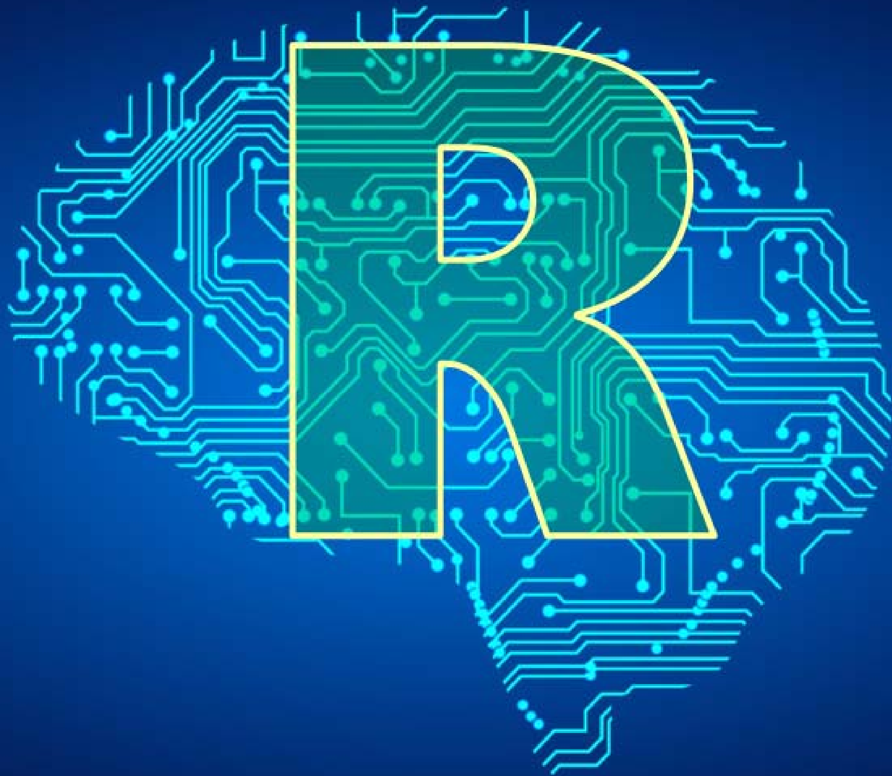


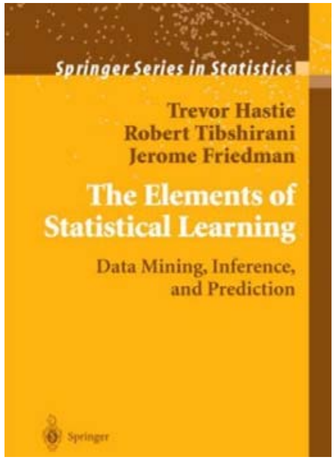
重抽法則
(整合學習)

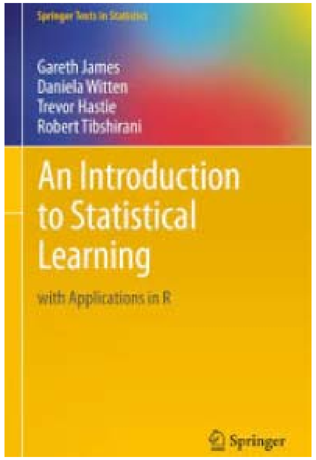
吳漢銘
國立臺北大學 統計學系

本章大綱

- Training data and Testing data
- Resampling methods
  - Jackknife (leave-one-out)
  - Bootstrapping
- Ensemble Learning
  - bagging
  - boosting
- Imbalanced Data Problem
  - under-sampling
  - over-sampling

Common Machine Learning Algorithms
Linear Regression, Logistic Regression, Decision Tree, SVM, Naive Bayes, KNN, K-Means, Random Forest, Dimensionality Reduction, Boosting

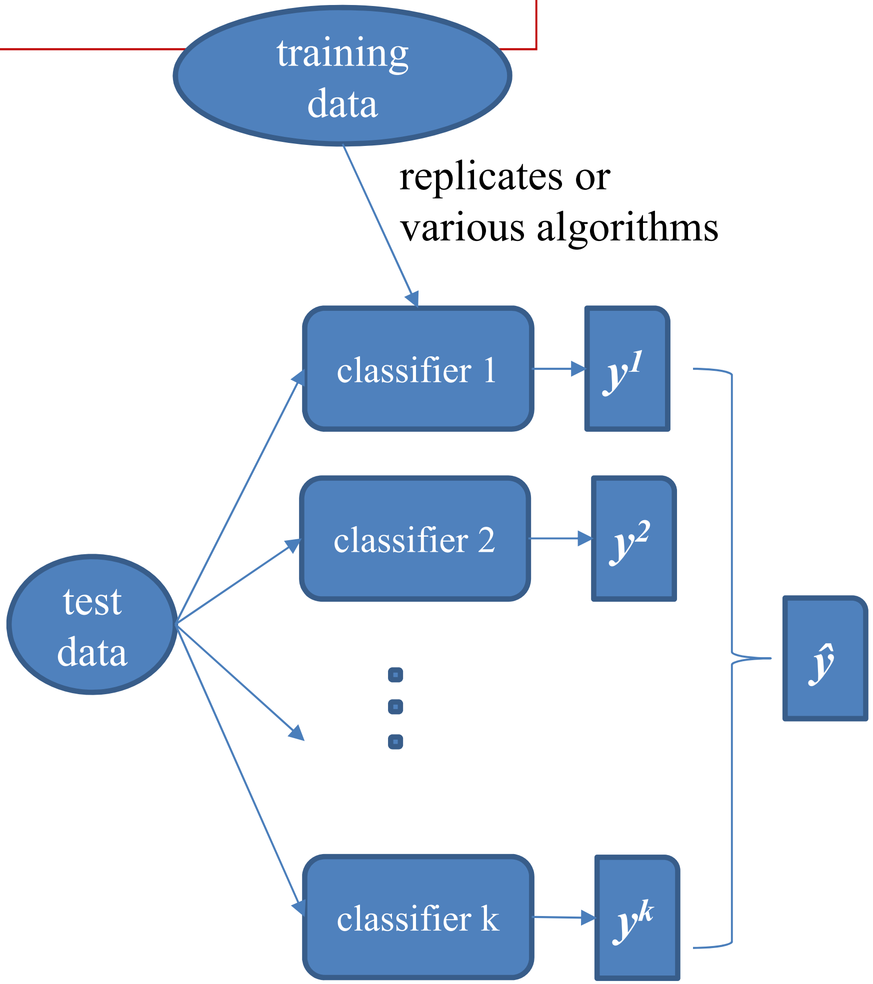




Why Ensemble Learning?

```
prediction.accuracy.rate <- function(no.classifier=1, accuracy.rate=0.5){
   c(no.classifiers=no.classifier,
     at.least.one.accuracy=1-(1-accuracy.rate)^no.classifier)
}
```

```
> prediction.accuracy.rate()
       no.classifiers at.least.one.accuracy
                  1.0                   0.5
> t(sapply(1:10, prediction.accuracy.rate))
      no.classifiers at.least.one.accuracy
 [1,]              1             0.5000000
 [2,]              2             0.7500000
 [3,]              3             0.8750000
 [4,]              4             0.9375000
 [5,]              5             0.9687500
 [6,]              6             0.9843750
 [7,]              7             0.9921875
 [8,]              8             0.9960938
 [9,]              9             0.9980469
[10,]             10             0.9990234
```



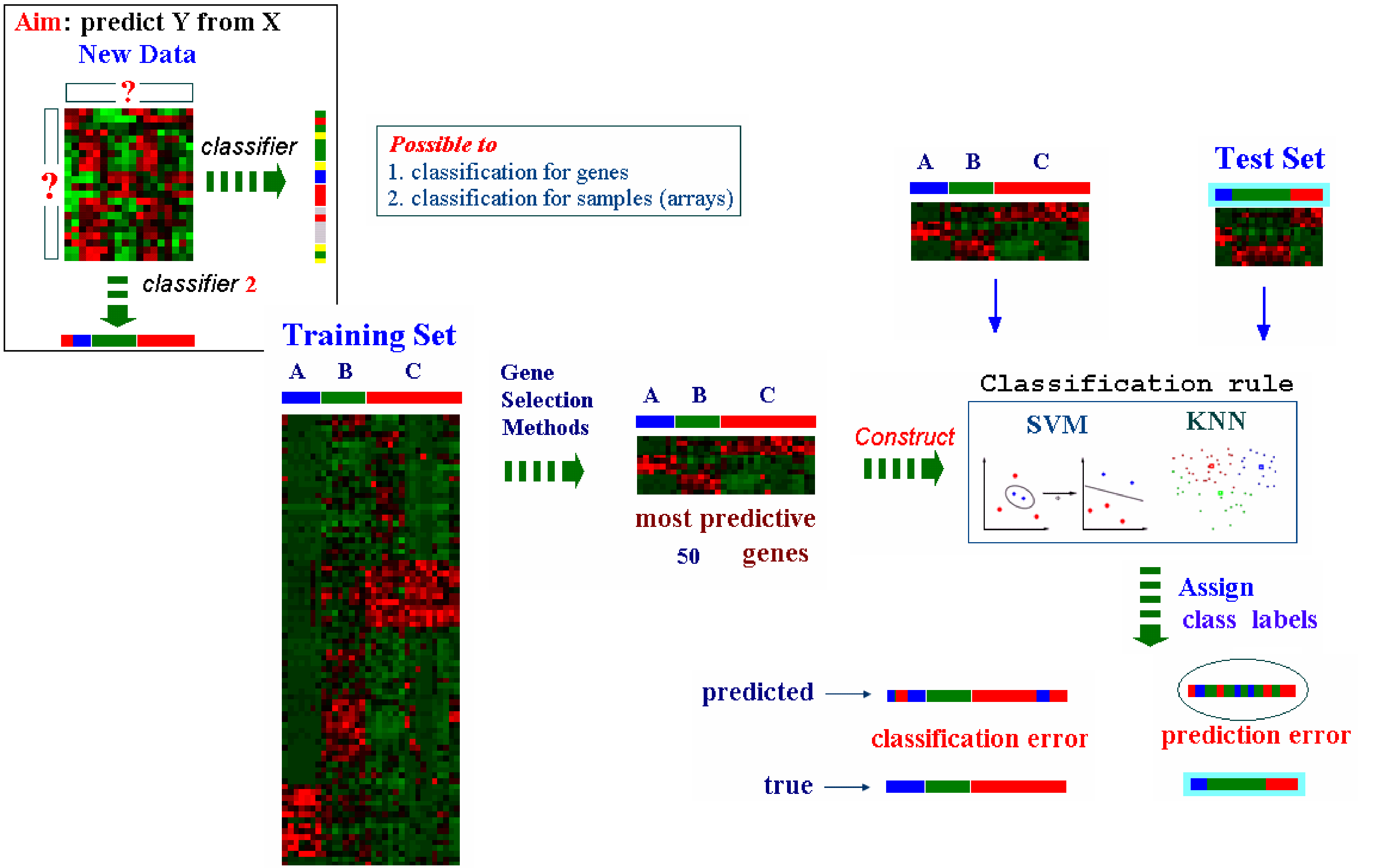
Why Resampling?

- Resampling is any of a variety of methods for:
  - Estimating the precision of sample statistics (medians, variances, percentiles) by using subsets of available data (**jackknifing**) or drawing randomly with replacement from a set of data points (**bootstrapping**).
  - Exchanging labels on data points when performing significance tests (**permutation tests**, **randomization tests**)
  - Validating models by using random subsets (bootstrapping, cross validation)

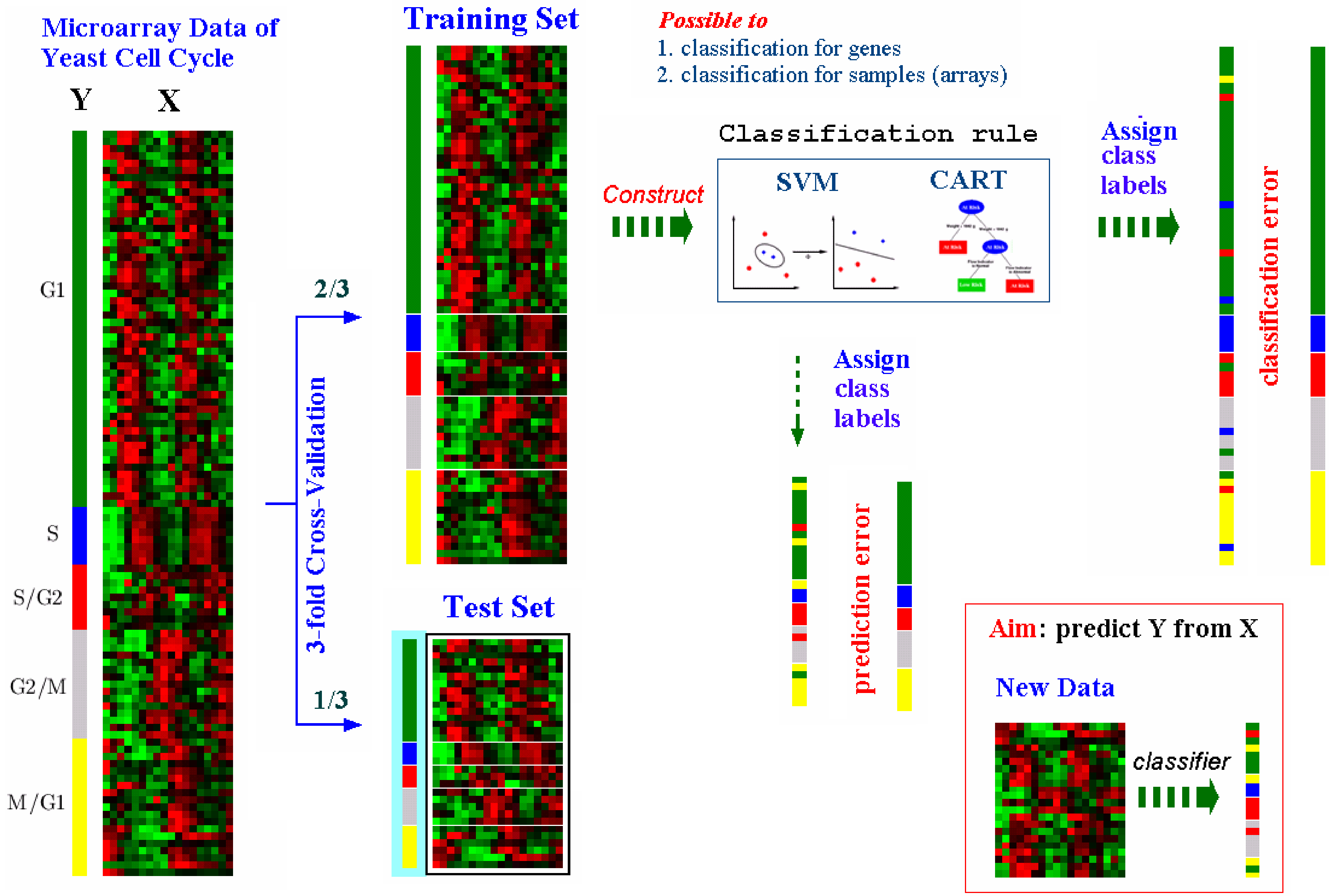
  https://en.wikipedia.org/wiki/Resampling_(statistics)

- This single sample method can serve as a **mini population**, from which repeated small samples are drawn with replacement over and over again.
- As well as saving time and money, bootstrapped samples can be **quite good approximations** for population parameters.

Classification of Genes, Tissues or Samples



k-fold Cross-Validation Error Rates



Split Data into Training Set and Test Set

```
> id <- sample(2, nrow(iris), replace = TRUE, prob = c(0.9, 0.1))
> id
  [1] 1 1 2 2 1 1 1 1 1 1 1 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1
 [38] 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [75] 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1
[112] 1 1 1 1 1 1 1 2 1 2 1 1 1 2 2 1 1 1 1 1 1 2 1 1 1 1 1 2 1 2 1 1 2 2 2 1 1
[149] 1 1
> train.data <- iris[id == 1, ]
> dim(train.data)
[1] 131   5
> test.data <- iris[id == 2, ]
> dim(test.data)
[1] 19  5
```

```
> id <- sample(nrow(iris), floor(nrow(iris) * 0.9))
> id
  [1]  39  27  96  33   4  98  12   3  32  48   2  22  18  24 126  93 140  85 110  60
 [21]  62  91 131  35 134 143  29 108 114  50  19  43  45  66  36  90 105  76 127  92
 [41]  68  57  65 147  69  41 130  82  31  20  51  17 149  61 107  70 139   5 115  72
 [61]  78 118 117  38  15  74 120 111 106  11 104  67  13  21 133  42  87 121 122  40
 [81]  84 135 123  77  83  97  52 116  55  88 142  16   7  49 125 112  34  10  56  26
[101]  99  63  37  46 144   9 141  59 138  80 101 132 129 113  73  30  44 136 119  79
[121]  95  64 109 148  28  14  86 150 137  81  94  75 128 102 124
> train.data <- iris[id, ]
> dim(train.data)
[1] 135   5
> test.data <- iris[-id, ]
> dim(test.data)
[1] 15  5
```

Split Data into Training Set and Test Set

```
splitdf <- function(df, train.ratio, seed=NULL) {
        if (!is.null(seed)) set.seed(seed)
        index <- 1:nrow(df)
        id <- sample(index, trunc(length(index)*train.ratio))
        train <- df[id, ]
        test <- df[-id, ]
        list(trainset=train,testset=test)
}
```

```
> splits <- splitdf(iris, 0.9, 12345)
> lapply(splits, dim)
$trainset
[1] 135   5

$testset
[1] 15  5

> iris.training <- splits$trainset
> iris.testing <- splits$testset
```

```
library(dplyr)
iris.train <- sample_frac(iris, 0.9)
id <- as.numeric(rownames(iris.train))
iris.test <- iris[-id, ]
```

Split Data into Test and Train Set According to Group Labels

```
> library(caTools)
> Y <- iris[,5] # extract labels from the data
> msk <- sample.split(Y, SplitRatio=4/5)
> msk
  [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE  TRUE
...
[144]  TRUE  TRUE  TRUE FALSE  TRUE  TRUE FALSE
> table(Y, msk)
            msk
Y             FALSE TRUE
  setosa        10   40
  versicolor    10   40
  virginica     10   40
> iris.train <- iris[msk, ]
> iris.test <- iris[!msk, ]
> dim(iris.train)
[1] 120   5
> dim(iris.test)
[1] 30  5
```

```
> library(caret)
> createFolds(iris$Species, k=3)
$Fold1
 [1]   2   8  15  22  25  27  30  ...

$Fold2
 [1]   5   6   9  10  11  12  17 ...

$Fold3
 [1]   1   3   4   7  13  14  16  20...
```

```
require(caTools)
set.seed(12345)
id <- sample.split(1:nrow(iris), SplitRatio = 0.90)
iris.train <- subset(iris, id == TRUE)
iris.test <- subset(iris, id == FALSE)
```

```
library(caret)
id <- createDataPartition(y=iris$Species, p=0.9, list=FALSE)
iris.train <- iris[id, ]
iris.test <- iris[-id, ]
```

Jackknife Resampling

$\hat{\theta}$ the calculated estimator of the parameter based on all n observations

$$\hat{\theta}_{(.)} = \frac{1}{n}\sum_{i=1}^{n}\hat{\theta}_{(\mathrm{i})}$$ the average of these "leave-one-out" estimates

$$\hat{\theta}_{\mathrm{Jack}} = n\hat{\theta} - (n-1)\hat{\theta}_{(.)}$$ the resulting bias-corrected jackknife estimate

```
> # install.packages("bootstrap")
> library(bootstrap)
> jackknife
function (x, theta, ...)
{
    call <- match.call()
    n <- length(x)
    u <- rep(0, n)
    for (i in 1:n) {
        u[i] <- theta(x[-i], ...)
    }
    thetahat <- theta(x, ...)
    jack.bias <- (n - 1) * (mean(u) - thetahat)
    jack.se <- sqrt(((n - 1)/n) * sum((u - mean(u))^2))
    return(list(jack.se = jack.se, jack.bias = jack.bias, jack.values = u,
        call = call))
}
<environment: namespace:bootstrap>
```

$$b_{jack} = (n-1)(\hat{\theta}_{(.)} - \hat{\theta})$$

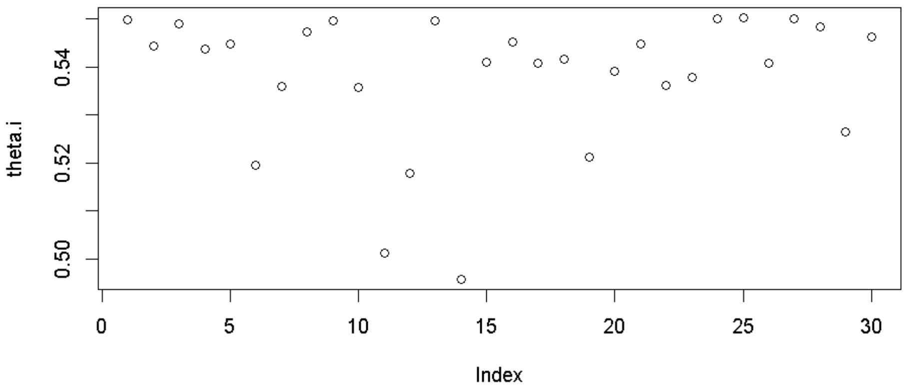
$$\hat{\theta}_{jack} = \hat{\theta} - b_{jack}$$

Example: Jackknife Estimate the Coefficient of Variation

```
> set.seed(12345)
> x <- runif(30)
> n <- length(x)
> theta <- CV(x)
> CV <- function(x) sqrt(var(x))/mean(x)
> theta.i <- sapply(1:n, function(i) CV(x[-i]))
> theta.i
 [1] 0.5497915 0.5442365 0.5489822 0.5436256 0.5448185 0.5195935 0.5359400 0.5472011
 [9] 0.5496842 0.5357489 0.5011942 0.5178517 0.5495427 0.4958063 0.5409312 0.5451245
[17] 0.5407236 0.5416770 0.5211182 0.5390234 0.5446755 0.5360780 0.5378925 0.5499674
[25] 0.5501676 0.5408382 0.5500584 0.5484004 0.5265137 0.5461715
> theta.jack <- n*theta - (n-1)*mean(theta.i)
> theta.jack
[1] 0.5356475
> plot(theta.i)
```

$$\mathrm{CV} = \sqrt{\mathrm{Var}} / \overline{x}$$

```
jack <- numeric(length(x)-1)
pseudo <- numeric(length(x))
for (i in 1:length(x))
{ for (j in 1:length(x))
{if(j < i) jack[j] <- x[j] else if(j > i) jack[j-1] <- x[j]}
pseudo[i] <- length(x)*CV(x) -(length(x)-1)*CV(jack)}
```



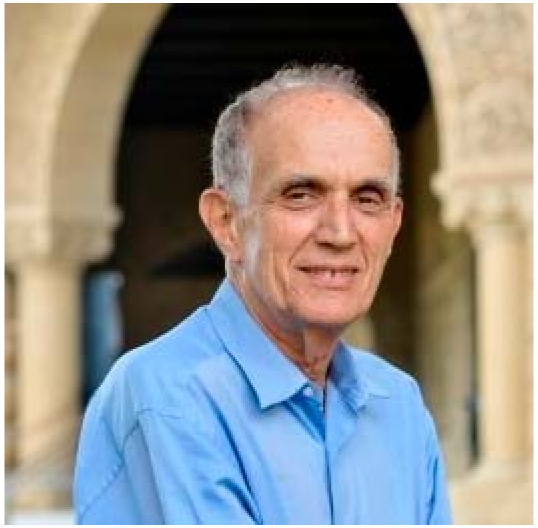
Jackknifing can be useful for analyzing if influential observations are affecting our estimates.

Jackknife the Coefficients of a Linear Regression Model

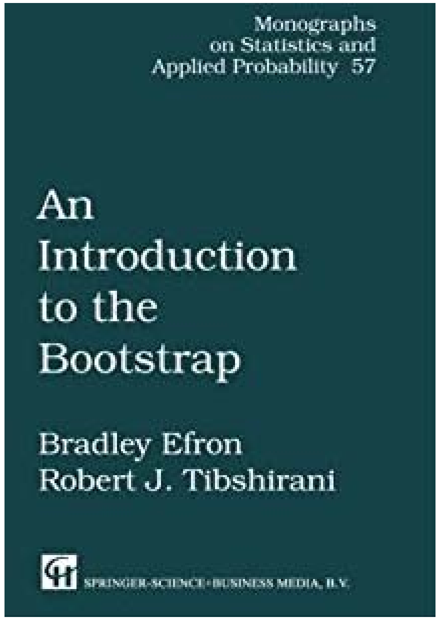
```
> library(bootstrap)
> set.seed(12345)
> n <- 50; p <- 5
> mydata <- as.data.frame(matrix(rnorm(n*p), ncol=p))
> head(mydata, 3)
          V1          V2         V3         V4         V5
1  0.5855288 -0.54038607  0.2239254 -1.6193283 -1.4361457
2  0.7094660  1.94769266 -1.1562233  0.5483979 -0.6292596
3 -0.1093033  0.05359027  0.4224185  0.1952822  0.2435218
> model.lm <- formula(V1 ~ V2 + V3 + V4)
> theta <- function(x, xdata, coefficient){
+    coef(lm(model.lm, data=xdata[x, ]))[coefficient]
+ }
> results <- jackknife(1:n, theta, xdata=mydata, coefficient="(Intercept)")
> results
$jack.se
[1] 0.1672309

$jack.bias
(Intercept)
0.003368696

$jack.values
 [1] 0.1412249 0.1570365 0.1723303 0.1703336 0.1529388 0.2038722 0.1620162 0.1754961
...
[41] 0.1384219 0.2296432 0.1793121 0.1429386 0.1545121 0.1456370 0.2016571 0.1582340
[49] 0.1536307 0.2034109

$call
jackknife(x = 1:n, theta = theta, xdata = mydata, coefficient = "(Intercept)")
```

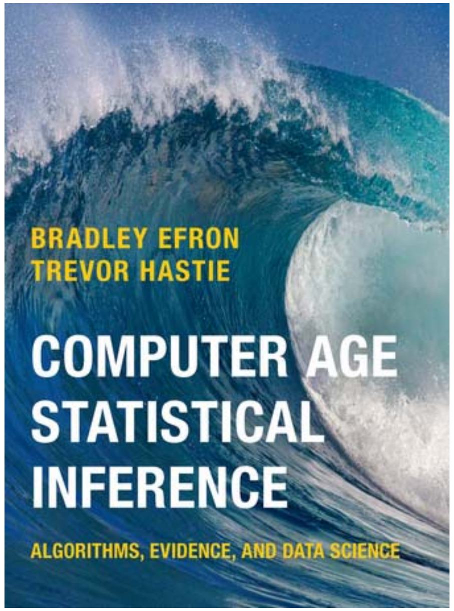
Bootstrap Methods

- Bootstrapping is a statistical method for estimating the sampling distribution of an estimator by sampling with replacement from the original sample, of the same size as the original sample.
- The name "bootstrapping" comes from the phrase: "To lift himself up by his bootstraps".
- This refers to something that is preposterous and impossible.
- Try as hard as you can, you cannot lift yourself into the air by tugging at pieces of leather on your boots.

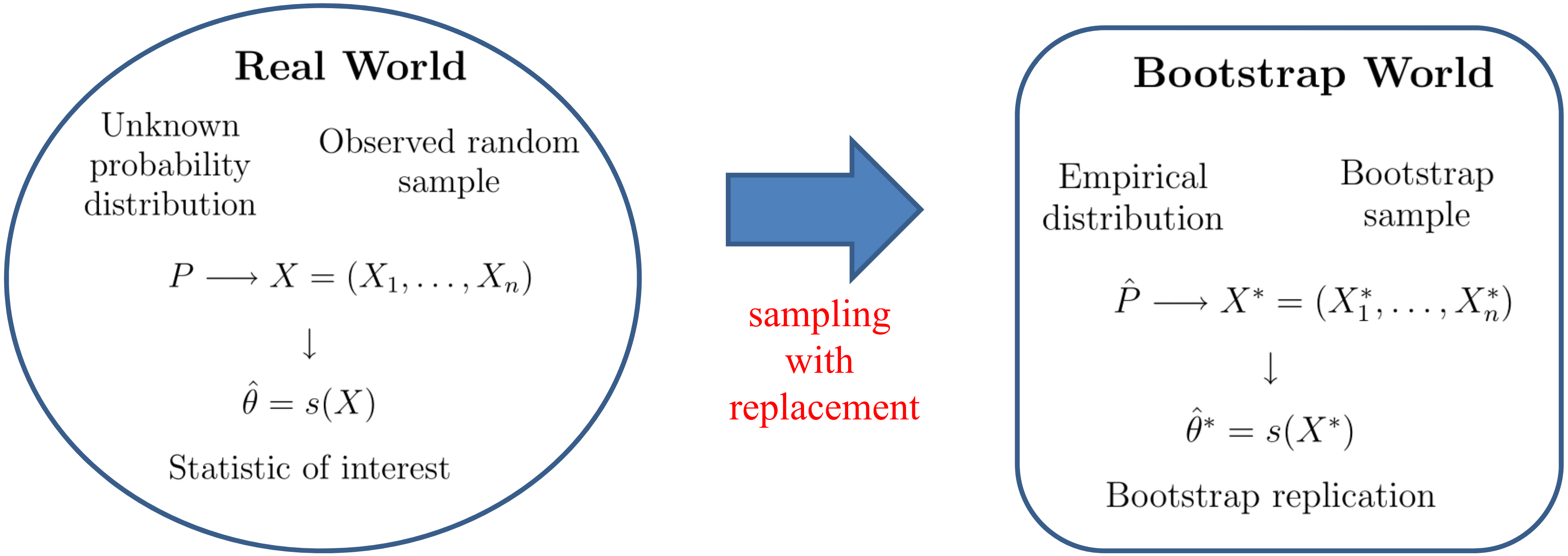


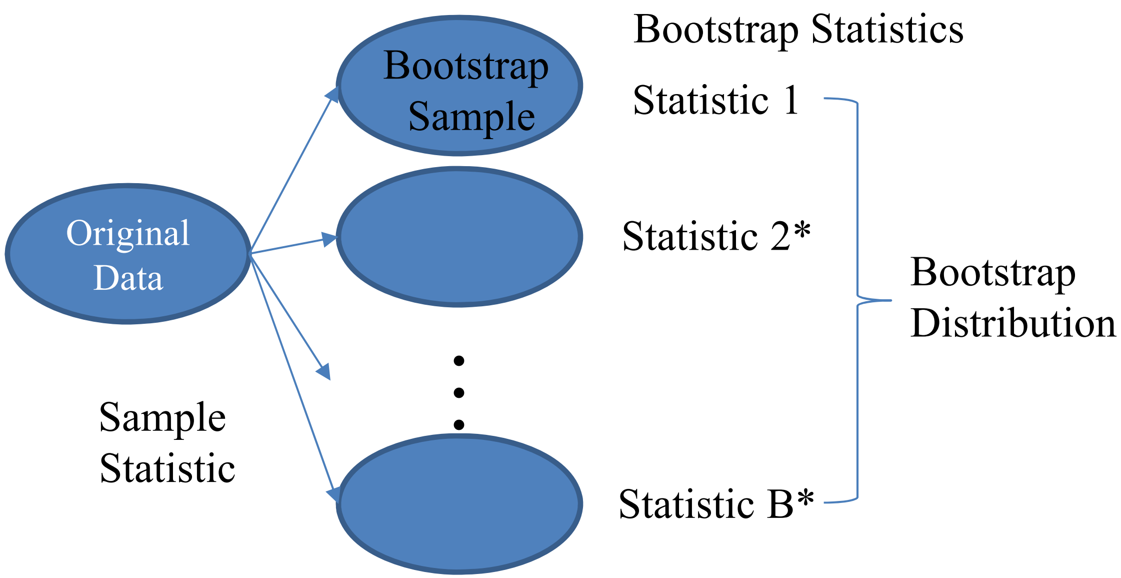
Bradley Efron 1938~
Department of Statistics
Stanford University





Bootstrapping

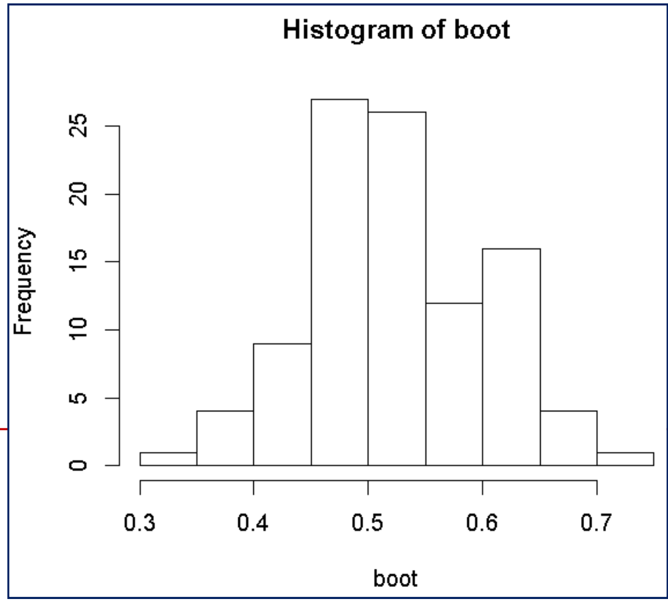




- Types of bootstrap scheme: Case resampling, Bayesian bootstrap, Smooth bootstrap, Parametric bootstrap, Resampling residuals, Gaussian process regression bootstrap, Wild bootstrap, Block bootstrap.
- An empirical bootstrap sample is drawn from observations.
- A parametric bootstrap sample is drawn from a parameterized distribution (e.g. a normal distribution).

Example: Bootstrap Estimate the Coefficient of Variation

$$CV = \sqrt{\text{Var}} / \overline{x}$$




```
> set.seed(12345)
> x <- runif(30)
> CV <- function(x) sqrt(var(x))/mean(x)
> CV(x)
[1] 0.5380304
> CV(sample(x, replace=T)) # a single bootstrap sample
[1] 0.5459389
> boot <- replicate(n=100, expr=CV(sample(x, replace=T)))
> boot
  [1] 0.5044811 0.5286011 0.4634611 0.5605438 0.4835447 0.5374531 0.4857342 0.4342565
 ...
[89] 0.5297020 0.5121274 0.4938053 0.5479498 0.5262306 0.6095145 0.5322045 0.6069263
 [97] 0.5374840 0.4921430 0.4674226 0.4573680
> mean(boot)
[1] 0.5251909
> var(boot)
[1] 0.006107636
> hist(boot)
```

bootstrap Package

Bootstrap Estimation of the Sample Mean

語法:

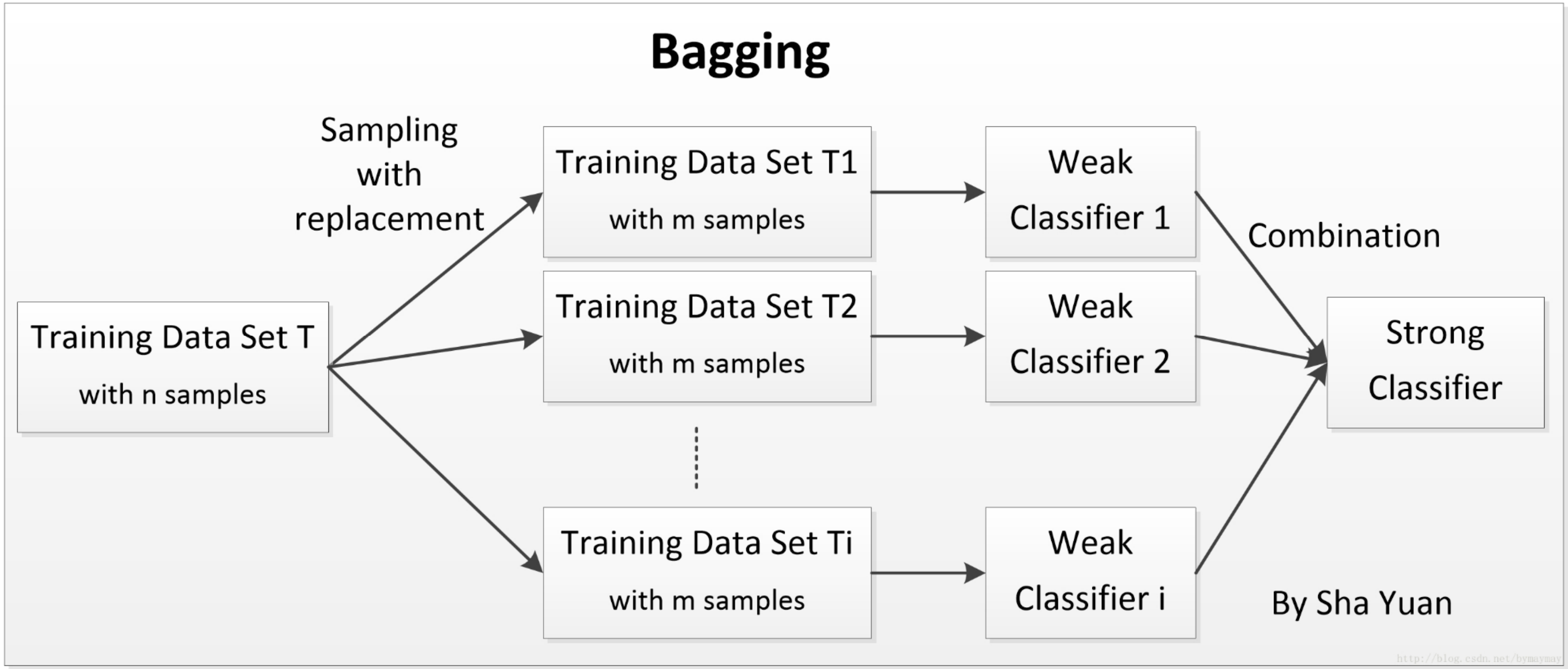
```
bootstrap(x, nboot, theta, ..., func=NULL)
```

- `x:` a vector containing the data.
- `nboot:` the number of bootstrap samples.
- `theta:` function to be bootstrapped.

```
> # install.packages("bootstrap")
> library(bootstrap)
> set.seed(12345)
> x <- rnorm(20)
> mean(x)
[1] 0.07651681
> (x.bootstrap.mean <- bootstrap(x, 50, theta=mean))
$thetastar
 [1]  0.486197466 -0.160488357  0.274920990  0.398499864 -0.399967845  0.116086370
...
[43] -0.348643786  0.185330636 -0.070823890  0.057609481  0.062067504  0.043716794
[49] -0.279597885  0.243843620

$func.thetastar
NULL

$jack.boot.val
NULL

$jack.boot.se
NULL

$call
bootstrap(x = x, nboot = 50, theta = mean)
> mean(x.bootstrap.mean$thetastar)
[1] 0.08647268
```

```
> mu.hat <- mean(x)
> n <- length(x)
> ja <- jackknife(x, mean)
> mu.hat.jack <- n*mu.hat - (n-1)*mean(ja$jack.values)
> # or
> mu.hat.jack <- mu.hat - ja$jack.bias
```

Bagging: Bootstrap Aggregating



http://blog.csdn.net/bymaymay/article/details/77824574

- Breiman, L. (1996). Bagging predictors, Machine Learning, Vol. 26, pp. 123-140.
- Freund, Y. and Schapire, R. E. (1996). Experiments with a new boosting algorithm, Proceedings of the Thirteenth International Conference, Machine Learning.

Boosting

$$W_{x_i}^{(1)} = N^{-1} \text{ for all } x_i.$$

a bootstrap sample $\mathcal{L}_t^{(B)}$

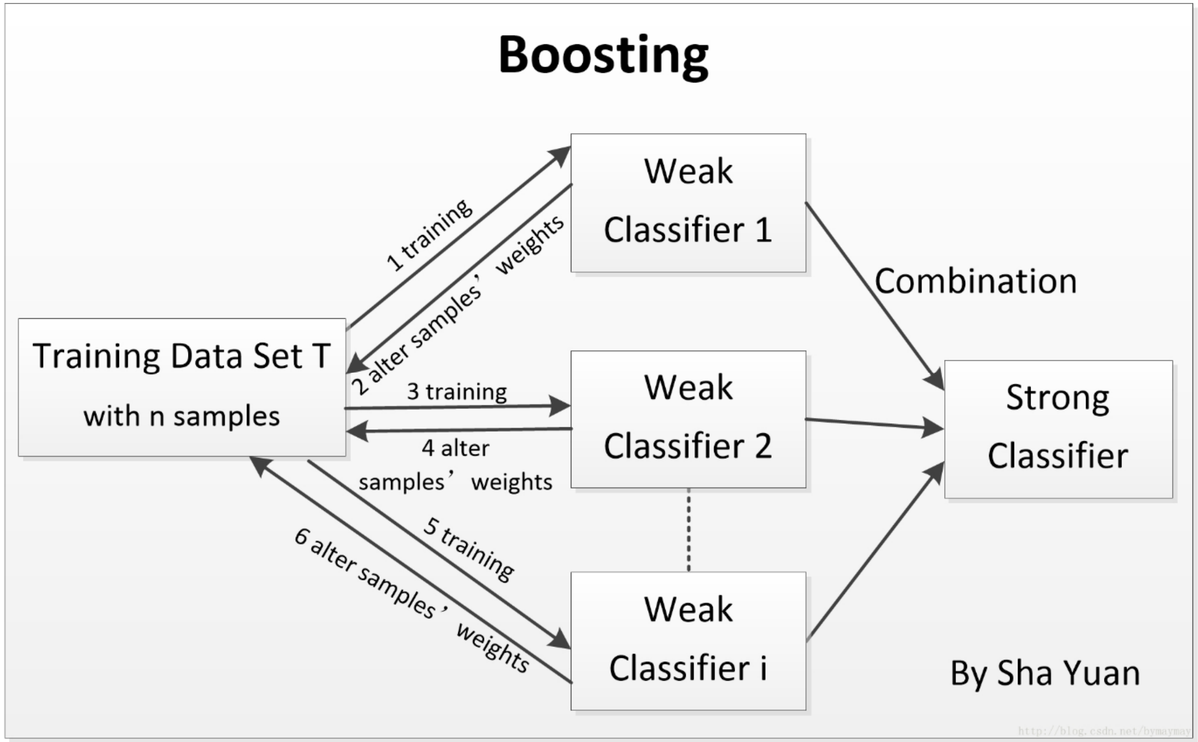
error ϵ_t of classifier $\varphi_t(\mathbf{x})$

$$\epsilon_t = \sum_{\{i:\varphi_t(x_i)\neq y_i\}} W_{x_i}^{(t)}.$$

$$\beta_t = (1-\epsilon_t)\epsilon_t^{-1}$$

$$W_{x_i}^{(t+1)} = \frac{W_{x_i}^{(t)}\beta_t^{d(i)}}{\sum_i W_{x_i}^{(t)}\beta_t^{d(i)}},$$

boosted classifier



http://blog.csdn.net/bymaymay/article/details/77824574

$d(i) = 1$ if ith case is classified incorrectly,

$d(i) = 0$, otherwise

$$\varphi_B(x_i) = arg\ max_j \sum_{t=1}^{T} \log \beta_t I[\varphi_t(x_i) = j]$$

Ad-Boost.M1 (Freund and Schapire, 1996)

Example: Apply rpart to Vehicle Data

```
> library(rpart); library(mlbench); library(adabag)
> data(Vehicle)
> dim(Vehicle)
[1] 846  19
> head(Vehicle)
 Comp Circ D.Circ Rad.Ra Pr.Axis.Ra Max.L.Ra Scat.Ra Elong Pr.Axis.Rect Max.L.Rect Sc.Var.Maxis
1   95   48     83    178         72       10     162    42            20        159          176
...
  Sc.Var.maxis Ra.Gyr Skew.Maxis Skew.maxis Kurt.maxis Kurt.Maxis Holl.Ra Class
1          379    184         70          6         16        187     197   van
...
6          957    264         85          5          9        181     183   bus
> table(Vehicle$Class)
 bus opel saab  van
 218  212  217  199
```

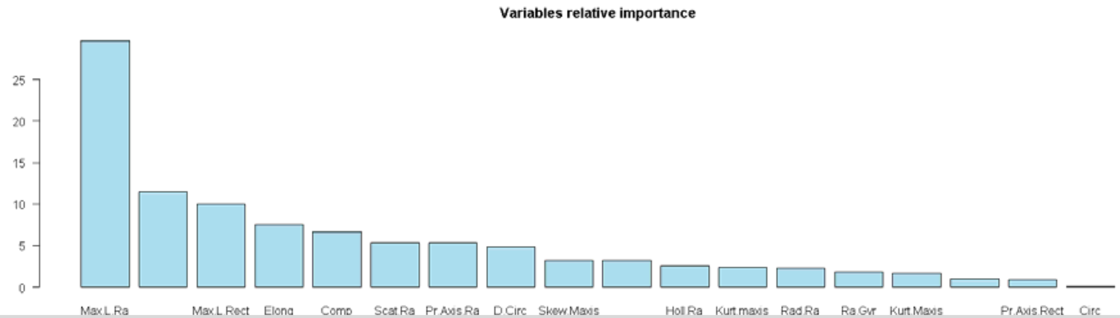
```
> n <- nrow(Vehicle)
> sub <- sample(1:n, 2*n/3)
> Vehicle.train <- Vehicle[sub, ]
> Vehicle.test <- Vehicle[-sub, ]
```

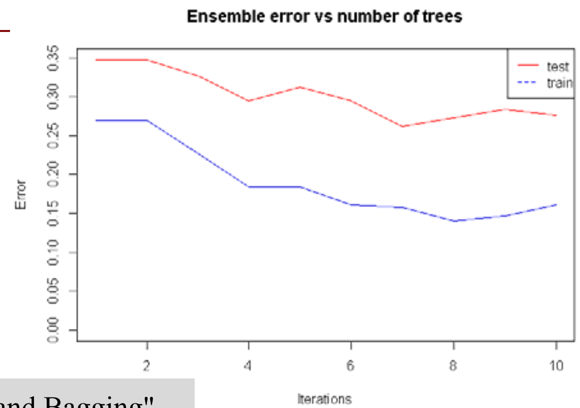
```
> mfinal <- 10 #Defaults to mfinal=100 iterations
> maxdepth <- 5
> Vehicle.rpart <- rpart(Class ~ ., data = Vehicle.train, maxdepth = maxdepth)
> Vehicle.rpart.pred <- predict(Vehicle.rpart, newdata = Vehicle.test, type = "class")
> (tb <- table(Vehicle.rpart.pred, Observed.Class=Vehicle.test$Class))
                  Observed.Class
Vehicle.rpart.pred bus opel saab van
              bus   69   10    6   2
              opel   1   25   13   3
              saab   1   34   37   8
              van    2    7    5  59
> (error.rpart <- 1 - (sum(diag(tb)) / sum(tb)))
[1] 0.3262411
```

adabag: An R Package for Classification with Boosting and Bagging

```
> library(adabag)
> Vehicle.adaboost <- boosting(Class ~., data = Vehicle.train, mfinal = mfinal,
+                              control = rpart.control(maxdepth=maxdepth))
> Vehicle.adaboost.pred <- predict.boosting(Vehicle.adaboost, newdata = Vehicle.test)
> Vehicle.adaboost.pred$confusion
               Observed Class
Predicted Class bus opel saab van
           bus   69    2    3   1
           opel   1   30   16   4
           saab   1   38   39   1
           van    2    6    3  66
> Vehicle.adaboost.pred$error
[1] 0.2765957
> importanceplot(Vehicle.adaboost)
>
> # comparing error evolution in training and test set
> evol.train <- errorevol(Vehicle.adaboost, newdata = Vehicle.train)
> evol.test <- errorevol(Vehicle.adaboost, newdata = Vehicle.test)
> plot.errorevol(evol.test, evol.train)
```

```
> sort(Vehicle.adaboost$importance, dec=T)[1:5]
   Max.L.Ra Sc.Var.maxis   Max.L.Rect
  29.623783    11.473254     9.956137
      Elong         Comp
   7.570798     6.656360
```





Alfaro, E., Gamez, M. and Garcia, N. (2013): "adabag: An R Package for Classification with Boosting and Bagging". Journal of Statistical Software, 54(2), 1–35.

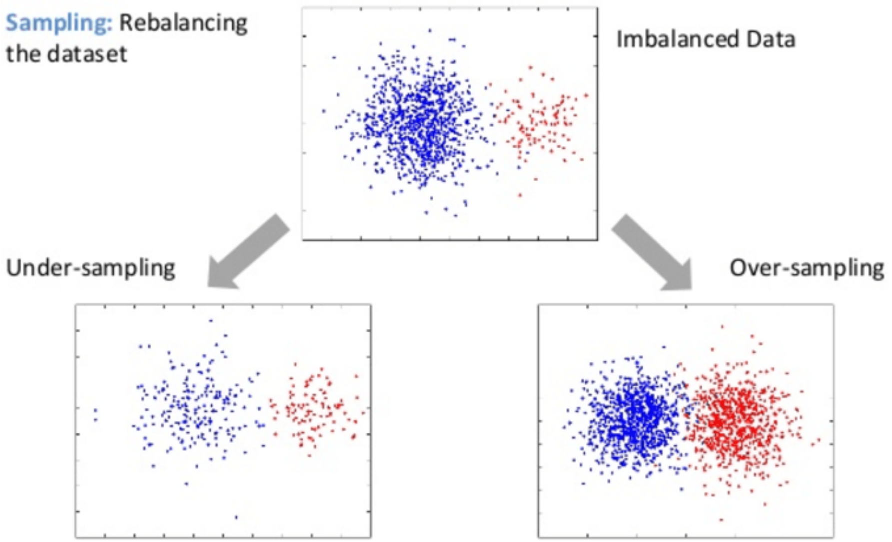
Example: 10-fold CV adaboost.M1

```
> # 10-fold CV adaboost.M1
> Vehicle.boost.cv <- boosting.cv(Class ~., data = Vehicle, v = 10, mfinal = 5,
+                                 control = rpart.control(maxdepth=maxdepth))
i:  1 Tue Dec 05 09:36:37 2017
...
> Vehicle.boost.cv$confusion
               Observed Class
Predicted Class bus opel saab van
           bus  209    9   11   3
           opel   1  101   72   2
           saab   0   88  117   6
           van    8   14   17 188
> Vehicle.boost.cv$error
[1] 0.2730496
```

```
> Vehicle.bag.cv <- bagging.cv(Class ~., data = Vehicle, v = 10, mfinal = 5,
+                              control = rpart.control(maxdepth=maxdepth))
> Vehicle.bag.cv$confusion
               Observed Class
Predicted Class bus opel saab van
           bus  208   20   24   2
           opel   1   91   53   1
           saab   1   81  116   5
           van    8   20   24 191
> Vehicle.bag.cv$error
[1] 0.2836879
```

The Imbalanced Data Problem

- A dataset is said to be **unbalanced** when the class of interest (minority class) is much rarer than normal behaviour (majority class).
- The cost of missing a minority class is typically much higher that missing a majority class. Most learning systems are not prepared to cope with unbalanced data and several techniques have been proposed.
- **Example**: 5% of the target class represents fraudulent transactions, 95% of the target class represents legitimate transactions.



http://www.srutisj.in/blog/research/statisticalmodeling/balancing-techniques-for-unbalanced-datasets-in-python-r/

unbalanced

Racing for Unbalanced Methods Selection

Re-balance or remove noisy instances in unbalanced datasets.

ubBalance {unbalanced}

Usage

```
ubBalance(X, Y, type="ubSMOTE", positive=1,
          percOver=200, percUnder=200,
          k=5, perc=50, method="percPos", w=NULL, verbose=FALSE)
```

Arguments

X: the input variables of the unbalanced dataset.
Y: the response variable of the unbalanced dataset.
type: the balancing technique to use (**ubOver, ubUnder, ubSMOTE, ubOSS, ubCNN, ubENN, ubNCL, ubTomek**).
positive: the majority class of the response variable.
percOver: parameter used in **ubSMOTE**
percUnder: parameter used in **ubSMOTE**
k: parameter used in **ubOver, ubSMOTE, ubCNN, ubENN, ubNCL**
perc: parameter used in **ubUnder**
method: parameter used in **ubUnder**
w: parameter used in **ubUnder**
verbose: print extra information (TRUE/FALSE)

ubSMOTE {unbalanced}: synthetic minority over-sampling technique

Usage

```
ubSMOTE(X, Y, perc.over = 200, k = 5, perc.under = 200, verbose = TRUE)
```

NOTE: **imbalance**: Preprocessing Algorithms for Imbalanced Datasets, Imbalanced Classification in R: **ROSE** (Random Over Sampling Examples) and **DMwR** (Data Mining with R).

The Balancing Technique

- **`ubOver`**: replicates randomly some instances from the minority class in order to obtain a final dataset with the same number of instances from the two classes.
- **`ubUnder`**: removes randomly some instances from the majority (negative) class and keeps all instances in the minority (positive) class in order to obtain a more balanced dataset.

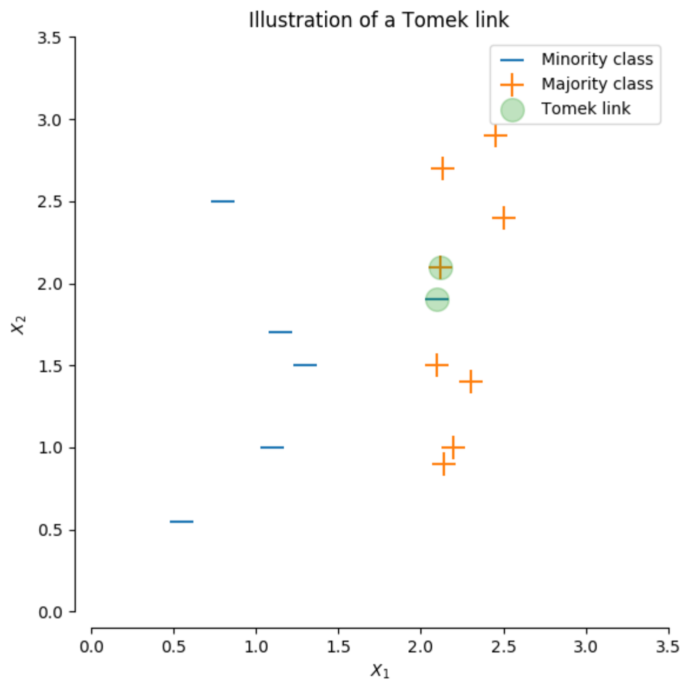
- **`ubCNN`**: Condensed Nearest Neighbor selects the subset of instances that are able to correctly classifying the original datasets using a one-nearest neighbor rule.
- **`ubENN`**: Edited Nearest Neighbor removes any example whose class label differs from the class of at least two of its three nearest neighbors.
- **`ubNCL`**: Neighborhood Cleaning Rule modifies the Edited Nearest Neighbor method by increasing the role of data cleaning.
  - Firstly, NCL removes negatives examples which are misclassified by their 3-nearest neighbors.
  - Secondly, the neighbors of each positive examples are found and the ones belonging to the majority class are removed.

The Balancing Technique

- **ubTomek**: finds the points in the dataset that are tomek link using 1-NN and then removes only majority class instances that are tomek links.

x's nearest neighbor is y
y's nearest neighbor is x
x and y are different classes

http://contrib.scikit-learn.org/imbalanced-learn/stable/auto_examples/under-sampling/plot_illustration_tomek_links.html

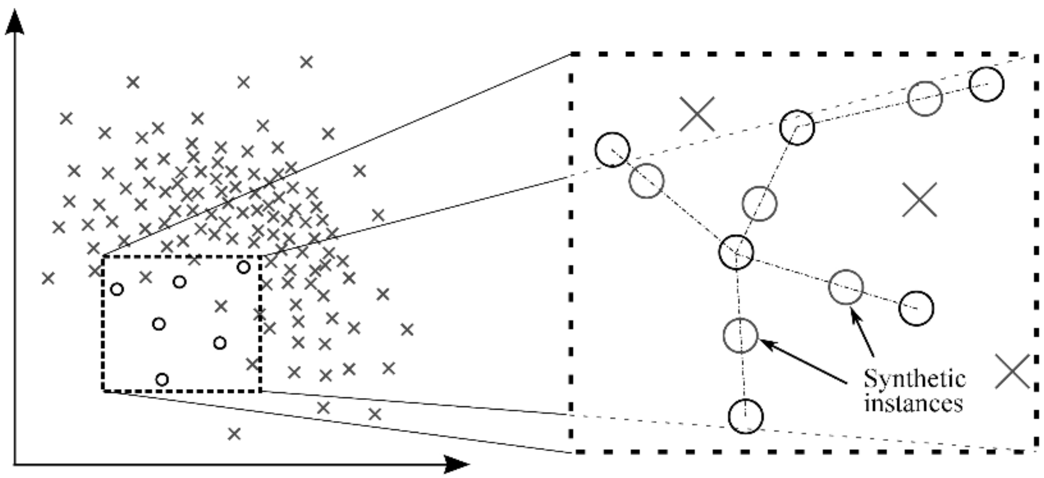


- **ubOSS**: One Side Selection is an undersampling method resulting from the application of Tomek links followed by the application of Condensed Nearest Neighbor.

The Balancing Technique

- **ubSMOTE**: synthetic minority over-sampling technique generates new examples by filling empty areas among the positive instances

N. V. Chawla, K. W. Bowyer, L. O. Hall, W. P. Kegelmeyer, SMOTE: Synthetic Minority Over-sampling Technique, *Journal Of Artificial Intelligence Research*, Volume 16, pages 321-357, 2002.(由 NV Chawla 著作 - 2002 - 被引用 5161 次)



- **ubRacing**: the Racing algorithm for selecting the best technique to re-balance or remove noisy instances in unbalanced datasets.

Ionosphere dataset
ubIonosphere {unbalanced}

- The datasets is a modification of Ionosphere dataset contained in "mlbench" package.

```
> data(ubIonosphere)
> dim(ubIonosphere)
[1] 351  33
> head(ubIonosphere)
       V3       V4           V34 Class
1 0.99539 -0.05889 ... -0.45300     0
...
6 0.02337 -0.00592 ...  0.12011     1
> table(ubIonosphere$Class)
  0   1
225 126
```

```
> # install.packages("unbalanced")
> library(unbalanced)
> p <- ncol(ubIonosphere)
> y <- ubIonosphere$Class
> x <- ubIonosphere[ ,-p]
> data <- ubBalance(X=x, Y=y, type="ubOver", k=0)
> overData <- data.frame(data$X, Class=data$Y)
> table(overData$Class)
  0   1
225 225
> data <- ubBalance(X=x, Y=y, type="ubUnder", perc=50, method="percPos")
> underData <- data.frame(data$X, Class=data$Y)
> table(underData$Class)
  0   1
126 126
> bdata <- ubBalance(X=x, Y=y, type="ubSMOTE", percOver=300, percUnder=150, verbose=TRUE)
Proportion of positives after ubSMOTE : 47.06 % of 1071 observations
> str(bdata)
List of 3
 $ X    :'data.frame':  1071 obs. of  32 variables:
  ..$ V3 : num [1:1071] -0.787 1 1 0.5 1 ...
...
..$ V34: num [1:1071] -0.576 0.714 -0.243 0.174 -0.892 ...
 $ Y    : Factor w/ 2 levels "0","1": 2 1 1 1 1 2 1 2 1 2 ...
 $ id.rm: logi NA
> table(bdata$Y)
  0   1
567 504
```

K=0: sample with replacement from the minority class until we have the same number of instances in each class.
If K>0: sample with replacement from the minority class until we have k-times the orginal number of minority instances

`per.over/100` : number of new instances generated for each rare instance
`perc.under/100`: number of "normal" (majority class) instances that are randomly selected for each smoted observation.

Compare the Performances using SVM

```
> set.seed(12345)
> n <- nrow(ubIonosphere) # 351
> no.train <- floor(0.5*n) # 175, keep half for training and half for testing
> id <- sample(1:n, no.train)
> x.train  <- x[id, ]  # 175 x 32
> y.train <- y[id]
> x.test <- x[-id, ] # 176  32
> y.test <- y[-id]
>
> library(e1071)
> model1 <- svm(x.train, y.train)
> y.pred1 <- predict(model1, x.test)
> table(y.pred1, y.test)
       y.test
y.pred1   0   1
      0 113  10
      1   4  49
>
> # rebalance the training set before building a model
> balancedData <- ubBalance(X=x.train, Y=y.train, type="ubSMOTE",
                            percOver=200, percUnder=150)
> table(balancedData$Y)
  0   1
201 201
```

```
> model2 <- svm(balancedData$X, balancedData$Y)
> y.pred2 <- predict(model2, x.test)
> table(y.pred2, y.test)
       y.test
y.pred2   0   1
      0 112   8
      1   5  51
```

ubRacing {unbalanced}
Racing for Strategy Selection

Selecting the best technique to re-balance or remove noisy instances in unbalanced datasets.

```
ubRacing(formula, data, algo, positive=1, ncore=1, nFold=10, maxFold=10, maxExp=100,
         stat.test="friedman", metric="f1", ubConf, verbose=FALSE, ...)
```

Arguments

- **algo**: the classification algorithm to use with the mlr package.
- **positive**: label of the positive (minority) class.
- **ncore**: the number of core (parallel execution) to use in the Race.

```
> set.seed(1234)
> # load(url("http://www.ulb.ac.be/di/map/adalpozz/data/creditcard.Rdata"))
> load("creditcard.Rdata")
> str(creditcard)
'data.frame':   284807 obs. of  31 variables:
 $ Time  : num  0 0 1 1 2 2 4 7 7 9 ...
 $ V1    : num  -1.36 1.192 -1.358 -0.966 -1.158 ...
...
 $ V28   : num  -0.0211 0.0147 -0.0598 0.0615 0.2152 ...
 $ Amount: num  149.62 2.69 378.66 123.5 69.99 ...
 $ Class : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
> table(creditcard$Class)

     0      1 
284315    492 
> # configuration of the sampling method used in the race
> ubConf <- list(percOver=200, percUnder=200, k=2, perc=50, method="percPos", w=NULL)
> # Race with 5 trees in the Random Forest
> results <- ubRacing(Class ~., creditcard, "randomForest",
+                      positive=1, metric="auc", ubConf=ubConf, ntree=5)
```

The function **ubRacing** compares the 8 unbalanced methods (ubUnder, ubOver, ubSMOTE, ubOSS, ubCNN, ubENN, ubNCL, ubTomek) against the unbalanced distribution.

Racing for Strategy Selection

```
Racing for unbalanced methods selection in 10 fold CV
Number of candidates...........................................9
Max number of folds in the CV.................................10
Max number of experiments....................................100
Statistical test...................................Friedman test

                           Markers:
                              x No test is performed.
                              - The test is performed and
                                some candidates are discarded.
                              = The test is performed but
                                no candidate is discarded.

+-+-----------+-----------+-----------+-----------+-----------+
| |       Fold|      Alive|       Best|  Mean best| Exp so far|
+-+-----------+-----------+-----------+-----------+-----------+
|x|          1|          9|          4|     0.9543|          9|
|=|          2|          9|          3|     0.9433|         18|
|-|          3|          3|          4|     0.9567|         27|
|-|          4|          2|          4|     0.9566|         30|
|=|          5|          2|          4|     0.9582|         32|
|=|          6|          2|          4|     0.9546|         34|
|=|          7|          2|          4|     0.9531|         36|
|=|          8|          2|          4|     0.9539|         38|
|=|          9|          2|          4|     0.9531|         40|
|=|         10|          2|          4|     0.9529|         42|
+-+-----------+-----------+-----------+-----------+-----------+
Selected candidate: ubSMOTE      metric: auc     mean value: 0.9529
```

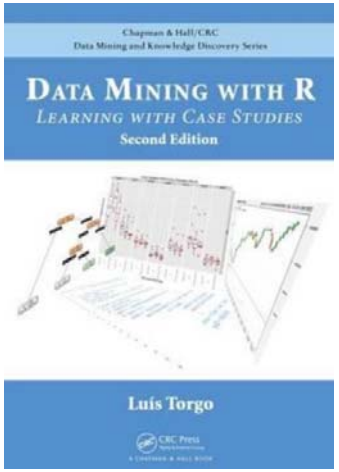
Racing for Strategy Selection

```
> # Race using 4 cores and 500 trees (default)
> results <- ubRacing(Class ~., creditcard, "randomForest",
                      positive=1, metric="auc", ubConf=ubConf, ncore=4)
> library(e1071)
> results <- ubRacing(Class ~., creditcard, "svm",
                      positive=1, ubConf=ubConf)
> library(rpart)
> results <- ubRacing(Class ~., creditcard, "rpart",
                      positive=1, ubConf=ubConf)
```

```
> results
$best
[1] "ubSMOTE"

$avg
[1] 0.9529177

$sd
[1] 0.009049014

$N.test
[1] 42

$Gain
[1] 53

$Race
         unbal    ubOver   ubUnder   ubSMOTE     ubOSS     ubCNN     ubENN     ubNCL   ubTomek
 [1,] 0.8844582 0.9138946 0.9354739 0.9543104 0.8957273 0.9139340 0.9024656 0.9014143 0.9048642
 [2,] 0.9116642 0.9104928 0.9511485 0.9507221 0.9037491 0.9104840 0.9139047 0.9094542 0.9105558
 [3,] 0.8979478 0.9013642 0.9502417 0.9649361 0.9092505 0.9081796 0.9103668 0.9036617 0.9058917
 [4,]        NA        NA 0.9503782 0.9564226        NA        NA 0.8999928        NA        NA
 [5,]        NA        NA 0.9537802 0.9647722        NA        NA        NA        NA        NA
 [6,]        NA        NA 0.9494913 0.9362763        NA        NA        NA        NA        NA
 [7,]        NA        NA 0.9411979 0.9440379        NA        NA        NA        NA        NA
 [8,]        NA        NA 0.9576971 0.9594249        NA        NA        NA        NA        NA
 [9,]        NA        NA 0.9530119 0.9473722        NA        NA        NA        NA        NA
[10,]        NA        NA 0.9633438 0.9509024        NA        NA        NA        NA        NA
```

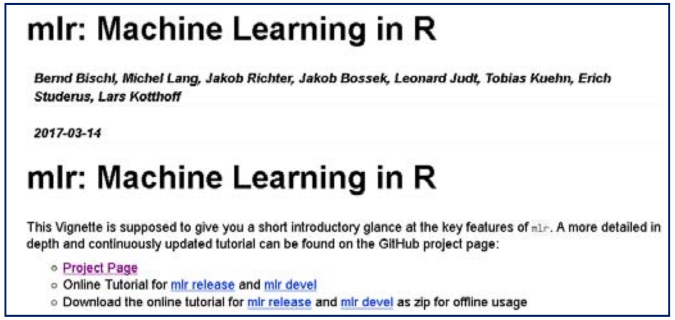
Useful R Packages

imbalance: Preprocessing Algorithms for Imbalanced Datasets
https://cran.r-project.org/web/packages/imbalance/index.html
Working with imbalanced datasets
https://cran.r-project.org/web/packages/imbalance/vignettes/imbalance.pdf



mlr: Machine Learning in R

https://cran.r-project.org/web/packages/mlr/vignettes/mlr.html



mlr: Machine Learning in R

Bernd Bischl, Michel Lang, Jakob Richter, Jakob Bossek, Leonard Judt, Tobias Kuehn, Erich Studerus, Lars Kotthoff

2017-03-14

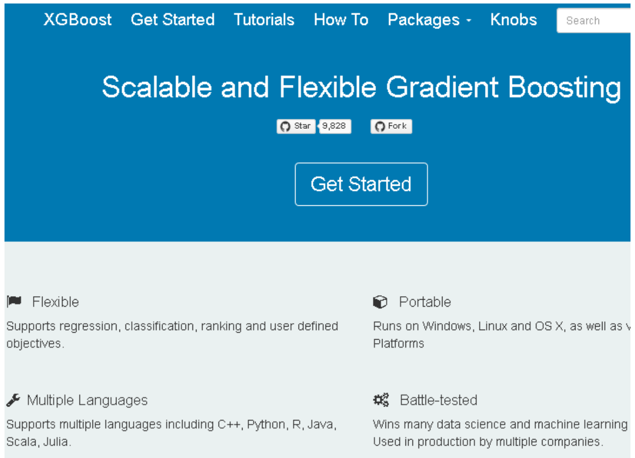
mlr: Machine Learning in R

This Vignette is supposed to give you a short introductory glance at the key features of mlr. A more detailed in depth and continuously updated tutorial can be found on the GitHub project page:

- Project Page
- Online Tutorial for mlr release and mlr devel
- Download the online tutorial for mlr release and mlr devel as zip for offline usage

DMwR: Functions and data for "Data Mining with R"

https://cran.r-project.org/web/packages/DMwR/index.html



XGBoost: eXtreme Gradient Boosting

(used for supervised learning tasks such as Regression, Classification, and Ranking)
https://github.com/dmlc/xgboost
http://xgboost.readthedocs.io/en/latest/
How to use XGBoost algorithm in R in easy steps
https://www.analyticsvidhya.com/blog/2016/01/xgboost-algorithm-easy-steps/
Kaggle 神器 XGBoost 入門：為什麼要用它?怎麼用?
https://weiwenku.net/d/100778240