An Ensemble Model to Categorize the Spending Habits of Students: Development and Evaluation

DA5030

Manyun Zou

Spring 2024

Contents

```r
# don't show warning text
knitr::opts_chunk$set(warning = FALSE, message = FALSE)

# Loading all the packages here
library(corrplot)
```

```
## corrplot 0.92 loaded
```

```r
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
library(tidyr)
library(gmodels)
library(ggplot2)
library(reshape2)
```

```
##
## Attaching package: 'reshape2'

## The following object is masked from 'package:tidyr':
##
##     smiths
```

```r
library(psych)
```

```
##
## Attaching package: 'psych'

## The following objects are masked from 'package:ggplot2':
##
##     %+%, alpha
```

```r
library(class)
library(caret)
```

```
## Loading required package: lattice
```

```r
library(rpart) # decision trees
library(C50) # decision trees
library(kernlab) # sum
```

```
##
## Attaching package: 'kernlab'

## The following object is masked from 'package:psych':
##
##     alpha

## The following object is masked from 'package:ggplot2':
##
##     alpha
```

```r
library(randomForest) # random forest
```

```
## randomForest 4.7-1.1

## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:psych':
##
##     outlier

## The following object is masked from 'package:ggplot2':
##
##     margin

## The following object is masked from 'package:dplyr':
##
##     combine
```

```r
library(irr) # to calculate kappa
```

```
## Loading required package: lpSolve
```

```r
library(pROC)
```

```
## Type 'citation("pROC")' for a citation.

##
## Attaching package: 'pROC'

## The following object is masked from 'package:gmodels':
##
##     ci
```

```
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```
library(mlbench)
```

1. Data Acquisition

```
sheet_id <- "1WnTuN4yD-ZM3cO56ISQGsCjA4tb8-7T4"
student <- read.csv(sprintf("https://docs.google.com/uc?id=%s&export=download", sheet_id))
```

The dataset I am using for my final project is the Student Spending Habits Dataset acquired from Kaggle (https://www.kaggle.com/datasets/sumanthnimmagadda/student-spending-dataset/data). This dataset records a total of 1,000 students' spending habits, including where they spent money, how they spent money, and their basic demographic traits.

The purpose of my project is to develop a model to predict their preferred payment method: credit/debit card, mobile payment app, and cash, based on their spending habits. This would be a classification task.

(Note that when I chose this dataset for my final project, it was just posted online, and not many people have tested it out. However, as I dug deeper into the model building, I realized it is a bad dataset, and so were many other people on Kaggle. I will explain more about my findings in the notebook.)

2. Data Exploration

```
# drop the id column
drops <- c("X")
student <- student[ , !(names(student) %in% drops)]

# read other columns
str(student)
```

```
## 'data.frame':    1000 obs. of  17 variables:
##  $ age                    : int  19 24 24 23 20 25 23 23 22 18 ...
##  $ gender                 : chr  "Non-binary" "Female" "Non-binary" "Female" ...
##  $ year_in_school         : chr  "Freshman" "Junior" "Junior" "Senior" ...
##  $ major                  : chr  "Psychology" "Economics" "Economics" "Computer Science" ...
##  $ monthly_income         : int  958 1006 734 617 810 523 1354 631 1402 1423 ...
##  $ financial_aid          : int  270 875 928 265 522 790 69 748 248 74 ...
##  $ tuition                : int  5939 4908 3051 4935 3887 3151 4973 3966 5638 3977 ...
##  $ housing                : int  709 557 666 652 825 413 812 571 599 626 ...
##  $ food                   : int  296 365 220 289 372 386 398 269 354 249 ...
##  $ transportation         : int  123 85 137 114 168 122 101 92 82 117 ...
##  $ books_supplies         : int  188 252 99 223 194 131 213 251 155 123 ...
##  $ entertainment          : int  41 74 130 99 48 73 21 37 123 51 ...
##  $ personal_care          : int  78 92 23 30 71 38 38 90 41 74 ...
##  $ technology             : int  134 226 239 163 88 234 157 152 162 243 ...
##  $ health_wellness        : int  127 129 112 105 71 108 117 56 172 34 ...
##  $ miscellaneous          : int  72 68 133 55 104 99 48 62 194 196 ...
##  $ preferred_payment_method: chr  "Credit/Debit Card" "Credit/Debit Card" "Cash" "Mobile Payment App
```

As shown above, this dataset is a mix of categorical and numerical features, as the former one describing students' demographic traits while the later one demonstrating the students' spent money on various categories.

```
# exploring categorical features
table(student$gender)
```

```
##
##     Female       Male Non-binary
##        323        356        321
```

```
table(student$year_in_school)
```

```
##
##  Freshman    Junior    Senior Sophomore
##       253       247       254       246
```

```
table(student$major)
```

```
##
##          Biology Computer Science        Economics      Engineering
##              228              192              204              192
##       Psychology
##              184
```

```
table(student$preferred_payment_method)
```

```
##
##               Cash  Credit/Debit Card Mobile Payment App
##                310                340                350
```

We can see that in this dataset, all the categorical features have 3 to 5 categories. We can apply one-hot encoding accordingly later. We can also notice that the distribution within all four categorical features is relatively even. No dominant classes have been spotted.

Note that in the target variable "preferred_payment_method", there is not a dominant class, as the share of "Cash" is 31%, that of "Credit/Debit Card" is 34%, and that of "Mobile Payment App" is 35%.

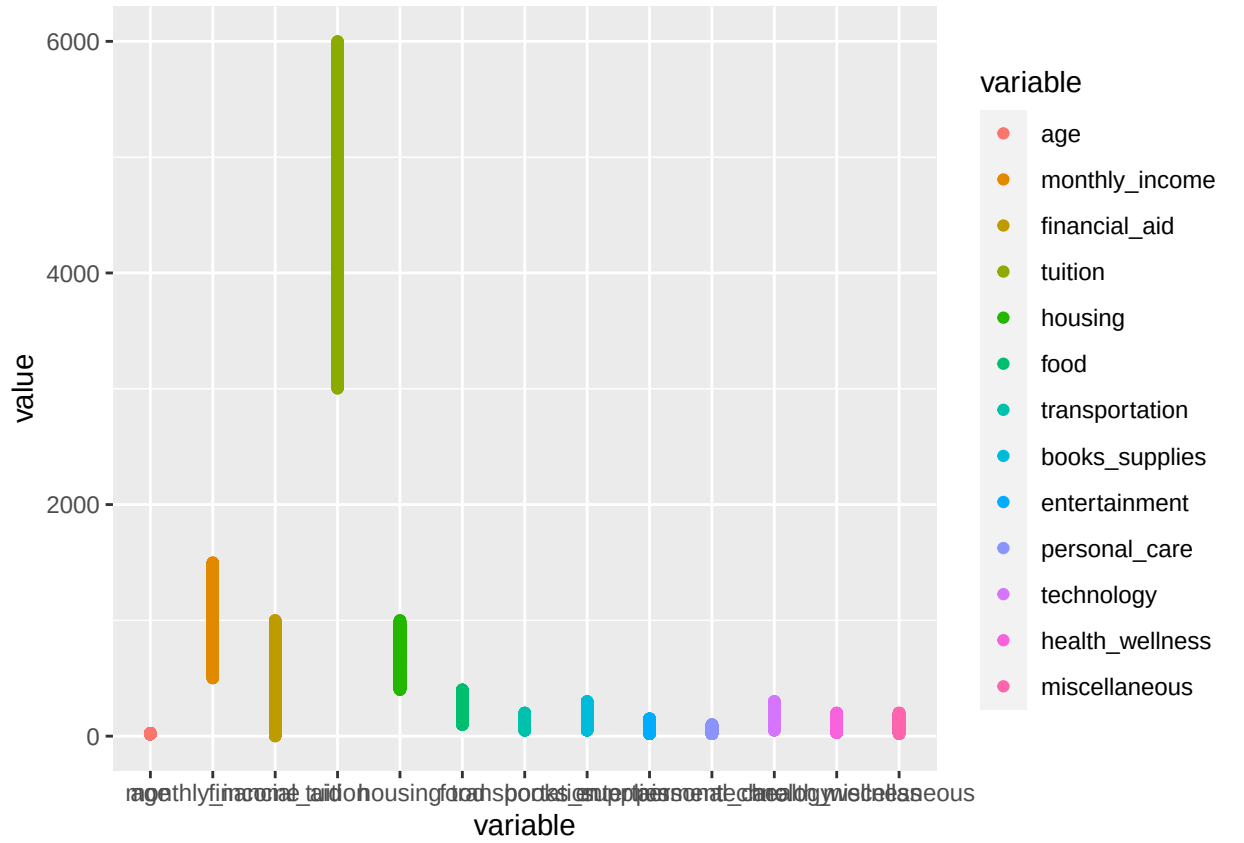
```
# exploring numerical features
summary(student[c("age", "monthly_income", "financial_aid", "tuition", "housing", "food", "transportati
```

```
##       age        monthly_income   financial_aid       tuition
##  Min.   :18.00   Min.   : 501.0   Min.   :   0.0   Min.   :3003
##  1st Qu.:20.00   1st Qu.: 770.8   1st Qu.: 261.0   1st Qu.:3780
##  Median :22.00   Median :1021.0   Median : 513.0   Median :4548
##  Mean   :21.68   Mean   :1020.6   Mean   : 504.8   Mean   :4520
##  3rd Qu.:24.00   3rd Qu.:1288.2   3rd Qu.: 751.5   3rd Qu.:5285
##  Max.   :25.00   Max.   :1500.0   Max.   :1000.0   Max.   :6000
##     housing            food       transportation  books_supplies
##  Min.   : 401.0   Min.   :100.0   Min.   : 50.0   Min.   : 50.0
##  1st Qu.: 538.8   1st Qu.:175.0   1st Qu.: 88.0   1st Qu.:112.0
```

```
##  Median : 704.5   Median :255.0   Median :123.0   Median :175.0
##  Mean   : 696.0   Mean   :252.6   Mean   :124.6   Mean   :174.8
##  3rd Qu.: 837.2   3rd Qu.:330.0   3rd Qu.:162.2   3rd Qu.:238.0
##  Max.   :1000.0   Max.   :400.0   Max.   :200.0   Max.   :300.0
##  entertainment    personal_care     technology    health_wellness
##  Min.   : 20.00   Min.   : 20.0   Min.   : 50.0   Min.   : 30.0
##  1st Qu.: 54.00   1st Qu.: 41.0   1st Qu.:114.0   1st Qu.: 73.0
##  Median : 86.00   Median : 62.0   Median :178.0   Median :115.0
##  Mean   : 84.81   Mean   : 60.7   Mean   :178.3   Mean   :114.3
##  3rd Qu.:116.00   3rd Qu.: 80.0   3rd Qu.:241.0   3rd Qu.:158.0
##  Max.   :150.00   Max.   :100.0   Max.   :300.0   Max.   :200.0
##  miscellaneous
##  Min.   : 20.00
##  1st Qu.: 63.75
##  Median :110.00
##  Mean   :108.91
##  3rd Qu.:153.00
##  Max.   :200.00
```

The numerical features of this dataset, on the other side, are in quite different ranges, which means that we need to do some normalization later. To have a better idea about how they distribute, I visualize them as following.

```
# Only keep the numerical features
categorical_features <- c("gender","year_in_school","major","preferred_payment_method")
# visualize the numerical features
student_numeric <- student[ , !(names(student) %in% categorical_features)]
ggplot(data = melt(student_numeric), aes(x=variable, y=value)) +  geom_point(aes(colour=variable))
```



As shown above, all the numerical features have different ranges, from 10x to 6000x. The fewest spending is on personal care (which defined as personal care items like hair care, lotion, etc.) while the highest spending is on tuition (not surprising). We will need to normalize them later.

```
# detect outliers using z-index
# z-index function
findOutliers <- function(x){
  m <- mean(x)
  sd <- sd(x)
  z <- abs((x - m) / sd)
  rows.outliers <- which(z > 2.5)
  return (rows.outliers)
}

# find rows with outliers
sapply(student[1], findOutliers)
```

```
## $age
## integer(0)
```

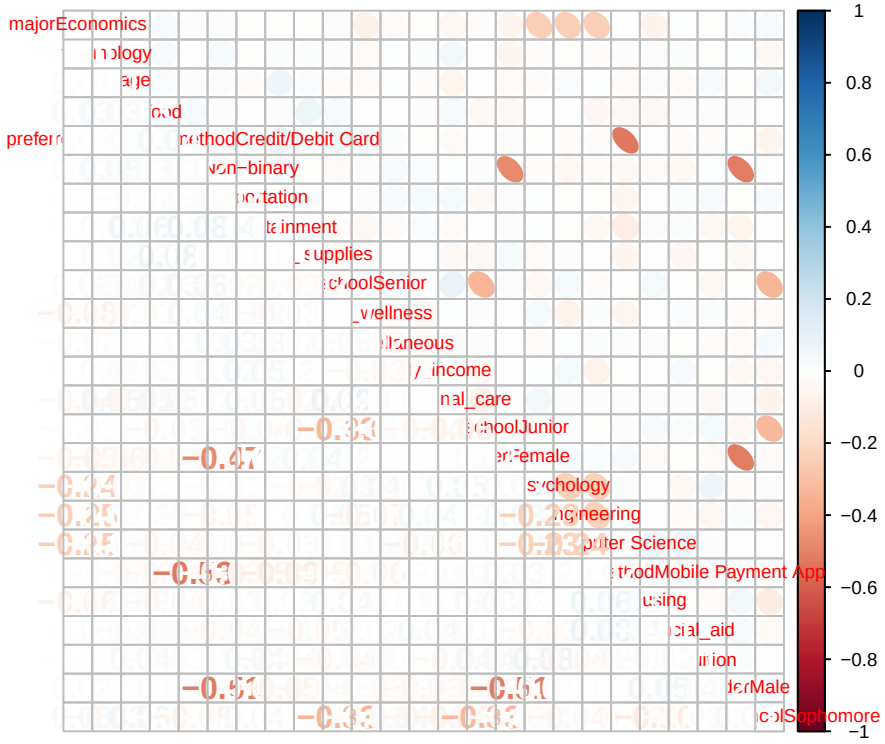
```
sapply(student[5:16], findOutliers)
```

```
## $monthly_income
## integer(0)
##
## $financial_aid
```

```
## integer(0)
##
## $tuition
## integer(0)
##
## $housing
## integer(0)
##
## $food
## integer(0)
##
## $transportation
## integer(0)
##
## $books_supplies
## integer(0)
##
## $entertainment
## integer(0)
##
## $personal_care
## integer(0)
##
## $technology
## integer(0)
##
## $health_wellness
## integer(0)
##
## $miscellaneous
## integer(0)
```

According to the Z-index, there is no outlier in the dataset.

```
# Using Correlation analysis to see if all types of spending are sort of related to each other
# one-hot encode all non-numeric features in the dataset
m <- model.matrix(~0+., data=student) %>%
  cor(use="pairwise.complete.obs")
# visualize the correlation, adjusting font size because there are too many to show
corrplot.mixed(m, order="AOE", upper="ellipse", lower="number", tl.cex=0.6, cl.cex=0.6)
```



The above chart is the correlation among all the features (with one-hot encoded categorical variables). As shown above, I don't see any strong correlation among predictors, which means that we don't need to eliminate any numerical variable due to a strong association. All the colored dots in the chart represent the correlation among one-hot encoded categorical variables, which is not a new information because of course the encoded categorical variables would influence each other.

However, in retrospect, this could be the first red flag about the dataset because there is no strong association between target variables and predictors either, but at that time, I thought I could do something to change it. The one-hot encoded target variables are represented by the 10th row in the chart above, "Credit/Debit Card," and by the 21st row, "Mobile Payment App."

Now, we will use chi-squared test to test the association among categorical variables.

```
# extract categorical features
categorical_features <- c("gender","year_in_school","major","preferred_payment_method")
student_categorical <- student[ , (names(student) %in% categorical_features)]

# chi-squared test between gender and else
lapply(student_categorical[,-1], function(x) chisq.test(student_categorical[,1], x))
```

```
## $year_in_school
##
##  Pearson's Chi-squared test
##
## data:  student_categorical[, 1] and x
## X-squared = 2.4488, df = 6, p-value = 0.8742
##
```

```
##
## $major
##
##  Pearson's Chi-squared test
##
## data:  student_categorical[, 1] and x
## X-squared = 7.3338, df = 8, p-value = 0.5011
##
##
## $preferred_payment_method
##
##  Pearson's Chi-squared test
##
## data:  student_categorical[, 1] and x
## X-squared = 1.4315, df = 4, p-value = 0.8387
```

I conducted a chi-squared test here to see if I would spot any strong correlation among categorical variables to eliminate some. The null hypothesis of the chi-squared test is that the two variables are independent of each other, and we can only reject the null hypothesis if $p < 0.05$. As shown above, since all the p-values are larger than 0.05, we can conclude that the "gender" variable is independent of all the categorical values.

```
# chi-squared test for year_in_school
lapply(student_categorical[,-2], function(x) chisq.test(student_categorical[,2], x))
```

```
## $gender
##
##  Pearson's Chi-squared test
##
## data:  student_categorical[, 2] and x
## X-squared = 2.4488, df = 6, p-value = 0.8742
##
##
## $major
##
##  Pearson's Chi-squared test
##
## data:  student_categorical[, 2] and x
## X-squared = 7.6281, df = 12, p-value = 0.8135
##
##
## $preferred_payment_method
##
##  Pearson's Chi-squared test
##
## data:  student_categorical[, 2] and x
## X-squared = 8.2527, df = 6, p-value = 0.2202
```

Since all the p-values are larger than 0.05, the "year_in_school" is also not associated with other categorical features.

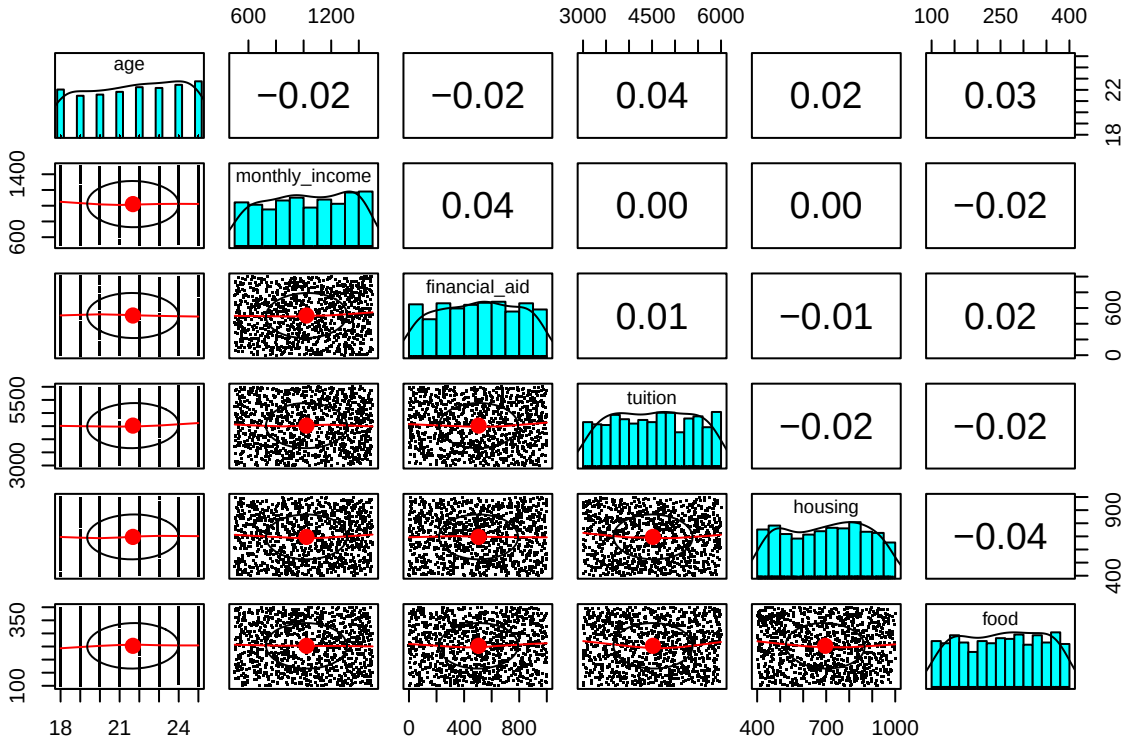
```
# chi-squared test for major
lapply(student_categorical[,-3], function(x) chisq.test(student_categorical[,3], x))
```

```
## $gender
##
##  Pearson's Chi-squared test
##
## data:  student_categorical[, 3] and x
## X-squared = 7.3338, df = 8, p-value = 0.5011
##
##
## $year_in_school
##
##  Pearson's Chi-squared test
##
## data:  student_categorical[, 3] and x
## X-squared = 7.6281, df = 12, p-value = 0.8135
##
##
## $preferred_payment_method
##
##  Pearson's Chi-squared test
##
## data:  student_categorical[, 3] and x
## X-squared = 2.352, df = 8, p-value = 0.9683
```

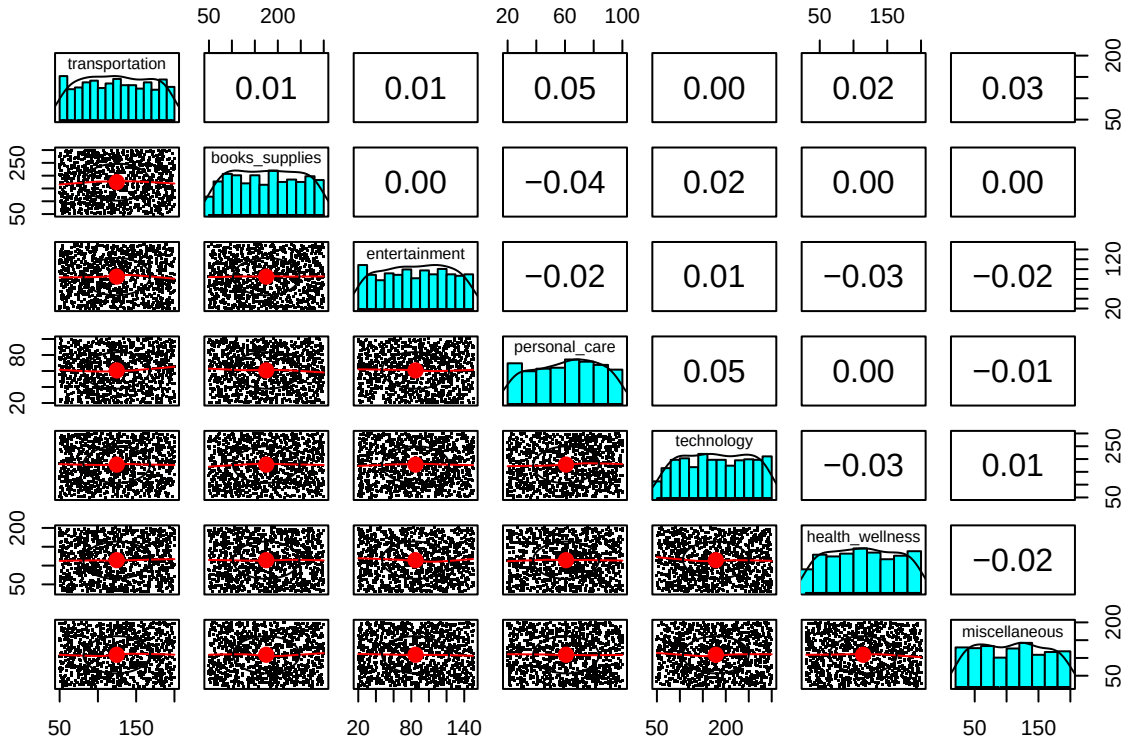
The "major" feature is not associated with other categorical features either.

Next, I want to evaluate the distribution of the numerical features.

```
pairs.panels(student[c("age", "monthly_income", "financial_aid", "tuition", "housing", "food")], pch =
```




```
pairs.panels(student[c("transportation", "books_supplies", "entertainment", "personal_care", "technology
```



As shown above, all the numeric variables are relatively evenly distributed, not skewed nor normally distributed. Since my choice of algorithms (kNN, Rule Learners, and Decision Trees) don't have assumptions about data distribution, it doesn't matter if the data is normally distributed or not.

Again, in retrospect, this could be another red flag of the dataset, as the data is so perfectly evenly distributed. When I went back to check the Kaggle page, I noticed that the creators added a sentence in the description - "...and all data is entirely fictional." I don't think the description was there when I decided to pick this dataset but it explained why the distribution is so even.

3. Data Cleaning & Shaping

To avoid data leaks, I decided to split train and test sets before any data cleaning. The ratio of my train and test sets is 7:3. The kNN algorithm will have one version of the sets since it requires data normalization while the Rule Learners and Decision Trees will share one version.

```
# split into train and test sets
set.seed(111)

# set samples range
samples <- sample.int(n = nrow(student), size = floor(.7*nrow(student)), replace = F)

# for k-NN
# train set
studentTrain <- student[samples, 1:16]
# validation test
studentTest <- student[-samples, 1:16]
```

```
# save target variable separately
studentTrain.labels <- student[samples, 17]
studentTest.labels <- student[-samples, 17]

# for Decision Trees
studentTrain_tree <- student[samples, ]
studentTest_tree <- student[-samples, ]
```

According to the outlier detection in the last section, we don't have any outliers in this dataset, and hence, I will skip the outlier imputation. If there are only a few outliers, I might remove rows with the outliers. If there are a lot of outliers, I might imputate outliers with the median of each feature (after splitting into train and test data).

Next, I want to see if there is any missing values in the dataset.

```
# identify missing values
any(is.na(studentTrain))
```

```
## [1] FALSE
```

```
any(is.na(studentTest))
```

```
## [1] FALSE
```

As shown above, there is no missing data. But if missing data exists, I would impute the missing data by replacing it with median. Here is the code for demonstration:

```
# impute missing data in training set
studentTrain <- studentTrain %>%  mutate(across(where(is.numeric), ~replace_na(., median(., na.rm=TRUE)))

# impute missing data in test set
studentTest <- studentTest %>%  mutate(across(where(is.numeric), ~replace_na(., median(., na.rm=TRUE))))
```

Then I will normalize the numerical features for the k-NN algorithm model using Z-score standardization.

```
set.seed(111)

# create Z-score Standardization
z_standardization <- function(data) {
  standardized.df <- data.frame(matrix(ncol = ncol(data), nrow=nrow(data)))
  names(standardized.df) <- names(data)

  for (i in seq_along(data)){
      column_data <- data[[i]]
      mean_value <- mean(column_data, na.rm=TRUE)
      sd_value <- sd(column_data, na.rm=TRUE)
      standardized.df[[i]] <- (column_data - mean_value) / sd_value
  }

  return (standardized.df)
}
```

```r
# normalize the numerical features in train set
studentTrain_numeric <- studentTrain[, !(colnames(studentTrain) %in% c("gender","year_in_school","major"
# create a new standardization dataset
studentTrain.standardized <- z_standardization(studentTrain_numeric)

# add categorical features to the new dataset
studentTrain.standardized$gender <- studentTrain$gender
studentTrain.standardized$year_in_school <- studentTrain$year_in_school
studentTrain.standardized$major <- studentTrain$major

# repeat the process for numerical features in test set
studentTest_numeric <- studentTest[, !(colnames(studentTest) %in% c("gender","year_in_school","major"))]
studentTest.standardized <-  z_standardization(studentTest_numeric)
studentTest.standardized$gender <- studentTest$gender
studentTest.standardized$year_in_school <- studentTest$year_in_school
studentTest.standardized$major <- studentTest$major
```

Now I will encode categorical feature for kNN algorithm. For ordinal variables, I will simply assign numbers, while for nominal variables, I will apply one-hot encoding.

```r
# train set
# "year_in_school": since this variable is ordinal, I will simply assign numbers to it
studentTrain.standardized$year_in_school[studentTrain.standardized$year_in_school == "Freshman"] <- 1
studentTrain.standardized$year_in_school[studentTrain.standardized$year_in_school == "Sophomore"] <- 2
studentTrain.standardized$year_in_school[studentTrain.standardized$year_in_school == "Junior"] <- 3
studentTrain.standardized$year_in_school[studentTrain.standardized$year_in_school == "Senior"] <- 4
# "gender" and "major": since these two variables are nominal, I will use one-hot encoding
studentTrain.standardized_oneHot <- studentTrain.standardized %>%
  model.matrix(~ gender + major, data = .) %>%
  as.data.frame()
studentTrain.standardized <- cbind(studentTrain.standardized, studentTrain.standardized_oneHot[2:7]) %>%
  select(-c("gender","major"))

# test set
studentTest.standardized$year_in_school[studentTest.standardized$year_in_school == "Freshman"] <- 1
studentTest.standardized$year_in_school[studentTest.standardized$year_in_school == "Sophomore"] <- 2
studentTest.standardized$year_in_school[studentTest.standardized$year_in_school == "Junior"] <- 3
studentTest.standardized$year_in_school[studentTest.standardized$year_in_school == "Senior"] <- 4

studentTest.standardized_oneHot <- studentTest.standardized %>%
  model.matrix(~ gender + major, data = .) %>%
  as.data.frame()
studentTest.standardized <- cbind(studentTest.standardized, studentTest.standardized_oneHot[2:7]) %>%
  select(-c("gender","major"))
```

```r
# for SVM
# train data
studentTrain_svm <- studentTrain.standardized
studentTrain_svm$preferred_payment_method <- as.factor(studentTrain.labels)

# test data
studentTest_svm <- studentTest.standardized
studentTest_svm$preferred_payment_method <- as.factor(studentTest.labels)
```

Based on the correlation analysis and chi-squared test before, I don't think I need to eliminate any features. Hence, I decide to construct models first and go through the feature selections after the evaluation of models.

4. k-NN

I chose k-NN as one of my models for three reasons: (1) I am aiming to build a classifier; (2) the majority of my predictors are numerical features; (3) not ton of predictors in the dataset so I don't need to worry about slow classification phase.

4.1 Model Construction

```
set.seed(111)
student_knn_pred <- knn(train=studentTrain.standardized, test=studentTest.standardized, cl=studentTrain
```

I chose the k = 26 here because there are 700 instances in the training set, and by default, the k value should roughly equal to the square root of the instance number. I will test out multiple k values in the model evaluation.

4.2 Model Evaluation

```
confusionMatrix(factor(student_knn_pred, levels=c("Cash", "Credit/Debit Card", "Mobile Payment App")), r
```

```
## Confusion Matrix and Statistics
##
##                     Reference
## Prediction           Cash Credit/Debit Card Mobile Payment App
##   Cash                 35                39                 33
##   Credit/Debit Card    32                29                 46
##   Mobile Payment App   27                29                 30
##
## Overall Statistics
##
##                Accuracy : 0.3133
##                  95% CI : (0.2613, 0.3691)
##     No Information Rate : 0.3633
##     P-Value [Acc > NIR] : 0.9697
##
##                   Kappa : -0.0268
##
##  Mcnemar's Test P-Value : 0.1616
##
## Statistics by Class:
##
##                      Class: Cash Class: Credit/Debit Card
## Sensitivity               0.3723                  0.29897
## Specificity               0.6505                  0.61576
## Pos Pred Value            0.3271                  0.27103
## Neg Pred Value            0.6943                  0.64767
```

```
## Prevalence                0.3133                  0.32333
## Detection Rate            0.1167                  0.09667
## Detection Prevalence      0.3567                  0.35667
## Balanced Accuracy         0.5114                  0.45737
##                       Class: Mobile Payment App
## Sensitivity                              0.2752
## Specificity                              0.7068
## Pos Pred Value                           0.3488
## Neg Pred Value                           0.6308
## Prevalence                               0.3633
## Detection Rate                           0.1000
## Detection Prevalence                     0.2867
## Balanced Accuracy                        0.4910
```

As shown above, the overall accuracy of the model is (35+29+30)/300 = 31.3%. Not an ideal advocacy rate. Looking closer, about 37% of Cash being predicted correctly, about 30% of Credit/Debit Card being predicted correctly, and about 28% of Mobile Payment App being predicted correctly. Not one single class has a significant high accuracy rate, which means that no data imbalance leads to the prediction failure.

The negative Kappa score, however, indicated that there is no agreement between the reference and the predicted values. We need some model tuning to improve the metrics.

4.3 Model Tuning & Performance Improvement

The hyperparameter of kNN model is the "k" value. Hence, we can test out different "k".

```
# hyperparameters tuning
k_values <- c(25,26,27,28, 29, 30,31,32,33,34,35)
k_values_this <- c()
knn_accuracy_list <- c()

for (k_val in k_values){
  set.seed(111)
  student_knn_pred <- knn(train=studentTrain.standardized,
                          test=studentTest.standardized,
                          cl=studentTrain.labels,
                          k=k_val)
  j <- confusionMatrix(factor(student_knn_pred, levels=c("Cash", "Credit/Debit Card", "Mobile Payment Ap
  k_values_this<- c(k_val, k_values_this)
  knn_accuracy_list <- c(j$overall[['Accuracy']], knn_accuracy_list)
  #print(j)

}

knn_improved_accuracy <- data.frame(
  k_values = k_values_this,
  accuracy_rate = knn_accuracy_list,
  stringsAsFactors = FALSE
)

knn_improved_accuracy
```

```
##    k_values accuracy_rate
```

```
## 1        35     0.3000000
## 2        34     0.3000000
## 3        33     0.3233333
## 4        32     0.3033333
## 5        31     0.3133333
## 6        30     0.3066667
## 7        29     0.3266667
## 8        28     0.2966667
## 9        27     0.3066667
## 10       26     0.3133333
## 11       25     0.3066667
```

As shown above, I tested out 11 different k values that ranged from 25 to 35. The accuracy rate reaches the highest of 31.3% when k= 31. Unfortunately, it is still not an ideal rate. Maybe kNN is not an ideal algorithm for this dataset, so I will try out the other two algorithms.

5. Decision Trees

Decision Trees is a good choice for this dataset because it can handle both categorical and numerical data, and it can return a readable model. Also, the Decision Trees don't require normalized data so I can see if that can help the classifier perform better.

5.1 Model Construction

```
studentTrain_tree$preferred_payment_method <- as.factor(studentTrain_tree$preferred_payment_method)

# Build the decision tree
myTree <- C5.0(preferred_payment_method ~ ., data = studentTrain_tree, trials=1)
myTree
```

```
##
## Call:
## C5.0.formula(formula = preferred_payment_method ~ ., data =
##  studentTrain_tree, trials = 1)
##
## Classification Tree
## Number of samples: 700
## Number of predictors: 16
##
## Tree size: 160
##
## Non-standard options: attempt to group attributes
```

Here we can see that C5.0 developed a decision tree with 160 branches.

```
# Predict with the tree model
mytree_predict <- predict(myTree, newdata = studentTest_tree, type="class")
```

5.2 Model Evaluation

5.2.1 Confusion Matrix

```
confusionMatrix(factor(mytree_predict, levels=c("Cash", "Credit/Debit Card", "Mobile Payment App")), fa
```

```
## Confusion Matrix and Statistics
##
##                     Reference
## Prediction           Cash Credit/Debit Card Mobile Payment App
##   Cash                 28                22                 33
##   Credit/Debit Card    35                42                 45
##   Mobile Payment App   31                33                 31
##
## Overall Statistics
##
##                Accuracy : 0.3367
##                  95% CI : (0.2834, 0.3932)
##     No Information Rate : 0.3633
##     P-Value [Acc > NIR] : 0.8463
##
##                   Kappa : 0.0051
##
##  Mcnemar's Test P-Value : 0.1813
##
## Statistics by Class:
##
##                      Class: Cash Class: Credit/Debit Card
## Sensitivity              0.29787                   0.4330
## Specificity              0.73301                   0.6059
## Pos Pred Value           0.33735                   0.3443
## Neg Pred Value           0.69585                   0.6910
## Prevalence               0.31333                   0.3233
## Detection Rate           0.09333                   0.1400
## Detection Prevalence     0.27667                   0.4067
## Balanced Accuracy        0.51544                   0.5195
##                      Class: Mobile Payment App
## Sensitivity                             0.2844
## Specificity                             0.6649
## Pos Pred Value                          0.3263
## Neg Pred Value                          0.6195
## Prevalence                              0.3633
## Detection Rate                          0.1033
## Detection Prevalence                    0.3167
## Balanced Accuracy                       0.4747
```

As shown above, the Decision Trees only successfully predicted $(28+42+31)/300 = 33.67\%$ of target variables. Again, not an ideal accuracy rate. According to the "statistics by class" table, we can see that this classifier doesn't prefer any class, as the positive pred value for each class is around 0.33 while the negative pred value for each class is around 0.69. In retrospect, this would have been a strong red flag to the quality of the dataset.

Looking closer, the accuracy rate for each class is 0.2979 for "Cash", 0.4330 for "Credit/Debit Card", and 0.2844 for "Mobile Payment App." The "Credit/Debit Card" class performs a little bit better than the other

two classes, but still not ideal. Since the class distribution of the target variable is 94:97:109, there is no class being dominatedly presented.

The good news is that the Kappa value finally become a positive number but only demonstrates very weak agreement between predicted values and reference ones. To evaluate the model a little bit more, I will apply a k-fold cross-validation.

5.2.2 K-fold cross-validation

```
set.seed(111)
folds <- createFolds(student$preferred_payment_method, k=10)

cv_results <- lapply(folds, function(x){
  set.seed(111)
  student_tree_cv_train <- student[ -x, ]
  student_tree_cv_test <- student[x, ]
  student_tree_cv_train$preferred_payment_method <- as.factor(student_tree_cv_train$preferred_payment_m
  student_tree_cv_test$preferred_payment_method <- as.factor(student_tree_cv_test$preferred_payment_met
  student_tree_cv_model <- C5.0(preferred_payment_method ~., data=student_tree_cv_train)
  student_tree_cv_pred <- predict(student_tree_cv_model, student_tree_cv_test)
  student_tree_cv_actual <- student_tree_cv_test$preferred_payment_method
  kappa <- kappa2(data.frame(student_tree_cv_actual, student_tree_cv_pred))$value
  return(kappa)
})

str(cv_results)
```

```
## List of 10
##  $ Fold01: num -0.0971
##  $ Fold02: num 0.0438
##  $ Fold03: num -0.0838
##  $ Fold04: num -0.0951
##  $ Fold05: num 0.0518
##  $ Fold06: num 0.0577
##  $ Fold07: num -0.0918
##  $ Fold08: num 0.0682
##  $ Fold09: num -0.0319
##  $ Fold10: num -0.0517
```

The Kappa values of the 10-fold-CV process shows that this classifier performed poorly, as only 4 out of 10 folds resulted in a positive Kappa value, and all of them are less than 0.1. This classifier results in a very weak agreement between predictors and reference values.

```
# the average of all kappas
mean(unlist(cv_results))
```

```
## [1] -0.02298955
```

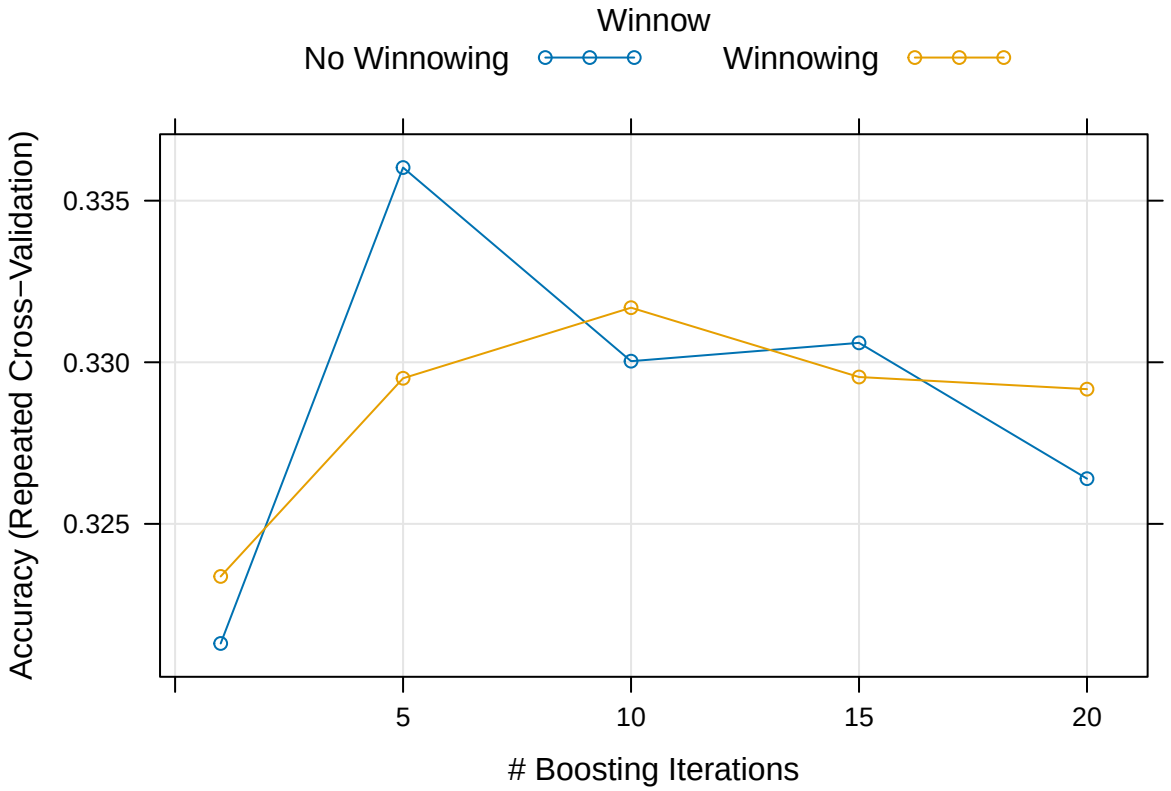
And of course, the mean Kappa value is negative, which means disagreement between predictors and reference values.

5.3 Model Tuning & Performance Improvement

5.3.1 Hyperparameter

The Decision Trees have three hyperparameters: model, trials, and winnow. Here I will make trials and winnow part of the control object to tune the model.

```
# reference: https://www.euclidean.com/machine-learning-in-practice/2015/6/12/r-caret-and-parameter-tun
set.seed(111)

fitControl <- trainControl(method="repeatedcv",
                           number = 10,
                           repeats = 10,
                           returnResamp="all")

grid <- expand.grid( .winnow = c(TRUE,FALSE), .trials=c(1,5,10,15,20), .model="tree" )
mdl <- train(x=studentTrain_tree[c("age","gender","year_in_school","major","monthly_income","financial_

plot(mdl)
```



As shown above, the classifier performs the best when winnow sets to "False" and iterations to "5". However, even the best-performed classifier can only score less than an overall accuracy of 0.34... (In retrospect, this is definitely a red flag and should be when I question the quality of the dataset.)

5.3.2 Mistake Cost

I also want to boost the accuracy by adding mistake cost to the model. According to the confusion matrix in section 5.2.1, about 37% of Cash is classified as "Credit/Debit Card" while 33% is classified as "Mobile Payment App"; 23% of "Credit/Debit Card" is classified as "Cash" while 34% is classified as "Mobile Payment App"; 30% of "Mobile Payment App" is classified as "Cash" while 41% is classified as "Credit/Debit Card". Accordingly, I plan to add more mistake costs to the wrong classifications with higher shares with a little bit adjustment.

```
matrix_dimensions <- list(c("Cash", "Credit/Debit Card", "Mobile Payment App"), c("Cash", "Credit/Debit
names(matrix_dimensions) <- c("predicted", "actual")
error_cost <- matrix(c(0, 2, 4, 2, 0, 8, 2, 10, 0), nrow=3, dimnames=matrix_dimensions)
error_cost
```

```
##                      actual
## predicted             Cash Credit/Debit Card Mobile Payment App
##   Cash                   0                 2                  2
##   Credit/Debit Card      2                 0                 10
##   Mobile Payment App     4                 8                  0
```

```
set.seed(111)
# build new model with mistake cost
mytree_cost <- C5.0(preferred_payment_method ~ ., data = studentTrain_tree, costs=error_cost, trials=1)
mytree_cost_pred <- predict(mytree_cost, newdata = studentTest_tree)

confusionMatrix(factor(mytree_cost_pred, levels=c("Cash", "Credit/Debit Card", "Mobile Payment App")), r
```

```
## Confusion Matrix and Statistics
##
##                      Reference
## Prediction            Cash Credit/Debit Card Mobile Payment App
##   Cash                  33                28                 33
##   Credit/Debit Card     35                42                 45
##   Mobile Payment App    26                27                 31
##
## Overall Statistics
##
##                Accuracy : 0.3533
##                  95% CI : (0.2993, 0.4103)
##     No Information Rate : 0.3633
##     P-Value [Acc > NIR] : 0.6611
##
##                   Kappa : 0.0328
##
##  Mcnemar's Test P-Value : 0.1065
##
## Statistics by Class:
##
##                      Class: Cash Class: Credit/Debit Card
## Sensitivity               0.3511                   0.4330
## Specificity               0.7039                   0.6059
## Pos Pred Value            0.3511                   0.3443
## Neg Pred Value            0.7039                   0.6910
```

```
## Prevalence                0.3133                 0.3233
## Detection Rate            0.1100                 0.1400
## Detection Prevalence      0.3133                 0.4067
## Balanced Accuracy         0.5275                 0.5195
##                       Class: Mobile Payment App
## Sensitivity                           0.2844
## Specificity                           0.7225
## Pos Pred Value                        0.3690
## Neg Pred Value                        0.6389
## Prevalence                            0.3633
## Detection Rate                        0.1033
## Detection Prevalence                  0.2800
## Balanced Accuracy                     0.5035
```

As shown above, after adding the mistake cost, the overall accuracy rate is boosted to 35.3%, still a low rate but slighly higher than the optimized hyperparameters tuning result. The original overall accuracy rate was 33.7%. Looking closer, the accuracy rate of each class doesn't increase much- the accuracy rate of predicting "Cash" is 35.1%, up from 29.8%; that of "Credit/Debit Card" is 43.3%, the same as before; that of "Mobile Payment App" is 28.4%, the same as before too. The Kappa score, again, is lower than 0.1.

Overall, the model tuning of Decision Trees doesn't improve its performance much.

6. SVM

I chose Support Vector Machines because it can serve as a classifier and can take dummy-coded categorical variables, as long as numerical variables. Also, the dataset is not too big for SVM to handle, as it is usually computationally demanding.

6.1 Model Construction

```
# train the model
svm_classifier <- ksvm(preferred_payment_method ~., data=studentTrain_svm, kernel="vanilladot")
```

```
##  Setting default kernel parameters
```

```
# predict with the model
svm_predictions <- predict(svm_classifier, studentTest_svm)
```

6.2 Model Evaluation

```
confusionMatrix(factor(svm_predictions, levels=c("Cash", "Credit/Debit Card", "Mobile Payment App")),
                factor(studentTest_svm$preferred_payment_method, levels=c("Cash", "Credit/Debit Card",
```

```
## Confusion Matrix and Statistics
##
##                     Reference
## Prediction           Cash Credit/Debit Card Mobile Payment App
```

```
##   Cash                 19                17                 19
##   Credit/Debit Card    45                42                 51
##   Mobile Payment App   30                38                 39
##
## Overall Statistics
##
##                Accuracy : 0.3333
##                  95% CI : (0.2802, 0.3898)
##     No Information Rate : 0.3633
##     P-Value [Acc > NIR] : 0.8733163
##
##                   Kappa : -0.0037
##
##  Mcnemar's Test P-Value : 0.0007023
##
## Statistics by Class:
##
##                      Class: Cash Class: Credit/Debit Card
## Sensitivity              0.20213                   0.4330
## Specificity              0.82524                   0.5271
## Pos Pred Value           0.34545                   0.3043
## Neg Pred Value           0.69388                   0.6605
## Prevalence               0.31333                   0.3233
## Detection Rate           0.06333                   0.1400
## Detection Prevalence     0.18333                   0.4600
## Balanced Accuracy        0.51369                   0.4800
##                      Class: Mobile Payment App
## Sensitivity                             0.3578
## Specificity                             0.6440
## Pos Pred Value                          0.3645
## Neg Pred Value                          0.6373
## Prevalence                              0.3633
## Detection Rate                          0.1300
## Detection Prevalence                    0.3567
## Balanced Accuracy                       0.5009
```

As shown above, again, the overall accuracy rate of this model is (19+42+39)/300 = 33.3%. The accuracy rate for each class is also about the same as the other two models, with "Credit/Debit Card" scoring the highest at about 43%. The "Cash" scored the lowest accuracy rate with 20% while the "Mobile Payment App" got about 36% right. The gap between each class' accuracy rate is a little bigger than before, but still, no dominant class has been spotted.

6.3 Model Tuning & Performance Improvement

6.3.1 Kernel

There are two ways for me to improve the SVM model performance. First, we can change the kernel from linear to radial basis.

```
# train the model
svm_classifier_rbf <- ksvm(preferred_payment_method ~., data=studentTrain_svm, kernel="rbfdot")
# predict with the model
svm_predictions_rbf <- predict(svm_classifier_rbf, studentTest_svm)
```

```r
# calculate agreement
agreement_rbf <- svm_predictions_rbf == studentTest_svm$preferred_payment_method
table(agreement_rbf)
```

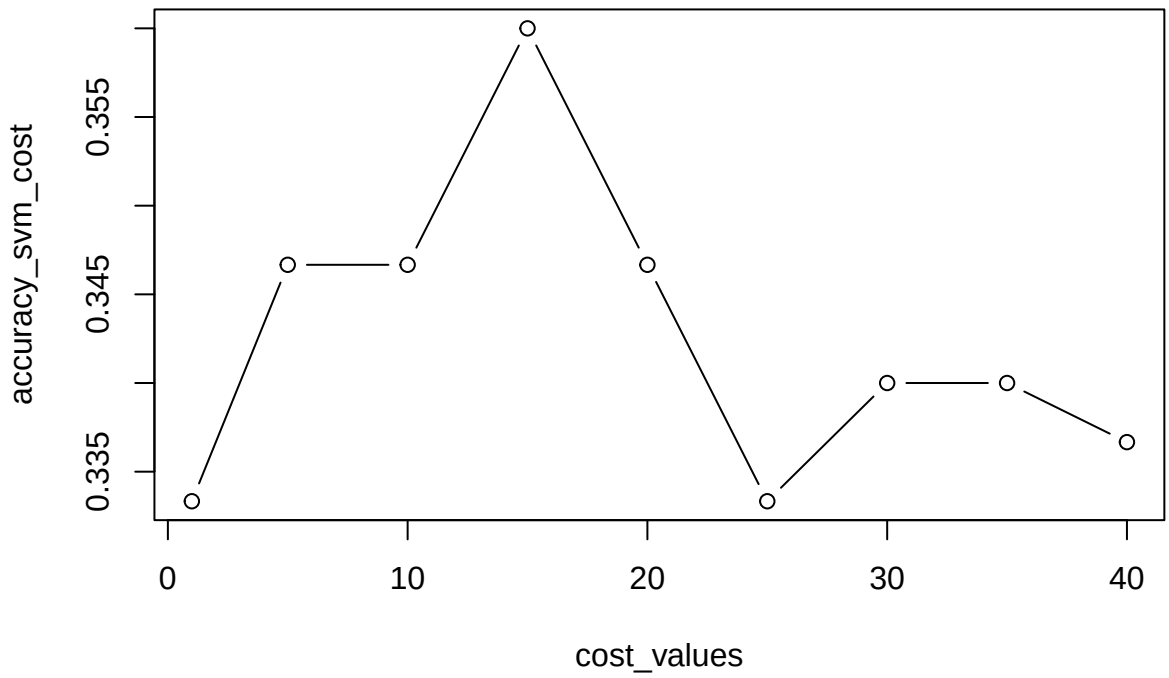
```
## agreement_rbf
## FALSE  TRUE
##   200   100
```

After we changed the kernerl to radial basis function, the overall accracy rate is still 100/300 = 33.3%. Nothing has been changed!! It feels like talking to a wall :(

6.3.2 Cost Parameter

But anyways, I can still try adjusting SVM cost parameter to see if I could improve the model performance.

```r
cost_values <- c(1, seq(from=5, to=40, by=5))
accuracy_svm_cost <- sapply(cost_values, function(x){
  set.seed(111)
  m <- ksvm(preferred_payment_method ~., data=studentTrain_svm, kernel="rbfdot", C=x)
  pred <- predict(m, studentTest_svm)
  agree <- table(pred == studentTest_svm$preferred_payment_method)
  accuracy <- agree[['TRUE']]/nrow(studentTest_svm)
  return(accuracy)
})

plot(cost_values, accuracy_svm_cost, type="b")
```



As shown above, after testing out different cost parameters, we can see that the overall accuracy rate reaches the highest when the cost value set to 15. But even the highest accuracy rate is still only around 36%, not an ideal rate, again.

7. Ensemble

7.1 Homogeneous ensemble

I will be using random forest as my homogeneous ensemble method, as it is also a tree-based method and built upon the principles of bagging. Also, all the categorical variables in the dataset don't have massive levels, so it won't impede the effectiveness of the random forest.

```
set.seed(111)
# construct random forests model
payment_rf <- randomForest(preferred_payment_method ~., data=studentTrain_tree, ntree=100)
payment_rf_predict <- predict(payment_rf, newdata=studentTest_tree)

# evaluate the model
confusionMatrix(factor(payment_rf_predict, levels=c("Cash", "Credit/Debit Card", "Mobile Payment App"))
```

```
## Confusion Matrix and Statistics
##
##                     Reference
## Prediction           Cash Credit/Debit Card Mobile Payment App
```

```
##   Cash                 21                23                27
##   Credit/Debit Card    42                38                42
##   Mobile Payment App   31                36                40
##
## Overall Statistics
##
##                Accuracy : 0.33
##                  95% CI : (0.277, 0.3864)
##     No Information Rate : 0.3633
##     P-Value [Acc > NIR] : 0.89688
##
##                   Kappa : -0.0079
##
##  Mcnemar's Test P-Value : 0.09827
##
## Statistics by Class:
##
##                      Class: Cash Class: Credit/Debit Card
## Sensitivity               0.2234                   0.3918
## Specificity               0.7573                   0.5862
## Pos Pred Value            0.2958                   0.3115
## Neg Pred Value            0.6812                   0.6685
## Prevalence                0.3133                   0.3233
## Detection Rate            0.0700                   0.1267
## Detection Prevalence      0.2367                   0.4067
## Balanced Accuracy         0.4903                   0.4890
##                      Class: Mobile Payment App
## Sensitivity                             0.3670
## Specificity                             0.6492
## Pos Pred Value                          0.3738
## Neg Pred Value                          0.6425
## Prevalence                              0.3633
## Detection Rate                          0.1333
## Detection Prevalence                    0.3567
## Balanced Accuracy                       0.5081
```

Even with the help of random forests, the overall accuracy rate still stagnates around 33%! The Kappa value is negative as well, emphasizing the disagreement between prediction and reference values. And as shown below, even within the train data, the error rate reaches almost 66%.

```
payment_rf
```

```
##
## Call:
##  randomForest(formula = preferred_payment_method ~ ., data = studentTrain_tree,      ntree = 100)
##                Type of random forest: classification
##                      Number of trees: 100
## No. of variables tried at each split: 4
##
##         OOB estimate of  error rate: 65.71%
## Confusion matrix:
##                    Cash Credit/Debit Card Mobile Payment App class.error
## Cash                 60                80                 76   0.7222222
```

```
## Credit/Debit Card    68                90                85   0.6296296
## Mobile Payment App   63                88                90   0.6265560
```

7.1.1 Model Tuning

I wonder if changing the number of trees, in other words, changing the model complexity, can make any difference to the result.

```
tree_values <- c(50,60,70,80,90,100,110,120,130,140,150)
tree_this <- c()
accuracy_rf_list <- c()

for (tree_val in tree_values){
  set.seed(111)
  payment_rf <- randomForest(preferred_payment_method ~., data=studentTrain_tree, ntree=tree_val)
  payment_rf_predict <- predict(payment_rf, newdata=studentTest_tree)

  cm <- confusionMatrix(factor(payment_rf_predict, levels=c("Cash", "Credit/Debit Card", "Mobile Paymen
  accuracy_val <- cm$overall[['Accuracy']]
  accuracy_rf_list <- c( accuracy_val, accuracy_rf_list)
  tree_this <- c(tree_val, tree_this)
}

rf_improved_accuracy <- data.frame(
  k_values = tree_this,
  accuracy_rate = accuracy_rf_list,
  stringsAsFactors = FALSE
)

rf_improved_accuracy
```

```
##    k_values accuracy_rate
## 1       150     0.3333333
## 2       140     0.3200000
## 3       130     0.3133333
## 4       120     0.3066667
## 5       110     0.3400000
## 6       100     0.3300000
## 7        90     0.3033333
## 8        80     0.3100000
## 9        70     0.3066667
## 10       60     0.3266667
## 11       50     0.3266667
```

However, the model complexity adjustment doesn't seem to help the overall accuracy rate much. As shown above, I tested out various tree numbers that ranged from 50 to 150, but the accuracy rate is still below 35%.

7.2 Heterogeneous ensemble

To set a heterogeneous ensemble model, I plan to build a function that can compare the prediction of three models (kNN, Decision Trees, and SVM) and then take the most frequent output as the answer. I am curious if this would boost the accuracy rate a little bit.

```r
set.seed(111)

# Model construction
payment_ensemble <- function(h, i, j){
  # knn
  student_knn_pred <- knn(train=studentTrain.standardized,
                          test=h,
                          cl=studentTrain.labels,
                          k=29)
  # Decision Trees
  mytree_cost <- C5.0(preferred_payment_method ~ ., data = studentTrain_tree)
  mytree_predict <- predict(myTree, newdata = i, type="class")

  # SVM
  svm_classifier <- ksvm(preferred_payment_method ~., data=studentTrain_svm, kernel="rbfdot")
  svm_predict<- predict(svm_classifier, j)

  # merge all the prediction into a dataframe
  predictOutcome <- data.frame(
                            model1 = student_knn_pred,
                            model2 = mytree_predict,
                            model3 = svm_predict)

  # select the most frequent one from each row
  finalPredictOutcome <- apply(predictOutcome, 1, function(x) names(which.max(table(x))))

  return (finalPredictOutcome)
}
```

```r
# Model prediction
payment_ensemble_predict <- payment_ensemble(studentTest.standardized, studentTest_tree, studentTest_svm
# evaluate the model
confusionMatrix(factor(payment_ensemble_predict, levels=c("Cash", "Credit/Debit Card", "Mobile Payment A
```

```
## Confusion Matrix and Statistics
##
##                     Reference
## Prediction           Cash Credit/Debit Card Mobile Payment App
##   Cash                 35                28                 31
##   Credit/Debit Card    34                39                 45
##   Mobile Payment App   25                30                 33
##
## Overall Statistics
##
##                Accuracy : 0.3567
##                  95% CI : (0.3025, 0.4137)
##     No Information Rate : 0.3633
##     P-Value [Acc > NIR] : 0.6160
##
##                   Kappa : 0.037
##
##  Mcnemar's Test P-Value : 0.2383
##
```

```
## Statistics by Class:
##
##                      Class: Cash Class: Credit/Debit Card
## Sensitivity               0.3723                    0.4021
## Specificity               0.7136                    0.6108
## Pos Pred Value            0.3723                    0.3305
## Neg Pred Value            0.7136                    0.6813
## Prevalence                0.3133                    0.3233
## Detection Rate            0.1167                    0.1300
## Detection Prevalence      0.3133                    0.3933
## Balanced Accuracy         0.5430                    0.5064
##                      Class: Mobile Payment App
## Sensitivity                             0.3028
## Specificity                             0.7120
## Pos Pred Value                          0.3750
## Neg Pred Value                          0.6415
## Prevalence                              0.3633
## Detection Rate                          0.1100
## Detection Prevalence                    0.2933
## Balanced Accuracy                       0.5074
```

As shown above, the heterogeneous ensemble method does boost the overall accuracy rate a tiny bit to 35.7%. The Kappa is positive but, again, is lower than 0.1.

8. Comparison & Conclusion

8.1 Comparison of individual models

As you may have noticed before, my project might be a special case because the dataset itself is not good enough to build a classifier. To recap: after I went through all the model construction and tuning and started suspecting the dataset, I noticed that the Kaggle page had added a sentence - "...and all data is entirely fictional", which might be why this dataset is bad.

No matter which model I tried to build - kNN, Decision Trees, or SVM - the overall accuracy rate of the prediction is always between 30% and 35%. Even after model tunning, such as trying different k values, adjusting mistake costs, or changing kernels, the accuracy rate still lies between 30% and 35%. Honestly, I cannot tell which model or which hyperparameter adjustment fits best for this classification task. Nevertheless, if I have to choose, I will pick Decision Trees because it supports the only classifier that (sometimes) returns both a positive Kappa value and an accuracy rate close to 35%.

To further prove my point that the poor quality of this dataset contributes to the low accuracy rates and Kappa scores, I found another dataset that can be used to build similar classifiers and did result in better performance. Thanks to Professor Assareh's advice, I will elaborate on the second version of my final project in my video presentation.

8.2 Comparison of ensemble to individual models

Unsurprisingly, the ensemble method failed to improve the classifier performance either. The homogeneous ensemble resulted in a 33% accuracy rate and a negative Kappa score while the heterogenous ensemble function resulted in a 35.7% accuracy rate. Again, the accuracy rates still fall between 30% and 35%... I think in this case, when the dataset is not qualified enough, the ensemble models can't improve the model performance either. Nevertheless, if I have to choose, the heterogenous ensemble model performs the best

because it results in the highest accuracy rate (although just 35.7%) and a positive Kappa score (although less than 0.1).

8.3 Last resort: logistic regression model with combined target variables

After all these failed efforts, I started wondering if a binary target variable would help. Therefore, I integrated the target variable into "Cash" and "Non-Cash." To demonstrate my assumption, I tried out a logistic regression model as following.

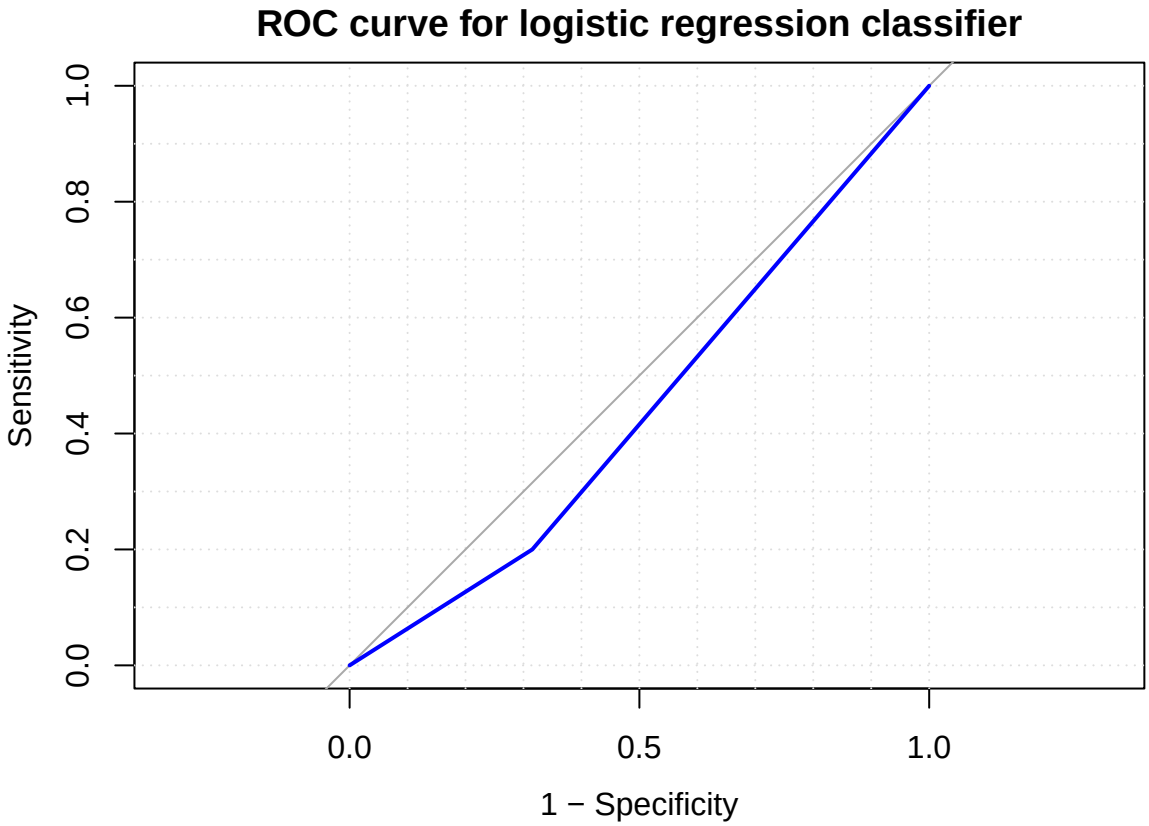
```
# integrate the target variable into two categories: cash, non-cash (credit card/ mobile app).
studentTrain.labels.logistic <- ifelse(studentTrain.labels=="Cash", 1,0)
studentTest.labels.logistic <- ifelse(studentTest.labels =="Cash", 1,0)

# build train and test set for logistic regression
studentTrain_reg <- cbind(studentTrain.standardized, studentTrain.labels.logistic) %>%
  rename(preferred_payment_method = studentTrain.labels.logistic)

studentTest_reg <- cbind(studentTest.standardized, studentTest.labels.logistic) %>%
  rename(preferred_payment_method = studentTest.labels.logistic)

# construct logistic regression model
m <- glm(preferred_payment_method ~. , family = binomial(link="logit"), data=studentTrain_reg)
#summary(m)

# predict with logistic regression model
payment_predict_reg <- predict(m, studentTest_reg, type="response")
payment_predict_reg <- ifelse(payment_predict_reg > 0.5, 1, 0)

# model evaluation
confusionMatrix(factor(payment_predict_reg, levels=c(0,1)), factor(studentTest_reg$preferred_payment_me
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 202  93
##          1   4   1
##
##                Accuracy : 0.6767
##                  95% CI : (0.6205, 0.7293)
##     No Information Rate : 0.6867
##     P-Value [Acc > NIR] : 0.6707
##
##                   Kappa : -0.0118
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.010638
##             Specificity : 0.980583
##          Pos Pred Value : 0.200000
##          Neg Pred Value : 0.684746
##              Prevalence : 0.313333
##          Detection Rate : 0.003333
```

```
##    Detection Prevalence : 0.016667
##       Balanced Accuracy : 0.495610
##
##        'Positive' Class : 1
##
```

As shown above, the overall accuracy rate of the logistic regression model is (202+1)/300 = 67.7%, a decent accuracy rate. However, if we look closer, we can notice that most of the correct predictions come from the "Non-Cash (0)" category, which is also the dominant class in both the train and test sets. In the train set, about 69.1% belong to the "Non-Cash" category while in the test set, about 68.7% belong to that category. Nevertheless, I don't think the integrated target variable would help the model performance because the Kappa value is still negative! I guess this also proved my point that this dataset is not good enough for classifier model building.

```
logistic_roc <- roc(payment_predict_reg, studentTest_reg$preferred_payment_method)
plot(logistic_roc, main="ROC curve for logistic regression classifier", col="blue", lwd=2, grid=TRUE, l
```



The ROC curve further demonstrates how bad this classifier is, despite a reasonable accuracy rate. As shown above, since the curve falls below the diagonal, it shows that predictions are opposite the reference values.

If you are wondering about the second version of my final project, where I can construct similar models and test out tuning techniques based on a better dataset, please feel free to check out the second part of my video presentation.