Predictive Models in Education

INFO 5200 Learning Analytics: Week 4 Homework

[[ADD YOUR NAME, CORNELL ID]]

The Dataset

For this homework, you will be analyzing the Assisstments dataset from last homework. You should be familiar with the properties of the data at this point. If I gave you new dataset, you would most likely be going through some of the same steps as in the previous homework to get familiar with the dataset. Below I'm copying some of the general info about the dataset from before just in case:

The dataset provides question-level data of students practicing math problems in academic year 2004-2005 using the Assisstments platform. 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 dataset:

```r
library(tidyverse, quietly = T)
```

```
## Warning: package 'tidyverse' was built under R version 3.4.2

## -- Attaching packages ---------------------------------- tidyverse 1.2.1 --

## v ggplot2 3.1.0     v purrr   0.2.5
## v tibble  1.4.2     v dplyr   0.7.8
## v tidyr   0.8.2     v stringr 1.2.0
## v readr   1.1.1     v forcats 0.2.0

## Warning: package 'ggplot2' was built under R version 3.4.4

## Warning: package 'tibble' was built under R version 3.4.3

## Warning: package 'tidyr' was built under R version 3.4.4

## Warning: package 'purrr' was built under R version 3.4.4

## -- Conflicts ------------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```r
# load info5200.2.assisstments.rds, use the readRDS() function.
asm = readRDS("info5200.2.assisstments.rds")
```

1. 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.

2. 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

3. Feature Engineering

This is where you create the dataset that you will be using 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.

(a) 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.

```r
# asm_outcomes = ...
asm_outcomes = asm %>%
    group_by(studentID) %>%
    summarise(
        num_quiz = n(),
        quiz300 = n() < 300
    )
```

```
## Warning: package 'bindrcpp' was built under R version 3.4.4
```

(b) 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 an 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.

```r
# 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)
```

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.

```r
# Now using the asm_sub dataset we can finally compute features like total time
asm_features = asm_sub %>%
    group_by(studentID) %>%
    summarise(
        total_time = sum(seconds),
        # TODO: You create some more features here for prediction
        count_quiz = n(),
        avg_time = mean(seconds),
        avg_time_sq = mean(seconds)^2,
        sd_time = ifelse(n_distinct(seconds) > 1, sd(seconds), 0),
        zero_time = mean(seconds == 0),
        avg_correct = mean(correctonfirstattempt),
        avg_correct_sq = mean(correctonfirstattempt)^2,
        avg_hints = mean(hints),
        avg_hints_sq = mean(hints)^2,
        sd_hints = ifelse(n_distinct(hints) > 1, sd(hints), 0),
        days_active = n_distinct(start_day)
    )

# 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
summary(asm_features)
```

```
##    studentID        total_time      count_quiz        avg_time
##  Min.   : 136.0   Min.   :   11   Min.   :  1.00   Min.   :  4.167
##  1st Qu.: 447.8   1st Qu.: 3202   1st Qu.:100.00   1st Qu.: 38.852
##  Median : 745.5   Median : 4426   Median :100.00   Median : 50.494
##  Mean   :1088.0   Mean   : 4462   Mean   : 85.58   Mean   : 56.143
##  3rd Qu.:1054.2   3rd Qu.: 5726   3rd Qu.:100.00   3rd Qu.: 66.635
##  Max.   :6802.0   Max.   :11264   Max.   :100.00   Max.   :343.500
##   avg_time_sq           sd_time         zero_time         avg_correct
##  Min.   :     17.36   Min.   :  0.00   Min.   :0.00000   Min.   :0.0000
##  1st Qu.:   1509.52   1st Qu.: 45.18   1st Qu.:0.00000   1st Qu.:0.2700
##  Median :   2549.61   Median : 59.89   Median :0.01112   Median :0.3900
##  Mean   :   4146.58   Mean   : 64.63   Mean   :0.01969   Mean   :0.3957
##  3rd Qu.:   4440.22   3rd Qu.: 78.78   3rd Qu.:0.03000   3rd Qu.:0.5100
##  Max.   :117992.25   Max.   :399.69   Max.   :0.40000   Max.   :1.0000
##  avg_correct_sq     avg_hints       avg_hints_sq        sd_hints
##  Min.   :0.0000   Min.   :0.0000   Min.   : 0.0000   Min.   :0.0000
##  1st Qu.:0.0729   1st Qu.:0.3479   1st Qu.: 0.1210   1st Qu.:0.8203
##  Median :0.1521   Median :0.6633   Median : 0.4400   Median :1.1859
##  Mean   :0.1881   Mean   :0.7645   Mean   : 0.8638   Mean   :1.1002
##  3rd Qu.:0.2601   3rd Qu.:1.1000   3rd Qu.: 1.2100   3rd Qu.:1.4276
##  Max.   :1.0000   Max.   :3.5000   Max.   :12.2500   Max.   :3.8513
##   days_active
##  Min.   : 1.000
##  1st Qu.: 2.000
##  Median : 3.000
##  Mean   : 3.413
##  3rd Qu.: 4.000
##  Max.   :15.000
```

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
```

4. 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).

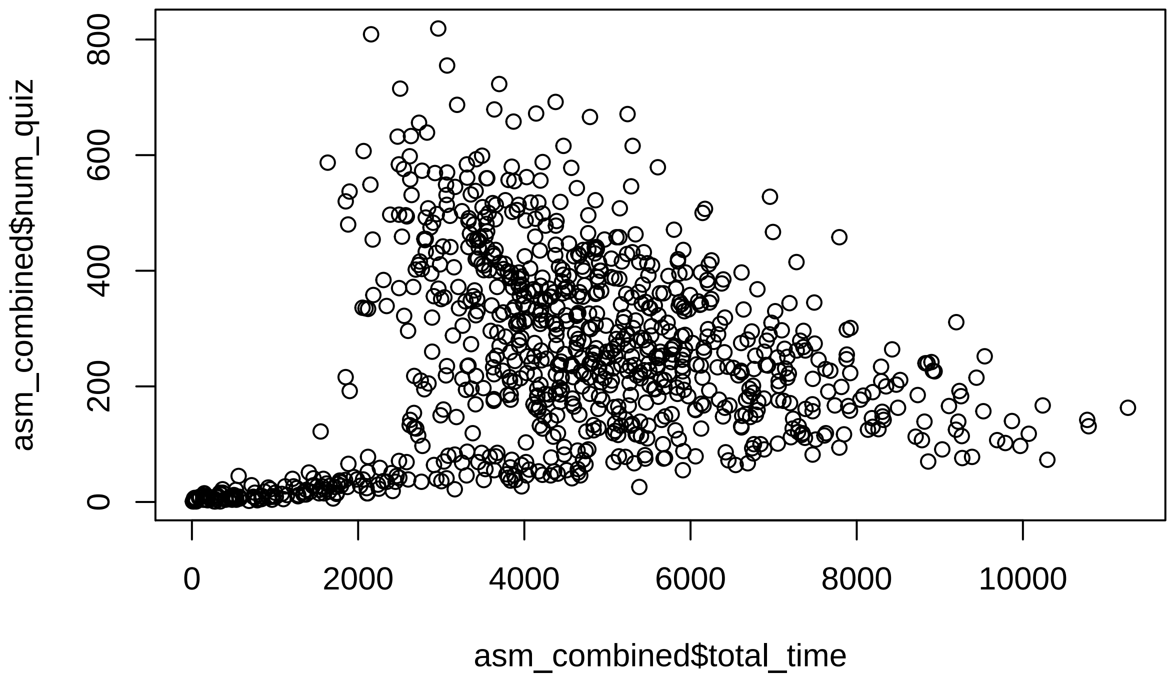
For now, I am assuming you created about 10-15 features in step 3. If you only have 5 or so, go back and come up with more.

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.

```
outcome_vars = c("num_quiz", "quiz300")
cor(asm_combined)[,outcome_vars]
```

```
##                   num_quiz     quiz300
## studentID      -0.40298119  0.24773508
## total_time      0.13147733  0.07463290
## count_quiz      0.64927941 -0.36768252
## avg_time       -0.40842415  0.32784553
## avg_time_sq    -0.29098145  0.20228079
## sd_time        -0.28657158  0.27106171
## zero_time      -0.13355812  0.08438442
## avg_correct     0.12723708 -0.12765339
## avg_correct_sq  0.08038016 -0.11634801
## avg_hints      -0.03318345  0.02968326
## avg_hints_sq   -0.06628572  0.03291985
## sd_hints        0.07110934 -0.01042106
## days_active    -0.07227974  0.21774127
## num_quiz        1.00000000 -0.82261259
## quiz300        -0.82261259  1.00000000
```

```
plot(asm_combined$total_time, asm_combined$num_quiz)
```



5. Model Selection / 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

Linear regression

To fit a linear regression model, use the lm() function like this: - lm(outcome ~ predictor1 + predictor2 + predictor3, data = train)

```
m_linreg = lm(num_quiz ~ . - quiz300 - studentID, data = train)

# the output are the coefficients:
m_linreg
```

```
##
## Call:
## lm(formula = num_quiz ~ . - quiz300 - studentID, data = train)
##
## Coefficients:
##    (Intercept)      total_time      count_quiz        avg_time
##     -7.354e+01      -3.243e-02       5.803e+00       5.755e-01
##    avg_time_sq         sd_time       zero_time     avg_correct
##     -6.708e-04       2.635e-01       7.061e+01      -2.646e+01
## avg_correct_sq       avg_hints    avg_hints_sq        sd_hints
##      9.860e+01      -2.439e+00       7.634e-01      -1.764e+01
##    days_active
##     -1.936e+01
```

Logistic regression

To fit a logistic regression model, use the glm() function like this: - glm(outcome ~ predictor1 + predictor2 + predictor3, data = train, family = "binomial")

```
m_logreg = glm(quiz300 ~ . - num_quiz - studentID, 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 ~ . - num_quiz - studentID, family = "binomial",
##     data = train)
```

```
##
## Coefficients:
##    (Intercept)      total_time      count_quiz        avg_time
##      3.544e+02       8.194e-03      -3.575e+00      -7.759e-01
##    avg_time_sq         sd_time       zero_time     avg_correct
##      5.166e-05      -1.167e-02      -5.818e+00      -7.782e-01
## avg_correct_sq       avg_hints    avg_hints_sq        sd_hints
##     -2.522e+00      -6.588e-01      -1.674e-02       6.192e-01
##    days_active
##      6.880e-01
##
## Degrees of Freedom: 728 Total (i.e. Null);  716 Residual
## Null Deviance:       962.1
## Residual Deviance: 598.3     AIC: 624.3
```

k Nearest Neighbor

To fit a kNN model, use 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)

Important: Do not forget to remove the studentID! It does not generalize well.

```r
# install.packages("class") # you may need to install this first
library(class)
m_knn = knn(train[,2:13], test[,2:13], train$quiz300, k = 5)

# the output are the predictions:
m_knn
```

```
##   [1] TRUE  FALSE FALSE FALSE TRUE  TRUE  TRUE  TRUE  TRUE  FALSE FALSE
##  [12] TRUE  FALSE TRUE  FALSE FALSE TRUE  FALSE FALSE TRUE  FALSE FALSE
##  [23] TRUE  TRUE  TRUE  FALSE FALSE FALSE TRUE  TRUE  FALSE FALSE TRUE
##  [34] TRUE  TRUE  TRUE  FALSE TRUE  FALSE TRUE  TRUE  FALSE TRUE  TRUE
##  [45] FALSE TRUE  FALSE TRUE  FALSE TRUE  FALSE TRUE  TRUE  TRUE  FALSE
##  [56] FALSE TRUE  TRUE  FALSE TRUE  TRUE  FALSE TRUE  FALSE TRUE  TRUE
##  [67] TRUE  TRUE  TRUE  TRUE  FALSE FALSE FALSE FALSE TRUE  TRUE  TRUE
##  [78] FALSE FALSE FALSE TRUE  TRUE  FALSE FALSE FALSE FALSE FALSE FALSE
##  [89] FALSE TRUE  TRUE  FALSE FALSE FALSE TRUE  FALSE TRUE  TRUE  FALSE
## [100] FALSE TRUE  FALSE FALSE TRUE  TRUE  TRUE  TRUE  TRUE  FALSE TRUE
## [111] FALSE TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [122] TRUE  FALSE FALSE FALSE FALSE TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [133] TRUE  TRUE  TRUE  FALSE TRUE  TRUE  FALSE FALSE TRUE  TRUE  FALSE
## [144] FALSE TRUE  TRUE  FALSE TRUE  FALSE TRUE  TRUE  FALSE TRUE  FALSE
## [155] FALSE TRUE  FALSE FALSE FALSE TRUE  TRUE  TRUE  FALSE TRUE  TRUE
## [166] TRUE  TRUE  TRUE  FALSE TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [177] TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## Levels: FALSE TRUE
```

Classification and Regression Trees

To fit a CART model, use 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.

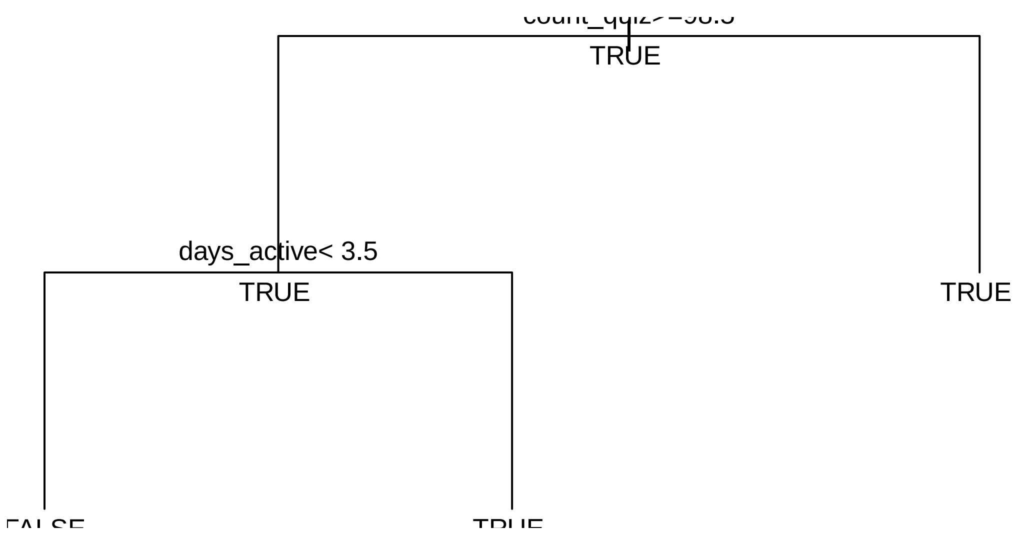
```
# install.packages("rpart") # you may need to install this first
library(rpart)
m_class_tree = rpart(quiz300 ~ . - num_quiz - studentID, data = train, method = "class")
m_reg_tree = rpart(num_quiz ~ . - quiz300 - studentID, data = train, method = "anova")

# the output are the decision trees
m_class_tree
```

```
## n= 729
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
## 1) root 729 271 TRUE (0.3717421 0.6282579)
##   2) count_quiz>=98.5 565 271 TRUE (0.4796460 0.5203540)
##     4) days_active< 3.5 302  83 FALSE (0.7251656 0.2748344) *
##     5) days_active>=3.5 263  52 TRUE (0.1977186 0.8022814) *
##   3) count_quiz< 98.5 164   0 TRUE (0.0000000 1.0000000) *
```

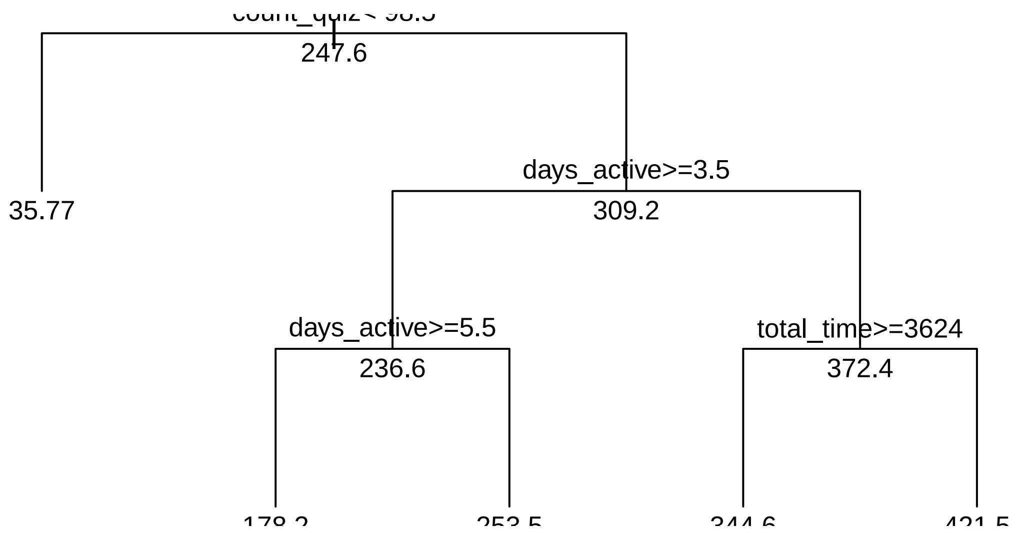
```
m_reg_tree
```

```
## n= 729
##
## node), split, n, deviance, yval
##       * denotes terminal node
##
##  1) root 729 19730050.0 247.64880
##    2) count_quiz< 98.5 164   131445.2  35.76829 *
##    3) count_quiz>=98.5 565 10099010.0 309.15040
##      6) days_active>=3.5 263  2363384.0 236.57790
##       12) days_active>=5.5 59   550450.3 178.16950 *
##       13) days_active< 5.5 204  1553439.0 253.47060 *
##      7) days_active< 3.5 302  5144187.0 372.35100
##       14) total_time>=3623.5 193  2354826.0 344.59590 *
##       15) total_time< 3623.5 109  2377429.0 421.49540 *
```

```
# you can even plot it!
plot(m_class_tree, uniform = T)
text(m_class_tree, use.n = F, all = TRUE, cex = .8)
```



```r
plot(m_reg_tree, uniform = T)
text(m_reg_tree, use.n = F, all = TRUE, cex = .8)
```



```r
# 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

To fit an NB model, use 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.

```r
# install.packages("e1071") # you may need to install this first
library(e1071)
```

```
## Warning: package 'e1071' was built under R version 3.4.4
```

```
m_nb = naiveBayes(quiz300 ~ . - num_quiz - studentID, data = train, method = "class")

# 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, method = "class")
##
## A-priori probabilities:
## Y
##     FALSE      TRUE
## 0.3717421 0.6282579
##
## Conditional probabilities:
##        studentID
## Y            [,1]      [,2]
##   FALSE  661.2952  424.9264
##   TRUE  1344.2664 1634.3621
##
##        total_time
## Y           [,1]     [,2]
##   FALSE 4265.694 1242.879
##   TRUE  4611.144 2527.281
##
##        count_quiz
## Y       [,1]     [,2]
##   FALSE  100  0.00000
##   TRUE    77 35.18566
##
##        avg_time
## Y           [,1]     [,2]
##   FALSE 42.65694 12.42879
##   TRUE  65.39991 38.96725
##
##        avg_time_sq
## Y           [,1]      [,2]
##   FALSE 1973.519  1173.095
##   TRUE  5792.280 11178.569
##
##        sd_time
## Y           [,1]     [,2]
##   FALSE 53.67716 19.58545
##   TRUE  71.89155 38.67707
##
##        zero_time
## Y             [,1]       [,2]
##   FALSE 0.01645756 0.01368958
##   TRUE  0.02219287 0.03532348
##
##        avg_correct
## Y             [,1]      [,2]
##   FALSE 0.4263838 0.1778624
##   TRUE  0.3771449 0.1761789
##
##        avg_correct_sq
## Y             [,1]      [,2]
##   FALSE 0.2133214 0.1600569
##   TRUE  0.1732095 0.1542904
##
##        avg_hints
## Y             [,1]      [,2]
##   FALSE 0.7389668 0.5157494
##   TRUE  0.7745972 0.5349662
##
##        avg_hints_sq
## Y             [,1]      [,2]
##   FALSE 0.8110878 0.9540938
##   TRUE  0.8855648 1.2405847
##
##        sd_hints
## Y            [,1]      [,2]
##   FALSE 1.096545 0.4180750
##   TRUE  1.093978 0.4528465
##
##        days_active
## Y            [,1]      [,2]
##   FALSE 2.940959 0.8883096
##   TRUE  3.679039 1.9878288
##
##        num_quiz
## Y            [,1]     [,2]
##   FALSE 422.8081 96.50795
##   TRUE  144.0066 93.33513
```

6. 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.

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 = …)

```
# logreg: this returns the probability of dropout, so you can set Prob > 0.5 to mean Dropout
p_logreg = predict(m_logreg, newdata = test, type = "response") > 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, newdata = test, type = "class")
```

Now you can 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)

```r
# here is the confusion matrix for the logreg model:
cm_logreg = table(true = test$quiz300, predicted = p_logreg)
cm_knn = table(true = test$quiz300, predicted = p_knn)
cm_class_tree = table(true = test$quiz300, predicted = p_class_tree)
cm_nb = table(true = test$quiz300, predicted = p_nb)

# convenience function for evaluation of confusion matrix
cm_eval = function(cm) {
    list(
        accur = sum(diag(cm)) / sum(cm),
        recall = cm[2,2] / sum(cm[2,]),
        precision = cm[2,2] / sum(cm[,2])
    )
}

cm_eval(cm_logreg)
```

```
## $accur
## [1] 0.8032787
##
## $recall
## [1] 0.825
##
## $precision
## [1] 0.8684211
```

```r
cm_eval(cm_knn)
```

```
## $accur
## [1] 0.7322404
##
## $recall
## [1] 0.7583333
##
## $precision
## [1] 0.8198198
```

```r
cm_eval(cm_class_tree)
```

```
## $accur
## [1] 0.8032787
##
## $recall
## [1] 0.8083333
##
## $precision
## [1] 0.8818182
```

```r
cm_eval(cm_nb)
```

```
## $accur
## [1] 0.6557377
##
## $recall
## [1] 0.475
##
```

```
## $precision
## [1] 1
```

Briefly summarize your findings

Which model has the highest/lowest accuracy, recall, precision?

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.