1 Data Manipulation

(a) Read in data (AngleClosure.csv),

(b) delete the columns corresponding to factor variablesEYE, GENDER, and ETHNIC, and (c) delete rows of the dataset which have any missing values.

Code for data manipulation

|  |
| --- |
| mydata = read.csv("AngleClosure.csv")  mydata = as.matrix(mydata)  ##delete EYE, GENDER, and ETHNIC  ##Omit the variables HGT, WT, ASPH, ACYL, SE, AXL, CACD, AGE, CCT.OD, and PCCURV\_mm  rml = c("EYE", "GENDER", "ETHNIC")  rml = c(rml,"HGT", "WT", "ASPH", "ACYL", "SE", "AXL", "CACD", "AGE", "CCT.OD", "PCCURV\_mm")  data = mydata[,!colnames(mydata) %in% rml]  ## delete rows of the dataset which have any missing values.  remove\_index\_set = apply(data,1,function(xx){  return(sum(is.na(xx))>0)  })  data = data[!remove\_index\_set,]  ##move response to first column  response = c("ANGLE.CLOSURE")  myy = data[,response,drop=FALSE]  myx = data[,!colnames(data) %in% response]  data = cbind(myy,myx)  write.csv(file="cleandata.csv", x=data) |

Number of predictors:11

Number of completed train samples: 1468

2 Develop Prediction Models

Please see code in part 3

3 Model and Tuning Parameter Selection

Tunning parameters for model

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Model | SVM | RandomForest | NeuralNetwork | Boosting | K Nearest Neighbour |
| Tunning Parametere | cost=10\*\*(-9:0:0.2) gamma=10\*\*(-9:0:0.2) | ntree=seq(30,800,30) | size=seq(6,40,3) decay=seq(0.5,10,0.5) | nu=10\*\*seq(-4,0.5,0.2) | k=seq(1,100,2) distance=seq(1,10,0.5) |
| Selected Parameter | cost=1, gamma=0.003981072 | ntree=750 | size=6 decay=0.5 | nu=0.1 | k=99 distance=1 |

support vector machine

|  |
| --- |
| rm(list=ls())  library(e1071)  library(pROC)  library(lattice)  data = read.csv("cleandata.csv")[,-1]  myy = data[,1,drop = F]  myx = data.matrix(data[,-1])  n =dim(myx)[1]  p =dim(myx)[2]  ##start CV  Niter = 100  kfold = 10  ##1.specify your parameter here  ##cost  para = sapply(seq(-9,0,0.2),function(xx){  return(10\*\*(xx))  })  ##gama  para2 = sapply(seq(-9,0,0.2),function(xx){  return(10\*\*(xx))  })  auc.res = array(NA,c(Niter,length(para),length(para2)))  for(j in 1:Niter){  testID = sample(1:n,round(n/kfold))  for(i in 1:length(para)){  for(k in 1:length(para2)){  print(j)  ##modelling  model = svm(myx[-testID,],myy[-testID,],cost = para[i],gamma = para2[k], probability = T)  yhat = predict(model,myx[testID,],probability = T)  yhat=attr(yhat,"probabilities")  roc = roc(myy[testID,], yhat[,1])  auc.res[j,i,k] = auc(roc)    }  }  }  dput(para,"svm.cost.para.r")  dput(para2,"svm.gamma.para.r")  dput(auc.res,"auc.res.svm.r")  ##read data  svm.cost.para.r = dget("svm.cost.para.r")  svm.gamma.para.r = dget("svm.gamma.para.r")  auc.res = dget("auc.res.svm.r")  mytest = dget("mytest.r")  data = read.csv("cleandata.csv")[,-1]  myy = data[,1,drop = F]  myx = data.matrix(data[,-1])  auc.list = apply(auc.res,c(2,3),mean)  levelplot(auc.list)  bestpara\_pos = which(auc.list == max(auc.list), arr.ind = TRUE)  bestpara1 = svm.cost.para.r[bestpara\_pos[1]]##1  bestpara2 = svm.gamma.para.r[bestpara\_pos[2]]##0.003981072  model = svm(myx,myy[,],cost = bestpara1,gamma = bestpara2, probability = T)  yhat = predict(model,mytest[,-1],probability = T)  yhat=attr(yhat,"probabilities")  roc = roc(mytest[,1], yhat[,1])  auc.res[j,i,k] = auc(roc)  auc(roc)##0.9482 |

RandomForest

|  |
| --- |
| library(randomForest)  library(pROC)  data = read.csv("cleandata.csv")[,-1]  myy = data[,1,drop = F]  myx = data.matrix(data[,-1])  n = dim(myx)[1]  p = dim(myx)[2]  set.seed(71)  #### start CV  Niter = 100  kfold = 10  ##1. change para  ##para for randomforest:ntree  para = seq(30,800,30)  auc.res = matrix(NA,Niter,length(para))  for(j in 1:Niter){  testID = sample(1:n,round(n/kfold))  for(i in 1:length(para)){  print(j)  print(para[i])  ##2. change modelling  model = randomForest(myx[-testID,],myy[-testID,],ntree = para[i], probability = T)  yhat = predict(model,myx[testID,],type = "prob")  roc = roc(myy[testID,], yhat[,1])  auc.res[j,i] = auc(roc)  }  }  auc.list = apply(auc.res,2,mean)  bestpara = para[which.max(auc.list)]  plot(para,auc.list)  dput(para,"2-rf.para.r")  dput(auc.res,"2-auc.res.rf.r")  dput(rf\_best\_model,"2-rf\_best\_model.r")  ## read para ###  para = dget("2-rf.para.r")  auc.res = dget("2-auc.res.rf.r")  mytest = dget("mytest.r")  data = read.csv("cleandata.csv")[,-1]  myy = data[,1,drop = F]  myx = data.matrix(data[,-1])  auc.list = apply(auc.res,2,mean)  bestpara = para[which.max(auc.list)]  plot(para,auc.list)  model = randomForest(myx,myy[,1],ntree = bestpara, probability = T)  yhat = predict(model,mytest,type = "prob")  roc = roc(mytest[,1], yhat[,1])  auc(roc) |

Neural Network

|  |
| --- |
| rm(list=ls())  library(nnet)  library(pROC)  library(lattice)  data = read.csv("cleandata.csv")[,-1]  myy = data[,1,drop=F]  myx = data.matrix(data[,-1])  n =dim(myx)[1]  p =dim(myx)[2]  ##learn model  model = nnet(ANGLE.CLOSURE~.,data = data, size = 6,decay = 0.5 )  pred = predict(model,data,type = "raw")  roc = roc(data[,1],pred)  ##start CV  Niter = 100  kfold = 10  ##1.specify your parameter here  ##size  para = sapply(seq(6,40,3),function(xx){  return((xx))  })  ##decay  para2 = sapply(seq(0.5,10,0.5),function(xx){  return((xx))  })  auc.res = array(NA,c(Niter,length(para),length(para2)))  for(j in 1:Niter){  testID = sample(1:n,round(n/kfold))  for(i in 1:length(para)){  for(k in 1:length(para2)){  print(j)  print(para[i])  print(para2[k])  ##modelling  model = nnet(ANGLE.CLOSURE~.,data = data[-testID,], size = para[i],decay = para2[k] )  yhat = predict(model,data[testID,],type = "raw")  roc = roc(myy[testID,], yhat)  auc.res[j,i,k] = auc(roc)  }  }  }  ##bestpara = para[which.max(auc.list)]  dput(para,"nn.size.para.r")  dput(para2,"nn.decay.para.r")  dput(auc.res,"auc.res.nn.r")  ##read data  nn.size.para.r = dget("nn.size.para.r")  nn.decay.para.r = dget("nn.decay.para.r")  auc.res = dget("auc.res.nn.r")  mytest = dget("mytest.r")  data = read.csv("cleandata.csv")[,-1]  myy = data[,1,drop=F]  myx = data.matrix(data[,-1])  auc.list = apply(auc.res,c(2,3),mean)  levelplot(auc.list)  bestpara\_pos = which(auc.list == max(auc.list), arr.ind = TRUE)  bestpara1 = nn.size.para.r[bestpara\_pos[1]]##6  bestpara2 = nn.decay.para.r[bestpara\_pos[2]]##0.5  model = nnet(ANGLE.CLOSURE~.,data = data, size = bestpara1,decay = bestpara2 )  yhat = predict(model,mytest,type = "raw")  roc = roc(mytest[,1], yhat)  auc(roc)##0.9711 |

Boosting

|  |
| --- |
| library(ada)  library(pROC)  data = read.csv("cleandata.csv")[,-1]  myy = data[,1,drop = F]  myx = data.matrix(data[,-1])  n = dim(myx)[1]  p = dim(myx)[2]  ##learn model  model = ada(myx[,],myy[,1] )  pred = predict(model,as.data.frame(myx))  ##check prediiction  #### start CV  Niter = 100  kfold = 10  ##1. change para  ##para for randomforest:nu  para = 10\*\*seq(-4,0.5,0.2)  auc.res = matrix(NA,Niter,length(para))  for(j in 1:Niter){  testID = sample(1:n,round(n/kfold))  for(i in 1:length(para)){  print(j)  print(para[i])  ##2. change modelling  model = ada(myx[-testID,],myy[-testID,] ,nu=para[i])  yhat = predict(model,as.data.frame(myx[testID,]),type = "prob")  roc = roc(myy[testID,], yhat[,1])  auc.res[j,i] = auc(roc)  print(auc.res[j,i])  }  }  #auc.list = apply(auc.res,2,mean)  #bestpara = para[which.max(auc.list)]  #plot(para,auc.list)  dput(para,"2-ada.para.r")  dput(auc.res,"2-au.res.ada.r")  ## read data ##  para = dget("2-ada.para.r")  auc.res = dget("2-au.res.ada.r")  mytest = dget("mytest.r")  data = read.csv("cleandata.csv")[,-1]  myy = data[,1,drop = F]  myx = data.matrix(data[,-1])  auc.list = apply(auc.res,2,mean)  bestpara = para[which.max(auc.list)]##0.1  plot(para,auc.list)  model = ada(myx[,],myy[,1] ,nu=bestpara)  yhat = predict(model,as.data.frame(mytest[,]),type = "prob")  roc = roc(mytest[,1], yhat[,1])  auc(roc)##0.9626 |

K Nearest Neighbour

|  |
| --- |
| rm(list=ls())  library(kknn)  library(pROC)  library(lattice)  data = read.csv("cleandata.csv")[,-1]  myy = data[,1,drop=F]  myx = data.matrix(data[,-1])  n =dim(myx)[1]  p =dim(myx)[2]  ##learn model  model = kknn(ANGLE.CLOSURE~.,train = data,test = data, k = 10,distance = 3 )  pred = predict(model,data,type = "raw")  roc = roc(data[,1],as.numeric(pred))  ##start CV  Niter = 100  kfold = 10  ##1.specify your parameter here  ##k  para = sapply(seq(1,100,2),function(xx){  return((xx))  })  ##distance  para2 = sapply(seq(1,10,0.5),function(xx){  return((xx))  })  auc.res = array(NA,c(Niter,length(para),length(para2)))  for(j in 1:Niter){  testID = sample(1:n,round(n/kfold))  for(i in 1:length(para)){  for(k in 1:length(para2)){  print(j)  print(para[i])  print(para2[k])  ##modelling  model = kknn(ANGLE.CLOSURE~.,train = data[-testID,],test = data[testID,], k = para[i],distance = para2[k] )  yhat = model$prob  roc = roc(myy[testID,], yhat[,1])  auc.res[j,i,k] = auc(roc)  print(auc.res[j,i,k])  }  }  }  ##bestpara = para[which.max(auc.list)]  dput(para,"knn.k.para.r")  dput(para2,"knn.distance.para.r")  dput(auc.res,"auc.res.knn.r")  ##read data  nn.k.para.r = dget("knn.k.para.r")  nn.distance.para.r = dget("knn.distance.para.r")  auc.res = dget("auc.res.knn.r")  mytest = dget("mytest.r")  data = read.csv("cleandata.csv")[,-1]  myy = data[,1,drop=F]  myx = data.matrix(data[,-1])  auc.list = apply(auc.res,c(2,3),mean)  levelplot(auc.list)  bestpara\_pos = which(auc.list == max(auc.list), arr.ind = TRUE)  bestpara1 = nn.k.para.r[bestpara\_pos[1]]##99  bestpara2 = nn.distance.para.r[bestpara\_pos[2]]##1  model = kknn(ANGLE.CLOSURE~.,train = data[,],test = mytest[,],k=bestpara1,distance = bestpara2 )  yhat = model$prob  roc = roc(mytest[,1], yhat[,1])  auc(roc)##0.959 |

4 Stacking

|  |
| --- |
| rm(list=ls())  library(e1071)  library(randomForest)  library(nnet)  library(ada)  library(kknn)  library(pROC)  library(lattice)  library(quadprog)  data = read.csv("cleandata.csv")[,-1]  myy = data[,1,drop = F]  myx = data.matrix(data[,-1])  n =dim(myx)[1]  p =dim(myx)[2]  ##start CV  Niter = 100  kfold = 10  ## parameter ##  cost = 1  gamma = 0.003981072  ntree=750  size=6  decay=0.5  nu=0.1    parak=99  distance=1  stack.res = array(NA,c(Niter, round(n/kfold),5))  stack.y.res = array(NA,c(Niter, round(n/kfold),1))  for(j in 1:Niter){  testID = sample(1:n,round(n/kfold))    print(j)  ##svm  model = svm(myx[-testID,],myy[-testID,],cost = cost,gamma = gamma, probability = T)  yhat = predict(model,myx[testID,],probability = T)  stack.res[j,,1] = attr(yhat,"probabilities")[,1]    ##randomforest  model = randomForest(myx[-testID,],myy[-testID,],ntree = ntree, probability = T)  rf\_yhat = predict(model,myx[testID,],type = "prob")  stack.res[j,,2] = rf\_yhat[,2]  ##Neural Network  model = nnet(ANGLE.CLOSURE~.,data = data[-testID,], size = size,decay = decay )  nn\_yhat = predict(model,data[testID,],type = "raw")  stack.res[j,,3] = nn\_yhat  ##Boosting  model = ada(myx[-testID,],myy[-testID,] ,nu=nu)  b\_yhat = predict(model,as.data.frame(myx[testID,]),type = "prob")  stack.res[j,,4] = b\_yhat[,2]  ## K Nearest Neighbour  model = kknn(ANGLE.CLOSURE~.,train = data[-testID,],test = data[testID,], k = parak,distance = distance )  k\_yhat = model$prob  stack.res[j,,5] = k\_yhat[,2]  stack.y.res[j,,1] = data[testID,1]  }  stack\_pres = matrix(0,dim(stack.res)[2]\*dim(stack.res)[1],dim(stack.res)[3])  stack\_ys = matrix(NA,dim(stack.y.res)[2]\*dim(stack.y.res)[1],1)  for(ii in 1:dim(stack.res)[1]){  for(jj in 1:dim(stack.res)[2]){  for(kk in 1:dim(stack.res)[3]){  stack\_pres[( (ii-1)\*dim(stack.res)[2]+jj),kk] = stack.res[ii,jj,kk]  }  stack\_ys[(ii-1)\*dim(stack.y.res)[2]+jj,1] = stack.y.res[ii,jj,1]  }  }  Dmat = t(stack\_pres)%\*%stack\_pres  dvec = t(stack\_ys)%\*%stack\_pres  Amat <- cbind(rep(1,5), diag(5))  bvec <- c(1,rep(0,5))  weightsConstrained = as.numeric(solve.QP(Dmat = Dmat, dvec = dvec, Amat = Amat, bvec = bvec, meq = 1)$solution)  weightsUnConstrained=solve.QP(Dmat = Dmat, dvec = dvec, Amat = Amat, bvec = bvec, meq = 1)$unconstrained.solution  weightsConstrained  weightsUnConstrained  dput(stack\_pres,"stack\_pres")  dput(stack\_ys,"stack\_ys")  dput(weightsUnConstrained,"weightsUnConstrained")  dput(weightsConstrained,"weightsConstrained")  ##read data  weightsUnConstrained =dget(weightsUnConstrained)  weightsConstrained = dget(weightsConstrained)  mytest = dget("mytest.r")  data = read.csv("cleandata.csv")[,-1]  myy = data[,1,drop = F]  myx = data.matrix(data[,-1])  ##svm  model = svm(myx,myy[,],cost = cost,gamma = gamma, probability = T)  yhat = predict(model,mytest[,-1],probability = T)  yhat = attr(yhat,"probabilities")[,1]  ##randomforest  model = randomForest(myx[,],myy[,],ntree = ntree, probability = T)  rf\_yhat = predict(model,mytest[,],type = "prob")[,2]  ##Neural Network  model = nnet(ANGLE.CLOSURE~.,data = data[,], size = size,decay = decay )  nn\_yhat = predict(model,mytest[,],type = "raw")  ##Boosting  model = ada(myx[,],myy[,] ,nu=nu)  b\_yhat = predict(model,as.data.frame(mytest[,]),type = "prob")[,2]  ## K Nearest Neighbour  model = kknn(ANGLE.CLOSURE~.,train = data[,],test = mytest[,], k = parak,distance = distance )  k\_yhat = model$prob[,2]  (cbind(yhat,rf\_yhat,nn\_yhat,b\_yhat,k\_yhat))  #weightsConstrained=c(0.4391,0.1338,-5.111e-18,4.004e-02,0.3871)  #weightsUnConstrained=c(0.4459,0.1128,-0.03761,0.1118,0.3923)  #weightsConstrained=c(0.2778,0,0.2677,0.1662,0.2883)  #weightsUnConstrained=c(1.0330,4.4781,1.0558,1.2125,1.0136)    yhat = cbind(yhat,rf\_yhat,nn\_yhat,b\_yhat,k\_yhat) %\*% matrix(weightsConstrained,,1)  roc = roc(mytest[,1], yhat[,1])  auc(roc)## 0.9628  plot(roc)  yhat = cbind(yhat,rf\_yhat,nn\_yhat,b\_yhat,k\_yhat) %\*% matrix(weightsUnConstrained,,1)  roc = roc(mytest[,1], yhat[,1])  auc(roc)## 0.9236  plot(roc) |

Weights Table

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| weightsConstrained | 0.000000e+00 | 5.871550e-01 | 4.128450e-01 | 6.950806e-16 | 3.693715e-17 |
| weightsUnConstrained | -3.6789782 | 0.2450442 | 1.8448642 | -1.4602581 | 6.2200210 |

5 Validation

Generate predictions on the angle closure glaucoma positive (“AngleClosureValidationCases.csv” on T-square) and angle closure glaucoma negative (“AngleClosure ValidationControls.csv” on T-square) validation datasets. Use right eye data preferentially. We will be interested in both the AUC and the actual ROC curve

Code for generating test data set

|  |
| --- |
| ## use input column index ##  case = read.csv("AngleClosure\_ValidationCases.csv")  control = read.csv("AngleClosure\_ValidationControls.csv")  myCasesNewL = c(7,9,11,12,13,14,15,30,31,32,36)  myCasesNewR = c(19,21, 23:27,30:32,36)  myControlsNewR <- c(18,20,22,23,24,25,26,29,30,31,35)  myControlsNewL <- c(6,8,10,11,12,13,14,29,30,31,35)  remove\_index\_set = apply(case[,myCasesNewR],1,function(xx){  return(sum(is.na(xx))>0)  })  caseR = case[!remove\_index\_set,]  remove\_index\_set = apply(case[remove\_index\_set,myCasesNewL],1,function(xx){  return(sum(is.na(xx))>0)  })  caseL = case[!remove\_index\_set,]  case = rbind(caseR,caseL)  remove\_index\_set = apply(control[,myControlsNewR],1,function(xx){  return(sum(is.na(xx))>0)  })  controlR = control[!remove\_index\_set,]  remove\_index\_set = apply(control[remove\_index\_set,myControlsNewL],1,function(xx){  return(sum(is.na(xx))>0)  })  controlL = control[!remove\_index\_set,]  control = rbind(controlR,controlL)  colnames(case)[30] = "ACW\_mm"  colnames(control)[29] = "ACW\_mm"  mycase = cbind(rep("YES",dim(case)[1] ),case[,myCasesNewR] )  mycontrol = cbind(rep("NO", dim(control)[1] ),control[,myControlsNewR] )  for(ii in 1:dim(mycase)[2] ){  if(ii==1){  colnames(mycase)[1] = colnames(myy)[ii]  colnames(mycontrol)[1] = colnames(myy)[ii]  }else{  colnames(mycase)[ii] = colnames(myx)[ii-1]  colnames(mycontrol)[ii] = colnames(myx)[ii-1]  }  }  mytest = rbind( mycase,mycontrol )  dput(mytest,"mytest.r") |

Number of completed test samples:400

6 Visualization

6.1 For each of the 5 base prediction models, generate plots of cross-validated AUC vs. tuning parameter values.

|  |  |
| --- | --- |
| SVM | Random Forest |
| Tunned Parameter:cost=1, gamma=0.003981072 | Tunned Parameter:ntree=750 |
|  |  |
| Neural Network | Boosting |
| Tunned Parameter:size=6, decay=0.5 | Tunned Parameter:nu=0.1 |
|  |  |
| K Nearest Neighbour |  |
| Tunned Parameter:k=99,distance=1 |  |
|  |  |

6.2 For each of the 7 prediction models (5 base prediction models + 2 stacked models), generate ROC curves (plots) annotated with the corresponding AUCs using the validation datasets.

|  |  |
| --- | --- |
| SVM | Random Forest |
| AUC=0.9482 | AUC=0.9549 |
|  |  |
| Neural Network | Boosting |
| AUC=0.9711 | AUC=0.9626 |
|  |  |
| K Nearest Neighbour | Constrained Stack Model |
| AUC=0.959 | AUC=0.9628 |
|  |  |
| Unconstrained Stack Model |  |
| AUC=0.9236 |  |
|  |  |