SDSC3006 Lab
8-Tree and SVM

Langming LIU langmiliu2-c@my.cityu.edu.hk

School of Data Science
City University of Hong Kong

Contents

- Bagging and Random Forests
- Boosting
- Support Vector Classifier

Bagging and Random Forests

Introduction

- Bagging: the average of prediction model from B seperate training sets.

$$\hat{f}_{\text{avg}}(x) = \frac{1}{B}\sum_{b=1}^{B}\hat{f}^{b}(x).$$

Reason: averaging a set of obs reduces variance.

Key: get seperate training sets by Bootstrap.

- Random Forests: just choose a random sample of m predictors as split candidates (usually set m ≈ √p)

R Implemention of Bagging

- Bagging can be viewed as a special case of a random forest with m=p. Therefore, it can be performed using the randomForest() function in the randomForest package.
- Boston data set in the MASS library: predict medv (median home price) of a neighborhood based on various predictors (totally 13 predictors)

```
install.packages("randomForest")
library(randomForest)
library(MASS)
attach(Boston)

##prepare training and test data
set.seed(1)
train = sample(1:nrow(Boston), nrow(Boston)/2)
boston.test=Boston[-train,"medv"]
```

R Implemention of Bagging

```
##bagging: randomforest with mtry=number of Predictors
set.seed(1)
bag.boston=randomForest(medv~.,data=Boston,subset=train, mtry=13,
ntree=100, importance=TRUE)
bag.boston

##calculate test MSE
yhat.bag=predict(bag.boston,newdata=Boston[-train ,])
mean((yhat.bag-boston.test)^2)

##actual observations of test data and predictions
plot(yhat.bag,boston.test)
abline(0,1)    #line with intercept 0 and slope 1
```

R Implemention of Random Forests

```
##mtry=number of Predictors
set.seed(1)
rf.boston=randomForest(medv~.,data=Boston,subset=train,
mtry=6,ntree=100,importance=TRUE)
yhat.rf=predict(rf.boston,newdata=Boston[-train,])
mean((yhat.rf-boston.test)^2)
```

Boosting

Introduction

- Boosting: at each iteration, we fit a tree using the current residuals, rather than the outcome Y, and add this new decision tree into fitted function

(a) Fit a tree $\hat{f}^b$ with d splits ($d+1$ terminal nodes) to the training data (X, r).

(b) Update $\hat{f}$ by adding in a shrunken version of the new tree:

$$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x). \tag{8.10}$$

(c) Update the residuals,

$$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i). \tag{8.11}$$

- Three parameters: number of trees B, shrinkage parameter lamda (usually 0.01 or 0.001), splits of tree d (usually 1, like a stump).

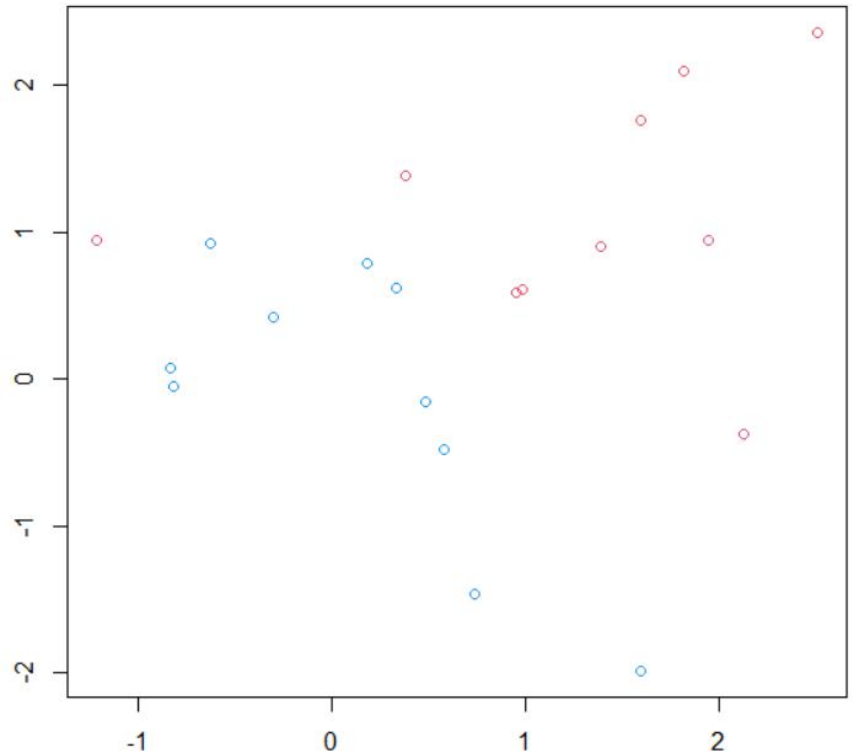
R Implementation of Boosting

```
##Use gbm() function in gbm package to fit boosted regression trees to the
Boston data
install.packages("gbm")
library(gbm)
set.seed(1)
boost.boston = gbm(medv~.,data=Boston[train,],
distribution="gaussian",n.trees=5000, interaction.depth=4)
##regression: distribution="gaussian" classification: distribution="bernoulli"

summary(boost.boston)                    ##relative influence plot
par(mfrow=c(1,2))
plot(boost.boston,i="rm")                ##partial dependence plot
plot(boost.boston,i="lstat")

yhat.boost = predict(boost.boston,newdata=Boston[-train,], n.trees=5000)
mean((yhat.boost-boston.test)^2)
```

Support Vector Classifier

Implemention

Use the e1071 library to demonstrate the support vector classifier and the SVM on a two-dimensional example.




```
install.packages("e1071")
library(e1071)

##Generate training data
set.seed(1)
x=matrix(rnorm(20*2),ncol=2)
y=c(rep(-1,10),rep(1,10))
x[y==1,]=x[y==1,]+1
plot(x,col=(3-y))        ##color=2(red), 4(blue)  red:1, blue:-1
```

Implemention

```
##Fit the support vector classifier
dat=data.frame(x=x,y=as.factor(y))
dat
svmfit=svm(y~.,data=dat,kernel="linear",cost=10,scale=FALSE)
##"cost" is similar to tuning parameter C, but with opposite effects:
small "cost", wide margin; large "cost", narrow margin

plot(svmfit,dat)
summary(svmfit)
##Find support vectors
svmfit$index

##Use a smaller value for cost
svmfit=svm(y~.,data=dat,kernel="linear",cost=0.1,scale=FALSE)
plot(svmfit,dat)
```

Implemention

```
##Use cross validation to find best value for cost
set.seed(1)
tune.out=tune(svm,y~.,data=dat,kernel="linear",
ranges=list(cost=c(0.001,0.01,0.1,1,5,10,100)))
summary(tune.out)
##Best model
bestmod = tune.out$best.model
summary(bestmod)
plot(bestmod ,dat)
```

Implemention

```
##Generate test data
xtest=matrix(rnorm(20*2),ncol=2)
ytest=sample(c(-1,1),20,rep=TRUE)
xtest[ytest==1,]=xtest[ytest==1,]+1
testdat=data.frame(x=xtest,y=as.factor(ytest))

##Prediction
ypred=predict(bestmod,testdat)
table(predict=ypred,truth=testdat$y)
```