PPG Paint Colors: Final Project

Example: read data, save, and reload model object

Grishma Palkar

Overview

This RMarkdown shows how to read in the final project data. It also shows how to calculate the logit-transformed response and setup the binary outcome for use with `caret` or `tidymodels`. It also demonstrates how to fit a simple model (with `lm()`), save that model, and load it back into the workspace. You may find these actions helpful as you work through the project.

You must download the data from Canvas and save the data in the same directory as this RMarkdown file.

Load packages

This example uses the `tidyverse` suite of packages.

```
library(tidyverse)
```

```
## ── Attaching packages ─────────────────────────────────────── tidyverse 1.3.2 ──
## ✔ ggplot2 3.4.0      ✔ purrr   1.0.1
## ✔ tibble  3.2.1      ✔ dplyr   1.1.2
## ✔ tidyr   1.3.0      ✔ stringr 1.5.0
## ✔ readr   2.1.3      ✔ forcats 0.5.2
## ── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
## ✖ dplyr::filter() masks stats::filter()
## ✖ dplyr::lag()    masks stats::lag()
```

```
library(caret)
```

```
## Loading required package: lattice
##
## Attaching package: 'caret'
##
## The following object is masked from 'package:purrr':
##
##     lift
```

```
library(corrplot)
```

```
## corrplot 0.92 loaded
```

```
library(coefplot)
```

Read data

Please download the final project data from Canvas. If this Rmarkdown file is located in the same directory as the downloaded CSV file, it will be able to load in the data for you. It is **highly** recommended that you use an RStudio RProject to easily manage the working directory and file paths of the code and objects associated with the final project.

The code chunk below reads in the final project data.

```
df <- readr::read_csv("paint_project_train_data.csv", col_names = TRUE)
```

```
## Rows: 835 Columns: 8
## ── Column specification ────────────────────────────────────────────────────────
## Delimiter: ","
## chr (2): Lightness, Saturation
## dbl (6): R, G, B, Hue, response, outcome
##
## 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 `readr::read_csv()` function displays the data types and column names associated with the data. However, a glimpse is shown below that reveals the number of rows and also shows some of the representative values for the columns.

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

```
## Rows: 835
## Columns: 8
## $ R          <dbl> 172, 26, 172, 28, 170, 175, 90, 194, 171, 122, 0, 88, 144, …
## $ G          <dbl> 58, 88, 94, 87, 66, 89, 78, 106, 68, 151, 121, 140, 82, 163…
## $ B          <dbl> 62, 151, 58, 152, 58, 65, 136, 53, 107, 59, 88, 58, 132, 50…
## $ Lightness  <chr> "dark", "dark", "dark", "dark", "dark", "dark", "dark", "da…
## $ Saturation <chr> "bright", "bright", "bright", "bright", "bright", "bright",…
## $ Hue        <dbl> 4, 31, 8, 32, 5, 6, 34, 10, 1, 21, 24, 22, 36, 16, 26, 12, …
## $ response   <dbl> 12, 10, 16, 10, 11, 16, 10, 19, 14, 25, 14, 19, 14, 38, 15,…
## $ outcome    <dbl> 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,…
```

The data consist of continuous and categorical inputs. The `glimpse()` shown above reveals the data type for each variable which state to you whether the input is continuous or categorical. The RGB color model inputs, `R`, `G`, and `B` are continuous (dbl) inputs. The HSL color model inputs consist of 2 categorical inputs, `Lightness` and `Saturation`, and a continuous input, `Hue`. Two outputs are provided. The continuous output, `response`, and the Binary output, `outcome`. However, the data type of the Binary outcome is numeric because the Binary `outcome` is **encoded** as `outcome = 1` for the EVENT and `outcome = 0` for the NON-EVENT.

Binary classification task

The Binary output variable, `outcome`, is a numeric variable.

```
df %>% pull(outcome) %>% class()
```

```
## [1] "numeric"
```

However, there are **only** two unique values for `outcome`.

```
df %>% count(outcome)
```

```
## # A tibble: 2 × 2
##   outcome     n
##     <dbl> <int>
## 1       0   644
## 2       1   191
```

As stated previously, `outcome = 1` denotes the **EVENT** while `outcome = 0` denotes the **NON-EVENT**. Thus, the `outcome` variable uses the 0/1 encoding! This encoding is appropriate for `glm()` and the functions we create in homework assignments, and lecture examples. However, `caret` and `tidymodels` prefer a different encoding. For those reasons, two different binary classification data sets are defined. The first should be used for Parts iiiA) and iiiB) while the second should be used for iiiD).

The data set associated with iiiA) and iiiB) is created for you below. It removes the `response` variable so that way you can focus on the inputs and binary outcome.

```
dfiiiA <- df %>%
  select(-response)

dfiiiA %>% glimpse()
```

```
## Rows: 835
## Columns: 7
## $ R          <dbl> 172, 26, 172, 28, 170, 175, 90, 194, 171, 122, 0, 88, 144, …
## $ G          <dbl> 58, 88, 94, 87, 66, 89, 78, 106, 68, 151, 121, 140, 82, 163…
## $ B          <dbl> 62, 151, 58, 152, 58, 65, 136, 53, 107, 59, 88, 58, 132, 50…
## $ Lightness  <chr> "dark", "dark", "dark", "dark", "dark", "dark", "dark", "da…
## $ Saturation <chr> "bright", "bright", "bright", "bright", "bright", "bright",…
## $ Hue        <dbl> 4, 31, 8, 32, 5, 6, 34, 10, 1, 21, 24, 22, 36, 16, 26, 12, …
## $ outcome    <dbl> 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,…
```

The data set associated with iiiD) changes the data type of the `outcome` variable. The `ifelse()` function is used to convert `outcome` to a character data type. The value of `outcome = 1` is converted to the string `'event'` and the value of `outcome = 0` is converted to `'non_event'`. The `outcome` data type is then converted to a factor (R's categorical variable data type) with `'event'` forced as the first level.

```
dfiiiD <- df %>%
  select(-response) %>%
  mutate(outcome = ifelse(outcome == 1, 'event', 'non_event'),
         outcome = factor(outcome, levels = c('event', 'non_event')))

dfiiiD %>% glimpse()
```

```
## Rows: 835
## Columns: 7
## $ R          <dbl> 172, 26, 172, 28, 170, 175, 90, 194, 171, 122, 0, 88, 144, …
## $ G          <dbl> 58, 88, 94, 87, 66, 89, 78, 106, 68, 151, 121, 140, 82, 163…
## $ B          <dbl> 62, 151, 58, 152, 58, 65, 136, 53, 107, 59, 88, 58, 132, 50…
## $ Lightness  <chr> "dark", "dark", "dark", "dark", "dark", "dark", "dark", "da…
## $ Saturation <chr> "bright", "bright", "bright", "bright", "bright", "bright",…
## $ Hue        <dbl> 4, 31, 8, 32, 5, 6, 34, 10, 1, 21, 24, 22, 36, 16, 26, 12, …
## $ outcome    <fct> event, event, event, non_event, non_event, non_event, non_e…
```

By converting `outcome` to a factor, the unique values of the variables are “always known”:

```
dfiiiD %>% pull(outcome) %>% levels()
```

```
## [1] "event"     "non_event"
```

However, the value counts are the same as the original encoding.

```
dfiiiD %>% count(outcome)
```

```
## # A tibble: 2 × 2
##   outcome       n
##   <fct>     <int>
## 1 event       191
## 2 non_event   644
```

```
dfiiiA_train <- dfiiiA %>%
  select(R,G,B,Hue,outcome) %>%
  scale() %>% as.data.frame() %>%
  bind_cols(dfiiiA %>% select(Lightness,Saturation))

dfiiiA_train %>% glimpse()
```

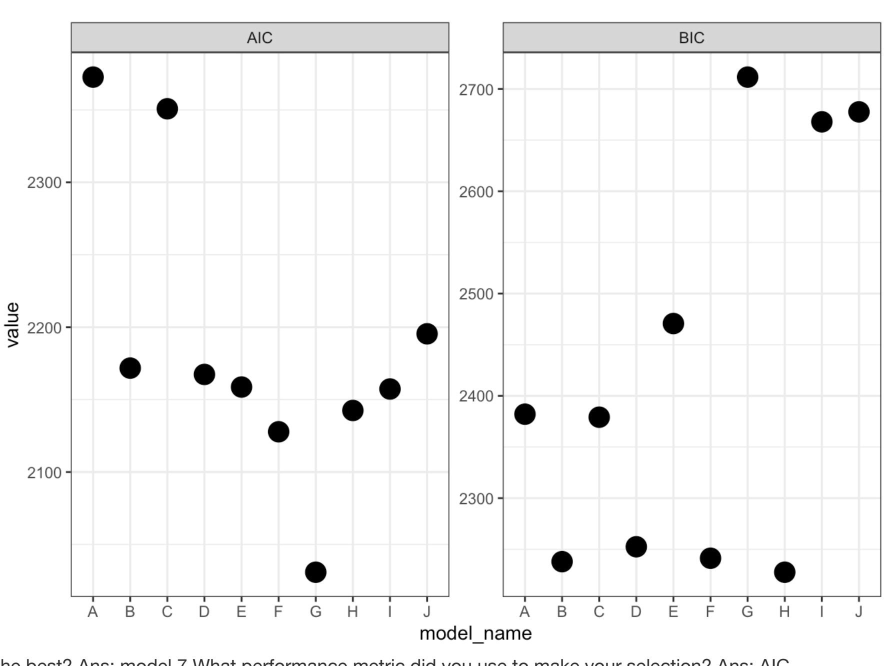
```
## Rows: 835
## Columns: 7
## $ R          <dbl> -0.19790120, -2.74419521, -0.19790120, -2.70931447, -0.2327…
## $ G          <dbl> -2.3619736, -1.7631189, -1.6433480, -1.7830807, -2.2022790,…
## $ B          <dbl> -1.7994266, -0.1706872, -1.8726283, -0.1523868, -1.8726283,…
## $ Hue        <dbl> -1.3548215, 1.3198239, -0.9585777, 1.4188848, -1.2557605, -…
## $ outcome    <dbl> 1.8351266, 1.8351266, 1.8351266, -0.5442689, -0.5442689, -0…
## $ Lightness  <chr> "dark", "dark", "dark", "dark", "dark", "dark", "dark", "da…
## $ Saturation <chr> "bright", "bright", "bright", "bright", "bright", "bright",…
```

(1) You must therefore train 10 different models!

```
# Model 1: Intercept-only model
mod01_glm <- glm(outcome ~ 1, data = dfiiiA_train)
mod01_glm %>% readr::write_rds("model01_glm.rds")

# Model 2:Categorical variables only
mod02_glm <- glm(outcome~(Lightness + Saturation ), data = dfiiiA_train)
mod02_glm %>% readr::write_rds("model02_glm.rds")

# Model 3:Continuous variables only
mod03_glm <- glm(outcome~(R+G+B+Hue), data = dfiiiA_train)
mod03_glm %>% readr::write_rds("model03_glm.rds")

# Model 4:All categorical and continuous variables – linear additive
mod04_glm <- glm(outcome~(R+G+B+Lightness + Saturation + Hue), data = dfiiiA_train)
mod04_glm %>% readr::write_rds("model04_glm.rds")

# Model 5:Interaction of the categorical inputs with all continuous inputs main effects
mod05_glm <- glm(outcome~(R+G+B+Hue)*(Lightness + Saturation), data = dfiiiA_train)
mod05_glm %>% readr::write_rds("model05_glm.rds")

# Model 6: Add categorical inputs to all main effect and all pairwise interactions of continuous inputs
mod06_glm <- glm(outcome ~ (R+G+B+Hue)*(R+G+B+Hue)+(Lightness + Saturation ), data = dfiiiA_train)
mod06_glm %>% readr::write_rds("model06_glm.rds")

# Model 7 : Interaction of the categorical inputs with all main effect and all pairwise interactions of co
ntinuous inputs
mod07_glm <- glm(outcome ~ (R+G+B+Hue)*(R+G+B+Hue)*(Lightness+Saturation) , data = dfiiiA_train)
mod07_glm %>% readr::write_rds("model07_glm.rds")

#  3 models with basis functions of your choice
# Model 8:Try non-linear basis functions based on your EDA.
mod08_glm <- glm(outcome ~ I(R^2) + I(G^2) + I(B^2) + I(Hue^2) + Lightness + Saturation, data = dfiiiA_tra
in)
mod08_glm %>% readr::write_rds("model08_glm.rds")

# Model 9:Can consider interactions of basis functions with other basis functions!
mod09_glm <- lm(outcome ~ (R+G+B+Lightness + Saturation + Hue)*(R+G+B+Lightness + Saturation + Hue), data
= dfiiiA_train)
mod09_glm %>% readr::write_rds("model09_glm.rds")

# Model 10:Can consider interactions of basis functions with the categorical inputs!
mod10_glm <- glm(outcome ~ (R+G+B+Lightness + Saturation + Hue)*(Lightness+Saturation), data = dfiiiA_trai
n)
mod10_glm %>% readr::write_rds("model10_glm.rds")
```

Did you experience any issues or warnings while fitting the generalized linear models? Ans: No.

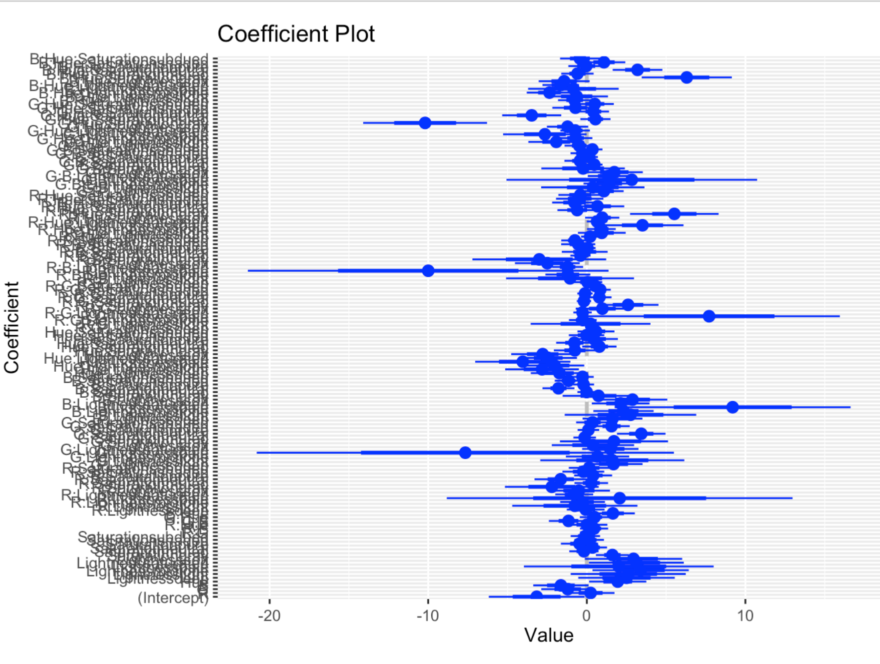
```
extract_metrics <- function(mod_object, mod_name)
{
  broom::glance(mod_object) %>%
    mutate(model_name = mod_name)
}

glm_mle_results <- purrr::map2_dfr(list(mod01_glm,mod02_glm,mod03_glm,mod04_glm,mod05_glm,mod06_glm,mod07_
glm,mod08_glm,mod09_glm,mod10_glm),
                                   LETTERS[1:10],extract_metrics)


glm_mle_results %>%
  select(model_name, AIC, BIC) %>%
  pivot_longer(c(AIC, BIC)) %>%
  ggplot(mapping = aes(x = model_name, y = value)) +
  geom_point(size = 5) +
  facet_wrap(~name, scales = 'free_y') +
  theme_bw()
```



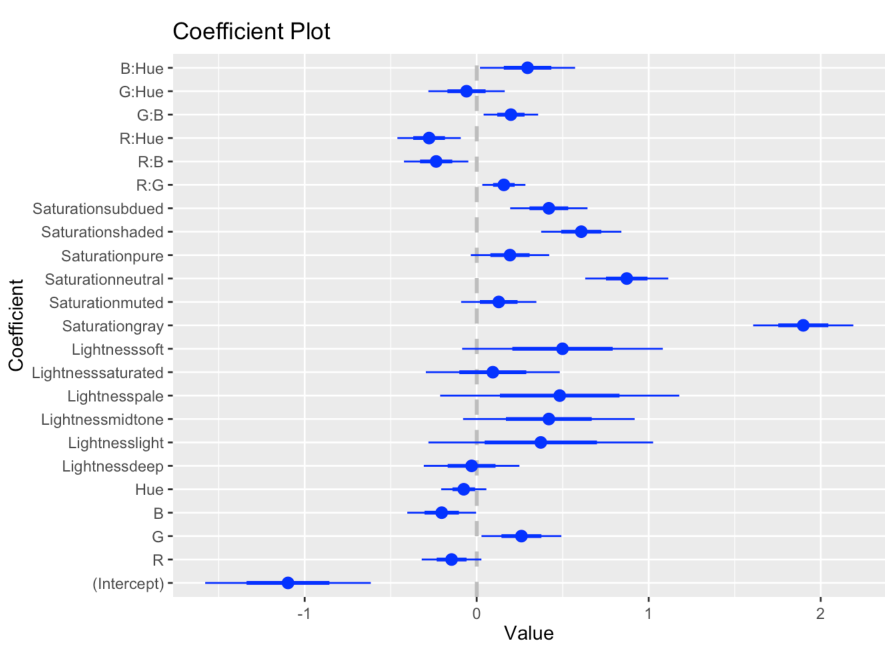
Which of the 10 models is the best? Ans: model 7 What performance metric did you use to make your selection? Ans: AIC

Visualize the coefficient summaries for your top 3 models.

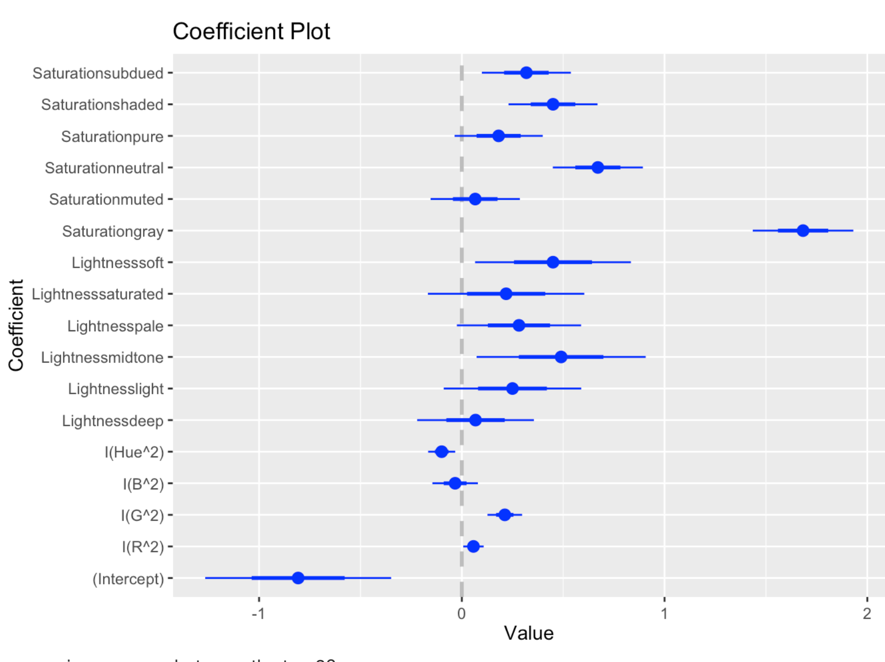
```
coefplot(mod07_glm)
```




```
coefplot(mod06_glm)
```




```
coefplot(mod08_glm)
```



How do the coefficient summaries compare between the top 3?

```
# Model 7
mod07_glm %>% broom::tidy() %>% filter(p.value<0.05) %>% arrange(desc(abs(estimate)))
```

```
## # A tibble: 38 × 5
##    term                     estimate std.error statistic     p.value
##    <chr>                       <dbl>     <dbl>     <dbl>       <dbl>
##  1 G:Hue:Saturationgray       -10.2      1.95      -5.22 0.000000234
##  2 B:Lightnesspale              9.19     3.72       2.47 0.0137
##  3 B:Hue:Saturationgray         6.30     1.42       4.44 0.0000107
##  4 R:Hue:Saturationgray         5.52     1.40       3.95 0.0000851
##  5 Hue:Lightnesspale           -4.04     1.50      -2.69 0.00730
##  6 R:Hue:Lightnesspale          3.51     1.29       2.71 0.00686
##  7 G:Hue:Saturationneutral     -3.48     0.931     -3.73 0.000206
##  8 G:Saturationneutral          3.43     0.765      4.49 0.00000852
##  9 Lightnessmidtone             3.39     1.52       2.24 0.0257
## 10 B:Hue:Saturationneutral      3.20     0.783      4.08 0.0000496
## # ℹ 28 more rows
```

```
# Model 6
mod06_glm %>% broom::tidy() %>% filter(p.value<0.05) %>% arrange(desc(abs(estimate)))
```

```
## # A tibble: 12 × 5
##    term              estimate std.error statistic  p.value
##    <chr>                <dbl>     <dbl>     <dbl>    <dbl>
##  1 Saturationgray       1.90     0.146      13.0  1.97e-35
##  2 (Intercept)         -1.10     0.241      -4.56 5.98e- 6
##  3 Saturationneutral    0.872    0.121       7.23 1.15e-12
##  4 Saturationshaded     0.608    0.117       5.21 2.34e- 7
##  5 Saturationsubdued    0.419    0.112       3.73 2.05e- 4
##  6 B:Hue                0.296    0.138       2.14 3.27e- 2
##  7 R:Hue               -0.277    0.0921     -3.01 2.73e- 3
##  8 G                    0.260    0.116       2.24 2.54e- 2
##  9 R:B                 -0.236    0.0935     -2.52 1.19e- 2
## 10 B                   -0.204    0.100      -2.04 4.21e- 2
## 11 G:B                  0.199    0.0793      2.51 1.24e- 2
## 12 R:G                  0.158    0.0627      2.52 1.21e- 2
```

```
# Model 8
mod08_glm %>% broom::tidy() %>% filter(p.value<0.05) %>% arrange(desc(abs(estimate)))
```

```
## # A tibble: 10 × 5
##    term              estimate std.error statistic  p.value
##    <chr>                <dbl>     <dbl>     <dbl>    <dbl>
##  1 Saturationgray      1.68      0.124      13.6  5.42e-38
##  2 (Intercept)        -0.807     0.229      -3.52 4.54e- 4
##  3 Saturationneutral   0.671     0.111       6.04 2.36e- 9
##  4 Lightnessmidtone    0.490     0.209       2.35 1.91e- 2
##  5 Saturationshaded    0.450     0.110       4.09 4.64e- 5
##  6 Lightnesssoft       0.450     0.192       2.34 1.95e- 2
##  7 Saturationsubdued   0.318     0.110       2.90 3.81e- 3
##  8 I(G^2)              0.212     0.0427      4.96 8.42e- 7
##  9 I(Hue^2)           -0.0994    0.0333     -2.98 2.96e- 3
## 10 I(R^2)              0.0574    0.0252      2.27 2.32e- 2
```

Which inputs seem important?

Ans: So all the inputs continuous and categorical are important