Exercise 15

Marc Dotson

Return one last time to soup\_data and the models from the previous exercises.

1. Create a discrete outcome variable top\_selling based on Sales > 375000 using if\_else() and factor().
2. Preprocess the data using initial\_time\_split() with just 75% of the data in training and fit the same models again using the preprocessed training data, now as logistic regressions with the new top\_selling variable as the outcome variable. You can ignore any warnings when fitting each model.
3. Compute the accuracy using the preprocessed testing data. Is the same model as last time the best-fitting model again?
4. Compute a confusion matrix for the best-fitting model and accurately interpret it.
5. Render the Quarto document into Word and upload to Canvas.

**Five points total, one point each for:**

* **Creating top\_selling as specified.**
* **Preprocessing the data and fitting logistic regressions with top\_selling as the outcome.**
* **Computing the accuracy using the preprocessed testing data and answering the question about how the best-fitting model compares to the previous exercise.**
* **Computing a confusion matrix and interpreting it.**
* **One point for submitting a rendered Word document.**

## Create a Discrete Outcome Variable

Let’s load the packages we’ll need, import the data, and create a discrete outcome variable as specified.

# Load packages.  
library(tidyverse)

── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──  
✔ dplyr 1.1.4 ✔ readr 2.1.4  
✔ forcats 1.0.0 ✔ stringr 1.5.1  
✔ ggplot2 3.4.4 ✔ tibble 3.2.1  
✔ lubridate 1.9.3 ✔ tidyr 1.3.0  
✔ purrr 1.0.2   
── Conflicts ────────────────────────────────────────── tidyverse\_conflicts() ──  
✖ dplyr::filter() masks stats::filter()  
✖ dplyr::lag() masks stats::lag()  
ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors

library(tidymodels)

── Attaching packages ────────────────────────────────────── tidymodels 1.1.1 ──  
✔ broom 1.0.5 ✔ rsample 1.2.0  
✔ dials 1.2.0 ✔ tune 1.1.2  
✔ infer 1.0.5 ✔ workflows 1.1.3  
✔ modeldata 1.2.0 ✔ workflowsets 1.0.1  
✔ parsnip 1.1.1 ✔ yardstick 1.2.0  
✔ recipes 1.0.9   
── Conflicts ───────────────────────────────────────── tidymodels\_conflicts() ──  
✖ scales::discard() masks purrr::discard()  
✖ dplyr::filter() masks stats::filter()  
✖ recipes::fixed() masks stringr::fixed()  
✖ dplyr::lag() masks stats::lag()  
✖ yardstick::spec() masks readr::spec()  
✖ recipes::step() masks stats::step()  
• Use suppressPackageStartupMessages() to eliminate package startup messages

# Import, filter, and mutate data.  
soup\_data <- read\_csv(here::here("Data", "soup\_data.csv")) |>   
 filter(Retailer\_Trade\_Areas == "WEST CENSUS TA", Brand\_High == "CAMPBELL'S") |>  
 mutate(top\_selling = if\_else(Sales > 375000, 1, 0) |> factor())

Rows: 3192 Columns: 66  
── Column specification ────────────────────────────────────────────────────────  
Delimiter: ","  
chr (5): Four\_Weeks\_Periods, Retailer\_Trade\_Areas, Category, Sub\_Category, ...  
dbl (61): Sales, Base\_Sales, Incr\_Sales, Units, Base\_Units, Incr\_Units, Perc...  
  
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.

Clearly we need to preprocess these variables by applying the log transform.

## Preprocess Data

Let’s split the data, prepare a recipe using the training data, and apply it to both the training and testing data.

# Split the data.  
soup\_split <- initial\_time\_split(soup\_data, prop = 0.75)  
  
# Prepare a recipe.  
soup\_recipe <- training(soup\_split) |>   
 recipe(top\_selling ~ Any\_Disp\_Spend + Any\_Feat\_Spend + Any\_Price\_Decr\_Spend) |>   
 step\_log(all\_numeric(), offset = 1) |>   
 prep()  
  
# Apply the recipe to the training data.  
soup\_training <- soup\_recipe |>  
 bake(training(soup\_split))  
  
# Apply the recipe to the testing data.  
soup\_testing <- soup\_recipe |>  
 bake(testing(soup\_split))

## Fit the Models

Now let’s refit the models we ran previously, now using the preprocessed training data and top\_selling as the outcome.

# Full model.  
fit\_01 <- logistic\_reg() |>   
 set\_engine(engine = "glm") |>   
 fit(  
 top\_selling ~ Any\_Disp\_Spend + Any\_Feat\_Spend + Any\_Price\_Decr\_Spend,   
 data = soup\_training  
 )

Warning: glm.fit: algorithm did not converge

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

# Model without display spend.  
fit\_02 <- logistic\_reg() |>   
 set\_engine(engine = "glm") |>   
 fit(  
 top\_selling ~ Any\_Feat\_Spend + Any\_Price\_Decr\_Spend,   
 data = soup\_training  
 )

Warning: glm.fit: algorithm did not converge  
  
Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

# Model without feature spend.  
fit\_03 <- logistic\_reg() |>   
 set\_engine(engine = "glm") |>   
 fit(  
 top\_selling ~ Any\_Disp\_Spend + Any\_Price\_Decr\_Spend,   
 data = soup\_training  
 )

Warning: glm.fit: algorithm did not converge  
  
Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

# Model without price decrease spend.  
fit\_04 <- logistic\_reg() |>   
 set\_engine(engine = "glm") |>   
 fit(  
 top\_selling ~ Any\_Disp\_Spend + Any\_Feat\_Spend,   
 data = soup\_training  
 )

Warning: glm.fit: algorithm did not converge  
  
Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Compute Predictive Fit

Now let’s compute and compare accuracy using the preprocessed testing data.

# Compute accuracy.  
accuracy\_01 <- fit\_01 |>   
 predict(new\_data = soup\_testing) |>  
 bind\_cols(soup\_testing) |>  
 accuracy(truth = top\_selling, estimate = .pred\_class)  
  
accuracy\_02 <- fit\_02 |>   
 predict(new\_data = soup\_testing) |>  
 bind\_cols(soup\_testing) |>  
 accuracy(truth = top\_selling, estimate = .pred\_class)  
  
accuracy\_03 <- fit\_03 |>   
 predict(new\_data = soup\_testing) |>  
 bind\_cols(soup\_testing) |>  
 accuracy(truth = top\_selling, estimate = .pred\_class)  
  
accuracy\_04 <- fit\_04 |>   
 predict(new\_data = soup\_testing) |>  
 bind\_cols(soup\_testing) |>  
 accuracy(truth = top\_selling, estimate = .pred\_class)  
  
# Compare accuracy.  
tibble(  
 model = c(  
 "Full model",   
 "Model without display spend",   
 "Model without feature spend",   
 "Model without price decrease spend"  
 )  
) |>   
 bind\_cols(  
 bind\_rows(  
 accuracy\_01,  
 accuracy\_02,  
 accuracy\_03,  
 accuracy\_04  
 )  
 ) |>   
 arrange(desc(.estimate))

# A tibble: 4 × 4  
 model .metric .estimator .estimate  
 <chr> <chr> <chr> <dbl>  
1 Full model accuracy binary 1   
2 Model without feature spend accuracy binary 1   
3 Model without display spend accuracy binary 0.923  
4 Model without price decrease spend accuracy binary 0.923

Based on accuracy, the best-fitting model is the full model or the model without feature spend. When it was Sales as the outcome, the model without price decrease spend was the best-fitting model.

## Create a Confusion Matrix

I’ll use the full model to create a confusion matrix.

fit\_01 |>   
 predict(new\_data = soup\_testing) |>  
 bind\_cols(soup\_testing) |>  
 conf\_mat(truth = top\_selling, estimate = .pred\_class)

Truth  
Prediction 0 1  
 0 10 0  
 1 0 3

This confusion matrix shows that we have *perfect* classification with this model. The diagonal shows the correct predictions. The off-diagonal shows where we’ve predicted incorrectly – which here is 0.