C\_07.R

TPRABHAN

Tue May 15 11:40:32 2018

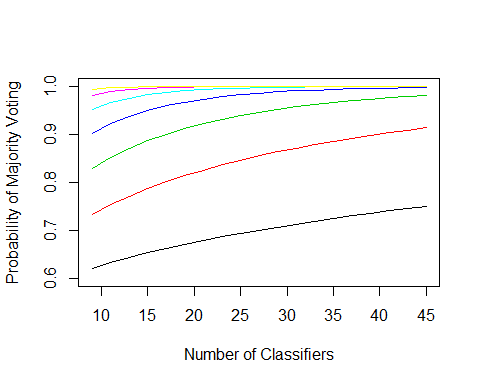
### Understanding the Ensembling Theory  
library(randomForest)

## Warning: package 'randomForest' was built under R version 3.4.1

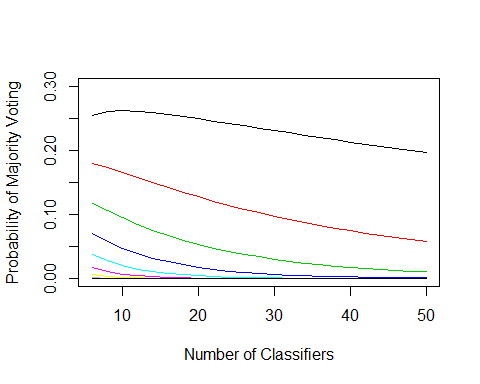
## randomForest 4.6-12

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

setwd("C:/Users/tprabhan/Documents/My\_Books/ELwR/R\_Programs/Chapter\_07/SRC")  
  
  
# Ensembling in Classification  
# Illustrating the ensemble accuracy with same accuracy for each classifier  
# Different p's and T's with p > 0.5  
T <- seq(9,45,2) # Number of classifiers   
p <- seq(0.55,0.85,.05)  
plot(0,type='n',xlim=range(T),ylim=c(0.6,1),  
 xlab="Number of Classifiers",ylab="Probability of Majority Voting")  
for(i in 1:length(p)){  
 Prob\_MV <- NULL  
 for(j in 1:length(T)){  
 Prob\_MV[j] <- sum(dbinom(floor(T[j]/2+1):T[j],prob=p[i],size=T[j]))  
 }  
 points(T,Prob\_MV,col=i,"l")  
}



# When p < 0.5, ensemble accuracy goes to zero  
T <- seq(6,50,2)  
p <- seq(0.45,0.05,-0.05)  
plot(0,type='n',xlim=range(T),ylim=c(0,0.3),  
 xlab="Number of Classifiers",ylab="Probability of Majority Voting")  
for(i in 1:length(p)){  
 Prob\_MV <- NULL  
 for(j in 1:length(T)){  
 Prob\_MV[j] <- sum(dbinom(floor(T[j]/2+1):T[j],prob=p[i],size=T[j]))  
 }  
 points(T,Prob\_MV,col=i,"l")  
}



# When p = 0.5, no increase in accuracy, irrespective of T's  
p <- 0.5  
T <- seq(5,45,2)  
Prob\_MV <- NULL  
for(j in 1:length(T)){  
 Prob\_MV[j] <- sum(dbinom(floor(T[j]/2+1):T[j],prob=p,size=T[j]))  
}  
Prob\_MV

## [1] 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5  
## [18] 0.5 0.5 0.5 0.5

T <- seq(10,50,2)  
Prob\_MV <- NULL  
for(j in 1:length(T)){  
 Prob\_MV[j] <- (sum(dbinom(floor(T[j]/2):T[j],prob=p,size=T[j]))+  
 sum(dbinom(floor(T[j]/2+1):T[j],prob=p,size=T[j])))/2  
}  
Prob\_MV

## [1] 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5  
## [18] 0.5 0.5 0.5 0.5

# Different accuracies T's illustration  
# For simplicity, we set the number of classifiers at odd number  
# Each p\_i's greater than 0.5  
Get\_Prob <- function(Logical,Probability){  
 return(t(ifelse(Logical,Probability,1-Probability)))  
 }  
  
accuracy <- c(0.5,0.55,0.6,0.65,0.7,0.75,0.8,0.85,0.9)  
NT <- length(accuracy) # Number of classifiers  
APC <- expand.grid(rep(list(c(TRUE,FALSE)),NT)) # All possible combinations  
head(APC)

## Var1 Var2 Var3 Var4 Var5 Var6 Var7 Var8 Var9  
## 1 TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE  
## 2 FALSE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE  
## 3 TRUE FALSE TRUE TRUE TRUE TRUE TRUE TRUE TRUE  
## 4 FALSE FALSE TRUE TRUE TRUE TRUE TRUE TRUE TRUE  
## 5 TRUE TRUE FALSE TRUE TRUE TRUE TRUE TRUE TRUE  
## 6 FALSE TRUE FALSE TRUE TRUE TRUE TRUE TRUE TRUE

Elements\_Prob <- t(apply(APC,1,Get\_Prob,Probability=accuracy))  
head(Elements\_Prob)

## [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]  
## [1,] 0.5 0.55 0.6 0.65 0.7 0.75 0.8 0.85 0.9  
## [2,] 0.5 0.55 0.6 0.65 0.7 0.75 0.8 0.85 0.9  
## [3,] 0.5 0.45 0.6 0.65 0.7 0.75 0.8 0.85 0.9  
## [4,] 0.5 0.45 0.6 0.65 0.7 0.75 0.8 0.85 0.9  
## [5,] 0.5 0.55 0.4 0.65 0.7 0.75 0.8 0.85 0.9  
## [6,] 0.5 0.55 0.4 0.65 0.7 0.75 0.8 0.85 0.9

Events\_Prob <- apply(Elements\_Prob,1,prod)  
Majority\_Events <- (rowSums(APC)>NT/2)  
sum(Events\_Prob\*Majority\_Events)

## [1] 0.9112646

accuracy <- c(0.7,0.7,0.7,0.9,0.9)  
NT <- length(accuracy) # Number of classifiers  
APC <- expand.grid(rep(list(c(TRUE,FALSE)),NT)) # All possible combinations  
Elements\_Prob <- t(apply(APC,1,Get\_Prob,Probability=accuracy))  
Events\_Prob <- apply(Elements\_Prob,1,prod)  
Majority\_Events <- (rowSums(APC)>NT/2)  
sum(Events\_Prob\*Majority\_Events)

## [1] 0.93268

# Each p\_i's lesser than 0.5  
accuracy <- 1-c(0.5,0.55,0.6,0.65,0.7,0.75,0.8,0.85,0.9)  
NT <- length(accuracy) # Number of classifiers  
APC <- expand.grid(rep(list(c(TRUE,FALSE)),NT)) # All possible combinations  
head(APC)

## Var1 Var2 Var3 Var4 Var5 Var6 Var7 Var8 Var9  
## 1 TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE  
## 2 FALSE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE  
## 3 TRUE FALSE TRUE TRUE TRUE TRUE TRUE TRUE TRUE  
## 4 FALSE FALSE TRUE TRUE TRUE TRUE TRUE TRUE TRUE  
## 5 TRUE TRUE FALSE TRUE TRUE TRUE TRUE TRUE TRUE  
## 6 FALSE TRUE FALSE TRUE TRUE TRUE TRUE TRUE TRUE

Elements\_Prob <- t(apply(APC,1,Get\_Prob,Probability=accuracy))  
head(Elements\_Prob)

## [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]  
## [1,] 0.5 0.45 0.4 0.35 0.3 0.25 0.2 0.15 0.1  
## [2,] 0.5 0.45 0.4 0.35 0.3 0.25 0.2 0.15 0.1  
## [3,] 0.5 0.55 0.4 0.35 0.3 0.25 0.2 0.15 0.1  
## [4,] 0.5 0.55 0.4 0.35 0.3 0.25 0.2 0.15 0.1  
## [5,] 0.5 0.45 0.6 0.35 0.3 0.25 0.2 0.15 0.1  
## [6,] 0.5 0.45 0.6 0.35 0.3 0.25 0.2 0.15 0.1

Events\_Prob <- apply(Elements\_Prob,1,prod)  
Majority\_Events <- (rowSums(APC)>NT/2)  
sum(Events\_Prob\*Majority\_Events)

## [1] 0.08873544

# Mixture of p\_i's, some > 0.5, and some < 0.5  
Random\_Accuracy <- function() {  
 accuracy <- runif(9)  
 NT <- length(accuracy)   
 APC <- expand.grid(rep(list(c(TRUE,FALSE)),NT))   
 Elements\_Prob <- t(apply(APC,1,Get\_Prob,Probability=accuracy))  
 Events\_Prob <- apply(Elements\_Prob,1,prod)  
 Majority\_Events <- (rowSums(APC)>NT/2)  
 return(sum(Events\_Prob\*Majority\_Events))  
}  
Random\_Accuracy()

## [1] 0.4819363

Random\_Accuracy()

## [1] 0.6297693

Random\_Accuracy()

## [1] 0.4290092

Random\_Accuracy()

## [1] 0.3861596

Random\_Accuracy()

## [1] 0.4088063

Random\_Accuracy()

## [1] 0.6798881

Random\_Accuracy()

## [1] 0.7289509

Random\_Accuracy()

## [1] 0.4469414

Random\_Accuracy()

## [1] 0.8955832

Random\_Accuracy()

## [1] 0.2615583

# Voting for Classification  
load("../Data/GC2.RData")  
set.seed(12345)  
Train\_Test <- sample(c("Train","Test"),nrow(GC2),replace = TRUE,prob = c(0.7,0.3))  
GC2\_Train <- GC2[Train\_Test=="Train",]  
GC2\_TestX <- within(GC2[Train\_Test=="Test",],rm(good\_bad))  
GC2\_TestY <- GC2[Train\_Test=="Test","good\_bad"]  
GC2\_Formula <- as.formula("good\_bad~.")  
  
  
# RANDOM FOREST ANALYSIS  
GC2\_RF <- randomForest(GC2\_Formula,data=GC2\_Train,keep.inbag=TRUE,  
 ntree=500)  
  
# New data voting  
GC2\_RF\_Test\_Margin <- predict(GC2\_RF,newdata = GC2\_TestX,  
 type="class")  
GC2\_RF\_Test\_Predict <- predict(GC2\_RF,newdata=GC2\_TestX,  
 type="class",predict.all=TRUE  
 )  
# Majority Voting  
Row\_Count\_Max <- function(x) names(which.max(table(x)))  
Voting\_Predict <- apply(GC2\_RF\_Test\_Predict$individual,1,Row\_Count\_Max)  
head(Voting\_Predict);tail(Voting\_Predict)

## 1 2 3 4 9 10   
## "good" "bad" "good" "bad" "good" "bad"

## 974 980 983 984 988 996   
## "bad" "bad" "good" "good" "good" "good"

all(Voting\_Predict==GC2\_RF\_Test\_Predict$aggregate)

## [1] TRUE

sum(Voting\_Predict==GC2\_TestY)/313

## [1] 0.7795527

# Analyzing Accuracy of Trees of the Fitted Forest  
GC2\_RF\_Train\_Predict <- predict(GC2\_RF,newdata=GC2\_Train[,-20],  
 type="class",predict.all=TRUE)  
head(GC2\_RF\_Train\_Predict$individual[,c(1:5,496:500)])

## [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]   
## 5 "bad" "bad" "bad" "bad" "good" "bad" "bad" "bad" "bad" "bad"   
## 6 "good" "good" "good" "good" "good" "good" "bad" "bad" "bad" "good"  
## 7 "good" "good" "good" "good" "good" "good" "good" "good" "good" "good"  
## 8 "good" "good" "good" "good" "good" "bad" "good" "bad" "good" "good"  
## 11 "bad" "bad" "bad" "bad" "bad" "bad" "bad" "bad" "bad" "bad"   
## 12 "good" "bad" "bad" "bad" "bad" "good" "bad" "bad" "bad" "bad"

RF\_Tree\_Train\_Accuracy <- NULL  
for(i in 1:GC2\_RF$ntree){  
 RF\_Tree\_Train\_Accuracy[i] <- sum(GC2\_RF\_Train\_Predict$individual[,i]==  
 GC2\_Train$good\_bad)/nrow(GC2\_Train)  
}  
sort(RF\_Tree\_Train\_Accuracy)

## [1] 0.8340611 0.8369723 0.8384279 0.8398836 0.8398836 0.8413392 0.8413392  
## [8] 0.8413392 0.8413392 0.8427948 0.8427948 0.8442504 0.8442504 0.8442504  
## [15] 0.8442504 0.8442504 0.8457060 0.8457060 0.8457060 0.8457060 0.8457060  
## [22] 0.8471616 0.8471616 0.8471616 0.8486172 0.8486172 0.8486172 0.8486172  
## [29] 0.8500728 0.8500728 0.8500728 0.8500728 0.8500728 0.8500728 0.8515284  
## [36] 0.8515284 0.8515284 0.8515284 0.8515284 0.8515284 0.8515284 0.8515284  
## [43] 0.8515284 0.8515284 0.8515284 0.8515284 0.8515284 0.8529840 0.8529840  
## [50] 0.8529840 0.8529840 0.8529840 0.8529840 0.8529840 0.8529840 0.8544396  
## [57] 0.8544396 0.8544396 0.8544396 0.8544396 0.8544396 0.8544396 0.8544396  
## [64] 0.8544396 0.8558952 0.8558952 0.8558952 0.8558952 0.8558952 0.8558952  
## [71] 0.8558952 0.8558952 0.8558952 0.8558952 0.8558952 0.8558952 0.8558952  
## [78] 0.8558952 0.8558952 0.8558952 0.8558952 0.8558952 0.8558952 0.8558952  
## [85] 0.8558952 0.8573508 0.8573508 0.8573508 0.8573508 0.8573508 0.8573508  
## [92] 0.8573508 0.8573508 0.8573508 0.8573508 0.8573508 0.8573508 0.8573508  
## [99] 0.8573508 0.8573508 0.8573508 0.8573508 0.8573508 0.8588064 0.8588064  
## [106] 0.8588064 0.8588064 0.8588064 0.8588064 0.8588064 0.8588064 0.8588064  
## [113] 0.8588064 0.8588064 0.8588064 0.8588064 0.8588064 0.8588064 0.8588064  
## [120] 0.8602620 0.8602620 0.8602620 0.8602620 0.8602620 0.8602620 0.8602620  
## [127] 0.8602620 0.8602620 0.8602620 0.8602620 0.8602620 0.8602620 0.8602620  
## [134] 0.8602620 0.8602620 0.8602620 0.8602620 0.8602620 0.8602620 0.8602620  
## [141] 0.8602620 0.8602620 0.8602620 0.8617176 0.8617176 0.8617176 0.8617176  
## [148] 0.8617176 0.8617176 0.8617176 0.8617176 0.8617176 0.8617176 0.8617176  
## [155] 0.8617176 0.8617176 0.8617176 0.8617176 0.8617176 0.8617176 0.8617176  
## [162] 0.8617176 0.8631732 0.8631732 0.8631732 0.8631732 0.8631732 0.8631732  
## [169] 0.8631732 0.8631732 0.8631732 0.8631732 0.8631732 0.8631732 0.8631732  
## [176] 0.8646288 0.8646288 0.8646288 0.8646288 0.8646288 0.8646288 0.8646288  
## [183] 0.8646288 0.8646288 0.8646288 0.8646288 0.8646288 0.8646288 0.8646288  
## [190] 0.8646288 0.8646288 0.8646288 0.8660844 0.8660844 0.8660844 0.8660844  
## [197] 0.8660844 0.8660844 0.8660844 0.8660844 0.8660844 0.8660844 0.8660844  
## [204] 0.8660844 0.8660844 0.8660844 0.8660844 0.8660844 0.8660844 0.8660844  
## [211] 0.8660844 0.8660844 0.8660844 0.8660844 0.8660844 0.8660844 0.8660844  
## [218] 0.8660844 0.8660844 0.8660844 0.8675400 0.8675400 0.8675400 0.8675400  
## [225] 0.8675400 0.8675400 0.8675400 0.8675400 0.8675400 0.8675400 0.8675400  
## [232] 0.8675400 0.8675400 0.8675400 0.8675400 0.8675400 0.8675400 0.8675400  
## [239] 0.8675400 0.8675400 0.8675400 0.8675400 0.8675400 0.8675400 0.8675400  
## [246] 0.8675400 0.8675400 0.8689956 0.8689956 0.8689956 0.8689956 0.8689956  
## [253] 0.8689956 0.8689956 0.8689956 0.8689956 0.8689956 0.8689956 0.8689956  
## [260] 0.8689956 0.8689956 0.8689956 0.8689956 0.8689956 0.8689956 0.8689956  
## [267] 0.8704512 0.8704512 0.8704512 0.8704512 0.8704512 0.8704512 0.8704512  
## [274] 0.8704512 0.8704512 0.8704512 0.8704512 0.8704512 0.8704512 0.8704512  
## [281] 0.8704512 0.8704512 0.8704512 0.8704512 0.8704512 0.8704512 0.8719068  
## [288] 0.8719068 0.8719068 0.8719068 0.8719068 0.8719068 0.8719068 0.8719068  
## [295] 0.8719068 0.8719068 0.8719068 0.8719068 0.8719068 0.8719068 0.8719068  
## [302] 0.8719068 0.8719068 0.8719068 0.8719068 0.8719068 0.8719068 0.8719068  
## [309] 0.8719068 0.8719068 0.8719068 0.8719068 0.8719068 0.8719068 0.8719068  
## [316] 0.8719068 0.8719068 0.8719068 0.8719068 0.8719068 0.8733624 0.8733624  
## [323] 0.8733624 0.8733624 0.8733624 0.8733624 0.8733624 0.8733624 0.8733624  
## [330] 0.8733624 0.8733624 0.8733624 0.8733624 0.8733624 0.8733624 0.8733624  
## [337] 0.8733624 0.8733624 0.8733624 0.8733624 0.8733624 0.8733624 0.8748180  
## [344] 0.8748180 0.8748180 0.8748180 0.8748180 0.8748180 0.8748180 0.8748180  
## [351] 0.8748180 0.8748180 0.8748180 0.8748180 0.8748180 0.8748180 0.8748180  
## [358] 0.8748180 0.8748180 0.8748180 0.8748180 0.8748180 0.8748180 0.8748180  
## [365] 0.8748180 0.8748180 0.8748180 0.8748180 0.8748180 0.8748180 0.8762737  
## [372] 0.8762737 0.8762737 0.8762737 0.8762737 0.8762737 0.8762737 0.8762737  
## [379] 0.8762737 0.8762737 0.8762737 0.8762737 0.8762737 0.8762737 0.8762737  
## [386] 0.8762737 0.8762737 0.8762737 0.8762737 0.8762737 0.8762737 0.8762737  
## [393] 0.8762737 0.8762737 0.8777293 0.8777293 0.8777293 0.8777293 0.8777293  
## [400] 0.8777293 0.8777293 0.8777293 0.8777293 0.8777293 0.8777293 0.8777293  
## [407] 0.8777293 0.8777293 0.8777293 0.8777293 0.8791849 0.8791849 0.8791849  
## [414] 0.8791849 0.8791849 0.8791849 0.8791849 0.8791849 0.8791849 0.8791849  
## [421] 0.8791849 0.8791849 0.8791849 0.8791849 0.8791849 0.8791849 0.8791849  
## [428] 0.8791849 0.8791849 0.8806405 0.8806405 0.8806405 0.8806405 0.8806405  
## [435] 0.8806405 0.8806405 0.8806405 0.8806405 0.8806405 0.8806405 0.8806405  
## [442] 0.8806405 0.8820961 0.8820961 0.8820961 0.8820961 0.8820961 0.8820961  
## [449] 0.8820961 0.8820961 0.8820961 0.8820961 0.8820961 0.8820961 0.8820961  
## [456] 0.8820961 0.8820961 0.8820961 0.8835517 0.8835517 0.8835517 0.8835517  
## [463] 0.8835517 0.8835517 0.8835517 0.8835517 0.8835517 0.8835517 0.8850073  
## [470] 0.8850073 0.8850073 0.8850073 0.8850073 0.8864629 0.8864629 0.8864629  
## [477] 0.8864629 0.8864629 0.8864629 0.8864629 0.8864629 0.8879185 0.8879185  
## [484] 0.8879185 0.8879185 0.8893741 0.8893741 0.8893741 0.8893741 0.8908297  
## [491] 0.8908297 0.8908297 0.8908297 0.8908297 0.8922853 0.8922853 0.8937409  
## [498] 0.8937409 0.8966521 0.8981077

# Bagging ANALYSIS  
GC2\_Bagg <- randomForest(GC2\_Formula,data=GC2\_Train,keep.inbag=TRUE,  
 mtry=ncol(GC2\_TestX),ntree=500)  
GC2\_Bagg\_Test\_Predict <- predict(GC2\_Bagg,newdata=GC2\_TestX,  
 type="class",predict.all=TRUE)  
GC2\_Bagg\_Train\_Predict <- predict(GC2\_Bagg,newdata=GC2\_Train[,-20],  
 type="class",predict.all=TRUE)  
  
Bagg\_Tree\_Train\_Accuracy <- NULL  
for(i in 1:GC2\_Bagg$ntree){  
 Bagg\_Tree\_Train\_Accuracy[i] <- sum(GC2\_Bagg\_Train\_Predict$individual[,i]==  
 GC2\_Train$good\_bad)/nrow(GC2\_Train)  
}  
sort(Bagg\_Tree\_Train\_Accuracy)

## [1] 0.8369723 0.8384279 0.8413392 0.8457060 0.8457060 0.8471616 0.8471616  
## [8] 0.8471616 0.8471616 0.8486172 0.8500728 0.8500728 0.8500728 0.8500728  
## [15] 0.8500728 0.8515284 0.8529840 0.8529840 0.8529840 0.8529840 0.8544396  
## [22] 0.8544396 0.8544396 0.8544396 0.8544396 0.8544396 0.8544396 0.8544396  
## [29] 0.8544396 0.8558952 0.8558952 0.8558952 0.8558952 0.8558952 0.8558952  
## [36] 0.8558952 0.8558952 0.8558952 0.8558952 0.8573508 0.8573508 0.8573508  
## [43] 0.8573508 0.8573508 0.8573508 0.8588064 0.8588064 0.8588064 0.8588064  
## [50] 0.8588064 0.8588064 0.8588064 0.8588064 0.8588064 0.8602620 0.8602620  
## [57] 0.8602620 0.8602620 0.8602620 0.8602620 0.8602620 0.8602620 0.8602620  
## [64] 0.8602620 0.8617176 0.8617176 0.8617176 0.8617176 0.8617176 0.8617176  
## [71] 0.8617176 0.8617176 0.8617176 0.8617176 0.8617176 0.8617176 0.8617176  
## [78] 0.8617176 0.8617176 0.8631732 0.8631732 0.8631732 0.8631732 0.8631732  
## [85] 0.8631732 0.8631732 0.8631732 0.8631732 0.8631732 0.8631732 0.8631732  
## [92] 0.8631732 0.8631732 0.8631732 0.8631732 0.8631732 0.8631732 0.8631732  
## [99] 0.8631732 0.8631732 0.8646288 0.8646288 0.8646288 0.8646288 0.8646288  
## [106] 0.8646288 0.8646288 0.8646288 0.8646288 0.8646288 0.8646288 0.8646288  
## [113] 0.8646288 0.8660844 0.8660844 0.8660844 0.8660844 0.8660844 0.8660844  
## [120] 0.8660844 0.8660844 0.8660844 0.8660844 0.8660844 0.8660844 0.8660844  
## [127] 0.8660844 0.8660844 0.8660844 0.8660844 0.8660844 0.8660844 0.8675400  
## [134] 0.8675400 0.8675400 0.8675400 0.8675400 0.8675400 0.8675400 0.8675400  
## [141] 0.8675400 0.8675400 0.8675400 0.8675400 0.8675400 0.8675400 0.8675400  
## [148] 0.8675400 0.8675400 0.8675400 0.8675400 0.8675400 0.8689956 0.8689956  
## [155] 0.8689956 0.8689956 0.8689956 0.8689956 0.8689956 0.8689956 0.8689956  
## [162] 0.8689956 0.8689956 0.8689956 0.8689956 0.8689956 0.8689956 0.8689956  
## [169] 0.8689956 0.8689956 0.8689956 0.8689956 0.8689956 0.8689956 0.8689956  
## [176] 0.8689956 0.8689956 0.8689956 0.8689956 0.8689956 0.8689956 0.8689956  
## [183] 0.8704512 0.8704512 0.8704512 0.8704512 0.8704512 0.8704512 0.8704512  
## [190] 0.8704512 0.8704512 0.8704512 0.8704512 0.8704512 0.8704512 0.8704512  
## [197] 0.8704512 0.8704512 0.8704512 0.8704512 0.8704512 0.8704512 0.8704512  
## [204] 0.8704512 0.8704512 0.8704512 0.8704512 0.8719068 0.8719068 0.8719068  
## [211] 0.8719068 0.8719068 0.8719068 0.8719068 0.8719068 0.8719068 0.8719068  
## [218] 0.8719068 0.8719068 0.8719068 0.8719068 0.8719068 0.8719068 0.8719068  
## [225] 0.8719068 0.8719068 0.8719068 0.8719068 0.8719068 0.8719068 0.8719068  
## [232] 0.8719068 0.8719068 0.8719068 0.8719068 0.8719068 0.8733624 0.8733624  
## [239] 0.8733624 0.8733624 0.8733624 0.8733624 0.8733624 0.8733624 0.8733624  
## [246] 0.8733624 0.8733624 0.8733624 0.8733624 0.8733624 0.8733624 0.8733624  
## [253] 0.8733624 0.8733624 0.8733624 0.8733624 0.8733624 0.8748180 0.8748180  
## [260] 0.8748180 0.8748180 0.8748180 0.8748180 0.8748180 0.8748180 0.8748180  
## [267] 0.8748180 0.8748180 0.8748180 0.8748180 0.8748180 0.8748180 0.8748180  
## [274] 0.8748180 0.8748180 0.8748180 0.8748180 0.8748180 0.8748180 0.8748180  
## [281] 0.8748180 0.8748180 0.8762737 0.8762737 0.8762737 0.8762737 0.8762737  
## [288] 0.8762737 0.8762737 0.8762737 0.8762737 0.8762737 0.8762737 0.8762737  
## [295] 0.8762737 0.8762737 0.8762737 0.8762737 0.8762737 0.8762737 0.8762737  
## [302] 0.8762737 0.8762737 0.8762737 0.8762737 0.8762737 0.8762737 0.8762737  
## [309] 0.8762737 0.8777293 0.8777293 0.8777293 0.8777293 0.8777293 0.8777293  
## [316] 0.8777293 0.8777293 0.8777293 0.8777293 0.8777293 0.8777293 0.8777293  
## [323] 0.8777293 0.8777293 0.8777293 0.8777293 0.8777293 0.8777293 0.8777293  
## [330] 0.8791849 0.8791849 0.8791849 0.8791849 0.8791849 0.8791849 0.8791849  
## [337] 0.8791849 0.8791849 0.8791849 0.8791849 0.8791849 0.8791849 0.8791849  
## [344] 0.8791849 0.8791849 0.8791849 0.8791849 0.8791849 0.8791849 0.8791849  
## [351] 0.8791849 0.8791849 0.8791849 0.8791849 0.8806405 0.8806405 0.8806405  
## [358] 0.8806405 0.8806405 0.8806405 0.8806405 0.8806405 0.8806405 0.8806405  
## [365] 0.8806405 0.8806405 0.8806405 0.8806405 0.8806405 0.8806405 0.8806405  
## [372] 0.8806405 0.8806405 0.8806405 0.8806405 0.8806405 0.8806405 0.8806405  
## [379] 0.8820961 0.8820961 0.8820961 0.8820961 0.8820961 0.8820961 0.8820961  
## [386] 0.8820961 0.8820961 0.8820961 0.8820961 0.8820961 0.8820961 0.8820961  
## [393] 0.8835517 0.8835517 0.8835517 0.8835517 0.8835517 0.8835517 0.8835517  
## [400] 0.8835517 0.8835517 0.8835517 0.8835517 0.8835517 0.8835517 0.8835517  
## [407] 0.8835517 0.8835517 0.8835517 0.8835517 0.8835517 0.8835517 0.8850073  
## [414] 0.8850073 0.8850073 0.8850073 0.8850073 0.8850073 0.8850073 0.8850073  
## [421] 0.8850073 0.8850073 0.8850073 0.8850073 0.8850073 0.8850073 0.8850073  
## [428] 0.8850073 0.8850073 0.8850073 0.8850073 0.8850073 0.8864629 0.8864629  
## [435] 0.8864629 0.8864629 0.8864629 0.8864629 0.8864629 0.8864629 0.8864629  
## [442] 0.8864629 0.8864629 0.8879185 0.8879185 0.8879185 0.8879185 0.8879185  
## [449] 0.8879185 0.8879185 0.8879185 0.8879185 0.8879185 0.8879185 0.8879185  
## [456] 0.8879185 0.8879185 0.8879185 0.8879185 0.8879185 0.8893741 0.8893741  
## [463] 0.8893741 0.8893741 0.8893741 0.8893741 0.8893741 0.8893741 0.8908297  
## [470] 0.8908297 0.8908297 0.8908297 0.8908297 0.8908297 0.8922853 0.8922853  
## [477] 0.8922853 0.8922853 0.8922853 0.8922853 0.8922853 0.8922853 0.8922853  
## [484] 0.8937409 0.8937409 0.8937409 0.8951965 0.8951965 0.8951965 0.8951965  
## [491] 0.8966521 0.8966521 0.8966521 0.8966521 0.8966521 0.8981077 0.8995633  
## [498] 0.8995633 0.9024745 0.9097525

# Weighted Voting with Random Forest  
RF\_Weights <- RF\_Tree\_Train\_Accuracy/sum(RF\_Tree\_Train\_Accuracy)  
Bagg\_Weights <- Bagg\_Tree\_Train\_Accuracy/sum(Bagg\_Tree\_Train\_Accuracy)  
RF\_Weighted\_Vote <- data.frame(matrix(0,nrow(GC2\_TestX),ncol=3))  
names(RF\_Weighted\_Vote) <- c("Good\_Weight","Bad\_Weight","Prediction")  
for(i in 1:nrow(RF\_Weighted\_Vote)){  
 RF\_Weighted\_Vote$Good\_Weight[i] <-   
 sum((GC2\_RF\_Test\_Predict$individual[i,]=="good")\*RF\_Weights)  
 RF\_Weighted\_Vote$Bad\_Weight[i] <-   
 sum((GC2\_RF\_Test\_Predict$individual[i,]=="bad")\*RF\_Weights)  
 RF\_Weighted\_Vote$Prediction[i] <- c("good","bad")[which.max(RF\_Weighted\_Vote[i,1:2])]  
}  
head(RF\_Weighted\_Vote,10)

## Good\_Weight Bad\_Weight Prediction  
## 1 0.8301541 0.16984588 good  
## 2 0.3260033 0.67399668 bad  
## 3 0.8397035 0.16029651 good  
## 4 0.4422527 0.55774733 bad  
## 5 0.9420565 0.05794355 good  
## 6 0.2378956 0.76210442 bad  
## 7 0.4759756 0.52402435 bad  
## 8 0.7443038 0.25569624 good  
## 9 0.8120180 0.18798195 good  
## 10 0.7799587 0.22004126 good

# Weighted Voting with Bagging  
Bagg\_Weights <- Bagg\_Tree\_Train\_Accuracy/sum(Bagg\_Tree\_Train\_Accuracy)  
Bagg\_Weights <- Bagg\_Tree\_Train\_Accuracy/sum(Bagg\_Tree\_Train\_Accuracy)  
Bagg\_Weighted\_Vote <- data.frame(matrix(0,nrow(GC2\_TestX),ncol=3))  
names(Bagg\_Weighted\_Vote) <- c("Good\_Weight","Bad\_Weight","Prediction")  
for(i in 1:nrow(Bagg\_Weighted\_Vote)){  
 Bagg\_Weighted\_Vote$Good\_Weight[i] <-   
 sum((GC2\_Bagg\_Test\_Predict$individual[i,]=="good")\*Bagg\_Weights)  
 Bagg\_Weighted\_Vote$Bad\_Weight[i] <-   
 sum((GC2\_Bagg\_Test\_Predict$individual[i,]=="bad")\*Bagg\_Weights)  
 Bagg\_Weighted\_Vote$Prediction[i] <- c("good","bad")[which.max(Bagg\_Weighted\_Vote[i,1:2])]  
}  
head(Bagg\_Weighted\_Vote,10)

## Good\_Weight Bad\_Weight Prediction  
## 1 0.9279982 0.07200181 good  
## 2 0.1634505 0.83654949 bad  
## 3 0.8219618 0.17803818 good  
## 4 0.4724477 0.52755226 bad  
## 5 0.9619528 0.03804725 good  
## 6 0.1698628 0.83013718 bad  
## 7 0.4540574 0.54594265 bad  
## 8 0.7883772 0.21162281 good  
## 9 0.8301772 0.16982283 good  
## 10 0.7585720 0.24142804 good

# Averaging for Regression Problems  
  
  
  
  
  
# Stack Ensemble Examples