Machine Learning in R - Predicting Student's Academic Success in Introductory Math Courses at Knox College

Name: Thu Anh Nguyen
MATH 361: Senior Research
Date: 03/12/19

Introduction

In the diverse world in which we inhabit, there's a lot of data out there waiting to be recorded and analyzed. However, we can only do so much when we have to deal with a massive dataset. The work itself is very tedious, and too complex that it is impractical for us handle. Thankfully, the development of Artificial Intelligence (AI) has enabled us to advance in our studies. While AI is a broad science of mimicking human abilities, Machine Learning is a subset of AI that teaches machines how to make inferences based on patterns in a given dataset (SAS 2019). In particular, machine learning algorithms build a mathematical model on the dataset, and make predictions by inputting the test dataset into model. According to Lantz, the author of "Machine Learning with R", previous applications of Machine Learning involve identifying spam messages in emails, predicting the outcome of elections, discovering genetic sequences in diseases, and many more (Lantz 2015). Some machine algorithms the author covered in his book include Naive Bayes classification, k Nearest Neighbors, Decision Trees, and Logistic Regression. For the purpose of our research project, we will predict the academic success of students in mathematics courses at Knox College using Logistic Regression, and Naive Bayes classification.

What is Logistic Regression?

Logistic regression is a classification algorithm used for dichotomous dependent variables. Instead of modeling the actual outcome of the dependent variable, we are modeling the probability of the occurrence of that particular outcome (Forsberg 2018). To predict the actual outcome, we would have to determine a threshold probability in which we would use to distinguish between a success and failure. A typical model for Logistic Regression is shown below:

$$logit(y) = log(\frac{y}{1-y}) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_n x_n$$

, where x_1, x_2, ..., x_n are our linear predictors.

Solving for *y*

$$y = \frac{e^{\beta_0+\beta_1 x_1+\beta_2 x_2+...+\beta_n x_n}}{1 + e^{\beta_0+\beta_1 x_1+\beta_2 x_2+...+\beta_n x_n}}$$

, gives a success probability. Because the exponential function is a non-decreasing function, the probabilities are guaranteed to be positive.

Logistic Regression is also a part of a larger class called Generalized Linear Model, which is an extension of the Classical Linear Model. The CLM assumes that there is a linear relationship between the dependent variable and the independent variable. However, we can't always presume that that is true with our data, so it is best to use GLM as a precaution. The GLM requires that the model maintains the following three requirements: linear predictors (independent variables), conditional distribution for the dependent variable, and a link function that links the expected value of the dependent variable with the linear predictor (Forsberg 2018). The link function must also be bijective, meaning the link function, and it's inverse must be functions.

What is Naive Bayes:

Naive Bayes is another classification method commonly used in detecting ham and spam messages in text messages or emails. The classification algorithm itself relies on Bayes Theorem to predict the most likely class for the new features in our model. It is "Naive", because the algorithm works under the assumption that predictor variables are independent and equally important (Lantz 2015). We have reasons to feel skeptical about this faulty assumption, but Naive Bayes classification has been proven very effective in the past. Recall the definition of Bayes Theorem, which states the following:

$$\mathbb{P}\,[A|B] = \frac{\mathbb{P}\,[A \cap B]}{\mathbb{P}\,[B]} = \frac{\mathbb{P}\,[B|A]\,\mathbb{P}\,[A]}{\mathbb{P}\,[B]}$$

, where P(A|B) is the posterior probability, P(A) is the prior probability, P(B|A) is the likelihood, and P(B) is the margin of likelihood. Once we have determined a threshold that distinguishes between a success and failure, we would classify the posterior probability according to that cut-off probability. Despite the efficiency of the method, it does not work for datasets with numeric predictor variables. It is possible to counter this weakness by discretizing the numerical independent variables. Discretizing could be done by assigning categories based on the cut points in the distribution of the data (Lantz 2015). Unlike the Logistic Regression, Naive Bayes handles missing values and noisy data quite efficiently. Thus, we would expect it to perform better than the Logistic Regression. We can't say for sure until we have obtained a side-by-side comparison of the two methods on our dataset.

Background on dataset

Before we can apply Logistic Regression, and Naive Bayes classification, we must know and understand our dataset. The math placement dataset consists of 1052 data points that were recorded over the course of 10 years at Knox College (2008-2017) by Professor Hastings. Each variable in the dataset stores information on Knox students' high school records, their personal background, as well as information regarding their performance in the introductory math courses. Students in the past have either chosen or have been placed in MATH 121, MATH 131, MATH 145, and MATH 151. Our objective is to assign students to the right math courses by making inferences on Knox's historical data. The task is indeed tricky business, but nevertheless, it has to be done. In comparison with the toy examples from our textbook, our placement dataset was quite an eye sore. There are a considerable amount of missing data in each variable, which could prove problematic when we train our model and use it for predictions. Aside from the missing data, we also don't have large enough sample sizes for MATH 131, and MATH 145 to split into training, and testing datasets. Therefore, it is only reasonable to apply the classification methods on MATH 151 and MATH 121 dataset.

Preparations and Filling in Missing Data:

Cleaning the dataset was perhaps the most important part about this research project. We simply cannot move forward with the classification methods unless the dataset set has been cleaned and the appropriate variables have been chosen for our models. After examining the dataset, we decided to remove the student's citizenship status, as there were a far too many missing data. It is also unnecessary to keep high school size, because high school percentage and academic rating are more valuable to our research question. Now that we have removed the unnecessary variables, we can regress the students' grade in each course as a function of each of the remaining variables in the dataset in order to determine

their significance. Because the grades in each course are recorded as letter grades, it is necessary to assign them GPA equivalents. Once the GPA equivalents have been added as a new column in our data frame, we can begin modeling the grades in each course using Ordinary Least Squares. One example of this is modeling the grades the students got in MATH 151 using their high school GPA. Because the p-value of the model is less than 0.05, we reject the null hypothesis that there is no relationship between the students' grades in MATH 151, and their high school GPA. In addition to high school GPA, other variables such as ACT composites, ACT math, and academic rating had a significant effect on both MATH 151, and MATH 121. Despite having a great impact on the grades, SAT math and ACT composites contained a lot of missing data points. In order to resolve this issue, we computed the missing ACT scores using a regression line created from Ordinary Least Squares. For example, we had reasons to believe that the ACT composites scores are reliant on the SAT verbal scores and the SAT math scores. A summary of this model confirms this with p-values much smaller than 0.05 for the two independent variables, thereby suggesting that each has a significant effect on the ACT composite scores. The regression table also illustrated an adjusted R-Squared of approximately 0.7, indicating that 70% of variation was explained by the model. With this regression line, we are guaranteed to make pretty good predictions about the ACT composite scores. We then create a function called `createnewactc()`, which computes the ACT composite scores if both SAT scores are available, and stores them as new elements in a temporary array. If the ACT composite score is already present in the dataset, we simply need to add it to our array. We then create a new column with the new ACT composite scores just as we did before. We would go through the same procedure to fill in the missing SAT math scores, which is a relatively straight-forward process. After executing numerous linear regressions, we managed to pick out several independent variables that had a significant effect on the students' grades in MATH 151, and MATH 121. Some of these variables include academic rating, high school, ACT scores, and so on.

Creating test and training datasets:

With our data all ready for analysis, we need to split our dataset into training and test datasets. We are essentially building the model based on the train dataset, and evaluating the test dataset using that model in order to obtain predictions. Suppose we want to split the MATH 151 dataset into training and test dataset. We would divide MATH 151 into two portions: 75% for training, and 25% for testing. Because the ACT scores in our placement dataset are sorted in ascending order, it is important that we randomly select values for each dataset. To elaborate, we would randomly sample 269 values from our MATH 151 dataset, and assign it to the training dataset. The remaining unselected values are designated to the test dataset. We would repeat the same procedure for splitting our MATH 121 dataset.

Application of Logistic Regression:

We have finally reached the fun and interesting part about machine learning, which is the actual execution of the classification algorithm. Since MATH 151, and MATH 121 have been separated into two subsets of the placement dataset, we would implement the Logistic Regression on each subset. This may also mean that each model uses different independent variables depending on their significance with respect to the dependent variable. However, the variable that we're modeling is no longer the students' grade in the respective class. Instead, we're modeling their success in the class. Success in this context is open for interpretation, so we came up with our own definition of success. A student is considered successful if they earned a C+ or better in the course.

Let us begin modeling the students' success in the MATH 151 training dataset as a function of the independent variables that were significant in our findings with linear regressions. Note that we're now

using Generalized Linear Model instead of the Classical Linear Model. An acceptable link function for this model would most definitely be the logit function, because we're predicting a probability, which varies between 0 and 1. This could be carried out using a function in R called `glm()`. Since there are too many independent variables, it would be more ideal to model success with different combinations of the linear predictors to select the ones that are most valuable to our prediction. We eventually narrowed down our list potential variables to high school GPA, and placement 2 scores as the linear predictors of our MATH 151 model. We would have expected academic rating to be included as well, but it was overshadowed when combined with high school GPA and placement 2 in the model. Once we have an appropriate model, we would use the `predict()` function to make predictions using the values from the MATH 151 test dataset. Because we want a success probability, it is necessary to back-transform our predicted values with the logistic function.

Now that we have our predicted values in terms of probabilities, we must classify them with a threshold. We've decided on a threshold probability of 0.75, since it produced the best predictions. The function `classifysuccess()` classifies the predicted probability as a 1 (success) if it is greater than 0.75, and assigns an "NA" when it is a missing value, and returns a 0 (failure) otherwise. Using the function `CrossTable()`, we can compare the predictions with the actual values. Calling this function outputs the following table:

	actual		
predicted	0	1	Row Total
0	26	17	43
	0.605	0.395	0.768
	0.867	0.654	
1	4	9	13
	0.308	0.692	0.232
	0.133	0.346	
Column Total	30	26	56
	0.536	0.464	

According to the table, a total of only 26+9=35 out of the 56 students were correctly classified, where as 17+4=21 out of 56 were misclassified (37.5 percent). Among the 21 that were misclassified, 4 students were predicted to be successful, when in reality, they failed MATH 151 by our definition of success. Generally we would want to avoid making this error, because we wouldn't want to overplace students, and jeopardize their grades. Out of curiosity, we also redefined our definition of success to a success being a C- or better. We expected the model to perform much better than the previous one, but it actually increased the number false positives by 1.

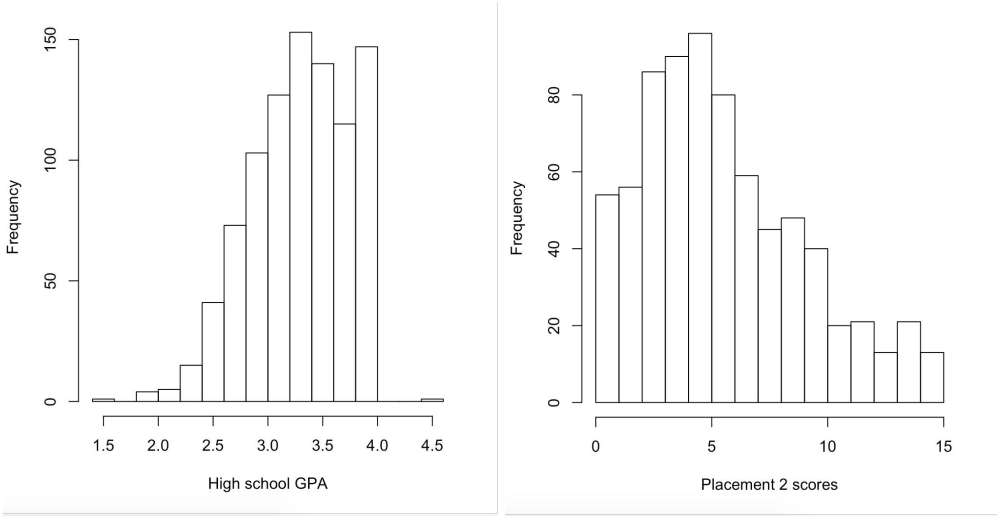
The rest is relatively straightforward once we've gone through Logistic Regression with the MATH 151 dataset. All that remains for us to do is to repeat the same procedure with MATH 121. However, our linear predictors are no longer the same as the ones for the MATH 151 training dataset. A summary of the MATH 121 model suggests that students' high school GPA, placement 1 scores, and ACT composites scores had a significant impact on their success in the course. Again, we would use the model to generate the predictions with values from MATH 121 test, and compare them with the true value. A confusion matrix of the predicted values and true values is illustrated below:

	actual		
predicted	Failure	Success	Row Total
0	24	25	49
	0.490	0.510	0.803
	0.889	0.735	
1	3	9	12
	0.250	0.750	0.197
	0.111	0.265	
Column Total	27	34	61
	0.443	0.557	

The accuracy is not that great for this model, as only 54% of the students were correctly identified. But that is okay, because we mainly care about reducing the number of false positives. Among the 28 misidentified classes, only 3 students were over-placed. Even though there are more misclassified classes than the MATH 151 model, we still can rejoice over the fact that there is a small number false positives.

Application of Naive Bayes

Allow us shift gears and implement Naives Bayes classification algorithm on our datasets. Out of habit, we will start off with the MATH 151 dataset. Because the linear predictors we used in Logistic Regression were significant to the students success, it makes sense to use them as features in our Naive Bayes model. Recall that Naive Bayes classification only works with categorical features. Since our features are numeric, we're going to encounter a problem. In order to fix this issue, we will examine the distribution of high school GPA, and placement 2 scores. A histogram of the distribution of each independent variable is shown below:



Based on the distribution of placement 2 scores, a reasonable cut-point for each category would be:

Category 1: $0 \leq x_1 \leq 4$

Category 2: $5 \leq x_1 \leq 7$

Category 3: $8 \leq x_1 \leq 15$

Similarly, we would define the three categories for the high school GPA as:

Category 1: $0.0 < x_2 \leq 2.7$

Category 2: $2.7 < x_2 \leq 3.4$

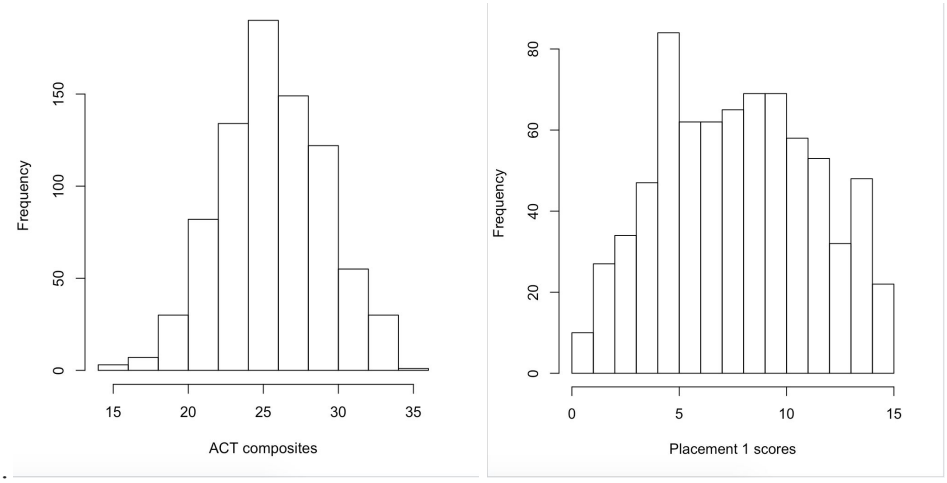
Category 3: $3.4 < x_2 \leq 4.5$

After we have discretized our numeric features, we can create a classifier with the function `naivebayes()`. Here's a little insight on how the Naive Bayes algorithm would work. Let B_1 be the event that the placement 2 category is equal to 1, 2, or 3, and B_2 be the event that the category for high school GPA is a 1, 2, or 3. Suppose we want to calculate the probability that a student is successful, given that B_1=1 and B_2=2. We would then apply Bayes Theorem to our posterior probability to further expand it. Since Naive Bayes classification works under the assumptions that predictors are independent, the joint probability of B_1=1 and B_2=2 can be computed as the product of P(B_1=1) and P(B_2=2). These values could easily be obtained from a frequency table for MATH 151 successes. Once we finish building our model, we can use the `predict()` function to make our predictions. The function `classifysuccessBayes()` then classifies the predicted probabilities as a success if it is greater or equal to 0.75. A confusion matrix of the actual and predicted outcomes is displayed below:

	actual		
predicted	Failure	Success	Row Total
0	26	22	48
	0.542	0.458	0.533
	0.591	0.478	
1	18	24	42
	0.429	0.571	0.467
	0.409	0.522	
Column Total	44	46	90
	0.489	0.511	

Looking at this table, we can see that 26+24=50 out of 90 observations were correctly classified (55.6 percent), where as 18+22=40 were misidentified (44.4 percent). Unfortunately, the Naive Bayes classification didn't perform as well as we had hoped it would. Among the 40 misclassified students, 18 of them were not placed into a class of their level.

The Naive Bayes classification of MATH 121 could be done in the exact same ways. First thing on our agenda is to discretize our predictor variables, which are high school gpa, placement 1 scores, and ACT composite score. For our convenience , we will be using the same categorization for high school GPA from our MATH 151 model. Technically, we would only have to examine the distribution of placement 1 scores, and ACT composite scores.



According to the first histogram, the most suitable categorization for ACT composites would be:

Category 1: $0 \leq x_3 \leq 21$

Category 2: $22 \leq x_3 \leq 27$

Category 3: $28 \leq x_3 \leq 36$

Similarly, we can create the following bins for placement 1 scores:

Category 1: $0 \leq x_2 \leq 5$

Category 2: $6 \leq x_2 \leq 9$

Category 3: $10 \leq x_2 \leq 15$

Next, we train the classifier using `naivebayes()`, and make predictions with `predict()`. We would then classify the predicted probability as either success or failure, and compare our results with the actual values through a confusion matrix.

	actual		
predicted	Failure	Success	Row Total
Failure	31	22	53
	0.585	0.415	0.393
	0.574	0.272	
Success	23	59	82
	0.280	0.720	0.607
	0.426	0.728	
Column Total	54	81	135
	0.400	0.600	

Unfortunately, we're still observing quite a few false positives in our model, which is concerning. In order to improve our model, we could increase the Laplace estimator by 1 to see whether that would reduce the number of false positives.

Conclusion

Out of the two classification algorithms, Logistic Regression performed a lot better for both of our MATH 151 and MATH 121 datasets in terms of reducing the false positive errors. Despite having higher accuracy for both datasets, Naive Bayes classification incorrectly predicted a concerning amount of successes, when in reality they were failures. If time permits, we could try improving our models by introducing a cost matrix, which assigns penalties to each error. Since the error of over-placing is more severe than under-placing students, we would assign it a higher cost. This discourages the algorithm from making this error, thereby, minimizing the number of false positives. Despite all the blood, sweat and tears that had gone into this project, the current placement model that the Math department has been utilizing outshines both of our models. Perhaps the key to making good predictions is to keep the model as simple as possible, even if it means using Ordinary Least Squares.

Reference list:

"Machine Learning: What It Is and Why It Matters." *What It Is and Why It Matters | SAS*, SAS, www.sas.com/en_us/insights/analytics/machine-learning.html.

"Machine Learning with R: Discover How to Build Machine Learning Algorithms, Prepare Data, and Dig Deep into Data Prediction Techniques with R." *Machine Learning with R: Discover How to Build Machine Learning Algorithms, Prepare Data, and Dig Deep into Data Prediction Techniques with R*, by Brett Lantz, Packt Publishing, 2015, pp. 1–124.

Forsberg, Ole. "Linear Models and Rurita Kralovstvi." *Linear Models and Rurita Kralovstvi*, 2018, pp. 276–287.

R Code:

```
#Read in data
placement <- read.csv("PlacementDataNoID.csv", stringsAsFactors =
FALSE)
str(placement)

attach(placement)
library(dplyr)
library(gmodels)
library(caTools)
library(e1071)

grade<-factor(grade, levels = c("A+","A","A-","B+","B",
"B-","C+","C","C-","D+","D","D-","F"))

#convert letter grades to GPA equivalents
placement<-mutate(placement,gradegpa=case_when(grade =="A+"~ 4.0,
grade =="A"~ 4.0,grade =="A-"~ 3.7,grade =="B+"~ 3.3,grade =="B"~
3.0,
grade =="B-"~ 2.7,grade =="C+"~ 2.3,grade =="C"~ 2.0,grade =="C-"~
1.7,
```

```
grade =="D+"~ 1.3,grade =="D"~ 1.0, grade =="D-"~ 0.7,grade =="F"~
0.0,
grade == "W"~ 0.0)
#separate as subsets of placement dataset
math151 <- subset(placement,course == "M151")

math121 <- subset(placement,course == "M121")


#Linear Regressions

mod1a = lm(math151$gradegpa~math151$hsgpa)
plot(math151$hsgpa,math151$gradegpa)
summary(mod1a)#0.12 variation explained
cor(math151$hsgpa,math151$gradegpa, method ="pearson",use
="complete.obs") # 0.350724 -> use variable in model


mod1b = lm(math121$gradegpa~math121$hsgpa)
plot(math121$hsgpa,math121$gradegpa)
summary(mod1b)#0.099 variation explained
cor(math121$hsgpa,math121$gradegpa, method ="pearson",use
="complete.obs") #0.3188458 -> use variable in model


modact = lm(actma~satm)
plot(satm,actma)

summary(modact)
#correlation of 0.7304357
cor(satm,actma, method ="pearson",use ="complete.obs")
#regression line: 1.773941 + 0.040874 x
#if else(assign)

#function that predict new act math scores
createactma <- function(actma,satm)
{
  results <-vector(mode="numeric",length=1052)
for (i in 1:1052)
{
if (is.na(actma[i]) && is.na(satm[i])==FALSE) {
  results[i]=round(1.773941 + 0.0408748*satm[i])
}
  else {
  results[i] = actma[i]
  }
}
  return (results)
```

```
}
#add newact scores to dataframe-> use variable in model
placement<-mutate(placement,newactm=createactma(actma,satm))
# test relationship between actc and grade in math class

mod2a = lm(math151$gradegpa~math151$actc)
summary(mod2a) #there's a relationship -> keep actc 0.044 variation
explained
cor(math151$actc,math151$gradegpa, method ="pearson",use
="complete.obs") #0.2226114 -> use variable in model

mod2b = lm(math121$gradegpa~math121$actc)
summary(mod2b) #0.1251 explained variation
cor(math121$actc,math121$gradegpa, method ="pearson",use
="complete.obs") #0.357-> use variable in model

mod3a = lm(math151$gradegpa~math151$actsc)
summary(mod3a) #there's no relationship, only 0.01499  of variation
is explained
plot(math151$actsc,math151$gradegpa)
cor(math151$actsc,math151$gradegpa, method ="pearson",use
="complete.obs") #0.1421173

mod3b = lm(math121$gradegpa~math121$actsc)
summary(mod3b) #there's a relationship, only  0.08834  of variation
is explained
plot(math121$actsc,math121$gradegpa)
cor(math121$actsc,math121$gradegpa, method ="pearson",use
="complete.obs") #0.3020741

mod4a = lm(math151$gradegpa~math151$satv)
summary(mod4a) #0.002301 variation explained and there's no
relationship
cor(math151$satv,math151$gradegpa, method ="pearson",use
="complete.obs") #0.1051304

mod4b = lm(math121$gradegpa~math121$satv)
summary(mod5) #0.001052 variation explained and there's no
relationship
cor(math121$satv,math121$gradegpa, method ="pearson",use
="complete.obs") #0.1046084

mod5a = lm(math151$gradegpa~math151$acten)
summary(mod5b)#0.02704  variation explained
cor(math151$acten,math151$gradegpa, method ="pearson",use
="complete.obs") #0.1794012 correlation

mod5b = lm(math121$gradegpa~math121$acten)
```

```
summary(mod5b)#0.02704  variation explained
cor(math121$acten,math121$gradegpa, method ="pearson",use
="complete.obs") #0.2624768 correlation-> *use variable in model

mod6 = lm(actc~satv + satm)
summary(mod6)

#fill in missing ACT composite scores
createactc <- function(actc,satv,satm)
{
  results <-vector(mode="numeric",length=1052)
  for (i in 1:1052)
  {
    if (is.na(actc[i]) && is.na(satv[i])==FALSE &&
is.na(satm[i])==FALSE) {
     results[i]=round(2.235279 +0.025336*satv[i] + 0.014926*satm[i])
    }
    else {
      results[i] = actc[i]
    }
  }
  return (results)
}
#-> *use variable in model
placement<-mutate(placement,newactc=createactc(actc,satv,satm))

#update dataset after new var added
math151 <- subset(placement,course == "M151")

math121 <- subset(placement,course == "M121")

mod7a = lm(math151$gradegpa~math151$place2)
summary(mod7a) #0.1313 variation explained
cor(math151$place2,math151$gradegpa, method ="pearson",use
="complete.obs") #0.3671207 -> *use variable in model

mod7b = lm(math121$gradegpa~math121$place2)
summary(mod7b) #0.01932 variation explained
cor(math121$place2,math121$gradegpa, method ="pearson",use
="complete.obs") #0.1480405

mod8a = lm(math121$gradegpa~math121$place1)
summary(mod8) #0.09  variation explained
cor(math121$place1,math121$gradegpa, method ="pearson",use
="complete.obs") #0.3039853-> *use variable in model

mod8b = lm(math151$gradegpa~math151$place1)
summary(mod8b) #0.1325   variation explained
```

```
cor(math151$place1,math151$gradegpa, method ="pearson",use
="complete.obs") #0.3687728-> *use variable in model


mod9a = lm(math151$gradegpa~math151$hspercen)
summary(mod9a) #0.08991 variation explained
cor(math151$hspercen,math151$gradegpa, method ="pearson",use
="complete.obs") #0.3091967 -> *use variable in model

mod9b = lm(math121$gradegpa~math121$hspercen)
summary(mod9b) #0.05446 variation explained
cor(math121$hspercen,math121$gradegpa, method ="pearson",use
="complete.obs") #0.2408925 -> *use variable in model

mod10a = lm(math151$gradegpa~math151$acadrate)
summary(mod10a) #0.1487  variation explained
cor(math151$acadrate,math151$gradegpa, method ="pearson",use
="complete.obs") #-0.389499 -> *use variable in model

mod10b = lm(math121$gradegpa~math121$acadrate)

summary(mod10b) # 0.12  variation explained
cor(math121$acadrate,math121$gradegpa, method ="pearson",use
="complete.obs") #-0.3488608 -> *use variable in model

plot(math121$acadrate,math121$gradegpa)

mod11a = lm(math151$gradegpa~math151$actr)
summary(mod11a) # 0.02842    variation explained
cor(math151$actr,math151$gradegpa, method ="pearson",use
="complete.obs") #0.1831997
plot(math151$actr,math151$gradegpa)

mod11b = lm(math121$gradegpa~math121$actr)
summary(mod11b) # 0.0734   variation explained
cor(math121$actr,math121$gradegpa, method ="pearson",use
="complete.obs") #0.2763349 -> *use variable in model
plot(math121$actr,math121$gradegpa)

mod12a = lm(math151$gradegpa~math151$newactc)
summary(mod12a) # 0.01719   variation explained
cor(math151$newactc,math151$gradegpa, method ="pearson",use
="complete.obs") #0.1437607 -> *use variable in model
plot(math151$newactc,math151$gradegpa)

mod12b = lm(math121$gradegpa~math121$newactc)
summary(mod12b) # 0.0998    variation explained
```

```
cor(math121$newactc,math121$gradegpa, method ="pearson",use
="complete.obs") #0.3195381 -> *use variable in model
plot(math121$newactc,math121$gradegpa)

mod13a = lm(math151$gradegpa~math151$newactm)
summary(mod13a) #0.0244  variation explained
cor(math151$newactc,math151$gradegpa, method ="pearson",use
="complete.obs") #0.1647298

mod13b = lm(math121$gradegpa~math121$reclevel)
summary(mod13b) #0.1024    variation explained
cor(math121$reclevel,math121$gradegpa, method ="pearson",use
="complete.obs") #0.3226322
plot(math121$reclevel,math121$gradegpa)

mod14a = lm(math151$gradegpa~math151$newactm)
summary(mod14a) #0.05593     variation explained
cor(math151$newactm,math151$gradegpa, method ="pearson",use
="complete.obs") #0.2434441
plot(math151$newactm,math151$gradegpa)

mod14b = lm(math121$gradegpa~math121$newactm)
summary(mod14b) #0.1148     variation explained
cor(math121$newactm,math121$gradegpa, method ="pearson",use
="complete.obs") #0.3421434
plot(math151$newactm,math151$gradegpa)

mod15a = lm(math151$gradegpa~math151$newactm + math151$newactc)
summary(mod15a) #0.05263 variation explained
cor(math151$newactm,math151$gradegpa, method ="pearson",use
="complete.obs") #0.3421434

mod15b = lm(math121$gradegpa~math121$newactm + math121$newactc)
summary(mod15b) #0.1302 variation explained, both actc and actma are
sig
cor(math121$newactm,math121$gradegpa, method ="pearson",use
="complete.obs") #0.3421434 -> *use variable in model

mod16a = lm(math151$gradegpa~math151$place1 + math151$place2)
summary(mod16a) #0.1568 variation explained
cor(math151$place1,math151$place2,math151$gradegpa, method
="pearson",use ="complete.obs") #0.3421434

mod16b = lm(math121$gradegpa~math121$place1 + math121$place2) #place1
is statistically sig
summary(mod16b) #0.08916  variation explained
```

```
mod17a = lm(math121$gradegpa~math121$actr + math121$acten) #actr is
statistically sig
summary(mod17a) #0.07918  variation explained


#Logistic regression

#defining success
codedsuccess  <- function(gradegpa)
{
  results <-vector(mode="numeric",length=1052)
  for (i in 1:1052)
  {
    if (is.na(placement$gradegpa[i])==FALSE && placement$gradegpa[i]
>= 2.3){
      results[i] = 1
    }
    else{
      results[i] = 0
    }
  }
  return (results)
}

placement<-mutate(placement,success=codedsuccess(placement$gradegpa))

#update dataset after new var added
math151 <- subset(placement,course == "M151")

math121 <- subset(placement,course == "M121")

#split training and test dataset MATH 151
smp_size <- floor(0.75*nrow(math151))
set.seed(123)
train_ind<-sample(seq_len(nrow(math151)),size=smp_size)

math151_train<-math151[train_ind,]
math151_test<-math151[-train_ind,]

#final model
logitmod151 = glm(success ~ hsgpa + place2,
family=binomial(link=logit),data=math151_train )

#predicted probabilities
predmodLgt = predict(logitmod151, newdata=math151_test)
predmod = logistic(predmodLgt)

#classify predicted probabilities
```

```
classifysuccess <- function(predmod){
  results <-vector(mode="numeric",length=length(predmod))
  for (i in 1:length(predmod)){
    if (is.na(predmod[i])==FALSE && predmod[i]>=0.75){
      results[i] = 1
    }
    else if(is.na(predmod[i])==TRUE){
      results[i] = NA
    }
    else{
      results[i] = 0
    }
  }
  return (results)
}
math151_test_pred = classifysuccess(predmod)

#confusion matrix
CrossTable(math151_test_pred,math151_test$success, prop.chisq =
FALSE,prop.t=FALSE,dnn = c('predicted','actual'))


#split training and test dataset MATH 121
smp_size <- floor(0.75*nrow(math121))
set.seed(123)
train_ind<-sample(seq_len(nrow(math121)),size=smp_size)

math121_train<-math121[train_ind,]
math121_test<-math121[-train_ind,]

#final mod
logitmod121 = glm(success ~ newactc +  hsgpa + place1,
family=binomial(link=logit),data=math121_train)
summary(logitmod121)

predmodLgt121 = predict(logitmod121,newdata=math121_test)
predmod121 = logistic(predmodLgt121)

math121_train_pred = classifysuccess(predmod121)

#confusion matrix
CrossTable(math121_train_pred,math121_test$success, prop.chisq =
FALSE,prop.t=FALSE,dnn = c('predicted','actual'))

###**** Naive Bayes
#factorize success and failure
placement$success<-factor(placement$success, levels = c(0,1),labels =
c("Failure","Success"))
```

```
#distribution
hist(hsgpa,xlab = "High school GPA")
hist(place2,xlab = "Placement 2 scores")
#binning high school gpa
placement<-mutate(placement,hsgpacat=case_when((hsgpa <= 4.5 &
hsgpa>3.4)~3, (hsgpa <= 3.4 & hsgpa>2.7)~2, (hsgpa <= 2.7 &
hsgpa>=0.0)~1))

#binning place2
placement<-mutate(placement,place2cat=case_when((place2 <= 4 &
place2>=0)~1, (place2 <= 7 & place2>=5)~2, (place2 <= 15 &
place2>=8)~3))

#updating dataset
math151 <- subset(placement,course == "M151")

math121 <- subset(placement,course == "M121")

smp_size <- floor(0.75*nrow(math151))
set.seed(123)
train_ind<-sample(seq_len(nrow(math151)),size=smp_size)

math151_train<-math151[train_ind,]
math151_test<-math151[-train_ind,]


#defining features
omitmath151_train =
subset(math151_train,select=c(hsgpacat,place2cat))

math151_classifier<- naiveBayes(omitmath151_train,
math151_train$success,laplace=0)
math151_test_pred<-predict(math151_classifier,
newdata=math151_test,type="raw")

#classify predicted values
classifysuccessBayes <- function(math151_test_pred){
results <-vector(mode="numeric",length=length(math151_test_pred)/2)
for (i in 2:length(math151_test_pred)/2){
  if (is.na(math151_test_pred[i,2])==FALSE &&
math151_test_pred[i,2]>=0.75){
    results[i] = 1
  }
  else if(is.na(math151_test_pred[i,2])==TRUE){
    results[i] = NA
  }
  else{
```

```
    results[i] = 0
  }
}
return (results)
}
math151_test_pred2 = classifysuccessBayes(math151_test_pred)

CrossTable(math151_test_pred2,math151_test$success, prop.chisq =
FALSE,prop.t=FALSE,dnn = c('predicted','actual'))

##MATH 121 Naive Bayes
smp_size <- floor(0.75*nrow(math121))
set.seed(123)
train_ind<-sample(seq_len(nrow(math121)),size=smp_size)

math121_train<-math121[train_ind,]
math121_test<-math121[-train_ind,]

hist(placement$newactc,xlab = "ACT composites")
hist(place1,xlab = "Placement 1 scores")

#binning act c
placement<-mutate(placement,actccat=case_when((placement$newactc <=
21 & placement$newactc >=0)~1, (placement$newactc  <= 27 &
placement$newactc >=22)~2, (placement$newactc  <= 36 &
placement$newactc >=28)~3))

#binning place 1
placement<-mutate(placement,place1cat=case_when((place1 <= 5 &
place1>=0)~1, (place1 <= 9 & place1>=6)~2, (place1 <= 15 &
place1>=10)~3))


#updating
math151 <- subset(placement,course == "M151")

math121 <- subset(placement,course == "M121")

#defining features
omitmath121_train =
subset(math121_train,select=c(hsgpacat,place1cat,actmcat))

math121_classifier<-
naiveBayes(omitmath121_train,math121_train$success,laplace=0)
math121_test_pred<-predict(math121_classifier,newdata=math121_test)

CrossTable(math121_test_pred,math121_test$success, prop.chisq =
FALSE,prop.t=FALSE,dnn = c('predicted','actual'))
```