Emotional Learning Analytics

INFO 5200 Learning Analytics Homework

[[Kimberly Williamson, khw44]]

In this homework, you will learn how to build a basic sensor-free affect detector. You are given ASSISTments data enhanced with coded affect data. The goal is for you to engineer features and predict affect as best as you can.

Learning Objectives

1. Engineer features that can detect affect in a dataset
2. Train a Random Forest model to identify boredom
3. Make recommendations to teachers based on the features that are important.

Data

The dataset contains information for 250 students at several schools with many teachers. The students were using the Assistments platform for learning mathematics and the granularity of the data is at the student-problem level (like the data in the first homework). Some more information on this data before I pre-processed it is available here: https://sites.google.com/site/assistmentsdata/home/2012-13-school-data-with-affect

Variable	Data Type	Definition
user_id, teacher_id, school_id, problem_id, skill_id	numeric	Unique identifiers
frustrated, confused, concentrating, bored	numeric	Indicator of coded affective state (1=present)
correct	numeric	Correct on first attempt
ms_first_response	numeric	Milliseconds until first response submitted
hint_count	numeric	Number of hints student asked for
attempt_count	numeric	Number of attempts until correct
user_event_index	numeric	For each user, a running index of events (first=1, second=2 … last)
time_spent	numeric	Seconds spent on problem overall

Exploring the Data

Before starting to answer any questions, take some time to understand the structure of the dataset. The block below will not be evaluated in the knitted report (eval=F). You can use this space to try out different approaches to explore the data and test your understanding of it.

```
head(a)
summary(a)
n_distinct(a$skill_id)
hist(table(a$user_id))
```

Part 1. Feature Engineering

Come up with features that are likely to predict boredom. Think of a time when you were learning something and you felt bored. What were you doing? How might this show up in this dataset. You should check out this paper which engineers features with a very similar dataset. The dataset you are working with is less detailed though; otherwise this would take too long to train: https://learning-analytics.info/journals/index.php/JLA/article/view/3536/4014

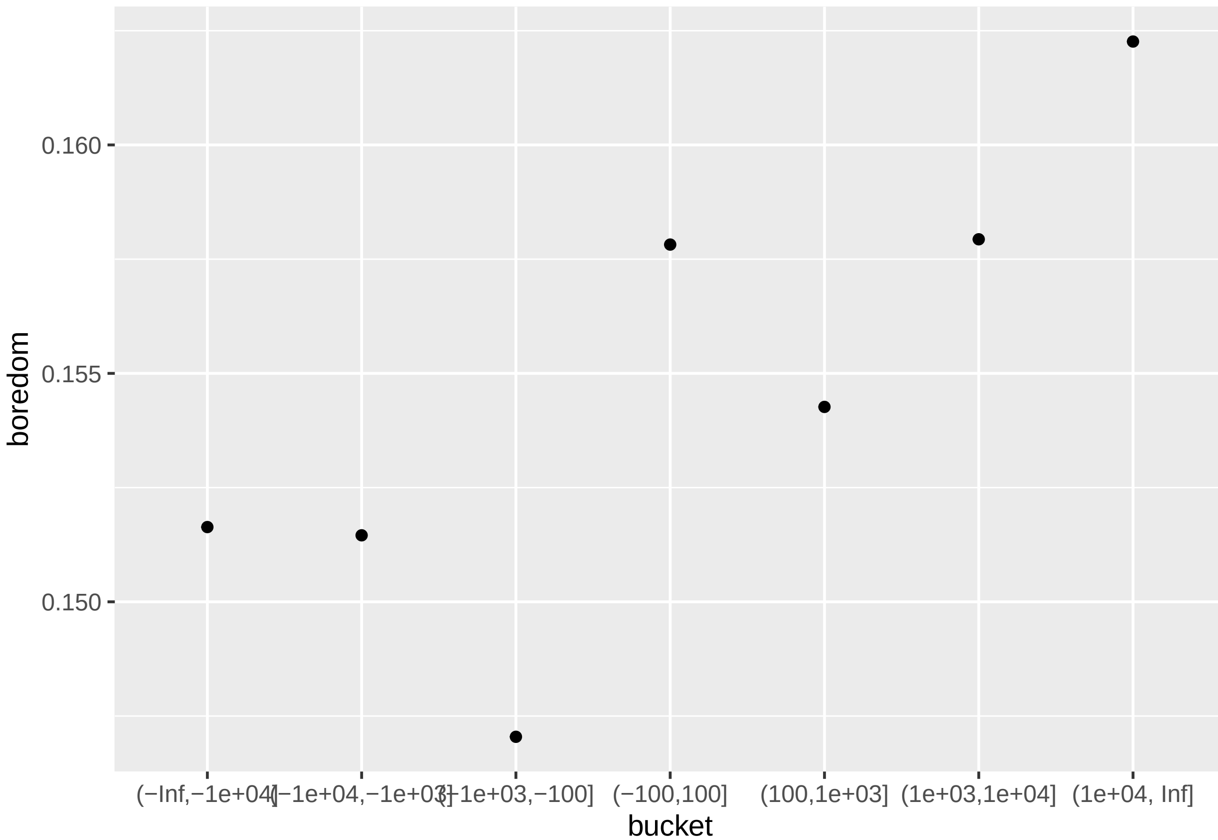
You will try out fitting random forest models using the `randomForest()` function. Note that the full dataset is quite large (the original one online is even bigger!), so you will want to experiment with a smaller, representative subset at the beginning. So Question 1 asks you to pair it down for now.

Instructor example: I hypothesize that students who answer a question fast and move on are not bored (yet); students who take a long time to figure out what the answer may be and don't submit are probably bored. Students in the middle of the distribution maybe a mix of the two. I first check this idea using a plot and then code a feature to capture the relationship.

```
# Inspecting the correlation
base_vars = c("bored","correct","ms_first_response","hint_count","attempt_count","user_event_index","ti
cor(a[,base_vars])
```

```
##                           bored     correct ms_first_response  hint_count
## bored              1.0000000000 -0.01309956     -0.0004095708  0.01894448
## correct           -0.0130995650  1.00000000     -0.0104848987 -0.50191069
## ms_first_response -0.0004095708 -0.01048490      1.0000000000  0.01734008
## hint_count         0.0189444794 -0.50191069      0.0173400793  1.00000000
## attempt_count      0.0079488414 -0.46616983      0.0089602507  0.32317260
## user_event_index  -0.0050735506  0.03017543     -0.0038824353 -0.03477949
## time_spent         0.0009631775 -0.04356123      0.0101244916  0.03571154
##                   attempt_count user_event_index    time_spent
## bored               0.007948841     -0.005073551  0.0009631775
## correct            -0.466169830      0.030175430 -0.0435612346
## ms_first_response   0.008960251     -0.003882435  0.0101244916
## hint_count          0.323172596     -0.034779493  0.0357115396
## attempt_count       1.000000000     -0.020391456  0.0667177477
## user_event_index   -0.020391456      1.000000000 -0.0014104681
## time_spent          0.066717748     -0.001410468  1.0000000000
```

```
### INSTRUCTOR EXAMPLE ###
a %>%
    group_by(user_id) %>%
    mutate(
        # Diff in response time to the typical response time
        diff = ms_first_response - median(ms_first_response)
    ) %>%
    group_by(
        # Group difference into 5 buckets
        bucket = cut(diff, c(-Inf,-10000, -1000, -100, 100, 1000, 10000, Inf))
    ) %>%
    summarise(
        # get the average prevalence of boredom in each bucket
        boredom = mean(bored)
    ) %>%
    ggplot(aes(x=bucket, y=boredom)) + geom_point() # plot it
```




```r
a = a %>%
    group_by(user_id) %>% # median will be relative to student
    mutate(
        rel_resp_time = ms_first_response - median(ms_first_response),
        slow_response = as.integer(rel_resp_time < -1000),
        fast_response = as.integer(rel_resp_time > 10000)
    ) %>%
    ungroup

# Checking correlations
cor(a[,c("bored","rel_resp_time","slow_response","fast_response")])
```

```
##                       bored rel_resp_time slow_response fast_response
## bored          1.0000000000 -0.0005368955   -0.01207557    0.01267754
## rel_resp_time -0.0005368955  1.0000000000   -0.03465763    0.04302525
## slow_response -0.0120755720 -0.0346576338    1.00000000   -0.71785739
## fast_response  0.0126775388  0.0430252525   -0.71785739    1.00000000
```

Question 1: Following the structure of the example above, inspect correlations between the `bored` variable (the outcome you want to predict) and learning logs in the dataset, and then create your own features for predicting boredom. Be sure to add **at least 7 new features**. (The authors of the paper linked above created over 170 features!) Do not use the other affect data as features.

```r
#######################################
####### BEGIN INPUT: Question 1 #######
#######################################

# Add your own exploration and feature engineering here (make at least 7 new features)
```

```r
a %>% group_by(user_id,problem_id) %>% summarise(n=n())
```

```
## # A tibble: 99,007 x 3
## # Groups:   user_id [250]
##    user_id problem_id     n
##      <int>      <int> <int>
##  1   80145      12302     1
##  2   80145      12312     2
##  3   80145      36947     2
##  4   80145      36963     2
##  5   80145      37094     1
##  6   80145      37098     1
##  7   80145      41130     1
##  8   80145      41177     1
##  9   80145      41194     1
## 10   80145      41195     1
## # ... with 98,997 more rows
```

```r
a %>% filter(user_id==80145, problem_id==12312)
```

```
## # A tibble: 2 x 18
##   user_id teacher_id school_id problem_id skill_id correct ms_first_respon~
##     <int>      <int>     <int>      <int>    <int>   <dbl>            <int>
## 1   80145      49343      5056      12312      276       1            47983
## 2   80145      49343      5056      12312      276       1            13974
## # ... with 11 more variables: hint_count <int>, attempt_count <int>,
## #   frustrated <dbl>, confused <dbl>, concentrating <dbl>, bored <dbl>,
## #   user_event_index <int>, time_spent <dbl>, rel_resp_time <dbl>,
## #   slow_response <int>, fast_response <int>
```

```r
a = a %>% mutate(
  stoppedAfterFirstAttempt = ifelse(attempt_count==1,1,0),
  neverAttempted = ifelse(attempt_count==0,1,0),
  stoppedAfterFirstAttemptAndIncorrect = ifelse(attempt_count==1&correct==0,1,0),
  noHintCorrect = ifelse(hint_count==0&correct==1,1,0)
)

a = a %>% group_by(user_id,problem_id) %>%
  mutate(
    multipleAttemtsToProblem = n(),
    percentCorrectMultipleAttempts = mean(correct),
    meanTimeSpent = mean(time_spent),
    meanHints = mean(hint_count),
    attemptLog = ifelse(attempt_count==0,0, log(attempt_count))
  ) %>%
  ungroup()
a = a %>% group_by(user_id) %>%
  mutate(
    totalEvents = n(),
    totalCorrect = sum(correct),
    attemptSDU = ifelse(attempt_count==0,0, sd(attempt_count)),
    attemptLogU = ifelse(attempt_count==0,0, log(attempt_count))
  ) %>% ungroup()

#######################################
```

```
########################################
```

Question 2: Sample 100 users for a training dataset and 50 users for the test dataset. You have to keep all of the rows for each user, so be sure to sample unique user ids and then keep all rows associated with those ids. Call the smaller datasets `train` and `test`.

```
########################################
####### BEGIN INPUT: Question 2 #######
########################################
uniqueStudents = unique(a$user_id)
ss = sample(rep(1:3, times = c(100,50,100)))
trainids = uniqueStudents[ss==1]
testids = uniqueStudents[ss==2]

train = a %>% filter(user_id %in% trainids)
test =  a %>% filter(user_id %in% testids)
########################################
########################################
```

Question 3: Fit a random forest model with your features using the `randomForest(xtrain, ytrain, xtest, ytest)` function. See how the random forest model lets you specify the training and test data (x are the predictors, y is the outcome, here boredom). Make sure to convert boredom to a `factor` so that the function understands that you want to run a classification (not regression). Be sure not to use any predictors that would not generalize (e.g. student id) or be unavailable (e.g. other affective states). You may want to fit at the beginning with just `ntree=100` to try out the performance quickly. Also remember that you can tweak the `mtry` parameter for how many variables to include in each tree.

Important: check your confusion matrix in the output of the randomForest model; if your model is just always predicting not-bored then it is clearly not a good enough model and you need to come up with better features. If so, go back to question 1 and adjust accordingly.

```
########################################
####### BEGIN INPUT: Question 3 #######
########################################

# Make a list of your features here to keep track of them
features = c("correct", "ms_first_response",
            "time_spent", "stoppedAfterFirstAttempt","attemptSDU",
             "multipleAttemtsToProblem","attemptLogU",
             "totalEvents","totalCorrect","meanTimeSpent","meanHints","attemptLog")

m.rf = randomForest(train[,features],
                    as.factor(train$bored),
                    test[,features],
                    as.factor(test$bored),
                    ntree = 100,
                    importance = TRUE)

m.rf
```

```
##
## Call:
##  randomForest(x = train[, features], y = as.factor(train$bored),      xtest = test[, features], ytes
##                Type of random forest: classification
##                      Number of trees: 100
## No. of variables tried at each split: 3
```

```
##
##         OOB estimate of  error rate: 15.99%
## Confusion matrix:
##       0   1 class.error
## 0 35542 245 0.006846061
## 1  6533  61 0.990749166
##                 Test set error rate: 15.82%
## Confusion matrix:
##       0  1  class.error
## 0 18682 11 0.0005884556
## 1  3501  2 0.9994290608
```

```
table(pred = m.rf$predicted, true=train$bored)
```

```
##     true
## pred     0     1
##    0 35542  6533
##    1   245    61
```

```
########################################
########################################
```

Question 4: Check on variable importance in the model that you developed. You can do this by fitting the randomForest above with the `importance = TRUE` parameter (if you didn't do so already). Then take the model object `m.rf` and run `importance(m.rf)` on it. Check out the help for what the different measures of importance mean. Write down the **3 best** predicting variables and **why** you think each one is a good predictor of boredom:

```
########################################
####### BEGIN INPUT: Question 4 #######
########################################

# Compute variable importance
importance(m.rf)
```

```
##                                  0          1 MeanDecreaseAccuracy
## correct                   8.518295  -7.029998             8.754520
## ms_first_response        24.080531 -24.294068            23.929716
## time_spent               27.175030 -27.140028            27.048353
## stoppedAfterFirstAttempt  7.491471  -6.106724             7.070302
## attemptSDU               15.702605  -7.583260            14.310771
## multipleAttemtsToProblem 18.077308 -12.305136            17.300018
## attemptLogU               4.353363  -4.238852             4.344323
## totalEvents              17.880048 -11.792234            17.821934
## totalCorrect             17.623380 -14.383985            16.745228
## meanTimeSpent            27.517802 -27.366007            27.349178
## meanHints                15.255461 -13.976794            14.163189
## attemptLog                9.806000  -9.116132             9.369857
##                          MeanDecreaseGini
## correct                          87.75036
## ms_first_response              1660.83670
## time_spent                     1638.69789
## stoppedAfterFirstAttempt         44.05108
## attemptSDU                      492.12588
## multipleAttemtsToProblem         82.69425
## attemptLogU                     106.21859
```

```
## totalEvents                         468.27510
## totalCorrect                        463.83316
## meanTimeSpent                      1631.61672
## meanHints                           181.21922
## attemptLog                          106.43876
```

```
# Write down your 3 most important variables and why you think they are good predictors
# ms_first_response: If students are answering the question very quickly, they probably are bored.
# time_spent: Same as above.
# meanTimeSpent: Problems that students attempt more than once gives a better indicator of time spent.

########################################
########################################
```

Question 5: Plot the ROC curve for the model you trained above.

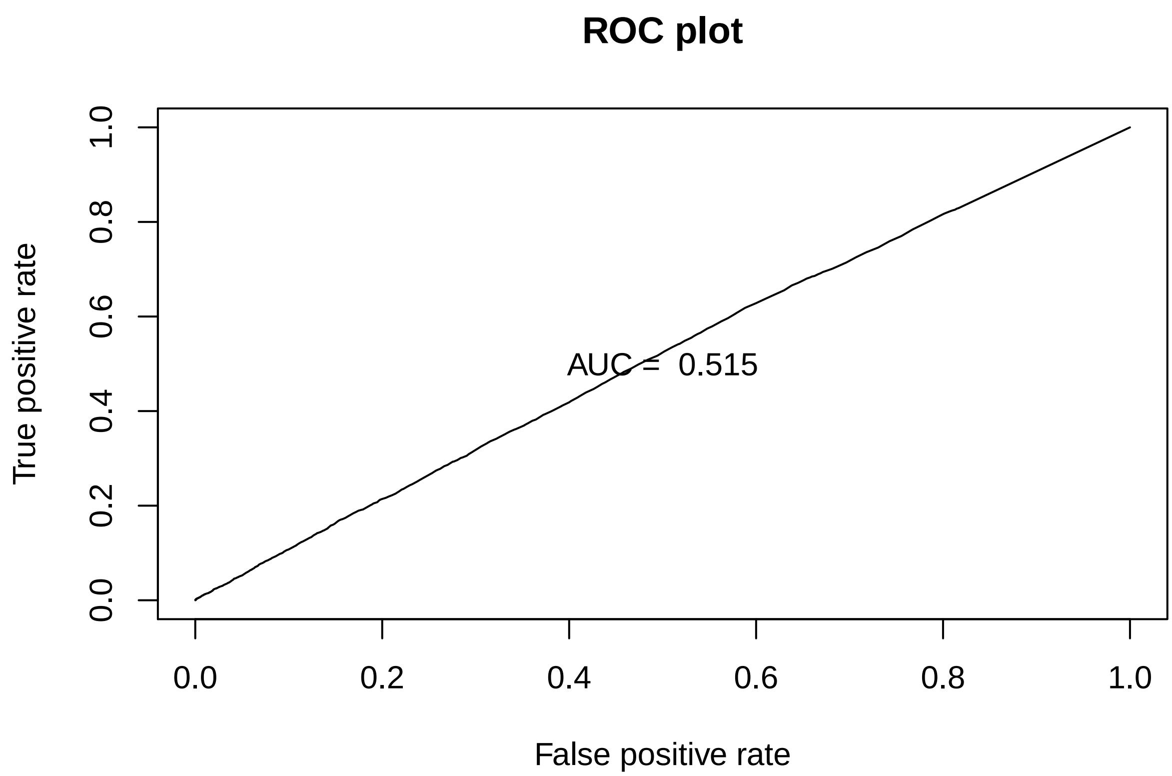
Here I show you how to do it with the ROCR package (be sure you installed and loaded it at the top). The ideal curve would be close to the top-left corner and have an AUC (area under the curve) value that is close to 1. An AUC value of 0.5 mean your model is not doing anything. If you see that, you should go back and improve your feature engineering and model parameters.

```
########################################
####### BEGIN INPUT: Question 5 #######
########################################

# Create the prediction object for ROCR
predictions = m.rf$votes[,2]
pred = prediction(predictions, train$bored)

# Calculate the AUC value
perf_AUC = performance(pred, measure = "auc")
AUC = perf_AUC@y.values[[1]]

# Plot the ROC curve
perf_ROC = performance(pred, "tpr", "fpr")
plot(perf_ROC, main="ROC plot")
text(0.5, 0.5, paste("AUC = ", format(AUC, digits=3, scientific=F)))
```




```
########################################
########################################
```

Self-reflection

Briefly summarize your experience on this homework. What was easy, what was hard, what did you learn?

This was hard. I tried multiple features to try to increase the AUC, but with very little success. The AUC score above was the best attempt.

Submit Homework

This is the end of the homework. Please **Knit a PDF report** that shows both the R code and R output and upload it on the EdX platform. Alternatively, you can Knit it as a "doc", open it in Word, and save that as a PDF.

Important: Be sure that all your code is visible. If the line is too long, it gets cut off. If that happens, organize your code on several lines.