Lab 8

Janine Lim

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 %>%
  dplyr::filter(!is.na(ts_diameter) & !is.na(hu_diameter) & ts_diameter > 0 & 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".

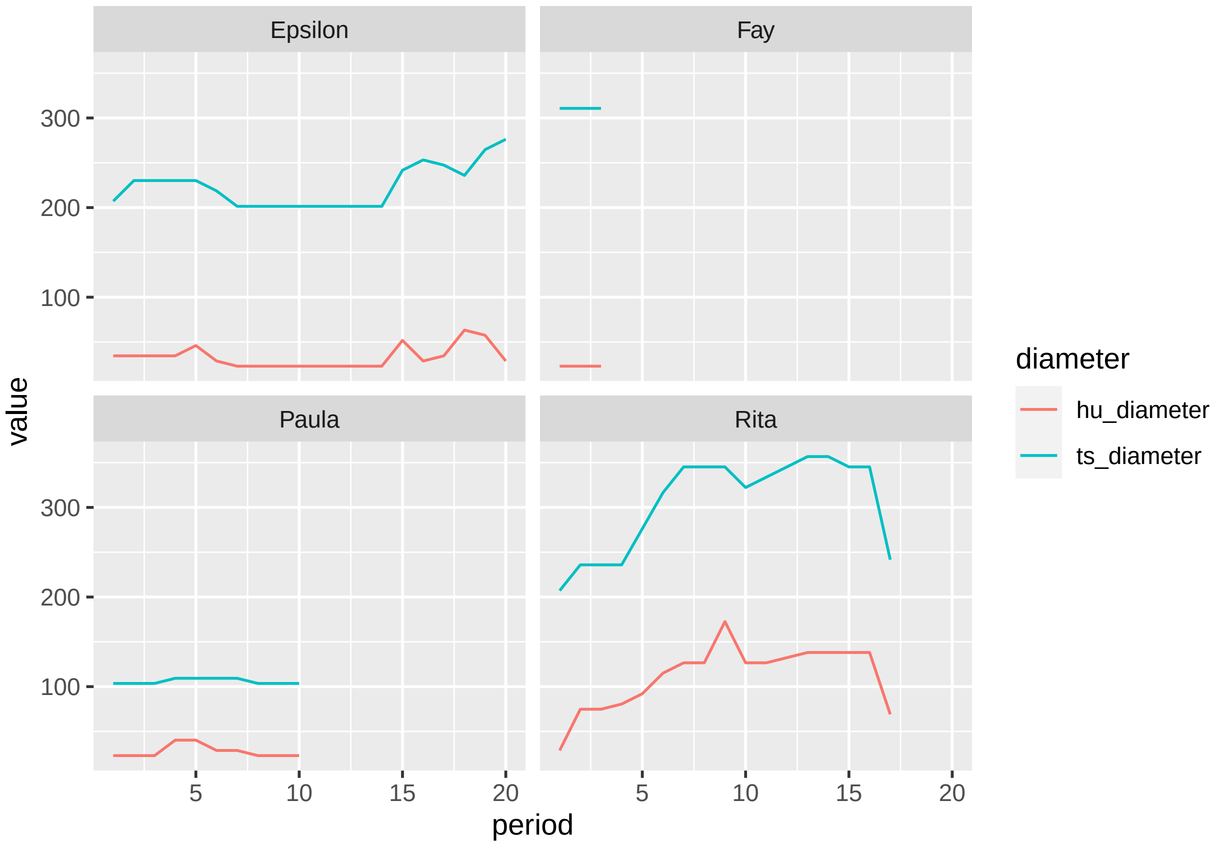
```
storms_long = pivot_longer(storms2,
                           cols = matches("diameter"),
                           names_to = "diameter"
                           )
storms_long
```

```
## # 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.

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

ggplot(storms_long %>%
         dplyr::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:

```r
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")
head(bills)
```

```
##          id   due_date invoice_date tot_amount customer_id discount_id
## 1: 15163811 2017-02-12   2017-01-13   99490.77    14290629     5693147
## 2: 17244832 2016-03-22   2016-02-21   99475.73    14663516     5693147
## 3: 16072776 2016-08-31   2016-07-17   99477.03    14569622     7302585
## 4: 15446684 2017-05-29   2017-05-29   99478.60    14488427     5693147
```

```
## 5: 16257142 2017-06-09   2017-05-10   99678.17    14497172     5693147
## 6: 17244880 2017-01-24   2017-01-24   99475.04    14663516     5693147
```

```
head(payments)
```

```
##          id paid_amount transaction_date  bill_id
## 1: 15272980    99165.60       2017-01-16 16571185
## 2: 15246935    99148.12       2017-01-03 16660000
## 3: 16596393    99158.06       2017-06-19 16985407
## 4: 16596651    99175.03       2017-06-19 17062491
## 5: 16687702    99148.20       2017-02-15 17184583
## 6: 16593510    99153.94       2017-06-11 16686215
```

```
head(discounts)
```

```
##         id num_days pct_off days_until_discount
## 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
```

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

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> <chr>      <chr>             <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 <chr>
```

```r
bills_with_payments_with_discounts = left_join(bills_with_payments, discounts, by = c("discount_id" = "i
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> <chr>      <chr>             <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 <chr>, 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!

```r
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_payment_amt = sum(paid_amount)) %>%
  mutate(paid_in_full = if_else(sum_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.

```r
pacman::p_load(lubridate)
bils_data = bills_data %<>%
  select(-id, -id.y, -transaction_date, -pct_off, -days_until_discount, -sum_payment_amt, -paid_amount,
  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) %>%
  relocate(paid_in_full, .after = last_col())
head(bills_data)
```

```
## # A tibble: 6 x 5
##   tot_amount discount_id num_days_to_pay bill_num paid_in_full
##        <dbl> <fct>                 <int>    <int>        <dbl>
## 1     99480. 7397895                  45        1            0
## 2     99529. 7397895                  30        1            0
## 3     99477. 7397895                  11        1            0
## 4     99479. 7397895                   0        2            0
## 5     99477. 7397895                  30        3            0
## 6     99477. 7397895                  30        1            0
```

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.

```
pacman::p_load(rpart)
mod1 = rpart(paid_in_full ~., data = bills_data_train, method = "class")
```

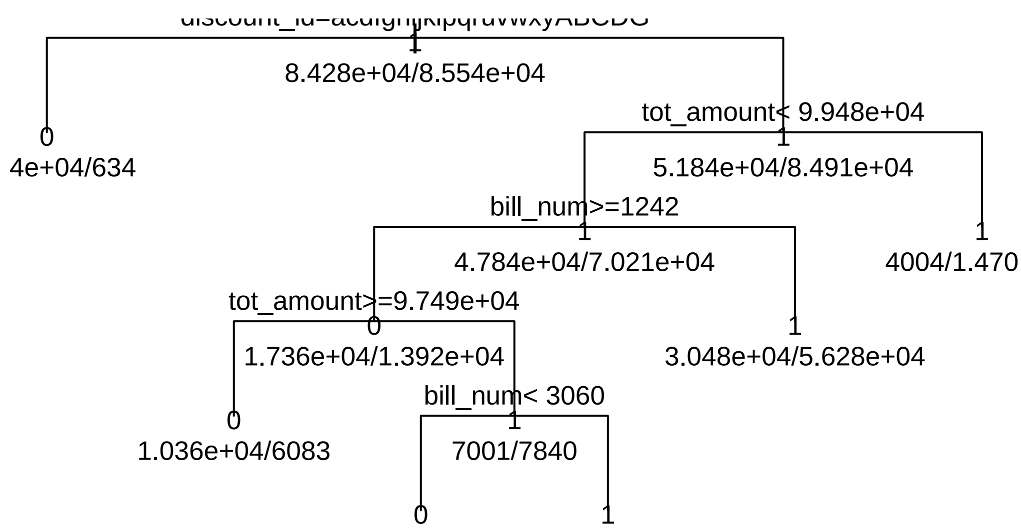
For those of you who installed `YARF`, what are the number of nodes and depth of the tree?

```
nrow(mod1$frame) ##number of nodes
```

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

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

```
plot(mod1, uniform=TRUE)
text(mod1, use.n=TRUE, all=TRUE, cex=.8)
```



Predict on the test set and compute a confusion matrix.

```
yhat = predict(mod1, bills_data_test, type = c("class"), na.action = na.pass)
confusion_matrix =table(bills_data_test$paid_in_full, yhat)
confusion_matrix
```

```
##    yhat
##         0     1
##   0 16287 12096
##   1  3524 24701
```

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

```
ME = sum(confusion_matrix[1,2], confusion_matrix[2,1])/sum(confusion_matrix)
precision = confusion_matrix[2,2]/sum(confusion_matrix[,2])
recall = confusion_matrix[2,2]/sum(confusion_matrix[2,])
F1 = 2/(1/recall + 1/precision)
FDR = confusion_matrix[1,2]/sum(confusion_matrix[,2])
FOR = confusion_matrix[2,1]/sum(confusion_matrix[,1])

cat("misclassifcation error: ", ME)
```

```
## misclassifcation error:  0.2759327
```

```
cat("precision:",precision)
```

```
## precision: 0.6712775
```

```
cat("recall:", recall)
```

```
## recall: 0.8751461
```

```
cat("F1:", F1)
```

```
## F1: 0.7597736
```

```
cat("FDR:", FDR)
```

```
## FDR: 0.3287225
```

```
cat("FOR:", FOR)
```

```
## FOR: 0.177881
```

Is this a good model? (yes/no and explain). This is not a good model because even though we have high recall error, we have low misclassification error and low precision error, which would be bad in this example when we are trying to predict if people will pay their bills back in full, or at all. Since the cost of a false positive would be higher, we would want to have a higher precision and recall rate, making this a bad model based on the asymmetric costs of this model.

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

```
cfn = -2
cfp = -10

fp = confusion_matrix[1,2]
fn = confusion_matrix[2,1]

oos_cost = cfp*fp + cfn*fn
oos_cost
```

```
## [1] -128008
```

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

```
log_mod = glm(paid_in_full ~., data = 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.

```r
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
```

```
    )
  }

  #finally return the matrix
  performance_metrics
}
p_hats_train = predict(log_mod, bills_data_train, type = "response")
y_train = bills_data_train$paid_in_full
performance_metrics_in_sample = compute_metrics_prob_classifier(p_hats_train, y_train) %>% data.table
performance_metrics_in_sample
```

```
##       p_th    TN    FP    FN    TP miscl_err precision       recall       FDR
##   1: 0.001 10516 72827     1 85517 0.4288389 0.5400710 9.999883e-01 0.4599290
##   2: 0.002 10516 72827     1 85517 0.4288389 0.5400710 9.999883e-01 0.4599290
##   3: 0.003 10516 72827     1 85517 0.4288389 0.5400710 9.999883e-01 0.4599290
##   4: 0.004 10516 72827     1 85517 0.4288389 0.5400710 9.999883e-01 0.4599290
##   5: 0.005 10516 72827     1 85517 0.4288389 0.5400710 9.999883e-01 0.4599290
##  ---
## 995: 0.995 83336     7 85515     3 0.5035860 0.3000000 3.508033e-05 0.7000000
## 996: 0.996 83336     7 85515     3 0.5035860 0.3000000 3.508033e-05 0.7000000
## 997: 0.997 83336     7 85515     3 0.5035860 0.3000000 3.508033e-05 0.7000000
## 998: 0.998 83336     7 85515     3 0.5035860 0.3000000 3.508033e-05 0.7000000
## 999: 0.999 83336     7 85516     2 0.5035919 0.2222222 2.338689e-05 0.7777778
##               FPR          FOR    miss_rate
##   1: 8.738226e-01 9.508415e-05 1.169344e-05
##   2: 8.738226e-01 9.508415e-05 1.169344e-05
##   3: 8.738226e-01 9.508415e-05 1.169344e-05
##   4: 8.738226e-01 9.508415e-05 1.169344e-05
##   5: 8.738226e-01 9.508415e-05 1.169344e-05
##  ---
## 995: 8.399026e-05 5.064524e-01 9.999649e-01
## 996: 8.399026e-05 5.064524e-01 9.999649e-01
## 997: 8.399026e-05 5.064524e-01 9.999649e-01
## 998: 8.399026e-05 5.064524e-01 9.999649e-01
## 999: 8.399026e-05 5.064554e-01 9.999766e-01
```

```
y_test = bills_data_test$paid_in_full
phats_test = predict(log_mod, bills_data_test, type = "response")
performance_metrics_oos = compute_metrics_prob_classifier(phats_test, y_test) %>% data.table
```

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

```
performance_metrics_in_sample$total_cost = cfp*performance_metrics_in_sample$FP + cfn * performance_met
performance_metrics_in_sample
```

```
##       p_th    TN    FP    FN    TP miscl_err precision       recall       FDR
##   1: 0.001 10516 72827     1 85517 0.4288389 0.5400710 9.999883e-01 0.4599290
##   2: 0.002 10516 72827     1 85517 0.4288389 0.5400710 9.999883e-01 0.4599290
##   3: 0.003 10516 72827     1 85517 0.4288389 0.5400710 9.999883e-01 0.4599290
##   4: 0.004 10516 72827     1 85517 0.4288389 0.5400710 9.999883e-01 0.4599290
##   5: 0.005 10516 72827     1 85517 0.4288389 0.5400710 9.999883e-01 0.4599290
##  ---
```

```
## 995: 0.995 83336     7 85515     3 0.5035860 0.3000000 3.508033e-05 0.7000000
## 996: 0.996 83336     7 85515     3 0.5035860 0.3000000 3.508033e-05 0.7000000
## 997: 0.997 83336     7 85515     3 0.5035860 0.3000000 3.508033e-05 0.7000000
## 998: 0.998 83336     7 85515     3 0.5035860 0.3000000 3.508033e-05 0.7000000
## 999: 0.999 83336     7 85516     2 0.5035919 0.2222222 2.338689e-05 0.7777778
##                FPR          FOR    miss_rate total_cost
##   1: 8.738226e-01 9.508415e-05 1.169344e-05    -728272
##   2: 8.738226e-01 9.508415e-05 1.169344e-05    -728272
##   3: 8.738226e-01 9.508415e-05 1.169344e-05    -728272
##   4: 8.738226e-01 9.508415e-05 1.169344e-05    -728272
##   5: 8.738226e-01 9.508415e-05 1.169344e-05    -728272
##  ---
## 995: 8.399026e-05 5.064524e-01 9.999649e-01    -171100
## 996: 8.399026e-05 5.064524e-01 9.999649e-01    -171100
## 997: 8.399026e-05 5.064524e-01 9.999649e-01    -171100
## 998: 8.399026e-05 5.064524e-01 9.999649e-01    -171100
## 999: 8.399026e-05 5.064554e-01 9.999766e-01    -171102
```

```
performance_metrics_oos$total_cost = cfp*performance_metrics_oos$FP + cfn * performance_metrics_oos$FN
performance_metrics_oos
```

```
##       p_th    TN    FP    FN    TP miscl_err precision    recall      FDR
##   1: 0.001  3622 24417     2 28219 0.4313701  0.536116 0.9999291 0.463884
##   2: 0.002  3622 24417     2 28219 0.4313701  0.536116 0.9999291 0.463884
##   3: 0.003  3622 24417     2 28219 0.4313701  0.536116 0.9999291 0.463884
##   4: 0.004  3622 24417     2 28219 0.4313701  0.536116 0.9999291 0.463884
##   5: 0.005  3622 24417     2 28219 0.4313701  0.536116 0.9999291 0.463884
##  ---
## 995: 0.995 28038     1 28221     0 0.4985514  0.000000 0.0000000 1.000000
## 996: 0.996 28038     1 28221     0 0.4985514  0.000000 0.0000000 1.000000
## 997: 0.997 28038     1 28221     0 0.4985514  0.000000 0.0000000 1.000000
## 998: 0.998 28038     1 28221     0 0.4985514  0.000000 0.0000000 1.000000
## 999: 0.999 28038     1 28221     0 0.4985514  0.000000 0.0000000 1.000000
##                FPR          FOR    miss_rate total_cost
##   1: 8.708228e-01 0.0005518764 7.086921e-05    -244174
##   2: 8.708228e-01 0.0005518764 7.086921e-05    -244174
##   3: 8.708228e-01 0.0005518764 7.086921e-05    -244174
##   4: 8.708228e-01 0.0005518764 7.086921e-05    -244174
##   5: 8.708228e-01 0.0005518764 7.086921e-05    -244174
##  ---
## 995: 3.566461e-05 0.5016264064 1.000000e+00     -56452
## 996: 3.566461e-05 0.5016264064 1.000000e+00     -56452
## 997: 3.566461e-05 0.5016264064 1.000000e+00     -56452
## 998: 3.566461e-05 0.5016264064 1.000000e+00     -56452
## 999: 3.566461e-05 0.5016264064 1.000000e+00     -56452
```

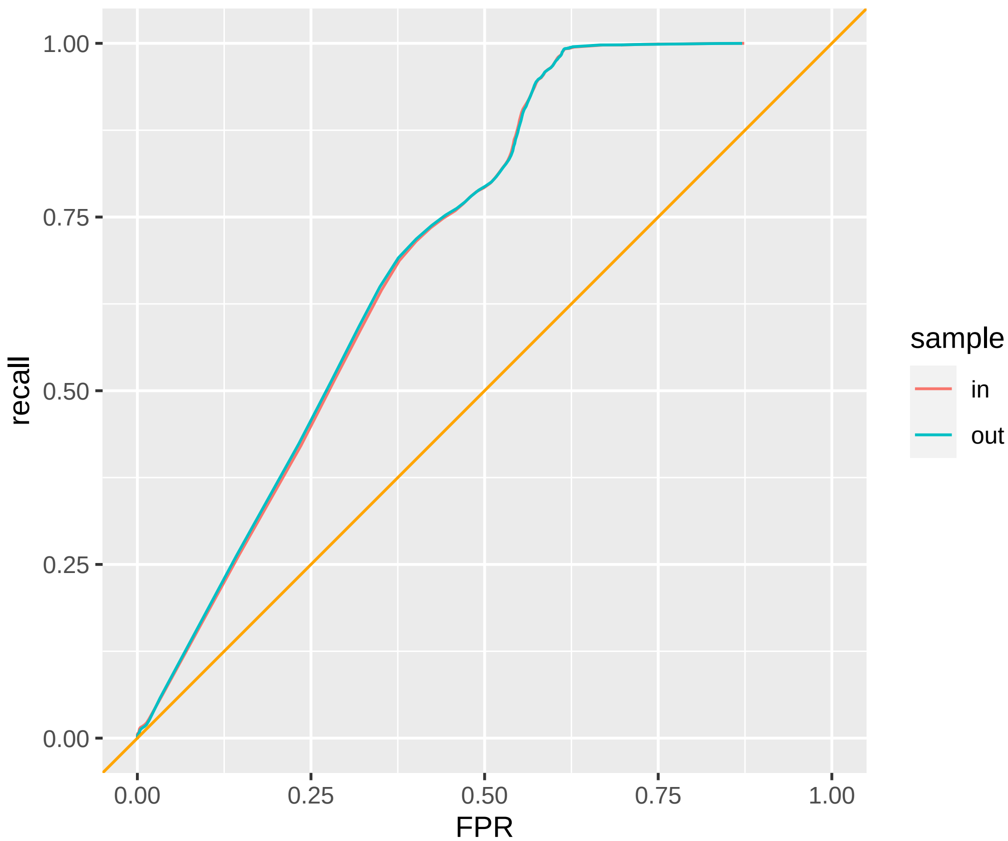
Which is the winning probability threshold value and the total cost at that threshold?

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

```
## [1] -244174
```

Plot an ROC curve and interpret.

```
performance_metrics_in_and_oos = rbind(
    cbind(performance_metrics_in_sample, data.table(sample = "in")),
    cbind(performance_metrics_oos, data.table(sample = "out"))
)
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)
```



The ROC curve shows that each of the p_threshold models does better than just randomly guessing, since each threshold model has increasingly higher recall as FPR increases.

Calculate AUC and interpret.

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

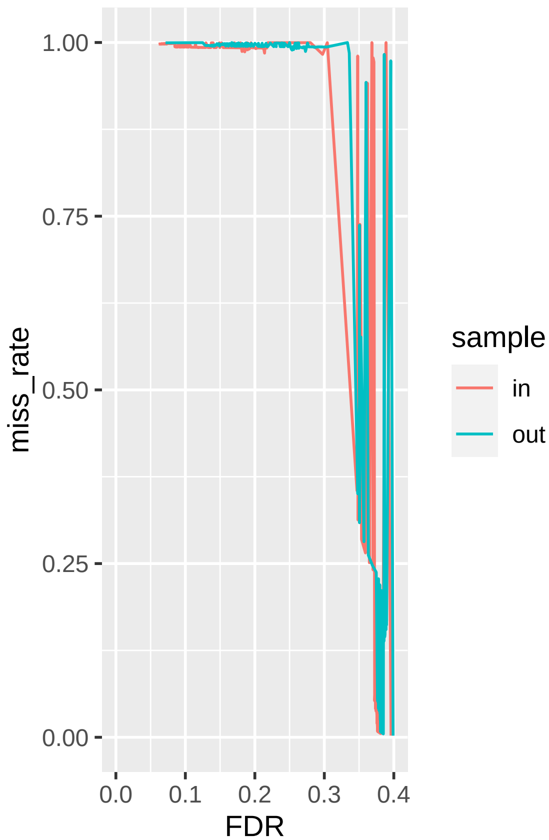
```
## [1] 0.5806194
```

The out of sample AUC is approximately 58.18%, meaning overall, it has slightly better class separation capacity than just randomly guessing.

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, .4) + ylim(0, 1)
```

```
## Warning: Removed 161 row(s) containing missing values (geom_path).
```



The DET curve shows that the tradeoff between FDR and FOR; here we see that an optimal tradeoff is when FDR is approximately 0.4 and FOR is close to 0. Once FDR increases beyond 0.4, we see that FOR also begins to increase, making this an optimal tradeoff point.