1.Data Manipulation

In the beginning of this problem, we read in data *AngleClosure.csv*, delete the columns corresponding to factor variables EYE, GENDER, ETHNIC, HGT, WT, ASPH,
ACYL, SE, AXL, CACD, AGE, CCT.OD, and PCCURV mm, and then delete rows of the dataset which have any missing values

	Observations	Predictors
Read The Data	1468	24
Omit specific attribute	1468	11
Delete missing value row	1468	11

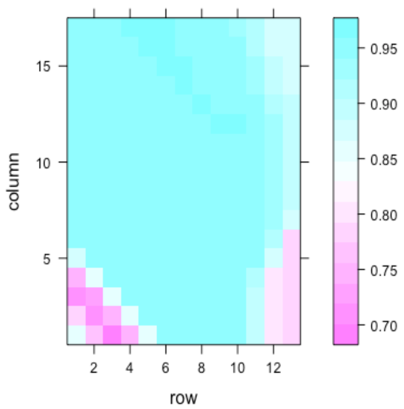
2.Develop Prediction Models

Five prediction models are chosen, and they are

- Support vector machine (e1071)
- Neural network (nnet)
- Random forest (randomForest)
- Boosted model (ada)
- Logistic regression

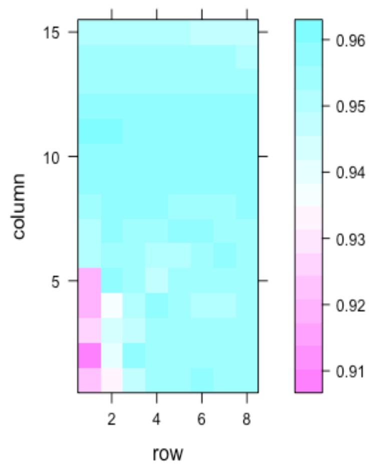
3.Model Parameter Selection

Support vector machine:
There are two parameters that need to be tuned in svm: *gamma* and *cost*. Therefore, we use a grid search to find the best combination of the parameters.



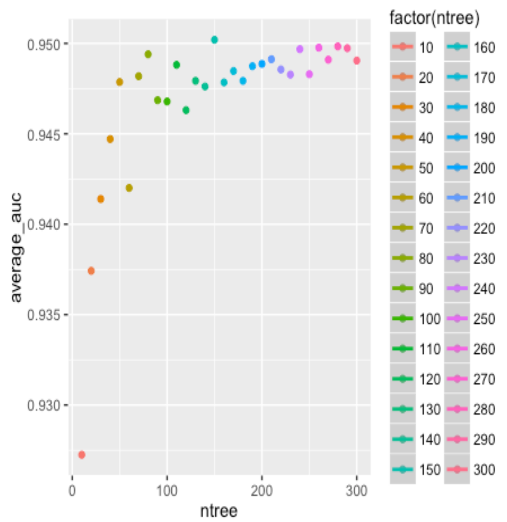
When Gamma=0.001, Cost=3.162, the max average auc is 0.95903

Neural network:
There are also two parameters that need to be tuned for neural network: the size and the decay. we use a grid search to find the best combination of the parameters.



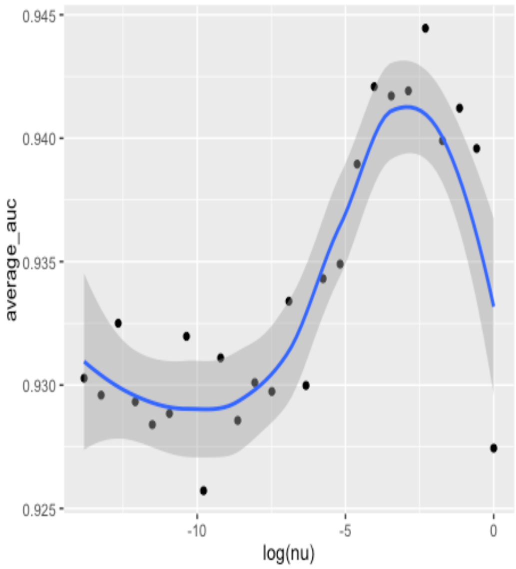
When Size = 3, Decay = 0.1, the max average auc is 0.95957

Random forest:
In random forest, the only parameter that could be tuned is the number of trees.



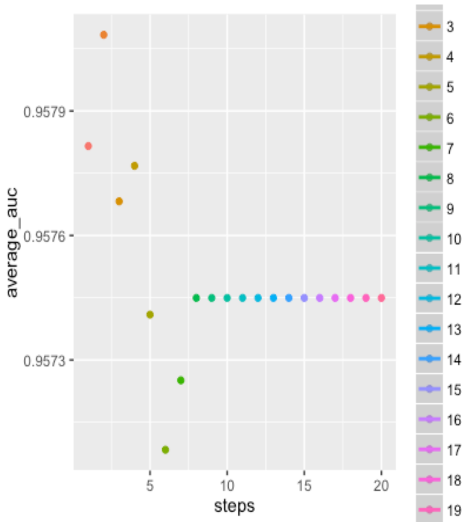
When number of tree = 150, the max average auc is 0.950198

Boosted model:
In Adaptive Boosting method, the parameter that need to be tuned there is nu.



When nu = 0.1, the max average auc is 0.94445

Logistic regression:
In Logistic Regression, the tuning parameter is the steps numbers allowed in the Logistic Regression Model using two-directional stepwise AIC regression.



When steps = 2, the max average auc is 0.95808

4.Stacking

We use 10-fold cross validation with the selected models on the original training data. We predicted each model for the testing fold data and then store with corresponding response. Using store data for constrained and unconstrained optimizations.
The stacking models with and without constraints are shown in table below.

Model	Constraint w	Unconstraint w
Support vector machine	0.1457315	0.15072539
Neural network	0.2662107	0.27214284
Random forest	4.312247e-19	-0.06550852
Boosted model	0.1891759	0.21353243
Logistic regression	0.3988819	0.41952495

5.Validation

We manipulated the Cases and Controls data to ensure we only have the necessary columns and that we preferentially choose right eye data over left, and try to test the performance of the trained 7 models.

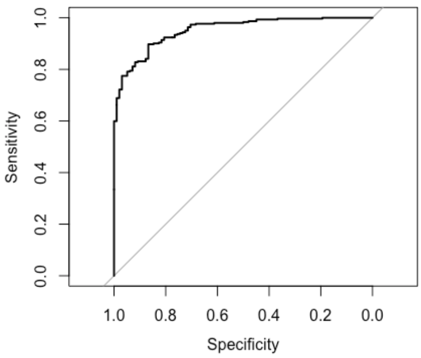
Model	AUC
Support vector machine	0.9519
Neural network	**0.9649**
Random forest	0.952
Boosted model	0.961
Logistic regression	0.9536
Stacked Constrained	0.9597
Stacked Unconstrained	0.959

As we can see, the best model generated on the validation data set was the **Neural Network with an AUC of 0.9649.**

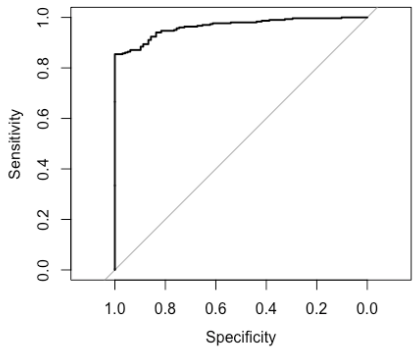
6.Visulazation

7 models' ROC Curve.

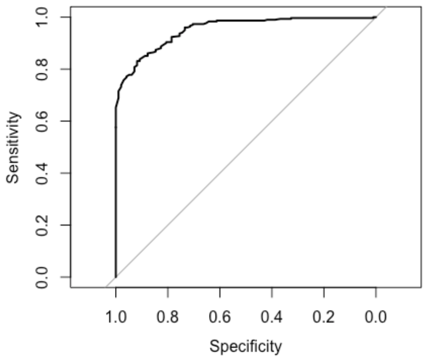
SVM: AUC = 0.9519



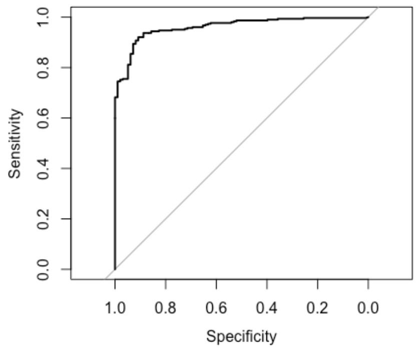
Neural Network: AUC = 0. 9649



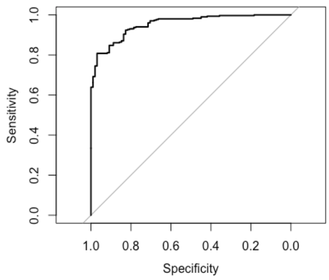
Random Forest: AUC = 0. 952



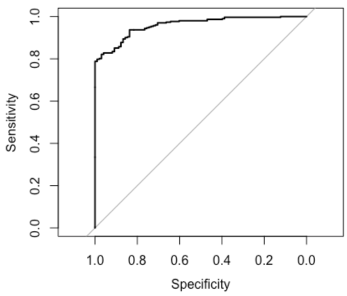
Ada Boost: AUC = 0. 961



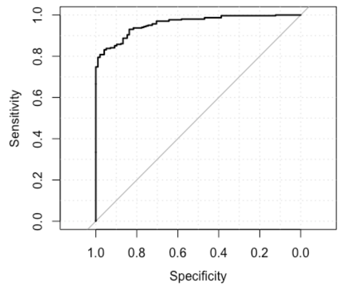
Logistic Regression: AUC = 0. 9536



Stacked Constraint: AUC = 0. 9597



Stacked Unonstraint: AUC = 0. 959



R CODE

```r
############ Data Manipulation #############
rm(list=ls())
myData=read.csv("AngleClosure.csv",header=TRUE,na.strings=c("NA","."))

myData1 = myData[,-c(1,15,16)]
myData1[,-21] = data.matrix(myData1[,-21])
myData1[,21] = factor(myData1[,21])
myData2 = myData1[,1:11]
myData3 = na.omit(myData2)
y = myData1[,21]
newData = cbind(y, myData3)

############ Develop Prediction Models #############
############ #1 SVM Model #############
library(e1071)
library(pROC)
library(lattice)
nIter = 25
gamma = 10^seq(-6,0,0.5)
cost = 10^seq(-6,2,0.5)
svm.auc = array(0, dim=c(nIter,length(gamma),length(cost)))

for(iter in 1:nIter){
  index = sample(nrow(newData))[1:round(nrow(newData)/10)]
  trainData = newData[-index,]
  testData = newData[index,]
  for(g in gamma){
    for(c in cost){
      model = svm(y~., data=trainData, gamma=g, cost=c, probability=TRUE)
      myPred = predict(model, testData, probability=TRUE)
      myProb = attr(myPred, "probabilities")[,1]
      myroc = roc(testData$y, myProb)
      svm.auc[iter,which(gamma==g),which(cost==c)] = myroc$auc
  }}
  print(iter)
}
svmtest = apply(svm.auc,c(2,3),mean)
levelplot(svmtest)
# AUC max 0.95903
# Cost = 3.162, Gamma = 0.001
```

```r
############ #2 Neural Network Model ##############
library(nnet)
library(pROC)
library(lattice)
nIter = 25
size = seq(3,24,3)
decay = 10^seq(-6,1,0.5)
nnet.auc = array(0, dim=c(nIter,length(size),length(decay)))

for(iter in 1:nIter){
  index = sample(nrow(newData))[1:round(nrow(newData)/10)]
  trainData = newData[-index,]
  testData = newData[index,]
  for(s in size){
    for(d in decay){
      model = nnet(y~., data=trainData, size=s, decay=d)
      myPred = predict(model, testData, probability = TRUE)
      myroc = roc(testData$y, myPred)
      nnet.auc[iter,which(size==s),which(decay==d)] = myroc$auc
  }}
  print(iter)
}
nnettest = apply(nnet.auc,c(2,3),mean)
levelplot(nnettest)
# AUC max 0.95957
# Size = 3, Decay = 0.1


############ #3 Random Forest Model ##############
library(randomForest)
library(pROC)
library(ggplot2)
nIter = 25
ntree = seq(10,300,10)
rf.auc = matrix(NA, nIter, length(ntree))

for(iter in 1:nIter){
  index = sample(nrow(newData))[1:round(nrow(newData)/10)]
  trainData = newData[-index,]
  testData = newData[index,]
  for(n in ntree){
    model  = randomForest(y~., data=trainData, ntree=n)
    myPred = predict(model, testData, type="prob")[,2]
    myroc = roc(testData$y, myPred)
    rf.auc[iter,which(ntree==n)] = as.numeric(myroc$auc)
  }
  print(iter)
}
rfplot = data.frame(ntree, colMeans(rf.auc))
names(rfplot) = c("ntree", "average_auc")
ggplot(rfplot, aes(x=ntree, y=average_auc, colour=factor(ntree))) +
geom_point() + geom_smooth()
```

```r
############ #4 Boosted Model ##############
library(ada)
library(pROC)
library(ggplot2)
nIter = 25
nu = 10^seq(-6,0,0.25)
ada.auc = matrix(NA, nIter, length(nu))

for(iter in 1:nIter){
  index = sample(nrow(newData))[1:round(nrow(newData)/10)]
  trainData = newData[-index,]
  testData = newData[index,]
  for(n in nu){
    model = ada(y~.,  data=trainData, nu=n)
    myPred = predict(model, testData, type="prob")[,2]
    myroc = roc(testData$y, myPred)
    ada.auc[iter,which(nu==n)] = as.numeric(myroc$auc)
  }
  print(iter)
}
adaplot = data.frame(nu, colMeans(ada.auc))
names(adaplot) = c("nu", "average_auc")
ggplot(adaplot, aes(x=log(nu), y=average_auc, colour=factor(nu))) +
geom_point() + geom_smooth()
```

```r
############ #5 Logistic Regression Model ##############
library(pROC)
library(ggplot2)
nIter = 25
mysteps = seq(1,20,1)
glm.auc = matrix(NA, nIter, length(mysteps))

for(iter in 1:nIter){
  index = sample(nrow(newData))[1:round(nrow(newData)/10)]
  trainData = newData[-index,]
  testData = newData[index,]
  for(s in mysteps){
  test = glm(y~.,data=trainData,family=binomial(link=probit))
  step = step(test, direction = "both", steps = s)
  newtrainData = step $ model
  model = glm(y~.,data=newtrainData,family=binomial(link=probit))
    myPred = predict(model, testData, family=binomial(link=probit), type = "response")
    myroc = roc(testData$y, myPred)
    glm.auc[iter,which(mysteps==s)] = as.numeric(myroc$auc)
  }
  print(iter)
}
lgplot = data.frame(mysteps, colMeans(glm.auc))
names(lgplot) = c("steps", "average_auc")
ggplot(lgplot, aes(x=steps, y=average_auc, colour=factor(steps))) +
geom_point() + geom_smooth()
```

Stacking

```
############ Stacking  ##############
library(e1071)
library(nnet)
library(randomForest)
library(ada)
library(pROC)
library(quadprog)

nIter=25
X = matrix(0,,5)
Y = matrix(0,,1)

for(iter in 1:nIter){
    index = sample(nrow(newData))[1:round(nrow(newData)/10)]
    trainData = newData[-index,]
    testData = newData[index,]
    fusionX = matrix(0,dim(testData)[1],)

    #SVM Model
    svm_model = svm(y~., data=trainData, gamma=0.001, cost=3.162, probability=TRUE)
    svm_myPred = predict(svm_model, testData, probability=TRUE)
    svm_myProb = attr(svm_myPred, "probabilities")[,1]
    fusionX = cbind(fusionX,svm_myProb)

    #NNET Model
    nnet_model = nnet(y~., data=trainData, size=3, decay=0.1)
    nnet_myPred = as.numeric(predict(nnet_model, testData, probability = TRUE))
    fusionX = cbind(fusionX,nnet_myPred)

    #RF Model
    rf_model  = randomForest(y~., data=trainData, ntree=150)
    rf_myPred = predict(rf_model, testData, type="prob")[,2]
    fusionX = cbind(fusionX,rf_myPred)

    #ADA Model
    ada_model = ada(y~.,  data=trainData, nu=0.1)
    ada_myPred = predict(ada_model, testData, type="prob")[,2]
    fusionX = cbind(fusionX,ada_myPred)

    #LG Model
    lm = glm(y~.,data=trainData,family=binomial(link=probit))
    step = step(lm, direction = "both", steps = 2)
    newtrainData = step $ model
    lg_model = glm(y~.,data=newtrainData,family=binomial(link=probit))
    lg_myPred = predict(lg_model, testData, family=binomial(link=probit), type = "response")
    fusionX = cbind(fusionX,lg_myPred)
    temp = as.numeric(testData$y=='YES')
    Y = rbind(Y,as.matrix(temp))
    X = rbind(X, fusionX[,2:6])
}
Miu = X
Response = Y
Miu = Miu[2:dim(Miu)[1],]
Response = Response[2:dim(Response)[1],]

D = t(Miu) %*% Miu
d = t(Response) %*% Miu
A = cbind(rep(1,5), diag(5))
b = c(1, 0, 0, 0, 0, 0)

solution = solve.QP(D, d, A, b,meq =1)
weightsConstrained = solution$solution
# weightsConstrained
# 1.457315e-01 2.662107e-01 4.312247e-19 1.891759e-01 3.988819e-01
weightsUnConstrained = solution$unconstrained.solution
#weightsUnConstrained
#  0.15072539  0.27214284 -0.06550852  0.21353243  0.41952495
```

Validation

```
attnames = names(newData)[-1]
caseData = read.csv("AngleClosure_ValidationCases.csv")
controlData = read.csv("AngleClosure_ValidationControls.csv")

myCase.right = caseData[,c(19,21,23,24,25,26,27,30,31,32,36)]
myCase.left = caseData[,c(7,9,11,12,13,14,15,30,31,32,36)]

myControl.right = controlData[,c(18,20,22,23,24,25,26,29,30,31,35)]
myControl.left = controlData[,c(6,8,10,11,12,13,14,29,30,31,35)]

# for cases data
# delete rows which have any missing values.
# preferentially take right eye data
logic_r = which(complete.cases(myCase.right))
logic_l = which(complete.cases(myCase.left))
logic_l = logic_l[which(!logic_l %in% logic_r)]

temp_r = myCase.right[logic_r,]
temp_l = myCase.left[logic_l,]
names(temp_r) = names(temp_l) = attnames
myCase = rbind(temp_r, temp_l)
y = rep("YES", nrow(myCase))
myCase = cbind(y, myCase)

# for controls data
logic_r = which(complete.cases(myControl.right))
# No missing value rows in control for right eye
temp_r = myControl.right[logic_r,]
names(temp_r) = attnames
myControl = temp_r
y = rep("NO", nrow(myControl))
myControl = cbind(y, myControl)

valid_data = rbind(myCase, myControl)
row.names(valid_data) = NULL
############ Validation ##############
library(e1071)
library(nnet)
library(randomForest)
library(ada)
library(pROC)
```

```r
svm_model = svm(y~., data=newData, gamma=0.001, cost=3.162, probability=TRUE)
svm_myPred = predict(svm_model, valid_data, probability=TRUE)
svm_myProb = attr(svm_myPred, "probabilities")[,1]
svm_roc = roc(as.numeric(valid_data[,1]),as.numeric(svm_myProb), plot=T)
svm_auc = svm_roc$auc
# Area under the curve: 0.9519

#NNET Model
nnet_model = nnet(y~., data=newData, size=3, decay=0.1)
nnet_myPred = as.numeric(predict(nnet_model, valid_data, probability = TRUE))
nnet_roc = roc(valid_data$y, nnet_myPred, plot=T)
nnet_auc = nnet_roc$auc
#Area under the curve: 0.9649

#RF Model
rf_model  = randomForest(y~., data=newData, ntree=150)
rf_myPred = predict(rf_model, valid_data, type="prob")[,2]
rf_roc = roc(valid_data$y, rf_myPred, plot=T)
rf_auc = rf_roc$auc
#Area under the curve: 0.952

#ADA Model
ada_model = ada(y~.,  data=newData, nu=0.1)
ada_myPred = predict(ada_model, valid_data, type="prob")[,2]
ada_roc = roc(valid_data$y, ada_myPred, plot=T)
ada_auc = ada_roc$auc
#Area under the curve: 0.961

#LG Model
lm = glm(y~.,data=newData,family=binomial(link=probit))
step = step(lm, direction = "both", steps = 2)
newtrainData = step $ model
lg_model = glm(y~.,data=newtrainData,family=binomial(link=probit))
lg_myPred = predict(lg_model, valid_data, family=binomial(link=probit), type = "response")
lg_roc = roc(valid_data$y, lg_myPred, plot=T)
lg_auc = lg_roc$auc
#Area under the curve: 0.9536

P = cbind (svm_myProb, nnet_myPred, rf_myPred, ada_myPred, lg_myPred)
# stacked constrained
stackedC_myPred = P %*% weightsConstrained
Con_roc = roc(as.numeric(valid_data[,1]),as.numeric(stackedC_myPred), plot=T)
Con_auc = Con_roc$auc
stackedU_myPred = P %*% weightsUnConstrained
Uncon_roc = roc(as.numeric(valid_data[,1]),as.numeric(stackedU_myPred), grid=TRUE, plot=T)
Uncon_auc = Uncon_roc$auc
```