Ensemble methods or classifier combination methods

LZ

May 5, 2019

- An ensemble method constructs a set of base classifiers from training data and performs classification by taking a vote on the predictions made by each base classifier

1. By manipulating the training set (Bagging, boosting)
2. By manipulating the input features. (Ranfom Forest)
3. By manipulating the class labels.
4. By manipulating the learning algorithm.

Create Multiple DataSets -> Build Multiple Classifiers -> Combine Classifiers

The Bootstrap

- Repeatedly sampling observations from the original data set
- The sampling is performed with replacement

```
library(boot)
library(ISLR) # for Portfolio dataset
pt <- Portfolio
str(pt)
```

```
## 'data.frame':    100 obs. of  2 variables:
##  $ X: num  -0.895 -1.562 -0.417 1.044 -0.316 ...
##  $ Y: num  -0.235 -0.885 0.272 -0.734 0.842 ...
```

A simple simulated data set containing 100 returns for each of two assets, X and Y. The data is used to estimate the optimal fraction to invest in each asset to minimize investment risk of the combined portfolio.

```
alpha.fn = function (data, index){
  X=data$X[index]
  Y=data$Y[index]
  return ((var(Y)-cov(X,Y))/(var(X)+var(Y) -2*cov(X,Y))) }
```

Index - vector indicating which observations should be used to estimate apha. For instance, the following command tells R to estimate alpha using all 100 observations:

```
alpha.fn(pt ,1:100)
```

```
## [1] 0.5758321
```

To randomly select 100 observations from the range 1 to 100, with replacement we will use the function sample()

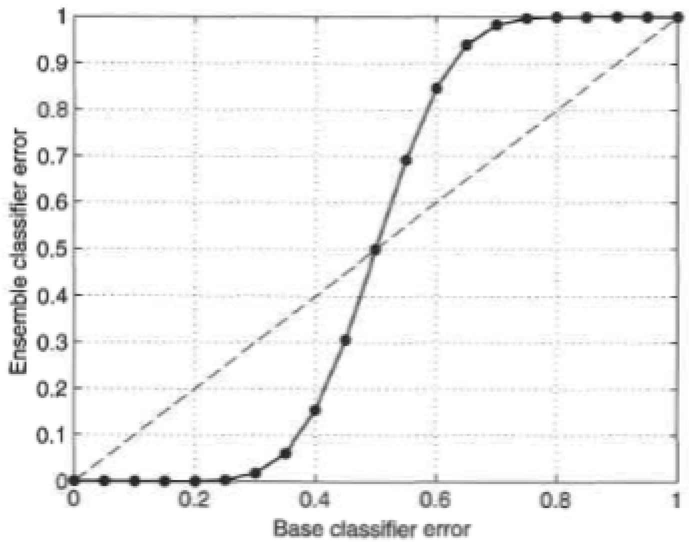


Figure 1:

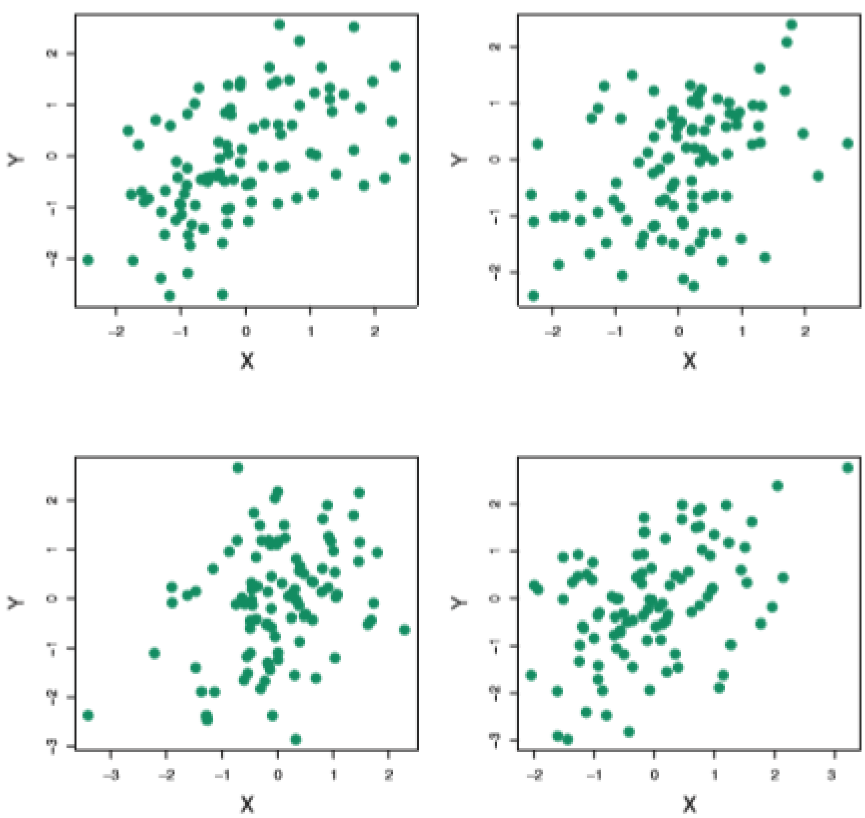


Figure 2:

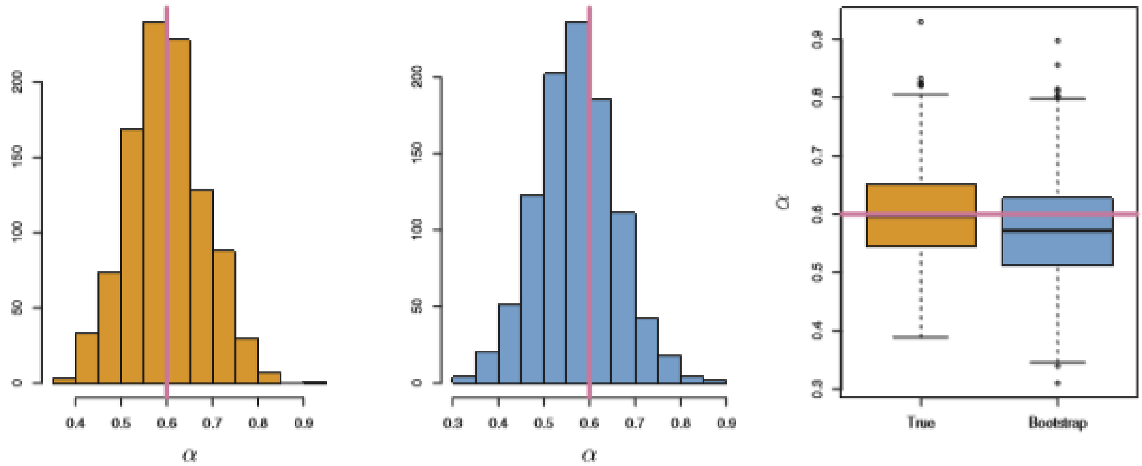


Figure 3:

$$\hat{\alpha} = \frac{\hat{\sigma}_Y^2 - \hat{\sigma}_{XY}}{\hat{\sigma}_X^2 + \hat{\sigma}_Y^2 - 2\hat{\sigma}_{XY}}.$$

Figure 4:

```
set.seed(1)
alpha.fn(pt ,sample (100, 100, replace=T))
```

```
## [1] 0.5963833
```

We can implement a bootstrap analysis by performing this command many times, recording all of the corresponding estimates for ??, and computingthe resulting standard deviation. However, the boot() function automates boot() this approach.

```
boot(data = pt, statistic =  alpha.fn, R = 1000 )
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = pt, statistic = alpha.fn, R = 1000)
##
##
## Bootstrap Statistics :
##      original        bias    std. error
## t1* 0.5758321 -7.315422e-05  0.08861826
```

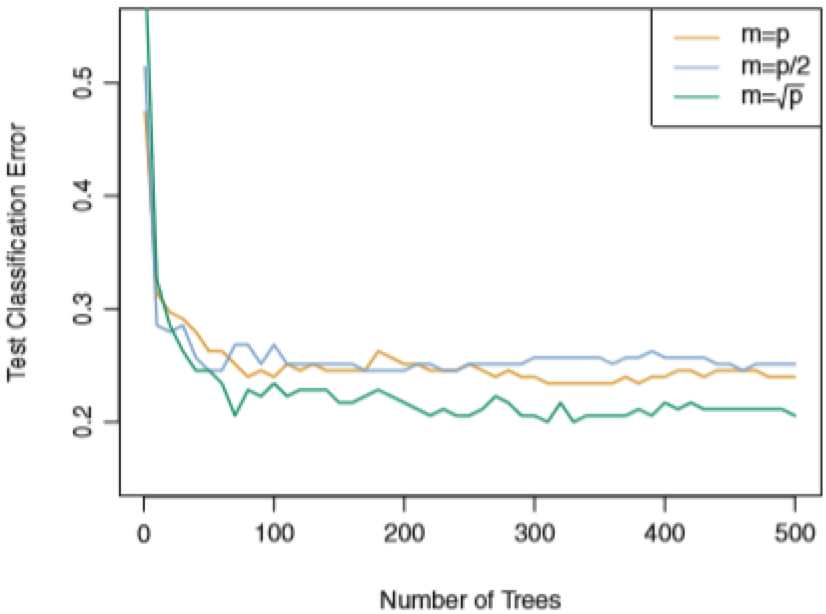


Figure 5:

```
# R number of bootstrap replicates
```

Using the original data alpha = 0.5758.

The bagging, boosting abd Random forests

Bagging

- The decision trees discussed suffer from high variance.
- Averaging a set of observations reduces variance.

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

Random forests

- We build a number of decision trees on bootstrapped training samples
- Each time a split in a tree is considered, a random sample of m predictors is chosen as split candidates from the full set of p predictors.
- Averaging many highly correlated quantities does not lead to as large of a reduction in variance as averaging many uncorrelated quantities.

```
credit<-read.csv("credit.csv")
library(caret)
index<-createDataPartition(credit$default, p=0.8, list=F)
Train<-credit[index,]
Test<-credit[-index,]
str(Train)
```

```
## 'data.frame':    561 obs. of  9 variables:
##  $ age     : int  41 27 40 41 24 41 39 43 24 36 ...
##  $ ed      : Factor w/ 5 levels "college degree",..: 1 3 3 3 2 2 3 3 3 3 ...
##  $ employ  : int  17 10 15 15 2 5 20 12 3 0 ...
##  $ address : int  12 6 14 14 0 5 9 11 4 13 ...
##  $ income  : int  176 31 55 120 28 25 67 38 19 25 ...
##  $ debtinc : num  9.3 17.3 5.5 2.9 17.3 10.2 30.6 3.6 24.4 19.7 ...
##  $ creddebt: num  11.359 1.362 0.856 2.659 1.787 ...
##  $ othdebt : num  5.009 4.001 2.169 0.821 3.057 ...
##  $ default : Factor w/ 2 levels "No","Yes": 2 1 1 1 2 1 1 1 2 1 ...
```

Checking test error for base classifier

```r
library(rpart)
model_c <- rpart(formula = default~.,
  data = Train,
  method = "class"
  )
pred_class<-predict(model_c, Test, type="class")
pred_class[1:20]
```

```
##  18  24  26  28  31  39  43  49  51  63  68  77  83  85  86  87  88  89
##  No  No Yes  No  No Yes  No Yes  No Yes  No  No Yes Yes  No Yes  No  No
##  94  96
## Yes  No
## Levels: No Yes
```

```r
confusionMatrix(pred_class, Test$default, positive="Yes")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction No Yes
##        No  90  17
##        Yes 13  19
##
##                Accuracy : 0.7842
##                  95% CI : (0.7065, 0.8494)
##     No Information Rate : 0.741
##     P-Value [Acc > NIR] : 0.1429
##
##                   Kappa : 0.4166
##  Mcnemar's Test P-Value : 0.5839
##
##             Sensitivity : 0.5278
##             Specificity : 0.8738
##          Pos Pred Value : 0.5937
##          Neg Pred Value : 0.8411
##              Prevalence : 0.2590
##          Detection Rate : 0.1367
##    Detection Prevalence : 0.2302
##       Balanced Accuracy : 0.7008
```

```
##
##        'Positive' Class : Yes
##
```

```r
mean(pred_class==Test$default)
```

```
## [1] 0.7841727
```

```r
1-mean(pred_class==Test$default)
```

```
## [1] 0.2158273
```

```r
library(randomForest)
set.seed(2708)

model_f1 <- randomForest(default~.,
  data=Train,
  ntree=25 #how many trees do you want to build
  )
#mtry - Number of variables randomly sampled as candidates at each split.
# default value is sqrt(p)
```

```r
model_f1
```

```
##
## Call:
##  randomForest(formula = default ~ ., data = Train, ntree = 25)
##                Type of random forest: classification
##                      Number of trees: 25
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 23.89%
## Confusion matrix:
##      No Yes class.error
## No  364  50   0.1207729
## Yes  84  63   0.5714286
```

```r
45/(369+45)
```

```
## [1] 0.1086957
```

```r
86/(86+61)
```

```
## [1] 0.585034
```

```r
set.seed(2708)

model_f_ <- randomForest(default~.,
  data=Train,
  ntree=25,
  do.trace=T
  )
```

```
## ntree      OOB      1      2
##     1:  27.17% 16.06% 59.57%
##     2:  28.88% 18.88% 60.00%
##     3:  29.23% 18.71% 60.58%
##     4:  26.45% 15.47% 59.48%
##     5:  26.37% 16.30% 56.00%
##     6:  26.01% 16.02% 55.30%
##     7:  24.25% 14.50% 52.94%
##     8:  22.32% 12.96% 49.30%
##     9:  24.01% 14.15% 52.08%
##    10:  23.26% 13.59% 50.34%
##    11:  24.24% 13.77% 53.74%
##    12:  24.06% 14.01% 52.38%
##    13:  23.17% 12.32% 53.74%
##    14:  24.96% 13.53% 57.14%
##    15:  23.89% 11.11% 59.86%
##    16:  23.17% 11.35% 56.46%
##    17:  23.71% 11.84% 57.14%
##    18:  23.89% 12.56% 55.78%
##    19:  22.82% 11.84% 53.74%
##    20:  24.06% 12.56% 56.46%
##    21:  23.35% 12.32% 54.42%
##    22:  23.71% 12.80% 54.42%
##    23:  24.24% 12.56% 57.14%
##    24:  24.42% 12.32% 58.50%
##    25:  23.89% 12.08% 57.14%
```

- Columns 1 and 2 in the output give the classification error for each class.
- The OOB value is the weighted average of the class errors (weighted by the fraction of observations in each class).

Increase in the number of trees

```r
model_f2 <- randomForest(default~.,
  data=Train,
  ntree=50)
model_f2
```

```
##
## Call:
##  randomForest(formula = default ~ ., data = Train, ntree = 50)
##                Type of random forest: classification
##                      Number of trees: 50
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 22.64%
## Confusion matrix:
##      No Yes class.error
## No  375  39   0.0942029
## Yes  88  59   0.5986395
```

```r
model_f3 <- randomForest(default~.,
  data=Train,
  ntree=100)
model_f3
```

```
##
## Call:
##  randomForest(formula = default ~ ., data = Train, ntree = 100)
##                Type of random forest: classification
##                      Number of trees: 100
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 22.1%
## Confusion matrix:
##      No Yes class.error
## No  376  38  0.09178744
## Yes  86  61  0.58503401
```

```r
model_f4 <- randomForest(default~.,
  data=Train,
  ntree=125)
model_f4
```

```
##
## Call:
##  randomForest(formula = default ~ ., data = Train, ntree = 125)
##                Type of random forest: classification
##                      Number of trees: 125
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 21.03%
## Confusion matrix:
##      No Yes class.error
## No  380  34   0.0821256
## Yes  84  63   0.5714286
```

```r
dim(Train)
```

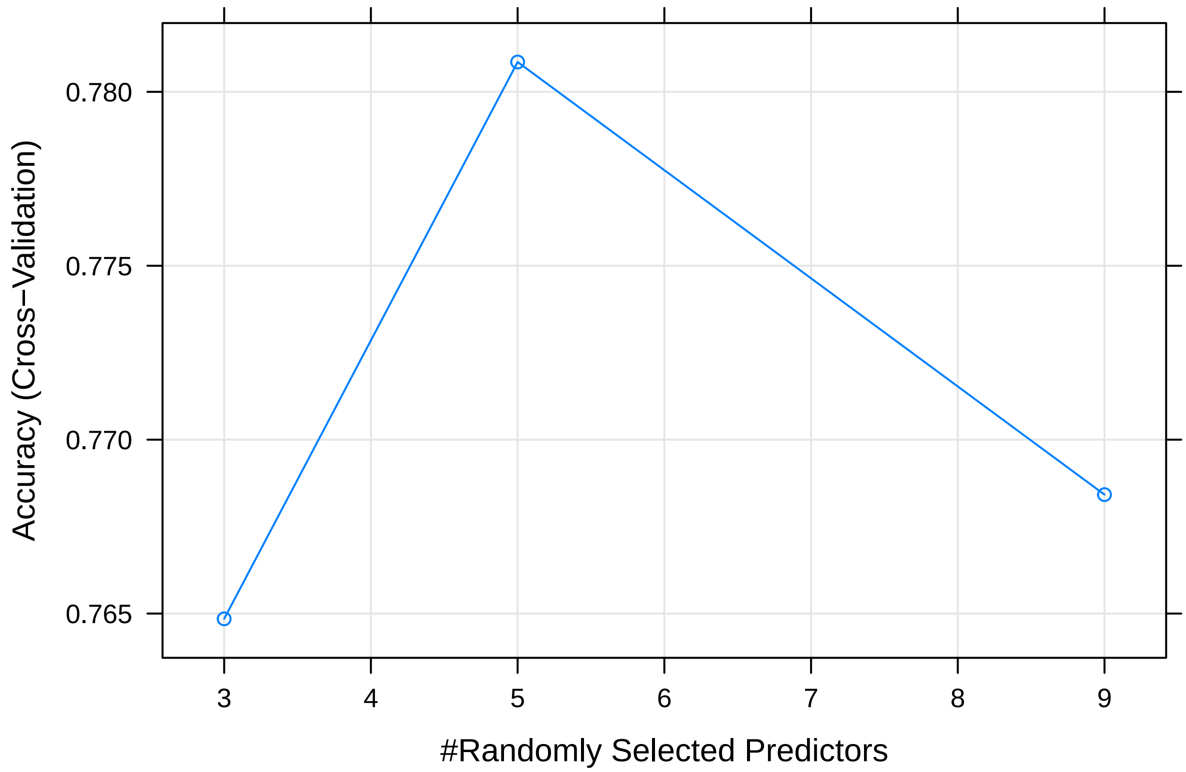
```
## [1] 561   9
```

We will do grid search for variable mtry

```r
set.seed(1)
trc <- trainControl(method="cv", number=5)
mtry_grid <- expand.grid(mtry=c(3,5,9))
set.seed(1)
modelbest <- train(default~., data=Train,
            trControl=trc,
            method="rf",
            ntree=100,
            tuneGrid=mtry_grid)
modelbest$results
```

```
##   mtry  Accuracy     Kappa AccuracySD   KappaSD
## 1    3 0.7648486 0.3114754 0.04980584 0.1277327
## 2    5 0.7808569 0.3696503 0.04831331 0.1444736
## 3    9 0.7684201 0.3373514 0.04839353 0.1457734
```

```r
plot(modelbest)
```




```r
modelbest$bestTune
```

```
##   mtry
## 2    5
```

```r
model_f3best <- randomForest(default~.,
  data=Train,
  ntree=100,
  mtry=5)
model_f3best
```

```
##
## Call:
##  randomForest(formula = default ~ ., data = Train, ntree = 100,      mtry = 5)
##                Type of random forest: classification
##                      Number of trees: 100
## No. of variables tried at each split: 5
```

```
##
##         OOB estimate of  error rate: 22.82%
## Confusion matrix:
##      No Yes class.error
## No  371  43   0.1038647
## Yes  85  62   0.5782313
```

```
pr <- predict(model_f3best, newdata = Test, type = "prob")
pr[1:10,]
```

```
##      No  Yes
## 18 0.99 0.01
## 24 0.98 0.02
## 26 0.14 0.86
## 28 0.84 0.16
## 31 0.83 0.17
## 39 0.37 0.63
## 43 0.87 0.13
## 49 0.22 0.78
## 51 0.82 0.18
## 63 0.70 0.30
```

```
pr.class <- predict(model_f3best, newdata = Test, type = "class")
pr.class[1:10]
```

```
##  18  24  26  28  31  39  43  49  51  63
##  No  No Yes  No  No Yes  No Yes  No  No
## Levels: No Yes
```

Comparing the results of DT and RF

```
confusionMatrix(pred_class, Test$default, positive="Yes")#DT
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction No Yes
##        No  90  17
##        Yes 13  19
##
##                Accuracy : 0.7842
##                  95% CI : (0.7065, 0.8494)
##     No Information Rate : 0.741
##     P-Value [Acc > NIR] : 0.1429
##
##                   Kappa : 0.4166
##  Mcnemar's Test P-Value : 0.5839
##
##             Sensitivity : 0.5278
##             Specificity : 0.8738
##          Pos Pred Value : 0.5937
##          Neg Pred Value : 0.8411
```

```
##               Prevalence : 0.2590
##           Detection Rate : 0.1367
##     Detection Prevalence : 0.2302
##        Balanced Accuracy : 0.7008
##
##         'Positive' Class : Yes
##
```

```r
confusionMatrix(pr.class, Test$default, positive="Yes")#RF
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction No Yes
##        No  93  19
##        Yes 10  17
##
##                Accuracy : 0.7914
##                  95% CI : (0.7143, 0.8556)
##     No Information Rate : 0.741
##     P-Value [Acc > NIR] : 0.1022
##
##                   Kappa : 0.4083
##  Mcnemar's Test P-Value : 0.1374
##
##             Sensitivity : 0.4722
##             Specificity : 0.9029
##          Pos Pred Value : 0.6296
##          Neg Pred Value : 0.8304
##              Prevalence : 0.2590
##          Detection Rate : 0.1223
##    Detection Prevalence : 0.1942
##       Balanced Accuracy : 0.6876
##
##        'Positive' Class : Yes
##
```

Boosting

- Boosting works in a similar way, except that the trees are grown sequentially.
- Each tree is fit on a modified version of the original data set.