Data Manipulation

First, we import the data and remove the columns corresponding to EYE, GENDER, ETHNIC, HGT, WT, ASPH, ACYL, SE, AXL, CACD, AGE, CCT.OD, PCCURV_mm.

We then remove the rows having at least one missing value such as (NA, or ".").

```
allData = read.csv("AngleClosure.csv",header=TRUE,na.strings=".")
colToRemove = sapply(attributes(allData)$names,function(name){
  if (name %in%
c("EYE","GENDER","ETHNIC","HGT","WT","ASPH","ACYL","SE","AXL","CACD","A
GE","CCT.OD","PCCURV_mm")){
    return(FALSE)
  } else {
    return(TRUE)
  }
})
DataWOcol = allData[,colToRemove]
log = is.na(DataWOcol) #1 if NA, 0 otherwise
RowsToRemove = apply(log,1,any)
Data = DataWOcol[!RowsToRemove,] #select the rows without missing
values
Y = Data$ANGLE.CLOSURE
X = as.matrix(Data[ , !(names(Data) %in% "ANGLE.CLOSURE")])
DataOfficial=data.frame(Y,X)
```

Develop Prediction Models

In this part, we will develop prediction models and tune the parameters to have the highest AUC as possible.
The models considered in this part are:

- Random Forest (try and Ntrees)
- Neural Net (size and decay)
- Adaboost (iter and nu)
- Support Vector Machine (cost and kernel)
- Logistic regression model (step)

For each combination of parameters, this model will be tested using 10 random iterations of 10-fold cross-validation.

In order to gain efficiency and build a standardized process, I coded a unique function allowing to choose the model, number of cross validations, and the range of the tuning parameters. It will also plot the corresponding graph and ROC.

```
frameworkTuning<-
function(AllDat,N_fold,param2,param1,modelName,axeName2,legendName1){
  #Create 10 equally size folds
  AllDa <- AllDat[sample(nrow(AllDat)),]
  folds <- cut(seq(1,nrow(AllDa)),breaks=N_fold,labels=FALSE)
```

```
  #Perform 10 fold cross validation
  AUC_vec=NULL
  xy=NULL
  AUC_RF_plot=list()
  #param1=c(1,5,10)
  #param2=c(500,1000,1500)
  AUCrange=NULL
  dev.new(width=10,height=10)
  par(mai=c(0.3,0.3,0.3,0.3),mfrow=c(length(param1),length(param2)))
  for (a in param1){
    xy=NULL
    for (b in param2){
      Act=NULL
      pre=NULL
      for(i in 1:N_fold){
        testIndexes <- which(folds==i,arr.ind=TRUE)
        testData <- AllDa[testIndexes, ]
        trainData <- AllDa[-testIndexes, ]
        print(c(a,b,i))
        if (modelName
=="RF"){fit=randomForest(Y~.,data=trainData,mtry=b,n.trees=a)}
        else if(modelName
=="nnet"){fit=nnet(Y~.,data=trainData,size=b,decay=a)}
        else if(modelName
=="ada"){fit=ada(Y~.,data=trainData,iter=a,nu=b)}
        else if(modelName
=="svm"){fit=svm(Y~.,data=trainData,cost=b,kernel=a,probability=TRUE)}
#
        else if(modelName =="glm"){fit=glm(Y~.,data=(step(glm(Y~.,
data=trainData, family=binomial),steps=b)$model), family=binomial)}

        if(modelName =="glm"){myPreds=predict(fit,newdata=testData[,-
1],type='response')}
        else if(modelName
=="svm"){myPreds=attr(predict(fit,newdata=testData[,-
1],type='response',probability=TRUE),"probabilities")[,1]}
        else if (modelName =="ada" || modelName
=="RF"){myPreds=predict(fit,newdata=testData[,-1], probability =
TRUE,"prob")[,2]}
        else if (modelName
=="nnet"){myPreds=predict(fit,newdata=testData[,-1], probability =
TRUE)[,1]}
        AUC_vec=
c(AUC_vec,auc(roc(as.numeric(testData[,1]),as.numeric(myPreds))))


        Act=c(Act,as.numeric(testData[,1])-1)
        pre=c(pre,as.numeric(myPreds))
      }
      AUC_RF=mean(AUC_vec)
      roc.plot(Act,pre,, plot.thres = NULL)
      mtext(paste(legendName1," = ",a, " and ",axeName2," = ",b
),cex=0.8)
      legend("bottomright",legend=paste("AUC = ",
round(AUC_RF,digits=3)), bty ="n", pch=NA)

      xy= cbind(xy,c(b,AUC_RF))
```

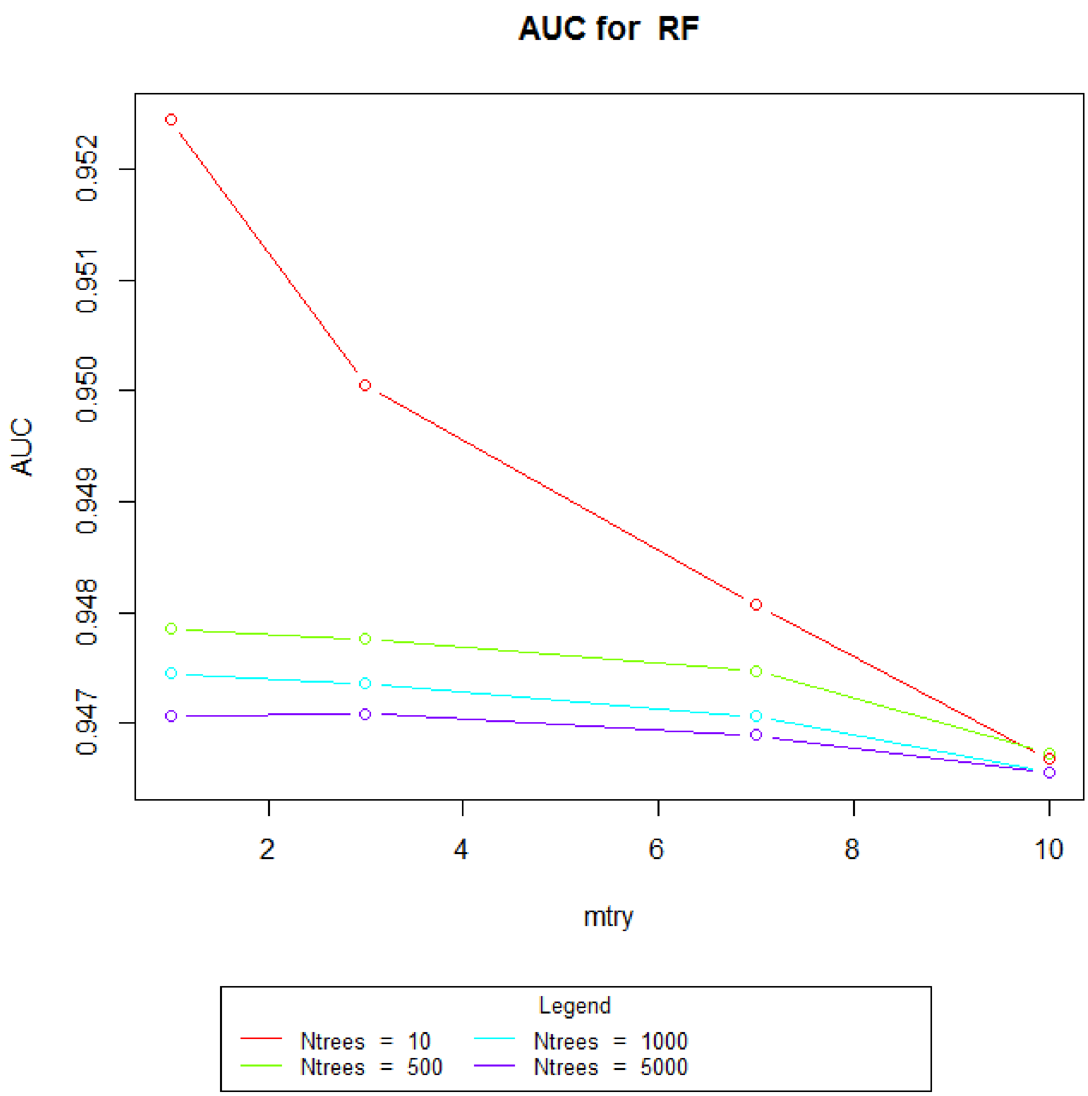
```
      AUCrange=c(AUCrange,AUC_RF)
    }
    AUC_RF_plot[[which(a==param1)]] = xy
  }
  dev.new(width=10,height=10)
  layout(rbind(1,2), heights=c(7,1))
  plot(range(param2), range(AUCrange), type="n", xlab=axeName2,
       ylab="AUC" )
  # linetype <- c(1:length(param1))
  colors <- rainbow(length(param1))
  # add a title and subtitle
  title(paste("AUC for ",modelName))
  #,lty=linetype[j]
  for (j in 1:length(param1)){lines(AUC_RF_plot[[j]][1,],
AUC_RF_plot[[j]][2,], type="b", lwd=1.5, col=colors[j])}
  par(mar=c(0, 0, 0, 0))
  plot.new()
  #, lty=linetype
  legend('center','groups', paste(legendName1,paste(" = ",param1)),
cex=0.8, col=colors, lty=1, title="Legend",ncol=3)
}
```

Model and Tuning Parameter Selection

In this section, we are going to run the function above in order to tune our models.

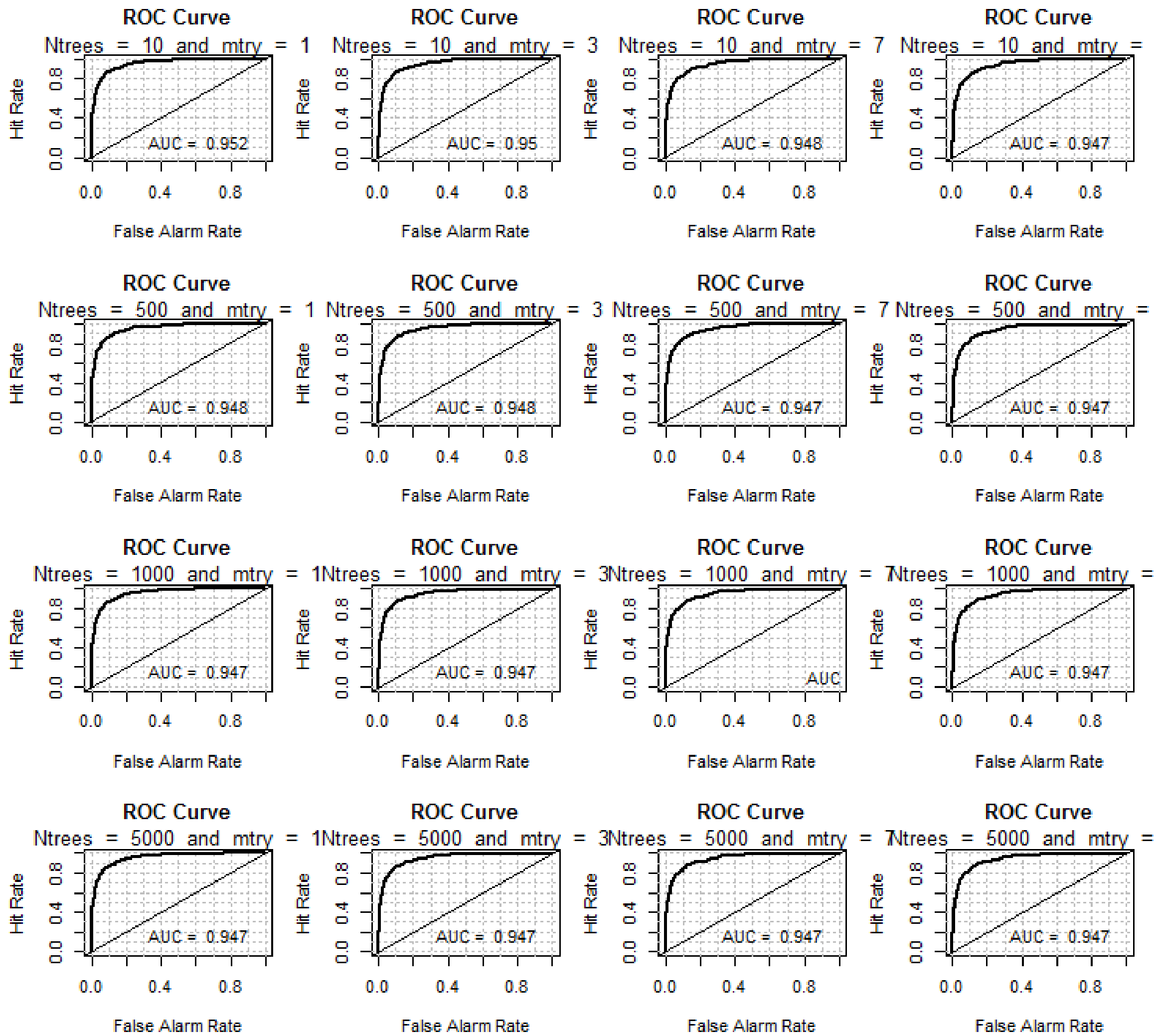
Random Forest

```
frameworkTuning(DataOfficial,10,c(1,3,7,10),c(10,500,1000,5000),"RF","m
try","Ntrees")
```



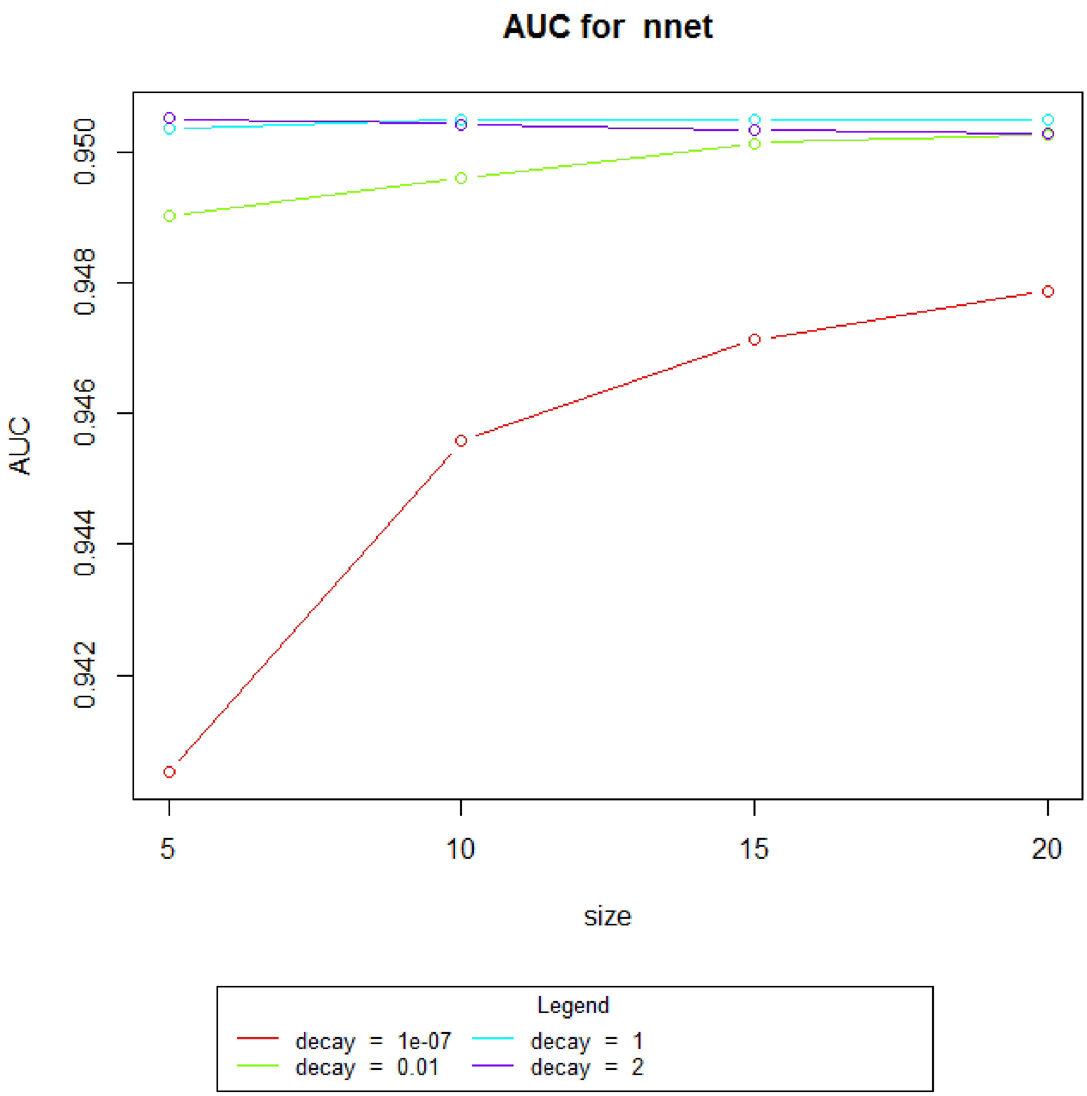
The AUC is maximized for mtry=1 and Ntrees =10. We will select this parameters for our model.

The ROC curves for each alternative is presented below.


ROC Curve
Ntrees = 10 and mtry = 1
AUC = 0.952
ROC Curve
Ntrees = 10 and mtry = 3
AUC = 0.95
ROC Curve
Ntrees = 10 and mtry = 7
AUC = 0.948
ROC Curve
Ntrees = 10 and mtry =
AUC = 0.947
ROC Curve
Ntrees = 500 and mtry = 1
AUC = 0.948
ROC Curve
Ntrees = 500 and mtry = 3
AUC = 0.948
ROC Curve
Ntrees = 500 and mtry = 7
AUC = 0.947
ROC Curve
Ntrees = 500 and mtry =
AUC = 0.947
ROC Curve
Ntrees = 1000 and mtry = 1
AUC = 0.947
ROC Curve
Ntrees = 1000 and mtry = 3
AUC = 0.947
ROC Curve
Ntrees = 1000 and mtry = 7
AUC
ROC Curve
Ntrees = 1000 and mtry =
AUC = 0.947
ROC Curve
Ntrees = 5000 and mtry = 1
AUC = 0.947
ROC Curve
Ntrees = 5000 and mtry = 3
AUC = 0.947
ROC Curve
Ntrees = 5000 and mtry = 7
AUC = 0.947
ROC Curve
Ntrees = 5000 and mtry =
AUC = 0.947
Hit Rate
False Alarm Rate
0.0
0.4
0.8

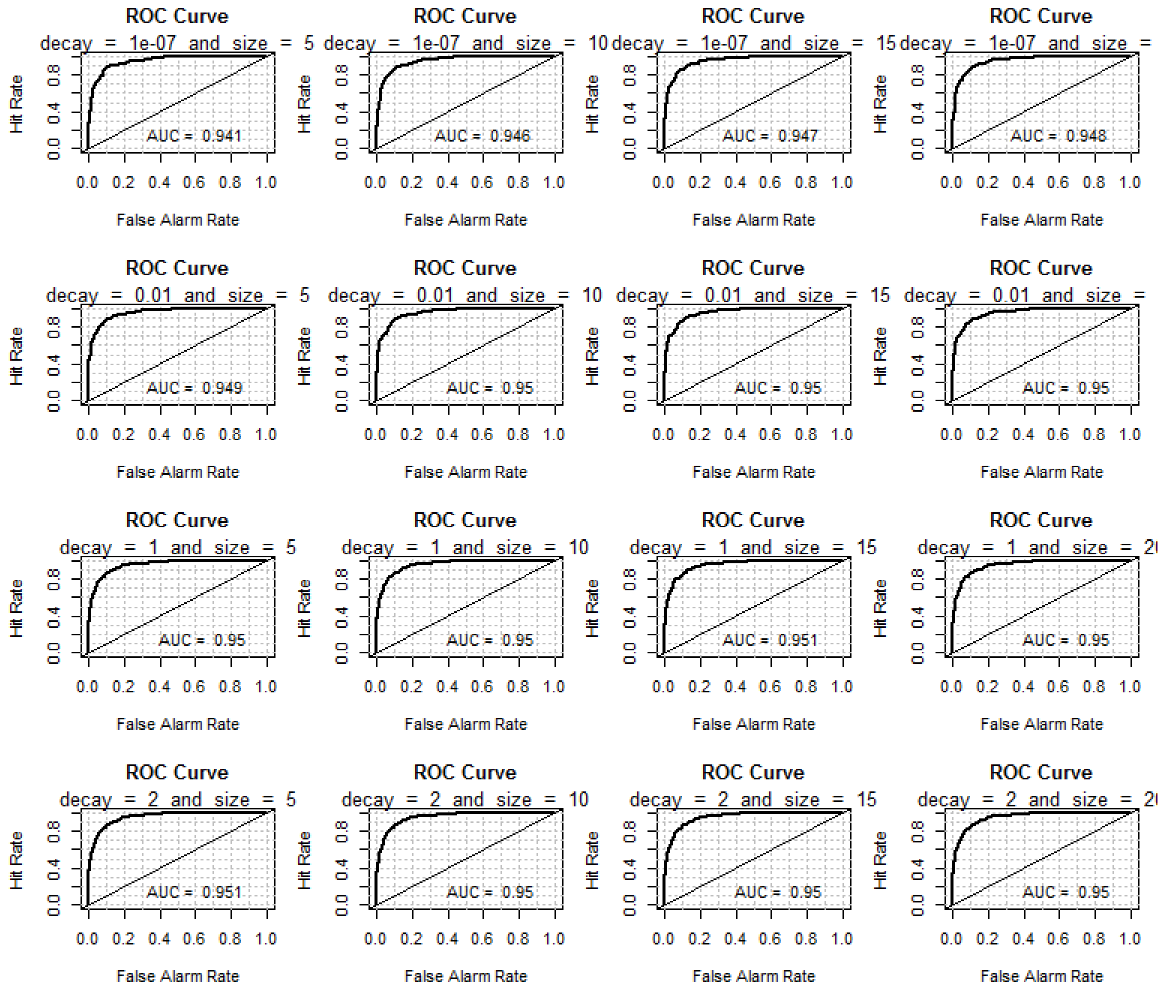
NeuralNet

```
frameworkTuning(DataOfficial,10,c(5,10,15,20),c(10^-7,10^-2,1,2),"nnet","size","decay")
```



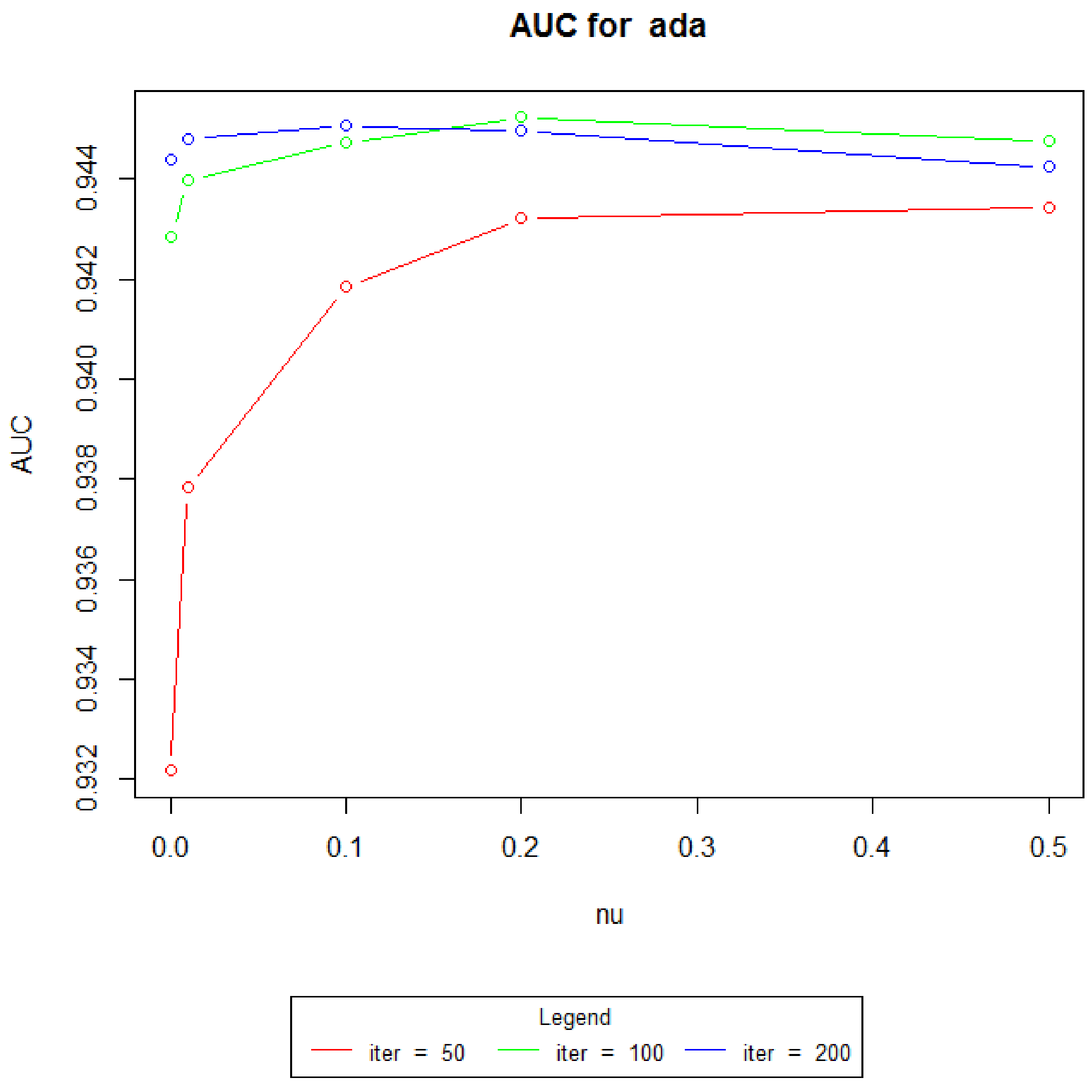
The AUC is maximized for size =15 and decay=1. We will select this parameters for our model.

The ROC curves for each alternative is presented below.


ROC Curve
decay = 1e-07 and size = 5
AUC = 0.941
ROC Curve
decay = 1e-07 and size = 10
AUC = 0.946
ROC Curve
decay = 1e-07 and size = 15
AUC = 0.947
ROC Curve
decay = 1e-07 and size =
AUC = 0.948
ROC Curve
decay = 0.01 and size = 5
AUC = 0.949
ROC Curve
decay = 0.01 and size = 10
AUC = 0.95
ROC Curve
decay = 0.01 and size = 15
AUC = 0.95
ROC Curve
decay = 0.01 and size =
AUC = 0.95
ROC Curve
decay = 1 and size = 5
AUC = 0.95
ROC Curve
decay = 1 and size = 10
AUC = 0.95
ROC Curve
decay = 1 and size = 15
AUC = 0.951
ROC Curve
decay = 1 and size = 2
AUC = 0.95
ROC Curve
decay = 2 and size = 5
AUC = 0.951
ROC Curve
decay = 2 and size = 10
AUC = 0.95
ROC Curve
decay = 2 and size = 15
AUC = 0.95
ROC Curve
decay = 2 and size = 2
AUC = 0.95
Hit Rate
False Alarm Rate
0.0 0.2 0.4 0.6 0.8 1.0
0.0 0.4 0.8

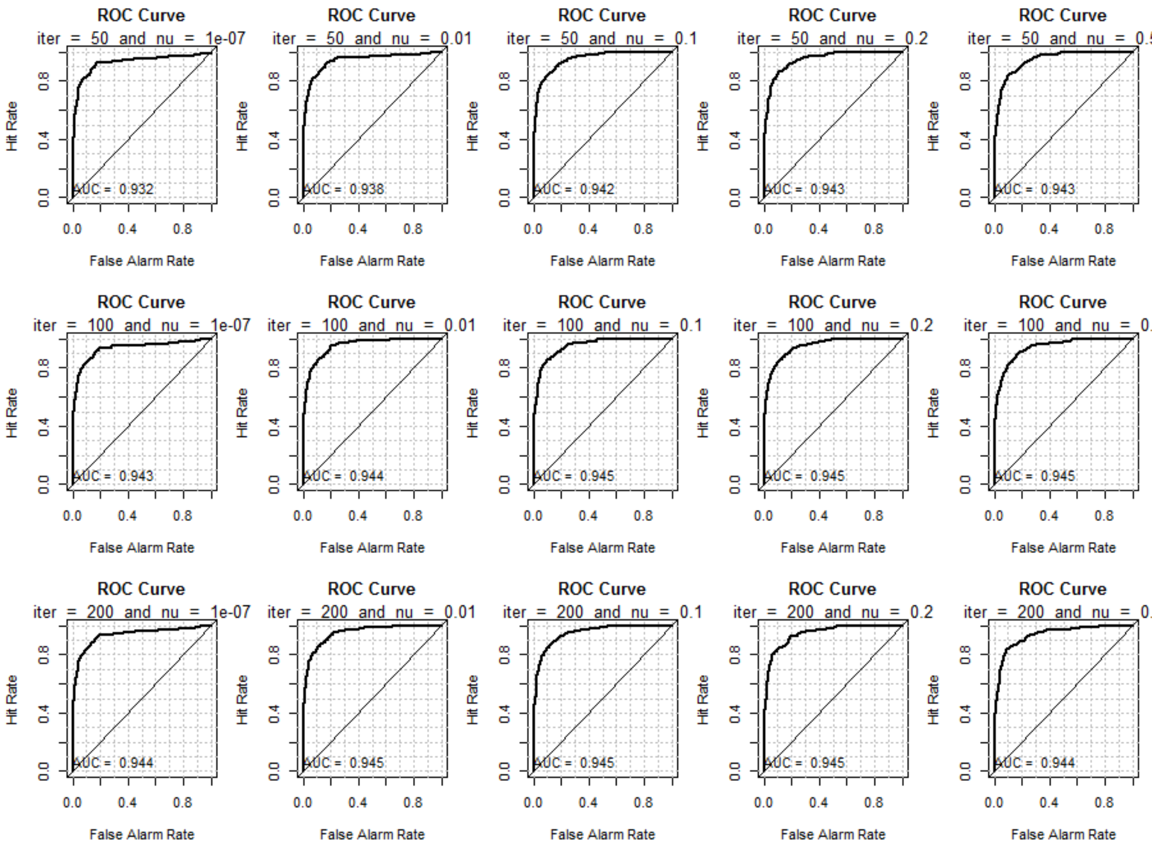
AdaBoost

```
frameworkTuning(DataOfficial,10,c(10^-7,10^-2,10^-
1,0.2,0.5),c(50,100,200),"ada","nu","iter")
```

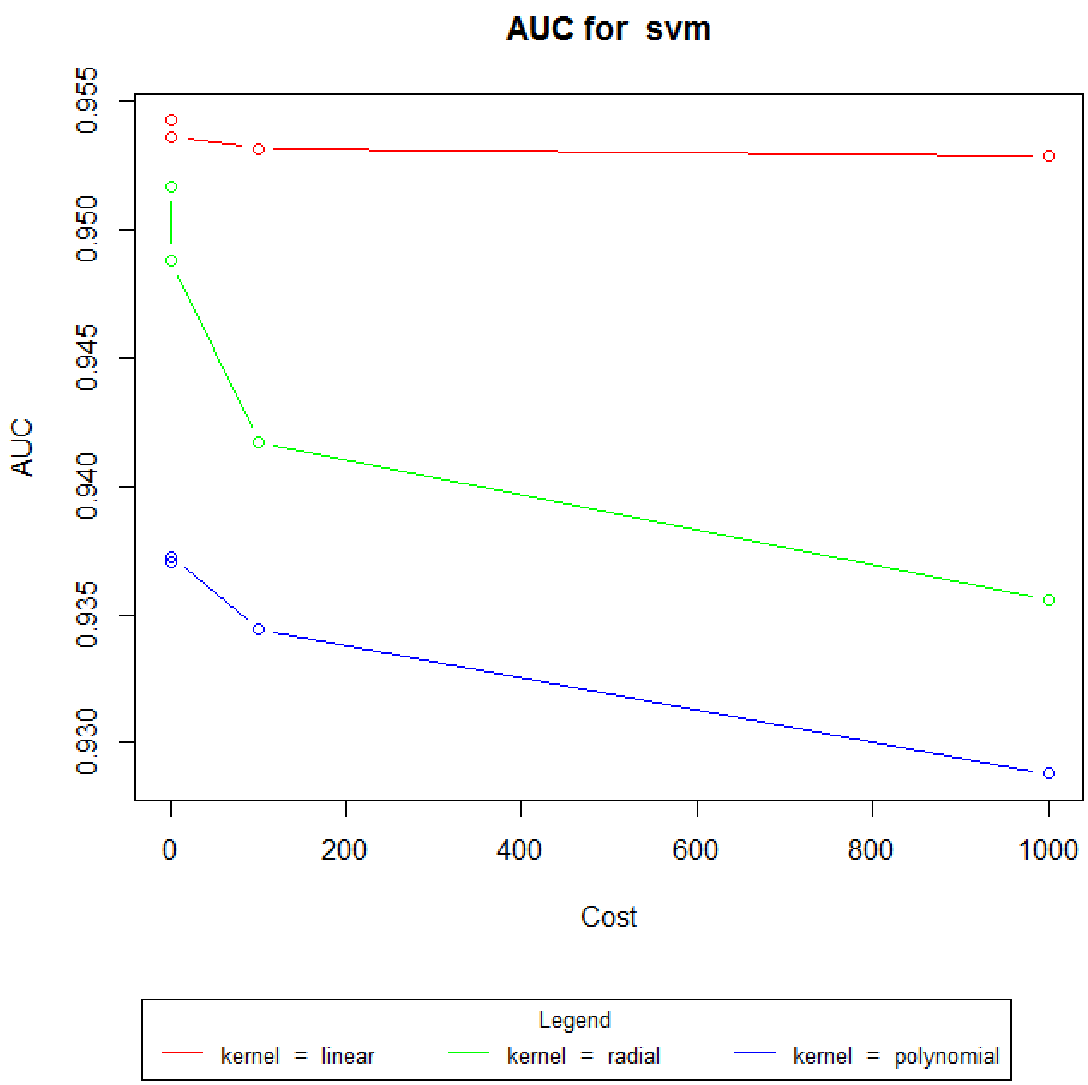


The AUC is maximized for Nu =0.2 and iter=100. We will select this parameters for our model.

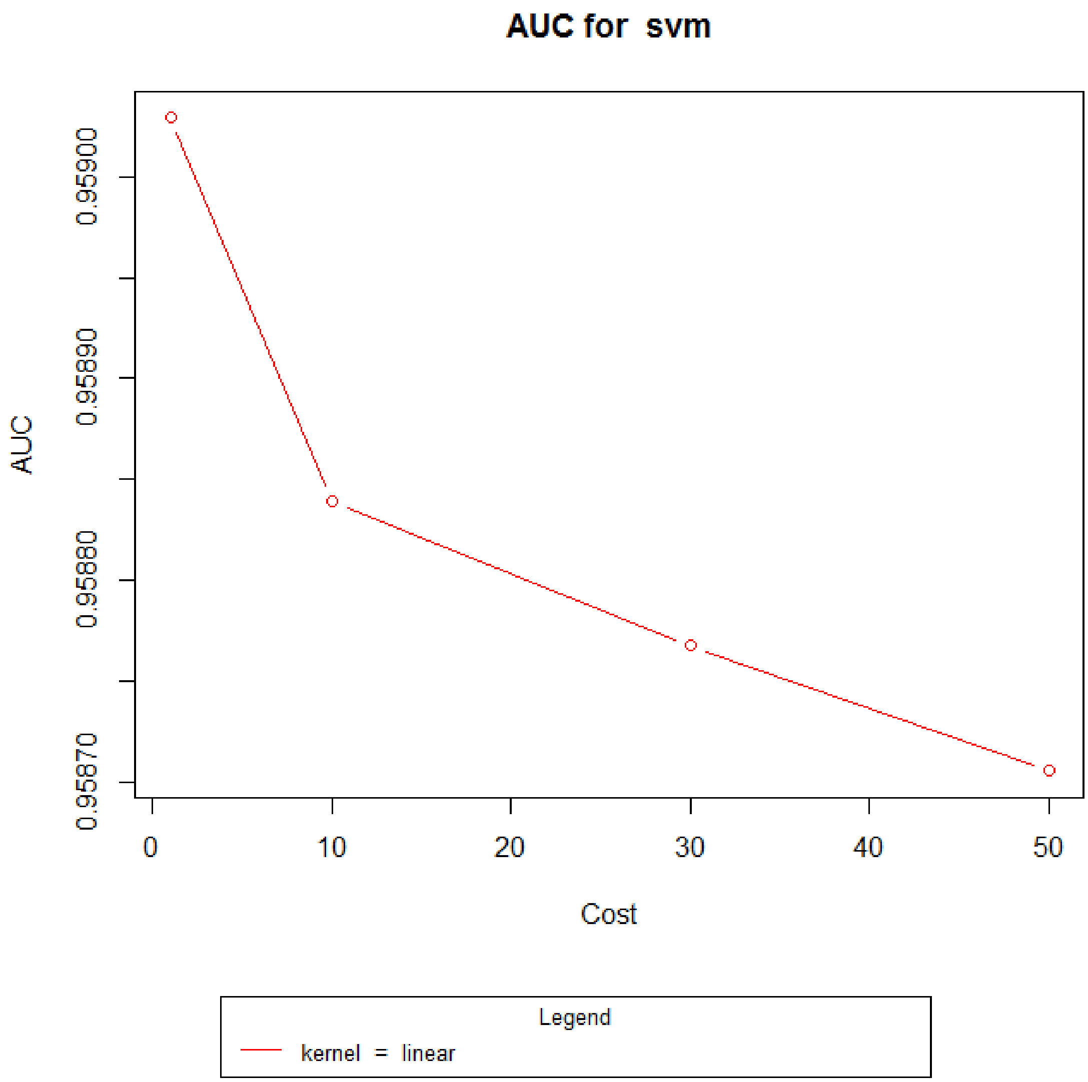
The ROC curves for each alternative is presented below.


ROC Curve
iter = 50 and nu = 1e-07
AUC = 0.932
ROC Curve
iter = 50 and nu = 0.01
AUC = 0.938
ROC Curve
iter = 50 and nu = 0.1
AUC = 0.942
ROC Curve
iter = 50 and nu = 0.2
AUC = 0.943
ROC Curve
iter = 50 and nu = 0.
AUC = 0.943
ROC Curve
iter = 100 and nu = 1e-07
AUC = 0.943
ROC Curve
iter = 100 and nu = 0.01
AUC = 0.944
ROC Curve
iter = 100 and nu = 0.1
AUC = 0.945
ROC Curve
iter = 100 and nu = 0.2
AUC = 0.945
ROC Curve
iter = 100 and nu = 0.
AUC = 0.945
ROC Curve
iter = 200 and nu = 1e-07
AUC = 0.944
ROC Curve
iter = 200 and nu = 0.01
AUC = 0.945
ROC Curve
iter = 200 and nu = 0.1
AUC = 0.945
ROC Curve
iter = 200 and nu = 0.2
AUC = 0.945
ROC Curve
iter = 200 and nu = 0.
AUC = 0.944
Hit Rate
False Alarm Rate
0.0
0.4
0.8

Support Vector Machine

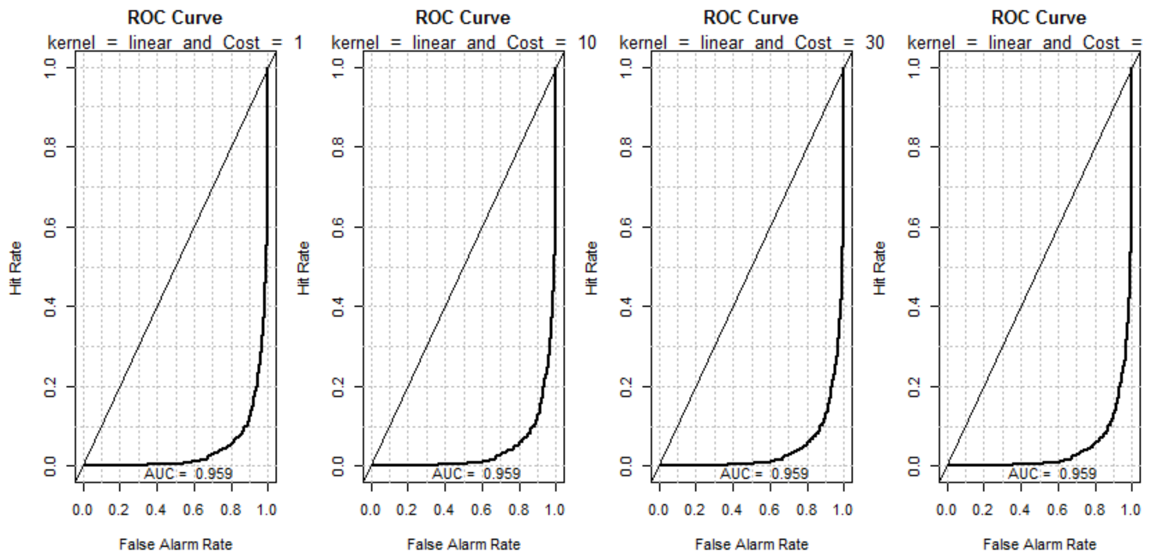



```
frameworkTuning(DataOfficial,10,c(1,10,30,50),c("linear"),"svm","Cost",
"kernel")
```



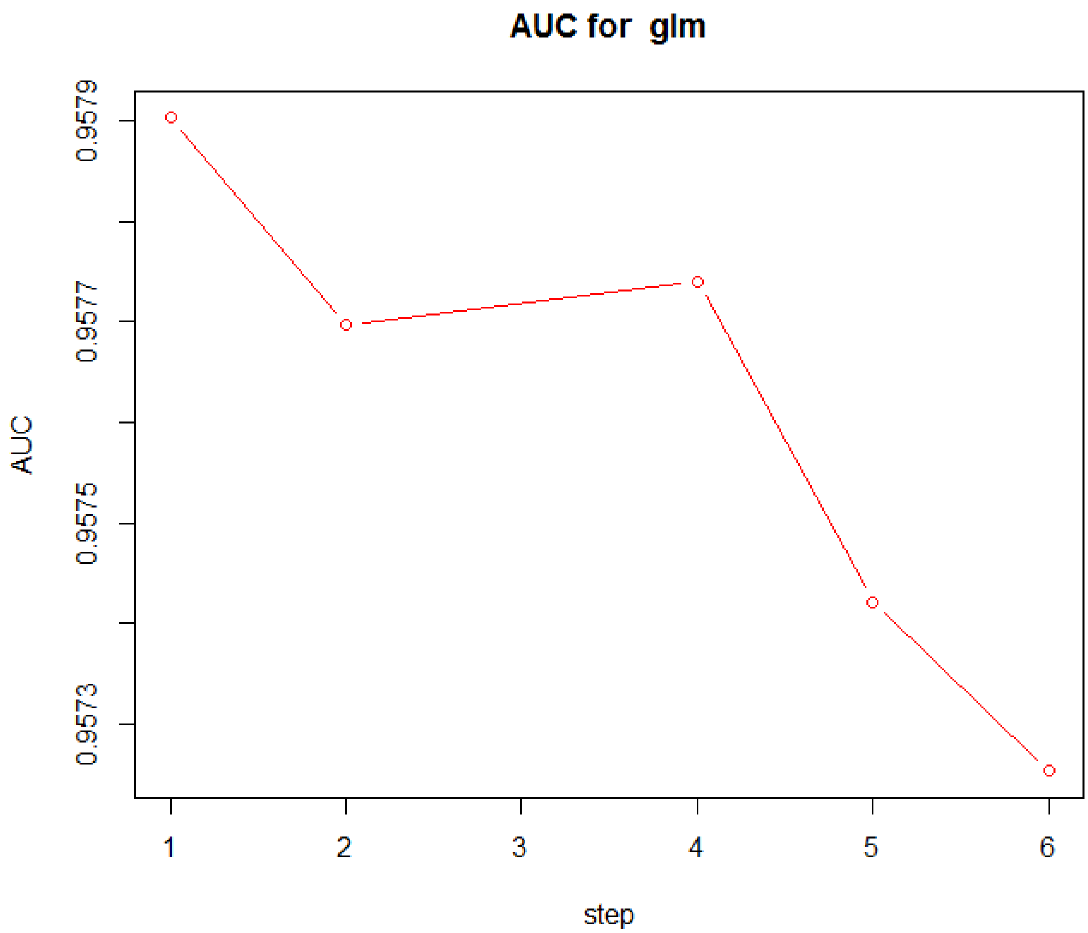
The AUC is maximized for kernel = linear and cost =1. We will select this parameters for our model.

The ROC curves for each alternative is presented below.


ROC Curve
kernel = linear and Cost = 1
AUC = 0.959
ROC Curve
kernel = linear and Cost = 10
AUC = 0.959
ROC Curve
kernel = linear and Cost = 30
AUC = 0.959
ROC Curve
kernel = linear and Cost = 5
AUC = 0.959
Hit Rate
False Alarm Rate

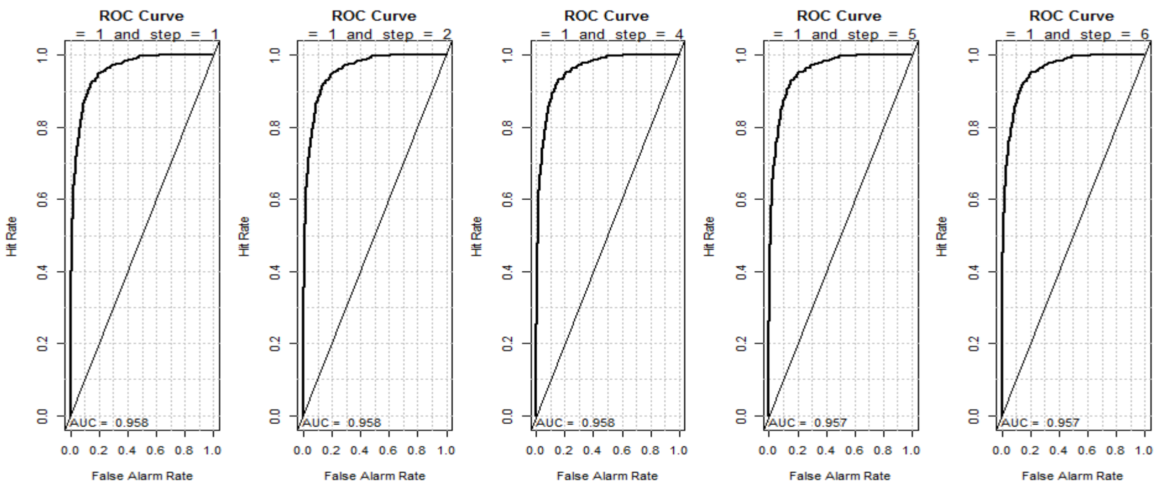
Logistic Regression

```
frameworkTuning(DataOfficial,10,c(1,2,4,5,6),c(1),"glm","step","")
```



The AUC is maximized for step =1. We will select this parameters for our model.

The ROC curves for each alternative is presented below.


ROC Curve
= 1 and step = 1
AUC = 0.958
ROC Curve
= 1 and step = 2
AUC = 0.958
ROC Curve
= 1 and step = 4
AUC = 0.958
ROC Curve
= 1 and step = 5
AUC = 0.957
ROC Curve
= 1 and step = 6
AUC = 0.957
Hit Rate
False Alarm Rate
0.0 0.2 0.4 0.6 0.8 1.0

Stacking

First, we gather the prediction for each model for each cross validation in a single matrix `resultSTacked`.

```
N_fold=10
AllDa <- DataOfficial[sample(nrow(DataOfficial)),]
folds <- cut(seq(1,nrow(AllDa)),breaks=N_fold,labels=FALSE)
resultSTacked=NULL
Actual=NULL
for(i in 1:N_fold){
  testIndexes <- which(folds==i,arr.ind=TRUE)
  testData <- AllDa[testIndexes, ]
  trainData <- AllDa[-testIndexes, ]
  print(c(a,b,i))
  fit=randomForest(Y~.,data=trainData,mtry=1,n.trees=10)
  RF=predict(fit,newdata=testData[,-1],"prob")[,2]
  fit=nnet(Y~.,data=trainData,size=15,decay=1)
  nenet=predict(fit,newdata=testData[,-1], probability = TRUE)[,1]
  fit=ada(Y~.,data=trainData,iter=100,nu=0.2)
  ada=predict(fit,newdata=testData[,-1], probability = TRUE,"prob")[,2]
  fit=svm(Y~.,data=trainData,cost=1,kernel="linear",probability=TRUE)
  svm=attr(predict(fit,newdata=testData[,-
1],type='response',probability=TRUE),"probabilities")[,1]
  fit=glm(Y~.,data=(step(glm(Y~., data=trainData,
family=binomial),steps=1)$model), family=binomial)
  glm=predict(fit,newdata=testData[,-1],type='response')

  predict_mat=cbind(RF,nenet,ada,svm,glm)
  resultSTacked=rbind(resultSTacked,predict_mat)
  Actual = c(Actual,as.numeric(testData[,1]=="YES"))
}
```

Unconstrained

$$Q = (y - Xw)^T(y - Xw)$$
$$w = (X^TX)^{-1}X^Ty$$

As a result, the following code given the unconstrained stacked model is:

```
weight=solve((t(resultSTacked)%*%resultSTacked))%*%t(resultSTacked)%*%A
ctual
pred=resultSTacked%*%weight
```

The weight obtained are:

RF	Nnet	Ada	SVM	GLM
0.046234257	0.116081849	0.231421295	0.005092313	0.604474940

Constrained

The constrained stacking impose that $\sum_{m=1}^{5} w_m = 1$ and $w_m \geq 0$.
We will use the quadratic forn and the R function solbe.QP form the library quadprog to solve it.

$$Q = \frac{1}{2} w^T X^T X w - y^T X w$$

The corresponding code is:

```
constraints=t(rbind(c(1,1,1,1,1),diag(1,5,5)))
weightConst = solve.QP( t(resultSTacked) %*% resultSTacked, t(Actual)
%*% resultSTacked, constraints, c(1,0,0,0,0,0),meq=1)$solution
predConst=resultSTacked%*%weightConst
```

The weights obtained are:

RF	Nnet	Ada	SVM	GLM
0.039289638	0.114430111	0.232744138	0.004938217	0.608597896

Validation

In this part, we take the data contained in the file "AngleClosure_ValidationCases.csv" and "AngleClosure_ValidationControls.csv". We will build our predictors in order to keep only the data for the right eye if available, otherwise will use the data of the left eye. We will remove all the rows with missing data.

The corresponding code is the following:

```
case=read.csv("AngleClosure_ValidationCases.csv",header=TRUE)
control=read.csv("AngleClosure_ValidationControls.csv",header=TRUE)


r=c(case$rAOD750,control$rAOD750)
l=c(case$lAOD750,control$lAOD750)
log=is.na(r)
AOD750=r
AOD750[log]=l[log]

r=c(case$rTISA750,control$rTISA750)
l=c(case$lTISA750,control$lTISA750)
log=is.na(r)
TISA750=r
TISA750[log]=l[log]

r=c(case$rIT750,control$rIT750)
l=c(case$lIT750,control$lIT750)
log=is.na(r)
IT750=r
IT750[log]=l[log]

r=c(case$IT2000,control$rIT2000)
l=c(case$lIT2000,control$lIT2000)
log=is.na(r)
IT2000=r
IT2000[log]=l[log]

r=c(case$rITCM,control$rITCM)
l=c(case$lITCM,control$lITCM)
log=is.na(r)
ITCM=r
ITCM[log]=l[log]


r=c(case$rIAREA,control$rIAREA)
l=c(case$lIAREA,control$lIAREA)
log=is.na(r)
IAREA=r
IAREA[log]=l[log]

r=c(case$rICURV,control$rICURV)
l=c(case$lICURV,control$lICURV)
log=is.na(r)
ICURV=r
ICURV[log]=l[log]
```

```
ACA=c(case$ACA,control$ACA)
ACV=c(case$ACV,control$ACV)
LENSVAULT=c(case$LENSVAULT,control$LENSVAULT)
ACW_mm=c(case$ACWmm,control$ACW.mm.)


Val=c(matrix(1,1,dim(case)[1]),matrix(0,1,dim(control)[1]))
MyDataValidation=na.omit(data.frame(Val,AOD750,TISA750,IT750,IT2000,ITC
M,IAREA,ICURV,ACW_mm,ACA,ACV,LENSVAULT))
```

The models are then fit and the ROC curves (shown in the next section) are plotted:

```
dev.new(width=10,height=10)
par(mai=c(0.3,0.3,0.3,0.3),mfrow=c(3,3))

fit=randomForest(Y~.,data=DataOfficial,mtry=1,n.trees=10)
RF=predict(fit,newdata=MyDataValidation[,-1],"prob")[,2]
roc.plot(MyDataValidation$Val,RF, plot.thres = NULL)
mtext(paste("RF: ","mtry = 1 and Ntrees = 10"),cex=0.8)
legend("bottomright",legend=paste("AUC = ",
round(auc(MyDataValidation$Val,RF),digits=3)), bty ="n", pch=NA)

fit=nnet(Y~.,data=DataOfficial,size=15,decay=1)
nenet=predict(fit,newdata=MyDataValidation[,-1], probability =
TRUE)[,1]
roc.plot(MyDataValidation$Val,nenet, plot.thres = NULL)
mtext(paste("Nnet: ","size = 15 and decay = 1"),cex=0.8)
legend("bottomright",legend=paste("AUC = ",
round(auc(MyDataValidation$Val,nenet),digits=3)), bty ="n", pch=NA)

fit=ada(Y~.,data=DataOfficial,iter=100,nu=0.2)
ada=predict(fit,newdata=MyDataValidation[,-1], probability =
TRUE,"prob")[,2]
roc.plot(MyDataValidation$Val,ada, plot.thres = NULL)
mtext(paste("Ada: ","iter = 50 and NNu = 1"),cex=0.8)
legend("bottomright",legend=paste("AUC = ",
round(auc(MyDataValidation$Val,ada),digits=3)), bty ="n", pch=NA)

fit=svm(Y~.,data=DataOfficial,cost=1,kernel='linear',probability=TRUE)
svm=attr(predict(fit,newdata=MyDataValidation[,-
1],type='response',probability=TRUE),"probabilities")[,1]
roc.plot(MyDataValidation$Val,svm, plot.thres = NULL)
mtext(paste("SVM: ","kernel = linear and cost = 1"),cex=0.8)
legend("bottomright",legend=paste("AUC = ",
round(auc(MyDataValidation$Val,svm),digits=3)), bty ="n", pch=NA)

fit=glm(Y~.,data=(step(glm(Y~., data=DataOfficial,
family=binomial),steps=1)$model), family=binomial)
glm=predict(fit,newdata=MyDataValidation[,-1],type='response')
roc.plot(MyDataValidation$Val,glm, plot.thres = NULL)
mtext(paste("GLM: ","step = 1"),cex=0.8)
legend("bottomright",legend=paste("AUC = ",
round(auc(MyDataValidation$Val,glm),digits=3)), bty ="n", pch=NA)

predict_mat_val=cbind(RF,nenet,ada,svm,glm)
```

```
#unconstrained stacking
predUnconst=predict_mat_val%*%weight
roc.plot(MyDataValidation$Val,predUnconst, plot.thres = NULL)
mtext(paste("Stacked: Unconstrained"),cex=0.8)
legend("bottomright",legend=paste("AUC = ",
round(auc(MyDataValidation$Val,predUnconst),digits=3)), bty ="n",
pch=NA)

#constrained stacking
predConst=predict_mat_val%*%weightConst
roc.plot(MyDataValidation$Val,RF, plot.thres = NULL)
mtext(paste("Stacked: Constrained "),cex=0.8)
legend("bottomright",legend=paste("AUC = ",
round(auc(MyDataValidation$Val,predConst),digits=3)), bty ="n", pch=NA)
```

As shown on the plots below, the best model is the nnet model and has an AUC of **0.973**.

Visualization

The ROC for each of the 5 base prediction models using the training set have been presented throughout the report.

The final ROC for the validation set is: