Assignment 2 File

Due March 29, 2022

Worth 40 points total

Problem Overview

Lending Club is an online, peer-to-peer marketplace that connects borrowers and investors. This assignment asks you to do some predictive modeling on a dataset of past Lending Club loans, including loan details and information about the borrowers. A full data dictionary can be found in LCDataDictionary.xlsx.

The goal of this assignment is to get hands-on practice with data cleaning, feature engineering, and predictive modeling algorithms beyond the basics, including classification trees, kNNs, and regularized logistic regression. You will also practice creating and interpreting ROC and lift curves.

You will be predicting whether loans were paid in full or not. Your intended use case is to help an organization decide which loans to "flag" as potentially risky investments.

RUBRIC: There are three possible grades on this assignment: Fail (F), Pass (P), and High Pass (H). If you receive an F then you will have one more chance to turn it in to receive a P. If you receive H on 3 out of the 4 assignments this semester you'll get a bonus point on your final average.

1. Turn in a well-formatted compiled HTML document using R markdown. If you turn in a different file type or your code doesn't compile, you will be asked to redo the assignment.
2. Provide clear answers to the questions and the correct R commands as necessary, in the appropriate places. You may answer up to three sub-questions incorrectly and still receive a P on this assignment (for example, 1(a) counts as one sub-question). If you answer all sub-questions correctly on your first submission you will receive an H.
3. The entire document must be clear, concise, readable, and well-formatted. If your assignment is unreadable or if you include more output than necessary to answer the questions you will be asked to redo the assignment.

Note that this assignment is somewhat open-ended and there are many ways to answer these questions. I don't require that we have exactly the same answers in order for you to receive full credit.

The following code block does some intial setup, including:

1. Reading the dataset (make sure to set your working diretory)
2. Creating the target variable
3. Setting the random seed
4. Selecting a subsample of 15k observations in order to speed up training time
5. Selecting the training and validation row numbers

```
#read in data
lc <- read_csv("LendingClub_LoanStats_2011_v2.csv")  #read the Lending Club dataset in R
```

```
## Warning: One or more parsing issues, see `problems()` for details
```

```
## Rows: 42538 Columns: 112
```

```
## ── Column specification ────────────────────────────────────────────────────────
## Delimiter: ","
## chr (23): term, int_rate, grade, sub_grade, emp_title, emp_length, home_owne...
## dbl (34): id, member_id, loan_amnt, funded_amnt, funded_amnt_inv, installmen...
## lgl (55): initial_list_status, mths_since_last_major_derog, annual_inc_joint...
```

```
##
## ℹ Use `spec()` to retrieve the full column specification for this data.
## ℹ Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
#create target variable: fully paid
#remove any rows where y is NA
lc <- lc %>%
  mutate(y = as.factor(ifelse(loan_status == "Fully Paid", "Paid", "Not Paid"))) %>%
  filter(!is.na(y))

#set seed and randomly downsample 15k instances
#(otherwise training kNN will take hours)
set.seed(1)
lc_small <- sample(nrow(lc), 15000)
lc <- lc[lc_small,]

#then calculate the training/validation row numbers, but don't yet split
va_inst <- sample(nrow(lc), .3*nrow(lc))
```

0: Example answer

What is the mean loan amount in this dataset??

ANSWER: The loan amount in this dataset is $10,993.60.

```
loan_mean <- lc %>%
  summarise(mean_amt = mean(loan_amnt))
loan_mean
```

mean_amt
<dbl>
10993.6

1 row

1: Data Cleaning and Feature Engineering

a. Clean and process the following variables in the Lending Club dataset:

- grade, sub_grade, home_ownership, addr_state: check if there are NAs and add a NULL value if so
- loan_amnt: check if there are NAs, and if there are, replace with the mean loan amount (by grade)
- emp_length: group into bins: <1 year, 1-3 years, 4-6 years, 7-9 years, 10+ years, and "unknown"
- annual_inc: replace NAs with the average value, then group into four bins based on quartile values
- purpose: any factor levels with fewer than 200 instances should get grouped into "other". Also, combine credit_card and debt_consolidation into "debt".
- dti: group into five equally-sized bins
- mths_since_last_delinq: group into five bins (< 1 year, 1-2 years, 2-3 years, 3+ years, never)
- int_rate: make sure this has the correct data type, process it if not. Check for NAs and replace by the mean.
- y: convert into a factor

ANSWER TO QUESTION 1a HERE:

```
#grade, sub_grade, home_ownership, addr_state: check if there are NAs and add a NULL value if so
lc_clean = lc %>%
  mutate(grade = ifelse(is.na(grade),NULL,grade),
         sub_grade = ifelse(is.na(sub_grade),NULL,sub_grade),
         home_ownership = ifelse(is.na(home_ownership),NULL,home_ownership),
         addr_state = ifelse(is.na(addr_state),NULL,addr_state)) %>%
  #loan_amnt: check if there are NAs, and if there are, replace with the mean loan amount (by grade)
  group_by(grade) %>%
         mutate(loan_amnt = ifelse(is.na(loan_amnt),mean(loan_amnt),loan_amnt)) %>%
  ungroup()
summary(lc_clean$loan_amnt)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     500    5000    9600   10994   15000   35000
```

```
#emp_length: group into bins: <1 year, 1-3 years, 4-6 years, 7-9 years, 10+ years, and "unknown"
lc_clean= lc_clean%>%
  mutate( emp_length = case_when(
          emp_length == "< 1 year" ~ "< 1 year",
          emp_length == "1 year" | emp_length =="2 years" | emp_length =="3 years" ~ "1-3 years",
          emp_length == "4 years" | emp_length =="5 years" | emp_length =="6 years" ~ "4-6 years",
          emp_length == "7 years" | emp_length =="8 years" | emp_length =="9 years" ~ "7-9 years",
          emp_length == "10+ years" ~ "10+ years",
          TRUE ~ "unknown"),
         #annual_inc: replace NAs with the average value, then group into four bins based on quartile val
ues
         annual_inc = ifelse(is.na(annual_inc),mean(lc_clean$annual_inc,na.rm=TRUE),annual_inc),
         annual_inc = ntile(annual_inc, 4)
         )
summary(as.factor(lc_clean$emp_length))
```

```
##  < 1 year 1-3 years 10+ years 4-6 years 7-9 years   unknown
##      1806      4480      3332      3355      1653       374
```

```
#filter(lc_clean,is.na(lc_clean$annual_inc))
summary(as.factor(lc_clean$annual_inc))
```

```
##    1    2    3    4
## 3750 3750 3750 3750
```

```
#purpose: any factor levels with fewer than 200 instances should get grouped into "other". Also, combine
 credit_card and debt_consolidation into "debt".
lc_clean= lc_clean%>%
  group_by(purpose) %>%
  mutate(count = n(),
         purpose = ifelse(purpose < 200, "other",purpose),
         purpose = ifelse(purpose == "credit_card" | purpose == "debt_consolidation", "debt", purpose)) %
>%
  ungroup()
head(lc_clean$purpose)
```

```
## [1] "debt"  "debt"  "debt"  "debt"  "debt"  "other"
```

```
#dti: group into five equally-sized bins
lc_clean=lc_clean%>%
   mutate(dti = cut(dti,breaks = 5))
summary(lc_clean$dti)
```

```
## (-0.03,6]    (6,12]   (12,18]   (18,24]   (24,30]
##      2486      3942      4351      3510       711
```

```
#mths_since_last_delinq: group into five bins (< 1 year, 1-2 years, 2-3 years, 3+ years, never)
lc_clean=lc_clean%>%
   mutate( mths_since_last_delinq = case_when(
           mths_since_last_delinq == 0 ~ "< 1 year",
           mths_since_last_delinq == 1 | mths_since_last_delinq == 2 ~ "1-2 years",
           mths_since_last_delinq == 2 | mths_since_last_delinq == 3 ~ "2-3 years",
           is.na(mths_since_last_delinq) ~ "never",
           TRUE ~ "3+ years"))
summary(as.factor(lc_clean$mths_since_last_delinq))
```

```
##  < 1 year 1-2 years 2-3 years  3+ years     never
##       306        49        47      5129      9469
```

```
#int_rate: make sure this has the correct data type, process it if not. Check for NAs and replace by the
 mean.
lc_clean=lc_clean%>%
  mutate(int_rate = as.numeric(str_sub(int_rate,start = 1L,end = -2L))/100, #e.g "6.17%"
         int_rate = ifelse(is.na(int_rate),mean(int_rate),int_rate))
summary(lc_clean$int_rate)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0542  0.0962  0.1199  0.1218  0.1474  0.2411
```

```
#y: convert into a factor
lc_clean=lc_clean%>%
  mutate(y=as.factor(y))
summary(lc_clean$y)
```

```
## Not Paid     Paid
##     3333    11667
```

b. At this point you should have 12 cleaned variables (including y). +Convert the set of cleaned and processed variables into dummy variables. Also create dummy variables that interact the (binned) annual income and (binned) employment length variables. +You will end up with dummy variables for both y = "Paid" and y = "Not Paid". Drop the y = "Paid" dummy variable and convert the "Not Paid" dummy variable into a factor. +How many variables do you have after converting to dummies and dropping the y = Paid dummy? +Finally, partition your dataset into "train" and "test" using the va_inst row numbers sampled above.

ANSWER TO QUESTION 1b HERE: There are 136 variables after converting to dummies and dropping the y=Paid dummy

```
#select 12 cleaned variables above
lc_clean = lc_clean %>%
  select(grade, sub_grade, home_ownership, addr_state, loan_amnt, emp_length,annual_inc,purpose,dti,mths_
since_last_delinq,int_rate,y)

#Convert the set of cleaned and processed variables into dummy variables(the dummyVars will transform all
characters and factors columns (the function never transforms numeric columns) and return the entire data
set:).
# dummify the data
dummy=dummyVars(~.,data=lc_clean)
one_hot_lc = data.frame(predict(dummy,newdata=lc_clean))
head(one_hot_lc) #132 columns
```

	gradeA <dbl>	gradeB <dbl>	gradeC <dbl>	gradeD <dbl>	gradeE <dbl>	gradeF <dbl>	gradeG <dbl>	sub_gradeA1 <dbl>	sub_gradeA2 <dbl>
1	1	0	0	0	0	0	0	0	0
2	0	1	0	0	0	0	0	0	0
3	1	0	0	0	0	0	0	0	0
4	0	0	1	0	0	0	0	0	0
5	0	0	1	0	0	0	0	0	0
6	1	0	0	0	0	0	0	0	0

6 rows | 1-10 of 132 columns

```
# Also create dummy variables that interact the (binned) annual income and (binned) employment length var
iables.
one_hot_lc$annual_inc = as.factor(one_hot_lc$annual_inc)
summary(one_hot_lc$annual_inc)
```

```
##    1    2    3    4 
## 3750 3750 3750 3750
```

```
dummy_interact=dummyVars(~(annual_inc*emp_length),data = lc_clean)
one_hot_interact=data.frame(predict(dummy_interact,newdata=lc_clean))
head(one_hot_interact)
```

	annual_inc <dbl>	emp_length..1.year <dbl>	emp_length1.3.years <dbl>	emp_length10..years <dbl>	emp_length4.6.years <dbl>
1	1	0	1	0	0
2	2	0	0	1	0
3	1	0	0	0	1
4	3	0	0	1	0
5	4	1	0	0	0
6	4	0	0	0	0

6 rows | 1-6 of 14 columns

```
#combine the interaction variable into the dummified dataset
one_hot_lc_all <- cbind(one_hot_lc,one_hot_interact)
head(one_hot_lc_all) #145 columns
```

	gradeA <dbl>	gradeB <dbl>	gradeC <dbl>	gradeD <dbl>	gradeE <dbl>	gradeF <dbl>	gradeG <dbl>	sub_gradeA1 <dbl>	sub_gradeA2 <dbl>
1	1	0	0	0	0	0	0	0	0
2	0	1	0	0	0	0	0	0	0
3	1	0	0	0	0	0	0	0	0
4	0	0	1	0	0	0	0	0	0
5	0	0	1	0	0	0	0	0	0
6	1	0	0	0	0	0	0	0	0

6 rows | 1-10 of 145 columns

```
#You will end up with dummy variables for both y = “Paid” and y = “Not Paid”. Drop the y = “Paid” dummy v
ariable and convert the “Not Paid” dummy variable into a factor.

# Find Duplicate Column Names
duplicated_names = duplicated(colnames(one_hot_lc_all))
summary(duplicated_names) #7 duplicated columns
```

```
##    Mode   FALSE    TRUE
## logical     137       7
```

```
# Remove Duplicate Column Names
one_hot_lc_all=one_hot_lc_all[!duplicated_names]

#drop y = paid
one_hot_lc_all<-one_hot_lc_all %>%
  select(-(y.Paid))

#convert y = not paid to a factor
one_hot_lc_all$y.Not.Paid = as.factor(one_hot_lc_all$y.Not.Paid)

#number of variables
ncol(one_hot_lc_all)
```

```
## [1] 136
```

```
#Finally, partition your dataset into “train” and “test” using the va_inst row numbers sampled above.
lc_test= one_hot_lc_all[va_inst,]
#test dataset
lc_train= one_hot_lc_all[-va_inst,]
data.frame(nrow(one_hot_lc_all[va_inst,]),nrow(one_hot_lc_all[-va_inst,]))
```

nrow.one_hot_lc_all.va_inst.... <int>	nrow.one_hot_lc_all..va_inst.... <int>
4500	10500

1 row

2: Trees

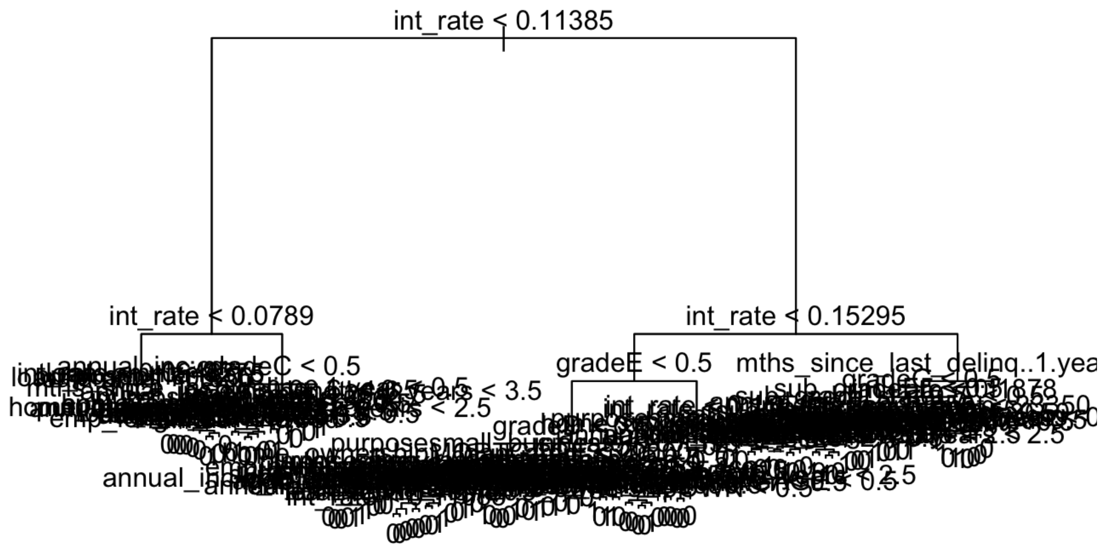
a. Use the following code to create an unpruned tree (replace YOUR_Y_VAR and YOUR_TRAINING_DATA with the appropriate variable names, then uncomment the line starting with “lc.full.tree=tree…”). How many terminal nodes are in the full tree? Which variable has the highest-information gain (leads to the biggest decrease in impurity)? How do you know?

ANSWER TO QUESTION 2a HERE:There are 136 terminal nodes. The root nodes int_rate<0.11385 has the highest-information gain since the first node has the highest decrease in impurity

```
# Specify unpruned tree
mycontrol = tree.control(nrow(one_hot_lc_all), mincut = 1, minsize = 2, mindev = 0.0004)
lc.full.tree=tree(y.Not.Paid ~ .,one_hot_lc_all,control = mycontrol)

summary(lc.full.tree)
```

```
##
## Classification tree:
## tree(formula = y.Not.Paid ~ ., data = one_hot_lc_all, control = mycontrol)
## Variables actually used in tree construction:
##  [1] "int_rate"                         "annual_inc"
##  [3] "loan_amnt"                        "purposeother"
##  [5] "gradeC"                           "mths_since_last_delinq..1.year"
##  [7] "purposedebt"                      "sub_gradeB2"
##  [9] "purposewedding"                   "home_ownershipRENT"
## [11] "addr_stateFL"                     "addr_stateTX"
## [13] "emp_lengthunknown"                "annual_inc.emp_length4.6.years"
## [15] "dti..24.30."                      "emp_length1.3.years"
## [17] "addr_stateOR"                     "addr_stateNM"
## [19] "dti...0.03.6."                    "gradeE"
## [21] "gradeD"                           "purposesmall_business"
## [23] "home_ownershipMORTGAGE"           "gradeF"
## [25] "sub_gradeC3"                      "annual_inc.emp_length1.3.years"
## [27] "mths_since_last_delinq3..years" "addr_stateMA"
## [29] "addr_stateKS"                     "addr_stateWI"
## [31] "emp_length10..years"              "annual_inc.emp_lengthunknown"
## [33] "emp_length7.9.years"              "sub_gradeB5"
## [35] "addr_stateSC"                     "sub_gradeC2"
## [37] "addr_stateNY"                     "mths_since_last_delinqnever"
## [39] "sub_gradeD4"                      "home_ownershipOWN"
## [41] "annual_inc.emp_length10..years" "gradeG"
## [43] "sub_gradeF5"                      "sub_gradeD1"
## [45] "sub_gradeF3"                      "sub_gradeE4"
## [47] "addr_stateAZ"                     "addr_statePA"
## [49] "addr_stateIL"                     "dti..12.18."
## [51] "addr_stateNC"                     "emp_length..1.year"
## [53] "addr_stateGA"
## Number of terminal nodes:  136
## Residual mean deviance:  0.8717 = 12960 / 14860
## Misclassification error rate: 0.1919 = 2878 / 15000
```

```
plot(lc.full.tree)
text(lc.full.tree)
```

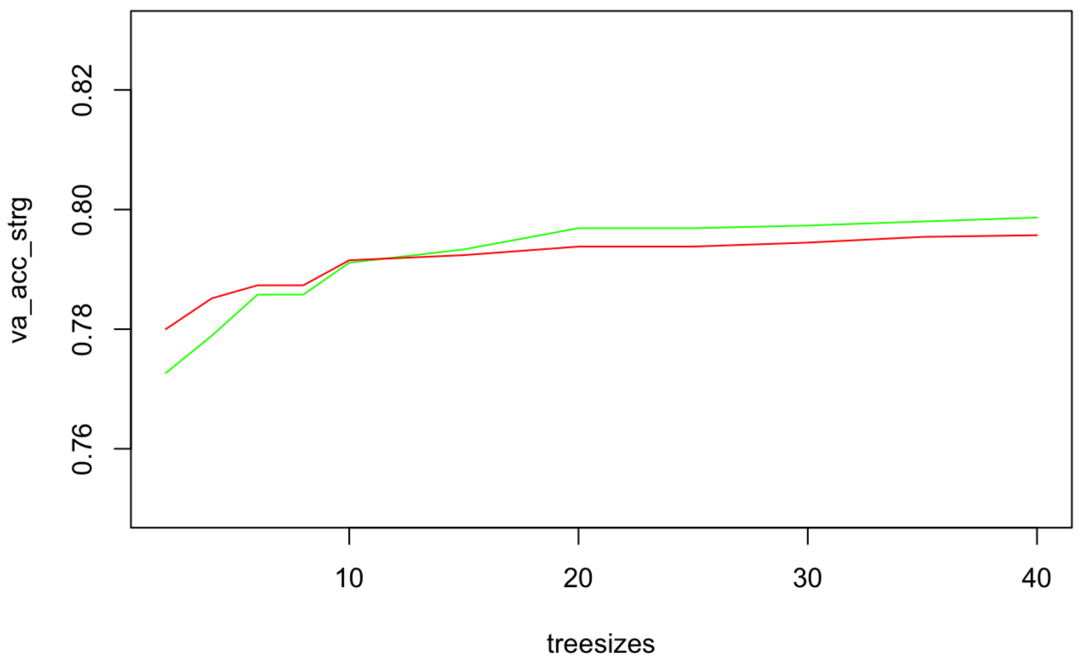


b. Create pruned trees of size 2, 4, 6, 8, 10, 15, 20, 25, 30, 35, and 40. Plot fitting curves consisting of the accuracy in the validation and training sets for each pruned tree (assuming a cutoff of 0.5). Make sure the two sets of points are different colors.

ANSWER TO QUESTION 2b HERE:

```r
#Define the function to calculate the model accuracy under a set of classification result
accuracy = function(classifications, actuals){
  correct_classifications = ifelse(classifications == actuals, 1,0)
  acc = sum(correct_classifications)/length(classifications)
  return(acc)
}
#Define the function to calculate the model accuracy given a specific number of pruned tree size
pred_and_eval=function(treename, pred_data, cutoff){
  predictions = predict(treename, newdata = pred_data)
  probs = predictions[,2]
  classifications <- ifelse(probs > cutoff, 1, 0)
  acc = accuracy(classifications,pred_data$y.Not.Paid)
  return(acc)
}
```

```
#define a vector of various tree sizes
treesizes <- c(2,4,6,8,10,15,20,25,30,35,40)
va_acc_strg=rep(0,length(treesizes))
tr_acc_strg=rep(0,length(treesizes))

for(i in 1:length(treesizes)){

  size = treesizes[i]

  pruned_tree = prune.tree(lc.full.tree, best = size)

  va_acc <- pred_and_eval(pruned_tree, lc_test, 0.5)
  tr_acc <- pred_and_eval(pruned_tree, lc_train, 0.5)
  va_acc_strg[i] <- va_acc
  tr_acc_strg[i] <- tr_acc

}

plot(treesizes, va_acc_strg, type = 'l', col= 'green', ylim = c(0.75,0.83))
lines(treesizes, tr_acc_strg, col = 'red')
```



c. Which tree size is the best, and how did you select the best one? Store the vector of probabilities estimated by your best tree in the validation set as a variable called best.tree.preds. We'll use these later.

ANSWER TO QUESTION 2c HERE: The tree size 40 is the best with the highest accuracy of 0.798666666666667 among the set of tree sizes.

```
table(treesizes,va_acc_strg)
```

```
##          va_acc_strg
## treesizes 0.772666666666667 0.778888888888889 0.785777777777778
##        2                 1                 0                 0
##        4                 0                 1                 0
##        6                 0                 0                 1
##        8                 0                 0                 1
##        10                0                 0                 0
##        15                0                 0                 0
##        20                0                 0                 0
##        25                0                 0                 0
##        30                0                 0                 0
##        35                0                 0                 0
##        40                0                 0                 0
##          va_acc_strg
## treesizes 0.791111111111111 0.793333333333333 0.796888888888889
##        2                 0                 0                 0
##        4                 0                 0                 0
##        6                 0                 0                 0
##        8                 0                 0                 0
##        10                1                 0                 0
##        15                0                 1                 0
##        20                0                 0                 1
##        25                0                 0                 1
##        30                0                 0                 0
##        35                0                 0                 0
##        40                0                 0                 0
##          va_acc_strg
## treesizes 0.797333333333333 0.798 0.798666666666667
##        2                 0     0                 0
##        4                 0     0                 0
##        6                 0     0                 0
##        8                 0     0                 0
##        10                0     0                 0
##        15                0     0                 0
##        20                0     0                 0
##        25                0     0                 0
##        30                1     0                 0
##        35                0     1                 0
##        40                0     0                 1
```

```
best.tree = prune.tree(lc.full.tree, best = 40)
best.tree.probs = predict(best.tree, newdata = lc_test)[,2]
head(best.tree.probs)
```

```
##       3004       5192       3307      10590       6128       5909
## 0.01044386 0.30000000 0.32241814 0.48407643 0.44476744 0.07347254
```

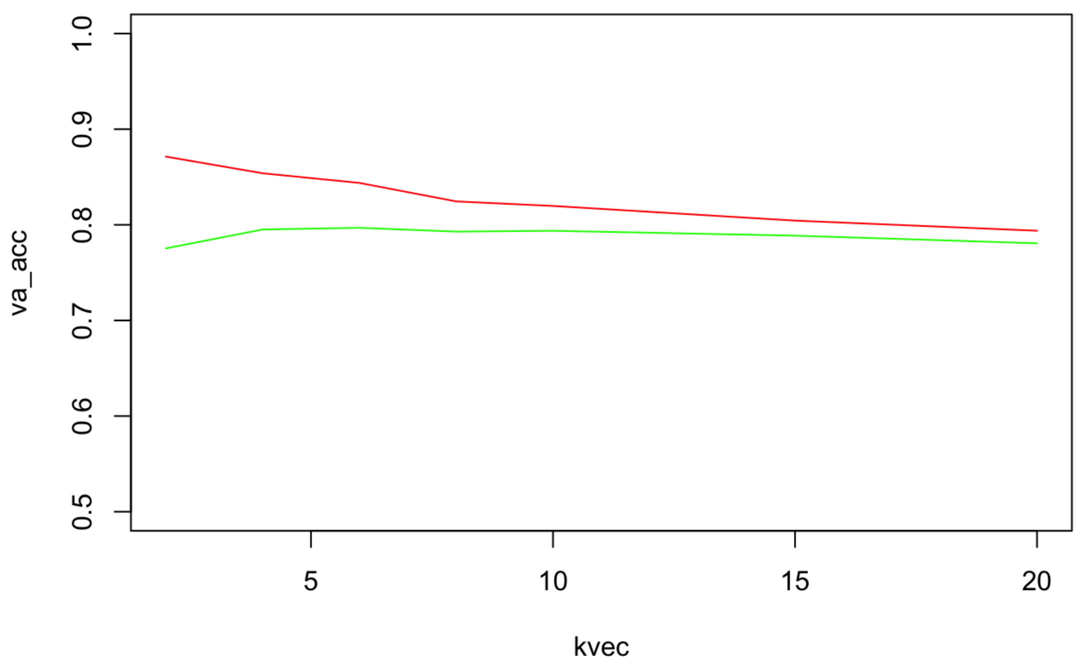
3: kNN

a. Compute kNN estimates in the training and validation data using k values of 2, 4, 6, 8, 10, 15, and 20. Assume a cutoff of 0.5. Plot the accuracy in the validation and training sets for each k value. Make sure the two sets of points are different colors!

+Note: you will need to separate your training and validation sets into X and y. +Note: Be patient - it will take several minutes for kNN to make its predictions!

ANSWER TO QUESTION 3a HERE:

```r
lc_train_no_na = na.omit(lc_train)
lc_test_no_na = na.omit(lc_test)

train.X = lc_train_no_na
valid.X = lc_test_no_na

train.y = lc_train_no_na$y.Not.Paid
valid.y = lc_test_no_na$y.Not.Paid

#establish K values of 2,4,6,8,10,15,20
kvec = c(2, 4, 6, 8, 10, 15, 20)

#initialize storage
va_acc = rep(0,length(kvec))
tr_acc = rep(0,length(kvec))


for(i in 1:length(kvec)){
  k = kvec[i]

  #compute predictions using train.x as the potential neighbors
  #new points to be classified are first validation, then training points
  va_preds = knn(train.X, valid.X, train.y, k = k )
  tr_preds = knn(train.X, train.X, train.y, k = k )

  #compute the accuracy for each set of predictions
  va_accuracy = accuracy(va_preds, valid.y)
  tr_accuracy = accuracy(tr_preds, train.y)

  #store in the appropriate place
  va_acc[i] = va_accuracy
  tr_acc[i] = tr_accuracy
}

#make the plot
plot(kvec,va_acc, col = "green", type = "l", ylim = c(0.5,1))
lines(kvec, tr_acc, col = "red")
```



b. Which k is the best, and how did you select the best one? Store the vector of probabilities estimated by your best k value in the validation set as a variable called best.knn.preds. +

+Note: you’ll need to convert these probabilities from the probability of the majority-class vote to the probability that y = the positive class.

ANSWER TO QUESTION 3b HERE:The best k is 6 with a accuracy of 0.796888888888889, which is the highest accuracy.

```
table(kvec,va_acc)
```

```
##      va_acc
## kvec 0.775333333333333 0.780666666666667 0.788666666666667 0.792888888888889
##   2                  1                 0                 0                 0
##   4                  0                 0                 0                 0
##   6                  0                 0                 0                 0
##   8                  0                 0                 0                 1
##   10                 0                 0                 0                 0
##   15                 0                 0                 1                 0
##   20                 0                 1                 0                 0
##      va_acc
## kvec 0.793777777777778 0.795111111111111 0.796888888888889
##   2                  0                 0                 0
##   4                  0                 1                 0
##   6                  0                 0                 1
##   8                  0                 0                 0
##   10                 1                 0                 0
##   15                 0                 0                 0
##   20                 0                 0                 0
```

```
#Store the vector of probabilities estimated by your best k value in the validation set as a variable cal
led best.knn.preds.
knn_onerow = knn(train.X,valid.X,train.y,k=6,prob = T)
summary(knn_onerow)
```

```
##    0    1
## 4158  342
```

```
knn_oneprob = attr(knn_onerow,"prob")
summary(knn_oneprob)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.5000  0.7143  0.8571  0.8453  1.0000  1.0000
```

```
best.knn.preds = ifelse(knn_onerow ==1, knn_oneprob, 1-knn_oneprob)
summary(best.knn.preds)
```

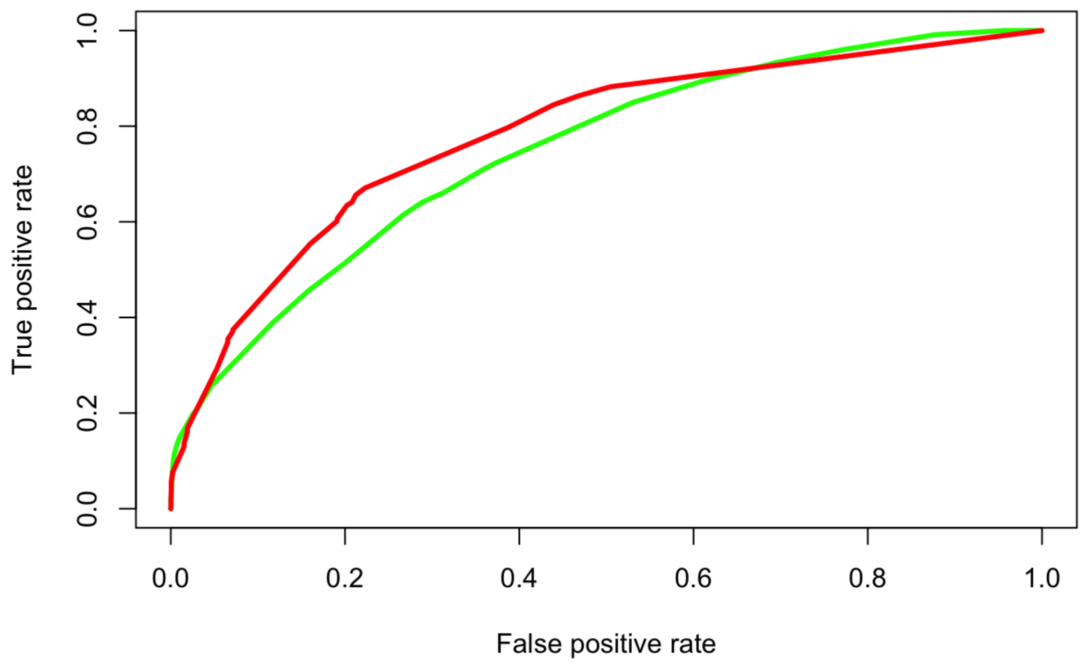
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0000  0.0000  0.1429  0.1757  0.2857  1.0000
```

4: ROC and Lift

a. Plot the ROC curves for the probability estimates generated by your best tree and kNN models on the same chart (make sure that your reader can tell which line is which).

ANSWER TO QUESTION 4a HERE: The green line represents the ROC curves for the best tree model. The red line represents the ROC curves for the knn model.

```
pred_tree = prediction(best.tree.probs, lc_test$y.Not.Paid)
roc_tree = performance(pred_tree,"tpr","fpr")
plot(roc_tree, col = "green", lwd =3)

pred_knn = prediction(best.knn.preds, lc_test$y.Not.Paid)
roc_knn = performance(pred_knn,"tpr","fpr")
plot(roc_knn, add = T, col = "red", lwd =3)
```



b. Compute the AUC for each model (in the validation data, of course. Which model has the highest AUC? Is this the same model as the one with the higher accuracy? Does the highest-AUC model have the highest TPR for every cutoff?

ANSWER TO QUESTION 4b HERE:The knn model has the highest AUC. It's not the same model as the one with the higher accuracy. The highest-AUC model doesn't have the highest TPR for every cutoff

```
auc_tree = performance(pred_tree, measure = "auc")@y.values[[1]]
auc_knn = performance(pred_knn, measure = "auc")@y.values[[1]]
data.frame(auc_tree,auc_knn)
```

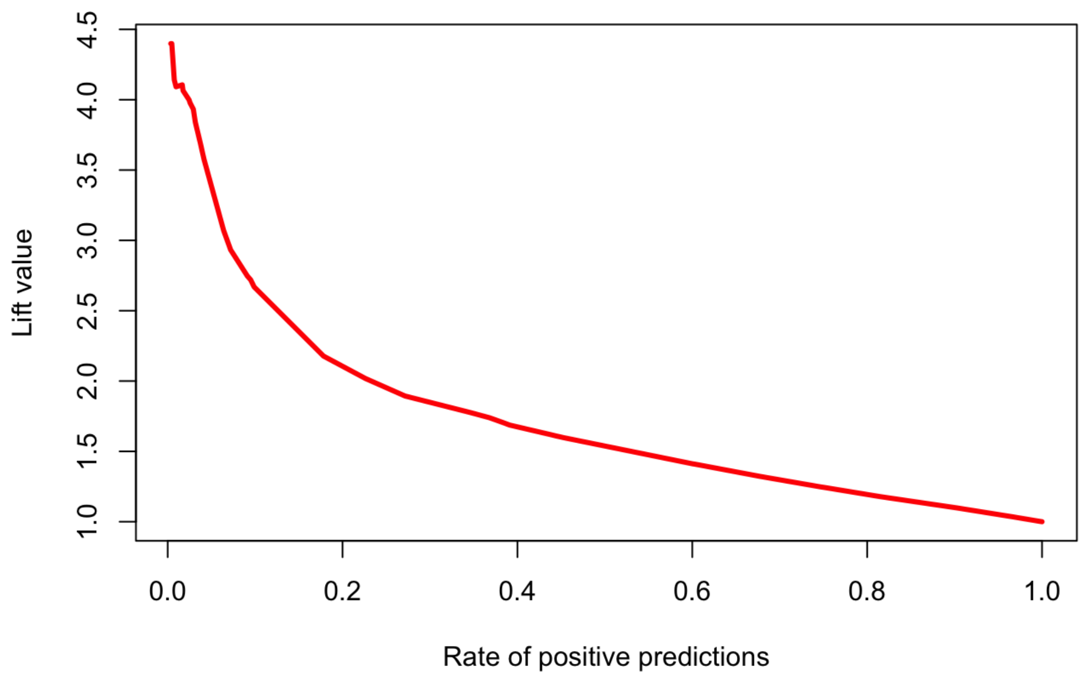
auc_tree <dbl>	auc_knn <dbl>
0.746574	0.7815081

1 row

c. Plot the lift curve for your highest-AUC model.

ANSWER TO QUESTION 4c HERE: The plot is shown as below

```
lift <- performance(pred_tree, "lift", "rpp")
plot(lift,col = "red",lwd = 3)
```



d. If we decide to flag the top 10% of loans most likely to be "Not Paid", what will our lift be? What if we flag the top 50% of loans? If we want to achieve a lift of at least 2.0, how many loans should we flag?

+Note: you can answer approximately by reading your lift chart, no need to calculate the exact amounts.

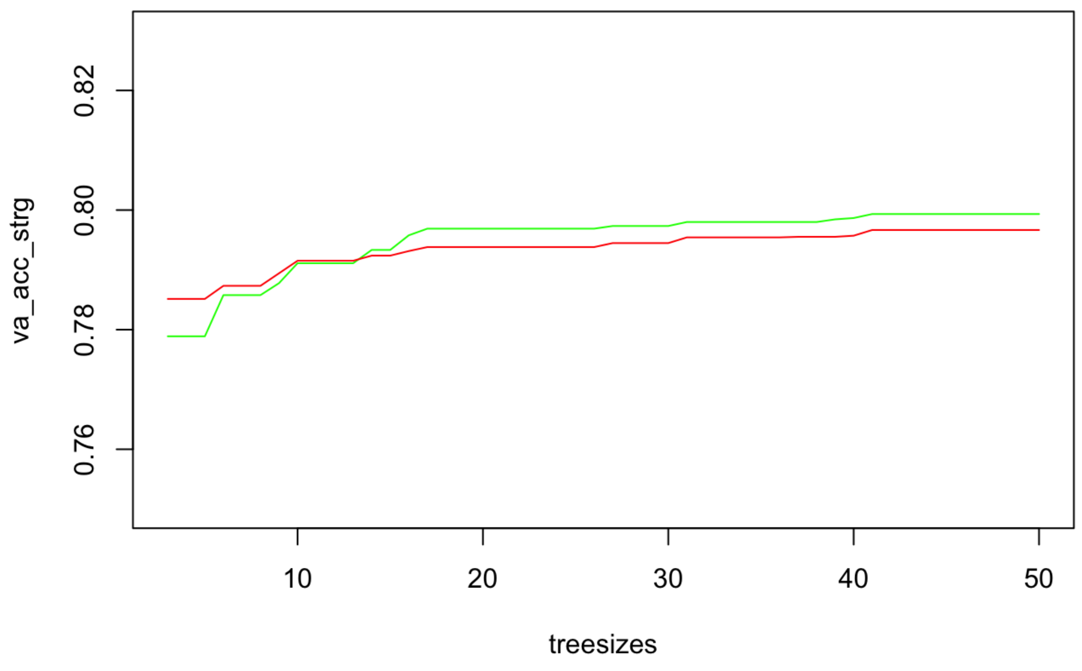
ANSWER TO QUESTION 4d HERE:If we decide to flag the top 10% of loans most likely to be "Not Paid", the lift will be 2.7. If we want to achieve a lift of at least 2.0, we should flag top 22% of loans.

5: OPTIONAL (Extra Challenge)

Can you improve on the best-performing model with either more/different features, a different model specification, or a different tuning parameter (or all of the above)? Report your best validation performance and give details on your best model.

ANSWER TO QUESTION 5 HERE:Since I cannot find a better KNN model, I turned out to choose tree model. For the tree model, I found that when the tree size is 41 to 50, the highest accuracy could be 0.799333333333333, which is better than the previous model with tree size 40. When I chose tree size to be 50, the AUC increased to be 0.7565516

```
mycontrol= tree.control(nrow(one_hot_lc_all), mincut = 1, minsize = 2, mindev = 0.0004)
lc.full.tree_=tree(y.Not.Paid ~ .,control = mycontrol, one_hot_lc_all)

treesizes= c(3:50)

va_acc_strg= rep(0,length(treesizes))
tr_acc_strg= rep(0,length(treesizes))

for(i in 1:length(treesizes)){

  size = treesizes[i]

  pruned_tree = prune.tree(lc.full.tree, best = size)

  va_acc = pred_and_eval(pruned_tree, lc_test, 0.5)
  tr_acc = pred_and_eval(pruned_tree, lc_train, 0.5)

  va_acc_strg[i] <- va_acc
  tr_acc_strg[i] <- tr_acc

}

best.tree = prune.tree(lc.full.tree, best = 50)
best.tree.probs = predict(best.tree, newdata = lc_test)[,2]
pred_tree = prediction(best.tree.probs, lc_test$y.Not.Paid)
#plot the accuracy graph
plot(treesizes, va_acc_strg, type = 'l', col= 'green', ylim = c(0.75,0.83))
lines(treesizes, tr_acc_strg, col = 'red')
```



```
auc_tree = performance(pred_tree, measure = "auc")@y.values[[1]]
auc_tree
```

```
## [1] 0.7565516
```

```
table(treesizes,va_acc_strg)
```

```
##          va_acc_strg
## treesizes 0.778888888888889 0.785777777777778 0.787777777777778
##        3                 1                 0                 0
##        4                 1                 0                 0
##        5                 1                 0                 0
##        6                 0                 1                 0
##        7                 0                 1                 0
##        8                 0                 1                 0
##        9                 0                 0                 1
##        10                0                 0                 0
##        11                0                 0                 0
##        12                0                 0                 0
##        13                0                 0                 0
##        14                0                 0                 0
##        15                0                 0                 0
##        16                0                 0                 0
##        17                0                 0                 0
##        18                0                 0                 0
##        19                0                 0                 0
##        20                0                 0                 0
##        21                0                 0                 0
##        22                0                 0                 0
##        23                0                 0                 0
##        24                0                 0                 0
##        25                0                 0                 0
##        26                0                 0                 0
##        27                0                 0                 0
##        28                0                 0                 0
##        29                0                 0                 0
##        30                0                 0                 0
##        31                0                 0                 0
##        32                0                 0                 0
##        33                0                 0                 0
##        34                0                 0                 0
##        35                0                 0                 0
##        36                0                 0                 0
##        37                0                 0                 0
##        38                0                 0                 0
##        39                0                 0                 0
##        40                0                 0                 0
##        41                0                 0                 0
##        42                0                 0                 0
##        43                0                 0                 0
##        44                0                 0                 0
##        45                0                 0                 0
##        46                0                 0                 0
##        47                0                 0                 0
##        48                0                 0                 0
##        49                0                 0                 0
##        50                0                 0                 0
##          va_acc_strg
## treesizes 0.791111111111111 0.793333333333333 0.795777777777778
##        3                 0                 0                 0
##        4                 0                 0                 0
##        5                 0                 0                 0
##        6                 0                 0                 0
##        7                 0                 0                 0
##        8                 0                 0                 0
##        9                 0                 0                 0
##        10                1                 0                 0
##        11                1                 0                 0
##        12                 1                 0                 0
##        13                 1                 0                 0
##        14                 0                 1                 0
##        15                 0                 1                 0
##        16                 0                 0                 1
##        17                 0                 0                 0
##        18                 0                 0                 0
##        19                 0                 0                 0
##        20                 0                 0                 0
##        21                 0                 0                 0
##        22                 0                 0                 0
##        23                 0                 0                 0
##        24                 0                 0                 0
##        25                 0                 0                 0
##        26                 0                 0                 0
##        27                 0                 0                 0
##        28                 0                 0                 0
##        29                 0                 0                 0
##        30                 0                 0                 0
##        31                 0                 0                 0
##        32                 0                 0                 0
##        33                 0                 0                 0
##        34                 0                 0                 0
##        35                 0                 0                 0
##        36                 0                 0                 0
##        37                 0                 0                 0
##        38                 0                 0                 0
##        39                 0                 0                 0
##        40                 0                 0                 0
##        41                 0                 0                 0
##        42                 0                 0                 0
##        43                 0                 0                 0
##        44                 0                 0                 0
##        45                 0                 0                 0
##        46                 0                 0                 0
##        47                 0                 0                 0
##        48                 0                 0                 0
##        49                 0                 0                 0
##        50                 0                 0                 0
##          va_acc_strg
## treesizes 0.796888888888889 0.797333333333333 0.798 0.798444444444444
##         3                 0                 0     0                 0
##         4                 0                 0     0                 0
##         5                 0                 0     0                 0
##         6                 0                 0     0                 0
##         7                 0                 0     0                 0
##         8                 0                 0     0                 0
##         9                 0                 0     0                 0
##        10                 0                 0     0                 0
##        11                 0                 0     0                 0
##        12                 0                 0     0                 0
##        13                 0                 0     0                 0
##        14                 0                 0     0                 0
##        15                 0                 0     0                 0
##        16                 0                 0     0                 0
##        17                 1                 0     0                 0
##        18                 1                 0     0                 0
##        19                 1                 0     0                 0
##        20                 1                 0     0                 0
##        21                 1                 0     0                 0
##        22                 1                 0     0                 0
##        23                 1                 0     0                 0
##         24                 1                 0     0                 0
##         25                 1                 0     0                 0
##         26                 1                 0     0                 0
##         27                 0                 1     0                 0
##         28                 0                 1     0                 0
##         29                 0                 1     0                 0
##         30                 0                 1     0                 0
##         31                 0                 0     1                 0
##         32                 0                 0     1                 0
##         33                 0                 0     1                 0
##         34                 0                 0     1                 0
##         35                 0                 0     1                 0
##         36                 0                 0     1                 0
##         37                 0                 0     1                 0
##         38                 0                 0     1                 0
##         39                 0                 0     0                 1
##         40                 0                 0     0                 0
##         41                 0                 0     0                 0
##         42                 0                 0     0                 0
##         43                 0                 0     0                 0
##         44                 0                 0     0                 0
##         45                 0                 0     0                 0
##         46                 0                 0     0                 0
##         47                 0                 0     0                 0
##         48                 0                 0     0                 0
##         49                 0                 0     0                 0
##         50                 0                 0     0                 0
##          va_acc_strg
## treesizes 0.798666666666667 0.799333333333333
##        3                 0                 0
##        4                 0                 0
##        5                 0                 0
##        6                 0                 0
##        7                 0                 0
##        8                 0                 0
##        9                 0                 0
##        10                0                 0
##        11                0                 0
##        12                0                 0
##        13                0                 0
##        14                0                 0
##        15                0                 0
##        16                0                 0
##        17                0                 0
##        18                0                 0
##        19                0                 0
##        20                0                 0
##        21                0                 0
##        22                0                 0
##        23                0                 0
##        24                0                 0
##        25                0                 0
##        26                0                 0
##        27                0                 0
##        28                0                 0
##        29                0                 0
##        30                0                 0
##        31                0                 0
##        32                0                 0
##        33                0                 0
##        34                0                 0
##        35                0                 0
##          36                  0                  0
##          37                  0                  0
##          38                  0                  0
##          39                  0                  0
##          40                  1                  0
##          41                  0                  1
##          42                  0                  1
##          43                  0                  1
##          44                  0                  1
##          45                  0                  1
##          46                  0                  1
##          47                  0                  1
##          48                  0                  1
##          49                  0                  1
##          50                  0                  1
```