SDSC 3006 L02
Class 8. Tree and SVM

Name: Yiren Liu
Email: yirenliu2-c@my.cityu.edu.hk

School of Data Science
City University of Hong Kong

Outline

- **Bagging and Random Forests**
- **Boosting**
- **Support Vector Classifier**

Bagging and Random Forests

Details

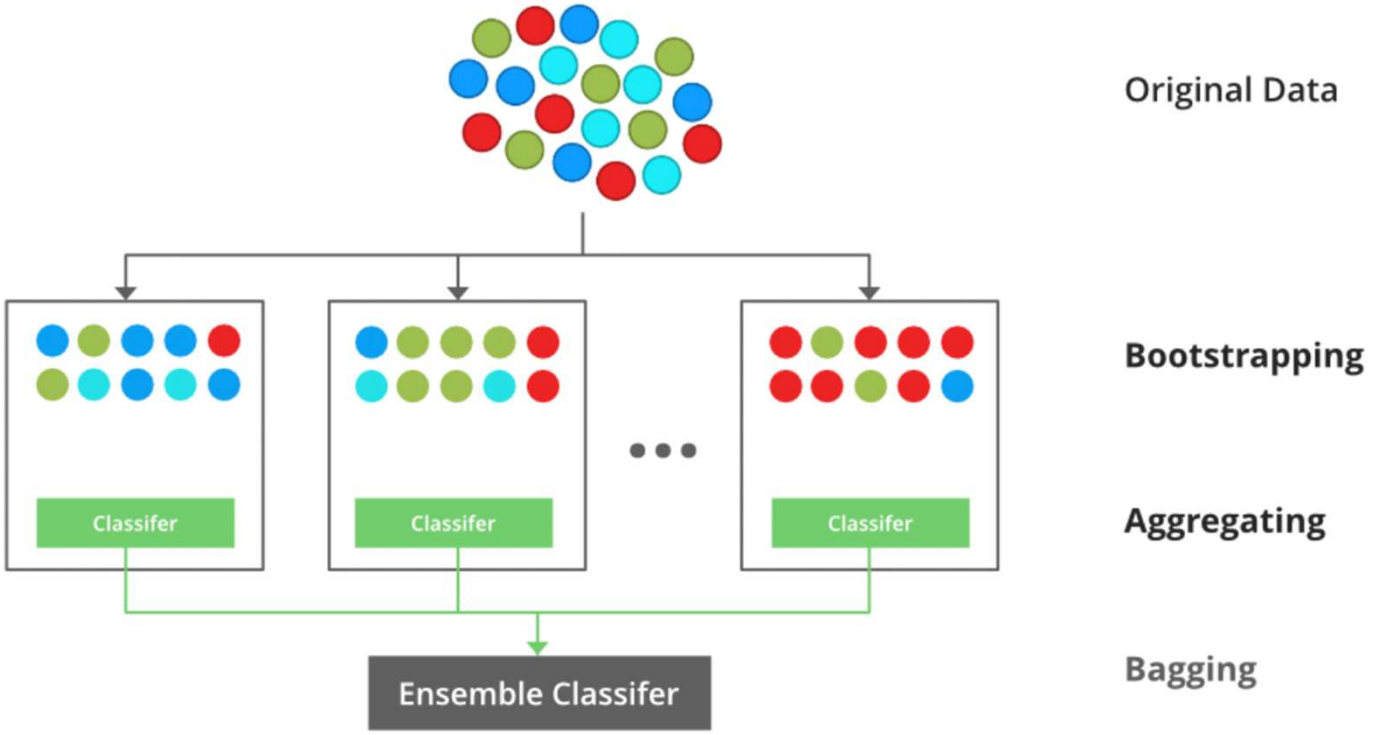
Bagging:

Bootstrap Aggregating, also known as bagging, is a machine learning ensemble meta-algorithm designed to improve the stability and accuracy of machine learning algorithms used in statistical classification and regression. It decreases the variance and helps to avoid overfitting. It is usually applied to decision tree methods. Bagging is a special case of the model averaging approach.

Details

Implementation Steps of Bagging

- **Step 1:** Multiple subsets are created from the original data set with equal tuples, selecting observations with replacement.
- **Step 2:** A base model is created on each of these subsets.
- **Step 3:** Each model is learned in parallel with each training set and independent of each other.
- **Step 4:** The final predictions are determined by combining the predictions from all the models.



Introduction

- Bagging: the average of prediction model from B separate training sets.
- $\hat{f}_{avg}(x) = \frac{1}{B}\sum_{b=1}^{B}\hat{f}^{b}(x)$
- Reason: averaging a set of obs reduces variance.
- Key: get separate training sets by Bootstrap.
- Random Forests: just choose a random sample of m predictors as split candidates (usually set m = sqrt(p))

R Implementation 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 Implementation 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 Implementation 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

Details

Boosting:

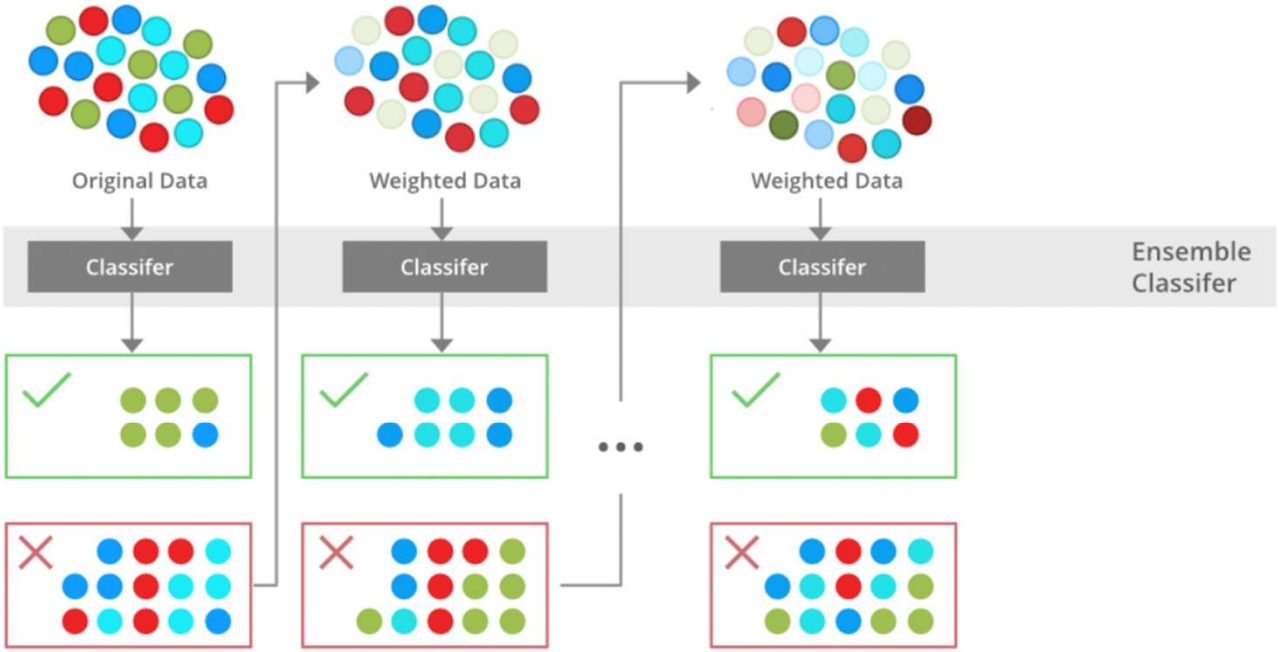
Boosting is an ensemble modeling technique that attempts to build a strong classifier from the number of weak classifiers.

Firstly, a model is built from the training data. Then the second model is built which tries to correct the errors present in the first model. This procedure is continued and models are added until either the complete training data set is predicted correctly or the maximum number of models is added.

Details

Algorithm:

1. *Initialise the dataset and assign equal weight to each of the data point.*
2. *Provide this as input to the model and identify the wrongly classified data points.*
3. *Increase the weight of the wrongly classified data points and decrease the weights of correctly classified data points. And then normalize the weights of all data points.*
4. *if (got required results)*
   Goto step 5
   else
   Goto step 2
5. *End*



An illustration presenting the intuition behind the boosting algorithm, consisting of the parallel learners and weighted dataset.

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

Algorithm 8.2 *Boosting for Regression Trees*

1. Set $\hat{f}(x) = 0$ and $r_i = y_i$ for all i in the training set.
2. For $b = 1, 2, \ldots, B$, repeat:
   (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,

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

3. Output the boosted model,

$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x). \tag{8.12}$$

- 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

Details

9.2.2 Details of the Support Vector Classifier

The support vector classifier classifies a test observation depending on which side of a hyperplane it lies. The hyperplane is chosen to correctly separate most of the training observations into the two classes, but may misclassify a few observations. It is the solution to the optimization problem

$$\underset{\beta_0,\beta_1,\ldots,\beta_p,\epsilon_1,\ldots,\epsilon_n,\,M}{\text{maximize}} \quad M \tag{9.12}$$

$$\text{subject to} \quad \sum_{j=1}^{p} \beta_j^2 = 1, \tag{9.13}$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip}) \geq M(1 - \epsilon_i), \tag{9.14}$$

$$\epsilon_i \geq 0, \quad \sum_{i=1}^{n} \epsilon_i \leq C, \tag{9.15}$$

Details

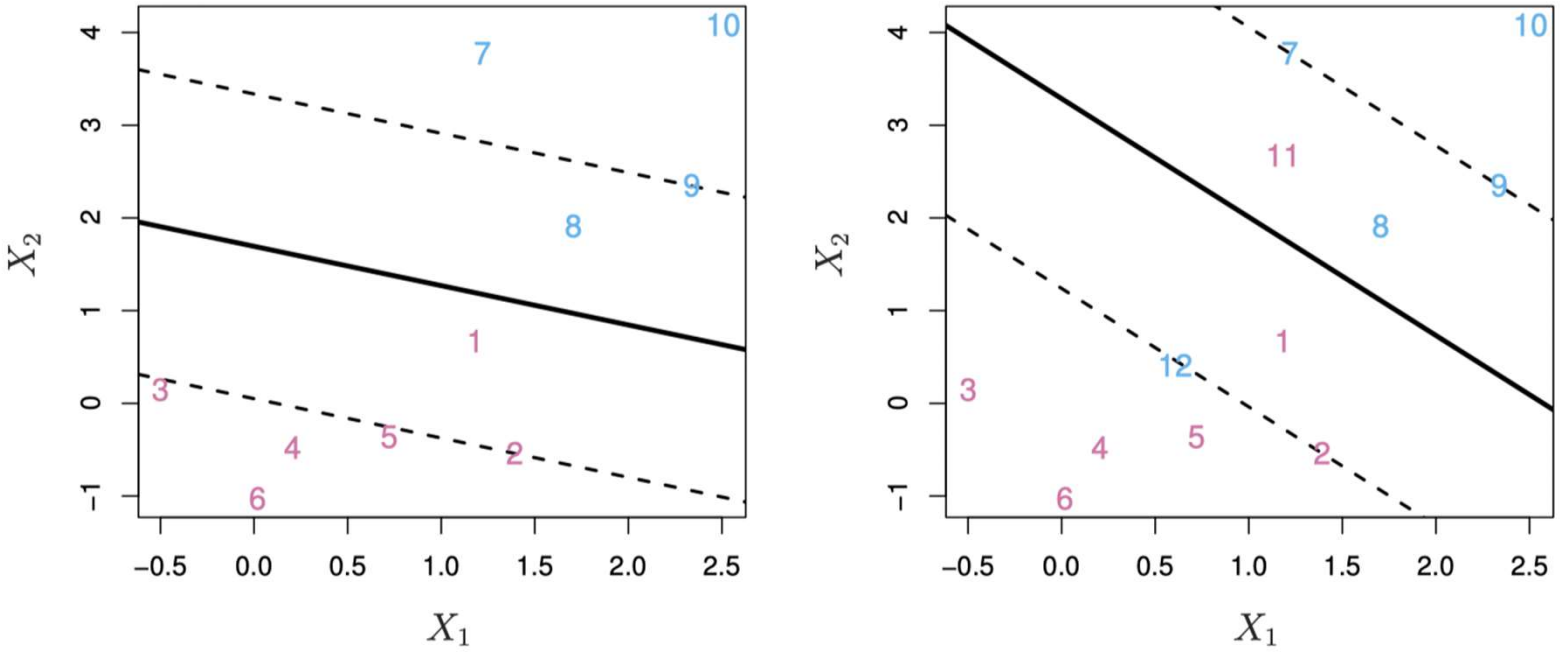


FIGURE 9.6. Left: *A support vector classifier was fit to a small data set. The hyperplane is shown as a solid line and the margins are shown as dashed lines.* Purple observations: *Observations* $3, 4, 5,$ *and* 6 *are on the correct side of the margin, observation* 2 *is on the margin, and observation 1 is on the wrong side of the margin.* Blue observations: *Observations* 7 *and* 10 *are on the correct side of the margin, observation* 9 *is on the margin, and observation* 8 *is on the wrong side of the margin. No observations are on the wrong side of the hyperplane.* Right: *Same as left panel with two additional points,* 11 *and* 12. *These two observations are on the wrong side of the hyperplane and the wrong side of the margin.*

Implementation

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
```

Implementation

```
##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)
```

Implementation

```
##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)
```

Implementation

```
##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)
```