Predictive Models in Education

INFO 4100 Learning Analytics

[[Shuhui Zhu, sz649]]

In this homework, you will go through all of the basic steps of conceptualizing, building, and evaluating predictive models.

Learning Objectives:

1. Understand how to identify a problem that can be encoded as a prediction task
2. Identify appropriate outcome variables and predictor variables
3. Create new features based on existing data
4. Build and evaluate several different prediction models

# Part 1: Loading the Dataset

You will analyze the Assisstments dataset from the previous homework, which means that you are already familiar with the properties of the data. To refresh your memory, here is some general information about the dataset: it provides question-level data of students practicing math problems in academic year 2004-2005 using the [Assisstments platform](https://www.assistments.org/). On this platform, students can attempt a problem many times to get it right and they can ask for more and more hints on a problem until the final hint tells them what the answer is. Based on the first few lines of data, and what we know about the dataset, we can infer the following:

* *studentID* is an identifier for students
* *itemid* is an identifier for math questions
* *correctonfirstattempt* is an indicator of whether a student answered correctly on the first attempt
* *attempts* is the number of answer attempts required
* *hints* the number of hints a student requested
* *seconds* time spent on the question in seconds
* the remaining columns provide start and end times and dates for each question

The dataset is in **long format** (1 row = 1 event) instead of wide format (1 row = 1 individual). However, as you can see from the *attempts* variable, you do not have data on each attempt, but a question-level rollup. The data is at the student-question level, which means that there is one row for each question a student attempted that summarizes interaction with the question (performance indicators and time spent).

Start by loading the tidyverse package and the dataset:

library(tidyverse, quietly = T)

## Warning: package 'tidyverse' was built under R version 4.0.5

## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --

## v ggplot2 3.3.3 v purrr 0.3.4  
## v tibble 3.1.0 v dplyr 1.0.6  
## v tidyr 1.1.3 v stringr 1.4.0  
## v readr 1.4.0 v forcats 0.5.1

## Warning: package 'ggplot2' was built under R version 4.0.5

## Warning: package 'tibble' was built under R version 4.0.5

## Warning: package 'tidyr' was built under R version 4.0.5

## Warning: package 'readr' was built under R version 4.0.5

## Warning: package 'purrr' was built under R version 4.0.5

## Warning: package 'dplyr' was built under R version 4.0.5

## Warning: package 'forcats' was built under R version 4.0.5

## -- Conflicts ------------------------------------------ tidyverse\_conflicts() --  
## x dplyr::filter() masks stats::filter()  
## x dplyr::lag() masks stats::lag()

asm = readRDS("info4100.data.assisstments.rds")

# Part 2: Problem Identification

In the real world, we usually start by identifying the problem and then collect data. Here we have a dataset to work with. So what problems might we solve? Here are some ideas:

* predict dropout, (how long) will students stay engaged to intervene before they disengage
* predict correctness on first attempt to start adapting content for at-risk students
* predict time spent, predict number of hints for improving the experience

For the purpose of this homework, we are going to predict dropout. It’s a common problem and it is at the student-level, which simplifies methodological considerations.

We can set this up two different ways:

* As a regression problem, the outcome can be the number quizzes completed i.e. how far did you get
* As a classification problem, the outcome can be returning after a given point – e.g. of those students who have come in and finish 100 questions, how many are going to do at least 300 questions?

For both outcomes, you will need to assume that you are observing these students for a while (say until they finished 100 questions) and then you try to predict the future. You can use the data you observed to make predictions but nothing thereafter.

# Part 3: Data Collection

Which of the variables in the dataset will be used. First, what is the outcome? Second, what are the predictors?

### Outcomes

* For the regression problem we are interested in the number (i.e. numeric) of quizzes.
* For the classification problem we are interested in whether (i.e. binary) they go on to complete at least 300, after completing 100 questions.

### Predictors

* there are no user attributes in this dataset (socio-demographic or other)
* however, you have access to information about quiz-taking that can be used to engineer features

# Part 4: Feature Engineering

This is where you create the dataset that you will use in the prediction model. **You need a student-level dataset.** Check out the previous homework to see how to use the group\_by and summarise functions from the tidyverse package to achieve this.

Usually, feature engineering focuses on just the predictors, but let’s also create the outcomes in this section.

**Question 1:** Create a dataset (call it *asm\_outcomes*) that has for each student the number of quizzes completed and and indicator of whether that below 300 (i.e. dropped out before). You are looking for a dataset with 912 rows (# of unique students) and three columns: studentID, num\_quiz, quiz300. You can refer to the last HW for help.

#######################################  
####### BEGIN INPUT: Question 1 #######  
#######################################  
library(dplyr)  
asm\_outcomes <- asm %>%  
 group\_by(studentID) %>%  
 summarise(num\_quiz = n())  
asm\_outcomes$quiz300 <- as.numeric(asm\_outcomes$num\_quiz > 300)  
asm\_outcomes

## # A tibble: 912 x 3  
## studentID num\_quiz quiz300  
## <int> <int> <dbl>  
## 1 136 518 1  
## 2 137 687 1  
## 3 139 538 1  
## 4 140 522 1  
## 5 141 113 0  
## 6 142 5 0  
## 7 143 560 1  
## 8 144 281 0  
## 9 145 593 1  
## 10 146 465 1  
## # ... with 902 more rows

#######################################  
#######################################

Now let’s engineer some features to predict dropout. I will leave this up to your creativity. You can create as many features as you can think of. You can also evaluate them by looking at their correlation with the outcome if you like. Here is just one example to get you started. I’ll create a feature that is the total time spent so far working on questions.

However, there is one critical step not to forget. The features can only be computed using data up to the 100th quiz, given the prediction problem. You will need to throw out the rest. First, keep only the first 100 question records for each student. In this dataset, it takes some (cumbersome) data processing because of how the dates are formatted.

Here is one way to do it: We make a timestamp that can be rank ordered. Then we create a variable i that counts the question order for each student. Now that we know the order in which questions were answered, we can filter out all but the first 100.

# We first need to go through this tedious process of   
# dealing with the dates to make them sortable  
  
# convert to character string  
asm$start\_day = as.character(asm$start\_day)   
# split up e.g. 03-OCT-05  
start\_day\_split = strsplit(asm$start\_day, split = "-", fixed = T)  
# get the day  
asm$start\_d = unlist(lapply(start\_day\_split, first))  
# get the year, add 20 in front  
asm$start\_y = paste0(20, unlist(lapply(start\_day\_split, last)))   
# get/convert month  
asm$start\_m = match(unlist(lapply(start\_day\_split, function(x) x[2])), toupper(month.abb))   
# convert time to character string  
asm$start\_time = as.character(asm$start\_time)   
# concat it all  
asm$start\_timestamp = paste0(asm$start\_y, asm$start\_m, asm$start\_d, asm$start\_time)   
  
# Compute the order in which students answered questions, keep first 100  
asm\_sub = asm %>%   
 group\_by(studentID) %>%  
 mutate(i = rank(start\_timestamp, ties.method = "random")) %>%  
 filter(i <= 100)

**Question 2:** Now that you have a dataset with only the information in it that you can use for prediction, you can start engineering features. Below, you should engineer 10-15 features. Be creative, think about what behaviors could signal that a student will/won’t drop out.

#######################################  
####### BEGIN INPUT: Question 2 #######  
#######################################  
  
# Now using the asm\_sub dataset we can finally compute features like total time  
asm\_features = asm\_sub %>%   
 group\_by(studentID) %>%  
 summarise(  
 cnt = n(),  
 total\_time = sum(seconds), #--1  
 avg\_time = 1/mean(seconds), #--2  
 avg\_correct\_first = mean(correctonfirstattempt), #--3  
 avg\_hints = mean(hints), #--4  
 avg\_attempts = mean(attempts), #--5  
 avg\_month = sd(start\_m), #--6  
 std\_start\_d = sd(start\_d),#--7  
 std\_sec = sd(seconds), #--8  
 std\_attempts = sd(attempts), #--9  
 std\_hints = sd(hints), #--11  
 diff\_attempts\_sec = sum(seconds)/sum(attempts+1)) #--10  
 # TODO: You create some more features here for prediction  
  
  
  
# check out your features to make sure you don't have   
# missing values and the distributions look reasonable  
# if there are missing values (NAs) then you should handle them before moving on  
asm\_features[rowSums(is.na(asm\_features)) > 0, ]

## # A tibble: 6 x 13  
## studentID cnt total\_time avg\_time avg\_correct\_first avg\_hints avg\_attempts  
## <int> <int> <int> <dbl> <dbl> <dbl> <dbl>  
## 1 479 1 28 0.0357 0 0 1  
## 2 5001 1 11 0.0909 0 1 0  
## 3 6134 1 340 0.00294 0 2 2  
## 4 6141 1 51 0.0196 0 3 1  
## 5 6620 1 270 0.00370 0 1 1  
## 6 6664 1 281 0.00356 0 1 0  
## # ... with 6 more variables: avg\_month <dbl>, std\_start\_d <dbl>, std\_sec <dbl>,  
## # std\_attempts <dbl>, std\_hints <dbl>, diff\_attempts\_sec <dbl>

asm\_features[is.na(asm\_features)] <- 0  
  
summary(asm\_features)

## studentID cnt total\_time avg\_time   
## Min. : 136.0 Min. : 1.00 Min. : 11 Min. :0.002911   
## 1st Qu.: 447.8 1st Qu.:100.00 1st Qu.: 3202 1st Qu.:0.015007   
## Median : 745.5 Median :100.00 Median : 4426 Median :0.019804   
## Mean :1088.0 Mean : 85.58 Mean : 4463 Mean :0.022594   
## 3rd Qu.:1054.2 3rd Qu.:100.00 3rd Qu.: 5726 3rd Qu.:0.025738   
## Max. :6802.0 Max. :100.00 Max. :11264 Max. :0.240000   
## avg\_correct\_first avg\_hints avg\_attempts avg\_month   
## Min. :0.0000 Min. :0.0000 Min. :0.000 Min. :0.0000   
## 1st Qu.:0.2700 1st Qu.:0.3479 1st Qu.:1.306 1st Qu.:0.4094   
## Median :0.3900 Median :0.6633 Median :1.490 Median :0.6110   
## Mean :0.3957 Mean :0.7645 Mean :1.518 Mean :1.4529   
## 3rd Qu.:0.5100 3rd Qu.:1.1000 3rd Qu.:1.680 3rd Qu.:2.8086   
## Max. :1.0000 Max. :3.5000 Max. :5.000 Max. :5.5701   
## std\_start\_d std\_sec std\_attempts std\_hints   
## Min. : 0.000 Min. : 0.00 Min. :0.0000 Min. :0.0000   
## 1st Qu.: 3.671 1st Qu.: 45.18 1st Qu.:0.9551 1st Qu.:0.8203   
## Median : 6.320 Median : 59.89 Median :1.1989 Median :1.1859   
## Mean : 5.734 Mean : 64.63 Mean :1.2929 Mean :1.1002   
## 3rd Qu.: 7.686 3rd Qu.: 78.78 3rd Qu.:1.4976 3rd Qu.:1.4276   
## Max. :14.537 Max. :399.69 Max. :5.6569 Max. :3.8513   
## diff\_attempts\_sec  
## Min. : 1.389   
## 1st Qu.: 15.805   
## Median : 19.733   
## Mean : 22.347   
## 3rd Qu.: 26.148   
## Max. :281.000

#######################################  
#######################################

Lastly, you will need to merge the two datasets back together: the one with the outcome data and the one with the features. This dataset should have 912 rows.

asm\_combined = left\_join(asm\_features, asm\_outcomes, by = "studentID")  
nrow(asm\_combined)

## [1] 912

asm\_combined

## # A tibble: 912 x 15  
## studentID cnt total\_time avg\_time avg\_correct\_first avg\_hints avg\_attempts  
## <int> <int> <int> <dbl> <dbl> <dbl> <dbl>  
## 1 136 100 4169 0.0240 0.48 0.4 1.62  
## 2 137 100 3191 0.0313 0.44 0.77 1.48  
## 3 139 100 3414 0.0293 0.28 1.42 1.51  
## 4 140 100 4856 0.0206 0.4 0.37 1.71  
## 5 141 100 8709 0.0115 0.55 0.14 1.79  
## 6 142 5 440 0.0114 0 0.6 1.6   
## 7 143 100 3554 0.0281 0.24 1.47 1.52  
## 8 144 100 6687 0.0150 0.67 0.32 1.25  
## 9 145 100 3422 0.0292 0.58 0.38 1.5   
## 10 146 100 4768 0.0210 0.3 1.17 1.5   
## # ... with 902 more rows, and 8 more variables: avg\_month <dbl>,  
## # std\_start\_d <dbl>, std\_sec <dbl>, std\_attempts <dbl>, std\_hints <dbl>,  
## # diff\_attempts\_sec <dbl>, num\_quiz <int>, quiz300 <dbl>

# Part 5: Feature Selection

This step is usually needed when you have thousands of features, or more features than data points. One option is to remove features that are not predictive, another is to combine many weaker features into one stronger one. A common method for the latter is Principle Component Analysis (PCA).

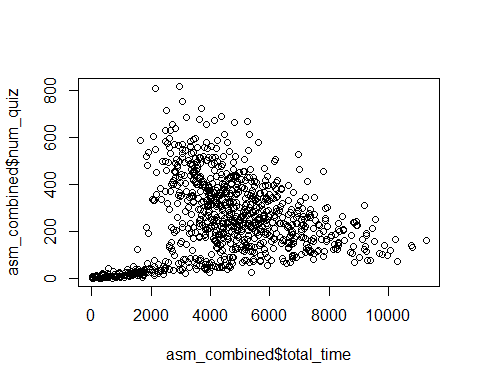
For now, I am assuming you created about 10-15 features in step 3. If you only have 5 or so, go back and come up with more.

**Question 3:** Take the opportunity to evaluate your features. Check out the correlation, make plots to see if you are perhaps trying to fit a straight line when the relationship is quadratic or cubic. If so, go back and refine your features.

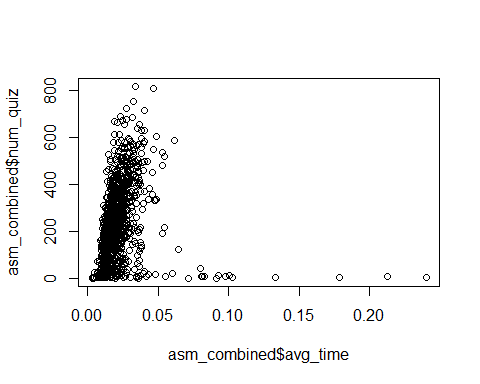
#######################################  
####### BEGIN INPUT: Question 3 #######  
#######################################  
  
# Specify your own feature variables here  
outcome\_vars = c("total\_time", "avg\_time", "avg\_correct\_first", 'avg\_hints', 'avg\_attempts','avg\_month','std\_start\_d','std\_sec','std\_attempts','std\_hints','diff\_attempts\_sec','num\_quiz','quiz300')   
  
# Look at their correlation  
cor(asm\_combined)[,outcome\_vars]

## total\_time avg\_time avg\_correct\_first avg\_hints  
## studentID -0.386042664 0.16602574 0.001047651 0.030074596  
## cnt 0.636035698 -0.11186931 0.098177604 -0.034518506  
## total\_time 1.000000000 -0.44879457 0.120274498 -0.179178920  
## avg\_time -0.448794566 1.00000000 0.101537788 0.020081955  
## avg\_correct\_first 0.120274498 0.10153779 1.000000000 -0.816654207  
## avg\_hints -0.179178920 0.02008196 -0.816654207 1.000000000  
## avg\_attempts 0.207108243 -0.24344427 -0.221394350 -0.039716685  
## avg\_month 0.450841201 -0.23332734 0.003697975 -0.056597396  
## std\_start\_d 0.611552004 -0.26807708 0.146743412 -0.148749011  
## std\_sec 0.379609353 -0.51801692 -0.098435516 -0.012255301  
## std\_attempts 0.208826288 -0.22891934 -0.220662033 0.066402674  
## std\_hints 0.001309867 -0.05418335 -0.647759620 0.775409274  
## diff\_attempts\_sec 0.142115234 -0.45025819 -0.093020952 -0.006199242  
## num\_quiz 0.131437185 0.10970760 0.127248406 -0.033185194  
## quiz300 -0.074711724 0.14611246 0.127623478 -0.029728148  
## avg\_attempts avg\_month std\_start\_d std\_sec  
## studentID -0.18377786 -0.166929659 -0.39434494 -0.03223748  
## cnt 0.00825062 0.190284615 0.56304835 -0.14018853  
## total\_time 0.20710824 0.450841201 0.61155200 0.37960935  
## avg\_time -0.24344427 -0.233327343 -0.26807708 -0.51801692  
## avg\_correct\_first -0.22139435 0.003697975 0.14674341 -0.09843552  
## avg\_hints -0.03971669 -0.056597396 -0.14874901 -0.01225530  
## avg\_attempts 1.00000000 0.151571261 0.07443812 0.53093664  
## avg\_month 0.15157126 1.000000000 0.24348430 0.27900899  
## std\_start\_d 0.07443812 0.243484297 1.00000000 0.13642198  
## std\_sec 0.53093664 0.279008986 0.13642198 1.00000000  
## std\_attempts 0.78810248 0.159625306 0.08356574 0.49653904  
## std\_hints 0.12464223 0.044730057 0.02035424 0.05966079  
## diff\_attempts\_sec -0.02587550 0.116453223 -0.01593698 0.43860694  
## num\_quiz -0.08547491 -0.190247937 0.23833710 -0.28657279  
## quiz300 -0.11703105 -0.283873052 0.11942403 -0.27110496  
## std\_attempts std\_hints diff\_attempts\_sec num\_quiz  
## studentID -0.20533113 -0.168901673 0.263123695 -0.40298119  
## cnt 0.08397543 0.170199325 -0.344835727 0.64927941  
## total\_time 0.20882629 0.001309867 0.142115234 0.13143719  
## avg\_time -0.22891934 -0.054183345 -0.450258187 0.10970760  
## avg\_correct\_first -0.22066203 -0.647759620 -0.093020952 0.12724841  
## avg\_hints 0.06640267 0.775409274 -0.006199242 -0.03318519  
## avg\_attempts 0.78810248 0.124642228 -0.025875501 -0.08547491  
## avg\_month 0.15962531 0.044730057 0.116453223 -0.19024794  
## std\_start\_d 0.08356574 0.020354245 -0.015936981 0.23833710  
## std\_sec 0.49653904 0.059660791 0.438606940 -0.28657279  
## std\_attempts 1.00000000 0.185124328 -0.040866711 -0.01424960  
## std\_hints 0.18512433 1.000000000 -0.161084184 0.07110902  
## diff\_attempts\_sec -0.04086671 -0.161084184 1.000000000 -0.35870900  
## num\_quiz -0.01424960 0.071109022 -0.358709003 1.00000000  
## quiz300 -0.05653139 0.010419360 -0.276183871 0.82261259  
## quiz300  
## studentID -0.24773508  
## cnt 0.36768252  
## total\_time -0.07471172  
## avg\_time 0.14611246  
## avg\_correct\_first 0.12762348  
## avg\_hints -0.02972815  
## avg\_attempts -0.11703105  
## avg\_month -0.28387305  
## std\_start\_d 0.11942403  
## std\_sec -0.27110496  
## std\_attempts -0.05653139  
## std\_hints 0.01041936  
## diff\_attempts\_sec -0.27618387  
## num\_quiz 0.82261259  
## quiz300 1.00000000

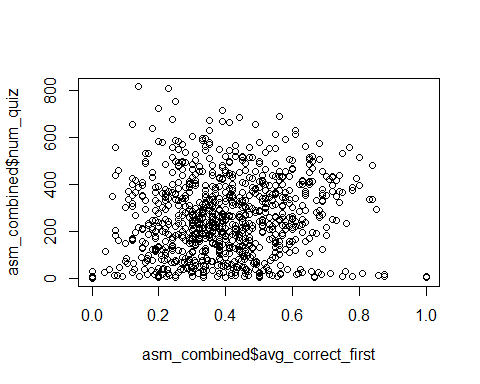
# Make whatever plots you find helpful here  
plot(asm\_combined$total\_time, asm\_combined$num\_quiz)



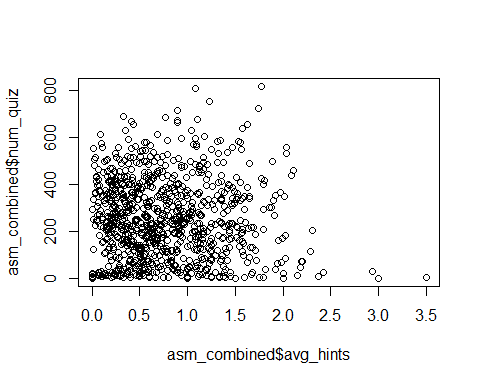
plot(asm\_combined$avg\_time, asm\_combined$num\_quiz) # time feature is adjusted based on the original graph



plot(asm\_combined$avg\_correct\_first, asm\_combined$num\_quiz)



plot(asm\_combined$avg\_hints, asm\_combined$num\_quiz)



#######################################  
#######################################

# Part 6: Model Selection and Building

Before we can start building models, we need to split our dataset into a training and a test set. (Note that it we should usually do this before feature engineering so that we are not influenced in our choices by data that we shouldn’t be seeing. But then we would have to do the engineering twice. So let’s just do it here.)

The dataset is now quite small: 912 students. We do want enough data to train our model, so let’s do a 80/20 split: 80% training, 20% test. It is important that the split is **random**. Why? Because we want it to be a representative sample.

# Sample 80% of studentIDs for training and the rest is for testing,   
# you want a vector of studentIDs  
ids\_train = sample(asm\_combined$studentID, size = 912 \* 0.8)  
   
# Split the dataset into two; use filter() and %in% to select rows  
train = asm\_combined %>% filter(studentID %in% ids\_train)  
test = asm\_combined %>% filter(!studentID %in% ids\_train)

### Need a just-in-time R tutorial?

<https://www.datacamp.com/community/tutorials/machine-learning-in-r>

### Before you start building

You should only use predictor variables that make sense. For example, your dataset has the studentID variable which does not make sense to use as a predictor, so you need to make sure you don’t use it. You also should not use outcome measures as predictors. You created two different outcome measures. Make sure you don’t accidentially use one of them as a predictor.

### Linear regression

**Question 4:** Fit a linear regression model using the lm() function like this:

* lm(outcome ~ predictor1 + predictor2 + predictor3, data = train)

#######################################  
####### BEGIN INPUT: Question 4 #######  
#######################################  
  
m\_linreg = lm(num\_quiz ~ total\_time + avg\_time + avg\_correct\_first + avg\_hints + avg\_attempts + avg\_month + std\_start\_d + std\_sec + std\_attempts + std\_hints + diff\_attempts\_sec, data = train)  
  
#m\_linreg\_binr = lm(quiz300 ~ total\_time + avg\_time + avg\_correct\_first + avg\_hints + avg\_attempts + avg\_month + std\_start\_d + std\_sec + std\_attempts + std\_hints + diff\_attempts\_sec, data = train)  
  
  
# the output are the coefficients:  
m\_linreg

##   
## Call:  
## lm(formula = num\_quiz ~ total\_time + avg\_time + avg\_correct\_first +   
## avg\_hints + avg\_attempts + avg\_month + std\_start\_d + std\_sec +   
## std\_attempts + std\_hints + diff\_attempts\_sec, data = train)  
##   
## Coefficients:  
## (Intercept) total\_time avg\_time avg\_correct\_first   
## 250.56539 0.02239 -801.41191 129.09556   
## avg\_hints avg\_attempts avg\_month std\_start\_d   
## 12.61517 -53.82476 -25.46018 7.65532   
## std\_sec std\_attempts std\_hints diff\_attempts\_sec   
## -1.93145 49.76233 37.61158 -2.43548

# m\_linreg\_binr  
#######################################  
#######################################

### Logistic regression

**Question 5:** Fit a logistic regression model using the glm() function like this:

* glm(outcome ~ predictor1 + predictor2 + predictor3, data = train, family = “binomial”)

#######################################  
####### BEGIN INPUT: Question 5 #######  
#######################################  
  
# m\_logreg = glm(num\_quiz ~ total\_time + avg\_time + avg\_correct\_first + avg\_hints + avg\_attempts + avg\_month + std\_start\_d + std\_sec + std\_attempts + std\_hints + diff\_attempts\_sec, data = train, family = "gaussian")  
  
m\_logreg = glm(quiz300 ~ total\_time + avg\_time + avg\_correct\_first + avg\_hints + avg\_attempts + avg\_month + std\_start\_d + std\_sec + std\_attempts + std\_hints + diff\_attempts\_sec, data = train, family = "binomial")

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

# the output are the coefficients:  
m\_logreg

##   
## Call: glm(formula = quiz300 ~ total\_time + avg\_time + avg\_correct\_first +   
## avg\_hints + avg\_attempts + avg\_month + std\_start\_d + std\_sec +   
## std\_attempts + std\_hints + diff\_attempts\_sec, family = "binomial",   
## data = train)  
##   
## Coefficients:  
## (Intercept) total\_time avg\_time avg\_correct\_first   
## 1.497e+01 2.265e-03 -1.260e+02 5.676e+00   
## avg\_hints avg\_attempts avg\_month std\_start\_d   
## 1.471e+00 -6.012e+00 -4.589e-01 6.614e-02   
## std\_sec std\_attempts std\_hints diff\_attempts\_sec   
## 8.261e-03 2.360e-01 -6.859e-01 -9.031e-01   
##   
## Degrees of Freedom: 728 Total (i.e. Null); 717 Residual  
## Null Deviance: 951   
## Residual Deviance: 617.3 AIC: 641.3

# m\_logreg\_binr  
#######################################  
#######################################

### k Nearest Neighbor

**Question 6:** Fit a kNN model using the knn() function from the class package. However, note that the syntax starts to get different here, and you would usually do some tuning, e.g. choosing the right value of *k*. For this case, just choose a number between 1 and 5. The function takes the predictor matrix for training and testing, and a vector of outcomes (binary) for training.

* knn(train = training\_predictors, test = testing\_predictors, cl = training\_outcome, k = k)

#######################################  
####### BEGIN INPUT: Question 6 #######  
#######################################  
input\_vars = c("total\_time", "avg\_time", "avg\_correct\_first", 'avg\_hints', 'avg\_attempts','avg\_month','std\_start\_d','std\_sec','std\_attempts','std\_hints','diff\_attempts\_sec')   
  
# install.packages("class") # uncomment to install the class package, then delete  
library(class)

## Warning: package 'class' was built under R version 4.0.5

m\_knn = knn(train = train[, input\_vars], test = test[, input\_vars], cl = train$quiz300, k = 3)  
  
# the output are the predictions:  
m\_knn

## [1] 1 0 1 1 0 0 1 0 1 0 0 0 0 1 1 0 0 0 0 0 1 0 0 1 0 1 0 0 0 0 1 0 1 0 0 0 1  
## [38] 1 0 1 0 0 1 0 0 0 1 0 1 1 0 1 1 1 0 1 0 0 0 0 1 1 1 1 1 1 0 1 1 0 1 0 0 0  
## [75] 1 0 0 0 1 1 0 0 1 1 1 1 0 0 1 0 0 0 0 1 1 1 0 0 1 1 0 0 1 1 0 1 1 0 1 0 1  
## [112] 0 1 1 0 1 0 1 1 0 0 0 0 0 0 0 0 1 0 1 0 1 1 1 1 0 0 1 1 0 0 0 0 1 1 0 0 1  
## [149] 1 1 0 0 0 0 0 1 0 0 0 1 1 0 0 0 0 1 1 1 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0  
## Levels: 0 1

#######################################  
#######################################

### Classification and Regression Trees

**Question 7:** Fit a CART model using the rpart() function from the rpart package. The syntax is pretty similar to the linear/logistic regression models. To build a classification tree you specify method as ‘class’, for a regression tree you specify it as ‘anova’.

* rpart(binary\_outcome ~ predictor1 + predictor2 + predictor3, data = train, method = “class”)
* rpart(numeric\_outcome ~ predictor1 + predictor2 + predictor3, data = train, method = “anova”)

Here’s an [R tutorial for CART](https://www.statmethods.net/advstats/cart.html).

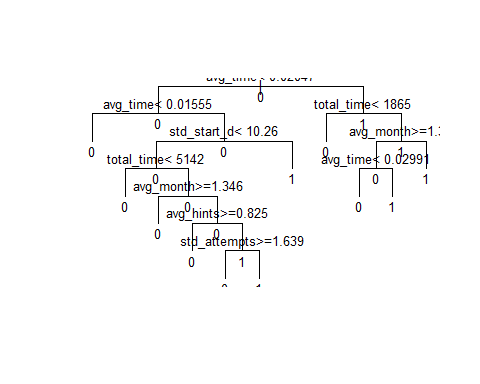
#######################################  
####### BEGIN INPUT: Question 7 #######  
#######################################  
  
# install.packages("rpart") # you may need to install this first  
library(rpart)  
m\_class\_tree = rpart(quiz300 ~ total\_time + avg\_time + avg\_correct\_first +   
 avg\_hints + avg\_attempts + avg\_month + std\_start\_d + std\_sec +   
 std\_attempts + std\_hints + diff\_attempts\_sec, data = train, method = "class")  
  
m\_reg\_tree = rpart(num\_quiz ~ total\_time + avg\_time + avg\_correct\_first +   
 avg\_hints + avg\_attempts + avg\_month + std\_start\_d + std\_sec +   
 std\_attempts + std\_hints + diff\_attempts\_sec, data = train, method = "anova")  
  
# the output are the decision trees  
m\_class\_tree

## n= 729   
##   
## node), split, n, loss, yval, (yprob)  
## \* denotes terminal node  
##   
## 1) root 729 261 0 (0.64197531 0.35802469)   
## 2) avg\_time< 0.02046875 381 66 0 (0.82677165 0.17322835)   
## 4) avg\_time< 0.0155534 196 10 0 (0.94897959 0.05102041) \*  
## 5) avg\_time>=0.0155534 185 56 0 (0.69729730 0.30270270)   
## 10) std\_start\_d< 10.25686 165 41 0 (0.75151515 0.24848485)   
## 20) total\_time< 5141.5 54 2 0 (0.96296296 0.03703704) \*  
## 21) total\_time>=5141.5 111 39 0 (0.64864865 0.35135135)   
## 42) avg\_month>=1.345631 41 6 0 (0.85365854 0.14634146) \*  
## 43) avg\_month< 1.345631 70 33 0 (0.52857143 0.47142857)   
## 86) avg\_hints>=0.825 24 5 0 (0.79166667 0.20833333) \*  
## 87) avg\_hints< 0.825 46 18 1 (0.39130435 0.60869565)   
## 174) std\_attempts>=1.639077 7 2 0 (0.71428571 0.28571429) \*  
## 175) std\_attempts< 1.639077 39 13 1 (0.33333333 0.66666667) \*  
## 11) std\_start\_d>=10.25686 20 5 1 (0.25000000 0.75000000) \*  
## 3) avg\_time>=0.02046875 348 153 1 (0.43965517 0.56034483)   
## 6) total\_time< 1865 37 0 0 (1.00000000 0.00000000) \*  
## 7) total\_time>=1865 311 116 1 (0.37299035 0.62700965)   
## 14) avg\_month>=1.350261 50 12 0 (0.76000000 0.24000000)   
## 28) avg\_time< 0.02991394 41 6 0 (0.85365854 0.14634146) \*  
## 29) avg\_time>=0.02991394 9 3 1 (0.33333333 0.66666667) \*  
## 15) avg\_month< 1.350261 261 78 1 (0.29885057 0.70114943) \*

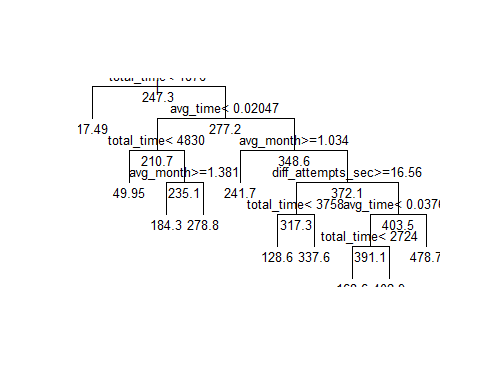
m\_reg\_tree

## n= 729   
##   
## node), split, n, deviance, yval  
## \* denotes terminal node  
##   
## 1) root 729 20099600.00 247.28670   
## 2) total\_time< 1875.5 84 60880.99 17.48810 \*  
## 3) total\_time>=1875.5 645 15025210.00 277.21400   
## 6) avg\_time< 0.02046875 334 4729348.00 210.74250   
## 12) total\_time< 4830 44 14753.91 49.95455 \*  
## 13) total\_time>=4830 290 3404482.00 235.13790   
## 26) avg\_month>=1.380629 134 1359539.00 184.31340 \*  
## 27) avg\_month< 1.380629 156 1401479.00 278.79490 \*  
## 7) avg\_time>=0.02046875 311 7235195.00 348.60130   
## 14) avg\_month>=1.034329 56 629032.60 241.66070 \*  
## 15) avg\_month< 1.034329 255 5825086.00 372.08630   
## 30) diff\_attempts\_sec>=16.56427 93 1820793.00 317.33330   
## 60) total\_time< 3758.5 9 122658.20 128.55560 \*  
## 61) total\_time>=3758.5 84 1343037.00 337.55950 \*  
## 31) diff\_attempts\_sec< 16.56427 162 3565436.00 403.51850   
## 62) avg\_time< 0.03769371 139 2907666.00 391.07910   
## 124) total\_time< 2724.5 7 60803.71 168.57140 \*  
## 125) total\_time>=2724.5 132 2481916.00 402.87880 \*  
## 63) avg\_time>=0.03769371 23 506274.90 478.69570 \*

# you can even plot it!  
plot(m\_class\_tree, uniform = T)  
text(m\_class\_tree, use.n = F, all = TRUE, cex = .8)



plot(m\_reg\_tree, uniform = T)  
text(m\_reg\_tree, use.n = F, all = TRUE, cex = .8)



# prune the trees to avoid overfitting by limiting tree complexity  
cp\_class\_tree = m\_class\_tree$cptable[which.min(m\_class\_tree$cptable[,"xerror"]),"CP"]  
m\_class\_tree\_pruned = prune(m\_class\_tree, cp = cp\_class\_tree)  
  
cp\_reg\_tree = m\_reg\_tree$cptable[which.min(m\_reg\_tree$cptable[,"xerror"]),"CP"]  
m\_reg\_tree\_pruned = prune(m\_reg\_tree, cp = cp\_reg\_tree)  
  
#######################################  
#######################################

### Naive Bayes Classifier

**Question 8:** Fit an NB model using the naiveBayes() function from the e1071 package. The syntax is pretty similar to the linear/logistic regression models again.

* naiveBayes(binary\_outcome ~ predictor1 + predictor2 + predictor3, data = train)

Here’s an [R tutorial for naive bayes](https://www.r-bloggers.com/understanding-naive-bayes-classifier-using-r/).

#######################################  
####### BEGIN INPUT: Question 8 #######  
#######################################  
  
# install.packages("e1071") # you may need to install this first  
library(e1071)

## Warning: package 'e1071' was built under R version 4.0.5

m\_nb = naiveBayes(quiz300 ~ total\_time + avg\_time + avg\_correct\_first +   
 avg\_hints + avg\_attempts + avg\_month + std\_start\_d + std\_sec +   
 std\_attempts + std\_hints + diff\_attempts\_sec, data = train)  
  
# the output are a-prior and conditional probabilities  
m\_nb

##   
## Naive Bayes Classifier for Discrete Predictors  
##   
## Call:  
## naiveBayes.default(x = X, y = Y, laplace = laplace)  
##   
## A-priori probabilities:  
## Y  
## 0 1   
## 0.6419753 0.3580247   
##   
## Conditional probabilities:  
## total\_time  
## Y [,1] [,2]  
## 0 4613.013 2471.809  
## 1 4246.594 1231.743  
##   
## avg\_time  
## Y [,1] [,2]  
## 0 0.02138742 0.020444643  
## 1 0.02565082 0.007820622  
##   
## avg\_correct\_first  
## Y [,1] [,2]  
## 0 0.3782568 0.1779856  
## 1 0.4206130 0.1799327  
##   
## avg\_hints  
## Y [,1] [,2]  
## 0 0.7867323 0.5500877  
## 1 0.7626054 0.5278885  
##   
## avg\_attempts  
## Y [,1] [,2]  
## 0 1.536582 0.3655170  
## 1 1.462069 0.2487993  
##   
## avg\_month  
## Y [,1] [,2]  
## 0 1.7742352 1.796564  
## 1 0.8106656 1.047208  
##   
## std\_start\_d  
## Y [,1] [,2]  
## 0 5.473475 3.354011  
## 1 6.187867 2.911363  
##   
## std\_sec  
## Y [,1] [,2]  
## 0 69.52569 31.44866  
## 1 53.41628 19.21061  
##   
## std\_attempts  
## Y [,1] [,2]  
## 0 1.302407 0.6334712  
## 1 1.247020 0.5247868  
##   
## std\_hints  
## Y [,1] [,2]  
## 0 1.098688 0.4703358  
## 1 1.115146 0.3994785  
##   
## diff\_attempts\_sec  
## Y [,1] [,2]  
## 0 25.09943 17.163770  
## 1 17.28190 4.813268

#######################################  
#######################################

# Part 7: Model Evaluation

You just trained a number of models and now you want to know which model performs the best on the test set (holdout data). For simplicity, let us just focus on the classification models here.

**Question 9:** Get the predictions for each model using the predict() function where the type is ‘response’ for the logistic model and ‘class’ for the other models:

* predict(model, newdata = test, type = …)

#######################################  
####### BEGIN INPUT: Question 9 #######  
#######################################  
  
# logreg: this returns the probability of dropout, so assume that when Prob > 0.5 it means Dropout  
p\_logreg = as.numeric(predict(m\_logreg, newdata = test) > 0.5)  
  
# knn: this already has the prediction  
p\_knn = m\_knn   
  
# class tree  
p\_class\_tree = predict(m\_class\_tree, newdata = test, type = "class")  
  
# naive bayes  
p\_nb = predict(m\_nb, test)  
  
#######################################  
#######################################

**Question 10:** Now create a contingency matrix for each model and compute the accuracy, recall, and precision:

* Accuracy: (TruePos + TrueNeg) / total
* Recall: TruePos / (TruePos + FalseNeg)
* Precision: TruePos / (TruePos + FalsePos)

#######################################  
####### BEGIN INPUT: Question 10 ######  
#######################################  
  
# here is the confusion matrix for the logreg model  
cm\_logreg = table(true = test$quiz300, predicted = p\_logreg)  
  
# you generate the other ones  
cm\_knn = table(true = test$quiz300, predicted = p\_knn)  
cm\_class\_tree = table(true = test$quiz300, predicted = p\_class\_tree)  
cm\_nb = table(true = test$quiz300, predicted = p\_nb)  
  
# now compute accuracy, recall, and precision for each model  
tt <-length(test$quiz300)  
calc\_ac\_rc\_prec <- function(tb){  
 prec<-as.matrix(tb)  
 accuracy <- (prec[2,2] + prec[1,1])/tt #(TruePos + TrueNeg) / total  
 recall <- (prec[2,2]/(prec[2,2]+prec[2,1])) # TruePos / (TruePos + FalseNeg)  
 precision <- (prec[2,2]/(prec[2,2]+prec[1,2])) # TruePos / (TruePos + FalsePos)  
   
 sprintf('accuracy %f', accuracy)  
 sprintf('recall %f', recall)  
 sprintf('precision %f', precision)  
   
 return(c('accuracy:',accuracy, ',recall:',recall, ',precision:',precision))  
}  
  
lr\_res <- calc\_ac\_rc\_prec(cm\_logreg)  
knn\_res <- calc\_ac\_rc\_prec(cm\_knn)  
tree\_res <- calc\_ac\_rc\_prec(cm\_class\_tree)  
nb\_res <- calc\_ac\_rc\_prec(cm\_nb)  
  
print(lr\_res)

## [1] "accuracy:" "0.765027322404372" ",recall:"   
## [4] "0.493150684931507" ",precision:" "0.857142857142857"

print(knn\_res)

## [1] "accuracy:" "0.617486338797814" ",recall:"   
## [4] "0.547945205479452" ",precision:" "0.519480519480519"

print(tree\_res)

## [1] "accuracy:" "0.743169398907104" ",recall:"   
## [4] "0.684931506849315" ",precision:" "0.675675675675676"

print(nb\_res)

## [1] "accuracy:" "0.748633879781421" ",recall:"   
## [4] "0.849315068493151" ",precision:" "0.639175257731959"

#######################################  
#######################################

### Summarize your model evalution findings

**Question 11:** Which model has the highest/lowest accuracy, recall, precision? **####### BEGIN INPUT: Question 11 ######**

Accuracy - highest: Decision Tree, lowest: KNN; Recall - highest: Naive Bayes, lowest: Logistic Regression; Precision - highest: Logistic Regression, lowest: KNN.

Answer depends on feature engineering above.

**#####################################**

# Self-reflection

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

* I find this homework is a good reference for future work when building model in rstudio. One important thaing I learned is the creation of train and test are set differently, here we observe performance based on first 100 quizzes result.

# Submit Homework

This is the end of the homework. Please **Knit to Word**. The resulting file has to show both the R code and R output. Upload it on the EdX platform before the due date.