Unit 2 Lecture 4: Classification

September 28, 2021

In this R demo, we explore classification with imbalanced classes in the context of KNN applied to the `Default` dataset in `ISLR2`.

Let's first load the tidyverse, the `class` package, as well as the default data:

```
# load packages
library(tidyverse)
library(class)      # for KNN
library(pROC)       # for ROC curves

# load default data
default_data = ISLR2::Default %>% as_tibble()
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
```

```
# recode `default` as binary
default_data = default_data %>% mutate(default = as.numeric(default == "Yes"))
default_data
```

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

```r
# what is the default rate in these data?
```

Next, we choose 1000 observations for training and reserve the rest for validation (we won't be doing cross-validation today for the sake of time, though in principle we could).

```r
set.seed(4712021)                   # set seed for reproducibility
n = nrow(default_data)
train_samples = sample(1:n, 1000)   # list of rows to be used for training
default_train = default_data %>%
  filter(row_number() %in% train_samples)
default_validation = default_data %>%
  filter(!(row_number() %in% train_samples))
```

```r
# visualize default as a function of `balance` and `income`

# visualize default as a function of just `balance`
```

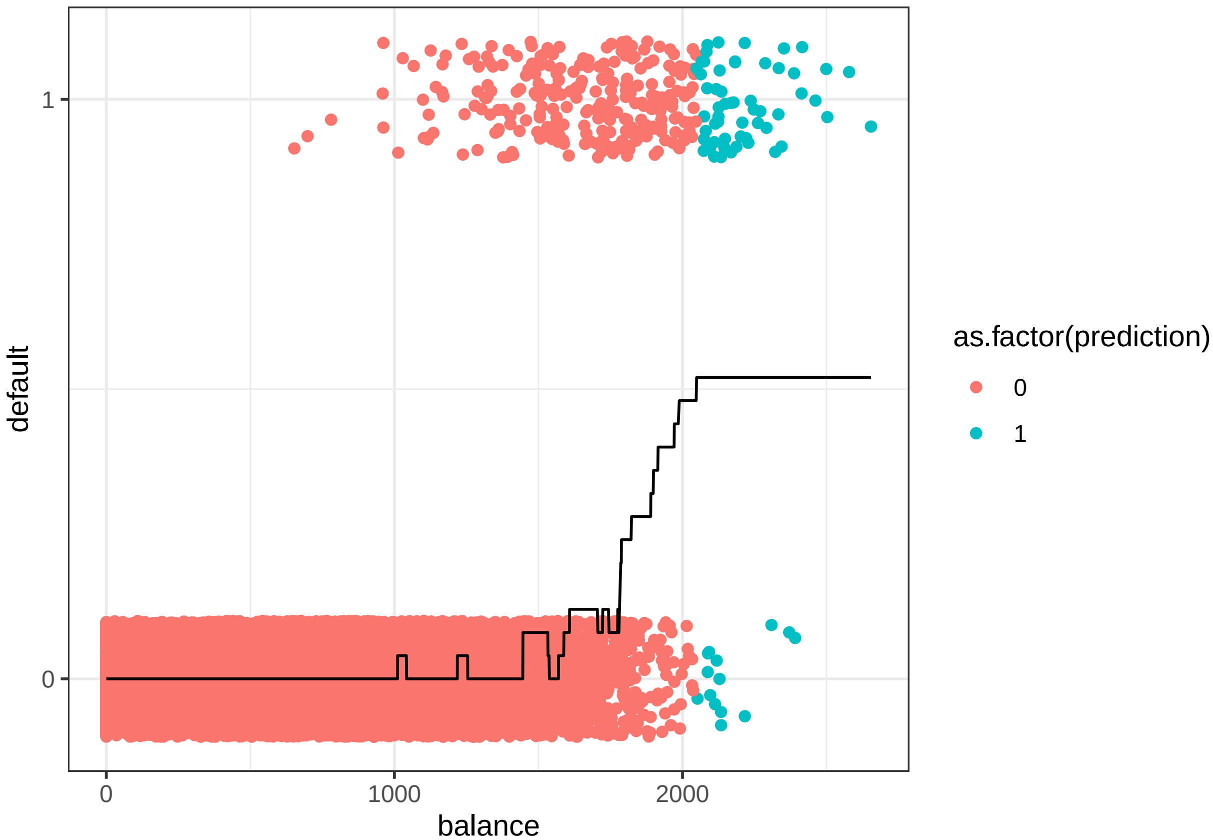
Next, let's apply KNN with K = 25, using just `balance` as a feature.

```r
# apply KNN with K = 25
knn_results = knn(
  train = default_train %>% select(balance),       # training features
  test = default_validation %>% select(balance),   # test features
  cl = default_train %>% pull(default),            # training class labels
  k = 25,                                          # value of K
  prob = TRUE)                                     # keep estimated probabilities

# examine KNN results

# add results to the tibble
default_validation = default_validation %>%
  mutate(prediction = as.numeric(knn_results == 1),
         probability = attributes(knn_results)$prob,
         probability = ifelse(prediction == 1, probability, 1-probability))

# plot the results
default_validation %>%
  ggplot(aes(x = balance)) +
  geom_jitter(aes(y = default, colour = as.factor(prediction)),
              width = 0, height = 0.1) +
  geom_line(aes(y = probability)) +
  scale_y_continuous(breaks = c(0,1)) +
  theme_bw()
```



Next let's compute a few performance metrics:

```
# compute misclassification error

# calculate the confusion matrix
conf_matrix = default_validation %>%
  select(default, prediction) %>%
  table()
conf_matrix
```

```
##         prediction
## default    0    1
##        0 8674   14
##        1  256   56
```

```
# calculate false positive and false negative rates
fpr = conf_matrix[1,2]/(conf_matrix[1,2] + conf_matrix[1,1])
fnr = conf_matrix[2,1]/(conf_matrix[2,1] + conf_matrix[2,2])
tibble(fpr = fpr, fnr = fnr)
```

```
## # A tibble: 1 x 2
##        fpr   fnr
##      <dbl> <dbl>
## 1 0.00161 0.821
```

```
# introduce costs associated with misclassifications and computed weighted
#  misclassification error
C_FN = 2000
C_FP = 150
```

```r
default_validation %>%
  summarise(weighted_error = mean(C_FP*(prediction == 1 & default == 0) +
                                  C_FN*(prediction == 0 & default == 1)))
```

```
## # A tibble: 1 x 1
##   weighted_error
##            <dbl>
## 1           57.1
```

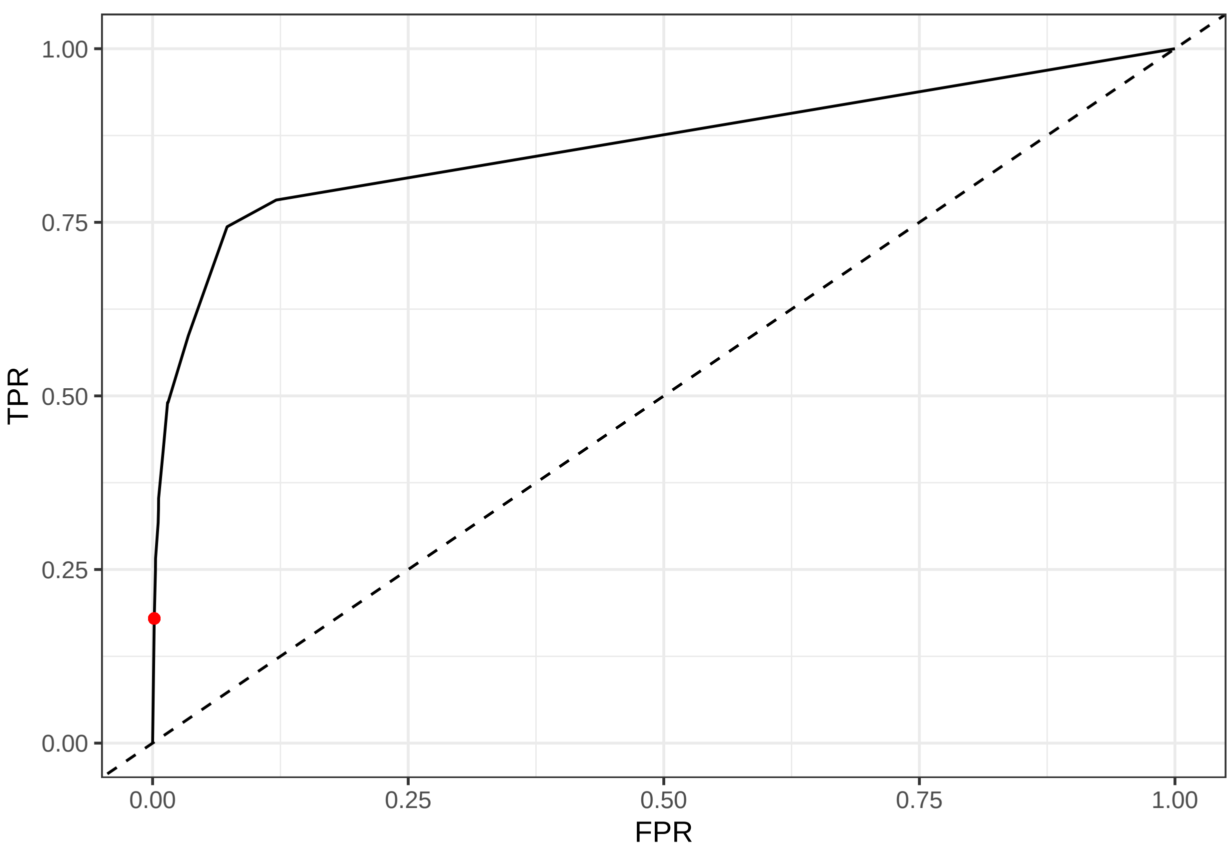
Next let's vary the probability thresholds and see how the classifier performance changes.

```r
# ROC curve
roc_data = roc(default_validation %>% pull(default),
          default_validation %>% pull(probability))
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```r
tibble(FPR = 1-roc_data$specificities,
       TPR = roc_data$sensitivities) %>%
  ggplot(aes(x = FPR, y = TPR)) +
  geom_line() +
  geom_abline(slope = 1, linetype = "dashed") +
  geom_point(x = fpr, y = 1-fnr, colour = "red") +
  theme_bw()
```

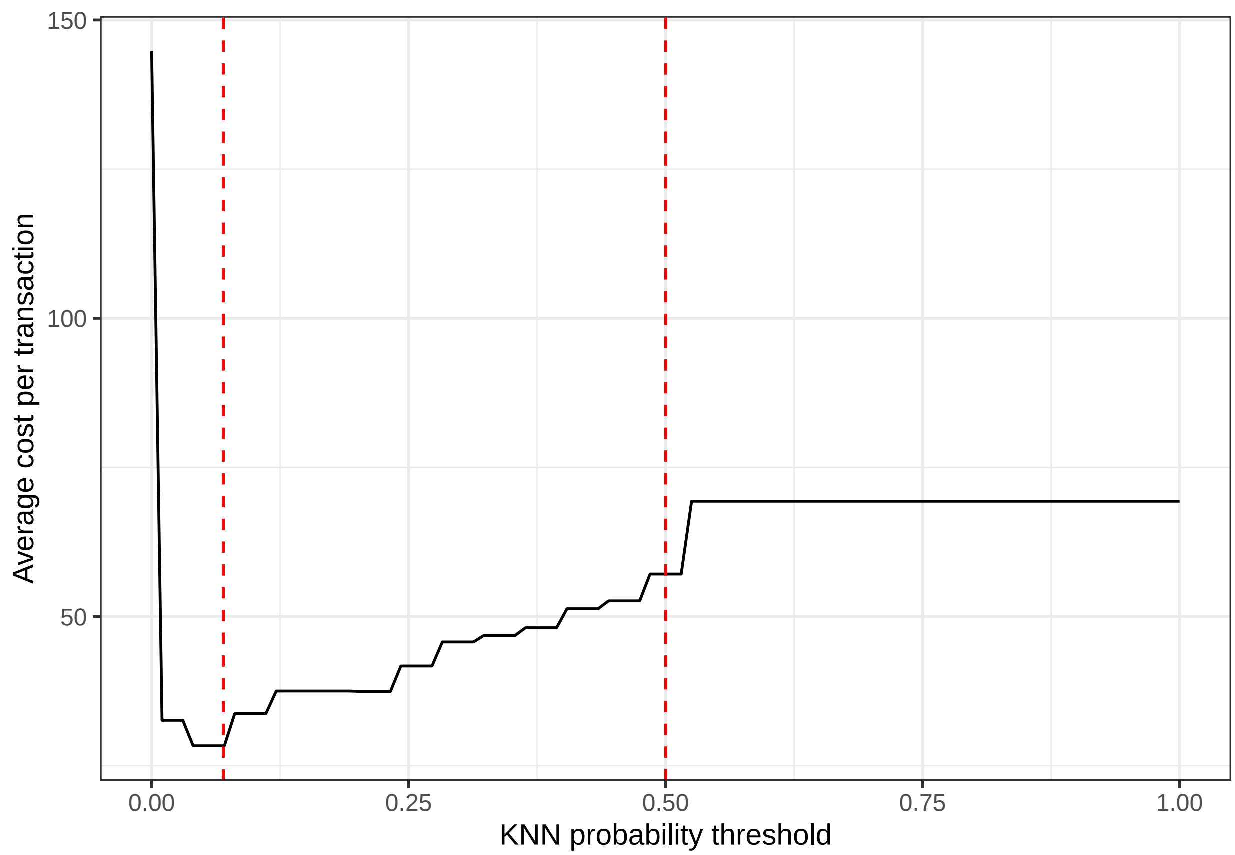



```r
# print the AUC
roc_data$auc
```

```
## Area under the curve: 0.8611
```

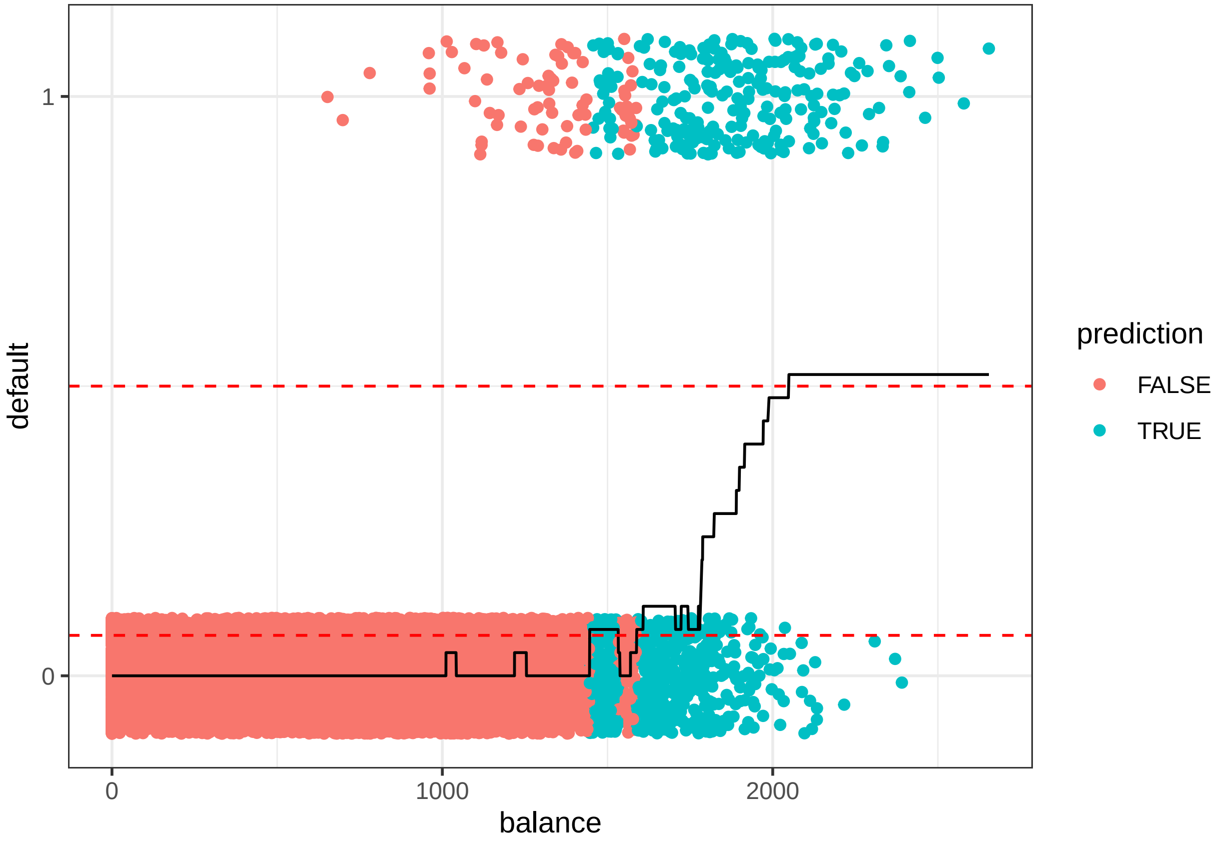
```r
# compute weighted misclassification error as a function of threshold
num_thresholds = 100
thresholds = seq(0, 1, length.out = num_thresholds)
weighted_errors = numeric(num_thresholds)
for(threshold_idx in 1:num_thresholds){
  threshold = thresholds[threshold_idx]
  weighted_errors[threshold_idx] =
    default_validation %>%
    mutate(prediction = probability >= threshold) %>%
    summarise(weighted_error = mean(C_FP*(prediction == 1 & default == 0) +
                                   C_FN*(prediction == 0 & default == 1))) %>%
    pull()
}

# plot the results
tibble(threshold = thresholds, weighted_error = weighted_errors) %>%
  ggplot(aes(x = threshold, y = weighted_error)) +
  geom_line() +
  geom_vline(xintercept = c(C_FP/(C_FP + C_FN), 0.5),
             linetype = "dashed", colour = "red") +
  labs(x = "KNN probability threshold", y = "Average cost per transaction") +
  theme_bw()
```




```r
# visualize the optimal threshold
default_validation %>%
  mutate(prediction = probability >= C_FP/(C_FP + C_FN)) %>%
```

```
ggplot(aes(x = balance)) +
geom_jitter(aes(y = default, colour = prediction), width = 0, height = 0.1) +
geom_line(aes(y = probability)) +
geom_hline(yintercept = c(0.5, C_FP/(C_FP + C_FN)),
           linetype = "dashed", colour = "red") +
scale_y_continuous(breaks = c(0,1)) +
theme_bw()
```




```
# Exercise: Apply KNN with downsampling. Compute the weighted misclassification
# error for different downsampling ratios, and see how well the theoretical best
# downsampling ratio works in practice.
```