# Unit 2 Lecture 4: Classification

September 27, 2022

In this R demo, we explore classification with imbalanced classes in the context of KNN applied to a dataset on credit card default.

Let's first load the tidyverse as well as the default data:

```
# load packages
library(tidyverse)
library(stat471)    # for knn(), classification_metrics()

# load default data
default_data = read_tsv("default.tsv", col_types = "ffdd")
default_data
```

```
## # A tibble: 10,000 x 4
##    default student balance income
##    <fct>   <fct>     <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
```

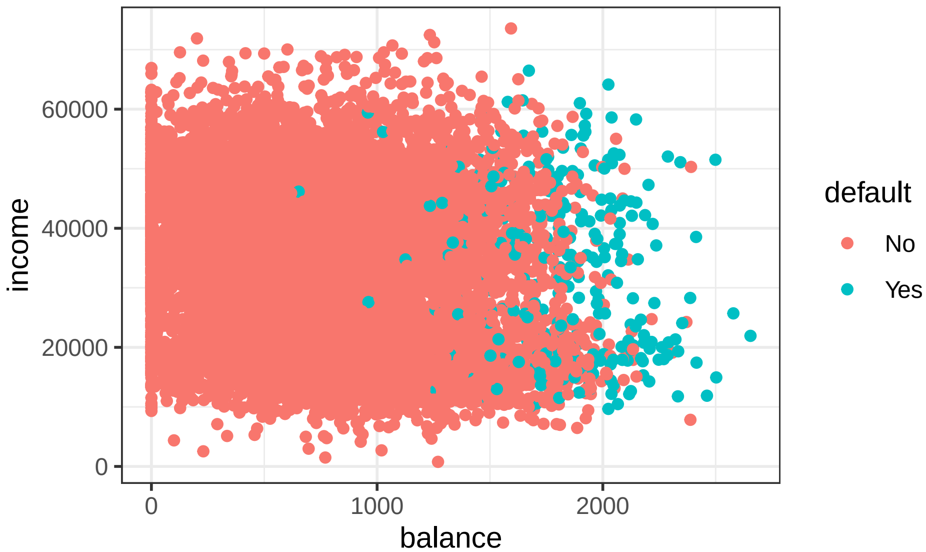
Let's take a look at the default rate in these data:

```
default_data %>%
  summarise(mean(default == "Yes"))
```

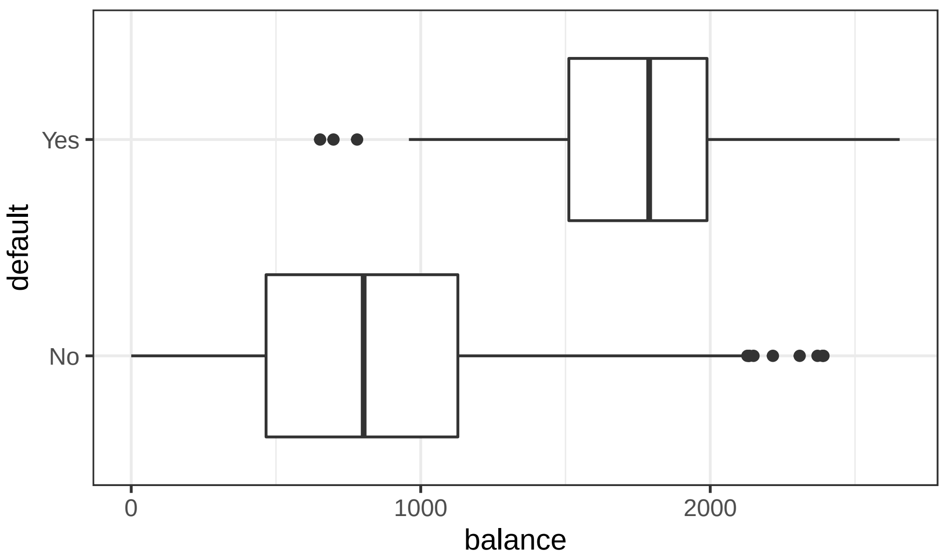
```
## # A tibble: 1 x 1
##   `mean(default == "Yes")`
##                      <dbl>
## 1                   0.0333
```

Uh-oh! It looks like we have imbalanced classes.

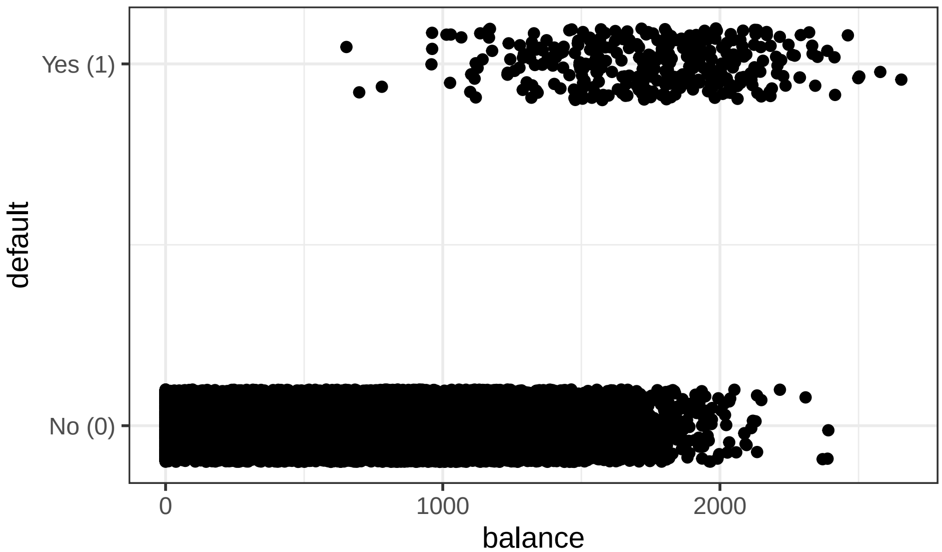
```
# visualize default as a function of `balance` and `income`
default_data %>%
  ggplot(aes(x = balance, y = income, colour = default)) +
  geom_point()
```




```
# visualize default as a function of just `balance`
default_data %>%
  ggplot(aes(x = balance, y = default)) +
  geom_boxplot()
```




```
# another useful visualization of default versus balance is the jitter plot
default_data %>%
  ggplot(aes(x = balance, y = as.numeric(default)-1)) +
  geom_jitter(height = 0.1) +
  scale_y_continuous(breaks = c(0,1), labels = c("No (0)", "Yes (1)")) +
  labs(y = "default")
```




Next, split the observations 50%/50% into training and testing (we won't be doing cross-validation today for the sake of time, though in principle we could).

```
set.seed(47102022)                    # set seed for reproducibility
n = nrow(default_data)
train_samples = sample(1:n, n/2)      # list of rows to be used for training
default_train = default_data %>%
  filter(row_number() %in% train_samples)
default_test = default_data %>%
  filter(!(row_number() %in% train_samples))
```

Actually, perhaps we can stratify on `default` before splitting, since we have imbalanced classes:

```
set.seed(47102022)                    # set seed for reproducibility
train_samples <- default_data %>%
  mutate(rownum = row_number()) %>%
  group_by(default) %>%
  slice_sample(prop = 0.5) %>%
  pull(rownum)
default_train = default_data %>%
  filter(row_number() %in% train_samples)
default_test = default_data %>%
  filter(!(row_number() %in% train_samples))
```

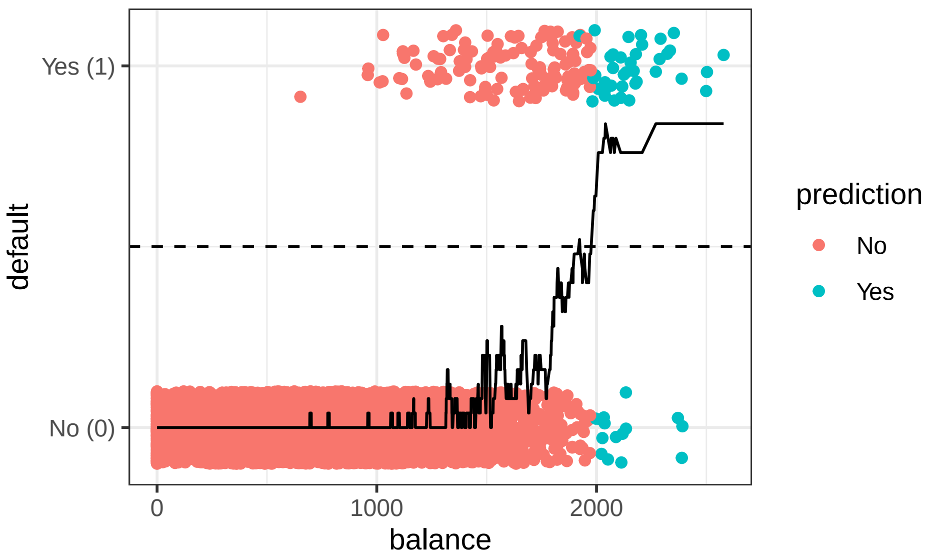
Next, let's apply KNN with K = 25, using just `balance` as a feature.

```
# apply KNN with K = 25
knn_results <- knn(default ~ balance,
                  training_data = default_train,
                  test_data = default_test,
                  k = 25)

# let's add the predictions and probabilities to the test data
predictions <- knn_results$predictions
probabilities <- knn_results$probabilities %>%
  filter(class == "Yes") %>%
  pull(probability)
default_test_with_pred <- default_test %>%
```

```
    mutate(prediction = predictions,
           probability = probabilities)

# visualize the KNN classification
default_test_with_pred %>%
  ggplot(aes(x = balance)) +
  geom_jitter(aes(y = as.numeric(default)-1, colour = prediction), height = 0.1) +
  geom_line(aes(y = probability)) +
  geom_hline(yintercept = 0.5, linetype = "dashed") +
  scale_y_continuous(breaks = c(0,1), labels = c("No (0)", "Yes (1)")) +
  labs(y = "default")
```




Next let's compute a few performance metrics:

```
# compute misclassification error
default_test_with_pred %>%
  summarise(mean(default != prediction))
```

```
## # A tibble: 1 x 1
##   `mean(default != prediction)`
##                           <dbl>
## 1                        0.0286
```

```
# calculate the confusion matrix
conf_matrix <- default_test_with_pred %>%
  select(default, prediction) %>%
  table()
conf_matrix
```

```
##        prediction
## default   No  Yes
##     No  4819   15
##     Yes  128   39
```

```
# calculate true positive rate
TP <- conf_matrix["Yes", "Yes"]
P <- sum(conf_matrix["Yes",])
TPR <- TP/P
```

```r
TPR
```

```
## [1] 0.2335329
```

```r
# calculate true negative rate
TN <- conf_matrix["No", "No"]
N <- sum(conf_matrix["No",])
TNR <- TN/N
TNR
```

```
## [1] 0.996897
```

```r
# calculate the F-score
1/(mean(c(1/TPR, 1/TNR)))
```

```
## [1] 0.3784178
```

```r
# introduce costs associated with misclassifications and computed weighted
#  misclassification error
C_FN = 2000
C_FP = 150
default_test_with_pred %>%
  summarise(weighted_error = mean(C_FP*(prediction == "Yes" & default == "No") +
                                  C_FN*(prediction == "No" & default == "Yes")))
```

```
## # A tibble: 1 x 1
##   weighted_error
##            <dbl>
## 1           51.6
```

A convenient way to calculate all these metrics at once is the `classification_metrics()` function from the `stat471` package.

```r
classification_metrics(
  test_responses = default_test$default,
  test_predictions = knn_results$predictions,
  C_FP = C_FP,
  C_FN = C_FN
)
```

```
## # A tibble: 1 x 5
##   misclass_err w_misclass_err   TPR   TNR     F
##          <dbl>          <dbl> <dbl> <dbl> <dbl>
## 1       0.0286           51.6 0.234 0.997 0.378
```

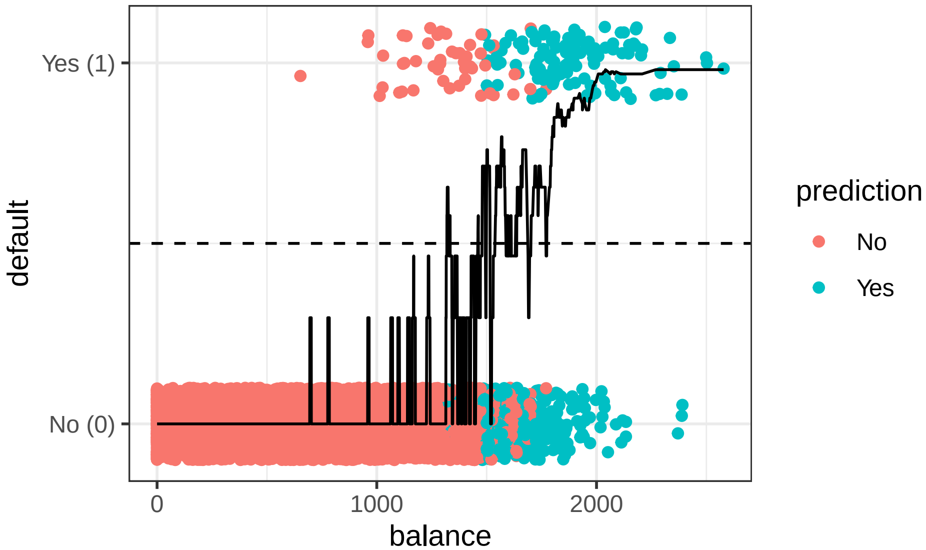
Let's see how the picture changes as we up-weight the observations where `default == "Yes"`. For example, let's up-weight these observations by a factor of 10.

```r
# define weights by up-weighting the positive class
upweighting_factor <- 10
weights <- 1*(default_train$default == "No") +
  upweighting_factor*(default_train$default == "Yes")

# rerun KNN with weights
knn_results_weighted <- knn(default ~ balance,
                  training_data = default_train,
                  test_data = default_test,
                  weights = weights,
                  k = 25)
```

```r
# let's add the predictions and probabilities to the test data
predictions <- knn_results_weighted$predictions
probabilities <- knn_results_weighted$probabilities %>%
  filter(class == "Yes") %>%
  pull(probability)
default_test_with_pred_weighted <- default_test %>%
    mutate(prediction = predictions,
           probability = probabilities)

# visualize the KNN classification
default_test_with_pred_weighted %>%
  ggplot(aes(x = balance)) +
  geom_jitter(aes(y = as.numeric(default)-1, colour = prediction), height = 0.1) +
  geom_line(aes(y = probability)) +
  geom_hline(yintercept = 0.5, linetype = "dashed") +
  scale_y_continuous(breaks = c(0,1), labels = c("No (0)", "Yes (1)")) +
  labs(y = "default")
```




Let's recompute the metrics and compare the weighted and unweighted fits:

```r
rbind(
  # metrics for unweighted KNN
  classification_metrics(test_responses <- default_test$default,
    test_predictions <- knn_results$predictions,
    C_FP = C_FP,
    C_FN = C_FN
  ) %>%
    mutate(weighting = FALSE, .before = 1),
  # metrics for weighted KNN
  classification_metrics(test_responses <- default_test$default,
    test_predictions <- knn_results_weighted$predictions,
    C_FP = C_FP,
    C_FN = C_FN
  ) %>%
    mutate(weighting = TRUE, .before = 1)
)
```

```
## # A tibble: 2 x 6
##   weighting misclass_err w_misclass_err   TPR   TNR     F
##   <lgl>            <dbl>          <dbl> <dbl> <dbl> <dbl>
## 1 FALSE           0.0286           51.6 0.234 0.997 0.378
## 2 TRUE            0.0644           27.8 0.707 0.944 0.808
```

Note that adding weighting slightly increased the misclassification error and slightly decreased the true negative rate. However, it decreased the weighted misclassification error, and significantly increased the true positive rate and the F-score.

How much should we upweight the minority class? We can scan over a range of upweighting factors and recompute the above metrics.

```r
# logarithmically spaced upweighting factors
upweighting_factors <- exp(seq(log(1), log(100), length.out = 10))
num_weights <- length(upweighting_factors)
# create tibble to store the results
results <- tibble(
  upweighting_factor = numeric(num_weights),
  misclass_err = numeric(num_weights),
  w_misclass_error = numeric(num_weights),
  TPR = numeric(num_weights),
  TNR = numeric(num_weights),
  `F` = numeric(num_weights)
)
# iterate over upweighting factors
for (weight_idx in 1:num_weights) {
  # define the weights
  upweighting_factor <- upweighting_factors[weight_idx]
  weights <- 1 * (default_train$default == "No") +
    upweighting_factor * (default_train$default == "Yes")
  # run KNN with those weights
  knn_results_weighted <- knn(default ~ balance,
    training_data = default_train,
    test_data = default_test,
    weights = weights,
    k = 25
  )
  # update results tibble
  results[weight_idx,] = classification_metrics(
    test_responses <- default_test$default,
    test_predictions <- knn_results_weighted$predictions,
    C_FP = C_FP,
    C_FN = C_FN
  ) %>%
    mutate(upweighting_factor = upweighting_factor, .before = 1)
}

# plot the results
results %>%
  pivot_longer(-upweighting_factor, names_to = "metric", values_to = "value") %>%
  mutate(metric = factor(metric,
                         levels = c("TPR", "TNR", "F", "misclass_err", "w_misclass_error"))) %>%
  ggplot(aes(x = upweighting_factor, y = value)) +
  geom_point() +
```

```
  geom_line() +
  geom_vline(xintercept = C_FN/C_FP, linetype = "dashed") +
  facet_wrap(~metric, scales = "free") +
  scale_x_log10() +
  labs(x = "Upweighting factor",
       y = element_blank())
```