Code Contributions

Clare M Maclean, Corinna M Mittmann, Nicholas Renotte

05/10/2017

# Clare M Maclean: Data Exploration Code Snipet

###CLARE MACLEAN EXPLORATION CODE   
##exploration code  
  
library(tidyverse)  
library(lubridate)  
library(glmnet)  
library(caret)  
library(plyr)  
setwd("")  
default <- read\_csv("training.csv")  
View(default)  
  
#count target distribution  
table(default$loan\_status)  
#Charged.Off Fully.Paid   
# 5670 34116   
prop.table(table(default$loan\_status))  
#Charged.Off Fully.Paid   
# 0.1425124 0.8574876   
#another class imbalance problem  
  
#look at proportion of categories in each of the variables (where applicable)  
default\_table <- subset(default, select = c("term", "grade", "sub\_grade", "emp\_length", "home\_ownership", "verification\_status", "loan\_status", "pymnt\_plan", "purpose", "addr\_state",   
 "delinq\_2yrs", "inq\_last\_6mths", "open\_acc", "pub\_rec", "total\_acc", "initial\_list\_status",  
 "out\_prncp", "out\_prncp\_inv", "collections\_12\_mths\_ex\_med", "policy\_code", "application\_type",  
 "acc\_now\_delinq", "delinq\_amnt", "pub\_rec\_bankruptcies", "tax\_liens", "hardship\_flag"))  
  
default\_table %>% lapply(table)  
  
#get rid of variables with only one category/value   
default\_noID <- default[,c(2:42)]  
default\_noID\_updated <- within(default\_noID, rm("initial\_list\_status","out\_prncp", "out\_prncp\_inv", "collections\_12\_mths\_ex\_med", "policy\_code", "application\_type",  
 "acc\_now\_delinq", "delinq\_amnt", "tax\_liens", "hardship\_flag", "pymnt\_plan"))  
#default corr plot with numeric variables  
library(corrplot)  
par(mfrow = c(1,1))  
default\_corplot <- default\_noID\_updated[sapply(default\_noID\_updated, function(x) is\_integer(x) || is\_double(x))]   
corrplot(cor(default\_corplot))   
cor(default\_corplot)  
#look at how similar the correlated columns are   
library(compare)  
compare(default\_noID\_updated["loan\_amnt"], default\_noID\_updated["funded\_amnt"])  
  
#density plots  
lapply(names(default\_noID\_updated),  
 function(i)   
 ggplot(default\_noID\_updated, aes\_string(x=i, colour = as.factor(default\_noID\_updated$loan\_status))) + geom\_density())  
  
  
#find out how many values in funded\_amnt and loan\_amnt differ from each other  
#clean\_default[!clean\_default$loan\_amnt%in%clean\_default$funded\_amnt,]  
test <- mutate(data,check\_sameness = ifelse(loan\_amnt == funded\_amnt, 1,0))  
test\_tot\_pay <- mutate(clean\_default, check\_sameness = ifelse(total\_pymnt == total\_pymnt\_inv, 1,0))  
#plot the difference  
plot(clean\_default$loan\_amnt, clean\_default$funded\_amnt)  
  
#look at clean data   
clean\_default <- read\_csv("clean\_data\_30var 2.csv")  
clean\_default <- clean\_default %>% mutate\_if(is.character, factor)  
  
#create a correlation plot of the numeric variables in the data  
library(corrplot)  
par(mfrow = c(1,1))  
default\_corplot\_clean <- clean\_default[sapply(clean\_default, function(x) is\_integer(x) || is\_double(x))]  
default\_corplot\_clean <- within(default\_corplot\_clean, rm("earliest\_cr\_line"))  
corrplot(cor(default\_corplot\_clean))   
  
#transform target loan\_status   
clean\_default$loan\_status <- ifelse(clean\_default$loan\_status == "Charged.Off",1,0)  
#try to do a vif on the variables  
clean\_default\_no\_ID <- within(clean\_default, rm("X1"))  
lm.fit <- lm(loan\_status ~.-grade, data = clean\_default)  
  
#look at the variance inflation factors in the variables  
#install.packages("car")  
library(car)  
alias(loan\_status ~., data = clean\_default)  
#need to remove grade to run vif as it is exactly colinear with sub grade  
#https://statisticalhorizons.com/multicollinearity  
"vif"(lm.fit, na.action = na.exclude)  
# GVIF Df GVIF^(1/(2\*Df))  
#X1 1.003578 1 1.001787  
#loan\_amnt 28.919163 1 5.377654  
#funded\_amnt 180.709389 1 13.442819  
#funded\_amnt\_inv 96.610384 1 9.829058  
#term 5.454204 1 2.335424  
#int\_rate 25.822923 1 5.081626  
#installment 54.023859 1 7.350092  
#sub\_grade 33.672903 34 1.053077  
#emp\_length 1.291730 10 1.012881  
#home\_ownership 1.467463 4 1.049110  
#annual\_inc 1.253627 1 1.119655  
#verification\_status 1.339877 2 1.075886  
#purpose 1.420056 13 1.013580  
#addr\_state 1.367149 49 1.003196  
#dti 1.293329 1 1.137246  
#delinq\_2yrs 1.087045 1 1.042615  
#earliest\_cr\_line 1.403015 1 1.184489  
#inq\_last\_6mths 1.108566 1 1.052884  
#open\_acc 2.035325 1 1.426648  
#pub\_rec 3.494421 1 1.869337  
#revol\_bal 1.403887 1 1.184857  
#total\_acc 2.401379 1 1.549638  
#total\_pymnt 319.574523 1 17.876647  
#total\_pymnt\_inv 135.167417 1 11.626152  
#total\_rec\_prncp 114.729746 1 10.711197  
#total\_rec\_int 24.550416 1 4.954838  
#total\_rec\_late\_fee 1.049715 1 1.024556  
#last\_pymnt\_amnt 2.143938 1 1.464219  
#pub\_rec\_bankruptcies 4.741866 3 1.296159  
  
#adjust variables to see vifs   
View(clean\_default)  
summary(lm.fit)  
lm.fit\_select <- lm(loan\_status ~.-grade -X1-funded\_amnt - total\_pymnt, data = clean\_default)  
summary(lm.fit\_select)  
"vif"(lm.fit\_select, na.action = na.exclude)  
#adjust variables to see vifs - this time with subgrade   
lm.fit\_select2 <- lm(loan\_status ~.-sub\_grade-X1-funded\_amnt-total\_pymnt-funded\_amnt-total\_pymnt, data = clean\_default)  
summary(lm.fit\_select2)  
"vif"(lm.fit\_select2, na.action = na.exclude)  
# GVIF Df GVIF^(1/(2\*Df))  
#loan\_amnt 19.510384 1 4.417056  
#term 3.310691 1 1.819530  
#int\_rate 15.116729 1 3.888024  
#installment 18.854630 1 4.342192  
#grade 15.485037 6 1.256491  
  
#look at vif if we remove installment  
lm.fit\_select3 <- lm(loan\_status ~.-grade-X1-funded\_amnt-total\_pymnt-funded\_amnt\_inv-total\_pymnt\_inv-installment, data = clean\_default)  
summary(lm.fit\_select3)  
"vif"(lm.fit\_select3, na.action = na.exclude)  
#GVIF Df GVIF^(1/(2\*Df))  
#loan\_amnt 4.959457 1 2.226984  
#term 1.902089 1 1.379162  
#int\_rate 24.162507 1 4.915537  
#sub\_grade 30.068336 34 1.051325  
#emp\_length 1.285963 10 1.012655  
  
#plot of installment vs loan amount with grade colour facet   
ggplot(clean\_default, aes(x = installment, y = loan\_amnt, colour = as.factor(grade))) + geom\_point()   
  
#read in new clean data   
clean\_just\_default <- read\_csv("clean\_justified\_data\_25var.csv")  
clean\_just\_default2 <- read\_csv("modellingdata1.csv")  
clean\_just\_default <- clean\_just\_default %>% mutate\_if(is.character, factor)

# Clare M Maclean: Modelling Code Snipet

####  
###CLARE MACLEAN MODEL CODE   
#model code  
##########################  
###lasso model code#####  
#install.packages("glmnet")  
library(glmnet)  
library(caret)  
  
# CRISP DM - create data partition row list  
set.seed(42)  
clean\_just\_default <- read\_csv("clean\_justified\_data\_25var.csv")  
clean\_just\_default <- clean\_just\_default %>% mutate\_if(is.character, factor)  
clean\_just\_default\_no\_ID <- within(clean\_just\_default, rm("member\_id"))  
train = createDataPartition(y = clean\_just\_default\_no\_ID$loan\_status, p = 0.7, list = F)  
# partition default data - remove the variable Store7  
training = clean\_just\_default\_no\_ID[train, ]  
testing = clean\_just\_default\_no\_ID[-train, ]  
  
# glmnet requires data be in separate x and y sets.   
# the predictor set, x, must be a matrix  
x = model.matrix(~ ., training[, -10])  
y = training$loan\_status  
  
  
###########################  
# Lasso Regression  
  
set.seed(42)  
# alpha = 1 specifies lasso regression  
cv.fit\_lasso = cv.glmnet(x, y, family = 'binomial', alpha = 1)  
  
# Results  
plot(cv.fit\_lasso)  
cv.fit\_lasso$lambda.min  
cv.fit\_lasso$lambda.1se  
coef(cv.fit\_lasso, s = cv.fit\_lasso$lambda.min)  
  
prediction\_lasso = predict(cv.fit\_lasso$glmnet.fit, newx = model.matrix(~ ., testing[, -10]),   
 type = "class",  
 s = cv.fit\_lasso$lambda.min)  
  
lasso\_confusion = confusionMatrix(data = prediction\_lasso, testing$loan\_status)  
#lasso\_confusion  
#Prediction Charged.Off Fully.Paid  
#Charged.Off 1627 2  
#Fully.Paid 74 10232  
  
#Accuracy : 0.9936   
#95% CI : (0.992, 0.995)  
#No Information Rate : 0.8575   
#P-Value [Acc > NIR] : < 2.2e-16   
#Kappa : 0.9735   
#Mcnemar's Test P-Value : 3.816e-16   
  
#Sensitivity : 0.9565   
#Specificity : 0.9998   
#Pos Pred Value : 0.9988   
#Neg Pred Value : 0.9928   
#Prevalence : 0.1425   
#Detection Rate : 0.1363   
#Detection Prevalence : 0.1365   
#Balanced Accuracy : 0.9782   
  
#'Positive' Class : Charged.Off  
  
#find the AUC  
library(ROCR)  
prediction\_lasso\_prob = predict(cv.fit\_lasso$glmnet.fit, newx = model.matrix(~ ., testing[, -10]),   
 type = "response",  
 s = cv.fit\_lasso$lambda.min)  
  
testing$probability = predict(cv.fit\_lasso$glmnet.fit, newx = model.matrix(~ ., testing[, -10]),   
 type = "response",  
 s = cv.fit\_lasso$lambda.min)  
testing\_prediction = prediction(testing$probability, testing$loan\_status)  
test\_tpr\_fpr = performance(testing\_prediction, "tpr","fpr")  
data.test.auc = performance(testing\_prediction, "auc")  
auc = unlist(slot(data.test.auc, "y.values"))  
auc  
#[1] 0.9928011  
  
#predict with validation set   
validation\_default <- read\_csv("cleaned\_validation.csv")  
validation\_default <- validation\_default %>% mutate\_if(is.character, factor)  
#there's missing values in the data set   
names(which(colSums(is.na(validation\_default))>0))  
[1] "annual\_inc" "delinq\_2yrs" "earliest\_cr\_line" "inq\_last\_6mths"   
[5] "open\_acc" "pub\_rec" "total\_acc"   
names(which(colSums(is.na(validation\_default\_test))>0))  
#impute data to reduct missing values and remove earliest\_cr\_line variable is it's a date   
library(randomForest)  
validation\_default <- na.roughfix(validation\_default[,-15])  
#this doesn't work  
validation\_default$prediction = predict(cv.fit\_lasso$glmnet.fit, newx = model.matrix(~ ., validation\_default[, -1]),   
 type = "class",  
 s = cv.fit\_lasso$lambda.min)  
  
##################################  
##################################  
#svm model   
  
#load e1071 if not loaded  
library(e1071)  
  
#data  
##data  
clean\_just\_default <- read\_csv("clean\_justified\_data\_25var.csv")  
clean\_just\_default <- clean\_just\_default %>% mutate\_if(is.character, factor)  
clean\_just\_default\_no\_date <- within(clean\_just\_default, rm("earliest\_cr\_line"))  
set.seed(42)  
clean\_just\_default\_no\_date[,"train"] <- ifelse(runif(nrow(clean\_just\_default\_no\_date))<0.8,1,0)  
#write dataframe to disk to check  
#write.csv(clean\_just\_default\_no\_date,"clean\_just\_default.csv")  
#separate training and test sets  
trainset <- clean\_just\_default\_no\_date[clean\_just\_default\_no\_date$train==1,]  
testset <- clean\_just\_default\_no\_date[clean\_just\_default\_no\_date$train==0,]  
trainColNum <- grep("train",names(trainset))  
typeColNum <- grep("loan\_status",names(clean\_just\_default\_no\_date))  
trainset <- trainset[,-trainColNum]  
testset <- testset[,-trainColNum]  
  
#Try linear kernel...  
svm\_model<- svm(loan\_status~ .-member\_id, data=trainset, method="C-classification", kernel="linear", probability = TRUE)  
pred\_train <- predict(svm\_model,trainset)  
mean(pred\_train==trainset$loan\_status)  
#[1] 0.9931981  
pred\_test <- predict(svm\_model,testset)  
mean(pred\_test==testset$loan\_status)  
#[1] 0.992279  
svm\_confusion = confusionMatrix(data = pred\_test, testset$loan\_status)  
  
#Prediction Charged.Off Fully.Paid  
#Charged.Off 1075 0  
#Fully.Paid 62 6893  
  
#Accuracy : 0.9923   
##95% CI : (0.9901, 0.9941)  
#No Information Rate : 0.8584   
#P-Value [Acc > NIR] : < 2.2e-16   
  
#Kappa : 0.9675   
#Mcnemar's Test P-Value : 9.408e-15   
  
#Sensitivity : 0.9455   
#Specificity : 1.0000   
#Pos Pred Value : 1.0000   
#Neg Pred Value : 0.9911   
#Prevalence : 0.1416   
#Detection Rate : 0.1339   
#Detection Prevalence : 0.1339   
#Balanced Accuracy : 0.9727   
  
#'Positive' Class : Charged.Off   
  
#predict on validation   
#add 50th level to addr state as it currently on has 49  
validation\_default$addr\_state <- factor(validation\_default$addr\_state, levels = c("AK", "AL", "AR", "AZ", "CA", "CO", "CT", "DC", "DE", "FL", "GA", "HI", "IA", "ID", "IL",  
 "IN", "KS", "KY", "LA", "MA", "MD", "ME", "MI", "MN", "MO", "MS", "MT", "NC", "NE", "NH",  
 "NJ", "NM", "NV", "NY", "OH", "OK", "OR", "PA", "RI", "SC", "SD", "TN", "TX", "UT", "VA", "VT", "WA", "WI", "WV", "WY"))  
  
#create a prediction   
validation\_test\_svm <- validation\_default  
validation\_test\_svm$prediction <- predict(svm\_model,validation\_test\_svm)  
validation\_test\_svm$probability <- attr((predict(svm\_model, validation\_test\_svm, probability = TRUE)), "probabilities")  
table(validation\_test\_svm$prediction)  
  
validation\_svm\_submission <- subset(validation\_test\_svm, select = c("member\_id", "prediction", "probability"))  
write.csv(validation\_svm\_submission,file="validation\_svm.csv")  
#Charged.Off Fully.Paid   
#712 2037  
  
######################################  
#neural net model   
  
library(nnet)  
library(NeuralNetTools)  
library(caret)  
###create a test and train set   
samp = sample(1:length(clean\_just\_default\_no\_date$loan\_status), 100)  
  
train = clean\_just\_default\_no\_date[-samp, ]  
test = clean\_just\_default\_no\_date[samp, ]  
  
nn\_fit = nnet(loan\_status ~ .-member\_id, train, size=8)  
plotnet(nn\_fit, nid = F)  
  
nnet\_confusion =confusionMatrix(predict(nn\_fit, test, type="class"), test$loan\_status)  
  
#nnet\_confusion  
#Prediction Charged.Off Fully.Paid  
#Charged.Off 15 0  
#Fully.Paid 1 84  
  
#Accuracy : 0.99   
#95% CI : (0.9455, 0.9997)  
#No Information Rate : 0.84   
#P-Value [Acc > NIR] : 5.37e-07   
  
#Kappa : 0.9618   
#Mcnemar's Test P-Value : 1   
  
#Sensitivity : 0.9375   
#Specificity : 1.0000   
#Pos Pred Value : 1.0000   
#Neg Pred Value : 0.9882   
#Prevalence : 0.1600   
#Detection Rate : 0.1500   
#Detection Prevalence : 0.1500   
#Balanced Accuracy : 0.9688   
  
#'Positive' Class : Charged.Off   
  
#create a prediction for validation test   
validation\_test\_nnet <- validation\_test  
nnet\_val=table(predict(nn\_fit, validation\_test\_nnet, type="class"))  
nnet\_val  
validation\_test\_nnet$probability <- predict(nn\_fit, validation\_test\_nnet, type="raw")  
validation\_nnet\_submission <- subset(validation\_test\_nnet, select = c("member\_id", "probability"))  
write.csv(validation\_nnet\_submission,file="validation\_nnet.csv")

# Corinna M Mittmann: Code Snipet

###########################################################  
### DATA ALGOITHMS AND MEANING  
### ASSESSMENT 2: Classification   
### LOAN DEFAULT CHALLENGE  
### Corinna Maher Mittmann  
### SEP 2017  
### DOCUMENT Purpose: Clean Data + Model CART tree & Logistic Regression  
###########################################################  
Original\_Data\_Location <- "/Users/corinnamm/Documents/GitHub/DAM\_Assignment2\_CM/03\_DAM\_Spring2017\_Assignment\_2/"  
working\_file\_path <- "/Users/corinnamm/Dropbox/02\_Working\_data\_folder/"  
code\_file\_path <- "/Users/corinnamm/Documents/GitHub/DAM\_Assignment2\_CM/01\_Code/"  
wd\_path <- "/Users/corinnamm/Documents/GitHub/DAM\_Assignment2\_CM/"  
###  
library(randomForest)  
library(glmnet)  
library(ggplot2)  
library(tidyverse)  
library(dplyr)  
library(lubridate)  
library(stats)  
require(caret)  
require(readr)  
require(e1071)  
library(gbm)  
library(caret)  
library(ISLR)  
library(corrplot)  
library(nnet)  
library(glmnet)  
##  
source(paste0(code\_file\_path,"/99\_general\_functions\_DAM\_Corinna.R"))  
  
require(dplyr)  
## === START CLEAN  
default <- read.csv("/Users/corinnamm/Documents/GitHub/DAM\_Assignment2\_CM/03\_DAM\_Spring2017\_Assignment\_2/training.csv", header = T)  
# #### ==== Pre-clean exploration  
#   
# ###   
# default <- read.csv(paste0(Original\_Data\_Location,"training.csv"), header = T)  
# data\_dict <- read.csv(paste0(Original\_Data\_Location,"data\_dictionary.csv"), header = T)  
# data\_dict  
#   
# check\_for\_missing(default)  
#   
# # for (i in 1:(length(names(default)))){  
# # print(i)  
# # s <- qplot(x = default[,i],y=default$loan\_amnt, geom = "point", col = default$loan\_status)   
# # plot(s)  
# # }  
#   
# # use custom function from function script to look at densities and descriptive stats  
# descriptive\_info(default, "loan\_status")  
  
review\_data\_func(default,"Charged.Off", "Fully.Paid")  
  
# remove columns with 1 level  
mylist\_levels <- default %>%  
 summarise\_each(funs(n\_distinct)) %>%  
 tidyr::gather("LoanStatNew", "level") %>% filter(level != 1) %>% select(LoanStatNew)  
cols\_to\_keep <- purrr::as\_vector(mylist\_levels, .type = "character")  
typeof(cols\_to\_keep)  
typeof(names(default))  
default2 = default[,cols\_to\_keep]  
### ======== may need to review this decision to remove as 0 and NA values only - ? does distinction btw 0 and NA have value?  
#collections # n.a  
unique(default2$collections\_12\_mths\_ex\_med)  
table(default2$collections\_12\_mths\_ex\_med)  
default2 <- select(default2, -collections\_12\_mths\_ex\_med)  
# remove tax liens  
unique(default2$tax\_liens)  
table(default2$tax\_liens)  
default2 <- select(default2, -tax\_liens)  
data <- default2  
# make interest rate numeric  
head(unique(data$int\_rate))  
data$int\_rate <- gsub("X", "", data$int\_rate)  
# check that "." is the last character for every level  
table((substr(data$int\_rate, nchar(data$int\_rate),nchar(data$int\_rate))=="."))[[1]]==nrow(data)  
n\_distinct((substr(data$int\_rate, 1,nchar(data$int\_rate)-1))) == n\_distinct(data$int\_rate)  
data$int\_rate <- substr(data$int\_rate, 1,nchar(data$int\_rate)-1)  
# write.csv(data, paste0(working\_file\_path,"clean\_data\_v2.csv"),row.names = F)  
# data<- read.csv(paste0(working\_file\_path,"clean\_data\_v2.csv"))  
str(data)  
# Make emp\_length consistent  
# change missing format  
data$emp\_length <- gsub("\\.","",data$emp\_length)  
data$emp\_length <- gsub("X", "", data$emp\_length)  
head(data$emp\_length)  
# deal with na  
table(data$emp\_length == "na")  
data$emp\_length <- ifelse(data$emp\_length == "na","unknown",data$emp\_length) #unknown for unknown values we can change this to "unknown"  
table(data$emp\_length == "unknown")  
# keep only the value and remove "years" year, ..years etc units  
data$emp\_length <- gsub("year","",data$emp\_length)  
data$emp\_length <- gsub("s","",data$emp\_length)  
unique(data$emp\_length)  
qplot(data$emp\_length, fill = "identity")  
# write.csv(data, paste0(working\_file\_path,"clean\_data\_v3.csv"),row.names = F)  
# data<- read.csv(paste0(working\_file\_path,"clean\_data\_v3.csv"))  
# Make term consistent  
# change missing format  
data$term <- gsub("\\.","",data$term)  
head(data$term)  
# deal with na  
table(data$term == "na")  
# no na values established  
# data$term <- ifelse(data$term == "na","unknown",data$term)  
# table(data$term == "unknown")  
# keep only the value and remove "years" year, ..years etc units  
data$term <- gsub("month","",data$term)  
data$term <- gsub("s","",data$term)  
head(data$term)  
unique(data$term)  
qplot(data$term, fill = "identity")  
### --------------------- Make verification\_status consistent  
# change missing format  
unique(data$verification\_status)  
data$verification\_status <- gsub("\\.","\_",data$verification\_status)  
qplot(data$verification\_status, fill = "identity")  
# pub rec bankruptcies # NA  
unique(data$pub\_rec\_bankruptcies)  
# deal with na  
table(is.na(data$pub\_rec\_bankruptcies))  
data$pub\_rec\_bankruptcies <- ifelse(is.na(data$pub\_rec\_bankruptcies),"unknown",data$pub\_rec\_bankruptcies)  
table(data$pub\_rec\_bankruptcies == "unknown")  
data$pub\_rec\_bankruptcies <- as.factor(data$pub\_rec\_bankruptcies)  
qplot(data$pub\_rec\_bankruptcies, fill = "identity")  
  
table(data$pub\_rec\_bankruptcies)  
### --------------------- Refomat the date column  
# earliest cred line convert to floor date  
data$earliest\_cr\_line = gsub("\\.","-",data$earliest\_cr\_line)  
data <- mutate(data, earliest\_cr\_line = as.Date(paste0("01-",earliest\_cr\_line), format = "%d-%b-%Y"))  
# test <- dplyr::sample\_n(as.tibble(data$earliest\_cr\_line), 10)  
unique(data$earliest\_cr\_line)  
# qplot(data$earliest\_cr\_line, data$total\_rec\_int, colour = data$inq\_last\_6mths)  
qplot(data$earliest\_cr\_line, fill = "identity") ## + facet\_wrap(data$loan\_status)  
## we may or may not want to convert the date into continuous time series  
## it may be advantageous to keep the time format as is  
####  
## ==== Please note that I am changing the format of this column to days since the publish date of the data.  
# These files contain complete loan data for all loans issued through the time period stated,   
# including the current loan status (Current, Late, Fully Paid, etc.) and latest payment information.   
# The file containing loan data through the "present" (note the data was uploaded a year ago to Kaggle by Wendy Kan)  
# contains complete loan data for all loans issued through the previous completed calendar quarter.  
row.names(data) <- data$member\_id  
# max(as.Date(new\_data$earliest\_cr\_line))  
max(as.Date(old\_data$earliest\_cr\_line))  
max(as.Date(data$earliest\_cr\_line))  
## ==== earliest credit line was not seen to be important for the tree model  
####  
backup\_data <- data  
data <- mutate(data, earliest\_cr\_line = as.Date("2017-09-25") - as.Date(earliest\_cr\_line))  
head(data$earliest\_cr\_line)  
data$earliest\_cr\_line = as.double(data$earliest\_cr\_line)  
names(data)  
data <- select(data, -total\_pymnt\_inv, -installment, -funded\_amnt\_inv, -grade, -total\_rec\_prncp)  
# write.csv(data, paste0(Original\_Data\_Location,"version2\_training\_26\_col"),row.names = F)  
# data2 <- data %>% mutate(grade = substr(sub\_grade, 1,1)) %>% select(-sub\_grade) ### to convert subgrade to grade  
data <- read\_csv(paste0(Original\_Data\_Location,"version2\_training\_26col.csv"))  
  
#####################  
# Validation set  
#####################  
default <- read.csv("/Users/corinnamm/Documents/GitHub/DAM\_Assignment2\_CM/03\_DAM\_Spring2017\_Assignment\_2/validation.csv", header = T)  
str(default)  
## === START CLEAN  
# remove columns not in model data  
targetcolnum <- grep("loan\_status",names(old\_data))  
# is\_there\_a\_target <- grep("loan\_status",names(default)) #no  
cols\_to\_keep <- names(data[,-targetcolnum]) # 10 is target (loan\_status)  
typeof(cols\_to\_keep)  
names(default)  
default2 = select(default, paste0(cols\_to\_keep))  
names(default2)  
### ======== may need to review this decision to remove as 0 and NA values only - ? does distinction btw 0 and NA have value?  
#collections # n.a  
unique(default2$collections\_12\_mths\_ex\_med)  
table(default2$collections\_12\_mths\_ex\_med)  
default2 <- select(default2, -collections\_12\_mths\_ex\_med)  
# remove tax liens  
unique(default2$tax\_liens)  
table(default2$tax\_liens)  
default2 <- select(default2, -tax\_liens)  
data <- default2  
# make interest rate numeric  
head(unique(data$int\_rate))  
data$int\_rate <- gsub("X", "", data$int\_rate)  
# check that "." is the last character for every level  
table((substr(data$int\_rate, nchar(data$int\_rate),nchar(data$int\_rate))=="."))[[1]]==nrow(data)  
n\_distinct((substr(data$int\_rate, 1,nchar(data$int\_rate)-1))) == n\_distinct(data$int\_rate)  
data$int\_rate <- substr(data$int\_rate, 1,nchar(data$int\_rate)-1)  
# write.csv(data, paste0(working\_file\_path,"clean\_data\_v2.csv"),row.names = F)  
# data<- read.csv(paste0(working\_file\_path,"clean\_data\_v2.csv"))  
str(data)  
# Make emp\_length consistent  
# change missing format  
data$emp\_length <- gsub("\\.","",data$emp\_length)  
data$emp\_length <- gsub("X", "", data$emp\_length)  
head(data$emp\_length)  
# deal with na  
table(data$emp\_length == "na")  
data$emp\_length <- ifelse(data$emp\_length == "na","unknown",data$emp\_length) #unknown for unknown values we can change this to "unknown"  
table(data$emp\_length == "unknown")  
# keep only the value and remove "years" year, ..years etc units  
data$emp\_length <- gsub("year","",data$emp\_length)  
data$emp\_length <- gsub("s","",data$emp\_length)  
unique(data$emp\_length)  
qplot(data$emp\_length, fill = "identity")  
# write.csv(data, paste0(working\_file\_path,"clean\_data\_v3.csv"),row.names = F)  
# data<- read.csv(paste0(working\_file\_path,"clean\_data\_v3.csv"))  
# Make term consistent  
# change missing format  
data$term <- gsub("\\.","",data$term)  
head(data$term)  
# deal with na  
table(data$term == "na")  
# no na values established  
# data$term <- ifelse(data$term == "na","unknown",data$term)  
# table(data$term == "unknown")  
# keep only the value and remove "years" year, ..years etc units  
data$term <- gsub("month","",data$term)  
data$term <- gsub("s","",data$term)  
head(data$term)  
unique(data$term)  
qplot(data$term, fill = "identity")  
### --------------------- Make verification\_status consistent  
# change missing format  
unique(data$verification\_status)  
data$verification\_status <- gsub("\\.","\_",data$verification\_status)  
qplot(data$verification\_status, fill = "identity")  
# pub rec bankruptcies # NA  
unique(data$pub\_rec\_bankruptcies)  
# deal with na  
table(is.na(data$pub\_rec\_bankruptcies))  
data$pub\_rec\_bankruptcies <- ifelse(is.na(data$pub\_rec\_bankruptcies),"unknown",data$pub\_rec\_bankruptcies)  
table(data$pub\_rec\_bankruptcies == "unknown")  
data$pub\_rec\_bankruptcies <- as.factor(data$pub\_rec\_bankruptcies)  
qplot(data$pub\_rec\_bankruptcies, fill = "identity")  
table(data$pub\_rec\_bankruptcies)  
### --------------------- Refomat the date column  
# earliest cred line convert to floor date  
data$earliest\_cr\_line = gsub("\\.","-",data$earliest\_cr\_line)  
data <- mutate(data, earliest\_cr\_line = as.Date(paste0("01-",earliest\_cr\_line), format = "%d-%b-%Y"))  
# test <- dplyr::sample\_n(as.tibble(data$earliest\_cr\_line), 10)  
unique(data$earliest\_cr\_line)  
# qplot(data$earliest\_cr\_line, data$total\_rec\_int, colour = data$inq\_last\_6mths)  
qplot(data$earliest\_cr\_line, fill = "identity") ## + facet\_wrap(data$loan\_status)  
## we may or may not want to convert the date into continuous time series  
## it may be advantageous to keep the time format as is  
####  
## ==== Please note that I am changing the format of this column to days since the publish date of the data.  
# These files contain complete loan data for all loans issued through the time period stated,   
# including the current loan status (Current, Late, Fully Paid, etc.) and latest payment information.   
# The file containing loan data through the "present" (note the data was uploaded a year ago to Kaggle by Wendy Kan)  
# contains complete loan data for all loans issued through the previous completed calendar quarter.  
row.names(data) <- data$member\_id  
# max(as.Date(new\_data$earliest\_cr\_line))  
max(as.Date(old\_data$earliest\_cr\_line))  
max(as.Date(data$earliest\_cr\_line))  
## == note == earliest credit line was not seen to be important for the tree model - as a time series consider the value of use  
####  
backup\_data <- data  
  
### PARTITION OUT PROBLEM ROWS  
problem\_rows <- data.frame(member\_id = NA)  
tbl\_df(data)  
problem\_rows\_func(problem\_rows = problem\_rows, new\_data = tbl\_df(data))  
class(problem\_rows)  
problem\_row\_id <- distinct(tbl\_df(problem\_rows))  
problem\_rows <- tbl\_df(problem\_rows)  
problem\_rows <- distinct(problem\_rows)  
problem\_rows <- left\_join(problem\_rows,data)  
data2 <- mutate(data, problem = if\_else(member\_id %in% problem\_row\_id[[1]], 1,0))  
table(data2$problem)  
# problem\_rows\_complete  
back\_up <- data2  
## remove problem rows from data  
  
data <- data2 %>% filter(problem == 0 ) %>% select(-problem)  
  
data <- mutate(data, earliest\_cr\_line = as.Date("2017-09-25") - as.Date(earliest\_cr\_line))  
head(data$earliest\_cr\_line)  
data$earliest\_cr\_line = as.double(data$earliest\_cr\_line)  
names(data)  
# write\_csv(data, paste0(Original\_Data\_Location, "version2\_validation\_noNA.csv"))  
# data <- select(data, -total\_pymnt\_inv, -installment, -funded\_amnt\_inv, -grade, -total\_rec\_prncp)  
# write\_csv(data, paste0(Original\_Data\_Location, "version2\_validation\_25col\_noNA.csv"))  
# data2 <- data %>% mutate(grade = substr(sub\_grade, 1,1)) %>% select(-sub\_grade) ### to convert subgrade to grade  
  
NAdata <- data2 %>% filter(problem == 1 ) %>% select(-problem)  
# write\_csv(NAdata, paste0(Original\_Data\_Location, "version2\_validation\_all\_columns\_NA.csv"))  
NAdata <- select(NAdata, -total\_pymnt\_inv, -installment, -funded\_amnt\_inv, -grade, -total\_rec\_prncp)  
# write\_csv(NAdata, paste0(Original\_Data\_Location, "version2\_validation\_25col\_NA.csv"))  
  
data <- bind\_rows(data,NAdata)  
# write\_csv(data, paste0(Original\_Data\_Location, "version2\_validation\_full\_25col.csv"))  
  
# write\_csv(data, paste0(Original\_Data\_Location, "version2\_validation\_full.csv"))  
  
## ===== END CLEANING  
  
## Data Understanding part 2 ## load data  
str(data)  
table(data$loan\_status)  
## make target binary  
data <- mutate(data, loan\_status = if\_else(loan\_status == "Charged.Off",1,0))  
table(data$loan\_status)  
## remove ID  
data <- select(data, -X)  
  
data = as.tibble(data)  
int\_data <- select(data, -term, -grade, -sub\_grade, -home\_ownership,   
 -verification\_status, -purpose, -addr\_state,  
 -pub\_rec\_bankruptcies, -earliest\_cr\_line) # loan\_status  
int\_data$emp\_length = gsub("X","", int\_data$emp\_length)  
int\_data$emp\_length = ifelse(int\_data$emp\_length == "UK", 0, int\_data$emp\_length)  
int\_data$emp\_length = as.numeric(int\_data$emp\_length)  
# corrplot::corrplot(int\_data, method = "shade")  
str(int\_data)  
default\_corplot <- int\_data[sapply(int\_data, function(x) is\_integer(x) || is\_double(x))]   
par(mfrow = c(1,1))  
corrplot::corrplot(cor(default\_corplot), method = "shade")   
  
## Clean up workspace  
data <- backup\_data  
rm(int\_data, default, default2, default\_corplot)  
##################  
## MODELLING  
##################  
  
# old\_data <- read.csv(paste0(working\_file\_path,"clean\_justified\_data\_25var.csv"))  
# default <- read.csv(paste0(Original\_Data\_Location,".csv"), header = T)  
# write\_csv(data, paste0(working\_file\_path,"v2\_clean\_justified\_data\_25var"))  
# data <- read\_csv(paste0(working\_file\_path,"v2\_clean\_justified\_data\_25var.csv"))  
data <- read\_csv(paste0(Original\_Data\_Location,"version2\_training\_26col.csv"))  
# Renove ID  
row.names(data) <- data$member\_id  
data <- select(data, -member\_id)  
str(data)  
data$loan\_status <- as.factor(data$loan\_status)  
data<- data %>% mutate( loan\_amnt = as.numeric(loan\_amnt),  
 funded\_amnt = as.numeric(funded\_amnt),  
 term = as.factor(term),  
 int\_rate = as.numeric(int\_rate),  
 sub\_grade = as.factor(sub\_grade), # = substr(sub\_grade, 1,1),  
 emp\_length = as.factor(emp\_length),  
 home\_ownership = as.factor(home\_ownership),  
 annual\_inc = as.numeric(annual\_inc),  
 verification\_status = as.factor(verification\_status),  
 loan\_status = as.factor(loan\_status),  
 purpose = as.factor(purpose),  
 addr\_state = as.factor(addr\_state),  
 dti = as.numeric(dti),  
 delinq\_2yrs = as.numeric(delinq\_2yrs),  
 earliest\_cr\_line = as.numeric(earliest\_cr\_line),  
 inq\_last\_6mths = as.numeric(inq\_last\_6mths),  
 open\_acc = as.numeric(open\_acc),  
 pub\_rec = as.numeric(pub\_rec),  
 revol\_bal = as.numeric(revol\_bal),  
 total\_acc = as.numeric(total\_acc),  
 total\_pymnt = as.numeric(total\_pymnt),  
 total\_rec\_int = as.numeric(total\_rec\_int),  
 total\_rec\_late\_fee = as.numeric(total\_rec\_late\_fee),  
 last\_pymnt\_amnt = as.numeric(last\_pymnt\_amnt),  
 pub\_rec\_bankruptcies = as.factor(pub\_rec\_bankruptcies))  
str(data)  
data <- tbl\_df(data)  
class(data)  
###############  
## SUB SAMPLING FOR UNBALANCED CLASS  
#to improve the model subsampling is also performed on an alternate dataset  
###############  
## Partitioning to train the glmnet model  
#dummy factors  
data\_backup <- data  
#######################################  
set.seed(42)  
train = createDataPartition(y = data$loan\_status, p = 0.7, list = F)  
# partition default data  
training = data[train, ]  
training <- mutate(training, loan\_status = if\_else(loan\_status == "Charged.Off",1,0))  
dmy <- dummyVars(" ~ .", data = training, fullRank = F)  
glmDF <- as.data.frame(predict(dmy,training))  
glmDF <- mutate(glmDF, loan\_status = if\_else(loan\_status == 1,"Charged.Off","Fully.Paid"))  
training <- mutate(training, loan\_status = if\_else(loan\_status == 1,"Charged.Off","Fully.Paid"))  
testing = data[-train, ]  
testing <- mutate(testing, loan\_status = if\_else(loan\_status == "Charged.Off",1,0))  
dmy <- dummyVars(" ~ .", data = testing, fullRank = F)  
testDF <- as.data.frame(predict(dmy,testing))  
testDF <- mutate(testDF, loan\_status = if\_else(loan\_status == 1,"Charged.Off","Fully.Paid"))  
testing <- mutate(testing, loan\_status = if\_else(loan\_status == 1,"Charged.Off","Fully.Paid"))  
table(testing$loan\_status)  
table(training$loan\_status)  
## upsampled dataframe  
# imbal\_training$ID <- rownames(imbal\_training)  
set.seed(42)  
x <- select(glmDF, -loan\_status)  
set.seed(42)  
up\_glmDF <- upSample(x = x,  
 y = as.factor(glmDF$loan\_status), list = F, yname = "loan\_status")  
set.seed(42)  
down\_training <- downSample(x = x,  
 y = training$loan\_status, list = F, yname = "loan\_status")  
# Even class balance will present problems for interpretability  
#  
up\_training <- up\_glmDF  
table(up\_training$loan\_status)  
# head(up\_training$x)  
class(up\_training)  
str(up\_training)  
backup\_up\_train\_up <- up\_training  
backup\_up\_train\_dn <- down\_training  
targetcol <- grep("loan\_status", names(up\_training))  
set.seed(42)  
  
## BAGGED CART WITH SMALLER DATA SET With either GRADE/SUBGRADE used  
ctrl <- trainControl(method = "repeatedcv", repeats = 5,  
 classProbs = TRUE,  
 summaryFunction = twoClassSummary)  
  
set.seed(42)  
upmod\_inside <- train(x = up\_training[,-ncol(up\_training)],y=up\_training[,ncol(up\_training)],   
 method = "treebag",  
 nbagg = 50,  
 metric = "ROC",  
 trControl = ctrl)  
  
## cannot have binary level labels for the caret packet will throw error as 0 and 1 levels are not valid format  
  
## Evaluation  
  
str(upmod\_inside)  
upmod\_inside$results  
## ===== confusion matrix  
# table(repurchase.rf$test$predicted,testing$Target)  
te <- grep("loan\_status",names((testDF)))  
predictions\_upmod <- predict(upmod\_inside$finalModel,testDF[,-te],type="class")  
probability\_upmod <- predict(upmod\_inside$finalModel,testing[,-te],type="prob")  
# cbind(testing, predictions\_upmod)  
# cbind(testing,probability\_upmod)  
# test\_validationfile <- testing  
wow <- confusionMatrix(predictions\_upmod,testDF[,te])  
wow  
wow$byClass  
  
## ==== ROCR  
typeof(predictions\_upmod)  
tesi<- testDF  
  
tesi$pred <- predictions\_upmod  
binary<- mutate(tesi, pred = ifelse(pred == "Charged.Off", 1,0), loan\_status = ifelse(loan\_status == "Charged.Off", 1,0))   
testing\_prediction = prediction(binary$pred, binary$loan\_status)  
data.test.auc <- ROCR::performance(testing\_prediction, "auc")  
auc = unlist(slot(data.test.auc, "y.values"))  
#  
  
# model\_evaluation(testDF,training, upmod\_inside$finalModel, target = "loan\_status")  
  
impmod<- varImp(upmod\_inside$finalModel)  
plot(impmod)  
### =========  
  
##############################  
## predict the validation set  
new\_data <- read\_csv(paste0(Original\_Data\_Location,"version2\_validation\_full\_25col.csv"))  
  
## ====== prep submission prediction data  
new\_data<- new\_data %>% mutate( loan\_amnt = as.numeric(loan\_amnt),  
 funded\_amnt = as.numeric(funded\_amnt),  
 term = as.factor(term),  
 int\_rate = as.numeric(int\_rate),  
 grade = as.factor(grade),  
 emp\_length = as.factor(emp\_length),  
 home\_ownership = as.factor(home\_ownership),  
 annual\_inc = as.numeric(annual\_inc),  
 verification\_status = as.factor(verification\_status),  
 purpose = as.factor(purpose),  
 addr\_state = as.factor(addr\_state),  
 dti = as.numeric(dti),  
 delinq\_2yrs = as.numeric(delinq\_2yrs),  
 earliest\_cr\_line = ymd(earliest\_cr\_line),  
 inq\_last\_6mths = as.numeric(inq\_last\_6mths),  
 open\_acc = as.numeric(open\_acc),  
 pub\_rec = as.numeric(pub\_rec),  
 revol\_bal = as.numeric(revol\_bal),  
 total\_acc = as.numeric(total\_acc),  
 total\_pymnt = as.numeric(total\_pymnt),  
 total\_rec\_int = as.numeric(total\_rec\_int),  
 total\_rec\_late\_fee = as.numeric(total\_rec\_late\_fee),  
 last\_pymnt\_amnt = as.numeric(last\_pymnt\_amnt),  
 pub\_rec\_bankruptcies = as.factor(pub\_rec\_bankruptcies))  
  
str(new\_data)  
row.names(new\_data) <- new\_data$member\_id  
id <- grep("member\_id",names((new\_data)))  
### If the dummy variables names are not the same - to insert the . for the dummy names below  
# colnames(new\_data)[4] <- paste0(colnames(new\_data)[4],".")  
# colnames(new\_data)[6] <- paste0(colnames(new\_data)[6],".")  
# colnames(new\_data)[7] <- paste0(colnames(new\_data)[7],".")  
# colnames(new\_data)[8] <- paste0(colnames(new\_data)[8],".")  
# colnames(new\_data)[10] <- paste0(colnames(new\_data)[10],".")  
# colnames(new\_data)[11] <- paste0(colnames(new\_data)[11],".")  
# colnames(new\_data)[12] <- paste0(colnames(new\_data)[12],".")  
# colnames(new\_data)[25] <- paste0(colnames(new\_data)[25],".")  
## ====================  
# testing2 <- mutate(new\_data, loan\_status = if\_else(loan\_status == "Charged.Off",1,0))  
dmy <- dummyVars(" ~ .", data = new\_data[,-id], fullRank = F)  
testDF2 <- as.data.frame(predict(dmy,new\_data[,-id]))  
# reorder names(testDF)  
testDF3 <- testDF2  
testDF3$addr\_state.ME <- 0  
# names(testDF3) <- (names(testDF))  
  
# testDF2 <- mutate(testDF2, loan\_status = if\_else(loan\_status == 1,"Charged.Off","Fully.Paid"))  
# testing2 <- mutate(testing2, loan\_status = if\_else(loan\_status == 1,"Charged.Off","Fully.Paid"))  
  
### ============== New Submission  
  
loan\_status\_pred <- predict(upmod\_inside$finalModel,testDF2[,-id],type="class")  
prob\_upmod <- predict(upmod\_inside$finalModel,testDF3,type="prob")  
  
new\_pred <- cbind(new\_data,prob\_upmod)  
names(new\_pred)  
new\_pred <- select(new\_pred, member\_id, Charged.Off)  
names(new\_pred) <- c("member\_id", "probability")  
write\_csv(new\_pred, paste0(Original\_Data\_Location, "bag\_cart\_upsamp\_5rptcv\_50nbagg\_submission2.csv"))  
  
##############################  
## predict the validation set  
new\_data <- read\_csv(paste0(Original\_Data\_Location,"version2\_validation\_full\_25col.csv"))  
  
## ====== prep submission prediction data for tree  
new\_data<- new\_data %>% mutate( loan\_amnt = as.numeric(loan\_amnt),  
 funded\_amnt = as.numeric(funded\_amnt),  
 term = as.factor(term),  
 int\_rate = as.numeric(int\_rate),  
 grade = as.factor(grade),  
 emp\_length = as.factor(emp\_length),  
 home\_ownership = as.factor(home\_ownership),  
 annual\_inc = as.numeric(annual\_inc),  
 verification\_status = as.factor(verification\_status),  
 purpose = as.factor(purpose),  
 addr\_state = as.factor(addr\_state),  
 dti = as.numeric(dti),  
 delinq\_2yrs = as.numeric(delinq\_2yrs),  
 earliest\_cr\_line = ymd(earliest\_cr\_line),  
 inq\_last\_6mths = as.numeric(inq\_last\_6mths),  
 open\_acc = as.numeric(open\_acc),  
 pub\_rec = as.numeric(pub\_rec),  
 revol\_bal = as.numeric(revol\_bal),  
 total\_acc = as.numeric(total\_acc),  
 total\_pymnt = as.numeric(total\_pymnt),  
 total\_rec\_int = as.numeric(total\_rec\_int),  
 total\_rec\_late\_fee = as.numeric(total\_rec\_late\_fee),  
 last\_pymnt\_amnt = as.numeric(last\_pymnt\_amnt),  
 pub\_rec\_bankruptcies = as.factor(pub\_rec\_bankruptcies))  
  
str(new\_data)  
row.names(new\_data) <- new\_data$member\_id  
id <- grep("member\_id",names((new\_data)))  
colnames(new\_data)[4] <- paste0(colnames(new\_data)[4],".")  
colnames(new\_data)[6] <- paste0(colnames(new\_data)[6],".")  
colnames(new\_data)[7] <- paste0(colnames(new\_data)[7],".")  
colnames(new\_data)[8] <- paste0(colnames(new\_data)[8],".")  
colnames(new\_data)[10] <- paste0(colnames(new\_data)[10],".")  
colnames(new\_data)[11] <- paste0(colnames(new\_data)[11],".")  
colnames(new\_data)[12] <- paste0(colnames(new\_data)[12],".")  
colnames(new\_data)[25] <- paste0(colnames(new\_data)[25],".")  
## ====================  
# testing2 <- mutate(new\_data, loan\_status = if\_else(loan\_status == "Charged.Off",1,0))  
dmy <- dummyVars(" ~ .", data = new\_data[,-id], fullRank = F)  
testDF2 <- as.data.frame(predict(dmy,new\_data[,-id]))  
# reorder names(testDF)  
testDF3 <- testDF2  
testDF3$addr\_state.ME <- 0  
# names(testDF3) <- (names(testDF))  
  
# testDF2 <- mutate(testDF2, loan\_status = if\_else(loan\_status == 1,"Charged.Off","Fully.Paid"))  
# testing2 <- mutate(testing2, loan\_status = if\_else(loan\_status == 1,"Charged.Off","Fully.Paid"))  
  
### ============== New Submission  
  
loan\_status\_pred <- predict(upmod\_inside$finalModel,testDF2[,-id],type="class")  
prob\_upmod <- predict(upmod\_inside$finalModel,testDF3,type="prob")  
  
new\_pred <- cbind(new\_data,prob\_upmod)  
names(new\_pred)  
new\_pred <- select(new\_pred, member\_id, Charged.Off)  
names(new\_pred) <- c("member\_id", "probability")  
write\_csv(new\_pred, paste0(Original\_Data\_Location, "bag\_cart\_upsamp\_5rptcv\_50nbagg\_submission2.csv"))  
  
  
### ============================================================  
## GLM MODELS - manual   
### ============================================================  
  
  
###########################  
# Partitioning  
###########################  
  
# We want to partition our data into 70% for training, 30% for testing  
  
# create data partition row list  
set.seed(42) # setting a random seed ensures we get the same result each time  
# We will use the function 'createDataPartition' from the caret package  
# ?createDataPartition  
train = createDataPartition(y = data$loan\_status, p = 0.75, list = F)  
# partition data data into two sets   
training = data[train, ]  
testing = data[-train, ]  
str(training)  
str(testing)  
traincol <- grep("loan\_status",names(training))  
set.seed(42)  
down\_training <- caret::downSample(x = training[,-traincol],  
 y = as.factor(training$loan\_status), list = FALSE, yname = "loan\_status")  
class(down\_training)  
up\_training <- caret::upSample(x = training[,-traincol],  
 y = as.factor(training$loan\_status), list = FALSE, yname = "loan\_status")  
  
###########################  
# Variable selection  
###########################  
table(down\_training$loan\_status)  
# In this section, we will select which variables we want to include in our model  
# We'll do this by backwards selection - start with everything and remove one by one  
  
# let's start by throwing all the variables into the logistic regression  
data.glm = glm(formula = loan\_status ~ .-member\_id -home\_ownership,  
 data = down\_training,  
 family = "binomial")  
summary(data.glm)  
data.glm$formula  
# AIC ~ 607  
## downsampled AIC ~ 268  
# It's clear that we can remove some variables. This should drop the AIC  
# AIC ~ 604  
  
print(names(data))  
# We can probably remove a few more  
data.glm = glm(formula = loan\_status ~ loan\_amnt + term + int\_rate + annual\_inc + dti + delinq\_2yrs + earliest\_cr\_line + inq\_last\_6mths + open\_acc + pub\_rec + revol\_bal + total\_acc   
 + total\_pymnt + total\_rec\_late\_fee + last\_pymnt\_amnt,  
 data = down\_training,  
 family = "binomial")  
summary(data.glm)  
# AIC ~ 11492  
data.glm = glm(formula = loan\_status ~ loan\_amnt + term + int\_rate + total\_acc   
 + total\_pymnt + total\_rec\_late\_fee + last\_pymnt\_amnt,  
 data = up\_training,  
 family = "binomial")  
summary(data.glm)  
# AIC ~11538  
  
glmDF <- mutate(down\_training, loan\_status = if\_else(loan\_status == "Charged.Off",1,0))  
  
glmDF <- mutate(data, loan\_status = if\_else(loan\_status == "Charged.Off",1,0))  
data.glm = glm(formula = loan\_status ~ loan\_amnt + purpose + funded\_amnt + term + int\_rate + annual\_inc + dti + addr\_state +installment +   
 earliest\_cr\_line + inq\_last\_6mths + open\_acc + pub\_rec + revol\_bal + total\_acc + pub\_rec\_bankruptcies   
 + total\_pymnt + total\_rec\_late\_fee + last\_pymnt\_amnt + home\_ownership +emp\_length +verification\_status +sub\_grade,  
 data = glmDF,  
 family = "binomial")  
# AIC ~ 6908.6  
  
data.glm = glm(formula = loan\_status ~ loan\_amnt + purpose + funded\_amnt + term + int\_rate + annual\_inc + dti + installment +   
 total\_pymnt + total\_rec\_late\_fee + last\_pymnt\_amnt + emp\_length + sub\_grade,  
 data = glmDF,  
 family = "binomial")  
summary(data.glm)  
# AIC ~ 6878.6  
glmDF <- mutate(data, loan\_status = if\_else(loan\_status == "Charged.Off",1,0))  
  
data.glm = glm(formula = loan\_status ~ loan\_amnt + purpose + funded\_amnt + term + int\_rate + annual\_inc + dti +installment +   
 inq\_last\_6mths + open\_acc + pub\_rec + revol\_bal + total\_acc + pub\_rec\_bankruptcies   
 + total\_pymnt + total\_rec\_late\_fee + last\_pymnt\_amnt + home\_ownership +emp\_length +verification\_status +sub\_grade,  
 data = glmDF,  
 family = "binomial")  
summary(data.glm)  
# AIC ~ 6876.6  
# Let's stick with this last model  
data.glm3 = glm(formula = loan\_status ~ loan\_amnt + purpose + funded\_amnt + term + int\_rate + annual\_inc + dti + installment +   
 earliest\_cr\_line + inq\_last\_6mths + open\_acc + pub\_rec + revol\_bal + total\_acc + pub\_rec\_bankruptcies   
 + total\_pymnt + total\_rec\_late\_fee + last\_pymnt\_amnt +emp\_length + sub\_grade,  
 data = glmDF,  
 family = "binomial")  
summary(data.glm3)  
# AIC ~ 6868.6  
  
data.glm2 = glm(formula = loan\_status ~ loan\_amnt + purpose + funded\_amnt + term + int\_rate + annual\_inc + dti +installment +   
 earliest\_cr\_line + inq\_last\_6mths + open\_acc + pub\_rec + revol\_bal + total\_acc + pub\_rec\_bankruptcies   
 + total\_pymnt + total\_rec\_late\_fee + last\_pymnt\_amnt +emp\_length + verification\_status + sub\_grade,  
 data = glmDF,  
 family = "binomial")  
summary(data.glm2)  
# AIC ~ 6871.1  
# Let's stick with this last model  
  
###########################  
# Create probabilities and predictions  
###########################  
data.glm <- data.glm2  
# add the probabilities to the testing data  
# grep("home\_ownership")  
testing$probability = predict(data.glm2, newdata = testing, type = "response")  
# ?predict.glm  
# assume that the optimum probability threshold is 0.5  
# Create the class prediction - our target is the "MM" class  
testing$prediction = "Fully.Paid"  
testing[testing$probability >= 0.5, "prediction"] = "Charged.Off"  
  
# Have a look at the data  
head(testing)  
  
###########################  
# Evaluation  
  
# Create a confusion matrix (along with other measures) using the   
# function 'confusionMatrix' from the caret package  
  
confusionMatrix(data = testing$prediction, testing$loan\_status)  
  
options(scipen = 999)  
testing$probability  
  
  
prob <- predict(data.glm,new\_data[,-1],type="response")  
  
## Function to run through different combinations of glm models  
  
new\_data <- read\_csv("/Users/corinnamm/Documents/GitHub/DAM\_Assignment2\_CM/03\_DAM\_Spring2017\_Assignment\_2/version2\_validation\_all\_col\_full.csv")  
new\_data$pub\_rec\_bankruptcies <- as.character(new\_data$pub\_rec\_bankruptcies)  
## remove experiemental rows from data  
# data2 <- select(data, -loan\_status,-funded\_amnt\_inv\_prop,-installment\_prop, -total\_p, -check)  
# data2 is used to ensure the same columns are selected for the validation set as those used to train the models  
# data2 <- select(data, -loan\_status)  
new\_data <- new\_data[,colnames(data2)]  
  
## use data2 to find the mean of the non-NA data to impute into validation set  
new\_data$annual\_inc = ifelse(is.na(new\_data$annual\_inc),(mean(data2$annual\_inc)),new\_data$annual\_inc)  
new\_data$delinq\_2yrs = ifelse(is.na(new\_data$delinq\_2yrs),(mean(data2$delinq\_2yrs)),new\_data$delinq\_2yrs)  
new\_data$earliest\_cr\_line = ifelse(is.na(new\_data$earliest\_cr\_line),(mean(data2$earliest\_cr\_line)),new\_data$earliest\_cr\_line)  
new\_data$open\_acc = ifelse(is.na(new\_data$open\_acc),(mean(data2$open\_acc)),new\_data$open\_acc)  
new\_data$pub\_rec = ifelse(is.na(new\_data$pub\_rec),(mean(data2$pub\_rec)),new\_data$pub\_rec)  
new\_data$total\_acc = ifelse(is.na(new\_data$total\_acc),(mean(data2$total\_acc)),new\_data$total\_acc)  
new\_data$inq\_last\_6mths = ifelse(is.na(new\_data$inq\_last\_6mths),(mean(data2$inq\_last\_6mths)),new\_data$inq\_last\_6mths)  
new\_data$acc\_now\_delinq = ifelse(is.na(new\_data$acc\_now\_delinq),(mean(data$acc\_now\_delinq)),new\_data$acc\_now\_delinq)  
  
# check the tranformation before prediction  
summary(new\_data)  
## lets look at the problem rows of the validation that have been imputed against the original  
new\_NA <- read\_csv("/Users/corinnamm/Documents/GitHub/DAM\_Assignment2\_CM/03\_DAM\_Spring2017\_Assignment\_2/version2\_validation\_25col\_NA.csv")  
temp <- left\_join(new\_NA,new\_data, by = "member\_id")  
  
# check that the new data has the same length as required post transformation  
nrow(new\_data)  
ncol(new\_data)  
table(is.na(new\_data))  
str(new\_data)  
  
  
# new\_data <- validation\_cleaned\_data\_nick  
# problem\_rows\_func(new\_data\_original)  
# new\_NA <- (validation\_cleaned\_data\_nick, member\_id == problem\_rows[,"member\_id"][[1]])  
  
## Predictions for the validation set for the best glm model  
# data.glm$formula  
# loan\_status ~ loan\_amnt + purpose + funded\_amnt + term + int\_rate +   
# annual\_inc + dti + installment + total\_pymnt + total\_rec\_late\_fee +   
# last\_pymnt\_amnt + emp\_length + grade  
testing <- mutate(testing, loan\_status = if\_else(loan\_status == "Fully.Paid",0,1))  
testing\_prediction = prediction(testing$probability, testing$loan\_status)  
test\_tpr\_fpr = performance(testing\_prediction, "tpr","fpr")  
data.test.auc = performance(testing\_prediction, "auc")  
auc = unlist(slot(data.test.auc, "y.values"))  
auc  
##  
prediction\_prob = predict(data.glm, newdata = new\_data[,-1],  
 type = "response")  
  
new\_data$probability = predict(data.glm, newdata = new\_data[,-1],  
 type = "response")  
  
new\_data$prediction = "Fully.Paid"  
new\_data[new\_data$probability >= 0.5, "prediction"] = "Charged.Off"  
  
new\_submission <- select(new\_data, member\_id, probability)  
table(is.na(new\_submission))  
# write\_csv <- write\_csv(new\_submission, paste0(Original\_Data\_Location, "glm\_submission\_oct2\_v4.csv"))  
# write\_csv <- write\_csv(new\_submission, paste0(Original\_Data\_Location, "glm\_submission\_oct2\_v3.csv"))  
  
### PLOT AUC RESULTS  
kaggle30pct <- data.frame(AUC = c(0.99731,0.98587,0.99727,0.99564, 0.98194,0.97813,0.98095,0.98053,0.97435,0.98095, 0.97343),  
 model\_name = c("SVM\_no\_addr\_state", "glm\_feature\_selected", "svm\_all\_vars", "lasso\_all\_var","lasso\_excl\_state","bag\_cart\_2\_subgrade", "svm\_var\_select", "lasso\_var\_selectv1", "nnet\_var\_select", "svm\_var\_selectv1","bag\_cart\_1\_grade"))  
kaggle30pct <<- as.data.frame(kaggle30pct[order(-kaggle30pct$AUC),])  
row.names(kaggle30pct) = seq(1:11)  
kaggle30pct$model\_name <- factor(kaggle30pct$model\_name, levels = kaggle30pct$model\_name[order(kaggle30pct$AUC)])   
names(kaggle30pct) <- c("AUC Score", "Model")  
ggplot(kaggle30pct, aes(y = `AUC Score`, x=Model, fill = `AUC Score`)) +geom\_col() +coord\_flip() + labs(title = "The Submitted Models by AUC Score")

# Nicholas Renotte: Code Snipet

library(plyr)  
library(dplyr)  
library(tibble)  
library(ggplot2)  
library(caret)  
library(pls)  
library(xgboost)  
library(parallel)  
library(doParallel)  
library(devtools)  
library(e1071)  
  
# set up cluster for parallel computing  
cluster = makeCluster(detectCores() - 1)  
registerDoParallel(cluster)  
  
#Mac working directory  
#setwd("~/Documents/GitHub/DAM\_Assignment2\_CM/01\_Code")  
setwd("~/GitHub/DAM\_Assignment2\_CM/01\_Code")  
  
#EC2 working directory  
#raw\_df = read.csv("~/Documents/GitHub/DAM\_Assignment2\_CM/02\_Working\_data\_folder/clean\_data\_v5.csv")  
raw\_df = read.csv("C:/Users/Administrator/Downloads/training\_cleaned.csv")  
  
df = raw\_df  
  
#basic exploration  
head(df, 10)  
tail(df, 10)  
dim(df)  
str(df)  
summary(df)  
  
#frequency of target and non-target variable  
count(df$loan\_status)  
  
#check for duplicated values  
sum(duplicated(df)==TRUE)  
  
pca\_var <- c(2,3,4,7,25:28)  
  
pcadf = df[,pca\_var]  
  
str(df)  
str(pcadf)  
pr\_out = prcomp(df[,pca\_var], scale = T)  
  
names(pr\_out)  
pr\_out$rotation  
#biplot(pr\_out)  
  
#scree plot  
pr\_var = pr\_out$sdev ^ 2  
pve = pr\_var/sum(pr\_var)  
  
plot(pve, type = "b", main ="Scree Plot",   
 ylab = "Proportion of Variance Explained",   
 xlab = "Principal Component")  
  
plot(cumsum(pve), type = 'b', main = "Cumulative Variance Explained",  
 xlab = "Number of Components",  
 ylab = "Cumulative Proportion of Variance Explained")  
  
#come back to this and revaluate groupings  
  
#===============================================================  
# Lasso Model - Finished Running  
#===============================================================  
  
set.seed(42)  
train = createDataPartition(y = df$loan\_status, p = 0.7, list = F)  
  
# drop ID column  
training = df[train,]  
testing = df[-train,]  
  
str(training)  
  
trainControl = trainControl(method = "cv",  
 number = 5,  
 classProbs = T,   
 summaryFunction = twoClassSummary,   
 allowParallel = TRUE  
)  
  
#gradient boosted hyperparams  
lasso\_hyperparams = expand.grid(alpha = 1,   
 lambda = 0.0001)  
  
#exclude member id, loan status and address  
exclude <- c(1,14)  
include <- c(9,15)  
x\_train <- model.matrix( ~ .-1, training[,-exclude])  
  
lasso\_fit = train(x = x\_train, y = training$loan\_status,  
 method='glmnet',  
 trControl= trainControl,   
 tuneGrid = lasso\_hyperparams,  
 metric = "ROC" )  
  
print(lasso\_fit)  
  
newx = model.matrix( ~ .-1, testing[,-exclude])  
  
lasso\_pred = predict(lasso\_fit, newx, type = "raw")  
confusionMatrix(data = lasso\_pred, testing$loan\_status, mode = "everything", positive="Charged.Off")  
  
testing$probability = predict(lasso\_fit, newx, type = "prob")[, 1]  
testing$class = predict(lasso\_fit, newx, type = "raw")  
  
training$probability = predict(lasso\_fit, x\_train, type = "prob")[, 1]  
training$class = predict(lasso\_fit, x\_train, type = "raw")  
  
write.csv(testing, "testingresults.csv")  
write.csv(training, "trainingresults.csv")  
  
  
lasso\_importance = varImp(lasso\_fit)  
lasso\_importance$variable = rownames(lasso\_importance)  
lasso\_importance = lasso\_importance[order(lasso\_importance$Overall, decreasing = T), ]$variable  
lasso\_importance  
  
#run through validation data and prepare results  
# mac dir  
#validation = read.csv("~/Documents/GitHub/DAM\_Assignment2\_CM/02\_Working\_data\_folder/cleaned\_validation.csv")  
#windows dir  
#https://stats.stackexchange.com/questions/67827/caret-and-coefficients-glmnet  
  
coef(lasso\_fit$finalModel)  
summary = coef(lasso\_fit$finalModel, lasso\_fit$bestTune$.lambda)  
  
validation = read.csv("C:/Users/Administrator/Downloads/validation\_cleaned\_data.csv")  
amendedvalidation = validation  
  
str(amendedvalidation)  
  
str(validation)  
  
#amendedvalidation <- cbind(validation,addr\_stateME = addr\_stateME)  
validationx <- model.matrix( ~ .-1, validation[,-1])  
  
#validationx = validationx[, c(1:89, 133, 90:132)]  
#write.csv(validationx, "validationx.csv")  
#write.csv(x\_train, "x\_train.csv")  
  
validation\_pred = predict(lasso\_fit, validationx, type = "raw")  
  
validation$class = predict(lasso\_fit, validationx, type = "raw")  
validation$probability = predict(lasso\_fit, validationx, type = "prob")[, 1]  
  
results = validation[, c(1,30)]  
  
dim(validation)  
results  
  
write.csv(results, "results.csv")  
  
  
  
  
  
#===============================================================  
# Ridge Model - Finished Running  
#===============================================================  
  
set.seed(42)  
train = createDataPartition(y = df$loan\_status, p = 0.7, list = F)  
  
# drop ID column  
training = df[train,]  
testing = df[-train,]  
  
str(training)  
str(testing)  
dim(training)  
  
trainControl = trainControl(method = "cv",  
 number = 5,  
 classProbs = T,   
 summaryFunction = twoClassSummary,   
 allowParallel = TRUE  
)  
  
  
#gradient boosted hyperparams  
ridge\_hyperparams = expand.grid(alpha = 0,   
 lambda = seq(0.0001, 1, length = 100))  
  
x\_train <- model.matrix( ~ ., training[,-14])  
  
ridge\_fit = train(x = x\_train, y = training$loan\_status,  
 method='glmnet',  
 trControl= trainControl,   
 tuneGrid = ridge\_hyperparams,  
 metric = "ROC" )  
  
ridge\_fit$results  
print(ridge\_fit)  
  
newx = model.matrix( ~ ., testing[,-14])  
  
ridge\_pred = predict(ridge\_fit, newx, type = "raw")  
confusionMatrix(data = ridge\_pred, testing$loan\_status, mode = "everything", positive="Charged.Off")  
  
ridge\_fit  
  
testing$probability = predict(ridge\_fit, newx, type = "prob")[, 1]  
training$probability = predict(ridge\_fit, x\_train, type = "prob")[, 1]  
  
ridge\_importance = varImp(ridge\_fit)$importance  
ridge\_importance$variable = rownames(ridge\_importance)  
ridge\_importance = ridge\_importance[order(ridge\_importance$Overall, decreasing = T), ]$variable  
  
# partial dependence plots  
par(mfrow = c(2,2))  
for (var in ridge\_importance[1:length(ridge\_importance)]) {  
 plot.glmnet(ridge\_fit$finalModel, i.var = var, type = "response")  
}  
  
#===============================================================  
# Part 2 - Tree based classification model - basic decision tree  
#===============================================================  
  
#===============================================================  
# Gradient Boosted Model - TIMEOUT  
#===============================================================  
  
set.seed(42)  
train = createDataPartition(y = df$Target, p = 0.7, list = F)  
  
# drop ID column  
training = df[train,]  
testing = df[-train,]  
  
trainControl = trainControl(method = "cv",  
 number = 5,  
 classProbs = T,   
 summaryFunction = twoClassSummary,   
 allowParallel = TRUE  
)  
  
#gradient boosted hyperparams  
gb\_hyperparams = expand.grid(interaction.depth = c(3,5),   
 n.trees = c(200, 300),  
 shrinkage = 0.05,  
 n.minobsinnode = c(10,20))  
  
x\_train <- model.matrix( ~ ., training[,-14])  
  
gbm\_fit = train(x = x\_train, y = training$loan\_status,  
 method = "gbm",  
 trControl = trainControl,  
 tuneGrid = gb\_hyperparams,  
 metric = "ROC")  
  
gbm\_fit$results  
print(gbm\_fit)  
  
newx = model.matrix( ~ ., testing[,-14])  
  
gbm\_pred = predict(gbm\_fit, newx, type = "raw")  
confusionMatrix(gbm\_pred, testing$loan\_status, mode = "everything", positive="Charged.Off")  
  
  
testing$probability = predict(gbm\_fit, testing, type = "prob")[, 1]  
training$probability = predict(gbm\_fit, training, type = "prob")[, 1]  
  
gbm\_importance = varImp(gbm\_fit)$importance  
gbm\_importance$variable = rownames(gbm\_importance)  
gbm\_importance = gbm\_importance[order(gbm\_importance$Overall, decreasing = T), ]$variable  
gbm\_importance  
# partial dependence plots  
par(mfrow = c(2,2))  
for (var in gbm\_importance[1:length(gbm\_importance)]) {  
 plot.gbm(gbm\_fit$finalModel, i.var = var, type = "response")  
}  
  
#===============================================================  
#Random Forest Model - TIMEOUT  
#===============================================================  
  
df$Target = make.names(df$Target)  
set.seed(42)  
train = createDataPartition(y = df$Target, p = 0.7, list = F)  
  
# drop ID column  
training = df[train,]  
testing = df[-train,]  
  
trainControl = trainControl(method = "cv",  
 number = 5,  
 classProbs = T,   
 summaryFunction = twoClassSummary,   
 allowParallel = TRUE  
)  
  
#random forrest hyperparams  
rf\_hyperparams = expand.grid(mtry = c(4,5,6))  
  
# need to fix this   
set.seed(42)  
x\_train <- model.matrix( ~ ., training[,-14])  
  
rf\_fit = train(x = x\_train, y = training$loan\_status,  
 method = "rf",  
 trControl = trainControl,  
 tuneGrid = rf\_hyperparams,  
 metric = "ROC")  
  
rf\_fit$results  
print(rf\_fit)  
newx = model.matrix( ~ ., testing[,-14])  
  
rf\_pred = predict(rf\_fit, newx, type = "raw")  
confusionMatrix(rf\_pred, testing$loan\_status, mode = "everything", positive="Charged.Off")  
  
testing$probability = predict(rf\_fit, testing, type = "prob")[, 1]  
training$probability = predict(rf\_fit, training, type = "prob")[, 1]  
  
rf\_importance = varImp(rf\_fit)$importance  
#rf\_importance$variable = rownames(rf\_importance)  
#rf\_importance = rf\_importance[order(rf\_importance$Overall, decreasing = T), ]$variable  
rf\_importance  
  
# partial dependence plots  
par(mfrow = c(2,2))  
counter = 0  
for (i in seq\_along(rf\_importance)[1:14]) {  
 counter = counter + 1  
 partialPlot(x = rf\_fit$finalModel, pred.data = training, x.var = rf\_importance[i],  
 main = paste("Partial Dependence on", rf\_importance[i]),  
 xlab = rf\_importance[i], col = "red",  
 which.class = "X1")  
}  
  
#validation data set  
#validation = read.csv("repurchase\_validation.csv")  
#validation$validation\_pred = predict(rf\_fit, validation, type = "raw")  
#validation$probability = predict(rf\_fit, validation, type = "prob")[, 1]  
#write.csv(validation, "validation.csv")  
  
#count(validation$validation\_pred)  
  
#=====================================================  
# XGBoost Model w/ k = 5 fold cv plus grid search  
#=====================================================  
  
set.seed(42)  
train = createDataPartition(y = df$loan\_status, p = 0.7, list = F)  
  
# drop ID column  
training = df[train,]  
testing = df[-train,]  
  
dim(training)  
dim(testing)  
  
xgb\_control = trainControl(method = "cv",  
 number = 5,  
 classProbs = T,   
 summaryFunction = twoClassSummary,   
 allowParallel = TRUE  
)  
  
# XGB Hyperparams  
# tuning params for XGB best practice https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/  
xgb\_hyperparams = expand.grid(nrounds = 100,  
 eta = 0.1,  
 max\_depth = c(3,6),  
 gamma = 1, #default = 0   
 colsample\_bytree = c(0.5,0.7),   
 min\_child\_weight = 2,   
 subsample = 0.5   
)  
  
# Consider scaling this back as model takes a while to train  
#xgb\_hyperparams = expand.grid(nrounds = 1000,  
# eta = c(0.01, 0.001, 0.001),  
# max\_depth = c(2,4,6,8,10),  
# gamma = c(0,1), #default = 0   
# colsample\_bytree = c(0.6,0.8,1),   
# min\_child\_weight = 1,   
# subsample = c(0.5,0.75,1)   
#)  
  
x\_train <- model.matrix( ~ ., training[,-14])  
  
xgb\_fit = train(x = x\_train, y = training$loan\_status,  
 method='xgbTree',  
 trControl= xgb\_control,   
 tuneGrid = xgb\_hyperparams,   
 metric = 'ROC')  
  
  
#Models using upsampling and downsampling  
#https://topepo.github.io/caret/subsampling-for-class-imbalances.html   
  
  
xgb\_fit$results  
print(xgb\_fit)  
  
newx = model.matrix( ~ ., testing[,-14])  
  
ridge\_pred = predict(xgb\_fit, newx, type = "raw")  
confusionMatrix(data = ridge\_pred, testing$loan\_status, mode = "everything", positive="Charged.Off")  
  
count(df$loan\_status)  
  
testing$probability = predict(xgb\_fit, newx, type = "prob")[, 1]  
training$probability = predict(xgb\_fit, x\_train, type = "prob")[, 1]

# Additionally used function script

(From Corinna & Class Provided)

###########################################################  
  
### DATA ALGOITHMS AND MEANING  
### Functions Collection  
### LOAN DEFAULT CHALLENGE  
  
### Corinna Maher Mittmann  
### SEP 2017  
  
### DOCUMENT NAME: 99\_functions\_DAM  
###########################################################  
  
  
########################################################### Load functions for evaaluation  
# Function to create the ROC chart, and the sensitivity / specificity charts  
  
  
model\_evaluation = function(training = training, testing = testing,   
 model = gbm\_fit,  
 target = "target") {  
 library(ROCR)  
   
 # create prediction object on the testing data  
 testing\_prediction = prediction(testing$probability, testing[, target])  
 # create prediction object on the training data  
 training\_prediction = prediction(training$probability, training[, target])  
   
   
 # Create model performance objects on test data set - uses ROCR  
 test\_tpr\_fpr = performance(testing\_prediction, "tpr","fpr")  
   
 data.test.auc = performance(testing\_prediction, "auc")  
 data.test.lift = performance(testing\_prediction, "lift", "rpp")  
 # Area under the ROC curve  
 auc = unlist(slot(data.test.auc, "y.values"))  
 cat("\nAUC on test data is ", round(100 \* auc, 2), "%", sep = "")  
 # Create model performance objects on training data set - uses ROCR  
 train\_tpr\_fpr <- performance(training\_prediction, "tpr","fpr")  
   
   
 # Plot the tpr and fpr gains chart ROC for both testing and training data  
 plot(test\_tpr\_fpr, main = "Gains Chart ROC", type = "l", col = "red", lwd = 2,   
 xlim = c(0,1), ylim = c(0,1))  
 plot(train\_tpr\_fpr, add = T, col = "blue", lwd = 2, lty = 2, xlim = c(0,1), ylim = c(0,1))  
 legend("bottomright", legend = c("Training","Testing"), col = c("blue","red"), lty = 1, lwd = 2)  
 abline(0,1, col = "darkgray")  
 grid()  
   
 # Sensitivity / Specificity charts  
   
 test\_sens\_spec = performance(testing\_prediction, "sens","spec")  
   
 sens = performance(testing\_prediction, "sens")  
 spec = performance(testing\_prediction, "spec")  
   
 plot(sens,   
 main = "Sensitivity Specificity Chart", type = "l", col = "red", lwd = 2,   
 xlim = c(0,1), ylim = c(0,1),   
 ylab = "Values")  
 axis(side = 1, at = seq(0, 1, 0.1))  
 axis(side = 2, at = seq(0, 1, 0.1))  
 plot(spec, add = T, col = "blue", lwd = 2,   
 xlim = c(0,1), ylim = c(0,1)  
 )  
 legend("bottomright", legend = c("Sensitivity","Specificity"), col = c("red", "blue"), lty = 1, lwd = 2)  
 abline(h = seq(0, 1, 0.1), v = seq(0, 1, 0.1), col="gray", lty=3)  
   
   
 # Probability Threshold calculation  
 threshold.df = data.frame(cut = test\_sens\_spec@alpha.values[[1]],   
 sens = test\_sens\_spec@x.values[[1]],  
 spec = test\_sens\_spec@y.values[[1]])  
 threshold = threshold.df[which.max(threshold.df$sens + threshold.df$spec), "cut"]  
 cat("\nProbability threshold is", threshold)  
 cat("\n")  
 return(threshold)  
}  
  
###########################################################  
# Corinna's personal functions  
##### Check for missing function  
check\_for\_missing <- function(data\_input) {  
 for (i in 1:length(names(data\_input))){  
 print(names(data\_input)[i])  
   
 if (sum(is.na(data\_input[,i]) | data\_input[,i] %in% c("", " "," ","N/A","NULL","na","n.a","NA",NA,"missing","unknown","Unknown","Missing")) !=0) {  
 message(paste0(names(data\_input)[i]), " has missing values")  
 number\_of\_missings <- table(data\_input[,i] %in% c("", " "," ","N/A","NULL","na","n.a","NA",NA,"missing","unknown","Unknown","Missing"))  
 print(number\_of\_missings)  
 message(paste0("The number of missing values is ",number\_of\_missings[[2]]))  
 message(paste0("the % of missing values is ",round((number\_of\_missings[[2]])/(length(data\_input[,i]))\*100, digits = 2)," %"))  
 } else {  
 message("no missing")  
 }  
   
 }  
}  
##### define a row function  
row\_distinction <- function(data\_input){  
 for (i in 1:(length(names(data\_input)))){  
 print(i)  
 print(n\_distinct(data\_input[,-i]))  
 print(n\_distinct(data\_input))  
 if (n\_distinct(data\_input[,-i]) == n\_distinct(data\_input)) {  
 message(paste0("dataset is distinct without ", names(data\_input)[i]))  
 } else { message(paste0("dataset is NOT distinct without ",names(data\_input)[i]))  
 }  
   
 }  
}  
  
# GEt the mean and standart deviations  
  
descriptive\_info = function(data\_input, target\_col) {  
 q\_temp <- NULL  
 for (i in 1:(length(names(data\_input)))){  
 print(i)  
 if (class(data\_input[,i]) == "integer") {  
 print(i)  
 this\_col <- names(data\_input)[i]  
 print(paste0("the column is ",this\_col))  
 temp = data\_input[,i]  
 mean = mean(temp)  
 print(paste0("mean = ", mean))  
 median = median(temp)  
 print(paste0("median = ", median))  
 standard\_deviation = sd(temp)  
 print(paste0("standard deviation = ", standard\_deviation))  
 minimum = min(temp)  
 print(paste0("minimum value = ", minimum))  
 maximum = max(temp)  
 print(paste0("maximum value = ", maximum))  
 number\_of\_rec = nrow(temp)  
   
 # density plot  
 x\_temp <- names(data\_input[,i])  
 q\_temp <- qplot(x = data\_input[,i], data = data\_input, geom = "density", fill = data\_input[,target\_col] , alpha = I(0.25)) +  
 labs(title = paste0("Density plot for ",this\_col),fill = names(paste0(data\_input[,target\_col]))) + xlab(paste0(this\_col))  
 plot(q\_temp)  
 } else {  
 message("non-numeric")  
   
 }  
   
 }   
 return()  
}  
###########################################################  
check\_for\_new\_levels <- function(old\_data,new\_data) {  
 for (i in 1:ncol(old\_data)){  
 old\_data = as.tibble(old\_data)  
 new\_data = as.tibble(new\_data)  
   
 this\_col\_old = names(old\_data)[i]  
 this\_col\_num\_old = grep(this\_col\_old,names(old\_data))  
   
 print(paste0("i = ",i," , ", this\_col\_old))  
   
 if (this\_col\_old %in% names(new\_data)) {  
   
 this\_col\_new = this\_col\_old  
 this\_col\_num\_new = grep(this\_col\_new, names(new\_data))  
 message(paste0("The columns exists in both data sets as new\_data[," ,this\_col\_new,"] num: ", this\_col\_num\_new," and old\_data[,",this\_col\_old,"] num: ", this\_col\_num\_old))  
   
 xnax<-table(!is.na(new\_data[,this\_col\_num\_new][[1]]))  
 message(paste0("number of NA ",((nrow(new\_data)-xnax["TRUE"][[1]])/nrow(new\_data)\*100), "%"))  
   
   
 if (sapply(new\_data[,this\_col\_num\_new], class)[[1]] == "factor") {  
 # message("is a factor")  
   
 print(paste0("distinct levels new data: ",n\_distinct(new\_data[,this\_col\_num\_new]),", old data: ", n\_distinct(old\_data[,this\_col\_num\_old])))  
 if (n\_distinct(new\_data[,this\_col\_num\_new][[1]]) <= n\_distinct(old\_data[,this\_col\_num\_old][[1]])) {  
 message("same or less n levels than old data (no new levels detected yet)")  
 temp = unique(new\_data[,this\_col\_num\_new][[1]])[grep(TRUE,(!(unique(new\_data[,this\_col\_num\_new][[1]]) %in% unique(old\_data[,this\_col\_num\_old][[1]]))))]  
   
 if (length(temp) > 0) {   
 print(paste0("number of levels of the new data that are not the same: ",  
 length(as.character(temp))))  
 print(as.character(temp))  
 } else if (n\_distinct(new\_data[,this\_col\_num\_new][[1]]) < n\_distinct(old\_data[,this\_col\_num\_old][[1]])){ print("no new levels confirmed")  
 temp2 = unique(old\_data[,this\_col\_num\_old][[1]])[grep(TRUE,(!(unique(old\_data[,this\_col\_num\_old][[1]]) %in% unique(new\_data[,this\_col\_num\_new][[1]]))))]  
 message(paste0("there are ", length(temp2)," levels in the old data that are not in the new data: ", temp2))  
 } else {print("no new levels confirmed")}  
   
 if (n\_distinct(new\_data[,this\_col\_num\_new][[1]]) <= 20) {  
 table(new\_data[,this\_col\_num\_new][[1]])[(unique(new\_data[,this\_col\_num\_new][[1]]) %in% unique(old\_data[,this\_col\_num\_old][[1]]))]  
 } else { message("too many levels to display")}  
 } else if (n\_distinct(new\_data[,this\_col\_num\_new][[1]]) > n\_distinct(old\_data[,this\_col\_num\_old][[1]])) {  
 message("new levels")  
 message("the number of new levels is ", length(unique(new\_data[,this\_col\_num\_new][[1]]))[grep(TRUE,!(new\_data[,this\_col\_num\_new][[1]] %in% unique(old\_data[,this\_col\_num\_old][[1]])))])  
 unique((new\_data[,this\_col\_num\_new][[1]])[grep(TRUE,!(unique(new\_data[,this\_col\_num\_new][[1]]) %in% unique(old\_data[,this\_col\_num\_old][[1]])))])  
   
 }   
   
 } else {  
 message("not a factor")  
 }  
 } else {  
 message("no such column")  
 }  
   
 }  
   
   
}  
  
## id location for problem rows (missing values)  
  
problem\_rows\_func <- function(problem\_rows, new\_data){  
 for (i in 1:ncol(new\_data)) {  
 temp = new\_data$member\_id[grep(TRUE, is.na(new\_data[,i][[1]]))]  
 temp = data\_frame(member\_id = temp)  
 problem\_rows <- bind\_rows(problem\_rows, temp)  
   
   
 }   
   
 problem\_rows <<- problem\_rows[-1,]  
}  
  
## a data review function:  
review\_data\_func = function(data, target\_binary\_level1, target\_binary\_level0) {  
 for (i in 1:(length(names(data)))){  
 print(i)  
 if (n\_distinct(data[,i]) >100) {  
 message("too many levels") # mainly to remove ID or other keys that are not variables of interest  
 } else { # rather than histograms, this will show the tabular distribution without requirement of order for categories  
 par(mfrow = c(2,2))  
 barplot(table(data[,i]), col = "pink", main = paste0(names(data)[i]) )  
 title(xlab = "Variable levels or categories", ylab = "Frequency")  
 tmp <- filter(data, loan\_status == target\_binary\_level1)  
 barplot(table(tmp[,i]), col = "red", main = paste0("Target = 1 for ",names(data)[i]) )  
 title(xlab = "Variable levels or categories", ylab = "Frequency")  
 tmp2 <- filter(data, loan\_status == target\_binary\_level0)  
 barplot(table(tmp2[,i]), col = "blue", main = paste0("Target = 0 for ",names(data)[i]) )  
 title(xlab = "Variable levels or categories", ylab = "Frequency")  
 }   
   
   
 }  
}

# For Report Writing Contributions See Appendix of the Report.