Homework 4

Jiaming Chen

1/5/2018

library(tidyverse) # my usual tools, ggplot2, dplyr

## Loading tidyverse: ggplot2  
## Loading tidyverse: tibble  
## Loading tidyverse: tidyr  
## Loading tidyverse: readr  
## Loading tidyverse: purrr  
## Loading tidyverse: dplyr

## Conflicts with tidy packages ----------------------------------------------

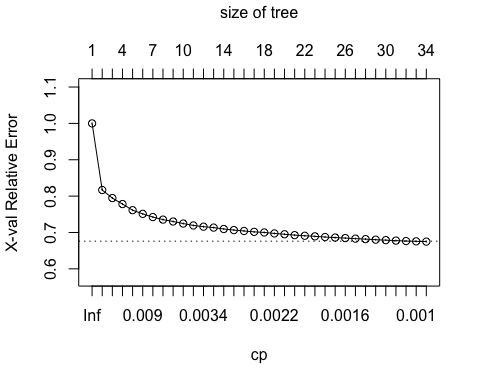
## filter(): dplyr, stats  
## lag(): dplyr, stats

setwd('/Users/jessie/Desktop/Bittiger/Month1')

## Simple Tree

load('fromHW3.RData')

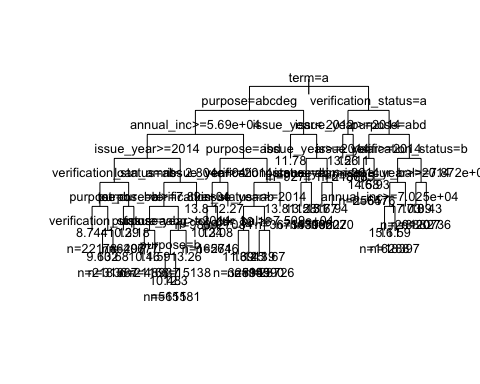
library(rpart)  
  
formula <- paste("int\_rate ~ ", paste(colnames(train.sub)[-1], collapse = " + "))  
  
tree0 <- rpart(formula, method = 'anova', data = train.sub,   
 control=rpart.control(cp = 0.001))  
  
plotcp(tree0)



bestcp <- tree0$cptable[which.min(tree0$cptable[,"xerror"]), "CP"]  
  
cp.tab <- as.data.frame(tree0$cptable)  
#with(cp.tab, min(which(xerror - 2\*xstd < min(xerror))))  
bestcp <- cp.tab$CP[with(cp.tab, min(which(xerror - xstd < min(xerror))))]  
# Step 3: Prune the tree using the best cp.  
tree.pruned <- prune(tree0, cp = bestcp)  
# tree.pruned$cptable  
# tree0$cptable  
# in this case tree.pruned and tree0 are the same  
# because not yet overfitting  
test.pred <- predict(tree.pruned, test)  
sqrt(sum((test.pred - test$int\_rate)^2) / length(test.pred)) # 3.6

## [1] 3.602657

plot(tree.pruned, uniform = TRUE)   
text(tree.pruned, cex = 0.8, use.n = TRUE, xpd = TRUE)

 ## Random Forest

# random forest  
library(randomForest)

## randomForest 4.6-12

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

##   
## Attaching package: 'randomForest'

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

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

# Categorical featues need to be factors when using randomForest  
train.sub$purpose <- as.factor(train.sub$purpose)  
train.sub$state\_mean\_int <- as.factor(train.sub$state\_mean\_int)  
train.sub$home\_ownership <- as.factor(train.sub$home\_ownership)  
train.sub$term <- as.factor(train.sub$term)  
train.sub$verification\_status <- as.factor(train.sub$verification\_status)  
set.seed(222)

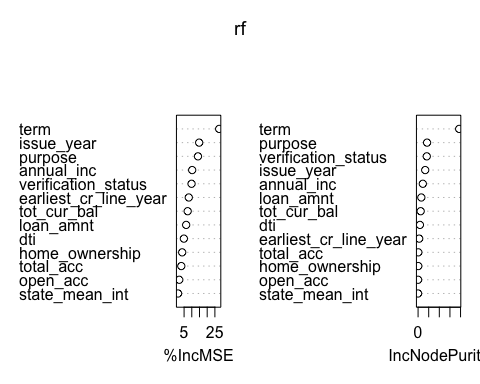
# random forest does not allow NA values in the dataframe  
train.sub <- na.omit(train.sub)  
  
rf <- randomForest(x = train.sub[, -1], y = train.sub[, 1], importance = TRUE,  
 do.trace = TRUE, nodesize = 6200, ntree = 10)

## | Out-of-bag |  
## Tree | MSE %Var(y) |  
## 1 | 13.74 71.52 |  
## 2 | 13.21 68.77 |  
## 3 | 13.02 67.78 |  
## 4 | 12.92 67.28 |  
## 5 | 12.8 66.62 |  
## 6 | 12.74 66.31 |  
## 7 | 12.66 65.92 |  
## 8 | 12.63 65.78 |  
## 9 | 12.6 65.57 |  
## 10 | 12.52 65.20 |

par(mar=rep(2,4)) # change margin in plot setting.  
# check the setting in par(), like   
par()$mfrow

## [1] 1 1

par(mfrow = c(1,1))  
varImpPlot(rf)



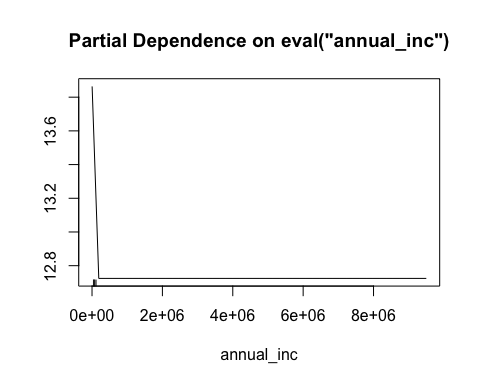
importance(rf, type = 1) # 2 for InNodePurity)

## %IncMSE  
## annual\_inc 10.249135  
## dti 4.860347  
## loan\_amnt 6.343773  
## total\_acc 3.183223  
## tot\_cur\_bal 7.333355  
## open\_acc 1.874528  
## issue\_year 14.796311  
## earliest\_cr\_line\_year 8.043541  
## purpose 14.020028  
## home\_ownership 3.771872  
## state\_mean\_int 1.099171  
## term 27.772737  
## verification\_status 9.846862

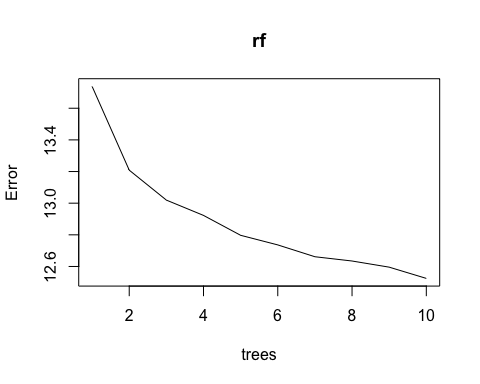
importanceOrder= order(rf$importance[, "%IncMSE"], decreasing = T)  
names=rownames(rf$importance)[importanceOrder]

it looks like that “Purpose”, “verification\_status” and “issue\_year” is good features to add.

# partialPlot is interpreted as the predicted value for a particular value of an explanatory variable  
partialPlot(rf, train.sub, eval('annual\_inc'), xlab='annual\_inc')



plot(rf) # see oob error



PartialPlot is a very good tool for interpreting our model, especially for business analysis

test.sub$purpose <- as.factor(test.sub$purpose)  
test.sub$state\_mean\_int <- as.factor(test.sub$state\_mean\_int)  
test.sub$home\_ownership <- as.factor(test.sub$home\_ownership)  
test.sub$term <- as.factor(test.sub$term)  
test.sub$verification\_status <- as.factor(test.sub$verification\_status)  
test.sub <- na.omit(test.sub)  
test.pred <- predict(rf, test.sub)  
sqrt(sum((test.pred - test.sub$int\_rate)^2) / dim(test)[1]) # 3.50

## [1] 3.504836

# Boosting Tree

library(xgboost)

##   
## Attaching package: 'xgboost'

## The following object is masked from 'package:dplyr':  
##   
## slice

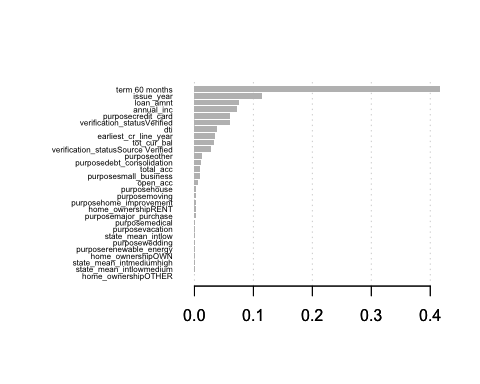
train.label <- train.sub$int\_rate  
# Xgboost manages only numeric vectors.  
feature.matrix <- model.matrix( ~ ., data = train.sub[, -1])   
# Remember we removed rows with NA in randomForest fitting. model.matrix will also remove rows with any NA.  
set.seed(222)  
gbt <- xgboost(data = feature.matrix,   
 label = train.label,   
 max\_depth = 8, # for each tree, how deep it goes  
 nround = 20, # number of trees  
 objective = "reg:linear",  
 nthread = 3,  
 verbose = 1)

## [1] train-rmse:9.765706   
## [2] train-rmse:7.276875   
## [3] train-rmse:5.660898   
## [4] train-rmse:4.659287   
## [5] train-rmse:4.068280   
## [6] train-rmse:3.736374   
## [7] train-rmse:3.554072   
## [8] train-rmse:3.451819   
## [9] train-rmse:3.397461   
## [10] train-rmse:3.363843   
## [11] train-rmse:3.342940   
## [12] train-rmse:3.329333   
## [13] train-rmse:3.319963   
## [14] train-rmse:3.309497   
## [15] train-rmse:3.302777   
## [16] train-rmse:3.298830   
## [17] train-rmse:3.289214   
## [18] train-rmse:3.280490   
## [19] train-rmse:3.277766   
## [20] train-rmse:3.266411

importance <- xgb.importance(feature\_names = colnames(feature.matrix), model = gbt)  
importance

## Feature Gain Cover  
## 1: term 60 months 4.159589e-01 6.379398e-02  
## 2: issue\_year 1.153889e-01 1.020777e-01  
## 3: loan\_amnt 7.569560e-02 1.998858e-01  
## 4: annual\_inc 7.213399e-02 5.530072e-02  
## 5: purposecredit\_card 6.075317e-02 4.420586e-02  
## 6: verification\_statusVerified 5.954049e-02 3.942059e-02  
## 7: dti 3.778307e-02 8.913424e-02  
## 8: earliest\_cr\_line\_year 3.415173e-02 8.264585e-02  
## 9: tot\_cur\_bal 3.343745e-02 3.188418e-02  
## 10: verification\_statusSource Verified 2.749041e-02 2.335107e-02  
## 11: purposeother 1.248959e-02 3.159150e-02  
## 12: purposedebt\_consolidation 1.103845e-02 5.428550e-03  
## 13: total\_acc 9.782143e-03 4.302755e-02  
## 14: purposesmall\_business 9.673761e-03 3.361284e-02  
## 15: open\_acc 6.650847e-03 1.973099e-02  
## 16: purposehouse 3.390947e-03 2.178088e-02  
## 17: purposemoving 3.307096e-03 2.797741e-02  
## 18: purposehome\_improvement 3.263046e-03 7.733493e-04  
## 19: home\_ownershipRENT 2.681436e-03 8.743836e-03  
## 20: purposemajor\_purchase 2.189948e-03 1.061089e-02  
## 21: purposemedical 1.577983e-03 2.432707e-02  
## 22: purposevacation 5.923571e-04 1.871337e-02  
## 23: state\_mean\_intlow 2.732563e-04 6.269261e-03  
## 24: purposewedding 2.474364e-04 3.071400e-03  
## 25: purposerenewable\_energy 2.252602e-04 1.232237e-02  
## 26: home\_ownershipOWN 1.096723e-04 2.190608e-04  
## 27: state\_mean\_intmediumhigh 8.349651e-05 2.086054e-05  
## 28: state\_mean\_intlowmedium 8.205582e-05 6.438289e-05  
## 29: home\_ownershipOTHER 7.575830e-06 1.436992e-05  
## Feature Gain Cover  
## Frequency  
## 1: 0.0115286080  
## 2: 0.1157130658  
## 3: 0.1731426132  
## 4: 0.1295900939  
## 5: 0.0166524338  
## 6: 0.0172929120  
## 7: 0.1270281810  
## 8: 0.0815542272  
## 9: 0.0999146029  
## 10: 0.0125960717  
## 11: 0.0132365500  
## 12: 0.0074722459  
## 13: 0.0638343296  
## 14: 0.0128095645  
## 15: 0.0625533732  
## 16: 0.0068317677  
## 17: 0.0049103330  
## 18: 0.0055508113  
## 19: 0.0117421008  
## 20: 0.0053373185  
## 21: 0.0042698548  
## 22: 0.0010674637  
## 23: 0.0036293766  
## 24: 0.0012809564  
## 25: 0.0006404782  
## 26: 0.0032023911  
## 27: 0.0036293766  
## 28: 0.0027754056  
## 29: 0.0002134927  
## Frequency

xgb.plot.importance(importance)

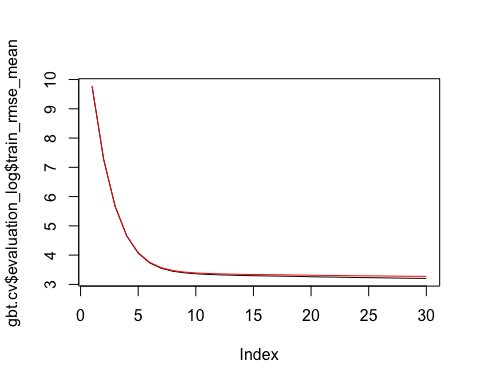


Cross Validation for gradient boosted tree

par <- list( max\_depth = 8,  
 objective = "reg:linear",  
 nthread = 3,  
 verbose = 2)  
gbt.cv <- xgb.cv(params = par,  
 data = feature.matrix, label = train.label,  
 nfold = 5, nrounds = 30)

## [1] train-rmse:9.765379+0.002076 test-rmse:9.766609+0.011800   
## [2] train-rmse:7.277411+0.002136 test-rmse:7.281322+0.011046   
## [3] train-rmse:5.662115+0.001141 test-rmse:5.668394+0.011254   
## [4] train-rmse:4.658696+0.001520 test-rmse:4.668931+0.011035   
## [5] train-rmse:4.067725+0.002436 test-rmse:4.082187+0.010816   
## [6] train-rmse:3.733422+0.001816 test-rmse:3.751840+0.011865   
## [7] train-rmse:3.549319+0.003291 test-rmse:3.571076+0.010351   
## [8] train-rmse:3.448408+0.002470 test-rmse:3.473169+0.010977   
## [9] train-rmse:3.393696+0.002817 test-rmse:3.421118+0.010979   
## [10] train-rmse:3.361291+0.003389 test-rmse:3.391302+0.009765   
## [11] train-rmse:3.340296+0.004015 test-rmse:3.372671+0.010270   
## [12] train-rmse:3.326210+0.002987 test-rmse:3.360476+0.009971   
## [13] train-rmse:3.315066+0.003387 test-rmse:3.351936+0.010003   
## [14] train-rmse:3.303567+0.004772 test-rmse:3.342228+0.007698   
## [15] train-rmse:3.295707+0.003912 test-rmse:3.336555+0.009133   
## [16] train-rmse:3.289219+0.005174 test-rmse:3.332247+0.007650   
## [17] train-rmse:3.283494+0.004994 test-rmse:3.328490+0.007948   
## [18] train-rmse:3.276483+0.004353 test-rmse:3.323357+0.009252   
## [19] train-rmse:3.269928+0.003555 test-rmse:3.318777+0.008565   
## [20] train-rmse:3.260283+0.005736 test-rmse:3.311364+0.008483   
## [21] train-rmse:3.252648+0.003655 test-rmse:3.305958+0.009705   
## [22] train-rmse:3.248574+0.004137 test-rmse:3.303655+0.008032   
## [23] train-rmse:3.241348+0.007027 test-rmse:3.298574+0.004567   
## [24] train-rmse:3.234397+0.009337 test-rmse:3.293541+0.004346   
## [25] train-rmse:3.229053+0.007322 test-rmse:3.289666+0.004925   
## [26] train-rmse:3.224788+0.006599 test-rmse:3.287564+0.005175   
## [27] train-rmse:3.218144+0.004195 test-rmse:3.282880+0.007892   
## [28] train-rmse:3.212242+0.005906 test-rmse:3.278894+0.007390   
## [29] train-rmse:3.207795+0.006877 test-rmse:3.276571+0.006877   
## [30] train-rmse:3.204024+0.006241 test-rmse:3.274709+0.006962

plot(gbt.cv$evaluation\_log$train\_rmse\_mean, type = 'l')  
lines(gbt.cv$evaluation\_log$test\_rmse\_mean, col = 'red')



nround = which(gbt.cv$evaluation\_log$test\_rmse\_mean ==   
 min(gbt.cv$evaluation\_log$test\_rmse\_mean)) # 30

Grid search for best parameter combinations

all\_param = NULL  
all\_test\_rmse = NULL  
all\_train\_rmse = NULL  
  
for (iter in 1:10) {  
 print(iter)  
 param <- list(objective = "reg:linear",  
 max\_depth = sample(5:12, 1),   
 subsample = runif(1, .5, .9)  
 #eta = runif(1, .01, .3),  
 #gamma = runif(1, 0.0, 0.2),  
 #colsample\_bytree = 1,  
 #min\_child\_weight = sample(1:40, 2),  
 #max\_delta\_step = sample(1:10, 2)  
 )  
 cv.nround = 30  
 cv.nfold = 5  
 set.seed(iter)  
 mdcv <- xgb.cv(data=feature.matrix, label = train.label, params = param,   
 nfold=cv.nfold, nrounds=cv.nround,  
 verbose = F, early\_stop\_round=8,   
 maximize = FALSE)  
 min\_train\_rmse = min(mdcv$evaluation\_log$train\_rmse\_mean)  
 min\_test\_rmse = min(mdcv$evaluation\_log$test\_rmse\_mean)  
   
 all\_param <- rbind(all\_param, c(param$max\_depth,param$subsample))  
 all\_train\_rmse <- c(all\_train\_rmse, min\_train\_rmse)  
 all\_test\_rmse <- c(all\_test\_rmse, min\_test\_rmse)  
}

## [1] 1  
## [1] 2  
## [1] 3  
## [1] 4  
## [1] 5  
## [1] 6  
## [1] 7  
## [1] 8  
## [1] 9  
## [1] 10

summary(all\_test\_rmse)

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 3.262 3.269 3.287 3.296 3.316 3.349

#find the best parameters  
best\_param <- list(objective = "reg:linear",  
 max\_depth = all\_param[which.min(all\_test\_rmse),1],  
 subsample = all\_param[which.min(all\_test\_rmse),2])  
  
gbt.cv.best <- xgb.cv(params = best\_param,  
 data = feature.matrix, label = train.label,  
 nfold = 5,  
 nrounds = 30,  
 nthread = 3,  
 verbose = T,early\_stop\_round=8,)

## [1] train-rmse:9.756260+0.002171 test-rmse:9.759750+0.010997   
## [2] train-rmse:7.256945+0.001479 test-rmse:7.266617+0.009452   
## [3] train-rmse:5.629804+0.001000 test-rmse:5.649082+0.007853   
## [4] train-rmse:4.615601+0.000727 test-rmse:4.645688+0.007764   
## [5] train-rmse:4.013470+0.001875 test-rmse:4.054732+0.007348   
## [6] train-rmse:3.671809+0.003024 test-rmse:3.723276+0.006643   
## [7] train-rmse:3.482781+0.003514 test-rmse:3.543878+0.006194   
## [8] train-rmse:3.378847+0.003056 test-rmse:3.448249+0.004900   
## [9] train-rmse:3.318379+0.003519 test-rmse:3.394970+0.004472   
## [10] train-rmse:3.282573+0.004591 test-rmse:3.365246+0.006973   
## [11] train-rmse:3.258548+0.002654 test-rmse:3.347334+0.006132   
## [12] train-rmse:3.241508+0.003883 test-rmse:3.335793+0.004948   
## [13] train-rmse:3.226762+0.005193 test-rmse:3.325829+0.007469   
## [14] train-rmse:3.214288+0.001516 test-rmse:3.318394+0.006134   
## [15] train-rmse:3.202351+0.004238 test-rmse:3.311573+0.005884   
## [16] train-rmse:3.193378+0.002798 test-rmse:3.306982+0.004628   
## [17] train-rmse:3.185796+0.003179 test-rmse:3.303986+0.004136   
## [18] train-rmse:3.176356+0.003251 test-rmse:3.298975+0.003720   
## [19] train-rmse:3.166699+0.004290 test-rmse:3.294159+0.005025   
## [20] train-rmse:3.156037+0.001024 test-rmse:3.289563+0.005177   
## [21] train-rmse:3.147723+0.002260 test-rmse:3.285584+0.004957   
## [22] train-rmse:3.139562+0.005291 test-rmse:3.281559+0.006916   
## [23] train-rmse:3.130768+0.005150 test-rmse:3.276510+0.005053   
## [24] train-rmse:3.124095+0.005424 test-rmse:3.274148+0.004273   
## [25] train-rmse:3.119570+0.006227 test-rmse:3.273675+0.004429   
## [26] train-rmse:3.112444+0.003975 test-rmse:3.270621+0.003319   
## [27] train-rmse:3.105563+0.005527 test-rmse:3.267525+0.003701   
## [28] train-rmse:3.099746+0.003173 test-rmse:3.265520+0.004413   
## [29] train-rmse:3.090740+0.002079 test-rmse:3.261777+0.004116   
## [30] train-rmse:3.086111+0.001189 test-rmse:3.260522+0.003430

# prediction  
test.sub <- test.sub[which(apply(test.sub, 1,   
 function(x) length(which(is.na(x))) == 0)), ]  
prediction <- predict(gbt, model.matrix( ~ ., data = test.sub[, -1]))  
# gradient boosted tree  
sqrt(sum((prediction - test.sub$int\_rate)^2)/dim(test.sub)[1])

## [1] 3.307856

#simple tree  
sqrt(sum((predict(tree0, test.sub) - test.sub$int\_rate)^2)/dim(test.sub)[1])

## [1] 3.600028

#random forest  
sqrt(sum((predict(rf, test.sub) - test.sub$int\_rate)^2)/dim(test.sub)[1])

## [1] 3.504909

#linear model  
sqrt(sum((predict(lm(formula, train.sub), test.sub) - test.sub$int\_rate)^2)/dim(test.sub)[1])

## [1] 3.612869