Lab 8

Jacob Minkin

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)
```

```
## YARF can now make use of 3 cores.
```

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>0)
```

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())
```

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".

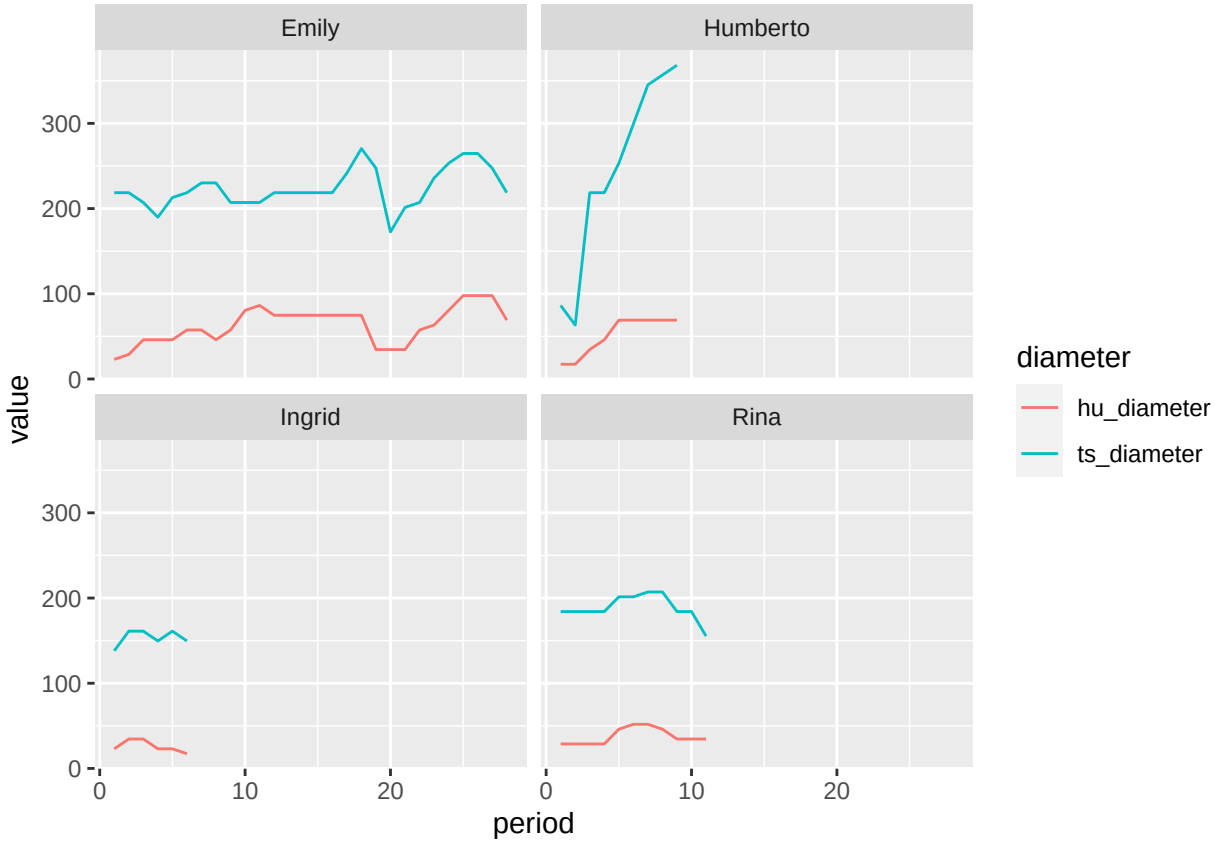
```
storms_long = pivot_longer(storms2, cols = matches("diameter"), names_to = "diameter")
storms_long
```

```
## # 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")
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" = "
```

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_payments_amount = sum(paid_amount)) %>%
  mutate(paid_in_full = as.factor( if_else( sum_of_payments_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.

```
pacman::p_load(lubridate)
bills_data = bills_data %>%
  select(-id, -id.y, -days_until_discount, -num_days, -transaction_date, -pct_off, -sum_of_payments_amou
  mutate(num_days_to_pay = as.integer(ymd(due_date) - ymd(invoice_date)) ) %>%
  select(-invoice_date, -due_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())
```

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.

```
y_train = bills_data_train$paid_in_full
X_train = bills_data_train
X_train$paid_in_full = NULL
n_train = nrow(X_train)
options(java.parameters = "-Xmx4000m")
pacman::p_load(YARF)
tree_mod = YARFCART(X_train, y_train,
            bootstrap_indices = list(1 : n_train), calculate_oob_error = FALSE)
```

```
## YARF initializing with a fixed 1 trees...
## YARF factors created...
## YARF after data preprocessed... 36 total features...
## Beginning YARF classification model construction...done.
```

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(tree_mod)
```

```
## $num_nodes
## [1] 70035
##
## $num_leaves
```

```
## [1] 35018
##
## $max_depth
## [1] 231
```

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

```
illustrate_trees(tree_mod, max_depth = 5, length_in_px_per_half_split = 30, open_file = TRUE)
```

Predict on the test set and compute a confusion matrix.

```
y_test = bills_data_test$paid_in_full
X_test = bills_data_test
X_test$paid_in_full = NULL

y_hat_test_tree = predict(tree_mod, X_test)

y_hats_test = ifelse(y_hat_test_tree != y_test, 0, 1)
mean(y_hat_test_tree != y_test)
```

```
## [1] 0.2264521
```

```
oos_conf_table = table(y_test, y_hats_test)
oos_conf_table
```

```
##       y_hats_test
## y_test     0     1
##      0  6263 21900
##      1  6556 21889
```

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

```
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 = y_test - y_hats_test
#cat("misclassifcation error", round(misclassifcation * 100, 2), "%\n")
precision = tp / num_pred_pos
cat("precision", round(precision * 100, 2), "%\n")
```

```
## precision 49.99 %
```

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

```
## recall 76.95 %
```

```
F_1 = 2/((1/recall)+(1/precision))
cat("F_1", round(F_1 * 100, 2), "%\n")
```

```
## F_1 60.61 %
```

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

```
## false_discovery_rate 50.01 %
```

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

```
## false_omission_rate 51.14 %
```

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

No! It is just as good as guessing!

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

```
C_FP = 1; C_FN = 5
cost = (C_FP * fp) + (C_FN * fn)
```

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

```
logistic_mod = glm(paid_in_full ~ ., bills_data_train, family = "binomial")

p_hats_train = predict(logistic_mod, bills_data_train, type = "response")
y_hats_train = ifelse(p_hats_train >= 0.5, 0, 1)
```

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.

```
#' Computes performance metrics for a binary probabilistic classifer
#'
#' Each row of the result will represent one of the many models and its elements record the performance
#'
#' @param p_hats  The probability estimates for n predictions
#' @param y_true  The true observed responses
#' @param res     The resolution to use for the grid of threshold values (defaults to 1e-3)
#'
#' @return        The matrix of all performance results
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, 0, 1))
    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
}

pacman::p_load(data.table, magrittr)
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 72924 10632 85303     0 0.5649017  0.000000 0.0000000 1.000000
##   2: 0.002 72924 10632 85303     0 0.5649017  0.000000 0.0000000 1.000000
##   3: 0.003 72924 10632 85303     0 0.5649017  0.000000 0.0000000 1.000000
##   4: 0.004 72924 10632 85303     0 0.5649017  0.000000 0.0000000 1.000000
##   5: 0.005 72924 10632 85303     0 0.5649017  0.000000 0.0000000 1.000000
##  ---
## 995: 0.995     6 83550     2 85301 0.4919859  0.505185 0.9999766 0.494815
## 996: 0.996     6 83550     2 85301 0.4919859  0.505185 0.9999766 0.494815
## 997: 0.997     6 83550     2 85301 0.4919859  0.505185 0.9999766 0.494815
## 998: 0.998     6 83550     2 85301 0.4919859  0.505185 0.9999766 0.494815
## 999: 0.999     6 83550     1 85302 0.4919800  0.505188 0.9999883 0.494812
```

```
##            FPR       FOR    miss_rate
##   1: 0.1272440 0.5391178 1.000000e+00
##   2: 0.1272440 0.5391178 1.000000e+00
##   3: 0.1272440 0.5391178 1.000000e+00
##   4: 0.1272440 0.5391178 1.000000e+00
##   5: 0.1272440 0.5391178 1.000000e+00
##  ---
## 995: 0.9999282 0.2500000 2.344583e-05
## 996: 0.9999282 0.2500000 2.344583e-05
## 997: 0.9999282 0.2500000 2.344583e-05
## 998: 0.9999282 0.2500000 2.344583e-05
## 999: 0.9999282 0.1428571 1.172292e-05
```

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

```
performance_metrics_in_sample = performance_metrics_in_sample %>%
  mutate(total_cost = (C_FP * FP) + (C_FN * FN) )

performance_metrics_in_sample
```

```
##       p_th    TN    FP    FN    TP miscl_err precision    recall      FDR
##   1: 0.001 72924 10632 85303     0 0.5649017  0.000000 0.0000000 1.000000
##   2: 0.002 72924 10632 85303     0 0.5649017  0.000000 0.0000000 1.000000
##   3: 0.003 72924 10632 85303     0 0.5649017  0.000000 0.0000000 1.000000
##   4: 0.004 72924 10632 85303     0 0.5649017  0.000000 0.0000000 1.000000
##   5: 0.005 72924 10632 85303     0 0.5649017  0.000000 0.0000000 1.000000
##  ---
## 995: 0.995     6 83550     2 85301 0.4919859  0.505185 0.9999766 0.494815
## 996: 0.996     6 83550     2 85301 0.4919859  0.505185 0.9999766 0.494815
## 997: 0.997     6 83550     2 85301 0.4919859  0.505185 0.9999766 0.494815
## 998: 0.998     6 83550     2 85301 0.4919859  0.505185 0.9999766 0.494815
## 999: 0.999     6 83550     1 85302 0.4919800  0.505188 0.9999883 0.494812
##            FPR       FOR    miss_rate total_cost
##   1: 0.1272440 0.5391178 1.000000e+00     437147
##   2: 0.1272440 0.5391178 1.000000e+00     437147
##   3: 0.1272440 0.5391178 1.000000e+00     437147
##   4: 0.1272440 0.5391178 1.000000e+00     437147
##   5: 0.1272440 0.5391178 1.000000e+00     437147
##  ---
## 995: 0.9999282 0.2500000 2.344583e-05      83560
## 996: 0.9999282 0.2500000 2.344583e-05      83560
## 997: 0.9999282 0.2500000 2.344583e-05      83560
## 998: 0.9999282 0.2500000 2.344583e-05      83560
## 999: 0.9999282 0.1428571 1.172292e-05      83555
```

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

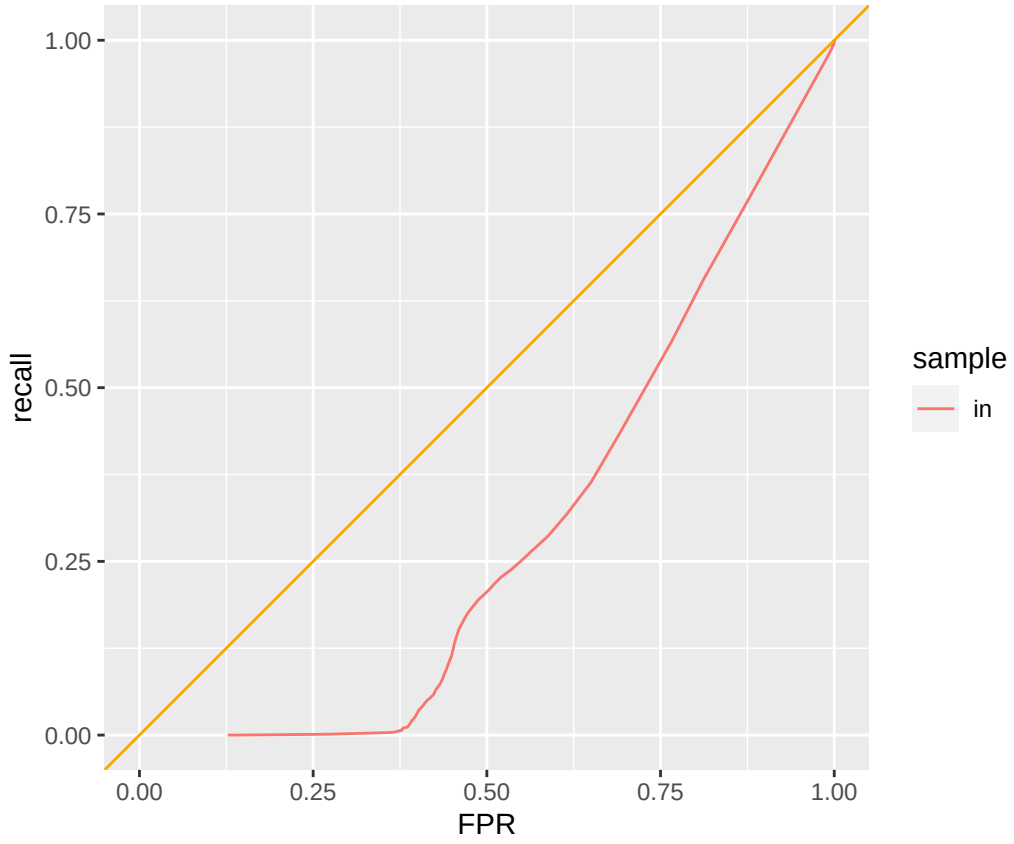
```
performance_metrics_in_sample %>%
  arrange(total_cost) %>%
  slice(1) %>%
  select(p_th, total_cost)
```

```
##     p_th total_cost
## 1: 0.999      83555
```

Plot an ROC curve and interpret.

```
performance_metrics_in_sample_long = cbind(performance_metrics_in_sample, data.table(sample = "in"))

ggplot(performance_metrics_in_sample_long) +
  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)
```



VERY BAD!

Calculate AUC and interpret.

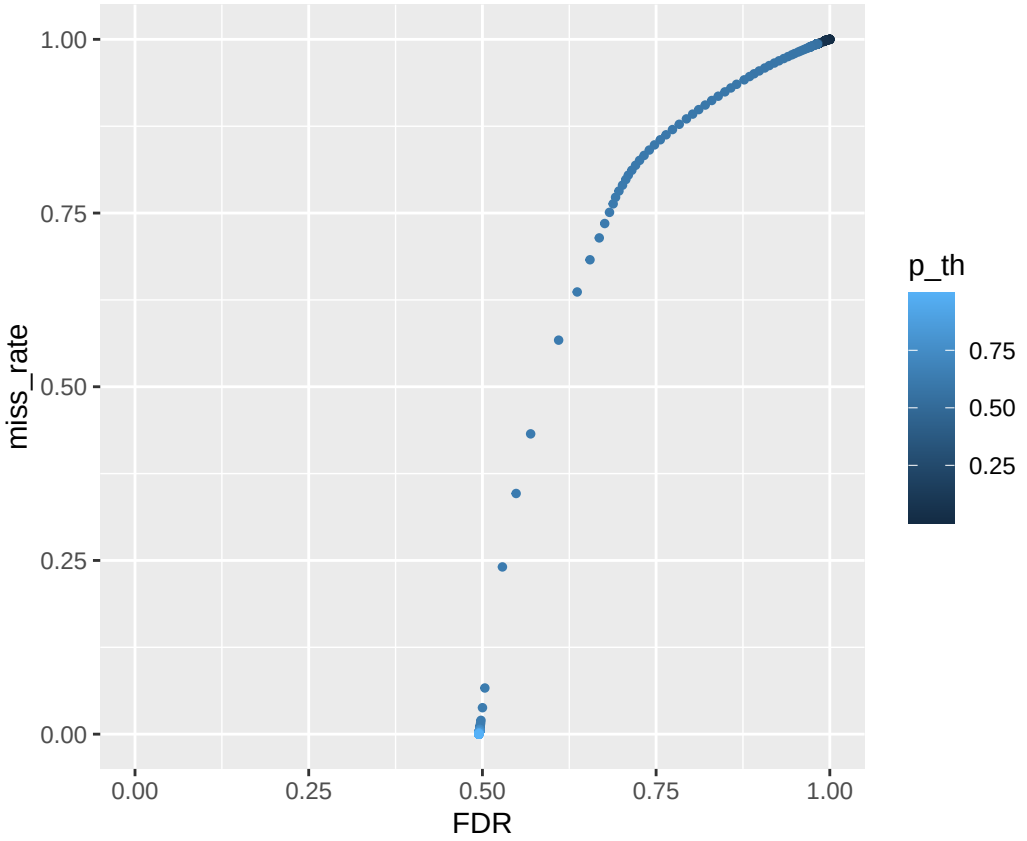
```
pacman::p_load(pracma)
-trapz(performance_metrics_in_sample$FPR, performance_metrics_in_sample$recall)
```

```
## [1] -0.2915431
```

This is very bad. This is worse than guessing.

Plot a DET curve and interpret.

```
ggplot(performance_metrics_in_sample_long) +
  geom_point(aes(x = FDR, y = miss_rate,  col = p_th), size = 1) +
  coord_fixed() + xlim(0, 1) + ylim(0, 1)
```



No matter what you geta at least a 50% FDR! Your specificity will always be less than 50%. This means that this model is useless!