```
# Receiver Operating Characteristic (ROC) curves
# examines the trade-off between the detection of true positives,
# while avoiding the false positives.
# graph between true pos rate (sensitivity) & false pos rate (1-specificity)


dt2<-read.csv(file.choose(), stringsAsFactors = F)
dt<-dt2
dt[c(2,3,6,7,9,13,14)]<-data.frame(lapply(dt[c(2,3,6,7,9,13,14)],factor))
dt <- dt[c(2,3,9,10,11,12,14)]


dummytize <- function(i, exclude=c("")){
  x<-dt[i]
  if(is.factor(dt[[i]]) & !identical(names(x) %in% exclude, TRUE)){
    x<-data.frame(model.matrix(~.-1,data=x))[-1]
  }
  return(x)
}
normalize<-function(x){
  if(is.numeric(x)) x <- (x-min(x))/(max(x)-min(x))
  return (x)
}

dt <- data.frame(lapply(seq_along(dt),dummytize, "statlogheart"))
str(dt)

#Many models assume that the 2nd label is positive event
#By default, Have disease can be 1st label because of chronological order
#  hence we will set it as 2nd label
dt$statlogheart <- factor(dt$statlogheart, levels=c("1", "2"),
                          labels=c("No disease", "Have disease"))


dt<-data.frame(lapply(dt, normalize))
str(dt)


rows <- nrow(dt)
train.index <- sample(1:rows, floor(0.7*rows), replace=FALSE)
dt.train <- dt[train.index,]
dt.test <- dt[-train.index,]

###########knn
library(class)
# install.packages('ROCR')
library(ROCR)
library(gmodels)

#if you have predicted probabilities for the class label, say pred.prob, and you have class
label, how would you calculated predicted probability for positive event.
# eg
# pred.class  pred.prob   pred.pos.prob
#   pos        0.6        0.6
#   neg        0.7        1-0.7 = 0.3
#   neg        0.8        1-0.8 = 0.2
#   pos        0.65       0.65
# pred.pos.prob <- ifelse(pred.class=="pos", pred.prob, 1-pred.prob)

class.knn.test <- knn(dt.train[-9],dt.test[-9],dt.train$statlogheart,k=3, prob=TRUE)
prob.knn.test <- ifelse(class.knn.test=="Have disease", attr(class.knn.test,"prob"), 1-
attr(class.knn.test,"prob"))

#to check if pred.knn.prob is correct we recalculate class labels and compare them with line
pred.knn.class
head (prob.knn.test)
# idx <- which(pred.knn.prob==0.5)
# pred.knn.prob[15]
class.check <- ifelse(prob.knn.test>0.5, "Have disease", "No disease")
table(class.knn.test,class.check)
#since there were no errors, prob calculated above for positive event is correct
CrossTable(dt.test$statlogheart, class.knn.test)
```

```r
# Every evaluation method such as lift, and ROC in ROCR needs a prediction object.
# It transforms the input data into a standardized format needed by ROCR package.
# 1st parameter for prediction is prob of positive event
# 2nd parameter is the actual class label
# 3rd parameter is label ordering
# NOTE: any time you are going to plot ROC curves, don't forget to set the
#  order of class labels of your target variable. Set 1st level as negative event
#  and 2nd level as postive event
predObj.knn.test <- prediction(prob.knn.test,dt.test$statlogheart,
                    label.ordering = c("No disease", "Have disease"))

#lift
lift.knn.test <- performance(predObj.knn.test, measure = "lift", x.measure = "rpp")
plot(lift.knn.test, main="Lift curve for KNN model")

#Cumulative gain is more informative than lift
#cumulative gain
gain.knn.test <- performance(predObj.knn.test, "tpr", "rpp")
plot(gain.knn.test, main="Cumulative gain curve for KNN model")

#roc
roc.knn.test<-performance(predObj.knn.test, "tpr", "fpr")
# add a reference line to the graph
# set the intercept to a = 0 and the slope to b = 1,
# lwd adjusts the line thickness, while the lty adjusts the type of line
# worst classifier
plot(roc.knn.test, main="ROC curve for KNN model", col = "khaki", lwd = 2)
# worst classifier
abline(a = 0, b = 1, lwd = 2, lty = 2)

#Area under the curve
auc.knn.test <- performance(predObj.knn.test, "auc")
#To get AUC, unlist auc.knn.test object, which is a list
unlist(auc.knn.test@y.values)



##############NB
library(e1071)
#NOTE: other models can take target as num, but bays has to take it as factor
model.nb<- naiveBayes(statlogheart~., data=dt.train, laplace=1)
class.nb.test <- predict(model.nb,dt.test)
CrossTable(dt.test$statlogheart, class.nb.test)

prob.nb.test <- predict(model.nb, dt.test, type="raw")
# output of type raw gives probabilities for both levels
head(prob.nb.test)
# since the prob for positive event is in 2nd column, & further analysis uses
#   only positive event probabilies, we need to subset prob.nb.test
prob.nb.test <- prob.nb.test[,2]

class.check <- ifelse(prob.nb.test>0.5, "Have disease", "No disease")
CrossTable(class.nb.test, class.check)
#predicted.nb

#calculate prediction object for various methods in ROCR package
predObj.nb.test <- prediction(prob.nb.test, dt.test$statlogheart,
                    label.ordering = c("No disease", "Have disease"))

#Lift
lift.nb.test <- performance(predObj.nb.test, "lift", "rpp")
plot(lift.nb.test, main="Lift curve for Naive Bayes model")

#Cumulative gain
gain.nb.test <- performance(predObj.nb.test, "tpr", "rpp")
plot(gain.nb.test, main="Cumulative gain curve for Naive Bayes model")

#ROC
roc.nb.test<-performance(predObj.nb.test, "tpr", "fpr")
plot(roc.nb.test, main="ROC curve for Naive Bayes model", col = "navy blue", lwd = 2)
# worst classifier
```

```
abline(a = 0, b = 1, lwd = 2, lty = 2)

#Area under the curve
auc.nb.test <- performance(predObj.nb.test, "auc")
#To get AUC, unlist auc.knn.test object, which is a list
unlist(auc.nb.test@y.values)


##########logit
model.logit <- glm(statlogheart~.,data=dt.train, family = binomial)
summary(model.logit)
#type="link" odds ratio
#type="response" probabilities
prob.logit.test <- predict(model.logit,dt.test, type="response")
# output is probability for positive event
class.logit.test <- ifelse(prob.logit.test>0.5,"Have disease","No disease")
CrossTable(dt.test$statlogheart,class.logit.test)

predObj.logit.test <- prediction(prob.logit.test,dt.test$statlogheart,
                             label.ordering = c("No disease", "Have disease"))

#lift
lift.logit.test <- performance(predObj.logit.test, "lift", "rpp")
plot(lift.logit.test, main="Lift curve for logit model")

#cumulative gain
gain.logit.test <- performance(predObj.logit.test, "tpr", "rpp")
plot(gain.logit.test, main="Cumulative gain curve for logit model")

#roc
roc.logit.test<-performance(predObj.logit.test, "tpr", "fpr")
plot(roc.logit.test, main="ROC curve for logit model", col = "lavender", lwd = 2)
# worst classifier
abline(a = 0, b = 1, lwd = 2, lty = 2)

#Area under the curve
auc.logit.test <- performance(predObj.logit.test, "auc")
#To get AUC, unlist auc.logit.test object, which is a list
unlist(auc.logit.test@y.values)


########decision tree
library(C50)
model.tree <- C5.0(statlogheart ~.,data=dt.train)
summary(model.tree)
class.tree.test <-predict(model.tree, dt.test)
prob.tree.test <- predict(model.tree,dt.test,type="prob")
head(prob.tree.test)
prob.tree.test <- prob.tree.test[,2]
CrossTable(dt.test$statlogheart,class.tree.test)

predObj.tree.test <- prediction(prob.tree.test,dt.test$statlogheart,
                            label.ordering = c("No disease", "Have disease"))

#lift
lift.tree.test <- performance(predObj.tree.test, "lift", "rpp")
plot(lift.tree.test, main="Lift curve for tree model")

#cumulative gain
gain.tree.test <- performance(predObj.tree.test, "tpr", "rpp")
plot(gain.tree.test, main="Cumulative gain curve for tree model")

#roc
roc.tree.test<-performance(predObj.tree.test, "tpr", "fpr")
plot(roc.tree.test, main="ROC curve for tree model", col = "turquoise", lwd = 2)
# worst classifier
abline(a = 0, b = 1, lwd = 2, lty = 2)

#Area under the curve
auc.tree.test <- performance(predObj.tree.test, "auc")
#To get AUC, unlist auc.tree.test object, which is a list
unlist(auc.tree.test@y.values)
```

```
########### SVM #######
library(kernlab)
model.svm <- ksvm(statlogheart ~., data=dt.train, kernel="rbfdot", prob.model=T)
class.svm.test <- predict(model.svm, dt.test)
head(class.svm.test)
prob.svm.test <- predict(model.svm, dt.test, type="probabilities")
head(prob.svm.test)
prob.svm.test <- prob.svm.test[,2]
class.check <- ifelse(prob.svm.test>0.5, "Have disease", "No disease")
table(class.svm.test,class.check)
CrossTable(dt.test$statlogheart,class.svm.test)
class(dt.test$statlogheart)
predObj.svm.test <- ROCR::prediction(prob.svm.test,dt.test$statlogheart,
                              label.ordering = c("No disease", "Have disease"))

#lift
lift.svm.test <- performance(predObj.svm.test, "lift", "rpp")
plot(lift.svm.test, main="Lift curve for svm model")

#cumulative gain
gain.svm.test <- performance(predObj.svm.test, "tpr", "rpp")
plot(gain.svm.test, main="Cumulative gain curve for svm model")

#roc
roc.svm.test<-performance(predObj.svm.test, "tpr", "fpr")
plot(roc.svm.test, main="ROC curve for svm model", col = "salmon", lwd = 2)
# worst classifier
abline(a = 0, b = 1, lwd = 2, lty = 2)

#Area under the curve
auc.svm.test <- performance(predObj.svm.test, "auc")
#To get AUC, unlist auc.svm.test object, which is a list
unlist(auc.svm.test@y.values)

########### Neuralnet #######
library(neuralnet)
model.nn <- neuralnet(statlogheart ~., data=dt.train, hidden = 1)
computed.nn <- compute(model.nn, dt.test)
#NOTE: neuralnet package sorts labels chronologically
head(computed.nn$net.result)
prob.nn.test <- computed.nn$net.result
head(prob.nn.test)
prob.nn.test <- (prob.nn.test[,1])
head(prob.nn.test)

class.nn.test <- ifelse(prob.nn.test>0.5, "Have disease", "No disease")
CrossTable(dt.test$statlogheart,class.nn.test)
summary(prob.nn.test)
predObj.nn.test <- ROCR::prediction(prob.nn.test,dt.test$statlogheart,
                             label.ordering = c("No disease", "Have disease"))

#lift
lift.nn.test <- performance(predObj.nn.test, "lift", "rpp")
plot(lift.nn.test, main="Lift curve for nn model")

#cumulative gain
gain.nn.test <- performance(predObj.nn.test, "tpr", "rpp")
plot(gain.nn.test, main="Cumulative gain curve for nn model")

#roc
roc.nn.test<-performance(predObj.nn.test, "tpr", "fpr")
plot(roc.nn.test, main="ROC curve for nn model", col = "orange", lwd = 2)
# worst classifier
abline(a = 0, b = 1, lwd = 2, lty = 2)

#Area under the curve
auc.nn.test <- performance(predObj.nn.test, "auc")
#To get AUC, unlist auc.nn.test object, which is a list
unlist(auc.nn.test@y.values)
```

```
##ROC curves on same plot
plot(roc.nb.test, xlab="False Positive Rate", ylab="True Positive Rate",
    type='l', col = "navy blue")
plot(roc.tree.test, col = "turquoise", add=T)
plot(roc.knn.test, col = "khaki", add=T)
plot(roc.logit.test, col = "lavender", add=T)
plot(roc.svm.test, col = "salmon", add=T)
plot(roc.nn.test, col = "green", add=T)
abline(v=0,lty=2, lwd=1, col="gray60")
abline(h=1,lty=2, lwd=1, col="gray60")
abline(a=0,b=1,lty=2, lwd=1, col="gray60")


##ROC curves on same plot using ggplot2()
library(ggplot2)
ggplot()+
  geom_line(data=data.frame(x=roc.tree.test@x.values[[1]], y=roc.tree.test@y.values[[1]]),
          aes(x=x, y=y, col="red")) +
  geom_line(data=data.frame(x=roc.svm.test@x.values[[1]], y=roc.svm.test@y.values[[1]]),
          aes(x=x, y=y, col="blue"))



##ROC curves on same plot
plot(xlab="False Positive Rate", ylab="True Positive Rate",
    roc.nb.test @x.values[[1]], roc.nb.test @y.values[[1]],type='l', col = "navy blue")
lines(roc.tree.test@x.values[[1]], roc.tree.test@y.values[[1]], col = "turquoise")
lines(roc.knn.test@x.values[[1]], roc.knn.test@y.values[[1]], col = "khaki")
lines(roc.logit.test@x.values[[1]], roc.logit.test@y.values[[1]], col = "lavender")
lines(roc.svm.test@x.values[[1]], roc.svm.test@y.values[[1]], col = "salmon")
lines(roc.nn.test@x.values[[1]], roc.nn.test@y.values[[1]], col = "green")
abline(v=0,lty=2, lwd=1, col="gray60")
abline(h=1,lty=2, lwd=1, col="gray60")
abline(a=0,b=1,lty=2, lwd=1, col="gray60")




lift.knn.test@y.values[[1]] <- ifelse(lift.knn.test@y.values[[1]]=="NaN",
                                      0,lift.knn.test@y.values[[1]])
```

```
################################Chapter 10

## Confusion matrixes in R ----
# use file sms_results.csv
dt<- read.csv(file.choose(),stringsAsFactors = T)

# the first several test cases
head(dt)

# test cases where the model is less confident say between 0.4 and 0.6
dt[dt$prob_spam >0.4 & dt$prob_spam<0.6,]
head(subset(dt, prob_spam > 0.40 & prob_spam < 0.60))

# test cases where the model was wrong
head(subset(dt, actual_type != predict_type))

# tablulate results
table(dt$actual_type, dt$predict_type)
#using CrossTable
library(gmodels)
CrossTable(dt$actual_type, dt$predict_type)

# # alternative solution using the formula interface (not shown in book)
# xtabs(~ actual_type + predict_type, sms_results)

# using the CrossTable function

# accuracy and error rate calculation --
# accuracy = TP+TN/Total
(152 + 1203) / (152 + 1203 + 4 + 31)
# error rate = FP+FN/total
(4 + 31) / (152 + 1203 + 4 + 31)
# error rate = 1 - accuracy
1 - 0.9748201

## Beyond accuracy: other performance measures ----
install.packages('caret')
library(caret)
confusionMatrix(sms_results$predict_type, sms_results$actual_type, positive = "spam")
CrossTable(sms_results$predict_type, sms_results$actual_type)
# Kappa statistic adjusts accuracy by accounting for the possibility that
# correct prediction can happen by chance alone
# pr of accuracy = TP/total + TN/total
pr_a <- 0.865 + 0.109
pr_a

#pr of being correct by chance alone = PP*AP/total^2 + PN*AN/total^2
pr_e <- 0.868 * 0.888 + 0.132 * 0.112
pr_e
#k= Pr(a)-Pr(e)/1-Pr(e)
k <- (pr_a - pr_e) / (1 - pr_e)
k

# # calculate kappa via the vcd package
# install.packages('vcd')
# library(vcd)
# Kappa(table(sms_results$actual_type, sms_results$predict_type))

# calculate kappa via the irr package
install.packages('irr')
library(irr)
kappa2(sms_results[1:2])$value


# Sensitivity
# Sensitivity proportion of positive examples that were correctly classified
# Sensitivity (true positive rate) = TP/TP+FN
sens <- 152 / (152 + 31)
sens
```

```
# Specificity
# measures the proportion of negative examples that were correctly classified.
# Specificity (true negative rate) = TN/TN+FP
spec <- 1203 / (1203 + 4)
spec

# example using the caret package
library(caret)
sensitivity(sms_results$predict_type, sms_results$actual_type, positive = "spam")
specificity(sms_results$predict_type, sms_results$actual_type, negative = "ham")

# Precision (positive predictive value) = TP/TP+FP
# is the proportion of positive examples that are truly positive;
# Google search results
prec <- 152 / (152 + 4)
prec

# Recall = Sensitivity = TP/TP+FN
# is a measure of how complete the results are.
rec <- 152 / (152 + 31)
rec

# example using the caret package
library(caret)
posPredValue(sms_results$predict_type, sms_results$actual_type, positive = "spam")
sensitivity(sms_results$predict_type, sms_results$actual_type, positive = "spam")

# F-measure = 2*Pre*Rec/Pre+Rec Harmonic mean of Pre & Rec
# harmonic mean is used rather than the common arithmetic mean
# since both precision and recall are expressed as proportions between zero and one,
# which can be interpreted as rates.
# eg average rate of a bucket getting filled when there are 2 taps
f <- (2 * prec * rec) / (prec + rec)
f

f <- (2 * 152) / (2 * 152 + 4 + 31)
f

## Visualizing Performance Tradeoffs ----
install.packages('ROCR')
library(ROCR)
pred <- prediction(predictions = sms_results$prob_spam,
labels = sms_results$actual_type)

# Receiver Operating Characteristic (ROC) curves
# examines the trade-off between the detection of true positives,
# while avoiding the false positives.
# graph between true pos rate (sensitivity) & false pos rate (1-specificity)
perf <- performance(pred, measure = "tpr", x.measure = "fpr")
plot(perf, main = "ROC curve for SMS spam filter", col = "blue", lwd = 2)

# add a reference line to the graph
# set the intercept to a = 0 and the slope to b = 1,
# lwd adjusts the line thickness, while the lty adjusts the type of line
# worst classifier
abline(a = 0, b = 1, lwd = 2, lty = 2)
# perfect classifier
abline(a = 1, b = 0, lwd = 2, lty = 2, col='red')
abline(v=0, lwd=2, lty=2, col='red')

# calculate AUC
# The closer the curve is to the perfect classifier,
# the better it is at identifying positive values
perf.auc <- performance(pred, measure = "auc")
# perf.auc is an R object (specifically known as an S4 object),
# use a special type of notation to access the values stored within.
# S4 objects hold information in positions known as slots.
# str() function can be used to see all of an object's slots:
  str(perf.auc)
# to see value in specific slot we can use @
# further to get value and not as list you can either use unlist or [[]]
auc <- perf.auc@y.values[[1]]
```

```
auc
unlist(perf.auc@y.values)

## Estimating Future Performance ----

# partitioning data
library(caret)
credit <- read.csv("credit.csv")

# Holdout method-- so far into training and testing
# using test for selecting model does not make it
# an unbiased measure of the performance on unseen data
rows <- nrow(credit)
set.seed(7)
train.size <- floor(rows*0.6)
val.size <- floor(rows*0.2)

train.vec <- sample(rows, train.size)
# get validation vector: from all rows index remove train.vec
val.vec <- sample (data.frame(1:rows)[-train.vec,], val.size)
train.set <- credit[train.vec,]
val.set <- credit[val.vec,]
test.set <-credit[-c(train.vec,val.vec),]

#other way of splitting
random_ids <- order(runif(rows))
credit_train <- credit[random_ids[1:train.size],]
credit_validate <- credit[random_ids[(train.size+1):(train.size+val.size)], ]
credit_test <- credit[random_ids[(train.size+val.size+1):1000], ]

# # using caret function
# in_train <- createDataPartition(credit$default, p = 0.75, list = FALSE)
# credit_train <- credit[in_train, ]
# credit_test <- credit[-in_train, ]

# Stratified sampling: each partition may have a larger or smaller proportion of some classes

# Repeated holdout method: k-fold CV
# 10-fold CV

folds <- createFolds(credit$default, k = 10)
str(folds)
credit01_train <- credit[-folds$Fold01, ]
credit01_test <- credit[folds$Fold01, ]

## Automating 10-fold CV for a C5.0 Decision Tree using lapply() ----
library(caret)
library(C50)
library(irr)

credit <- read.csv("credit.csv")

set.seed(123)
folds <- createFolds(credit$default, k = 10)

#let us try to run 10-fold CV on decision tree
library(C50)
#if you have to do this manually then for the first iteration following is what you have to do
train.set<-dt[-folds$Fold01,]
test.set <-dt[folds$Fold01,]

foldAnalysis <- function(x){
  train.set <- dt[-x,]
  test.set <- dt[x,]
  model<- C5.0(default~., data=train.set)
  predicted<-predict(model,test.set)
  kappa<-kappa2(data.frame(test.set$default,predicted))$value
  return (kappa)
}
allFolds<-unlist(lapply(folds,foldAnalysis))
allFolds
#following gives you mean kappa using 10-fold cross validation
```

```
mean(allFolds)



##### CHAPTER 11 #########

#Extract credit.csv from the class data
dt <- read.csv(file.choose())
str(dt)
## Classification and Regression Training (CART) ##
install.packages("caret")
library(caret)
modelLookup("knn")
modelLookup("C5.0")

model <- train(default ~. , data=dt, method="C5.0")
model


#instead of it using boostrap sampling by default, we want caret to use
# 10-fold cross validation and the selection function should not be the
# model with best performance over a metric, rather a model that is within
# one standard error of the best performing model and less complex
trCtrl <- trainControl(method="cv", n=10, selectionFunction = "oneSE")
# instead of relying on caret to pick random 3 values of parameters, we
# specified values it should try for running models.
pRange <- expand.grid(.model="tree", .winnow=F, .trials=c(1,5,10,15,30,50))
model <- train(default ~., data=dt, method="C5.0",
          metric= "Kappa",
          trCtrl=trCtrl,
          tuneGrid=pRange)
model

predicted <- predict(model, dt.test)

## Ensembles combine various weak learners to create a strong learner

# Weak learner is a model that has error rate slightly less than 0.5

## Bootstrap aggregation (Bagging)
# You give equal weight to the prediction from various models

## Adaboost (Adaptive boosting)
#  Steps:
# 1) In the first iteration all records get equal weighted. Based on training
# set you will estimate a model, and make predictions for the entire data,
# and calculate errors;
# 2) Errors that are bigger will get higher weight in the next iteration;
# You will run say 20 models, and calculate misclassification for each of
# the 20 models. But the weigth while calcualting misclassification the
# weight will not be the same across instances. So you will pick a model
# that gives more importance to records that have higher error in them.


### REGULARIZATION
In regression models, in order to reduce variance we introduce some bias in the model.
Instead of minimizing just Sum of square of errors, we also add a penlty on coefficients.
#Ridge regression
# Specificially, in case of Ridgre regression, we add L2 norm of the coeffiencts of features.
# L2 norm is sum of square of elements of a vector
# Objective: Min SSE + lambda * sum of square of coefficients
# This is equivalent to adding the constraint of a circle in the case of 2 features;
# Hence, the coefficients will never be zero.
# They may be very small.
# If lambda is high then you reduce the risk of overfitting;

#Lasso regression
# we add L1 norm of the coefficients of features;
# L1 norm is sum of absolute value of elements of a vector;
# Objective: Min SSE + lambda * sum of absolute value of coefficients
# This is equivalent to adding the constraint of a rhombus in the case of 2 features.
# Hence, the coefficients can be zero, and this can help in feature selection.
# Elastic net regression
# we add linear combination of L2 and L1 norm of the coefficients of features;
#  objective: Min SSE + lambda * ((1-alpha)*sum of square of coefficients + alpha* abs value
of coefficients)
# if alpha =1 => Lasso regression
# if alpha =0 => Ridge regression
#  0 <= alpha <= 1

# XGBoost is one of the most prominent machine learning technique and has been a top
choice by people winning
# Kaggle competitions. It is pretty versatile and can be used for both classification, numeric
prediction and ranking
# It improves GBM by 10X.


## ******After this, you may not worry about it for the exam****
## Good balanced reading: https://www.datacamp.com/community/tutorials/tutorial-
ridge-lasso-elastic-net
## Good reading for Ridge, Lasso and Elasticnet R code is
https://web.stanford.edu/~hastie/glmnet/glmnet_alpha.html
## Good reading for maths behind Ridge, Lasso and Elasticnet is
http://statweb.stanford.edu/~tibs/sta305files/Rudyregularization.pdf
## For graphical explanation see this:
https://en.wikipedia.org/wiki/Regularization_(mathematics)#Regularizers_for_sparsity

#   Gradient Boosting Machine (GBM) is like adaptive boosting
#   it can be used for classification, numeric prediction / regression, ranking data
# GBM: Gradient descent is partial of Loss function wrt to predicted target;
# Friedman (1999) showed that in case of MSE (L2 loss), gradient is equal to the mean of
residuals,
#    while in MAE (L1 loss) the gradient is the sign of residuals;
# Fm(x) = Fm-1(x) + rho*hm(x); where rho is the learning rate;
# where hm(x) = y - Fm-1(x)
#
# Steps:
#   1) Take the median (mean) value of target for the MAE (MSE) loss. F0(x) = median (mean)
of X; Calculate residuals;
#   2) Take the sign (values) of residuals as target, and find a split that minimizes the
variance in
#       sign (actual) values of residuals. Residual values is called direction vector, which means
both sign & value;
#   3) Take the prediction in each leaf as the median (mean) residual. The sign vector is used
for grouping/splitting
#       purposes in L1 loss, but the actual prediction is in fact a residual, just like it is for the L2
loss algorithm.
#   4) Multiply this predicted error with learning rate, and add it into summed predicted
value;
#   5) Find the new residual by subtracting summed predicted value from the actual target;
Repeat step 2


## XGBoost is an ensemble learning method;
#  It implements parallel processing, and is hence fast (typically 10X faster than GBM)
#  it implements regularisation helping reduce overfit
#  it generally has more predictive power than traditonal Gradient boosting method (GBM)
or Adaboost.
# While Gradient Boosting uses greedy approach to try various models, XGBoost uses Taylor
expansion till 2nd order
# to calculate the optimum weight for tree leaves.
#  XGBoost doesn't explore all possible tree structures but builds a tree greedily.


## CatBoost is even more efficient version of XGBoost. It is open sourced by Yandex;


# A model hyperparameter is a configuration that is external to the model and whose value
cannot be estimated from data.
# They are often used in processes to help estimate model parameters.
# They are often specified by the practitioner.
# They can often be set using heuristics.
# They are often tuned for a given predictive modeling problem.
```