Emotional Learning Analytics

INFO 5200 Learning Analytics: Week 13 Homework

[[Cole Walsh, 4399966]]

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 that you felt bored. What were you doing? How might this show up in this dataset. You should check out this paper which engineers features for 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.

Question 1: Inspect the correlations between the `bored` variable (the outcome you want to predict) and the other learning logs in the dataset. Then begin creating your own features for predicting boredom. I give you an example below. Please add at least 7 new features; note that the authors of the paper linked above created over 170 features!

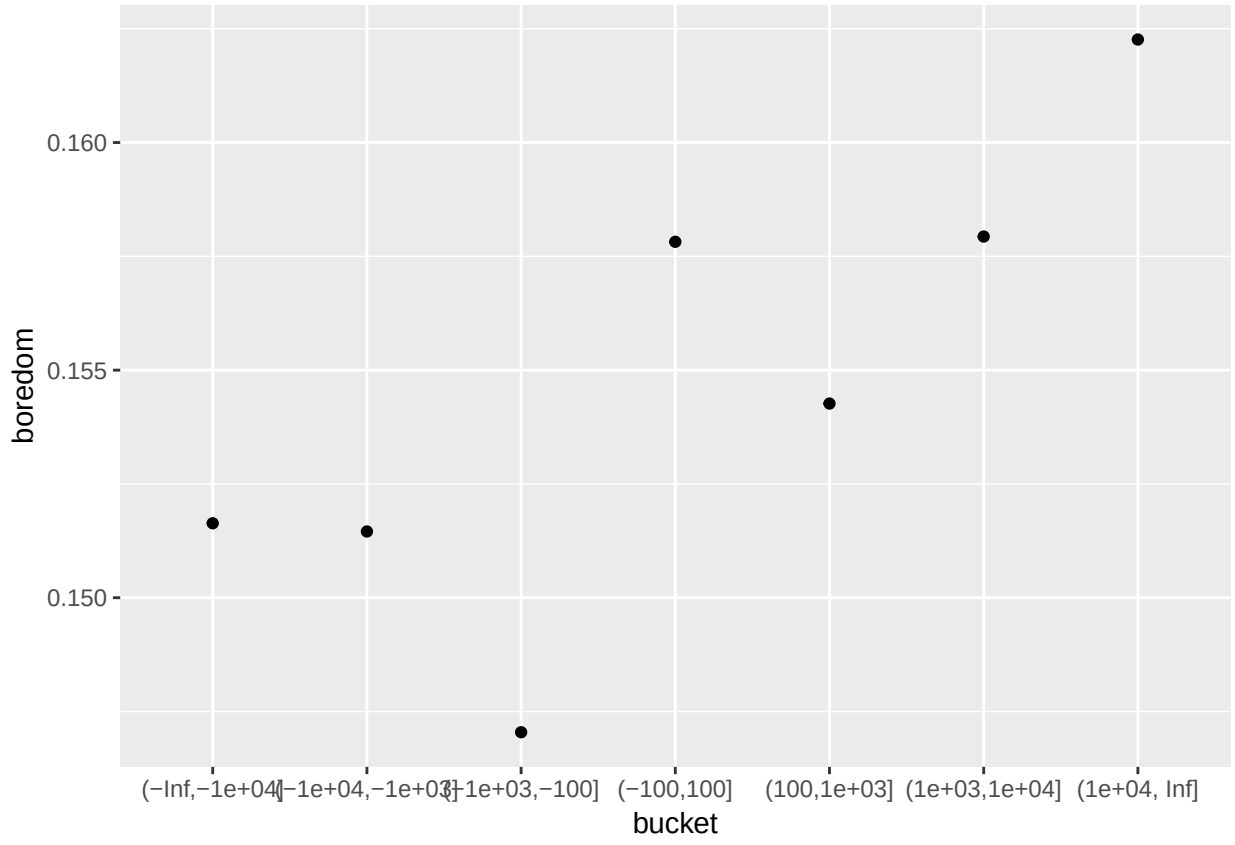
Instructor example: I would think that students who answer a question fast and move on are not bored (yet); students who take a long time to figure out what the answer may be and don't submit are probably bored. Students in the middle of the distribution maybe a mix of the two. I first check this idea 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","time_spent")
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
```




```
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
```

```
########################################
####### BEGIN INPUT: Question 2 #######
########################################

# Add your own exploration and feature engineering here (make at least 7 new features)
a.dt <- as.data.table(a)

# Data is time ordered within user, features ---> the number of consecutive questions answered
# correctly on the first attempt, if help was asked for on the first problem, if help was asked
# for on the last problem, number of last 5 first responses that were wrong, number of last 5
# questions where hints were used, total number of hints used up to that point, total time of
# last 5 problems, and total attempts of last 5 problems
a.dt[, Consecutive.Correct := cumsum(correct),
     by = .(user_id, rleid(correct))][,
                                      `:=`(First.Problem.Help = head(hint_count, 1) > 0,
                                           Last.Problem.Help = tail(hint_count, 1) > 0,
                                           Cumulative.Hints = cumsum(hint_count),
                                           Wrong.Previous.5 = Reduce(`+`, shift(1 - correct, 0:4)),
                                           Hints.Previous.5 = Reduce(`+`,
                                                                     shift(1 * (hint_count > 0),
                                                                           0:4)),
                                           Total.Time.Previous.5 = Reduce(`+`,
                                                                          shift(time_spent, 0:4)),
                                           Total.Attempts.Previous.5 = Reduce(`+`,
                                                                              shift(attempt_count,
                                                                                    0:4))),
                                      .(user_id)]

# Fill NAs with 0's
a.dt[is.na(a.dt)] <- 0

cor(a.dt[, c('bored', 'Consecutive.Correct', 'First.Problem.Help', 'Last.Problem.Help',
             'Cumulative.Hints', 'Wrong.Previous.5', 'Hints.Previous.5', 'Total.Time.Previous.5',
             'Total.Attempts.Previous.5')])
```

```
##                                bored Consecutive.Correct
## bored                     1.0000000000         0.002461607
## Consecutive.Correct       0.0024616071         1.000000000
## First.Problem.Help       -0.0048039954        -0.030378311
## Last.Problem.Help        -0.0005688717        -0.063658784
## Cumulative.Hints          0.0145906132        -0.141688589
## Wrong.Previous.5         -0.0002738029        -0.519790515
## Hints.Previous.5          0.0118631721        -0.290407618
## Total.Time.Previous.5    -0.0001879104        -0.029555218
## Total.Attempts.Previous.5 -0.0049236834       -0.251489875
##                          First.Problem.Help Last.Problem.Help
## bored                        -0.004803995     -0.0005688717
## Consecutive.Correct          -0.030378311     -0.0636587836
## First.Problem.Help            1.000000000      0.0230118367
## Last.Problem.Help             0.023011837      1.0000000000
## Cumulative.Hints              0.102416257      0.1023272661
## Wrong.Previous.5              0.038451308      0.0789940189
## Hints.Previous.5              0.074547210      0.1275208986
```

```
## Total.Time.Previous.5           -0.007987700      0.0146776667
## Total.Attempts.Previous.5        0.044248632      0.0751708266
##                           Cumulative.Hints Wrong.Previous.5
## bored                            0.01459061    -0.0002738029
## Consecutive.Correct             -0.14168859    -0.5197905153
## First.Problem.Help               0.10241626     0.0384513083
## Last.Problem.Help                0.10232727     0.0789940189
## Cumulative.Hints                 1.00000000     0.2162025430
## Wrong.Previous.5                 0.21620254     1.0000000000
## Hints.Previous.5                 0.28277896     0.6419628806
## Total.Time.Previous.5            0.01365024     0.0538870158
## Total.Attempts.Previous.5        0.14574527     0.4810168528
##                           Hints.Previous.5 Total.Time.Previous.5
## bored                          0.01186317         -0.0001879104
## Consecutive.Correct           -0.29040762         -0.0295552183
## First.Problem.Help             0.07454721         -0.0079877005
## Last.Problem.Help              0.12752090          0.0146776667
## Cumulative.Hints               0.28277896          0.0136502370
## Wrong.Previous.5               0.64196288          0.0538870158
## Hints.Previous.5               1.00000000          0.0435801868
## Total.Time.Previous.5          0.04358019          1.0000000000
## Total.Attempts.Previous.5      0.45392707          0.0868133027
##                           Total.Attempts.Previous.5
## bored                                -0.004923683
## Consecutive.Correct                  -0.251489875
## First.Problem.Help                    0.044248632
## Last.Problem.Help                     0.075170827
## Cumulative.Hints                      0.145745269
## Wrong.Previous.5                      0.481016853
## Hints.Previous.5                      0.453927072
## Total.Time.Previous.5                 0.086813303
## Total.Attempts.Previous.5             1.000000000
```

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

Question 2: Sample 100 users for a training dataset and 50 users for the test dataset (you should keep all of the rows for each user). Call the smaller datasets `train` and `test`.

```
#######################################
####### BEGIN INPUT: Question 2 #######
#######################################

set.seed(11)

users = sample(unique(a.dt$user_id), 150)

train = as.data.frame(a.dt[a.dt$user_id %in% users[1:100]])
test = as.data.frame(a.dt[a.dt$user_id %in% users[101:length(users)]])

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

Question 3: Fit a random forest model with your features using the `randomForest(xtrain, ytrain, xtest, ytest)` function. See how the random forest model let's 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", "hint_count",
             "attempt_count", "time_spent",
             "rel_resp_time", "slow_response", "fast_response", "Consecutive.Correct",
             "First.Problem.Help", "Last.Problem.Help", "Cumulative.Hints", "Wrong.Previous.5",
             "Hints.Previous.5", "Total.Time.Previous.5", "Total.Attempts.Previous.5")

m.rf = randomForest(train[, features], as.factor(train$bored), test[, features],
                    as.factor(test$bored), 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: 500
## No. of variables tried at each split: 4
##
##         OOB estimate of  error rate: 15.7%
## Confusion matrix:
##       0  1 class.error
## 0 35647 49 0.001372703
## 1  6594 22 0.996674728
##                 Test set error rate: 15.94%
## Confusion matrix:
##       0  1  class.error
## 0 16210 15 0.0009244992
## 1  3058  0 1.0000000000
```

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

```
##    true
##         0     1
##   0 35647  6594
##   1    49    22
```

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

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 write why you think it is a good predictor of boredom:

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

# Compute variable importance

Imp = importance(m.rf)
Rows = rownames(Imp)

Imp %>%
  data.frame(.) %>%
  mutate(Variable = Rows) %>%
  arrange(desc(MeanDecreaseGini))
```

```
##           X0         X1 MeanDecreaseAccuracy MeanDecreaseGini
## 1  63.592645 -38.101151            62.931603       1712.79411
## 2  61.499266 -58.028999            61.100980       1670.06446
## 3  60.635648 -56.079913            61.066257       1609.67852
## 4  61.188454 -58.633850            61.234021       1604.26030
## 5  46.214509 -12.037050            42.172095       1272.79110
## 6  26.587823 -18.393597            26.920260        565.66348
## 7  49.796677 -29.834619            45.013703        506.55450
## 8  28.956229 -21.417461            27.553859        333.42521
## 9  34.403140 -20.531281            33.392193        258.60353
## 10 23.350425 -16.227257            22.451303        166.99910
## 11 11.915002  -9.249526            12.282456        142.29852
## 12 21.596624  -8.307967            19.114525        104.21629
## 13 12.402148 -11.547326            12.371553         61.91648
## 14  7.884378  -1.821229             7.596104         56.29685
## 15 15.831660 -14.771877            15.932109         53.44860
## 16 16.147669 -15.482567            16.148789         43.93642
##                     Variable
## 1      Total.Time.Previous.5
## 2               rel_resp_time
## 3                  time_spent
## 4           ms_first_response
## 5            Cumulative.Hints
## 6         Consecutive.Correct
## 7   Total.Attempts.Previous.5
## 8            Wrong.Previous.5
## 9            Hints.Previous.5
## 10              attempt_count
## 11                 hint_count
## 12          Last.Problem.Help
## 13                    correct
## 14         First.Problem.Help
## 15              slow_response
## 16              fast_response
```

```
# Write down your 3 most important variables and why you think they are good predictors

# The total time that a student spent on the last 5 problems was most important in predicting
# boredom. Conceivably, longer times spent on most recent problems is indicative of persisting or
# oncoming boredom.

# The first response time relative to the median first response time was also important for a likely
# similar reason. When a student takes longer to click on a problem than usual this is indicative of
# boredom.

# Additionally, the time spent on a problem indicates that a student is likely to get bored if they
# spend a lot of time on a problem.

# Time features are apparently important...

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

Here I show you how to do it with the ROCR package. 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.

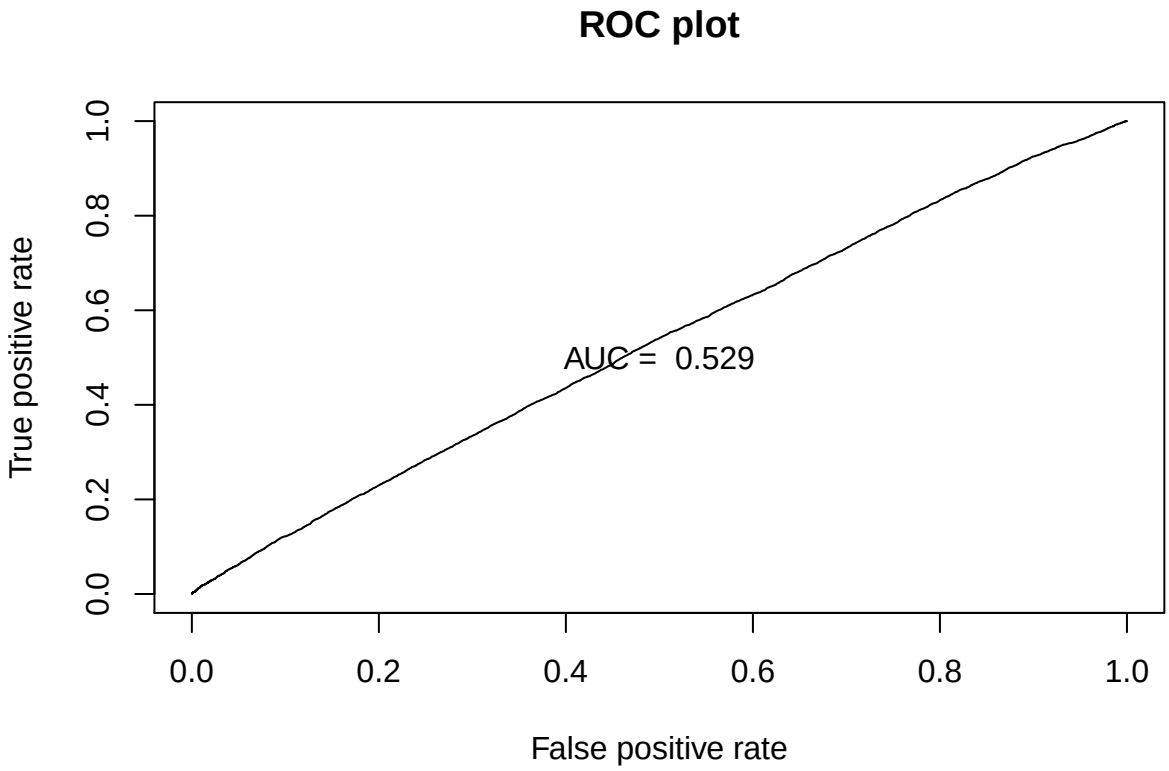
Question 4: Plot the ROC curve for the model you trained above.

```
# install.packages("ROCR")
library(ROCR)
```

```
## Warning: package 'ROCR' was built under R version 3.5.3

## Loading required package: gplots

## Warning: package 'gplots' was built under R version 3.5.3

##
## Attaching package: 'gplots'

## The following object is masked from 'package:stats':
##
##     lowess
```

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

# 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]]
```

```r
# 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)))
```




```r
#####################################
#####################################
```

Self-reflection

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

Interesting homework. Feature engineering felt easier than earlier times partly because of the paper which provided some great ideas for useful features. Using data.table also helped. In the paper, they used 173 features and found an AUC of 0.632, so with only 10-20 features its not surprising that my AUC was 0.528; this is a difficult thing to model and we can do only slightly better than chance.

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.