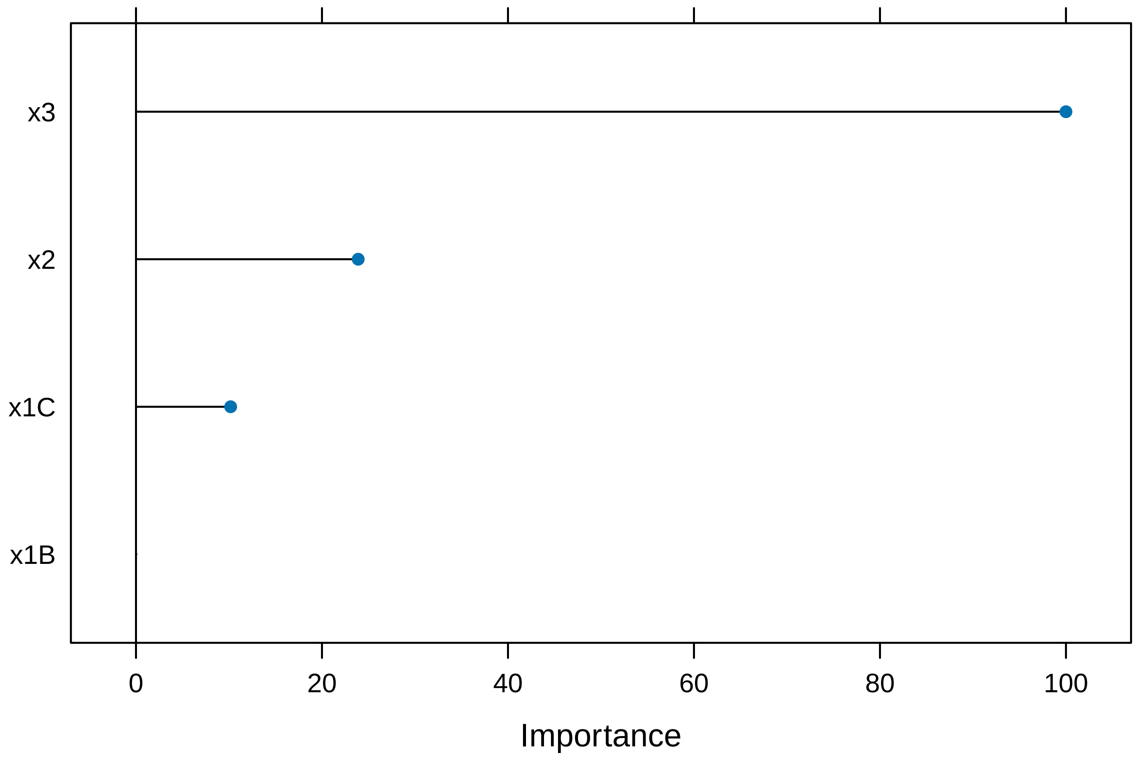
SOLUTION

6g)

You should have included `importance = TRUE` in the `caret::train()` call in 6d). 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?

```r
rf_var_imp <- varImp(rf_default)
plot(rf_var_imp)
```



SOLUTION

Problem 07

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

7a)

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

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

```
caret_acc_compare <- resamples(list(ENET_default = enet_default,
                                    ENET_tune = enet_tune,
                                    NNET_default = nnet_default,
                                    RF_default = rf_default))
```

SOLUTION

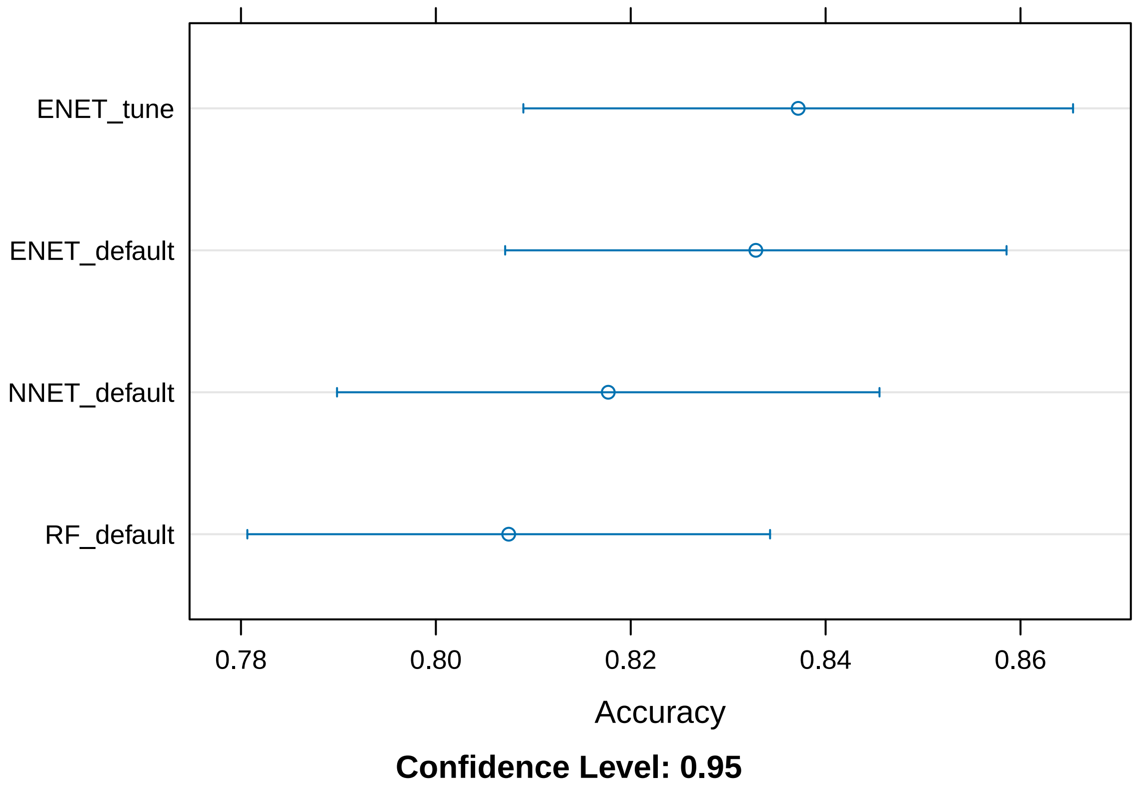
7b)

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

7c)

How would you describe the differences in the predictions between the 3 types of models you trained in this application?

SOLUTION

1) Elastic net offers a balance between interpretability and performance, especially useful when feature selection is important.
2) Neural Network, while powerful, it might lack interpretability and requires careful tuning to avoid overfitting.
3) Random Forest offers insights into feature importance, good for complex datasets with interactions, but can be less interpretable due to its ensemble nature.
4) The dotplot from the model comparison in Problem 7b) provides insights into which model had the highest average accuracy and the least variability in its predictions. Based on these results, we can conclude that "ENET_tune" gives the best performance in this case.