Various R Machine learning Algorithms

Rajdeep Mondal

April 18, 2016

1. Data Manipulation:
2. Read in data (AngleClosure.csv), (b) delete the columns corresponding to factor variables EYE, GENDER, and ETHNIC, and (c) delete rows of the dataset which have any missing values.

# Read in angle closures data  
myData=read.csv("AngleClosure.csv",header=TRUE)  
  
myvars = names(myData) %in% c('EYE', 'GENDER', 'ETHNIC', 'HGT', 'WT', 'ASPH', 'ACYL', 'SE', 'AXL', 'CACD', 'AGE', 'CCT.OD', 'PCCURV\_mm')  
  
myData = myData[!myvars]  
  
#myData = myData[complete.cases(myData),]  
  
for(C in names(myData)){  
 myData = myData[!myData[C] %in% c(".", "NA"), ]  
}  
myData$ANGLE.CLOSURE = as.numeric(myData$ANGLE.CLOSURE)-1  
  
options(warn=-1)

1. Develop Prediction Models: Develop 5 prediction models for angle closure glaucoma. Your suite of prediction models should include at least 3 of a (i) support vector machine (e1071, kernlab, klaR, svmpath),
2. neural network (nnet, neuralnet), (iii) random forest (randomForest, randomForestSRC),
3. boosted model (ada, adabag, mboost, gbm), and (v) logistic regression model with AIC or BIC variable selection (glm, family="binomial" with step), and potentially 2 additional prediction models of your choosing. Omit the variables HGT, WT, ASPH, ACYL, SE, AXL, CACD, AGE, CCT.OD, and PCCURV\_mm when building the prediction models. For generating prediction models, you may use "canned" R functionality, your own code, or code freely available on the internet (include URL/reference). All actively chosen tuning parameter selections should be justified via cross-validation.

library(e1071)  
library(pROC)

## Type 'citation("pROC")' for a citation.  
##   
## Attaching package: 'pROC'  
##   
## The following objects are masked from 'package:stats':  
##   
## cov, smooth, var

library(nnet)  
library(randomForest)

## randomForest 4.6-12  
## Type rfNews() to see new features/changes/bug fixes.

#library(MASS)  
library(adabag)

## Loading required package: rpart  
## Loading required package: mlbench  
## Loading required package: caret  
## Loading required package: lattice  
## Loading required package: ggplot2

#detach("package:MASS", unload=TRUE)

Model 1: SVM

set.seed(123)  
  
index = 1:nrow(myData)  
testsz = trunc(0.1 \* nrow(myData))  
vg\_vgamma\_auc = matrix(NA, 25,3)  
  
vgamma=10^seq(-4,0,1)  
vcost =10^seq(1,5,1)  
  
svm.auc.max = 0  
gammamax = 0  
costmax = 0  
iteri = 0  
svm.model.best = NULL  
  
for (mgamma in vgamma){  
 for (mcost in vcost){  
 iteri = iteri +1  
   
 svm.auc = 0  
 for (i in seq(10)){  
 testn = sample(index, size = testsz)  
   
 trainsamp = myData[-testn,]  
 testsamp = myData[testn,]  
   
 svm.model = svm(ANGLE.CLOSURE ~ ., data = trainsamp, gamma = mgamma, cost = mcost)  
 svm.pred = predict(svm.model, testsamp, type="response")  
 svm.auc = svm.auc + auc(roc(testsamp$ANGLE.CLOSURE, svm.pred))  
 }  
   
 svm.auc = svm.auc/10  
 cat("Processing...:", iteri, svm.auc, "\n")  
 vg\_vgamma\_auc[iteri,1] = svm.auc  
 vg\_vgamma\_auc[iteri,2] = mgamma  
 vg\_vgamma\_auc[iteri,3] = mcost  
   
 if(svm.auc > svm.auc.max){  
 gammamax = mgamma  
 costmax = mcost  
 svm.auc.max = svm.auc  
 svm.model.best = svm.model  
 }  
   
 }  
}

## Processing...: 1 0.9452239   
## Processing...: 2 0.9476128   
## Processing...: 3 0.9418338   
## Processing...: 4 0.9444899   
## Processing...: 5 0.9505298   
## Processing...: 6 0.9488201   
## Processing...: 7 0.9516297   
## Processing...: 8 0.9505865   
## Processing...: 9 0.9412623   
## Processing...: 10 0.932702   
## Processing...: 11 0.9520549   
## Processing...: 12 0.925532   
## Processing...: 13 0.9348441   
## Processing...: 14 0.9185572   
## Processing...: 15 0.8862495   
## Processing...: 16 0.911486   
## Processing...: 17 0.8985985   
## Processing...: 18 0.8080333   
## Processing...: 19 0.7092247   
## Processing...: 20 0.7286897   
## Processing...: 21 0.8784936   
## Processing...: 22 0.8791736   
## Processing...: 23 0.8701451   
## Processing...: 24 0.8771185   
## Processing...: 25 0.8805373

cat("Max AUC Gamma for SVM:",gammamax,"\n")

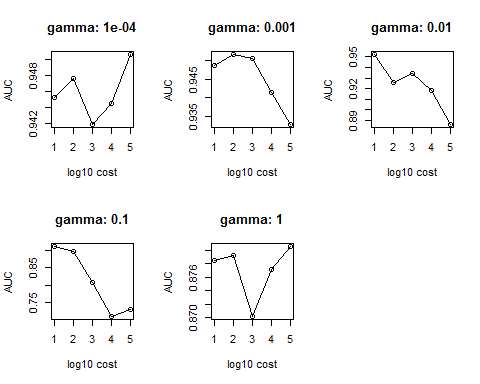
**## Max AUC Gamma for SVM: 0.01**

cat("Max AUC Cost for SVM:",costmax,"\n")

**## Max AUC Cost for SVM: 10**

Visualization for Q6

colnames(vg\_vgamma\_auc) = c("AUC", "gamma", "cost")  
  
par(mfrow=c(2,3))  
  
for (i in seq(5)){  
starti = (i-1)\*5+1  
endi = i\*5  
x = log10(vg\_vgamma\_auc[starti:endi,"cost"])  
y = vg\_vgamma\_auc[starti:endi,"AUC"]  
plot(x, y, ylab="AUC", xlab="log10 cost", main = paste("gamma:", vg\_vgamma\_auc[starti,"gamma"]))  
lines(x, y, type='l')  
}



Model 2: NN

set.seed(123)  
myLambdas=10^seq(-1,5,1)  
  
nnet.auc.max = 0  
iteri = 0  
lambdamax = 0  
nnet.model = NULL  
vlambda\_auc = matrix(NA, 7, 2)  
  
for (lambda in myLambdas){  
 nnet.auc = 0  
 iteri = iteri +1  
   
 for (i in seq(10)){  
 testn = sample(index, size = testsz)  
   
 trainsamp = myData[-testn,]  
 testsamp = myData[testn,]  
   
 nnet.model = nnet(ANGLE.CLOSURE ~ ., data = trainsamp, size=10, decay=lambda, MaxNWts=250, trace = FALSE)  
 nnet.pred = predict(nnet.model, testsamp)  
 nnet.auc = nnet.auc + auc(roc(testsamp$ANGLE.CLOSURE, nnet.pred))  
 }  
   
 nnet.auc = nnet.auc/10  
 vlambda\_auc[iteri,1] = nnet.auc  
 vlambda\_auc[iteri,2] = lambda  
   
 cat("Processing...:", iteri, nnet.auc, "\n")  
 if(nnet.auc > nnet.auc.max){  
 lambdamax = lambda  
 nnet.auc.max = nnet.auc  
 nnet.model.best = nnet.model  
 }  
}

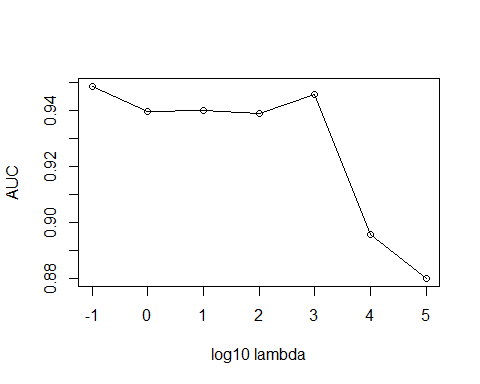
## Processing...: 1 0.9485795   
## Processing...: 2 0.9397947   
## Processing...: 3 0.9398551   
## Processing...: 4 0.9388905   
## Processing...: 5 0.9457621   
## Processing...: 6 0.8958468   
## Processing...: 7 0.8800817

cat("Max AUC Lambda for NN:",lambdamax,"\n")

**## Max AUC Lambda for NN: 0.1**

Visualization for Q6

x = log10(vlambda\_auc[,2])  
y = vlambda\_auc[,1]  
plot(x, y, xlab = "log10 lambda", ylab="AUC")  
lines(x, y, type='l')



Model 3: Random Forest

set.seed(123)  
  
mTrys=seq(1,5,1)  
  
randomForest.auc.max = 0  
iteri = 0  
trymax = 0  
randomForest.model.best = NULL  
  
vtry\_auc = matrix(NA, 5, 2)  
  
  
for (mt in mTrys){  
 randomForest.auc = 0  
 iteri = iteri +1  
   
 for (i in seq(10)){  
 testn = sample(index, size = testsz)  
   
 trainsamp = myData[-testn,]  
 testsamp = myData[testn,]  
   
 randomForest.model = randomForest(ANGLE.CLOSURE ~ ., data = trainsamp, mtry = mt)  
 randomForest.pred = predict(randomForest.model, testsamp, type = "response")  
 randomForest.auc = randomForest.auc + auc(roc(testsamp$ANGLE.CLOSURE, randomForest.pred))  
 }  
   
 randomForest.auc = randomForest.auc/10  
   
   
 cat("Processing...:", iteri, randomForest.auc, "\n")  
 vtry\_auc[iteri,1] = randomForest.auc  
 vtry\_auc[iteri,2] = mt  
   
 if(randomForest.auc > randomForest.auc.max){  
 trymax = mt  
 randomForest.auc.max = randomForest.auc  
 randomForest.model.best = randomForest.model  
 }  
}

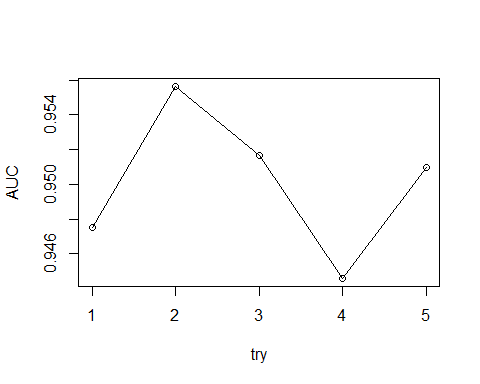
## Processing...: 1 0.9475248   
## Processing...: 2 0.955639   
## Processing...: 3 0.951685   
## Processing...: 4 0.9445819   
## Processing...: 5 0.9509642

cat("Max AUC Try for Random Forest:",trymax,"\n")

**## Max AUC Try for Random Forest: 2**

Visualization for Q6

x = vtry\_auc[,2]  
y = vtry\_auc[,1]  
plot(x, y, xlab = "try", ylab="AUC")  
lines(x, y, type='l')



Model 4: Boosted model

set.seed(123)  
myMs=c(50,100,200,500,1000)  
  
boosting.auc.max = 0  
iteri = 0  
mmax = 0  
boosting.model.best = NULL  
vm\_auc = matrix(NA, 5, 2)  
  
myData$ANGLE.CLOSURE = as.factor(myData$ANGLE.CLOSURE)  
  
for (m in myMs){  
 boosting.auc = 0  
 iteri = iteri +1  
   
 for (i in seq(10)){  
 testn = sample(index, size = testsz)  
   
 trainsamp = myData[-testn,]  
 testsamp = myData[testn,]  
 boosting.model = boosting(ANGLE.CLOSURE ~ ., data = trainsamp, mfinal=m, coeflearn="Freund",  
 control = rpart.control(maxdepth = 10))  
   
 boosting.pred = predict(boosting.model, testsamp)$prob[,1]  
 boosting.auc = boosting.auc + auc(roc(testsamp$ANGLE.CLOSURE, boosting.pred))  
 }  
   
 boosting.auc = boosting.auc/10  
   
 vm\_auc[iteri,1] = boosting.auc  
 vm\_auc[iteri,2] = m  
   
   
 cat("Processing...:", iteri, boosting.auc, "\n")  
   
 if(boosting.auc > boosting.auc.max){  
 mmax = m  
 boosting.auc.max = boosting.auc  
 boosting.model.best = boosting.model  
 }  
}

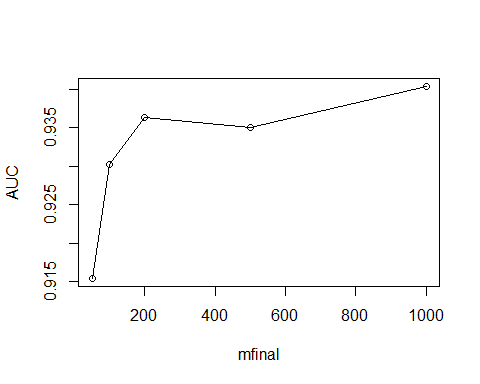
## Processing...: 1 0.9154242   
## Processing...: 2 0.9301887   
## Processing...: 3 0.9363499   
## Processing...: 4 0.9350594   
## Processing...: 5 0.9403988

cat("Max AUC mfinal for Adboost:",mmax,"\n")

**## Max AUC mfinal for Adboost: 1000**

Visualization for Q6

x = vm\_auc[,2]  
y = vm\_auc[,1]  
plot(x, y, xlab = "mfinal", ylab="AUC")  
lines(x, y, type='l')



Model 5: Logistic regression

set.seed(123)  
myEps=10^seq(-5,0,1)  
  
glm.auc.max = 0  
iteri = 0  
epsmax = 0  
glm.model.best = NULL  
glm.model.best

## NULL

myData$ANGLE.CLOSURE = as.numeric(myData$ANGLE.CLOSURE)-1  
veps\_auc = matrix(NA, 6, 2)  
  
for (meps in myEps){  
 glm.auc = 0  
 iteri = iteri +1  
   
 for (i in seq(10)){  
 testn = sample(index, size = testsz)  
   
 trainsamp = myData[-testn,]  
 testsamp = myData[testn,]  
   
 glm.full.model = glm(ANGLE.CLOSURE ~ ., data = trainsamp, family = "binomial", control = glm.control(epsilon=meps))  
 glm.null.model = glm(ANGLE.CLOSURE ~ 1, data = trainsamp, family = "binomial", control = glm.control(epsilon=meps))  
   
 #Forward stepwise AIC  
 glm.model = step(glm.null.model, scope=list(lower=glm.null.model, upper=glm.full.model), direction="forward", trace=FALSE)  
   
 glm.pred = predict(glm.model, testsamp, type="response")  
 glm.auc = glm.auc + auc(roc(testsamp$ANGLE.CLOSURE, glm.pred))  
 }  
   
 glm.auc = glm.auc/10  
   
 veps\_auc[iteri,1] = glm.auc  
 veps\_auc[iteri,2] = meps  
   
 cat("Processing...:", iteri, glm.auc, "\n")  
 if(glm.auc > glm.auc.max){  
 epsmax = meps  
 glm.auc.max = glm.auc  
 glm.model.best = glm.model  
 }  
   
}

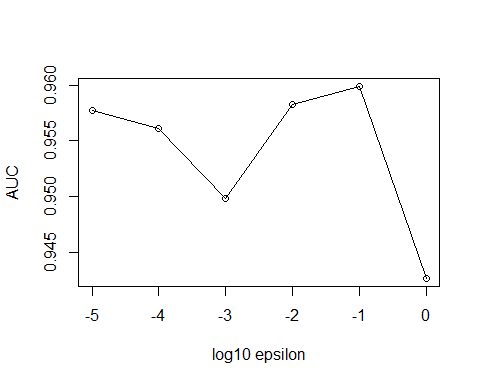
## Processing...: 1 0.9576768   
## Processing...: 2 0.956112   
## Processing...: 3 0.9498378   
## Processing...: 4 0.9582277   
## Processing...: 5 0.9598782   
## Processing...: 6 0.9426774

cat("Max AUC epsilon for Logit:", epsmax,"\n")

**## Max AUC epsilon for Logit: 0.1**

Visualization for Q6

x = log10(veps\_auc[,2])  
y = veps\_auc[,1]  
plot(x, y, xlab = "log10 epsilon", ylab="AUC")  
lines(x, y, type='l')



1. Generate 2 stacked ensemble models based upon the 5 selected (with optimized tuning parameters) prediction models with weights wm; m = 1,....,5 minimizing...
2. Unconstrained least squares solution

set.seed(123)  
all.pred = NULL  
for (i in seq(10)){ ###RAJDEEP change to 10  
 cat("Processing... i:", i, "\n")  
 testn = sample(index, size = testsz)  
 trainsamp = myData[-testn,]  
 testsamp = myData[testn,]  
   
 #SVM  
 cat("SVM \n")  
 svm.model = svm(ANGLE.CLOSURE ~ ., data = trainsamp, gamma = gammamax, cost = costmax)  
 svm.pred = predict(svm.model, testsamp)  
   
 #NN  
 cat("NN \n")  
 nnet.model = nnet(ANGLE.CLOSURE ~ ., data = trainsamp, size=10, decay=lambdamax, MaxNWts=250, trace = FALSE)  
 nnet.pred = predict(nnet.model, testsamp)  
   
 #RF  
 cat("RF \n")  
 randomForest.model = randomForest(ANGLE.CLOSURE ~ ., data = trainsamp, mtry = trymax)  
 randomForest.pred = predict(randomForest.model, testsamp)  
   
 #Boosted model  
 cat("Adaboost \n")  
 trainsamp$ANGLE.CLOSURE = as.factor(trainsamp$ANGLE.CLOSURE)  
 testsamp$ANGLE.CLOSURE = as.factor(testsamp$ANGLE.CLOSURE)  
 boosting.model = boosting(ANGLE.CLOSURE ~ ., data = trainsamp, mfinal=mmax, coeflearn="Freund",  
 control = rpart.control(maxdepth = 10))  
 boosting.pred = predict(boosting.model, testsamp)$prob[,1]  
 trainsamp$ANGLE.CLOSURE = as.numeric(trainsamp$ANGLE.CLOSURE)-1  
 testsamp$ANGLE.CLOSURE = as.numeric(testsamp$ANGLE.CLOSURE)-1  
   
 #Logistic regression  
 cat("Logit \n")  
 glm.full.model = glm(ANGLE.CLOSURE ~ ., data = trainsamp, family = "binomial",   
 control = glm.control(epsilon=epsmax))  
 glm.null.model = glm(ANGLE.CLOSURE ~ 1, data = trainsamp, family = "binomial",   
 control = glm.control(epsilon=epsmax))  
 glm.model = step(glm.null.model, scope=list(lower=glm.null.model, upper=glm.full.model),   
 direction="forward", trace=FALSE)  
   
 glm.pred = predict(glm.model, testsamp)  
   
 all.pred = rbind(all.pred, data.frame(testsamp$ANGLE.CLOSURE, svm.pred, nnet.pred, randomForest.pred, boosting.pred, glm.pred))  
  
}

## Processing... i: 1   
## SVM   
## NN   
## RF   
## Adaboost   
## Logit   
## Processing... i: 2   
## SVM   
## NN   
## RF   
## Adaboost   
## Logit   
## Processing... i: 3   
## SVM   
## NN   
## RF   
## Adaboost   
## Logit   
## Processing... i: 4   
## SVM   
## NN   
## RF   
## Adaboost   
## Logit   
## Processing... i: 5   
## SVM   
## NN   
## RF   
## Adaboost   
## Logit   
## Processing... i: 6   
## SVM   
## NN   
## RF   
## Adaboost   
## Logit   
## Processing... i: 7   
## SVM   
## NN   
## RF   
## Adaboost   
## Logit   
## Processing... i: 8   
## SVM   
## NN   
## RF   
## Adaboost   
## Logit   
## Processing... i: 9   
## SVM   
## NN   
## RF   
## Adaboost   
## Logit   
## Processing... i: 10   
## SVM   
## NN   
## RF   
## Adaboost   
## Logit

best.model = lm(testsamp.ANGLE.CLOSURE ~., data = all.pred)  
  
summary(best.model)$r.squared

## [1] 0.6363503

coefficients(best.model)

**## (Intercept) svm.pred nnet.pred randomForest.pred   
## -0.283283655 0.006222856 0.689895196 0.536199918   
## boosting.pred glm.pred   
## 0.386347449 0.001956960**

1. Constrained model (quadprog)

library(quadprog)  
  
Amat = cbind(rep(1,5),diag(5))  
  
bvec = c(1,rep(0,5))  
  
Dmat = t(all.pred[,-1]) %\*% as.matrix(all.pred[,-1])  
  
dvec = t(all.pred[,1]) %\*% as.matrix(all.pred[,-1])  
  
wopt = solve.QP(Dmat,dvec,Amat,bvec, meq=1)  
  
wopt$unconstrained.solution

## [1] -4.274292e-03 7.325020e-01 3.239533e-01 -2.992156e-05 8.105332e-04

wopt$solution

**## [1] -3.067766e-19 6.898969e-01 3.091514e-01 0.000000e+00 9.517123e-04**

1. Validation

Loading original case and control data-

# Read in angle closures data  
oDataCase = read.csv("AngleClosure\_ValidationCases.csv",header=TRUE)  
oDataCon = read.csv("AngleClosure\_ValidationControls.csv",header=TRUE)

Data cleansing to remove unwanted columns -

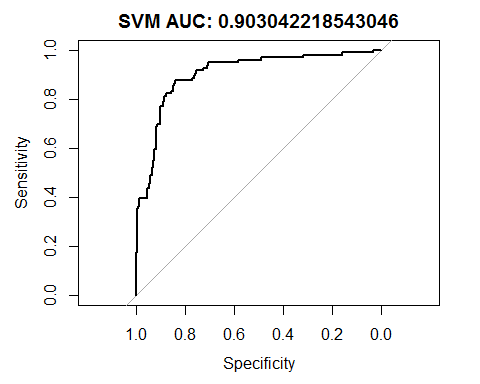
# Cleaning Case data  
  
mnames = names(myData)  
  
extraCols = NULL  
onames = names(oDataCase)  
  
sub\_rl = function(cname, mnames){  
 if(substr(cname, 1, 1) == 'r' || substr(cname, 1, 1) == 'l'){  
 return (substr(cname, 2, nchar(cname)) %in% mnames)  
 }  
 else {  
 return (cname %in% mnames)  
 }  
}  
  
for (cname in names(oDataCase)){  
 if (!(cname %in% mnames) && !sub\_rl(cname, mnames)){  
 extraCols = c(extraCols, cname)  
 }  
}  
  
  
# From extracols we notice that ACWmm is present in the original dataset as ACW\_mm. So we remove that from extraCols  
  
extraCols = extraCols[extraCols != "ACWmm"]  
colnames(oDataCase)[colnames(oDataCase) == 'ACWmm'] = 'ACW\_mm'  
remCols = colnames(oDataCase) %in% extraCols  
  
oDataCase = oDataCase[!remCols]  
  
  
# Cleaning Control data  
  
extraCols = NULL  
onames = colnames(oDataCon)  
  
  
for (cname in colnames(oDataCon)){  
 if (!(cname %in% mnames) && !sub\_rl(cname, mnames)){  
 extraCols = c(extraCols, cname)  
 }  
}  
  
  
# From extracols we notice that ACW.mm. is present in the original dataset as ACW\_mm. So we remove that from extraCols and rename the column  
  
extraCols = extraCols[extraCols != "ACW.mm."]  
colnames(oDataCon)[colnames(oDataCon) == 'ACW.mm.'] = 'ACW\_mm'  
remCols = colnames(oDataCon) %in% extraCols  
  
oDataCon = oDataCon[!remCols]

Further data cleansing to get aligned columns with original data

# Case data  
oDataValCase = matrix(NA,nrow(oDataCase),dim(myData)[2])  
colnames(oDataValCase) = colnames(myData)  
oDataValCase[,12] = 1  
  
for (i in seq(dim(myData)[2])){  
 for (j in seq(dim(oDataCase)[2])){  
 #colnames with no r or l  
 if(colnames(oDataCase)[j] == colnames(myData)[i]){  
 oDataValCase[,i] = oDataCase[,j] #rajdeep  
 }  
   
 #colnames with r  
 if (substr(colnames(oDataCase)[j],1,1) == 'r' &&   
 substr(colnames(oDataCase)[j],2,nchar(oDataCase)[j]) == colnames(myData)[i]){  
 oDataValCase[,i] = oDataCase[,j] #rajdeep  
 }  
   
 #colnames with l  
 if (substr(colnames(oDataCase)[j],1,1) == 'l' &&   
 substr(colnames(oDataCase)[j],2,nchar(oDataCase)[j]) == colnames(myData)[i]){  
 if(colnames(myData)[i] %in% colnames(oDataValCase)){  
 oDataValCase[which(is.na(oDataValCase[,i])),i] =  
 oDataCase[which(is.na(oDataValCase[,i])),j]  
 }  
 else{  
 oDataValCase[,i] = oDataCase[,j]  
 }  
 }  
 }  
}  
  
  
  
  
# Control data  
oDataValCon = matrix(NA,nrow(oDataCon),dim(myData)[2])  
colnames(oDataValCon) = colnames(myData)  
oDataValCon[,12] = 0  
  
for (i in seq(dim(myData)[2])){  
 for (j in seq(dim(oDataCon)[2])){  
 #colnames with no r or l  
 if(colnames(oDataCon)[j] == colnames(myData)[i]){  
 oDataValCon[,i] = oDataCon[,j] #rajdeep  
 }  
   
 #colnames with r  
 if (substr(colnames(oDataCon)[j],1,1) == 'r' &&   
 substr(colnames(oDataCon)[j],2,nchar(oDataCon)[j]) == colnames(myData)[i]){  
 oDataValCon[,i] = oDataCon[,j] #rajdeep  
 }  
   
 #colnames with l  
 if (substr(colnames(oDataCon)[j],1,1) == 'l' &&   
 substr(colnames(oDataCon)[j],2,nchar(oDataCon)[j]) == colnames(myData)[i]){  
 if(colnames(myData)[i] %in% colnames(oDataValCon)){  
 oDataValCon[which(is.na(oDataValCon[,i])),i] =  
 oDataCon[which(is.na(oDataValCon[,i])),j]  
 }  
 else{  
 oDataValCon[,i] = oDataCon[,j]  
 }  
 }  
 }  
}  
  
oDataValCase = oDataValCase[complete.cases(oDataValCase),]  
oDataValCon = oDataValCon[complete.cases(oDataValCon),]  
  
odataValAll = data.frame(rbind(oDataValCase, oDataValCon))

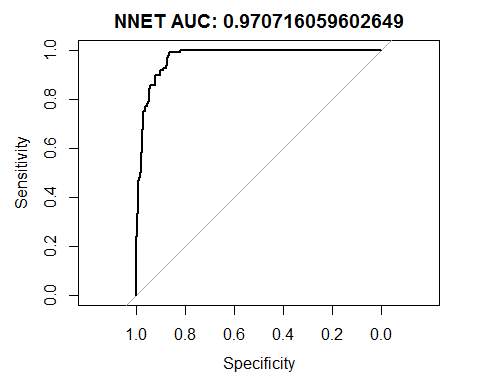
Now we will train the different models with optimum tuning parameter on original data and test it on validation data

#SVM  
  
svm.o.model = svm(ANGLE.CLOSURE ~ ., data = myData, gamma = gammamax, cost = costmax)  
svm.val.pred = predict(svm.o.model, odataValAll, type="response")  
svm.val.auc = auc(roc(odataValAll$ANGLE.CLOSURE, svm.val.pred))  
plot(roc(odataValAll$ANGLE.CLOSURE, svm.val.pred), main = paste("SVM AUC:", svm.val.auc))



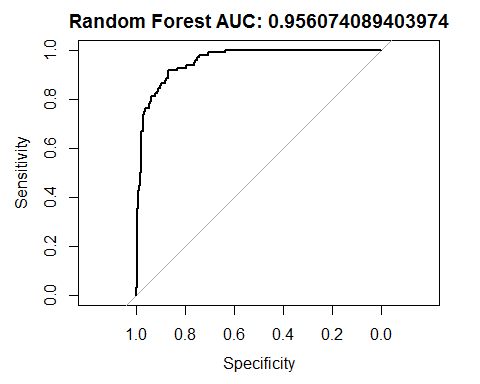
##   
## Call:  
## roc.default(response = odataValAll$ANGLE.CLOSURE, predictor = svm.val.pred)  
##   
## Data: svm.val.pred in 302 controls (odataValAll$ANGLE.CLOSURE 0) < 96 cases (odataValAll$ANGLE.CLOSURE 1).  
## Area under the curve: 0.903

#Neural Network  
  
nnet.o.model = nnet(ANGLE.CLOSURE ~ ., data = myData, size=10, decay=lambdamax, MaxNWts=250, trace = FALSE)  
nnet.val.pred = predict(nnet.o.model, odataValAll, type="raw")  
nnet.val.auc = auc(roc(odataValAll$ANGLE.CLOSURE, nnet.val.pred))  
plot(roc(odataValAll$ANGLE.CLOSURE, nnet.val.pred), main = paste("NNET AUC:", nnet.val.auc))



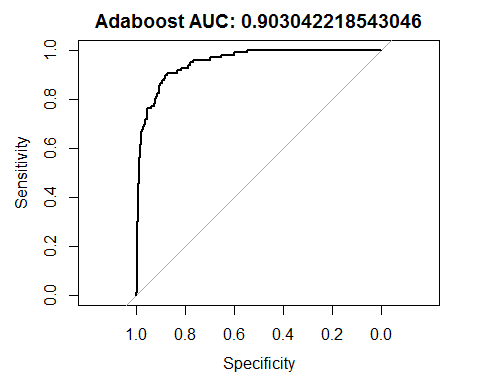
##   
## Call:  
## roc.default(response = odataValAll$ANGLE.CLOSURE, predictor = nnet.val.pred)  
##   
## Data: nnet.val.pred in 302 controls (odataValAll$ANGLE.CLOSURE 0) < 96 cases (odataValAll$ANGLE.CLOSURE 1).  
**## Of all models, Neural Network has the highest Area under the curve: 0.9707**

#Random Forest  
  
randomForest.o.model = randomForest(ANGLE.CLOSURE ~ ., data = myData, mtry = trymax)  
randomForest.val.pred = predict(randomForest.o.model, odataValAll, type = "response")  
randomForest.val.auc = auc(roc(odataValAll$ANGLE.CLOSURE, randomForest.val.pred))  
plot(roc(odataValAll$ANGLE.CLOSURE, randomForest.val.pred), main = paste("Random Forest AUC:", randomForest.val.auc))



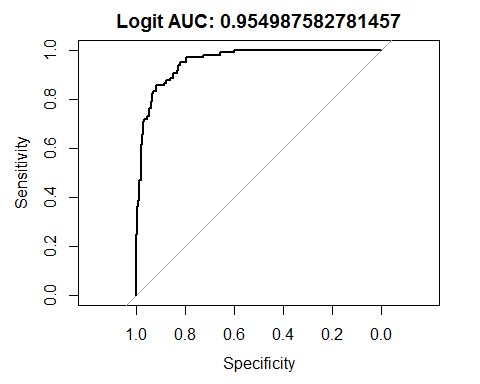
##   
## Call:  
## roc.default(response = odataValAll$ANGLE.CLOSURE, predictor = randomForest.val.pred)  
##   
## Data: randomForest.val.pred in 302 controls (odataValAll$ANGLE.CLOSURE 0) < 96 cases (odataValAll$ANGLE.CLOSURE 1).  
## Area under the curve: 0.9561

#Boosted model - Adaboost  
myData$ANGLE.CLOSURE = as.numeric(myData$ANGLE.CLOSURE)  
odataValAll[,"ANGLE.CLOSURE"] = as.numeric(odataValAll$ANGLE.CLOSURE)  
myData$ANGLE.CLOSURE = as.factor(myData$ANGLE.CLOSURE)  
odataValAll$ANGLE.CLOSURE = as.factor(odataValAll$ANGLE.CLOSURE)  
boosting.o.model = boosting(ANGLE.CLOSURE ~ ., data = myData, mfinal=mmax, coeflearn="Freund", control = rpart.control(maxdepth = 10))  
boosting.val.pred = predict(boosting.o.model, odataValAll)$prob[,2]  
boosting.val.auc = auc(roc(odataValAll$ANGLE.CLOSURE, boosting.val.pred))  
plot(roc(odataValAll$ANGLE.CLOSURE, boosting.val.pred), main = paste("Adaboost AUC:", svm.val.auc))



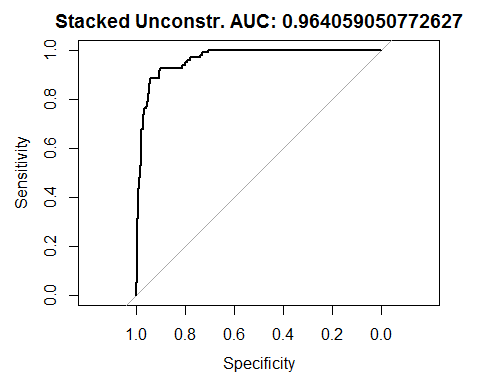
##   
## Call:  
## roc.default(response = odataValAll$ANGLE.CLOSURE, predictor = boosting.val.pred)  
##   
## Data: boosting.val.pred in 302 controls (odataValAll$ANGLE.CLOSURE 0) < 96 cases (odataValAll$ANGLE.CLOSURE 1).  
## Area under the curve: 0.9515

myData$ANGLE.CLOSURE = as.numeric(myData$ANGLE.CLOSURE) -1  
odataValAll$ANGLE.CLOSURE = as.numeric(odataValAll$ANGLE.CLOSURE)-1  
  
#Logistic regression  
  
glm.o.full.model = glm(ANGLE.CLOSURE ~ ., data = myData, family = "binomial", control = glm.control(epsilon=epsmax))  
glm.o.null.model = glm(ANGLE.CLOSURE ~ 1, data = myData, family = "binomial", control = glm.control(epsilon=epsmax))  
  
#Forward stepwise AIC  
glm.o.model = step(glm.o.null.model, scope=list(lower=glm.o.null.model, upper=glm.o.full.model), direction="forward", trace=FALSE)  
  
  
glm.val.pred = predict(glm.o.model, odataValAll, type="response")  
glm.val.auc = auc(roc(odataValAll$ANGLE.CLOSURE, glm.val.pred))  
plot(roc(odataValAll$ANGLE.CLOSURE, glm.val.pred), main = paste("Logit AUC:", glm.val.auc))



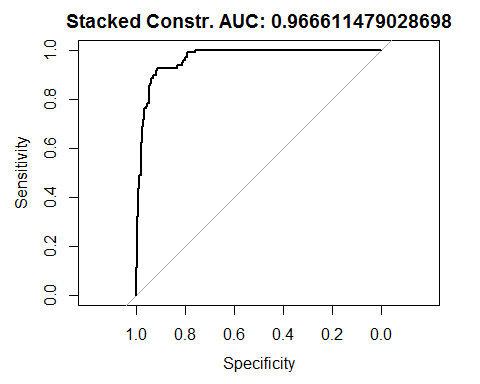
##   
## Call:  
## roc.default(response = odataValAll$ANGLE.CLOSURE, predictor = glm.val.pred)  
##   
## Data: glm.val.pred in 302 controls (odataValAll$ANGLE.CLOSURE 0) < 96 cases (odataValAll$ANGLE.CLOSURE 1).  
## Area under the curve: 0.955

#Create predicted data set for fitting with stacked model  
#Original data set  
#all.pred = rbind(all.pred, data.frame(testsamp$ANGLE.CLOSURE, svm.pred, nnet.pred, randomForest.pred, boosting.pred, glm.pred))  
  
all.val.pred = cbind(svm.val.pred, nnet.val.pred, randomForest.val.pred, boosting.val.pred, glm.val.pred)  
  
#Stacked model (unconstrained)  
stck\_uconstr.val.pred = coefficients(best.model)[2:6] %\*% t(all.val.pred)  
stck\_uconstr.val.auc = auc(roc(odataValAll$ANGLE.CLOSURE, stck\_uconstr.val.pred))  
plot(roc(odataValAll$ANGLE.CLOSURE, stck\_uconstr.val.pred), main = paste("Stacked Unconstr. AUC:", stck\_uconstr.val.auc))



##   
## Call:  
## roc.default(response = odataValAll$ANGLE.CLOSURE, predictor = stck\_uconstr.val.pred)  
##   
## Data: stck\_uconstr.val.pred in 302 controls (odataValAll$ANGLE.CLOSURE 0) < 96 cases (odataValAll$ANGLE.CLOSURE 1).  
## Area under the curve: 0.9641

#Stacked model (constrained)  
stck\_constr.val.pred = wopt$solution %\*% t(all.val.pred)  
stck\_constr.val.auc = auc(roc(odataValAll$ANGLE.CLOSURE, stck\_constr.val.pred))  
plot(roc(odataValAll$ANGLE.CLOSURE, stck\_constr.val.pred), main = paste("Stacked Constr. AUC:", stck\_constr.val.auc))



##   
## Call:  
## roc.default(response = odataValAll$ANGLE.CLOSURE, predictor = stck\_constr.val.pred)  
##   
## Data: stck\_constr.val.pred in 302 controls (odataValAll$ANGLE.CLOSURE 0) < 96 cases (odataValAll$ANGLE.CLOSURE 1).  
## Area under the curve: 0.9666