Lab 8

Hubert Majewski

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 = "-Xmx8000m")
#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) & ts_diameter > 0 & hu_diameter :

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
```

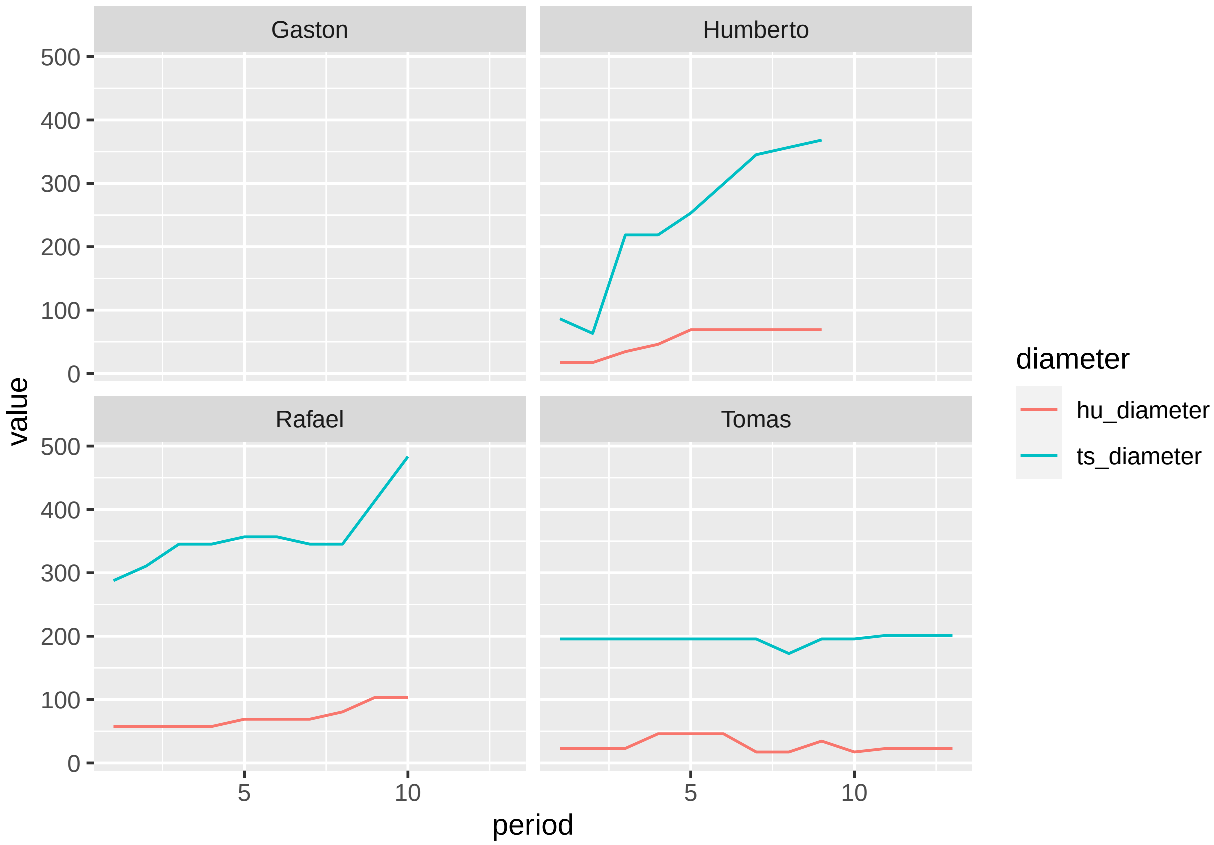
Using this long-formatted data frame, use a line plot to illustrate both "ts_diameter" and "hu_diameter" metrics by observation period for four random storms using a 2x2 faceting. The two diameters should appear in two different colors and there should be an appropriate legend.

```
storms_sample <- sample(unique(storms2$name), 4)
storms_sample
```

```
## [1] "Tomas"    "Humberto" "Gaston"   "Rafael"
```

```
ggplot(storms_long %>% filter(name %in% storms_sample)) +
    geom_line(aes(x = period, y = value, col = diameter)) +
    facet_wrap(name~., nrow = 2)
```

```
## geom_path: Each group consists of only one observation. Do you need to adjust
## the group aesthetic?
```



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")
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_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_amount = sum(paid_amount)) %>%
    mutate(paid_in_full = if_else(sum_of_payment_amount >= 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.

```
bills_data <- bills_data %>%
    select(-id, -id.y, -num_days, -transaction_date, -pct_off, -days_until_discount, -sum_of_payment_amo
    mutate(num_days_to_pay = as.integer(difftime(due_date, invoice_date, units = c("days")))) %>%
    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())

bills_data
```

```
## # A tibble: 226,434 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
## 7     99477. 7397895                    0        1             0
## 8     99477. 7397895                   30        2             0
## 9     99485. 7397895                   30        4             0
## 10    99477. 7397895                   30        2             0
## # ... with 226,424 more rows
```

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)
mod = rpart(paid_in_full ~., data = bills_data_train, method = "class")
mod
```

```
## n= 169826
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 169826 84615 1 (0.49824526 0.50175474)
##    2) discount_id=5e+06,6098612,6609438,7079442,7197225,7302585,7397895,7484907,7564949,7708050,7890
##    3) discount_id=5693147,6945910,7944439,7995732,8258097,8367296,8806662,9060443,9077537 136623 520
##      6) tot_amount< 99476.98 117986 48011 1 (0.40692116 0.59307884)
##       12) bill_num>=1242.5 31288 13748 0 (0.56059831 0.43940169)
##         24) tot_amount>=97486.99 16499  5986 0 (0.63719013 0.36280987) *
##         25) tot_amount< 97486.99 14789  7027 1 (0.47515045 0.52484955)
##           50) bill_num< 3062.5 9474  4208 0 (0.55583703 0.44416297) *
##           51) bill_num>=3062.5 5315  1761 1 (0.33132643 0.66867357) *
##       13) bill_num< 1242.5 86698 30471 1 (0.35146139 0.64853861) *
##      7) tot_amount>=99476.98 18637  4027 1 (0.21607555 0.78392445) *
```

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

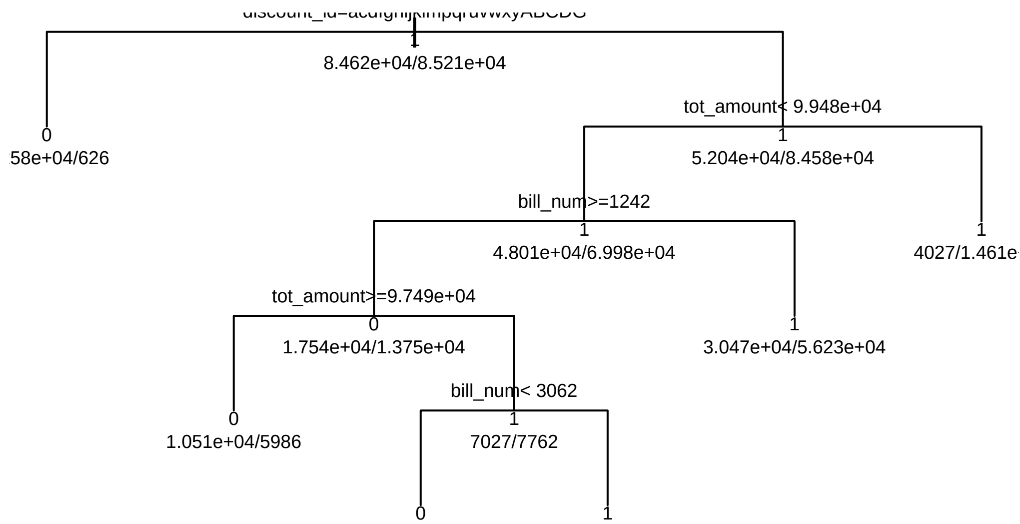
```
#Number of cells
nrow(mod$frame)
```

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

```
#Levels of a BALANCED tree
#ceiling(log(nrow(mod1$frame), 2))
```

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

```
#NOT YARF
plot(mod, uniform=TRUE)
text(mod, use.n=TRUE, all=TRUE, cex=.6)
```



Predict on the test set and compute a confusion matrix.

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

```
##    yhat
##         0     1
##   0 15917 12132
##   1  3668 24891
```

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

```
n = sum(oos_conf_table)
fp = oos_conf_table[1, 2]
```

```r
fn = oos_conf_table[2, 1]
tp = oos_conf_table[2, 2]
tn = oos_conf_table[1, 1]
f1 = 2 * tp / (2 * tp + fp + fn)
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, ])
precision = tp / num_pred_pos
recall = tp / num_pos
false_discovery_rate = 1 - precision
false_omission_rate = fn / num_pred_neg
missclassification_error = (fn + fp) / n

cat("precision", round(precision * 100, 2), "%\n")
```

```
## precision 67.23 %
```

```r
cat("recall", round(recall * 100, 2), "%\n")
```

```
## recall 87.16 %
```

```r
cat("false_discovery_rate", round(false_discovery_rate * 100, 2), "%\n")
```

```
## false_discovery_rate 32.77 %
```

```r
cat("false_omission_rate", round(false_omission_rate * 100, 2), "%\n")
```

```
## false_omission_rate 18.73 %
```

```r
cat("missclassification_error", round(missclassification_error * 100, 2), "%\n")
```

```
## missclassification_error 27.91 %
```

```r
#cat("F1 Score", round(f1 * 100, 2), "%\n")
```

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

We want to minimize the number of people that we predict will pay but do not pay. Therefore we want to minimise the FP or the false positives in which we pridct such people not paying. By assigning weights, we can state that the false positives are more heavy for each false negative. Here we are seeing that about 30% of all people you are predicting are gonna pay back do not pay back.

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

```r
wfp = 165
wfn = 1

weight <- wfp * fp + wfn * fn
weight
```

```
## [1] 2005448
```

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

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

```
##
## Call:  glm(formula = paid_in_full ~ ., family = binomial(link = "logit"),
##     data = bills_data_train)
##
## Coefficients:
##       (Intercept)         tot_amount  discount_id5693147  discount_id6098612
##        -4.720e+00          1.087e-05           4.169e+00          -1.394e+01
## discount_id6609438  discount_id6945910  discount_id7079442  discount_id7197225
##        -1.393e+01           3.861e+00          -8.818e-01          -9.575e-01
## discount_id7302585  discount_id7397895  discount_id7484907  discount_id7564949
##        -1.051e+00          -3.865e-01          -1.394e+01           2.327e+00
## discount_id7708050  discount_id7890372  discount_id7944439  discount_id7995732
##         8.360e-01          -1.392e+01           4.972e+00           2.119e+01
## discount_id8091042  discount_id8178054  discount_id8218876  discount_id8258097
##        -1.394e+01          -1.394e+01          -3.357e+00           5.015e+00
## discount_id8367296  discount_id8401197  discount_id8433987  discount_id8713572
##         3.801e+00          -1.393e+01          -1.393e+01          -1.394e+01
## discount_id8737670  discount_id8784190  discount_id8806662  discount_id8828641
##        -1.393e+01          -1.393e+01           4.147e+00          -1.396e+01
## discount_id8850148  discount_id8871201  discount_id9043051  discount_id9060443
##        -2.036e-01          -1.394e+01          -1.394e+01           2.941e+00
## discount_id9077537  discount_id9094345     num_days_to_pay            bill_num
##         2.896e+00          -1.394e+01           4.920e-04          -1.842e-05
##
## Degrees of Freedom: 168841 Total (i.e. Null);  168806 Residual
##   (984 observations deleted due to missingness)
## Null Deviance:       234100
## Residual Deviance: 186400    AIC: 186400
```

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.

```
#From notes
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

}

train_p_hat = predict(mod, bills_data_train, type = "response")
y_real = bills_data_train$paid_in_full
classifier_metric_is = compute_metrics_prob_classifier(train_p_hat, y_real)

test_p_hat = predict(mod, bills_data_test, type = "response")
```

```
y_real = bills_data_test$paid_in_full
classifier_metric_oos = compute_metrics_prob_classifier(test_p_hat, y_real)
```

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

```
classifier_table_is = as_tibble(classifier_metric_is) %>% mutate(total_cost = wfp * FP + wfn * FN)
classifier_table_is
```

```
## # A tibble: 999 x 13
##     p_th    TN    FP    FN    TP miscl_err precision recall   FDR   FPR      FOR
##    <dbl> <dbl> <dbl> <dbl> <dbl>     <dbl>     <dbl>  <dbl> <dbl> <dbl>    <dbl>
##  1 0.001 10853 72798     2 85189     0.429     0.539  1.00  0.461 0.870 0.000184
##  2 0.002 10853 72798     2 85189     0.429     0.539  1.00  0.461 0.870 0.000184
##  3 0.003 10853 72798     2 85189     0.429     0.539  1.00  0.461 0.870 0.000184
##  4 0.004 10853 72798     2 85189     0.429     0.539  1.00  0.461 0.870 0.000184
##  5 0.005 10853 72798     2 85189     0.429     0.539  1.00  0.461 0.870 0.000184
##  6 0.006 10853 72798     2 85189     0.429     0.539  1.00  0.461 0.870 0.000184
##  7 0.007 10853 72798     2 85189     0.429     0.539  1.00  0.461 0.870 0.000184
##  8 0.008 10854 72797     2 85189     0.429     0.539  1.00  0.461 0.870 0.000184
##  9 0.009 13324 70327     2 85189     0.414     0.548  1.00  0.452 0.841 0.000150
## 10 0.01  20184 63467    89 85102     0.374     0.573  0.999 0.427 0.759 0.00439
## # ... with 989 more rows, and 2 more variables: miss_rate <dbl>,
## #   total_cost <dbl>
```

```
classifier_table_oos = as_tibble(classifier_metric_oos) %>% mutate(total_cost = wfp * FP + wfn * FN)
classifier_table_oos
```

```
## # A tibble: 999 x 13
##     p_th    TN    FP    FN    TP miscl_err precision recall   FDR   FPR     FOR
##    <dbl> <dbl> <dbl> <dbl> <dbl>     <dbl>     <dbl>  <dbl> <dbl> <dbl>   <dbl>
##  1 0.001  3494 24237     5 28543     0.428     0.541  1.00  0.459 0.874 0.00143
##  2 0.002  3494 24237     5 28543     0.428     0.541  1.00  0.459 0.874 0.00143
##  3 0.003  3494 24237     5 28543     0.428     0.541  1.00  0.459 0.874 0.00143
##  4 0.004  3494 24237     5 28543     0.428     0.541  1.00  0.459 0.874 0.00143
##  5 0.005  3494 24237     5 28543     0.428     0.541  1.00  0.459 0.874 0.00143
##  6 0.006  3494 24237     5 28543     0.428     0.541  1.00  0.459 0.874 0.00143
##  7 0.007  3494 24237     5 28543     0.428     0.541  1.00  0.459 0.874 0.00143
##  8 0.008  3494 24237     5 28543     0.428     0.541  1.00  0.459 0.874 0.00143
##  9 0.009  4335 23396     5 28543     0.413     0.550  1.00  0.450 0.844 0.00115
## 10 0.01   6671 21060    39 28509     0.373     0.575  0.999 0.425 0.759 0.00581
## # ... with 989 more rows, and 2 more variables: miss_rate <dbl>,
## #   total_cost <dbl>
```

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

```
threshold = min(classifier_table_is[which.min(classifier_table_is$total_cost), ]$total_cost)
threshold
```

```
## [1] 85960
```

```r
threshold = min(classifier_table_oos[which.min(classifier_table_oos$total_cost), ]$total_cost)
threshold
```

```
## [1] 28861
```

```r
classifier_table_oos[which.min(classifier_table_oos$total_cost), ]
```

```
## # A tibble: 1 x 13
##    p_th    TN    FP    FN    TP miscl_err precision   recall   FDR     FPR   FOR
##   <dbl> <dbl> <dbl> <dbl> <dbl>     <dbl>     <dbl>    <dbl> <dbl>   <dbl> <dbl>
## 1 0.959 27729     2 28531    17     0.504     0.895 0.000595 0.105 7.21e-5 0.507
## # ... with 2 more variables: miss_rate <dbl>, total_cost <dbl>
```

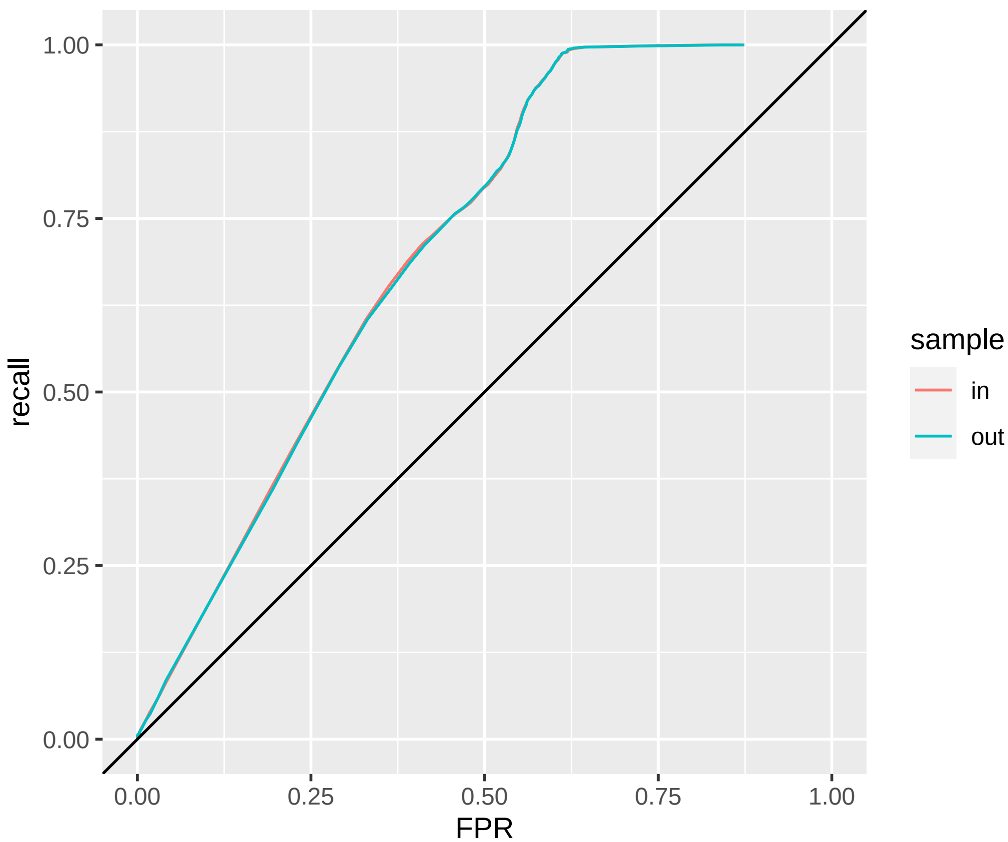
Plot an ROC curve and interpret.

```r
pacman::p_load(ggplot2)

classifier_table_is_oos = rbind(
    cbind(classifier_table_is, data.table(sample="in")),
    cbind(classifier_table_oos, data.table(sample="out"))
)

ggplot(classifier_table_is_oos) +
    geom_line(aes(x = FPR, y = recall, col= sample)) +
    geom_abline(inline = 0, slope = 1) +
    coord_fixed() +
    xlim(0, 1) +
    ylim(0, 1)
```

```
## Warning: Ignoring unknown parameters: inline
```



We take the ROS to compare probability estimation models by calculating the area under the curve to measure the models predictive power.

Calculate AUC and interpret.

```
pacman::p_load(pracma)
auc_in = -trapz(classifier_table_is$FPR, classifier_table_is$recall)
cat("AUC in-sample: ", auc_in, "\n")
```

```
## AUC in-sample:  0.580456
```

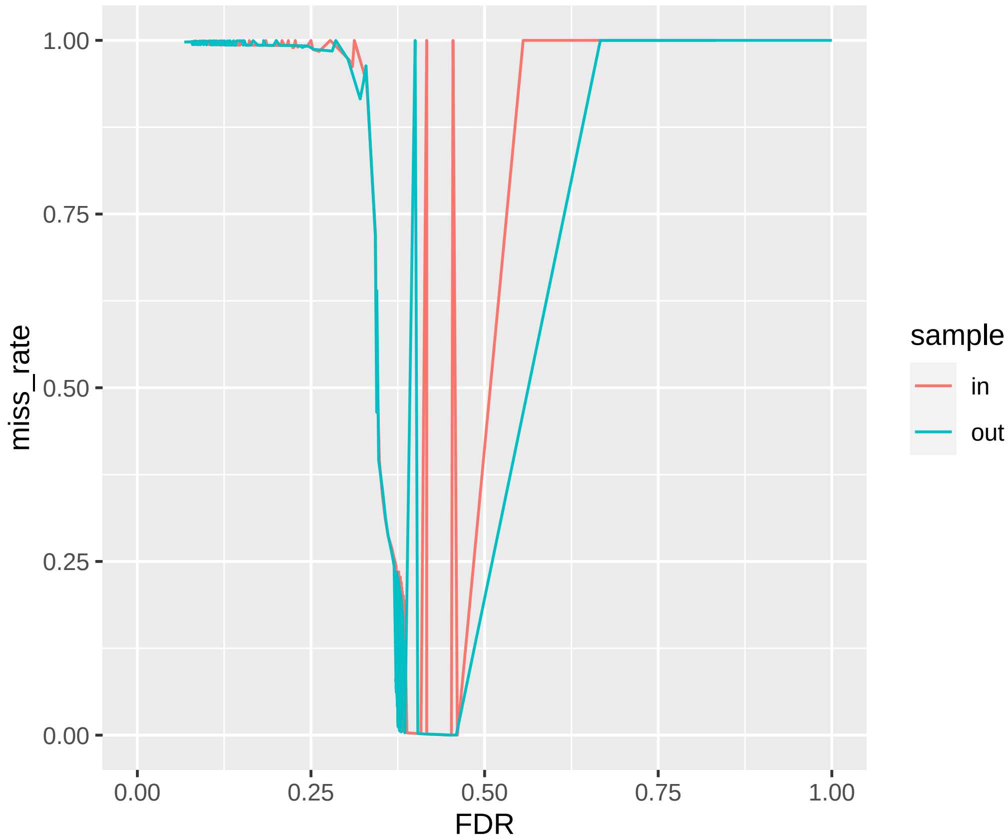
```
auc_oos = -trapz(classifier_table_oos$FPR, classifier_table_oos$recall)
cat("AUC oos-sample: ", auc_oos)
```

```
## AUC oos-sample:  0.5830701
```

The AUC values are really similar to the nearest thousandth. Because the AUC is larger than 0.5, it shows that the model has more predictive power.

Plot a DET curve and interpret.

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



The DUC is defined by the pth. There are multiple classification models which allows us to visualize for multiple models. The graph indicates that the FDR gets the FOR to nearly 0 while repetitively fluctuating at distinct points for both the in sample and out of sample.