Lab 8

Sara Jedwab

11:59PM April 29, 2021

I want to make some use of my CART package. Everyone please try to run the following:

```
if (!pacman::p_isinstalled(YARF)){
  pacman::p_install_gh("kapelner/YARF/YARFJARs", ref = "dev")
  pacman::p_install_gh("kapelner/YARF/YARF", ref = "dev", force = TRUE)
}
options(java.parameters = "-Xmx4000m")
pacman::p_load(YARF)
```

For many of you it will not work. That's okay.

Throughout this part of this assignment you can use either the `tidyverse` package suite or `data.table` to answer but not base R. You can mix `data.table` with `magrittr` piping if you wish but don't go back and forth between `tbl_df`'s and `data.table` objects.

```
pacman::p_load(tidyverse, magrittr, data.table)
```

We will be using the `storms` dataset from the `dplyr` package. Filter this dataset on all storms that have no missing measurements for the two diameter variables, "ts_diameter" and "hu_diameter".

```
data(storms)

storms2 = storms %>%
  filter(!is.na(ts_diameter) & !is.na(hu_diameter) & hu_diameter>0)

storms2
```

```
## # A tibble: 1,022 x 13
##    name   year month   day  hour   lat  long status    category  wind pressure
##    <chr> <dbl> <dbl> <int> <dbl> <dbl> <dbl> <chr>     <ord>    <int>    <int>
##  1 Alex   2004     8     3     6  33   -77.4 hurricane 1           70      983
##  2 Alex   2004     8     3    12  34.2 -76.4 hurricane 2           85      974
##  3 Alex   2004     8     3    18  35.3 -75.2 hurricane 2           85      972
##  4 Alex   2004     8     4     0  36   -73.7 hurricane 1           80      974
##  5 Alex   2004     8     4     6  36.8 -72.1 hurricane 1           80      973
##  6 Alex   2004     8     4    12  37.3 -70.2 hurricane 2           85      973
##  7 Alex   2004     8     4    18  37.8 -68.3 hurricane 2           95      965
##  8 Alex   2004     8     5     0  38.5 -66   hurricane 3          105      957
##  9 Alex   2004     8     5     6  39.5 -63.1 hurricane 3          105      957
## 10 Alex   2004     8     5    12  40.8 -59.6 hurricane 3          100      962
## # ... with 1,012 more rows, and 2 more variables: ts_diameter <dbl>,
## #   hu_diameter <dbl>
```

From this subset, create a data frame that only has storm, observation period number for each storm (i.e., 1, 2, ..., T) and the "ts_diameter" and "hu_diameter" metrics.

```
storms2 = storms2 %>%
  select(name, ts_diameter, hu_diameter) %>%
  group_by(name) %>%
  mutate(period = row_number())

storms2
```

```
## # A tibble: 1,022 x 4
## # Groups:   name [63]
##    name  ts_diameter hu_diameter period
##    <chr>       <dbl>       <dbl>  <int>
##  1 Alex         150.        46.0      1
##  2 Alex         150.        46.0      2
##  3 Alex         190.        57.5      3
##  4 Alex         178.        63.3      4
##  5 Alex         224.        74.8      5
##  6 Alex         224.        74.8      6
##  7 Alex         259.        74.8      7
##  8 Alex         259.        80.6      8
##  9 Alex         345.        80.6      9
## 10 Alex         437.        80.6     10
## # ... with 1,012 more rows
```

Create a data frame in long format with columns "diameter" for the measurement and "diameter_type" which will be categorical taking on the values "hu" or "ts".

```
storms_long = pivot_longer(
  storms2,
  cols = matches("diameter"),
  names_to = "diameter"
)
storms_long
```

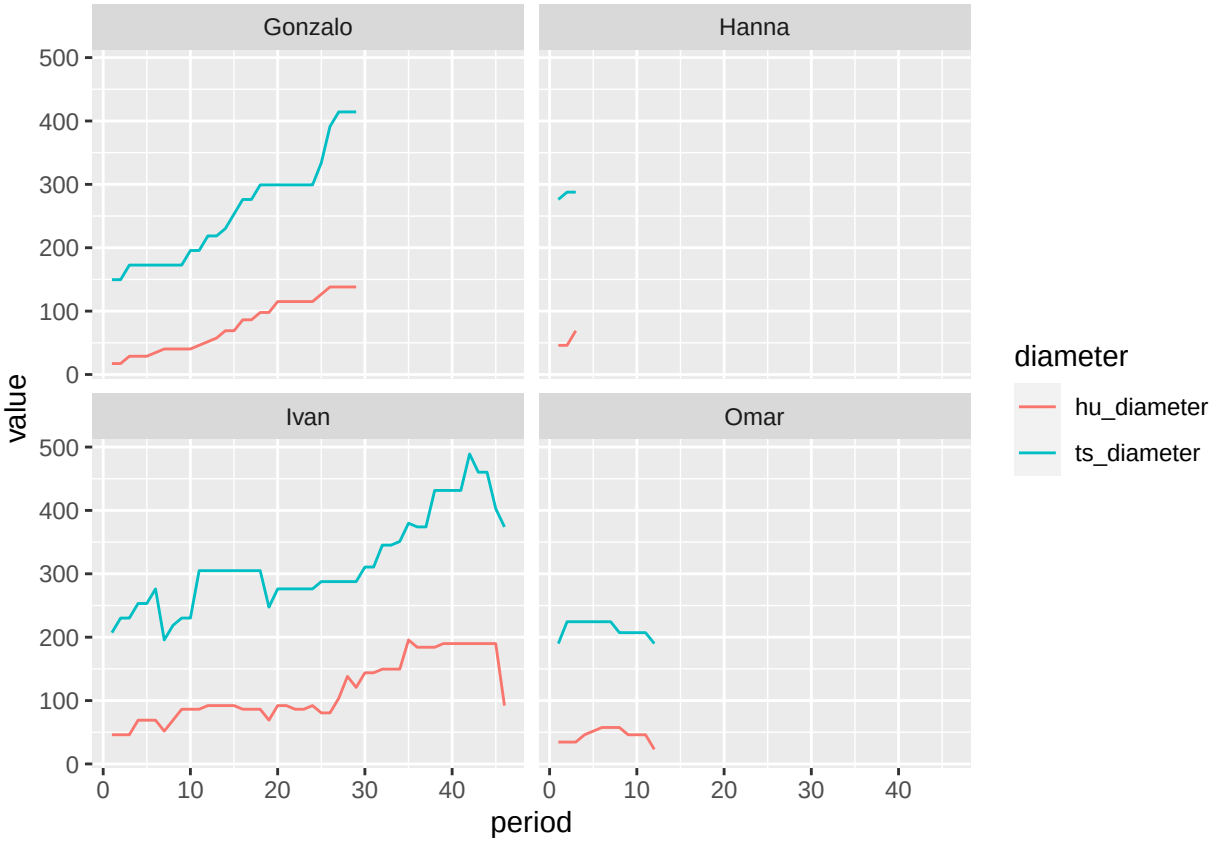
```
## # A tibble: 2,044 x 4
## # Groups:   name [63]
##    name  period diameter    value
##    <chr>  <int> <chr>       <dbl>
##  1 Alex       1 ts_diameter 150.
##  2 Alex       1 hu_diameter  46.0
##  3 Alex       2 ts_diameter 150.
##  4 Alex       2 hu_diameter  46.0
##  5 Alex       3 ts_diameter 190.
##  6 Alex       3 hu_diameter  57.5
##  7 Alex       4 ts_diameter 178.
##  8 Alex       4 hu_diameter  63.3
##  9 Alex       5 ts_diameter 224.
## 10 Alex       5 hu_diameter  74.8
## # ... with 2,034 more rows
```

Using this long-formatted data frame, use a line plot to illustrate both "ts_diameter" and "hu_diameter" metrics by observation period for four random storms using a 2x2 faceting. The two diameters should appear in two different colors and there should be an appropriate legend.

```
storms_sample = sample(unique(storms2$name), 4)

ggplot(storms_long %>% filter(name %in% storms_sample)) +
```

```
  geom_line(aes(x = period, y = value, col = diameter)) +
  facet_wrap(name ~ ., nrow=2)
```



In this next first part of this lab, we will be joining three datasets in an effort to make a design matrix that predicts if a bill will be paid on time. Clean up and load up the three files. Then I'll rename a few features and then we can examine the data frames:

```
rm(list = ls())
pacman::p_load(tidyverse, magrittr, data.table, R.utils)
bills = fread("https://github.com/kapelner/QC_MATH_342W_Spring_2021/raw/master/labs/bills_dataset/bills
payments = fread("https://github.com/kapelner/QC_MATH_342W_Spring_2021/raw/master/labs/bills_dataset/pa
discounts = fread("https://github.com/kapelner/QC_MATH_342W_Spring_2021/raw/master/labs/bills_dataset/d
setnames(bills, "amount", "tot_amount")
setnames(payments, "amount", "paid_amount")

bills = as_tibble(bills)
payments = as_tibble(payments)
discounts = as_tibble(discounts)
head(bills)
```

```
## # A tibble: 6 x 6
##         id due_date   invoice_date tot_amount customer_id discount_id
##      <dbl> <date>     <date>            <dbl>       <int>       <dbl>
## 1 15163811 2017-02-12 2017-01-13       99491.    14290629     5693147
## 2 17244832 2016-03-22 2016-02-21       99476.    14663516     5693147
## 3 16072776 2016-08-31 2016-07-17       99477.    14569622     7302585
## 4 15446684 2017-05-29 2017-05-29       99479.    14488427     5693147
## 5 16257142 2017-06-09 2017-05-10       99678.    14497172     5693147
```

```
## 6 17244880 2017-01-24 2017-01-24       99475.    14663516     5693147
```

```
head(payments)
```

```
## # A tibble: 6 x 4
##         id paid_amount transaction_date  bill_id
##      <dbl>       <dbl> <date>              <dbl>
## 1 15272980      99166. 2017-01-16       16571185
## 2 15246935      99148. 2017-01-03       16660000
## 3 16596393      99158. 2017-06-19       16985407
## 4 16596651      99175. 2017-06-19       17062491
## 5 16687702      99148. 2017-02-15       17184583
## 6 16593510      99154. 2017-06-11       16686215
```

```
head(discounts)
```

```
## # A tibble: 6 x 4
##        id num_days pct_off days_until_discount
##     <dbl>    <int>   <dbl>               <int>
## 1 5000000       20      NA                  NA
## 2 5693147       NA       2                  NA
## 3 6098612       20      NA                  NA
## 4 6386294      120      NA                  NA
## 5 6609438       NA       1                   7
## 6 6791759       31       1                  NA
```

The unit we care about is the bill. The y metric we care about will be "paid in full" which is 1 if the company paid their total amount (we will generate this y metric later).

Since this is the response, we would like to construct the very best design matrix in order to predict y.

I will create the basic steps for you guys. First, join the three datasets in an intelligent way. You will need to examine the datasets beforehand.

```
bills_with_payments = left_join(bills, payments, by = c("id" = "bill_id"))
bills_with_payments
```

```
## # A tibble: 279,118 x 9
##          id due_date   invoice_date tot_amount customer_id discount_id     id.y
##       <dbl> <date>     <date>            <dbl>       <int>       <dbl>    <dbl>
##  1 15163811 2017-02-12 2017-01-13       99491.    14290629     5693147 14670862
##  2 17244832 2016-03-22 2016-02-21       99476.    14663516     5693147 16691206
##  3 16072776 2016-08-31 2016-07-17       99477.    14569622     7302585       NA
##  4 15446684 2017-05-29 2017-05-29       99479.    14488427     5693147 16591210
##  5 16257142 2017-06-09 2017-05-10       99678.    14497172     5693147 16538398
##  6 17244880 2017-01-24 2017-01-24       99475.    14663516     5693147 16691231
##  7 16214048 2017-03-08 2017-02-06       99475.    14679281     5693147 16845763
##  8 15579946 2016-06-13 2016-04-14       99476.    14450223     5693147 16593380
##  9 15264234 2014-06-06 2014-05-07       99480.    14532786     7708050 16957842
## 10 17031731 2017-01-12 2016-12-13       99476.    14658929     5693147       NA
## # ... with 279,108 more rows, and 2 more variables: paid_amount <dbl>,
## #   transaction_date <date>
```

```
bills_with_payments_with_discounts = left_join(bills_with_payments, discounts, by = c("discount_id" = "
bills_with_payments_with_discounts
```

```
## # A tibble: 279,118 x 12
##          id due_date   invoice_date tot_amount customer_id discount_id     id.y
```

```
##       <dbl> <date>     <date>           <dbl>       <int>       <dbl>    <dbl>
##  1 15163811 2017-02-12 2017-01-13       99491.    14290629     5693147 14670862
##  2 17244832 2016-03-22 2016-02-21       99476.    14663516     5693147 16691206
##  3 16072776 2016-08-31 2016-07-17       99477.    14569622     7302585       NA
##  4 15446684 2017-05-29 2017-05-29       99479.    14488427     5693147 16591210
##  5 16257142 2017-06-09 2017-05-10       99678.    14497172     5693147 16538398
##  6 17244880 2017-01-24 2017-01-24       99475.    14663516     5693147 16691231
##  7 16214048 2017-03-08 2017-02-06       99475.    14679281     5693147 16845763
##  8 15579946 2016-06-13 2016-04-14       99476.    14450223     5693147 16593380
##  9 15264234 2014-06-06 2014-05-07       99480.    14532786     7708050 16957842
## 10 17031731 2017-01-12 2016-12-13       99476.    14658929     5693147       NA
## # ... with 279,108 more rows, and 5 more variables: paid_amount <dbl>,
## #   transaction_date <date>, num_days <int>, pct_off <dbl>,
## #   days_until_discount <int>
```

Now create the binary response metric `paid_in_full` as the last column and create the beginnings of a design matrix `bills_data`. Ensure the unit / observation is bill i.e. each row should be one bill!

```
bills_data = bills_with_payments_with_discounts %>%
  mutate(tot_amount = if_else(is.na(pct_off), tot_amount, tot_amount*(1-pct_off/100))) %>%
  group_by(id) %>%
  mutate(sum_of_payment_amt = sum(paid_amount)) %>%
  mutate(paid_in_full = if_else(sum_of_payment_amt >= tot_amount, 1, 0, missing = 0)) %>%
  slice(1) %>%
  ungroup()
table(bills_data$paid_in_full, useNA = "always")
```

```
##
##      0      1   <NA>
## 112664 113770      0
```

How should you add features from transformations (called "featurization")? What data type(s) should they be? Make some features below if you think of any useful ones. Name the columns appropriately so another data scientist can easily understand what information is in your variables.

```
pacman::p_load("lubridate")
bills_data = bills_data %>%
  select(-id, -id.y, -num_days, -transaction_date, -pct_off, -days_until_discount, -sum_of_payment_amt,
  mutate(num_days_to_pay = as.integer(ymd(due_date) - ymd(invoice_date))) %>%
  select(-due_date, -invoice_date) %>%
  mutate(discount_id = as.factor(discount_id)) %>%
  group_by(customer_id) %>%
  mutate(bill_num = row_number()) %>%
  ungroup() %>%
  select(-customer_id, -discount_id) %>%
  relocate(paid_in_full, .after = last_col())
```

Now let's do this exercise. Let's retain 25% of our data for test.

```
K = 4
test_indices = sample(1 : nrow(bills_data), round(nrow(bills_data) / K))
train_indices = setdiff(1 : nrow(bills_data), test_indices)
bills_data_test = bills_data[test_indices, ]
bills_data_train = bills_data[train_indices, ]
```

Now try to build a classification tree model for `paid_in_full` with the features (use the `Xy` parameter in `YARF`). If you cannot get `YARF` to install, use the package `rpart` (the standard R tree package) instead. You

will need to install it and read through some documentation to find the correct syntax.

Warning: this data is highly anonymized and there is likely zero signal! So don't expect to get predictive accuracy. The value of the exercise is in the practice. I think this exercise (with the joining exercise above) may be one of the most useful exercises in the entire semester.

```
#install.packages('rpart')
pacman::p_load(rpart)
tree_mod  = rpart(paid_in_full ~., data = bills_data_train, method = "class")
tree_mod
```

```
## n= 169826
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 169826 84631 1 (0.4983395 0.5016605)
##    2) tot_amount>=99291.29 63977 19246 0 (0.6991731 0.3008269)
##      4) bill_num< 1108.5 32941  2115 0 (0.9357943 0.0642057) *
##      5) bill_num>=1108.5 31036 13905 1 (0.4480281 0.5519719)
##       10) tot_amount< 99476.94 13122  3884 0 (0.7040085 0.2959915) *
##       11) tot_amount>=99476.94 17914  4667 1 (0.2605225 0.7394775) *
##    3) tot_amount< 99291.29 105849 39900 1 (0.3769521 0.6230479)
##      6) bill_num>=1236.5 20088  9980 0 (0.5031860 0.4968140)
##       12) bill_num< 3061.5 14155  6017 0 (0.5749205 0.4250795) *
##       13) bill_num>=3061.5 5933  1970 1 (0.3320411 0.6679589) *
##      7) bill_num< 1236.5 85761 29792 1 (0.3473840 0.6526160) *
```

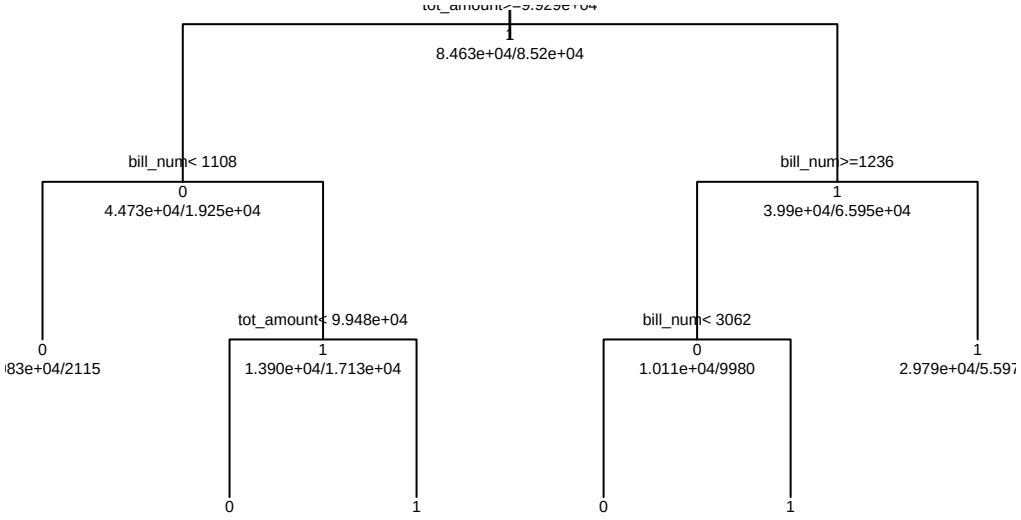
For those of you who installed `YARF`, what are the number of nodes and depth of the tree?

```
#get_tree_num_nodes_leaves_max_depths(paid_in_full_tree)
#nrow(bills_data)/ get_tree_num_nodes_leaves_max_depths(paid_in_full_tree)$num_leaves
nrow(tree_mod$frame) #number of nodes
```

```
## [1] 11
```

For those of you who installed `YARF`, print out an image of the tree.

```
#illustrate_trees(paid_in_full_tree, max_depth = 8, length_in_px_per_half_split = 30, open_file = TRUE)
plot(tree_mod, uniform=TRUE)
text(tree_mod, use.n=TRUE, all=TRUE, cex=.5)
```



Predict on the test set and compute a confusion matrix.

```
y_hat = predict(tree_mod, bills_data_test, type = c("class"), na.action = na.pass)
oos_conf_table = table(bills_data_test$paid_in_full, y_hat)
oos_conf_table
```

```
##    y_hat
##         0     1
##   0 16015 12018
##   1  3906 24669
```

```
#y_hats_test = predict(paid_in_full_tree, bills_data_test, type = c("class"))
#y_hats_test = factor(ifelse(y_hat == 1, "paid in full", "NOT paid in full"))
#mean(y_test != y_hats_test)
#oos_conf_table = table(y_test, y_hats_test)
#oos_conf_table
```

Report the following error metrics: misclassifcation error, precision, recall, F1, FDR, FOR.

```
n = sum(oos_conf_table)
fp = oos_conf_table[1, 2]
fn = oos_conf_table[2, 1]
tp = oos_conf_table[2, 2]
tn = oos_conf_table[1, 1]
num_pred_pos = sum(oos_conf_table[, 2])
num_pred_neg = sum(oos_conf_table[, 1])
num_pos = sum(oos_conf_table[2, ])
num_neg = sum(oos_conf_table[1, ])
misclassifcation_error = (fn + fp)/n
cat("Misclassification error", round(misclassifcation_error * 100, 2), "%\n")
```

```
## Misclassification error 28.13 %
```

```
precision = tp / num_pred_pos
cat("precision", round(precision * 100, 2), "%\n")
```

```
## precision 67.24 %
```

```
recall = tp / num_pos
cat("recall", round(recall * 100, 2), "%\n")
```

```
## recall 86.33 %
```

```
false_discovery_rate = 1 - precision
cat("false_discovery_rate", round(false_discovery_rate * 100, 2), "%\n")
```

```
## false_discovery_rate 32.76 %
```

```
false_omission_rate = fn / num_pred_neg
cat("false_omission_rate", round(false_omission_rate * 100, 2), "%\n")
```

```
## false_omission_rate 19.61 %
```

```
F1 = (2 * tp)/(2 * tp + fp + fn)
cat("F1 score", round(F1 * 100, 2), "%\n")
```

```
## F1 score 75.6 %
```

Is this a good model? (yes/no and explain).

This is a pretty bad model since the false discovery rate is about 30%, which means that about a third of

all observations/bills that we predicted to be paid in full, were in fact NOT paid in full; this is not a good buisness model.

There are probability asymmetric costs to the two types of errors. Assign the costs below and calculate oos total cost.

```
c_fp = 100
c_fn = 1
cost = c_fp * fp + c_fn * fn
cost
```

```
## [1] 1205706
```

We now wish to do asymmetric cost classification. Fit a logistic regression model to this data.

```
log_mod = glm(paid_in_full ~ ., bills_data_train, family = binomial(link = "logit"))
```

Use the function from class to calculate all the error metrics for the values of the probability threshold being 0.001, 0.002, ..., 0.999 in a data frame.

```
compute_metrics_prob_classifier = function(p_hats, y_true, res = 0.001){
  #we first make the grid of all prob thresholds
  p_thresholds = seq(0 + res, 1 - res, by = res) #values of 0 or 1 are trivial

  #now we create a matrix which will house all of our results
  performance_metrics = matrix(NA, nrow = length(p_thresholds), ncol = 12)
  colnames(performance_metrics) = c(
    "p_th",
    "TN",
    "FP",
    "FN",
    "TP",
    "miscl_err",
    "precision",
    "recall",
    "FDR",
    "FPR",
    "FOR",
    "miss_rate"
  )

  #now we iterate through each p_th and calculate all metrics about the classifier and save
  n = length(y_true)
  for (i in 1 : length(p_thresholds)){
    p_th = p_thresholds[i]
    y_hats = factor(ifelse(p_hats >= p_th, 1, 0))
    confusion_table = table(
      factor(y_true, levels = c(0, 1)),
      factor(y_hats, levels = c(0, 1))
    )

    fp = confusion_table[1, 2]
    fn = confusion_table[2, 1]
    tp = confusion_table[2, 2]
    tn = confusion_table[1, 1]
    npp = sum(confusion_table[, 2])
    npn = sum(confusion_table[, 1])
```

```
    np = sum(confusion_table[2, ])
    nn = sum(confusion_table[1, ])

    performance_metrics[i, ] = c(
      p_th,
      tn,
      fp,
      fn,
      tp,
      (fp + fn) / n,
      tp / npp, #precision
      tp / np,  #recall
      fp / npp, #false discovery rate (FDR)
      fp / nn,  #false positive rate (FPR)
      fn / npn, #false omission rate (FOR)
      fn / np   #miss rate
    )
  }

  performance_metrics

}

p_hats_test = predict(log_mod, bills_data_test, type = "response")
p_hats_train = predict(log_mod, bills_data_train, type = "response")

performance_metrics_in_sample = compute_metrics_prob_classifier(p_hats_train, bills_data_train$paid_in_
performance_metrics_in_sample
```

```
##       p_th    TN    FP    FN    TP miscl_err precision recall       FDR FPR
##   1: 0.001     0 84631     0 85195 0.4983395 0.5016605      1 0.4983395   1
##   2: 0.002     0 84631     0 85195 0.4983395 0.5016605      1 0.4983395   1
##   3: 0.003     0 84631     0 85195 0.4983395 0.5016605      1 0.4983395   1
##   4: 0.004     0 84631     0 85195 0.4983395 0.5016605      1 0.4983395   1
##   5: 0.005     0 84631     0 85195 0.4983395 0.5016605      1 0.4983395   1
##  ---
## 995: 0.995 84631     0 85195     0 0.5016605       NaN      0       NaN   0
## 996: 0.996 84631     0 85195     0 0.5016605       NaN      0       NaN   0
## 997: 0.997 84631     0 85195     0 0.5016605       NaN      0       NaN   0
## 998: 0.998 84631     0 85195     0 0.5016605       NaN      0       NaN   0
## 999: 0.999 84631     0 85195     0 0.5016605       NaN      0       NaN   0
##            FOR miss_rate
##   1:       NaN         0
##   2:       NaN         0
##   3:       NaN         0
##   4:       NaN         0
##   5:       NaN         0
##  ---
## 995: 0.5016605         1
## 996: 0.5016605         1
## 997: 0.5016605         1
## 998: 0.5016605         1
## 999: 0.5016605         1
```

```
performance_metrics_oos = compute_metrics_prob_classifier(p_hats_test, bills_data_test$paid_in_full) %>%
performance_metrics_oos
```

```
##       p_th    TN    FP    FN    TP miscl_err precision recall       FDR FPR
##   1: 0.001     0 28033     0 28575 0.4952127 0.5047873      1 0.4952127   1
##   2: 0.002     0 28033     0 28575 0.4952127 0.5047873      1 0.4952127   1
##   3: 0.003     0 28033     0 28575 0.4952127 0.5047873      1 0.4952127   1
##   4: 0.004     0 28033     0 28575 0.4952127 0.5047873      1 0.4952127   1
##   5: 0.005     0 28033     0 28575 0.4952127 0.5047873      1 0.4952127   1
##  ---
## 995: 0.995 28033     0 28575     0 0.5047873       NaN      0       NaN   0
## 996: 0.996 28033     0 28575     0 0.5047873       NaN      0       NaN   0
## 997: 0.997 28033     0 28575     0 0.5047873       NaN      0       NaN   0
## 998: 0.998 28033     0 28575     0 0.5047873       NaN      0       NaN   0
## 999: 0.999 28033     0 28575     0 0.5047873       NaN      0       NaN   0
##            FOR miss_rate
##   1:       NaN         0
##   2:       NaN         0
##   3:       NaN         0
##   4:       NaN         0
##   5:       NaN         0
##  ---
## 995: 0.5047873         1
## 996: 0.5047873         1
## 997: 0.5047873         1
## 998: 0.5047873         1
## 999: 0.5047873         1
```

Calculate the column `total_cost` and append it to this data frame.

```
performance_metrics_oos %<>%
  mutate(total_cost = (c_fp * performance_metrics_oos$FP) + (c_fn * performance_metrics_oos$FN))
```

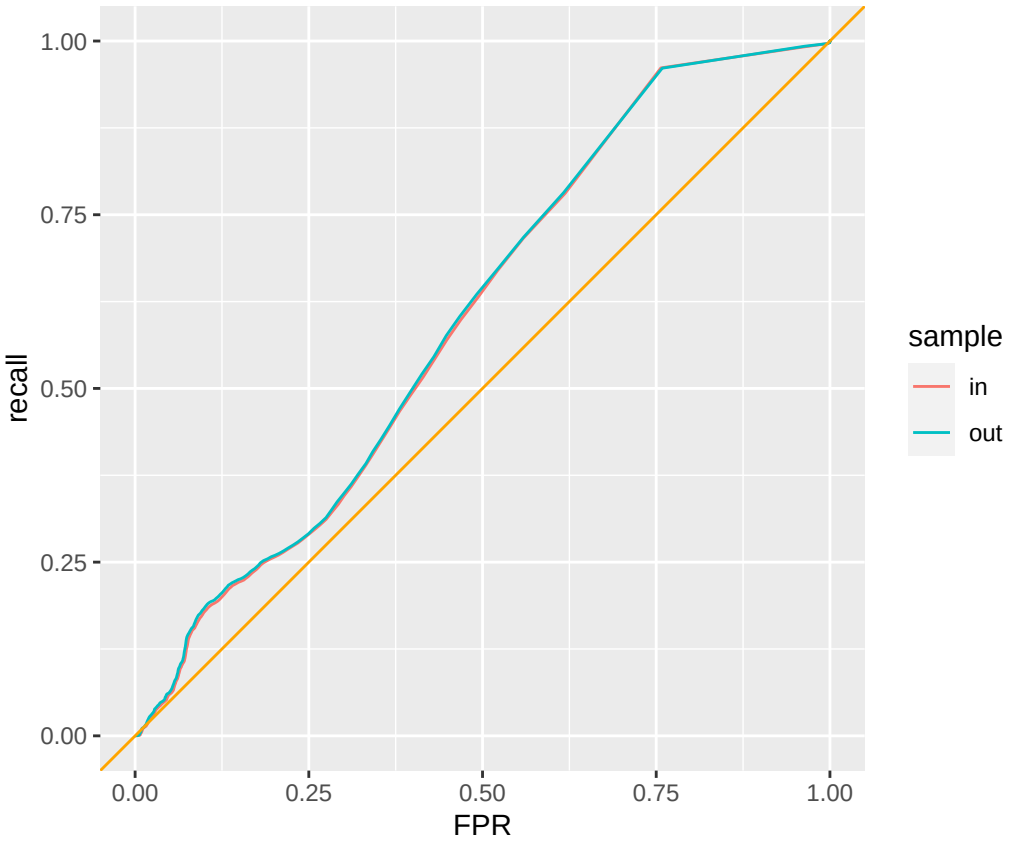
Which is the winning probability threshold value and the total cost at that threshold?

```
i_star = which.min(performance_metrics_oos$total_cost)
performance_metrics_oos[i_star, ]
```

```
##     p_th    TN FP    FN TP miscl_err precision recall FDR FPR       FOR
## 1: 0.699 28033  0 28575  0 0.5047873       NaN      0 NaN   0 0.5047873
##    miss_rate total_cost
## 1:         1      28575
```

Plot an ROC curve and interpret.

```
pacman::p_load(ggplot2)
performance_metrics_in_and_oos = rbind(
    cbind(performance_metrics_in_sample, data.table(sample = "in")),
    cbind(performance_metrics_oos, data.table(sample = "out")), fill = TRUE
)
ggplot(performance_metrics_in_and_oos) +
  geom_line(aes(x = FPR, y = recall, col = sample)) +
  geom_abline(intercept = 0, slope = 1, col = "orange") +
  coord_fixed() + xlim(0, 1) + ylim(0, 1)
```



Tracing out the performance (recall and specificity) for all p_th in [0,1] gives you the red/blue lines which are called the ROC's. The ROC is a metric which compares probability estimation models by calculating the AUC between the blue/red lines and the yellow reference point line. The p value found above will be on point which can be identified by its FDR, recall values, (FPR, recall).

Calculate AUC and interpret.

```
pacman::p_load(pracma)
-trapz(performance_metrics_in_sample$FPR, performance_metrics_in_sample$recall)
```

```
## [1] 0.6009215
```

```
-trapz(performance_metrics_oos$FPR, performance_metrics_oos$recall)
```

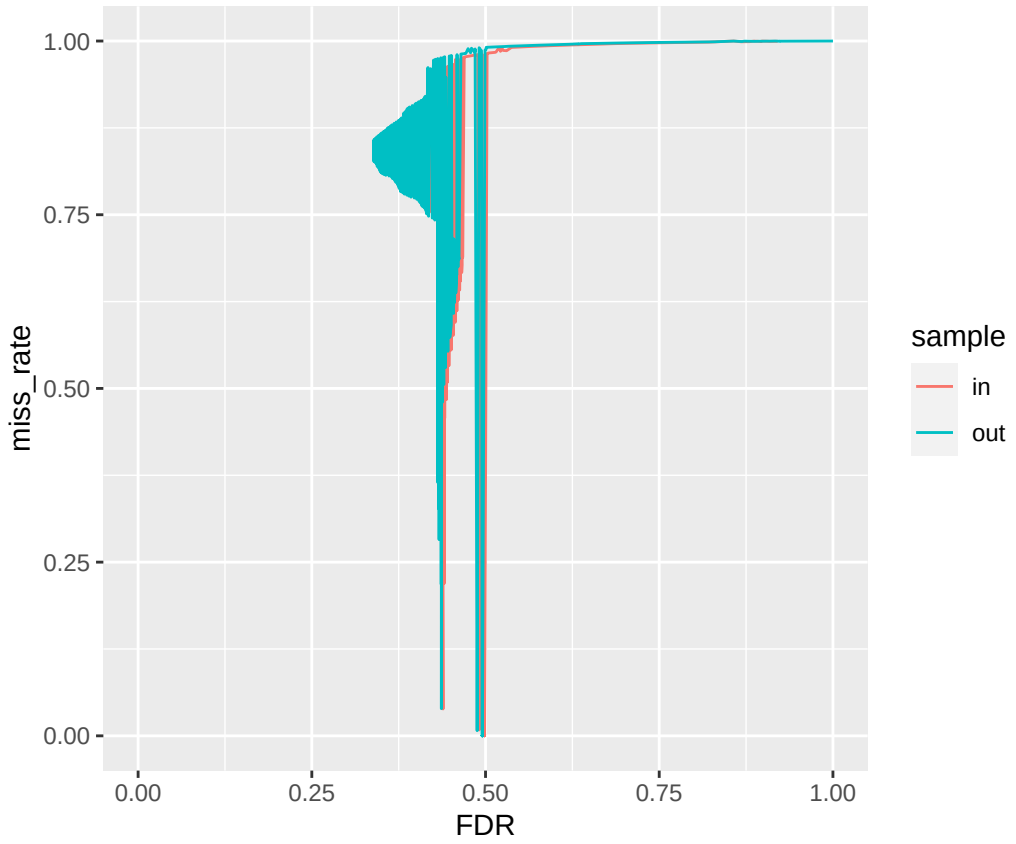
```
## [1] 0.6034794
```

AUC is a metric that gauges the overall fir of a probability estimation model (kind of like a scoring rule). An AUC $> 1/2$ has predictive power, and the model is deemed better and better as the AUC gets closer and closer to 1. Hence this model does have predictive power, but it is still far from being a great model.

Plot a DET curve and interpret.

```
ggplot(performance_metrics_in_and_oos) +
  geom_line(aes(x = FDR, y = miss_rate, col = sample)) +
  coord_fixed() + xlim(0, 1) + ylim(0, 1)
```

```
## Warning: Removed 602 row(s) containing missing values (geom_path).
```



The DET is traced out by varying p_th in [0,1]. This classification model is one point on this curve, and this graph allows you to visually see the tradeoff of the two errors (FOR and FDR) that are critical to prediction. In this case the point on the curve will match the p_th value found above with its FDR and FOR values.