Unit 3 Lecture 1: Logistic Regression

October 4, 2022

```
library(tidyverse)
library(splines)
library(cowplot)
library(stat471)
```

Linear regression

In the context of splines, we worked with `age` and `income` from the income data. In fact, these data have more columns as well, which we might want to use for predicting income:

```
income_data <- read_tsv("income_data.tsv")
income_data
```

```
## # A tibble: 3,000 x 8
##    income  year   age maritl           race     education       jobclass  health
##     <dbl> <dbl> <dbl> <chr>            <chr>    <chr>           <chr>     <chr>
##  1   75.0  2006    18 1. Never Married 1. White 1. < HS Grad    1. Indus~ 1. <=~
##  2   70.5  2004    24 1. Never Married 1. White 4. College Grad 2. Infor~ 2. >=~
##  3  131.   2003    45 2. Married       1. White 3. Some College 1. Indus~ 1. <=~
##  4  155.   2003    43 2. Married       3. Asian 4. College Grad 2. Infor~ 2. >=~
##  5   75.0  2005    50 4. Divorced      1. White 2. HS Grad      2. Infor~ 1. <=~
##  6  127.   2008    54 2. Married       1. White 4. College Grad 2. Infor~ 2. >=~
##  7  170.   2009    44 2. Married       4. Other 3. Some College 1. Indus~ 2. >=~
##  8  112.   2008    30 1. Never Married 3. Asian 3. Some College 2. Infor~ 1. <=~
##  9  119.   2006    41 1. Never Married 2. Black 3. Some College 2. Infor~ 2. >=~
## 10  129.   2004    52 2. Married       1. White 2. HS Grad      2. Infor~ 2. >=~
## # ... with 2,990 more rows
## # i Use `print(n = ...)` to see more rows
```

Let's split our data into train and test:

```
set.seed(4710)
train_samples <- sample(1:nrow(income_data), 0.8 * nrow(income_data))
income_train <- income_data %>% filter(row_number() %in% train_samples)
income_test <- income_data %>% filter(!(row_number() %in% train_samples))
```

We can run a linear regression using `lm()`, which we have already used for spline fits:

```
lm(income ~ age, data = income_train)
```

```
##
## Call:
## lm(formula = income ~ age, data = income_train)
##
## Coefficients:
## (Intercept)          age
##     81.9531       0.7225
```

We can specify multiple predictors using the + syntax:

```
lm(income ~ age + education, data = income_train)
```

```
##
## Call:
## lm(formula = income ~ age + education, data = income_train)
##
## Coefficients:
##                 (Intercept)                          age
##                     60.8669                       0.5683
##         education2. HS Grad     education3. Some College
##                     11.2193                      24.5819
##    education4. College Grad  education5. Advanced Degree
##                     40.3823                      64.8561
```

We can include all predictors except for the response using the . syntax:

```
lm(income ~ ., data = income_train)
```

```
##
## Call:
## lm(formula = income ~ ., data = income_train)
##
## Coefficients:
##                 (Intercept)                         year
##                  -2550.4114                       1.2973
##                         age             maritl2. Married
##                      0.3631                      17.7661
##            maritl3. Widowed            maritl4. Divorced
##                      4.1442                       4.7896
##          maritl5. Separated                 race2. Black
##                     13.0303                      -5.5724
##                race3. Asian                 race4. Other
##                     -3.4709                      -3.8735
##         education2. HS Grad     education3. Some College
##                     10.7993                      23.3761
##    education4. College Grad  education5. Advanced Degree
##                     37.0926                      59.2820
##      jobclass2. Information         health2. >=Very Good
##                      5.0515                       7.3695
```

How do we interpret the coefficient for `age`? What about for `education2. HS Grad`?

We can make predictions for a given linear regression model using `predict()`:

```
lm_fit <- lm(income ~ ., data = income_train) # first save the model fit
predictions <- predict(lm_fit, newdata = income_test)
```

We can then assess test error metrics like RMSE based on these predictions:

```
sqrt(mean((income_test$income - predictions)^2))
```

```
## [1] 33.12452
```

In Unit 2, we found that, if modeling `income` as a spline function of `age`, the best choice of degrees of freedom is around 3. How does the RMSE of this model compare to that of the model above using all of the variables?

Logistic regression

Now, let's move on to logistic regression. Recall the default data:

```
default_data <- read_tsv("default_data.tsv")
default_data
```

```
## # A tibble: 10,000 x 4
##    default student balance income
##    <chr>   <chr>     <dbl>  <dbl>
##  1 No      No         730. 44362.
##  2 No      Yes        817. 12106.
##  3 No      No        1074. 31767.
##  4 No      No         529. 35704.
##  5 No      No         786. 38463.
##  6 No      Yes        920.  7492.
##  7 No      No         826. 24905.
##  8 No      Yes        809. 17600.
##  9 No      No        1161. 37469.
## 10 No      No           0  29275.
## # ... with 9,990 more rows
## # i Use `print(n = ...)` to see more rows
```

The rest of the activity will be easier if we code `default` as 0-1:

```
default_data <- default_data %>% mutate(default = as.numeric(default == "Yes"))
default_data
```

```
## # A tibble: 10,000 x 4
##    default student balance income
##      <dbl> <chr>     <dbl>  <dbl>
##  1       0 No         730. 44362.
##  2       0 Yes        817. 12106.
##  3       0 No        1074. 31767.
##  4       0 No         529. 35704.
##  5       0 No         786. 38463.
##  6       0 Yes        920.  7492.
##  7       0 No         826. 24905.
##  8       0 Yes        809. 17600.
##  9       0 No        1161. 37469.
## 10       0 No           0  29275.
## # ... with 9,990 more rows
## # i Use `print(n = ...)` to see more rows
```

Let's split the default data into training and test sets:

```
set.seed(4710)
train_samples <- sample(1:nrow(default_data), 0.8 * nrow(default_data))
default_train <- default_data %>% filter(row_number() %in% train_samples)
default_test <- default_data %>% filter(!(row_number() %in% train_samples))
```

Running a logistic regression

The way to run a logistic regression is through the `glm` function:

```
glm_fit <- glm(default ~ student + balance + income,
  family = "binomial",
  data = default_train
)
coef(glm_fit)
```

```
##   (Intercept)    studentYes       balance        income 
## -1.088496e+01 -5.701843e-01  5.635043e-03  7.560014e-06
```

- What is the coefficient estimate for `student`?
- Does this suggest that being a student increases or decreases the probability of default, other things being equal?
- According to this estimate, how does being a student impact the log-odds of default? How does it impact the odds of default?

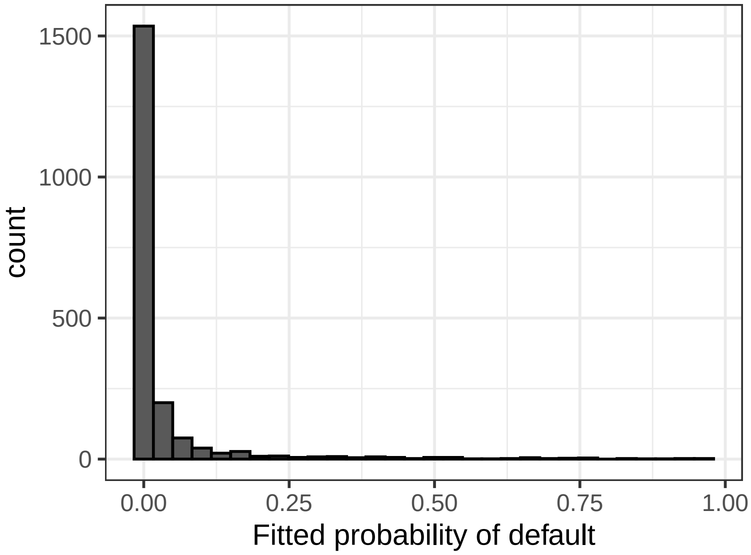
Fitted probabilities and making predictions

We can extract the fitted probabilities of default for a test set using the `predict` function:

```
fitted_probabilities <- predict(glm_fit,
  newdata = default_test,
  type = "response" # to get output on probability scale
)
head(fitted_probabilities)
```

```
##            1            2            3            4            5            6 
## 0.0099981865 0.0019920721 0.0112420255 0.0117355375 0.0001321415 0.0051567773
```

```
tibble(fitted_probabilities) %>%
  ggplot(aes(x = fitted_probabilities)) +
  geom_histogram(color = "black") +
  labs(x = "Fitted probability of default")
```



We can now make predictions based on the fitted probabilities using the standard 0.5 threshold:

```
predictions <- as.numeric(fitted_probabilities > 0.5)
head(predictions)
```

```
## [1] 0 0 0 0 0 0
```

Evaluating the classifier

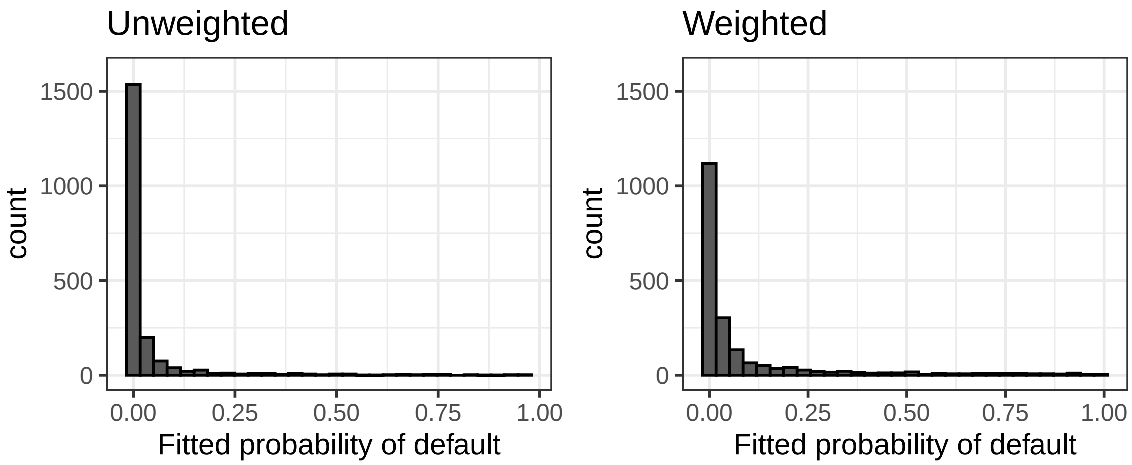
Let's evaluate the performance of the above logistic regression classifier on the test set rate. We can use the `classification_metrics()` function from the `stat471` package.

```
classification_metrics(
  test_responses = default_test$default,
  test_predictions = predictions
)
```

```
## # A tibble: 1 x 5
##   misclass_err w_misclass_err   TPR   TNR     F
##          <dbl> <lgl>          <dbl> <dbl> <dbl>
## 1        0.028 NA             0.370 0.995 0.539
```

If we want the classifier to pay more attention to the positive class, we'll need to upweight it. Fortunately, logistic regression accommodates observation weights, via the `weights` argument to `glm()`. Let's upweight the positive class by a factor of 5:

```
# fit the weighted GLM
glm_fit_weighted <- glm(default ~ student + balance + income,
  family = "binomial",
  weights = 5 * (default_train$default == 1) + 1 * (default_train$default == 0),
  data = default_train
)
# extract the fitted probabilities
fitted_probabilities_weighted <- predict(glm_fit_weighted,
  newdata = default_test,
  type = "response" # to get output on probability scale
)
# plot the predicted probabilities based on the unweighted and weighted logistic
# regression fits
plot_grid(
  tibble(fitted_probabilities) %>%
    ggplot(aes(x = fitted_probabilities)) +
    geom_histogram(color = "black") +
    scale_y_continuous(limits = c(0, 1600)) +
    labs(
      x = "Fitted probability of default",
      title = "Unweighted"
    ),
  tibble(fitted_probabilities_weighted) %>%
    ggplot(aes(x = fitted_probabilities_weighted)) +
    geom_histogram(color = "black") +
    scale_y_continuous(limits = c(0, 1600)) +
    labs(
      x = "Fitted probability of default",
      title = "Weighted"
    )
)
```



How did the predicted probabilities change? Is this what we expected?

Let's make predictions by thresholding at 0.5 and evaluate the resulting classifier:

```
predictions_weighted <- as.numeric(fitted_probabilities_weighted > 0.5)
bind_rows(
  classification_metrics(
    test_responses = default_test$default,
    test_predictions = predictions
  ) %>%
    mutate(weighting = FALSE, .before = 1),
  classification_metrics(
    test_responses = default_test$default,
    test_predictions = predictions_weighted
  ) %>%
    mutate(weighting = TRUE, .before = 1)
)
```

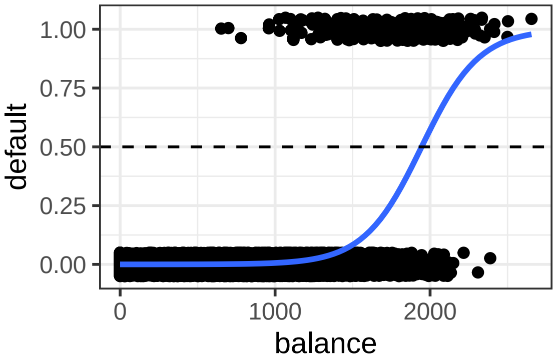
```
## # A tibble: 2 x 6
##   weighting misclass_err w_misclass_err   TPR   TNR     F
##   <lgl>            <dbl> <lgl>          <dbl> <dbl> <dbl>
## 1 FALSE            0.028 NA             0.370 0.995 0.539
## 2 TRUE             0.041 NA             0.726 0.968 0.830
```

How did these test error metrics change, compared to the unweighted case? Is this what we expected?

Plotting a univariate logistic regression fit

Univariate logistic regression fits can be plotted using `geom_smooth()`:

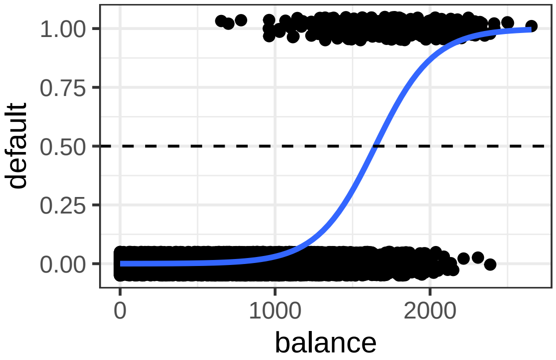
```
default_train %>%
  ggplot(aes(x = balance, y = default)) +
  geom_jitter(height = .05) +
  geom_smooth(
    method = "glm",
    formula = "y~x",
    method.args = list(family = "binomial"), # extra argument
    se = FALSE
  ) +
  geom_hline(yintercept = 0.5, linetype = "dashed")
```



Roughly at what value of balance do we switch from predicting no default to predicting default?

`geom_smooth()` accommodates weighted logistic regression as well:

```
default_train %>%
  ggplot(aes(x = balance, y = default)) +
  geom_jitter(height = .05) +
  geom_smooth(
    method = "glm",
    formula = "y~x",
    method.args = list(family = "binomial"),
    aes(weight = ifelse(default == 1, 5, 1)), # specify weights inside aes()
    se = FALSE
  ) +
  geom_hline(yintercept = 0.5, linetype = "dashed")
```



Now, for roughly what value of balance do we switch from predicting no default to predicting default? How does this compare to the above? Is this what we expected?

Exercise

For logistic regression, if we use `type = "response"` within `predict()` we get the fitted probabilities, whereas `type = "link"` gives us the probabilities on the log-odds scale (also called the "score" in the slides). For `glm_fit` defined above, use `predict()` with `type = "response"` to define a vector called `probabilities` and with `type = "link"` to define a vector called `scores`. Then, create a scatter plot of `probabilities` versus `scores`. What is the relationship between the two?