Prediction Model for Angle Closure Glaucoma

Xiaochen Zhang Georgia Tech

Angle closure glaucoma is a major cause of blindness worldwide. In this project, we built a prediction model for angle closure prediction. Our predictors including various eye measurements, gender, age, ethnic and so forth. As a binary predictive model, various models are used and compared including:

- Support vector machine
- Neural network
- Random forest
- Ada boost
- Logistic regression

1. Data Manipulation

To begin with, the raw data are processed by either omitting some poorly recorded variables and performing data imputation until the dataset is valid for further investigation. Table I shows how we perform the initial data manipulation.

TABLE 1 Data Manipulation

Actions	Observations	Predictors
Read in raw data	1468	24
Omit certain predictors	1468	11
Delete rows with missing data	1468	11

Please check Appendix I for r code.

2. Develop Prediction Models

Five prediction models are chosen, and they are

- Support vector machine,
- Neural network,
- Random forest,
- Boosted model, and
- Logistic regression.

All these prediction models have been realized by existing packages in R. Please check Appendix II for r code.

3. Model Parameter Selection

Most models are tuned with 2 parameters except random forest, where only number of trees is tuned. The detailed behaviors of the tuning parameters can be found in the visualization section.

From Table II, we can see that with proper parameters, all five models works very well with AUC around 0.95. *We cannot say one model is better than the other because we use 10 fold cross validation with a*

small iteration number = 10. When we increase the iteration number, we might be able to stabilize the performance of the five algorithms.

However, when it comes to efficiency, **boosted model is very slow** compared with other algorithms.

TABLE II Parameter Selection

Models	Parameter 1	Value	Parameter 2	Value	AUC
Support vector machine	Penalty cost C	0.1	Gamma	0.01	0.9527155
Neural network	Network size	2	Decay	0.0005	0.9557656
Random forest	Num. of trees M	100			0.9548375
Boosted model	Complexity M	100	Max depth	2	0.9325812
Logistic regression	Penalty factor K	K=5	Step direction	"both"	0.9507953

Please check Appendix III for r code.

4. Stacking

The stacking models with and without constraints are shown in Table III. Again, we cannot guarantee the weights we have here is the "true" weight. This is because 10 fold cross validation is used, and the iteration number is set as 10, which makes *the results strongly depend on the data which are sampled*.

TABLE III Stacking Model Weights

Models	Weight (with Constraints)	Weight (without Constraints)		
Support vector machine	0	-0.03079		
Neural network	0	-0.21070		
Random forest	0.11378	0.14418		
Boosted model	0.00252	-0.02147		
Logistic regression	0.88369	1.07496		

Please check Appendix IV for r code.

5. Validation

We read in the validation data, and try to test the performance of the trained 7 models. Please note that *the training data and the validation data have very different prevalence*, which will make the validation data look worse.

TABLE IV Validation Results

Models	Parameter 1	Value	Parameter 2	Value	AUC
Support vector machine	Penalty cost C	0.1	Gamma	0.01	0.9391
Neural network	Network size	2	Decay	0.0005	0.9506
Random forest	Num. of trees M	100			0.9548
Boosted model	Complexity M	100	Max depth	2	0.9643
Logistic regression	Penalty factor K	K=5	Step direction	"both"	0.9531
Stacking constrained	Weight see	Table III			0.9557
Stacking Unconstrained	Weight see	Table III			0.9553

Please check Appendix V for r code.

6. Visualization

Part I: AUC vs. tuning parameters

Figure 1-5 shows the plots of AUC vs. tuning parameters for the adopted 5 algorithm discussed above. Please note that the AUC value is visualized using ellipse map, where *the area of the ellipse is inversely proportional to the AUC value*.

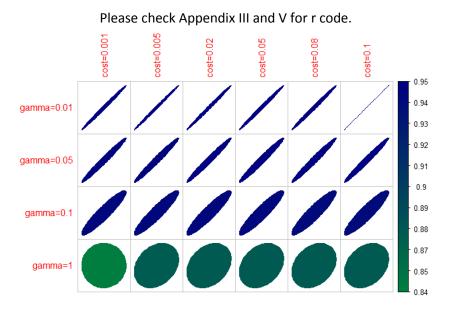


Figure 1 Support Vector Machine: Gamma = 0.01 & Cost = 0.1

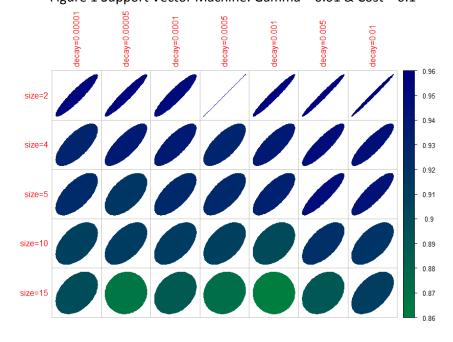


Figure 2 Neural Network: size = 2 & decay = 0.0005

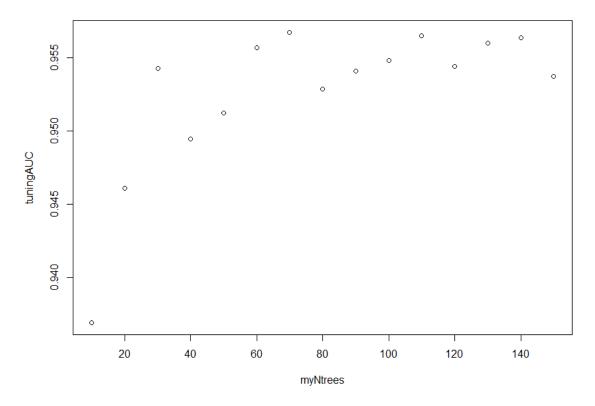


Figure 3 Random Forest: number of trees = 100

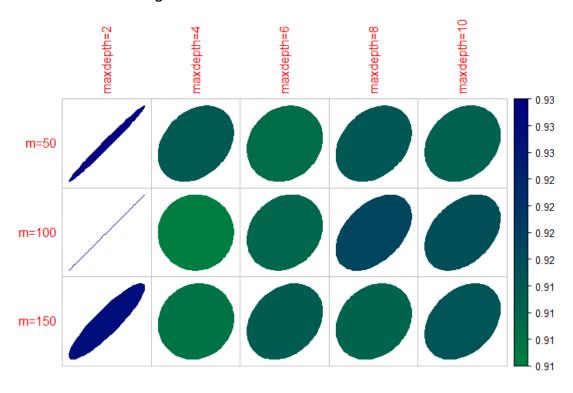


Figure 4 Boosted Model: m = 100 & maxdepth = 2

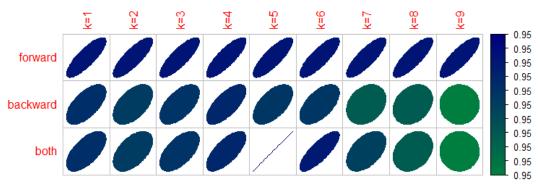


Figure 5 Logistic Regression: direction = "both" & K = 5

Part II: ROC Curves and AUCs of 7 models

From Figure 6, we can see that the boosted model works slightly better than the rest 6 models, and support vector machine has the lowest AUC value.

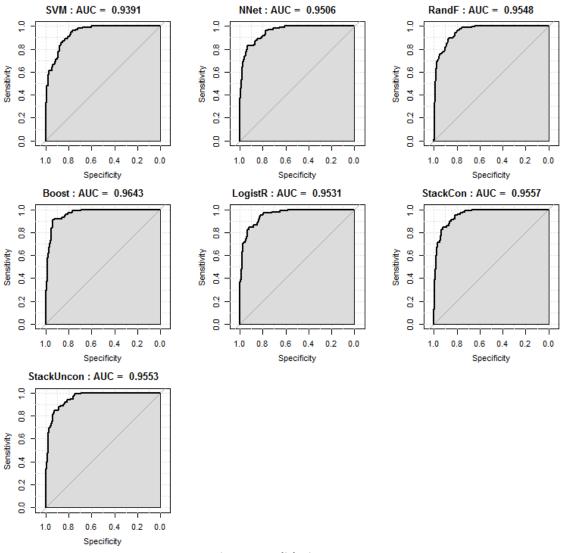


Figure 6 Validation AUCs

Appendix I Data Manipulation

```
# Read in angle closure data
myData=read.csv("AngleClosure.csv",header=TRUE,na.strings=c("NA","."))
# Set-up response and predictor variables
myResponse=as.numeric(myData$ANGLE.CLOSURE=="YES")
# omit the variables HGT, WT, ASPH, ACYL, SE, AXL, CACD, AGE, CCT.OD, and PCCURV_mm
myPredictors=data.matrix(myData[,!(attributes(myData)$names %in%
                   c("EYE","GENDER","ETHNIC","ANGLE.CLOSURE","HGT","WT","ASPH", "ACYL", "SE", "AXL", "CACD", "AGE",
"CCT.OD", "PCCURV_mm"))])
# Remove rows with any missingness
myLogical=apply(cbind(myResponse,myPredictors),1,function(xx){
 return(!any(is.na(xx)))
})
myResponse=myResponse[myLogical]
myPredictors=myPredictors[myLogical,]
# make all data mean zero and variance one
myPredictors_mean = apply(myPredictors,2,function(xx){
 return(mean(xx,na.rm=TRUE))
})
myPredictors sd = apply(myPredictors,2,function(xx){
 return(sd(xx,na.rm=TRUE))
})
myX=apply(myPredictors,2,function(xx){
 return((xx-mean(xx,na.rm=TRUE))/sd(xx,na.rm=TRUE))
})
myY = myResponse
```

Appendix II Building 5 Models

```
# Read in angle closure data
                                                                  testYFactor = myYFactor[myIndices]
myData=read.csv("AngleClosure.csv",header=TRUE,na.stri
ngs=c("NA","."))
                                                                  # Support vector machine ~ cost
# Set-up response and predictor variables
                                                                  library(e1071)
myResponse=as.numeric(myData$ANGLE.CLOSURE=="YES
                                                                  svm.model = svm(trainingX, trainingY, type = "C", cost =
                                                                  10, probability = TRUE)
# omit the variables HGT, WT, ASPH, ACYL, SE, AXL, CACD,
                                                                  svm.pred <- predict(svm.model, testX)</pre>
AGE, CCT.OD, and PCCURV_mm
                                                                 svm.prob = predict(svm.model, testX, probability = TRUE)
myPredictors=data.matrix(myData[,!(attributes(myData)$
                                                                 svmt=table(pred = svm.pred, true = testY)
names %in%
                                                                 svmt
c("EYE", "GENDER", "ETHNIC", "ANGLE.CLOSURE", "HGT", "W
                                                                  # neural network ~ size ~ decay
T","ASPH", "ACYL", "SE", "AXL", "CACD", "AGE", "CCT.OD",
                                                                 library(nnet)
"PCCURV_mm"))])
                                                                  lambda = 0.0001
# Remove rows with any missingness
                                                                  fit=nnet(trainingYFactor~.,data=trainingX,weights=rep(1,le
myLogical=apply(cbind(myResponse,myPredictors),1,funct
                                                                  ngth(trainingYFactor)),size=10,
                                                                      decay=lambda,MaxNWts=10000,maxit=250)
ion(xx){
return(!any(is.na(xx)))
                                                                  NNPred=predict(fit,newdata=testX, type = "class")
})
                                                                  NNProb=predict(fit,newdata=testX, type = "raw")
myResponse=myResponse[myLogical]
                                                                  NNt=table(pred = NNPred, true = testY)
myPredictors=myPredictors[myLogical,]
                                                                 NNt
                                                                  # random forest ~ ntree
# make all data mean zero and variance one
                                                                  library(randomForest)
myPredictors mean = apply(myPredictors,2,function(xx){
return(mean(xx,na.rm=TRUE))
                                                                  RFmodel = randomForest(x = trainingX, y = trainingYFactor,
                                                                  ntree = 10
myPredictors_sd = apply(myPredictors,2,function(xx){
                                                                  RFPred = predict(RFmodel, testX, type = "class")
return(sd(xx,na.rm=TRUE))
                                                                  RFProb = predict(RFmodel, testX, type = "prob")[,2]
})
                                                                  RFt=table(pred = RFPred, true = testY)
                                                                 RFt
myX=apply(myPredictors,2,function(xx){
  return((xx-mean(xx,na.rm=TRUE))/sd(xx,na.rm=TRUE))
                                                                  # boosted model ~ mfinal ~ maxdepth
                                                                  library(adabag)
})
myY = myResponse
                                                                  myYfactor = as.factor(myY)
                                                                  adadata = data.frame(myYfactor,myX)
######### Develop Prediction Models
                                                                  adaTrain = adadata[-myIndices,]
adaTest = adadata[myIndices,]
# creat a taining set and varification set
                                                                  myM = 100
set.seed(30126)
                                                                  adamodel =
nFolds=10
                                                                  boosting(myYfactor~.,data=adaTrain,mfinal=myM,coeflear
                                                                  n="Freund",
myIndices=sample(length(myResponse),ceiling(length(myR
esponse)/nFolds))
                                                                            control=rpart.control(maxdepth=10))
                                                                  adaPred = predict(adamodel, newdata=adaTest)
trainingX = myX[-myIndices,]
trainingY = myY[-myIndices]
                                                                 adaProb = adaPred$prob[,2]
testX = myX[myIndices,]
                                                                 adat=table(pred = adaPred$class, true = testY)
                                                                  adat
testY = myY[myIndices]
myYFactor = factor(myY)
                                                                  # logistic regression ~ k ~ direction
                                                                  LRdata = data.frame(trainingX, trainingYFactor)
trainingYFactor = myYFactor[-myIndices]
```

```
LRtestdata = data.frame(testX, testYFactor)
                                                                   myRoc <- roc(response = myYtest, predictor =
                                                                  attr(svm.prob,"probabilities")[,colnames(attr(svm.prob,"pr
LRmodel = glm( trainingYFactor~ AOD750 + TISA750 +
                                                                  obabilities"))==1], auc.polygon=TRUE, grid=TRUE,
IT750 +IT2000 + ITCM + IAREA + ICURV + ACW mm + ACA
+ ACV + LENSVAULT, family = binomial(logit),data = LRdata)
                                                                  plot=FALSE)
LRmodelstep = step(LRmodel,k=2,direction = "both")
                                                                   plot(myRoc)
LRProb = predict(LRmodelstep, newdata =
                                                                  }
data.frame(testX), type = "response")
# LRPred = ifelse(predict(LRmodel, newdata =
                                                                  ######################### tuning Parameter
                                                                  data.frame(testX), type = "response")>.5,1,0)
# LRProb = predict(LRmodel, newdata = data.frame(testX),
type = "response")
                                                                  ## SVM ~ cost
LRt=table(pred = LRPred, true = testY)
LRt
                                                                  library(e1071)
                                                                  set.seed(123456789)
## 10 folds validation
                                                                  nFolds = 10
library(pROC)
                                                                  table = matrix(0,2,2)
# Support vector machine ~ cost
                                                                  for (jj in 1:nFolds){
library(e1071)
                                                                   # Generate training and testing responses and predictors
                                                                  for each fold
set.seed(123456789)
nFolds = 10
                                                                  myIndices=sample(length(myY))[1:round(length(myY)/nFol
table = matrix(0,2,2)
for (jj in 1:nFolds){
                                                                  ds)]
# Generate training and testing responses and predictors
                                                                   myXtrain = myX[-myIndices,]
for each fold
                                                                   myYtrain = myY[-myIndices]
                                                                   myXtest = myX[myIndices,]
myIndices=sample(length(myY))[1:round(length(myY)/nFol
                                                                   myYtest = myY[myIndices]
ds)]
myXtrain = myX[-myIndices,]
                                                                   # Set tuning parameters
myYtrain = myY[-myIndices]
                                                                   myC = 10
myXtest = myX[myIndices,]
myYtest = myY[myIndices]
                                                                   # Calculate the SVM model
                                                                   svm.model = svm(myXtrain, myYtrain, type = "C", cost =
# Set tuning parameters
                                                                  myC,probability = TRUE)
myC = 10
                                                                   # Set threshold
# Calculate the SVM model
                                                                   threshold = 0.5
svm.model = svm(myXtrain, myYtrain, type = "C", cost =
                                                                   svm.prob = predict(svm.model, myXtest, probability =
myC,probability = TRUE)
                                                                  TRUE)
                                                                   svm.pred =
# Set threshold
                                                                  ifelse(attr(svm.prob,"probabilities")[,colnames(attr(svm.pr
threshold = 0.5
                                                                  ob,"probabilities"))==1]>threshold,1,0)
svm.prob = predict(svm.model, myXtest, probability =
                                                                   svmt = table(pred = svm.pred, true = myYtest)
TRUE)
                                                                   print(svmt)
svm.pred =
                                                                   table = table + svmt
ifelse(attr(svm.prob,"probabilities")[,colnames(attr(svm.pr
                                                                   myRoc <- roc(response = myYtest, predictor =
                                                                  attr(svm.prob,"probabilities")[,colnames(attr(svm.prob,"pr
ob,"probabilities"))==1]>threshold,1,0)
svmt = table(pred = svm.pred, true = myYtest)
                                                                  obabilities"))==1], auc.polygon=TRUE, grid=TRUE,
print(svmt)
                                                                  plot=FALSE)
table = table + symt
                                                                   plot(myRoc)
```

Appendix III Tuning Parameters

```
tuningProb = array(NA,dim =
# Read in angle closure data
myData=read.csv("AngleClosure.csv",header=TRUE,na.stri
                                                               c(iter,length(myCs),length(myGammas),round(length(myY)
ngs=c("NA","."))
                                                               /nFolds)))
# Set-up response and predictor variables
                                                               tuningRes = array(NA,dim =
myResponse=as.numeric(myData$ANGLE.CLOSURE=="YES
                                                               c(iter,round(length(myY)/nFolds)))
                                                               for (jj in 1:iter){
# omit the variables HGT, WT, ASPH, ACYL, SE, AXL, CACD,
AGE, CCT.OD, and PCCURV_mm
                                                                # Generate training and testing responses and predictors
myPredictors=data.matrix(myData[,!(attributes(myData)$
                                                               for each fold
names %in%
                                                               myIndices=sample(length(myY))[1:round(length(myY)/nFol
c("EYE", "GENDER", "ETHNIC", "ANGLE.CLOSURE", "HGT", "W
T","ASPH", "ACYL", "SE", "AXL", "CACD", "AGE", "CCT.OD",
                                                                myXtrain = myX[-myIndices,]
"PCCURV mm"))])
                                                                myYtrain = myY[-myIndices]
# Remove rows with any missingness
                                                                myXtest = myX[myIndices,]
myLogical=apply(cbind(myResponse,myPredictors),1,funct
                                                                myYtest = myY[myIndices]
                                                                for (ii in 1:length(myCs)){
ion(xx){
return(!any(is.na(xx)))
                                                                 # Set tuning parameters
})
                                                                 myC = myCs[ii]
myResponse=myResponse[myLogical]
                                                                 for (kk in 1:length(myGammas)){
myPredictors=myPredictors[myLogical,]
                                                                  myGamma = myGammas[kk]
                                                                  # Calculate the SVM model
                                                                  svm.model = svm(myXtrain, myYtrain, type = "C",
# make all data mean zero and variance one
                                                               gamma = myGamma, cost = myC,probability = TRUE)
myPredictors mean = apply(myPredictors,2,function(xx){
return(mean(xx,na.rm=TRUE))
                                                                  # Predict probability
                                                                  svm.prob = predict(svm.model, myXtest, probability =
myPredictors_sd = apply(myPredictors,2,function(xx){
                                                               TRUE)
return(sd(xx,na.rm=TRUE))
                                                                  tuningProb[jj,ii,kk,] =
})
                                                               attr(svm.prob,"probabilities")[,colnames(attr(svm.prob,"pr
                                                               obabilities"))==1]
myX=apply(myPredictors,2,function(xx){
                                                                }
return((xx-mean(xx,na.rm=TRUE))/sd(xx,na.rm=TRUE))
                                                               }
                                                                tuningRes[jj,] = myYtest
})
myY = myResponse
                                                               # compute ROC
roc res = array(NA,dim =
c(iter*round(length(myY)/nFolds),1))
                                                               roc prob = array(NA,dim =
## SVM ~ cost ~ gamma
                                                               c(iter*round(length(myY)/nFolds),length(myCs),length(my
                                                               Gammas)))
                                                               for (jj in 1:iter){
library(e1071)
                                                                roc res[((jj-
set.seed(123456789)
                                                               1)*round(length(myY)/nFolds)+1):(jj*round(length(myY)/n
nFolds = 10
                                                               Folds))] = tuningRes[jj,]
iter = 10
                                                                for (ii in 1:length(myCs)){
myCs = c(0.001, 0.005, 0.02, 0.05, 0.08, 0.1)
                                                                 for (kk in 1:length(myGammas)){
myGammas = c(0.01,0.05,0.1,1)
```

```
roc_prob[((jj-
1)*round(length(myY)/nFolds)+1):(jj*round(length(myY)/n
                                                                 # Generate training and testing responses and predictors
Folds)),ii,kk] = tuningProb[jj,ii,kk,]
                                                                for each fold
                                                                myIndices=sample(length(myY))[1:round(length(myY)/nFol
}
}
                                                                 myXtrain = myX[-myIndices,]
                                                                 myYtrain = myY[-myIndices]
# compute ROC and plot ROC vs. tuning parameters
                                                                 myYtrainFactor = as.factor(myYtrain)
myRocList = list()
                                                                 myXtest = myX[myIndices,]
tuningAUC = matrix(NA,length(myCs),length(myGammas))
                                                                 myYtest = myY[myIndices]
for (ii in 1:length(myCs)){
                                                                 for (ii in 1:length(mySizes)){
for (kk in 1:length(myGammas)){
                                                                  # Set tuning parameters
  myRocList[[(ii-1)*length(myGammas)+kk]] =
                                                                  mySize = mySizes[ii]
roc(response = roc_res, predictor = roc_prob[,ii,kk],
                                                                  for (kk in 1:length(myDecays)){
auc.polygon=TRUE, grid=TRUE, plot=TRUE)
                                                                   myDecay = myDecays[kk]
  tuningAUC[ii,kk] = myRocList[[(ii-
                                                                   # Calculate the NN model
1)*length(myGammas)+kk]]$auc
                                                                fit=nnet(myYtrainFactor~.,data=myXtrain,weights=rep(1,le
}
                                                                ngth(myYtrainFactor)),size=mySize,
library(corrplot)
rownames(tuningAUC) =
                                                                decay=myDecay,MaxNWts=10000,maxit=10000,trace=FAL
c("cost=0.001","cost=0.005","cost=0.02","cost=0.05","cost
                                                                SE)
=0.08","cost=0.1")
                                                                   # Predict probability
colnames(tuningAUC) =
                                                                   NNProb=predict(fit,newdata=myXtest, type = "raw")
c("gamma=0.01","gamma=0.05","gamma=0.1","gamma=1
                                                                   tuningProb[jj,ii,kk,] = NNProb
                                                                  }
corrplot(t(tuningAUC), method = "ellipse", order =
                                                                 }
"original", is.corr = FALSE,col =
                                                                 tuningRes[jj,] = myYtest
colorRampPalette(c("green","navyblue"))(100))
                                                                # compute ROC
                                                                roc res = array(NA,dim =
                                                                c(iter*round(length(myY)/nFolds),1))
roc prob = array(NA,dim =
c(iter*round(length(myY)/nFolds),length(mySizes),length(
                                                                myDecays)))
## Neural Network ~ size ~ decay
                                                                for (jj in 1:iter){
library(nnet)
                                                                 roc res[((jj-
set.seed(123456789)
                                                                1)*round(length(myY)/nFolds)+1):(jj*round(length(myY)/n
nFolds = 10
                                                                Folds))] = tuningRes[jj,]
iter = 10
                                                                 for (ii in 1:length(mySizes)){
mySizes = c(2,4,5,10,15)
                                                                  for (kk in 1:length(myDecays)){
myDecays =
                                                                   roc prob[((jj-
c(0.00001,0.00005,0.0001,0.0005,0.001,0.005,0.01)
                                                                1)*round(length(myY)/nFolds)+1):(jj*round(length(myY)/n
tuningProb = array(NA,dim =
                                                                Folds)),ii,kk] = tuningProb[jj,ii,kk,]
c(iter,length(mySizes),length(myDecays),round(length(myY
                                                                  }
)/nFolds)))
tuningRes = array(NA,dim =
                                                                 }
c(iter,round(length(myY)/nFolds)))
                                                                }
for (jj in 1:iter){
```

```
# compute ROC and plot ROC vs. tuning parameters
                                                               for (ii in 1:length(myNtrees)){
myRocList = list()
                                                                # Set tuning parameters
tuningAUC = matrix(NA,length(mySizes),length(myDecays))
                                                                myNtree = myNtrees[ii]
for (ii in 1:length(mySizes)){
                                                                # Calculate the RF model
for (kk in 1:length(myDecays)){
                                                                RFmodel = randomForest(x = myXtrain, y =
  myRocList[[(ii-1)*length(myDecays)+kk]] = roc(response
                                                              myYtrainFactor, ntree = myNtree)
= roc_res, predictor = roc_prob[,ii,kk], auc.polygon=TRUE,
                                                                # Predict probability
grid=TRUE, plot=TRUE)
                                                                RFProb = predict(RFmodel, myXtest, type = "prob")[,2]
  tuningAUC[ii,kk] = myRocList[[(ii-
                                                                tuningProb[jj,ii,] = RFProb
1)*length(myDecays)+kk]]$auc
}
                                                               tuningRes[jj,] = myYtest
}
plot(tuningAUC)
image(t(tuningAUC[nrow(tuningAUC):1,]), axes=FALSE,
                                                              # compute ROC
zlim=c(-4,4), col=rainbow(21))
                                                              roc_res = array(NA,dim =
library(corrplot)
                                                              c(iter*round(length(myY)/nFolds),1))
rownames(tuningAUC) =
                                                              roc_prob = array(NA,dim =
c("size=2","size=4","size=5","size=10","size=15")
                                                              c(iter*round(length(myY)/nFolds),length(myNtrees)))
colnames(tuningAUC) =
                                                              for (jj in 1:iter){
c("decay=0.00001","decay=0.00005","decay=0.0001","dec
                                                               roc_res[((jj-
ay=0.0005","decay=0.001","decay=0.005","decay=0.01")
                                                              1)*round(length(myY)/nFolds)+1):(jj*round(length(myY)/n
corrplot(tuningAUC, method = "ellipse", order = "original",
                                                              Folds))] = tuningRes[jj,]
is.corr = FALSE,col =
                                                               for (ii in 1:length(myNtrees)){
colorRampPalette(c("green", "navyblue"))(100))
                                                                 roc_prob[((jj-
                                                              1)*round(length(myY)/nFolds)+1):(jj*round(length(myY)/n
                                                              Folds)),ii] = tuningProb[jj,ii,]
}
}
## random forest ~ ntree
                                                              # compute ROC and plot ROC vs. tuning parameters
library(randomForest)
                                                              myRocList = list()
set.seed(123456789)
                                                              tuningAUC = matrix(NA,length(myNtrees),1)
nFolds = 10
                                                              for (ii in 1:length(myNtrees)){
iter = 10
                                                                myRocList[[ii]] = roc(response = roc_res, predictor =
myNtrees = seq(10,150,10)
                                                              roc_prob[,ii], auc.polygon=TRUE, grid=TRUE, plot=TRUE)
tuningProb = array(NA,dim =
                                                                tuningAUC[ii,1] = myRocList[[ii]]$auc
c(iter,length(myNtrees),round(length(myY)/nFolds)))
tuningRes = array(NA,dim =
c(iter,round(length(myY)/nFolds)))
                                                              plot(myNtrees,tuningAUC)
for (jj in 1:iter){
# Generate training and testing responses and predictors
                                                              for each fold
myIndices=sample(length(myY))[1:round(length(myY)/nFol
                                                              # boosted model ~ mfinal ~ maxdepth
                                                              library(adabag)
myXtrain = myX[-myIndices,]
                                                              set.seed(123456789)
                                                              nFolds = 10
myYtrain = myY[-myIndices]
myYtrainFactor = as.factor(myYtrain)
                                                              iter = 6
myXtest = myX[myIndices,]
                                                              myMs = c(50,100,150)
                                                              # myMs = c(50,100,150,200,250)
myYtest = as.factor(myY[myIndices])
```

```
# myMaxdepths = c(5,10,15,20,25)
                                                                roc_res[((jj-
                                                               1)*round(length(myY)/nFolds)+1):(jj*round(length(myY)/n
myMaxdepths = c(2,4,6,8,10)
tuningProb = array(NA,dim =
                                                               Folds))] = tuningRes[jj,]
c(iter,length(myMs),length(myMaxdepths),round(length(
                                                                for (ii in 1:length(myMs)){
                                                                 for (kk in 1:length(myMaxdepths)){
myY)/nFolds)))
tuningRes = array(NA,dim =
                                                                  roc prob[((jj-
c(iter,round(length(myY)/nFolds)))
                                                               1)*round(length(myY)/nFolds)+1):(jj*round(length(myY)/n
                                                               Folds)),ii,kk] = tuningProb[jj,ii,kk,]
for (jj in 1:iter){
# Generate training and testing responses and predictors
for each fold
                                                                }
                                                               }
myIndices=sample(length(myY))[1:round(length(myY)/nFol
                                                               # compute ROC and plot ROC vs. tuning parameters
myYfactor = as.factor(myY)
                                                               myRocList = list()
adadata = data.frame(myYfactor,myX)
                                                               tuningAUC =
adaTrain = adadata[-myIndices,]
                                                               matrix(NA,length(myMs),length(myMaxdepths))
adaTest = adadata[myIndices,]
                                                               for (ii in 1:length(myMs)){
                                                                for (kk in 1:length(myMaxdepths)){
                                                                 myRocList[[(ii-1)*length(myMaxdepths)+kk]] =
for (ii in 1:length(myMs)){
                                                               roc(response = roc_res, predictor = roc_prob[,ii,kk],
  print("Fuck")
  # Set tuning parameters
                                                               auc.polygon=TRUE, grid=TRUE, plot=TRUE)
  myM = myMs[ii]
                                                                 tuningAUC[ii,kk] = myRocList[[(ii-
  for (kk in 1:length(myMaxdepths)){
                                                               1)*length(myMaxdepths)+kk]]$auc
                                                                }
   print(kk)
   myMaxdepth = myMaxdepths[kk]
                                                               }
   # Calculate the NN model
                                                               plot(tuningAUC)
   adamodel =
                                                               image(t(tuningAUC[nrow(tuningAUC):1,]), axes=FALSE,
boosting(myYfactor~.,data=adaTrain,mfinal=myM,coeflear
                                                               zlim=c(-4,4), col=rainbow(21))
n="Freund",
                                                               library(corrplot)
                                                               # rownames(tuningAUC) =
                                                               c("m=50","m=100","m=150","m=200","m=250")
control=rpart.control(maxdepth=myMaxdepth))
   # Predict probability
                                                               # colnames(tuningAUC) =
   adaPred = predict(adamodel, newdata=adaTest)
                                                               c("maxdepth=5","maxdepth=10","maxdepth=15","maxdep
   adaProb = adaPred$prob[,2]
                                                               th=20","maxdepth=25")
   tuningProb[jj,ii,kk,] = adaProb
                                                               rownames(tuningAUC) = c("m=50", "m=100", "m=150")
 }
                                                               colnames(tuningAUC) =
                                                               c("maxdepth=2","maxdepth=4","maxdepth=6","maxdepth
tuningRes[jj,] = adaTest[,1]
                                                               =8","maxdepth=10")
print("You!")
                                                               corrplot(tuningAUC, method = "ellipse", order = "original",
                                                               is.corr = FALSE,col =
                                                               colorRampPalette(c("green", "navyblue"))(100))
# compute ROC
roc res = array(NA,dim =
c(iter*round(length(myY)/nFolds),1))
roc prob = array(NA,dim =
                                                               c(iter*round(length(myY)/nFolds),length(myMs),length(my
                                                               Maxdepths)))
for (jj in 1:iter){
                                                               # logistic regression ~ k ~ direction
                                                               library(adabag)
                                                               set.seed(123456789)
```

```
nFolds = 10
                                                                                                                             # compute ROC
iter = 10
myDirections = c("forward", "backward", "both")
                                                                                                                             roc res = array(NA,dim =
myKs = c(1,2,3,4,5,6,7,8,9)
                                                                                                                             c(iter*round(length(myY)/nFolds),1))
tuningProb = array(NA,dim =
                                                                                                                             roc prob = array(NA,dim =
c(iter,length(myDirections),length(myKs),round(length(my
                                                                                                                             c(iter*round(length(myY)/nFolds), length(myDirections), length(m
Y)/nFolds)))
                                                                                                                             gth(myKs)))
tuningRes = array(NA,dim =
                                                                                                                             for (jj in 1:iter){
c(iter,round(length(myY)/nFolds)))
                                                                                                                               roc_res[((jj-
                                                                                                                             1)*round(length(myY)/nFolds)+1):(jj*round(length(myY)/n
                                                                                                                             Folds))] = tuningRes[jj,]
for (jj in 1:iter){
 # Generate training and testing responses and predictors
                                                                                                                               for (ii in 1:length(myDirections)){
for each fold
                                                                                                                                 for (kk in 1:length(myKs)){
                                                                                                                                   roc prob[((jj-
myIndices=sample(length(myY))[1:round(length(myY)/nFol
                                                                                                                             1)*round(length(myY)/nFolds)+1):(jj*round(length(myY)/n
                                                                                                                             Folds)),ii,kk] = tuningProb[jj,ii,kk,]
 myYfactor = as.factor(myY)
                                                                                                                                }
 myXtrain = myX[-myIndices,]
 myYtrain = myYfactor[-myIndices]
                                                                                                                               }
 myXtest = myX[myIndices,]
                                                                                                                             }
 myYtest = myYfactor[myIndices]
                                                                                                                             # compute ROC and plot ROC vs. tuning parameters
 LRtrain = data.frame(myXtrain, myYtrain)
                                                                                                                             myRocList = list()
                                                                                                                             tuningAUC = matrix(NA,length(myDirections),length(myKs))
 for (ii in 1:length(myDirections)){
                                                                                                                             for (ii in 1:length(myDirections)){
    # Set tuning parameters
                                                                                                                               for (kk in 1:length(myKs)){
                                                                                                                                  myRocList[[(ii-1)*length(myKs)+kk]] = roc(response =
    myDirection = myDirections[ii]
    for (kk in 1:length(myKs)){
                                                                                                                             roc_res, predictor = roc_prob[,ii,kk], auc.polygon=TRUE,
     myK = myKs[kk]
                                                                                                                             grid=TRUE, plot=TRUE)
     # Calculate the NN model
                                                                                                                                 tuningAUC[ii,kk] = myRocList[[(ii-
     LRmodel = glm( myYtrain~ AOD750 + TISA750 + IT750
                                                                                                                             1)*length(myKs)+kk]]$auc
+IT2000 + ITCM + IAREA + ICURV + ACW mm + ACA + ACV
                                                                                                                               }
+ LENSVAULT, family = binomial(logit),data = LRtrain)
     LRmodelstep = step(LRmodel,k=myK,direction =
                                                                                                                             plot(tuningAUC)
myDirection)
                                                                                                                             image(t(tuningAUC[nrow(tuningAUC):1,]), axes=FALSE,
                                                                                                                             zlim=c(-4,4), col=rainbow(21))
     # Predict probability
                                                                                                                             library(corrplot)
     LRProb = predict(LRmodelstep, newdata =
                                                                                                                             rownames(tuningAUC) = c("forward","backward","both")
data.frame(myXtest), type = "response")
                                                                                                                             colnames(tuningAUC) =
     tuningProb[jj,ii,kk,] = LRProb
                                                                                                                             c("k=1","k=2","k=3","k=4","k=5","k=6","k=7","k=8","k=9")
   }
                                                                                                                             corrplot(tuningAUC, method = "ellipse", order = "original",
                                                                                                                             is.corr = FALSE,col =
 tuningRes[jj,] = myYtest
                                                                                                                             colorRampPalette(c("green", "navyblue"))(100))
```

Appendix IV Stacking Model

```
# Read in angle closure data
myData=read.csv("AngleClosure.csv",header=TRUE,na.stri
                                                               for (kk in 1:niter){
ngs=c("NA","."))
                                                                myIndices=sample(length(myY))[1:foldLength]
# Set-up response and predictor variables
                                                                myXtrain = myX[-myIndices,]
myResponse=as.numeric(myData$ANGLE.CLOSURE=="YES
                                                                myYtrain = myY[-myIndices]
                                                                myXtest = myX[myIndices,]
# omit the variables HGT, WT, ASPH, ACYL, SE, AXL, CACD,
                                                                myYtest = myY[myIndices]
AGE, CCT.OD, and PCCURV_mm
myPredictors=data.matrix(myData[,!(attributes(myData)$
                                                                # yMatrix
names %in%
                                                                yMatrix[((kk-1)*foldLength+1):(kk*foldLength)] = myYtest
c("EYE", "GENDER", "ETHNIC", "ANGLE.CLOSURE", "HGT", "W
                                                                T","ASPH", "ACYL", "SE", "AXL", "CACD", "AGE", "CCT.OD",
                                                                # SVM
"PCCURV_mm"))])
# Remove rows with any missingness
                                                                svm.model = svm(myXtrain, myYtrain, type = "C", gamma
myLogical=apply(cbind(myResponse,myPredictors),1,funct
                                                               = 0.01, cost = 0.1, probability = TRUE)
                                                                svm.prob = predict(svm.model, myXtest, probability =
ion(xx){
return(!any(is.na(xx)))
                                                               TRUE)
})
                                                                miuMatrix[((kk-1)*foldLength+1):(kk*foldLength),1] =
myResponse=myResponse[myLogical]
                                                               attr(svm.prob,"probabilities")[,colnames(attr(svm.prob,"pr
myPredictors=myPredictors[myLogical,]
                                                               obabilities"))==1]
                                                                # Neural Network
# make all data mean zero and variance one
                                                                myYtrainFactor = as.factor(myYtrain)
myPredictors mean = apply(myPredictors,2,function(xx){
return(mean(xx,na.rm=TRUE))
                                                               fit=nnet(myYtrainFactor~.,data=myXtrain,weights=rep(1,le
myPredictors_sd = apply(myPredictors,2,function(xx){
                                                               ngth(myYtrainFactor)),size=2,
return(sd(xx,na.rm=TRUE))
})
                                                               decay=0.0005, MaxNWts=10000, maxit=10000, trace=FALSE)
                                                                NNProb=predict(fit,newdata=myXtest, type = "raw")
myX=apply(myPredictors,2,function(xx){
                                                                miuMatrix[((kk-1)*foldLength+1):(kk*foldLength),2] =
return((xx-mean(xx,na.rm=TRUE))/sd(xx,na.rm=TRUE))
                                                               NNProb
})
myY = myResponse
                                                                # random forest
                                                                RFmodel = randomForest(x = myXtrain, y =
########################### calculate the cross-
                                                               myYtrainFactor, ntree = 100)
validation results using five tuned
                                                                RFProb = predict(RFmodel, myXtest, type = "prob")[,2]
miuMatrix[((kk-1)*foldLength+1):(kk*foldLength),3] =
nFolds = 10
                                                               RFProb
niter = 10
foldLength = round(length(myY)/nFolds)
                                                                # boosted model
yMatrix = matrix(NA, foldLength*nFolds,1)
                                                                myYfactor = as.factor(myY)
miuMatrix = matrix(NA, foldLength*nFolds,5)
                                                                adadata = data.frame(myYfactor,myX)
                                                                adaTrain = adadata[-myIndices,]
library(e1071)
                                                                adaTest = adadata[myIndices,]
library(nnet)
                                                                adamodel =
library(randomForest)
                                                               boosting(myYfactor~.,data=adaTrain,mfinal=100,coeflearn
library(adabag)
                                                               ="Freund",
library(adabag)
                                                                           control=rpart.control(maxdepth=2))
```

```
adaPred = predict(adamodel, newdata=adaTest)
                                                                  miuMatrix[((kk-1)*foldLength+1):(kk*foldLength),5] =
adaProb = adaPred$prob[,2]
                                                                  LRProb
miuMatrix[((kk-1)*foldLength+1):(kk*foldLength),4] =
adaProb
                                                                  }
                                                                  D = 2*t(miuMatrix)%*%miuMatrix
# Logistic Regression
myYtrain = myYfactor[-myIndices]
                                                                  d = 2*t(miuMatrix)%*%yMatrix
LRtrain = data.frame(myXtrain, myYtrain)
                                                                  A = t(rbind(matrix(1,1,5),matrix(-1,1,5),diag(c(1,1,1,1,1))))
LRmodel = glm( myYtrain~ AOD750 + TISA750 + IT750
                                                                  b = c(1, -1, 0, 0, 0, 0, 0)
+IT2000 + ITCM + IAREA + ICURV + ACW_mm + ACA + ACV
                                                                  library(quadprog)
+ LENSVAULT, family = binomial(logit),data = LRtrain)
                                                                  solution = solve.QP(D,d,A,b,factorized = FALSE)
LRmodelstep = step(LRmodel,k=5,direction = "both")
LRProb = predict(LRmodelstep, newdata =
                                                                  weight1 = solution$solution
data.frame(myXtest), type = "response")
                                                                  weight2 = solution$unconstrained.solution
```

Appendix V Validation

```
# Read in training data
myData=read.csv("AngleClosure.csv",header=TRUE,na.stri
                                                              decay=0.0005, MaxNWts=10000, maxit=10000, trace=FALSE)
ngs=c("NA","."))
                                                              # random forest
# Set-up response and predictor variables
                                                               RFmodel = randomForest(x = myXtrain, y = myYtrainFactor,
myResponse=as.numeric(myData$ANGLE.CLOSURE=="YES
                                                              ntree = 100)
# omit the variables HGT, WT, ASPH, ACYL, SE, AXL, CACD,
                                                              # boosted model
AGE, CCT.OD, and PCCURV_mm
                                                               myYfactor = as.factor(myYtrain)
myPredictors=data.matrix(myData[,!(attributes(myData)$
                                                              adaTrain = data.frame(myYfactor,myXtrain)
names %in%
                                                               adamodel =
                                                              boosting(myYfactor~.,data=adaTrain,mfinal=100,coeflearn
c("EYE", "GENDER", "ETHNIC", "ANGLE.CLOSURE", "HGT", "W
                                                               ="Freund",
T","ASPH", "ACYL", "SE", "AXL", "CACD", "AGE", "CCT.OD",
                                                                        control=rpart.control(maxdepth=2))
"PCCURV mm"))])
                                                              # Logistic Regression
# Remove rows with any missingness
                                                               myYtrain = myYfactor
myLogical=apply(cbind(myResponse,myPredictors),1,funct
                                                               LRtrain = data.frame(myXtrain, myYtrain)
                                                              LRmodel = glm( myYtrain~ AOD750 + TISA750 + IT750
ion(xx){
return(!any(is.na(xx)))
                                                              +IT2000 + ITCM + IAREA + ICURV + ACW mm + ACA + ACV
})
                                                              + LENSVAULT, family = binomial(logit),data = LRtrain)
myResponse=myResponse[myLogical]
                                                               LRmodelstep = step(LRmodel,k=5,direction = "both")
myPredictors=myPredictors[myLogical,]
                                                               ##################################### Read in the
# make all data mean zero and variance one
                                                              myPredictors mean = apply(myPredictors,2,function(xx){
return(mean(xx,na.rm=TRUE))
myPredictors_sd = apply(myPredictors,2,function(xx){
                                                              myData1=read.csv("AngleClosure_ValidationCases.csv",he
return(sd(xx,na.rm=TRUE))
                                                               ader=TRUE,na.strings=c("NA","."))
})
                                                              myData2=read.csv("AngleClosure ValidationControls.csv",
                                                              header=TRUE,na.strings=c("NA","."))
                                                              # Set-up response and predictor variables
myX=apply(myPredictors,2,function(xx){
return((xx-mean(xx,na.rm=TRUE))/sd(xx,na.rm=TRUE))
                                                               myResponse1 = matrix(1,dim(myData1)[1],1)
                                                               myResponse2= matrix(0,dim(myData2)[1],1)
})
myY = myResponse
############################### get the 5 models
                                                              # create the the predictors Y=1
right1 =
                                                              c("rAOD750","rTISA750","rIT750","IT2000","rITCM","rIARE
                                                               A","rICURV", "ACWmm", "ACA", "ACV", "LENSVAULT")
myXtrain = myX
                                                              myPredictors1_right=data.matrix(myData1[,(attributes(my
myYtrain = myY
                                                               Data1)$names %in% right1)])
                                                              colnames(myPredictors1 right)=
                                                               c("AOD750","TISA750","IT750","IT2000","ITCM","IAREA","I
svm.model = svm(myXtrain, myYtrain, type = "C", gamma =
                                                              CURV", "ACW mm", "ACA", "ACV", "LENSVAULT")
0.01, cost = 0.1, probability = TRUE)
# Neural Network
mvYtrainFactor = as.factor(myYtrain)
                                                              c("IAOD750"."ITISA750"."IIT750"."IIT2000"."IITCM"."IIARE
fit=nnet(myYtrainFactor~.,data=myXtrain,weights=rep(1,le
                                                              A","IICURV", "ACWmm", "ACA", "ACV", "LENSVAULT")
ngth(myYtrainFactor)),size=2,
```

```
myPredictors1_left=data.matrix(myData1[,(attributes(myD
ata1)$names %in% left1)])
                                                                 myPredictors = rbind(myPredictors1,myPredictors2)
colnames(myPredictors1 left)=
c("AOD750","TISA750","IT750","IT2000","ITCM","IAREA","I
                                                                  # make all data mean zero and variance one uisng the
CURV", "ACW mm", "ACA", "ACV", "LENSVAULT")
                                                                  training mean and training variance
                                                                 for (ii in 1:dim(myPredictors)[2]){
for (ii in 1:dim(myPredictors1 right)[2]){
                                                                  for (jj in 1:dim(myPredictors)[1]){
for (jj in 1:dim(myPredictors1 right)[1]){
                                                                   myPredictors[jj,ii] = (myPredictors[jj,ii]-
 if (is.na(myPredictors1_right[jj,ii])){
                                                                 myPredictors_mean[ii])/myPredictors_sd[ii]
   myPredictors1_right[jj,ii] = myPredictors1_left[jj,ii]
                                                                  }
 }
                                                                 }
}
                                                                 myXtest = myPredictors
                                                                 myYtest =
                                                                 rbind(matrix(1,dim(myPredictors1)[1],1),matrix(0,dim(myP
# Remove rows with any missingness
myLogical=apply(myPredictors1_right,1,function(xx){
                                                                 redictors2)[1],1))
return(!any(is.na(xx)))
                                                                  ########################## get the results from 7
})
myPredictors1 = myPredictors1_right[myLogical,]
                                                                  # create the the predictors Y=0
                                                                 miuMatrix = matrix(NA,length(myYtest),7)
right2 =
c("rAOD750","rTISA750","rIT750","rIT2000","rITCM","rIAR
                                                                 svm.prob = predict(svm.model, myXtest, probability =
EA", "rICURV", "ACW.mm.", "ACA", "ACV", "LENSVAULT")
                                                                 TRUE)
myPredictors2_right=data.matrix(myData2[,(attributes(my
                                                                 miuMatrix[,1] =
Data2)$names %in% right2)])
                                                                 attr(svm.prob,"probabilities")[,colnames(attr(svm.prob,"pr
colnames(myPredictors2 right)=
                                                                 obabilities"))==1]
c("AOD750","TISA750","IT750","IT2000","ITCM","IAREA","I
CURV", "ACW_mm", "ACA", "ACV", "LENSVAULT")
                                                                 # Neural Network
                                                                 NNProb=predict(fit,newdata=myXtest, type = "raw")
left2 =
                                                                  miuMatrix[,2] = NNProb
c("IAOD750","ITISA750","IIT750","IIT2000","IITCM","IIARE
A","IICURV.", "ACW.mm.", "ACA", "ACV", "LENSVAULT")
                                                                 # random forest
                                                                  RFProb = predict(RFmodel, myXtest, type = "prob")[,2]
myPredictors2_left=data.matrix(myData2[,(attributes(myD
ata2)$names %in% left2)])
                                                                  miuMatrix[,3] = RFProb
colnames(myPredictors2 left)=
c("AOD750","TISA750","IT750","IT2000","ITCM","IAREA","I
                                                                 # boosted model
CURV", "ACW_mm", "ACA", "ACV", "LENSVAULT")
                                                                 myYfactor = as.factor(myYtest)
                                                                 adaTest = data.frame(myYfactor,myXtest)
for (ii in 1:dim(myPredictors2 right)[2]){
                                                                 adaPred = predict(adamodel, newdata=adaTest)
for (jj in 1:dim(myPredictors2 right)[1]){
                                                                 adaProb = adaPred$prob[,2]
 if (is.na(myPredictors2_right[jj,ii])){
                                                                 miuMatrix[,4] = adaProb
   myPredictors2_right[jj,ii] = myPredictors2_left[jj,ii]
                                                                 # Logistic Regression
 }
}
                                                                 LRProb = predict(LRmodelstep, newdata =
                                                                  data.frame(myXtest), type = "response")
# Remove rows with any missingness
                                                                 miuMatrix[,5] = LRProb
myLogical=apply(myPredictors2_right,1,function(xx){
return(!any(is.na(xx)))
                                                                 # Stacking 1
})
                                                                 weight1 = c(0,0,0.11378,0.00252,0.88369)
                                                                 miuMatrix[,6] = miuMatrix[,1:5]%*%weight1
myPredictors2 = myPredictors2_right[myLogical,]
```

```
# Stacking 2
                                                                   par(mar=c(0.075,0.075,1.5,0.075),mfrow=c(3,3))
weight2 = c(-0.03079, -0.21070, 0.14418, -0.02147, 1.07496)
                                                                   model = c("SVM", "NNet", "RandF", "Boost",
                                                                   "LogistR", "StackCon", "StackUncon")
miuMatrix[,7] = miuMatrix[,1:5]%*%weight2
                                                                   for (ii in 1:7){
# plot the ROC and compute the AUC
                                                                    myRocList[[ii]] = roc(response = myYtest, predictor =
library(pROC)
                                                                   miuMatrix[,ii], auc.polygon=TRUE, grid=TRUE, plot=TRUE)
myRocList = list()
                                                                    myAUC[ii,1] = myRocList[[ii]]$auc
myAUC = matrix(NA,7,1)
                                                                    title(main = paste(model[ii],": AUC =
                                                                   ",as.character(round(myAUC[ii,1], digits=4))))
dev.new(width=8,height=8)
                                                                   }
```