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***.

Please check Appendix III and V for r code.

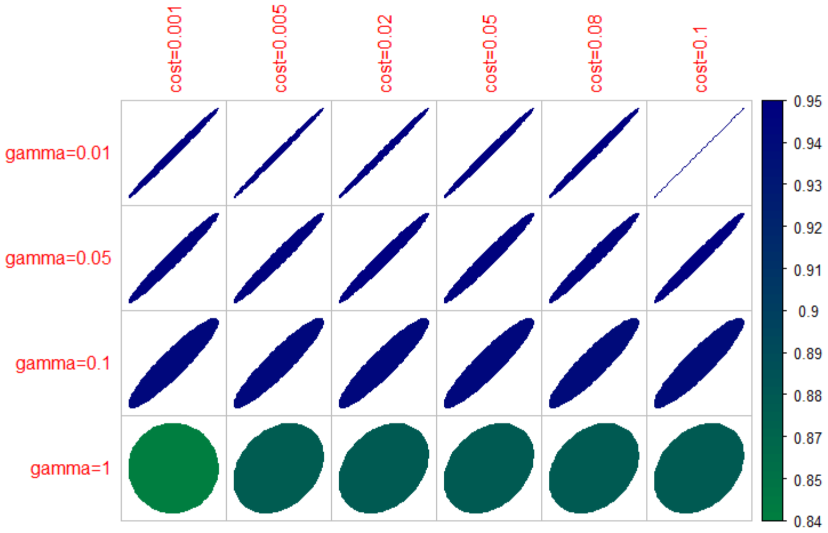

Figure 1 Support Vector Machine: Gamma = 0.01 & Cost = 0.1

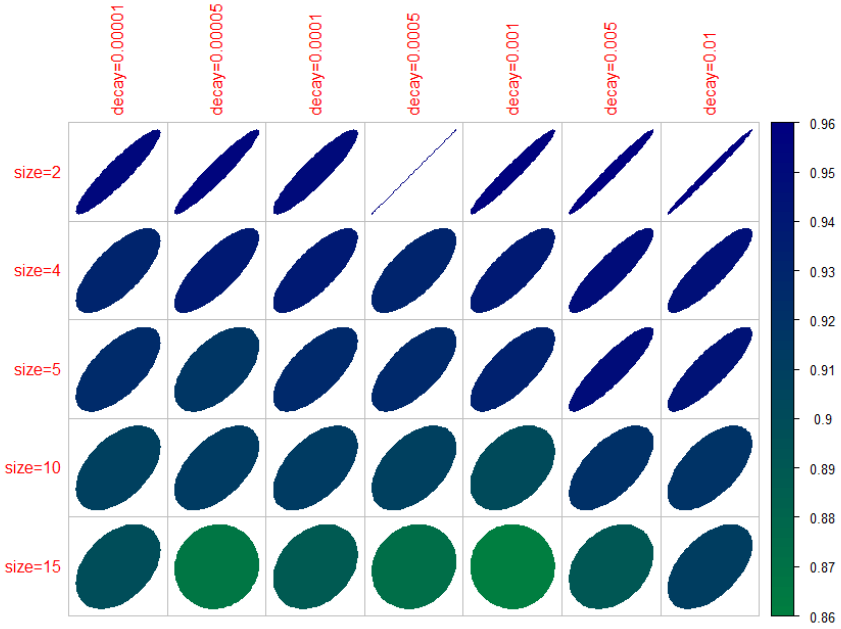

Figure 2 Neural Network: size = 2 & decay = 0.0005

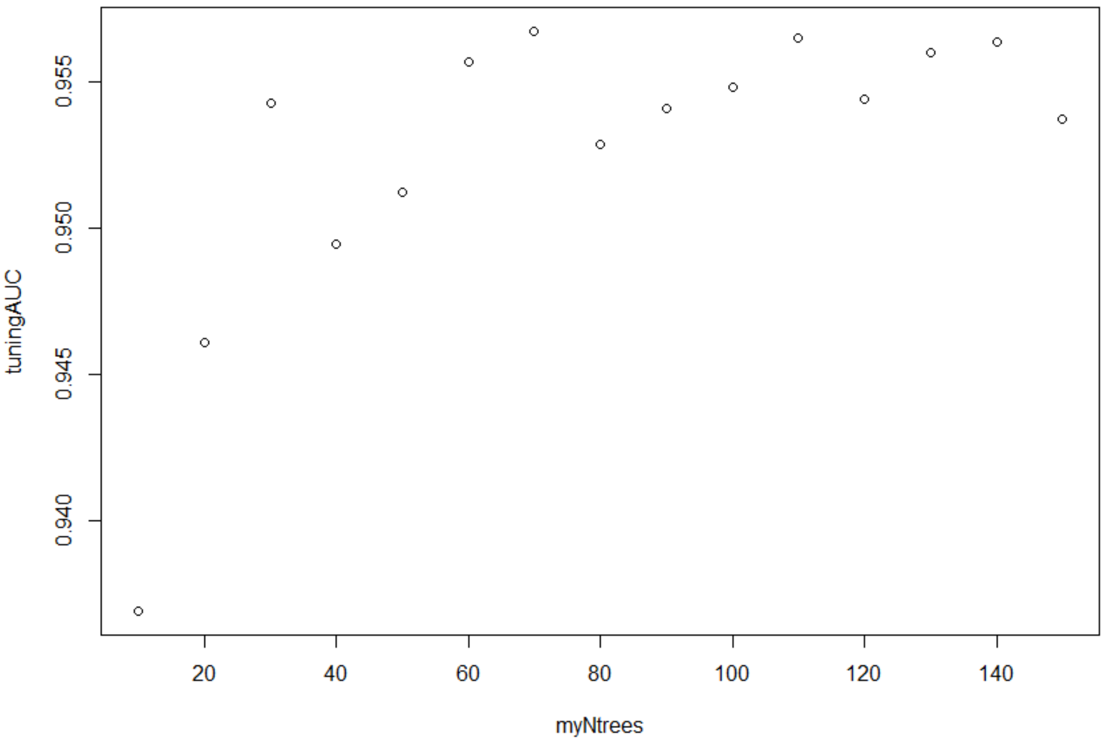

Figure 3 Random Forest: number of trees = 100

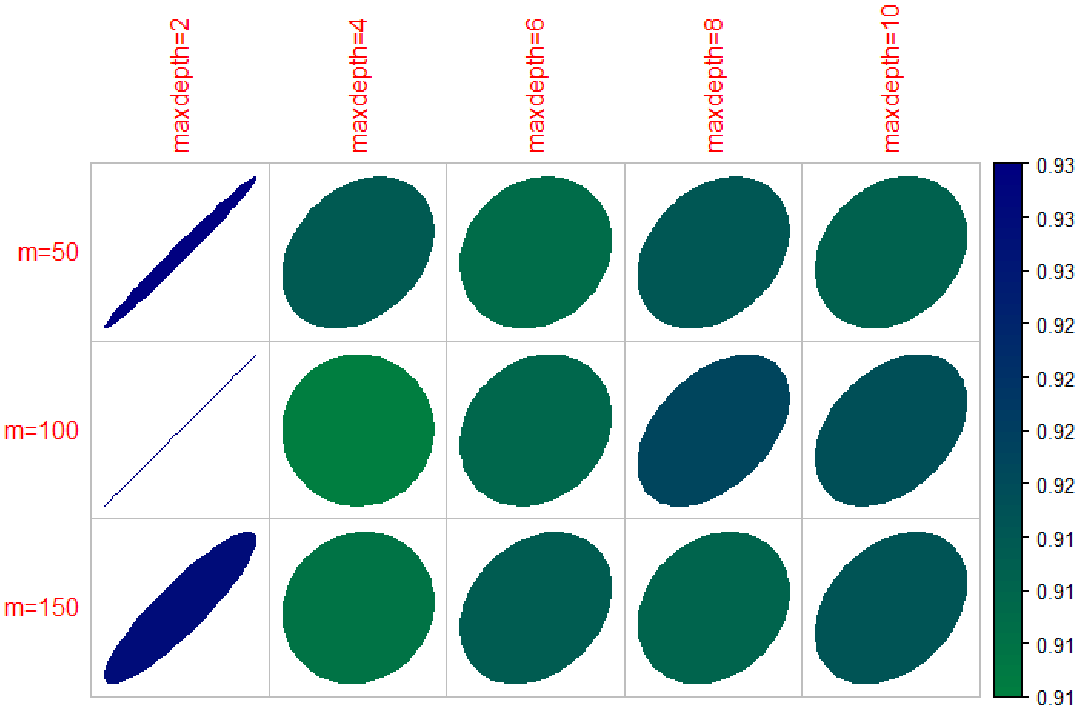

Figure 4 Boosted Model: m = 100 & maxdepth = 2

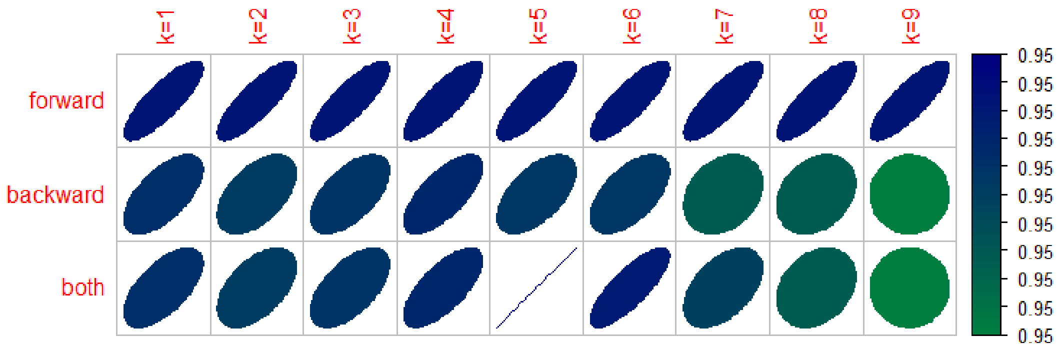

Figure 5 Logistic Regression: direction = "both" & K = 5

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.

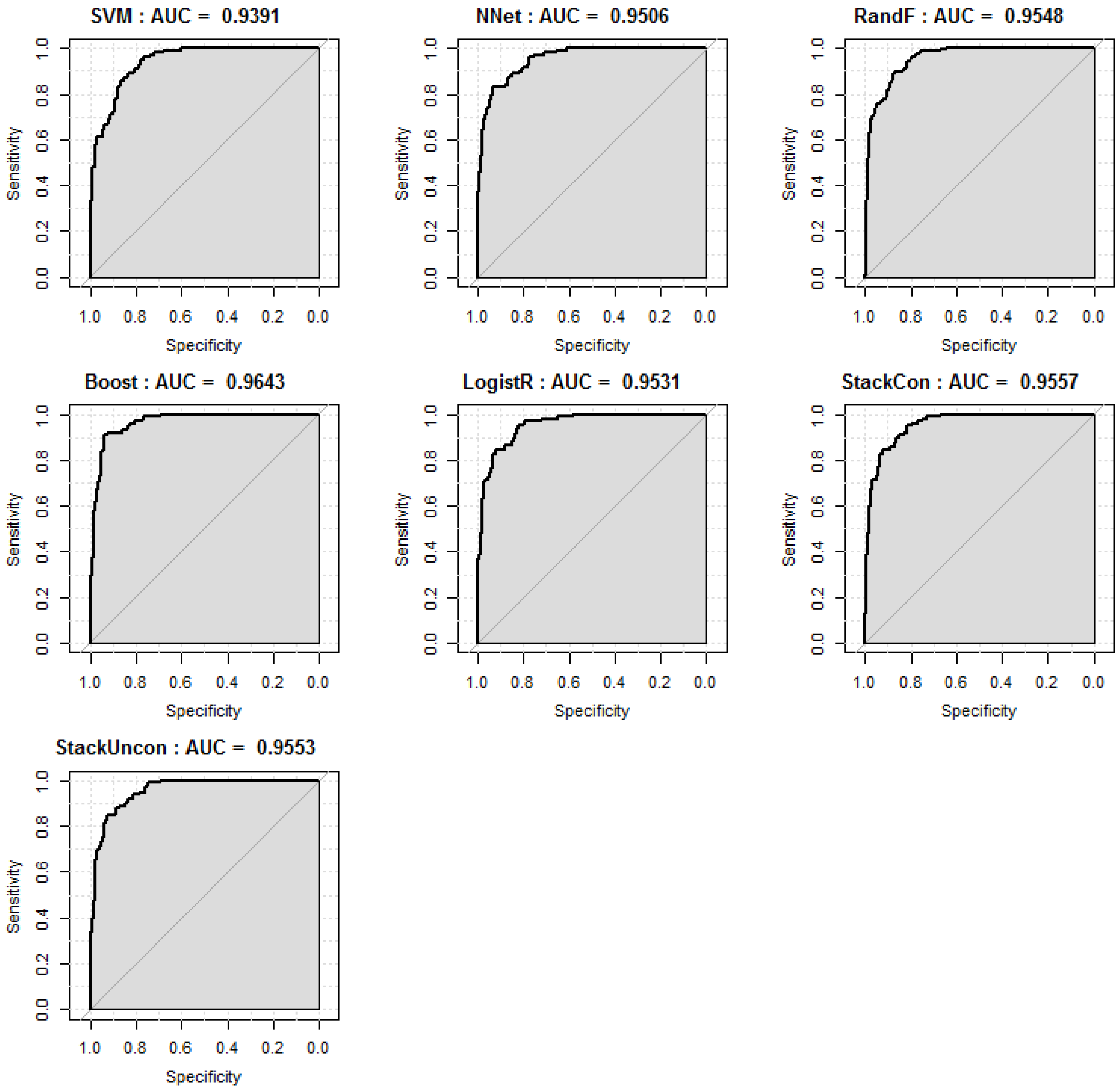

Figure 6 Validation AUCs

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
myData=read.csv("AngleClosure.csv",header=TRUE,na.stri
ngs=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","W
T","ASPH", "ACYL", "SE", "AXL", "CACD", "AGE", "CCT.OD",
"PCCURV_mm"))])
# Remove rows with any missingness
myLogical=apply(cbind(myResponse,myPredictors),1,funct
ion(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

############# Develop Prediction Models
###################
# creat a taining set and varification set
set.seed(30126)
nFolds=10
myIndices=sample(length(myResponse),ceiling(length(myR
esponse)/nFolds))
trainingX = myX[-myIndices,]
trainingY = myY[-myIndices]
testX = myX[myIndices,]
testY = myY[myIndices]

myYFactor = factor(myY)
trainingYFactor = myYFactor[-myIndices]
testYFactor = myYFactor[myIndices]

# Support vector machine ~ cost
library(e1071)
svm.model = svm(trainingX, trainingY, type = "C", cost =
10,probability = TRUE)
svm.pred <- predict(svm.model, testX)
svm.prob = predict(svm.model, testX, probability = TRUE)
svmt=table(pred = svm.pred, true = testY)
svmt

# neural network ~ size ~ decay
library(nnet)
lambda = 0.0001
fit=nnet(trainingYFactor~.,data=trainingX,weights=rep(1,le
ngth(trainingYFactor)),size=10,
        decay=lambda,MaxNWts=10000,maxit=250)
NNPred=predict(fit,newdata=testX, type = "class")
NNProb=predict(fit,newdata=testX, type = "raw")
NNt=table(pred = NNPred, true = testY)
NNt

# random forest ~ ntree
library(randomForest)
RFmodel = randomForest(x = trainingX, y = trainingYFactor,
ntree = 10)
RFPred = predict(RFmodel, testX, type = "class")
RFProb = predict(RFmodel, testX, type = "prob")[,2]
RFt=table(pred = RFPred, true = testY)
RFt

# boosted model ~ mfinal ~ maxdepth
library(adabag)
myYfactor = as.factor(myY)
adadata = data.frame(myYfactor,myX)
adaTrain = adadata[-myIndices,]
adaTest = adadata[myIndices,]
myM = 100
adamodel =
boosting(myYfactor~.,data=adaTrain,mfinal=myM,coeflear
n="Freund",
                  control=rpart.control(maxdepth=10))
adaPred = predict(adamodel, newdata=adaTest)
adaProb = adaPred$prob[,2]
adat=table(pred = adaPred$class, true = testY)
adat

# logistic regression ~ k ~ direction
LRdata = data.frame(trainingX, trainingYFactor)
```

```
LRtestdata = data.frame(testX, testYFactor)
LRmodel = glm( trainingYFactor~ AOD750 + TISA750 +
IT750 +IT2000 + ITCM + IAREA + ICURV + ACW_mm + ACA
+ ACV + LENSVAULT, family = binomial(logit),data = LRdata)
LRmodelstep = step(LRmodel,k=2,direction = "both")
LRProb = predict(LRmodelstep, newdata =
data.frame(testX), type = "response")
# LRPred = ifelse(predict(LRmodel, newdata =
data.frame(testX), type = "response")>.5,1,0)
# LRProb = predict(LRmodel, newdata = data.frame(testX),
type = "response")
LRt=table(pred = LRPred, true = testY)
LRt

## 10 folds validation
library(pROC)
# Support vector machine ~ cost
library(e1071)
set.seed(123456789)
nFolds = 10
table = matrix(0,2,2)
for (jj in 1:nFolds){
  # Generate training and testing responses and predictors
for each fold

myIndices=sample(length(myY))[1:round(length(myY)/nFol
ds)]
  myXtrain = myX[-myIndices,]
  myYtrain = myY[-myIndices]
  myXtest = myX[myIndices,]
  myYtest = myY[myIndices]

  # Set tuning parameters
  myC = 10

  # Calculate the SVM model
  svm.model = svm(myXtrain, myYtrain, type = "C", cost =
myC,probability = TRUE)

  # Set threshold
  threshold = 0.5
  svm.prob = predict(svm.model, myXtest, probability =
TRUE)
  svm.pred =
ifelse(attr(svm.prob,"probabilities")[,colnames(attr(svm.pr
ob,"probabilities"))==1]>threshold,1,0)
  svmt = table(pred = svm.pred, true = myYtest)
  print(svmt)
  table = table + svmt
  myRoc <- roc(response = myYtest, predictor =
attr(svm.prob,"probabilities")[,colnames(attr(svm.prob,"pr
obabilities"))==1], auc.polygon=TRUE, grid=TRUE,
plot=FALSE)
  plot(myRoc)
}

######################### tuning Parameter
############################

## SVM ~ cost

library(e1071)
set.seed(123456789)
nFolds = 10
table = matrix(0,2,2)
for (jj in 1:nFolds){
  # Generate training and testing responses and predictors
for each fold

myIndices=sample(length(myY))[1:round(length(myY)/nFol
ds)]
  myXtrain = myX[-myIndices,]
  myYtrain = myY[-myIndices]
  myXtest = myX[myIndices,]
  myYtest = myY[myIndices]

  # Set tuning parameters
  myC = 10

  # Calculate the SVM model
  svm.model = svm(myXtrain, myYtrain, type = "C", cost =
myC,probability = TRUE)

  # Set threshold
  threshold = 0.5
  svm.prob = predict(svm.model, myXtest, probability =
TRUE)
  svm.pred =
ifelse(attr(svm.prob,"probabilities")[,colnames(attr(svm.pr
ob,"probabilities"))==1]>threshold,1,0)
  svmt = table(pred = svm.pred, true = myYtest)
  print(svmt)
  table = table + svmt
  myRoc <- roc(response = myYtest, predictor =
attr(svm.prob,"probabilities")[,colnames(attr(svm.prob,"pr
obabilities"))==1], auc.polygon=TRUE, grid=TRUE,
plot=FALSE)
  plot(myRoc)
}
```

Appendix III Tuning Parameters

```
# 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


########################################## tuning Parameter ########################################

## SVM ~ cost ~ gamma


library(e1071)
set.seed(123456789)
nFolds = 10
iter = 10
myCs = c(0.001,0.005,0.02,0.05,0.08,0.1)
myGammas = c(0.01,0.05,0.1,1)
tuningProb = array(NA,dim = c(iter,length(myCs),length(myGammas),round(length(myY)/nFolds)))
tuningRes = array(NA,dim = c(iter,round(length(myY)/nFolds)))
for (jj in 1:iter){

  # Generate training and testing responses and predictors for each fold

myIndices=sample(length(myY))[1:round(length(myY)/nFolds)]
  myXtrain = myX[-myIndices,]
  myYtrain = myY[-myIndices]
  myXtest = myX[myIndices,]
  myYtest = myY[myIndices]
  for (ii in 1:length(myCs)){
    # Set tuning parameters
    myC = myCs[ii]
    for (kk in 1:length(myGammas)){
      myGamma = myGammas[kk]
      # Calculate the SVM model
      svm.model = svm(myXtrain, myYtrain, type = "C", gamma = myGamma, cost = myC,probability = TRUE)
      # Predict probability
      svm.prob = predict(svm.model, myXtest, probability = TRUE)
      tuningProb[jj,ii,kk,] = attr(svm.prob,"probabilities")[,colnames(attr(svm.prob,"probabilities"))==1]
    }
  }
  tuningRes[jj,] = myYtest
}

# 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(myCs),length(myGammas)))
for (jj in 1:iter){
  roc_res[((jj-1)*round(length(myY)/nFolds)+1):(jj*round(length(myY)/nFolds))] = tuningRes[jj,]
  for (ii in 1:length(myCs)){
    for (kk in 1:length(myGammas)){
      roc_prob[((jj-
1)*round(length(myY)/nFolds)+1):(jj*round(length(myY)/n
Folds)),ii,kk] = tuningProb[jj,ii,kk,]
    }

  }
}

# compute ROC and plot ROC vs. tuning parameters
myRocList = list()
tuningAUC = matrix(NA,length(myCs),length(myGammas))
for (ii in 1:length(myCs)){
  for (kk in 1:length(myGammas)){
    myRocList[[(ii-1)*length(myGammas)+kk]] =
roc(response = roc_res, predictor = roc_prob[,ii,kk],
auc.polygon=TRUE, grid=TRUE, plot=TRUE)
    tuningAUC[ii,kk] = myRocList[[(ii-
1)*length(myGammas)+kk]]$auc
  }
}
library(corrplot)
rownames(tuningAUC) =
c("cost=0.001","cost=0.005","cost=0.02","cost=0.05","cost
=0.08","cost=0.1")
colnames(tuningAUC) =
c("gamma=0.01","gamma=0.05","gamma=0.1","gamma=1
")
corrplot(t(tuningAUC), method = "ellipse", order =
"original", is.corr = FALSE,col =
colorRampPalette(c("green","navyblue"))(100))
```

```
########################################### tuning
Parameter #########################################

## Neural Network ~ size ~ decay
library(nnet)
set.seed(123456789)
nFolds = 10
iter = 10
mySizes = c(2,4,5,10,15)
myDecays =
c(0.00001,0.00005,0.0001,0.0005,0.001,0.005,0.01)
tuningProb = array(NA,dim =
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){
```

```
  # Generate training and testing responses and predictors
for each fold

myIndices=sample(length(myY))[1:round(length(myY)/nFol
ds)]
  myXtrain = myX[-myIndices,]
  myYtrain = myY[-myIndices]
  myYtrainFactor = as.factor(myYtrain)
  myXtest = myX[myIndices,]
  myYtest = myY[myIndices]
  for (ii in 1:length(mySizes)){
    # Set tuning parameters
    mySize = mySizes[ii]
    for (kk in 1:length(myDecays)){
      myDecay = myDecays[kk]
      # Calculate the NN model

fit=nnet(myYtrainFactor~.,data=myXtrain,weights=rep(1,le
ngth(myYtrainFactor)),size=mySize,

decay=myDecay,MaxNWts=10000,maxit=10000,trace=FAL
SE)
      # Predict probability
      NNProb=predict(fit,newdata=myXtest, type = "raw")
      tuningProb[jj,ii,kk,] = NNProb
    }
  }
  tuningRes[jj,] = myYtest
}

# 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)))
for (jj in 1:iter){
  roc_res[((jj-
1)*round(length(myY)/nFolds)+1):(jj*round(length(myY)/n
Folds))] = tuningRes[jj,]
  for (ii in 1:length(mySizes)){
    for (kk in 1:length(myDecays)){
      roc_prob[((jj-
1)*round(length(myY)/nFolds)+1):(jj*round(length(myY)/n
Folds)),ii,kk] = tuningProb[jj,ii,kk,]
    }

  }
}
```

```
# compute ROC and plot ROC vs. tuning parameters
myRocList = list()
tuningAUC = matrix(NA,length(mySizes),length(myDecays))
for (ii in 1:length(mySizes)){
  for (kk in 1:length(myDecays)){
    myRocList[[(ii-1)*length(myDecays)+kk]] = roc(response
= roc_res, predictor = roc_prob[,ii,kk], auc.polygon=TRUE,
grid=TRUE, plot=TRUE)
    tuningAUC[ii,kk] = myRocList[[(ii-
1)*length(myDecays)+kk]]$auc
  }
}
plot(tuningAUC)
image(t(tuningAUC[nrow(tuningAUC):1,] ), axes=FALSE,
zlim=c(-4,4), col=rainbow(21))
library(corrplot)
rownames(tuningAUC) =
c("size=2","size=4","size=5","size=10","size=15")
colnames(tuningAUC) =
c("decay=0.00001","decay=0.00005","decay=0.0001","dec
ay=0.0005","decay=0.001","decay=0.005","decay=0.01")
corrplot(tuningAUC, method = "ellipse", order = "original",
is.corr = FALSE,col =
colorRampPalette(c("green","navyblue"))(100))


########################################### tuning
Parameter #########################################

## random forest ~ ntree
library(randomForest)
set.seed(123456789)
nFolds = 10
iter = 10
myNtrees = seq(10,150,10)
tuningProb = array(NA,dim =
c(iter,length(myNtrees),round(length(myY)/nFolds)))
tuningRes = array(NA,dim =
c(iter,round(length(myY)/nFolds)))
for (jj in 1:iter){

  # Generate training and testing responses and predictors
for each fold

myIndices=sample(length(myY))[1:round(length(myY)/nFol
ds)]
  myXtrain = myX[-myIndices,]
  myYtrain = myY[-myIndices]
  myYtrainFactor = as.factor(myYtrain)
  myXtest = myX[myIndices,]
  myYtest = as.factor(myY[myIndices])
  for (ii in 1:length(myNtrees)){
    # Set tuning parameters
    myNtree = myNtrees[ii]
    # Calculate the RF model
    RFmodel = randomForest(x = myXtrain, y =
myYtrainFactor, ntree = myNtree)
    # Predict probability
    RFProb = predict(RFmodel, myXtest, type = "prob")[,2]
    tuningProb[jj,ii,] = RFProb
  }
  tuningRes[jj,] = myYtest
}

# 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(myNtrees)))
for (jj in 1:iter){
  roc_res[((jj-
1)*round(length(myY)/nFolds)+1):(jj*round(length(myY)/n
Folds))] = tuningRes[jj,]
  for (ii in 1:length(myNtrees)){
      roc_prob[((jj-
1)*round(length(myY)/nFolds)+1):(jj*round(length(myY)/n
Folds)),ii] = tuningProb[jj,ii,]
  }
}

# compute ROC and plot ROC vs. tuning parameters
myRocList = list()
tuningAUC = matrix(NA,length(myNtrees),1)
for (ii in 1:length(myNtrees)){
    myRocList[[ii]] = roc(response = roc_res, predictor =
roc_prob[,ii], auc.polygon=TRUE, grid=TRUE, plot=TRUE)
    tuningAUC[ii,1] = myRocList[[ii]]$auc
}

plot(myNtrees,tuningAUC)


########################################### tuning
Parameter #########################################

# boosted model ~ mfinal ~ maxdepth
library(adabag)
set.seed(123456789)
nFolds = 10
iter = 6
myMs = c(50,100,150)
# myMs = c(50,100,150,200,250)
# myMaxdepths = c(5,10,15,20,25)
myMaxdepths = c(2,4,6,8,10)
tuningProb = array(NA,dim =
c(iter,length(myMs),length(myMaxdepths),round(length(
myY)/nFolds)))
tuningRes = array(NA,dim =
c(iter,round(length(myY)/nFolds)))

for (jj in 1:iter){
  # Generate training and testing responses and predictors
for each fold

myIndices=sample(length(myY))[1:round(length(myY)/nFol
ds)]
  myYfactor = as.factor(myY)
  adadata = data.frame(myYfactor,myX)
  adaTrain = adadata[-myIndices,]
  adaTest = adadata[myIndices,]

  for (ii in 1:length(myMs)){
    print("Fuck")
    # Set tuning parameters
    myM = myMs[ii]
    for (kk in 1:length(myMaxdepths)){
      print(kk)
      myMaxdepth = myMaxdepths[kk]
      # Calculate the NN model
      adamodel =
boosting(myYfactor~.,data=adaTrain,mfinal=myM,coeflear
n="Freund",

control=rpart.control(maxdepth=myMaxdepth))
      # Predict probability
      adaPred = predict(adamodel, newdata=adaTest)
      adaProb = adaPred$prob[,2]
      tuningProb[jj,ii,kk,] = adaProb
    }
  }
  tuningRes[jj,] = adaTest[,1]
  print("You!")
}

# 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){
  roc_res[((jj-
1)*round(length(myY)/nFolds)+1):(jj*round(length(myY)/n
Folds))] = tuningRes[jj,]
  for (ii in 1:length(myMs)){
    for (kk in 1:length(myMaxdepths)){
      roc_prob[((jj-
1)*round(length(myY)/nFolds)+1):(jj*round(length(myY)/n
Folds)),ii,kk] = tuningProb[jj,ii,kk,]
    }

  }
}

# compute ROC and plot ROC vs. tuning parameters
myRocList = list()
tuningAUC =
matrix(NA,length(myMs),length(myMaxdepths))
for (ii in 1:length(myMs)){
  for (kk in 1:length(myMaxdepths)){
    myRocList[[(ii-1)*length(myMaxdepths)+kk]] =
roc(response = roc_res, predictor = roc_prob[,ii,kk],
auc.polygon=TRUE, grid=TRUE, plot=TRUE)
    tuningAUC[ii,kk] = myRocList[[(ii-
1)*length(myMaxdepths)+kk]]$auc
  }
}
plot(tuningAUC)
image(t(tuningAUC[nrow(tuningAUC):1,] ), axes=FALSE,
zlim=c(-4,4), col=rainbow(21))
library(corrplot)
# rownames(tuningAUC) =
c("m=50","m=100","m=150","m=200","m=250")
# colnames(tuningAUC) =
c("maxdepth=5","maxdepth=10","maxdepth=15","maxdep
th=20","maxdepth=25")
rownames(tuningAUC) = c("m=50","m=100","m=150")
colnames(tuningAUC) =
c("maxdepth=2","maxdepth=4","maxdepth=6","maxdepth
=8","maxdepth=10")
corrplot(tuningAUC, method = "ellipse", order = "original",
is.corr = FALSE,col =
colorRampPalette(c("green","navyblue"))(100))



########################################## tuning
Parameter ########################################

# logistic regression ~ k ~ direction
library(adabag)
set.seed(123456789)
```

```
nFolds = 10
iter = 10
myDirections = c("forward","backward","both")
myKs = c(1,2,3,4,5,6,7,8,9)
tuningProb = array(NA,dim =
c(iter,length(myDirections),length(myKs),round(length(my
Y)/nFolds)))
tuningRes = array(NA,dim =
c(iter,round(length(myY)/nFolds)))

for (jj in 1:iter){
  # Generate training and testing responses and predictors
for each fold

myIndices=sample(length(myY))[1:round(length(myY)/nFol
ds)]
  myYfactor = as.factor(myY)
  myXtrain = myX[-myIndices,]
  myYtrain = myYfactor[-myIndices]
  myXtest = myX[myIndices,]
  myYtest = myYfactor[myIndices]

  LRtrain = data.frame(myXtrain, myYtrain)

  for (ii in 1:length(myDirections)){
    # Set tuning parameters
    myDirection = myDirections[ii]
    for (kk in 1:length(myKs)){
      myK = myKs[kk]
      # Calculate the NN model
      LRmodel = glm( myYtrain~ AOD750 + TISA750 + IT750
+IT2000 + ITCM + IAREA + ICURV + ACW_mm + ACA + ACV
+ LENSVAULT, family = binomial(logit),data = LRtrain)
      LRmodelstep = step(LRmodel,k=myK,direction =
myDirection)

      # Predict probability
      LRProb = predict(LRmodelstep, newdata =
data.frame(myXtest), type = "response")
      tuningProb[jj,ii,kk,] = LRProb
    }
  }
  tuningRes[jj,] = myYtest
}
```

```
# 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(myDirections),len
gth(myKs)))
for (jj in 1:iter){
  roc_res[((jj-
1)*round(length(myY)/nFolds)+1):(jj*round(length(myY)/n
Folds))] = tuningRes[jj,]
  for (ii in 1:length(myDirections)){
    for (kk in 1:length(myKs)){
      roc_prob[((jj-
1)*round(length(myY)/nFolds)+1):(jj*round(length(myY)/n
Folds)),ii,kk] = tuningProb[jj,ii,kk,]
    }

  }
}

# compute ROC and plot ROC vs. tuning parameters
myRocList = list()
tuningAUC = matrix(NA,length(myDirections),length(myKs))
for (ii in 1:length(myDirections)){
  for (kk in 1:length(myKs)){
    myRocList[[(ii-1)*length(myKs)+kk]] = roc(response =
roc_res, predictor = roc_prob[,ii,kk], auc.polygon=TRUE,
grid=TRUE, plot=TRUE)
    tuningAUC[ii,kk] = myRocList[[(ii-
1)*length(myKs)+kk]]$auc
  }
}
plot(tuningAUC)
image(t(tuningAUC[nrow(tuningAUC):1,] ), axes=FALSE,
zlim=c(-4,4), col=rainbow(21))
library(corrplot)
rownames(tuningAUC) = c("forward","backward","both")
colnames(tuningAUC) =
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 =
colorRampPalette(c("green","navyblue"))(100))
```

Appendix IV Stacking Model

```
# Read in angle closure data
myData=read.csv("AngleClosure.csv",header=TRUE,na.stri
ngs=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","W
T","ASPH", "ACYL", "SE", "AXL", "CACD", "AGE", "CCT.OD",
"PCCURV_mm"))])
# Remove rows with any missingness
myLogical=apply(cbind(myResponse,myPredictors),1,funct
ion(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

################################ calculate the cross-
validation results using five tuned
models###########################
nFolds = 10
niter = 10
foldLength = round(length(myY)/nFolds)
yMatrix = matrix(NA, foldLength*nFolds,1)
miuMatrix = matrix(NA, foldLength*nFolds,5)

library(e1071)
library(nnet)
library(randomForest)
library(adabag)
library(adabag)
```

```
for (kk in 1:niter){
  myIndices=sample(length(myY))[1:foldLength]
  myXtrain = myX[-myIndices,]
  myYtrain = myY[-myIndices]
  myXtest = myX[myIndices,]
  myYtest = myY[myIndices]

  # yMatrix
  yMatrix[((kk-1)*foldLength+1):(kk*foldLength)] = myYtest

  ################ miuMatrix ###########

  # SVM
  svm.model = svm(myXtrain, myYtrain, type = "C", gamma
= 0.01, cost = 0.1,probability = TRUE)
  svm.prob = predict(svm.model, myXtest, probability =
TRUE)
  miuMatrix[((kk-1)*foldLength+1):(kk*foldLength),1] =
attr(svm.prob,"probabilities")[,colnames(attr(svm.prob,"pr
obabilities"))==1]

  # Neural Network
  myYtrainFactor = as.factor(myYtrain)

fit=nnet(myYtrainFactor~.,data=myXtrain,weights=rep(1,le
ngth(myYtrainFactor)),size=2,

decay=0.0005,MaxNWts=10000,maxit=10000,trace=FALSE)
  NNProb=predict(fit,newdata=myXtest, type = "raw")
  miuMatrix[((kk-1)*foldLength+1):(kk*foldLength),2] =
NNProb

  # random forest
  RFmodel = randomForest(x = myXtrain, y =
myYtrainFactor, ntree = 100)
  RFProb = predict(RFmodel, myXtest, type = "prob")[,2]
  miuMatrix[((kk-1)*foldLength+1):(kk*foldLength),3] =
RFProb

  # boosted model
  myYfactor = as.factor(myY)
  adadata = data.frame(myYfactor,myX)
  adaTrain = adadata[-myIndices,]
  adaTest = adadata[myIndices,]
  adamodel =
boosting(myYfactor~.,data=adaTrain,mfinal=100,coeflearn
="Freund",
             control=rpart.control(maxdepth=2))
  adaPred = predict(adamodel, newdata=adaTest)
  adaProb = adaPred$prob[,2]
  miuMatrix[((kk-1)*foldLength+1):(kk*foldLength),4] =
adaProb

  # Logistic Regression
  myYtrain = myYfactor[-myIndices]
  LRtrain = data.frame(myXtrain, myYtrain)
  LRmodel = glm( myYtrain~ AOD750 + TISA750 + IT750
+IT2000 + ITCM + IAREA + ICURV + ACW_mm + ACA + ACV
+ LENSVAULT, family = binomial(logit),data = LRtrain)
  LRmodelstep = step(LRmodel,k=5,direction = "both")
  LRProb = predict(LRmodelstep, newdata =
data.frame(myXtest), type = "response")
  miuMatrix[((kk-1)*foldLength+1):(kk*foldLength),5] =
LRProb

}

D = 2*t(miuMatrix)%*%miuMatrix
d = 2*t(miuMatrix)%*%yMatrix
A = t(rbind(matrix(1,1,5),matrix(-1,1,5),diag(c(1,1,1,1,1))))
b = c(1, -1, 0, 0, 0, 0, 0)
library(quadprog)
solution = solve.QP(D,d,A,b,factorized = FALSE)

weight1 = solution$solution
weight2 = solution$unconstrained.solution
```

Appendix V Validation

```
# Read in training data
myData=read.csv("AngleClosure.csv",header=TRUE,na.stri
ngs=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","W
T","ASPH", "ACYL", "SE", "AXL", "CACD", "AGE", "CCT.OD",
"PCCURV_mm"))])
# Remove rows with any missingness
myLogical=apply(cbind(myResponse,myPredictors),1,funct
ion(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

################################## get the 5 models
##############################



myXtrain = myX
myYtrain = myY

# SVM
svm.model = svm(myXtrain, myYtrain, type = "C", gamma =
0.01, cost = 0.1,probability = TRUE)
# Neural Network
myYtrainFactor = as.factor(myYtrain)
fit=nnet(myYtrainFactor~.,data=myXtrain,weights=rep(1,le
ngth(myYtrainFactor)),size=2,
decay=0.0005,MaxNWts=10000,maxit=10000,trace=FALSE)
# random forest
RFmodel = randomForest(x = myXtrain, y = myYtrainFactor,
ntree = 100)

# boosted model
myYfactor = as.factor(myYtrain)
adaTrain = data.frame(myYfactor,myXtrain)
adamodel =
boosting(myYfactor~.,data=adaTrain,mfinal=100,coeflearn
="Freund",
              control=rpart.control(maxdepth=2))
# Logistic Regression
myYtrain = myYfactor
LRtrain = data.frame(myXtrain, myYtrain)
LRmodel = glm( myYtrain~ AOD750 + TISA750 + IT750
+IT2000 + ITCM + IAREA + ICURV + ACW_mm + ACA + ACV
+ LENSVAULT, family = binomial(logit),data = LRtrain)
LRmodelstep = step(LRmodel,k=5,direction = "both")


################################## Read in the
validation data ##################################


myData1=read.csv("AngleClosure_ValidationCases.csv",he
ader=TRUE,na.strings=c("NA","."))
myData2=read.csv("AngleClosure_ValidationControls.csv",
header=TRUE,na.strings=c("NA","."))
# Set-up response and predictor variables
myResponse1 = matrix(1,dim(myData1)[1],1)
myResponse2= matrix(0,dim(myData2)[1],1)


# create the the predictors Y=1
right1 =
c("rAOD750","rTISA750","rIT750","IT2000","rITCM","rIARE
A","rICURV", "ACWmm", "ACA", "ACV", "LENSVAULT")
myPredictors1_right=data.matrix(myData1[,(attributes(my
Data1)$names %in% right1)])
colnames(myPredictors1_right)=
c("AOD750","TISA750","IT750","IT2000","ITCM","IAREA","I
CURV", "ACW_mm", "ACA", "ACV", "LENSVAULT")

left1 =
c("lAOD750","lTISA750","lIT750","lIT2000","lITCM","lIARE
A","lICURV", "ACWmm", "ACA", "ACV", "LENSVAULT")
```

```
myPredictors1_left=data.matrix(myData1[,(attributes(myData1)$names %in% left1)])
colnames(myPredictors1_left)=
c("AOD750","TISA750","IT750","IT2000","ITCM","IAREA","ICURV", "ACW_mm", "ACA", "ACV", "LENSVAULT")

for (ii in 1:dim(myPredictors1_right)[2]){
  for (jj in 1:dim(myPredictors1_right)[1]){
    if (is.na(myPredictors1_right[jj,ii])){
      myPredictors1_right[jj,ii] = myPredictors1_left[jj,ii]
    }
  }
}
# Remove rows with any missingness
myLogical=apply(myPredictors1_right,1,function(xx){
  return(!any(is.na(xx)))
})
myPredictors1 = myPredictors1_right[myLogical,]

# create the the predictors Y=0
right2 =
c("rAOD750","rTISA750","rIT750","rIT2000","rITCM","rIAREA","rICURV", "ACW.mm.", "ACA", "ACV", "LENSVAULT")
myPredictors2_right=data.matrix(myData2[,(attributes(myData2)$names %in% right2)])
colnames(myPredictors2_right)=
c("AOD750","TISA750","IT750","IT2000","ITCM","IAREA","ICURV", "ACW_mm", "ACA", "ACV", "LENSVAULT")

left2 =
c("lAOD750","lTISA750","lIT750","lIT2000","lITCM","lIAREA","lICURV.", "ACW.mm.", "ACA", "ACV", "LENSVAULT")
myPredictors2_left=data.matrix(myData2[,(attributes(myData2)$names %in% left2)])
colnames(myPredictors2_left)=
c("AOD750","TISA750","IT750","IT2000","ITCM","IAREA","ICURV", "ACW_mm", "ACA", "ACV", "LENSVAULT")

for (ii in 1:dim(myPredictors2_right)[2]){
  for (jj in 1:dim(myPredictors2_right)[1]){
    if (is.na(myPredictors2_right[jj,ii])){
      myPredictors2_right[jj,ii] = myPredictors2_left[jj,ii]
    }
  }
}
# Remove rows with any missingness
myLogical=apply(myPredictors2_right,1,function(xx){
  return(!any(is.na(xx)))
})

myPredictors2 = myPredictors2_right[myLogical,]
```

```
myPredictors = rbind(myPredictors1,myPredictors2)

# make all data mean zero and variance one uisng the training mean and training variance
for (ii in 1:dim(myPredictors)[2]){
  for (jj in 1:dim(myPredictors)[1]){
    myPredictors[jj,ii] = (myPredictors[jj,ii]-myPredictors_mean[ii])/myPredictors_sd[ii]
  }
}
myXtest = myPredictors
myYtest =
rbind(matrix(1,dim(myPredictors1)[1],1),matrix(0,dim(myPredictors2)[1],1))

############################# get the results from 7 models #############################

miuMatrix = matrix(NA,length(myYtest),7)
# SVM
svm.prob = predict(svm.model, myXtest, probability = TRUE)
miuMatrix[,1] =
attr(svm.prob,"probabilities")[,colnames(attr(svm.prob,"probabilities"))==1]

# Neural Network
NNProb=predict(fit,newdata=myXtest, type = "raw")
miuMatrix[,2] = NNProb

# random forest
RFProb = predict(RFmodel, myXtest, type = "prob")[,2]
miuMatrix[,3] = RFProb

# boosted model
myYfactor = as.factor(myYtest)
adaTest = data.frame(myYfactor,myXtest)
adaPred = predict(adamodel, newdata=adaTest)
adaProb = adaPred$prob[,2]
miuMatrix[,4] = adaProb

# Logistic Regression
LRProb = predict(LRmodelstep, newdata = data.frame(myXtest), type = "response")
miuMatrix[,5] = LRProb

# Stacking 1
weight1 = c(0,0,0.11378,0.00252,0.88369)
miuMatrix[,6] = miuMatrix[,1:5]%*%weight1
```

```
# Stacking 2
weight2 = c(-0.03079,-0.21070,0.14418,-0.02147,1.07496)
miuMatrix[,7] = miuMatrix[,1:5]%*%weight2

# plot the ROC and compute the AUC
library(pROC)
myRocList = list()
myAUC = matrix(NA,7,1)

dev.new(width=8,height=8)
par(mar=c(0.075,0.075,1.5,0.075),mfrow=c(3,3))
model = c("SVM", "NNet", "RandF", "Boost",
"LogistR","StackCon","StackUncon")
for (ii in 1:7){
  myRocList[[ii]] = roc(response = myYtest, predictor =
miuMatrix[,ii], auc.polygon=TRUE, grid=TRUE, plot=TRUE)
  myAUC[ii,1] = myRocList[[ii]]$auc
  title(main = paste(model[ii],": AUC =
",as.character(round(myAUC[ii,1], digits=4))))
}
```