Lab 8

Amir ElTabakh

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)
# Keep this cell commented out, it breaks R if you do not have the right software installed.
```

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)

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
```

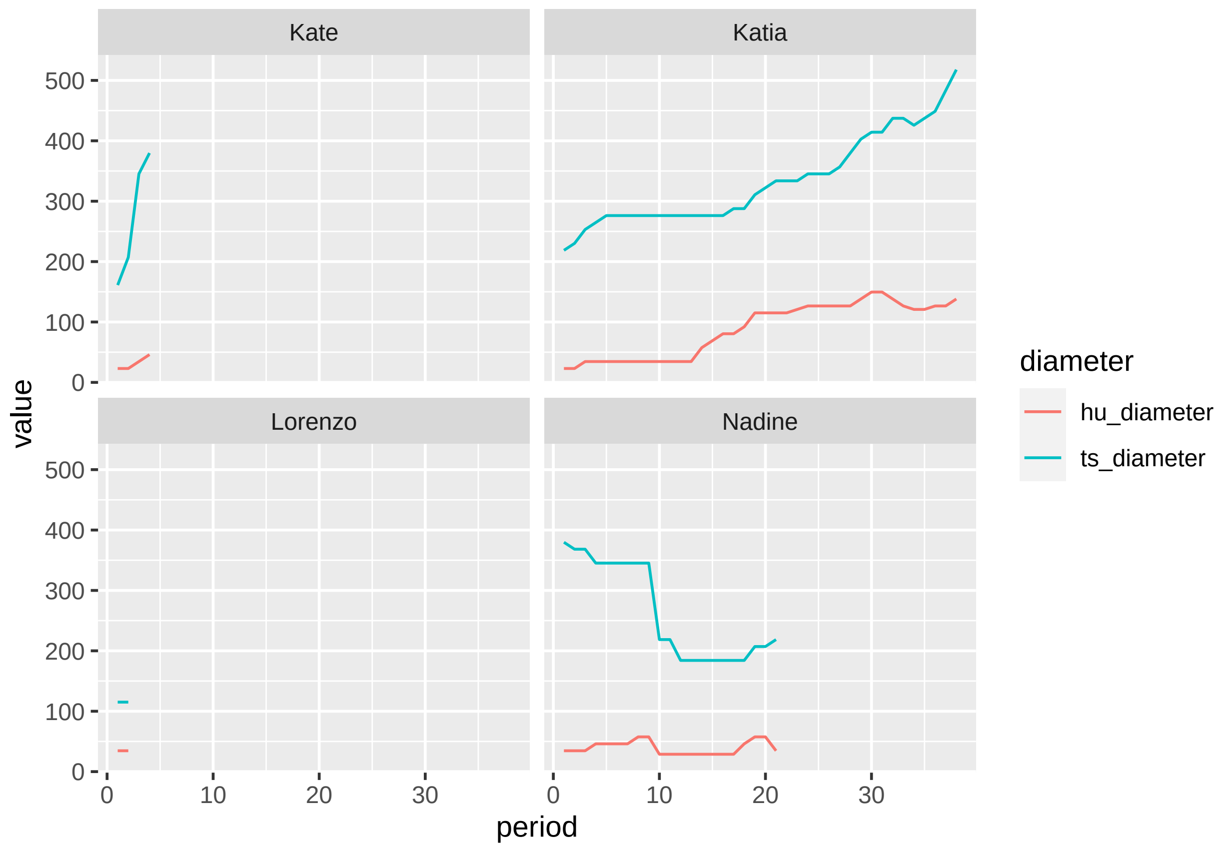
```
## # A tibble: 2,044 x 4
## # Groups:   name [63]
##    name  period diameter    value
##    <chr>  <int> <chr>       <dbl>
##  1 Alex       1 ts_diameter 150.
##  2 Alex       1 hu_diameter  46.0
##  3 Alex       2 ts_diameter 150.
##  4 Alex       2 hu_diameter  46.0
##  5 Alex       3 ts_diameter 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)
storms_sample
```

```
## [1] "Lorenzo" "Kate"    "Katia"   "Nadine"
```

```r
pacman::p_load(ggplot2)
ggplot(data = 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:

```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
```

```r
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
```

```r
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
```

```r
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.

```r
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" = "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> <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!

```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_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.

```r
pacman::p_load("lubridate")

bills_data = bills_data %>%
  select(-id, -id.y, -num_days, -transaction_date, -pct_off, -days_until_discount, -sum_of_payment_amou
  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())

bills_data[order(-bills_data$bill_num), ]
```

```
## # A tibble: 226,434 x 4
##    tot_amount num_days_to_pay bill_num paid_in_full
##         <dbl>           <int>    <int>        <dbl>
##  1     99556.              30    14691            0
##  2     99528.              45    14690            0
##  3     99968.              30    14689            0
##  4    100830.               0    14688            1
##  5     99475.               0    14687            0
```

```
## 6     105252.               30     14686           1
## 7     105375.                0     14685           0
## 8      99526.               30     14684           0
## 9      99483.               60     14683           0
## 10     99478.               30     14682           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.

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

```
## n= 169826
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 169826 84575 1 (0.49800973 0.50199027)
##    2) tot_amount>=99025.92 64091 19382 0 (0.69758624 0.30241376)
##      4) bill_num< 1087.5 32821  2094 0 (0.93619938 0.06380062) *
##      5) bill_num>=1087.5 31270 13982 1 (0.44713783 0.55286217)
##       10) tot_amount< 99476.96 13255  3941 0 (0.70267823 0.29732177) *
##       11) tot_amount>=99476.96 18015  4668 1 (0.25911740 0.74088260) *
##    3) tot_amount< 99025.92 105735 39866 1 (0.37703693 0.62296307)
##      6) bill_num>=1242.5 20014  9878 0 (0.50644549 0.49355451)
##       12) bill_num< 3063.5 14047  5882 0 (0.58126290 0.41873710) *
##       13) bill_num>=3063.5 5967  1971 1 (0.33031674 0.66968326) *
##      7) bill_num< 1242.5 85721 29730 1 (0.34682283 0.65317717) *
```

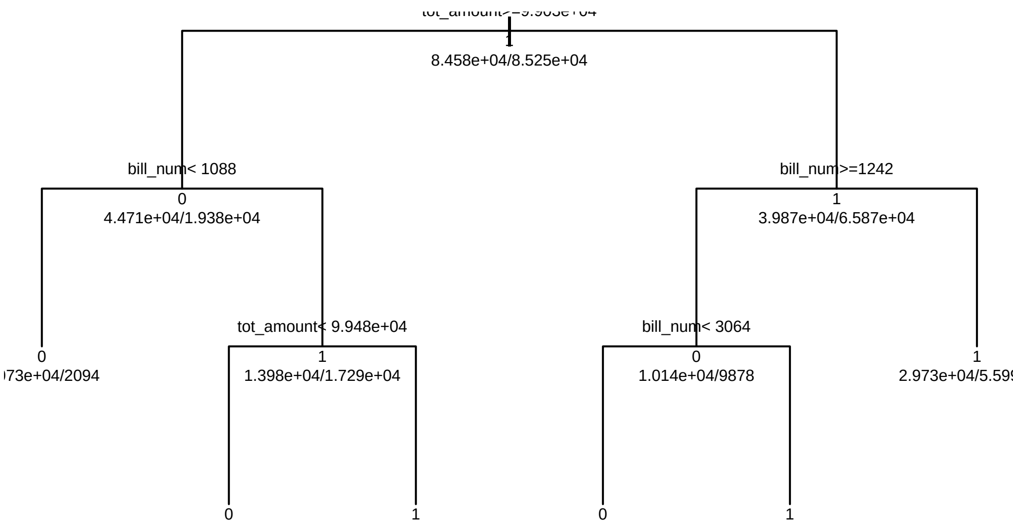
For those of you who installed `YARF`, what are the number of nodes and depth of the tree?

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

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

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

```r
plot(mod1, uniform=TRUE)
text(mod1, use.n=TRUE, all=TRUE, cex=0.5)
```




```r
# This line only helps find the depth of the tree if the tree were balanced.
ceiling(log(nrow(mod1$frame), 2))
```

```
## [1] 4
```

Predict on the test set and compute a confusion matrix.

```r
yhat = predict(mod1, 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 15947 12142
##   1  3917 24602
```

^ Columns are yhats, and the rows are the actuals.

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

```r
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 28.37 %
```

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

```
## precision 66.96 %
```

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

```
## recall 86.27 %
```

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

```
## false_discovery_rate 33.04 %
```

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

```
## false_omission_rate 19.72 %
```

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

```
## F1 score 75.39 %
```

I believe a good metric to judge this model by in our context is false discovery rate. False discovery rate is defined as the number of times the model predicted an individual would pay their bill and the individual did not pay their bill, over the total number of times the model predicted an individual would pay their bill, or false positives over predicted positives. A false discovery rate of 33.06% means that the model wrongly predicts for an individual to pay back their bill 1 in 3 times. A company might find such a model useless, I judge this is a bad model.

There are probability asymmetric costs to the two types of errors. Assign the costs below and calculate oos total cost. Its more important that fp stays low, so it costs more to have an fp. I arbitrarily say that 30 false negatives is equivalent to 1 false positive. Its okay to predict someone to not pay and they pay, however its damaging to the business to predict for people to pay and they do not

```r
C_fp <- 30
C_fn <- 1
cost = C_fp * fp + C_fn * fn
cost
```

```
## [1] 368177
```

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

```r
logistic_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.

```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
    )
  }

  performance_metrics

}

y_train = bills_data_train$paid_in_full
p_hats_train = predict(logistic_mod, bills_data_train, type = "response")
metric_prob_classifier_tibble_in_sample = compute_metrics_prob_classifier(p_hats_train, y_train) %>% da

y_test = bills_data_test$paid_in_full
p_hats_test = predict(logistic_mod, bills_data_test, type = "response")
metric_prob_classifier_tibble_oos = compute_metrics_prob_classifier(p_hats_test, y_test) %>% data.table
```

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

```
C_fp <- 50
C_fn <- 1

metric_prob_classifier_tibble_in_sample <- metric_prob_classifier_tibble_in_sample %>%
  mutate(total_cost = (C_fp * FP) + (C_fn * FN))

metric_prob_classifier_tibble_oos <- metric_prob_classifier_tibble_oos %>%
  mutate(total_cost = (C_fp * FP) + (C_fn * FN))
```

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

```
best_prob_threshold_index_in_sample = which.min(metric_prob_classifier_tibble_in_sample$total_cost)
best_prob_threshold_metrics_in_sample = metric_prob_classifier_tibble_in_sample[best_prob_threshold_ind

cat("The total cost of the winning probability threshold in sample is", min(best_prob_threshold_metrics
```

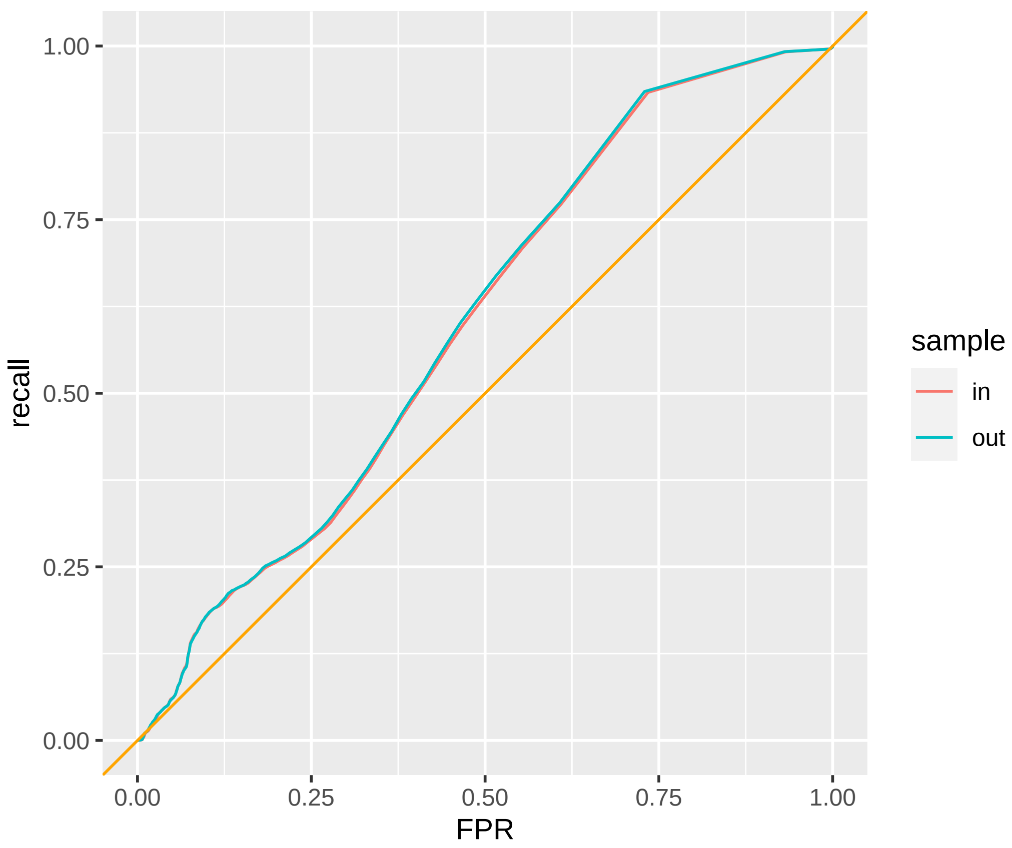
```
## The total cost of the winning probability threshold in sample is 85251
```

```
best_prob_threshold_index_oos = which.min(metric_prob_classifier_tibble_oos$total_cost)
best_prob_threshold_metrics_oos = metric_prob_classifier_tibble_oos[best_prob_threshold_index_oos, ]

cat("\n\nThe total cost of the winning probability threshold OOS is", min(best_prob_threshold_metrics_o
```

```
##
##
## The total cost of the winning probability threshold OOS is 28519
```

Plot an ROC curve and interpret.

```
pacman::p_load(ggplot2)
performance_metrics_in_and_oos = rbind(
    cbind(metric_prob_classifier_tibble_in_sample, data.table(sample = "in")),
    cbind(metric_prob_classifier_tibble_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 can be used to compares the models recall against its false positive rate at various threshold settings. The orange line is the generic model, where the true y values are independent of the false positive rates at all threshold settings. Because the ROC curve is above the generic model curve, we can say that the model performs better than the generic model.

Calculate AUC and interpret.

```
pacman::p_load(pracma)
auc_in_sample = -trapz(metric_prob_classifier_tibble_in_sample$FPR, metric_prob_classifier_tibble_in_sar
cat("AUC in-sample: ", auc_in_sample)
```

```
## AUC in-sample:  0.599748
```

```
auc_oos = -trapz(metric_prob_classifier_tibble_oos$FPR, metric_prob_classifier_tibble_oos$recall)
cat("\n\nAUC OOS: ", auc_oos)
```

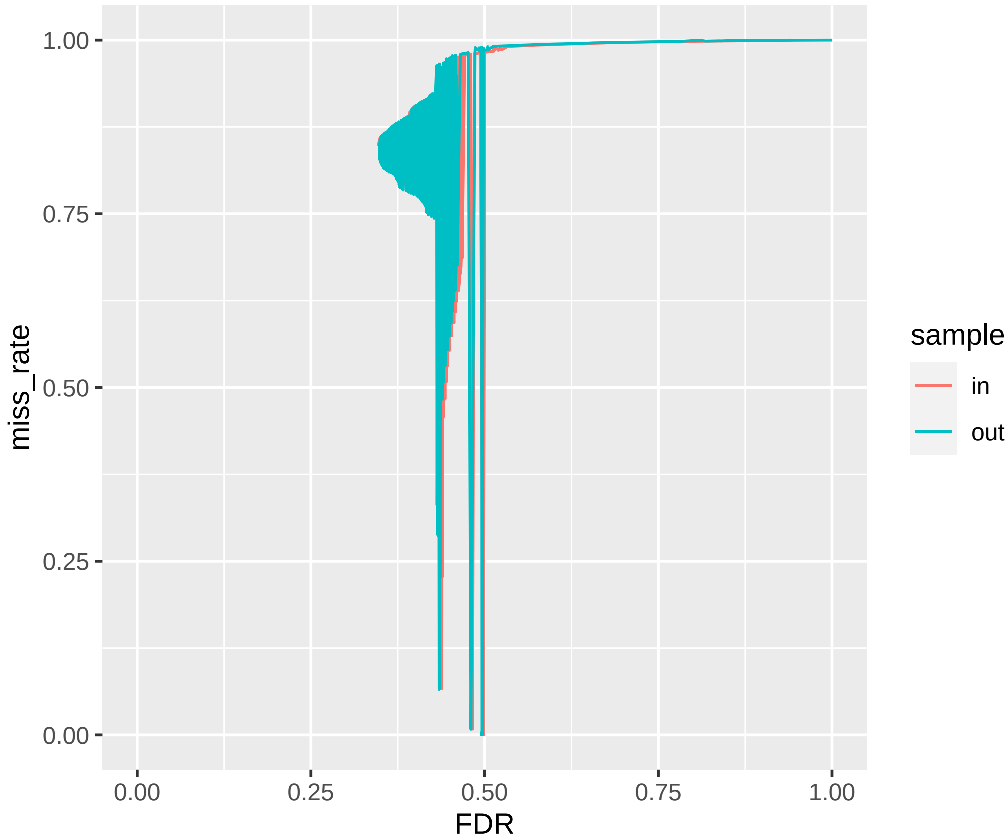
```
##
##
## AUC OOS:  0.6032098
```

AUC is a metric that gauges the overall fit of a probility estimation model. The area under the generic model curve is exactly 0.5. The area under the OOS curve is about 0.59, that means that it does indeed perform better than the generic model. An AUC greater that 0.5 indicates that the model has predictive power. AUC's closer to 1 indicate a better 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 593 row(s) containing missing values (geom_path).
```



The DET is traced out by varying p_th in [0,1]. Thhis 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.