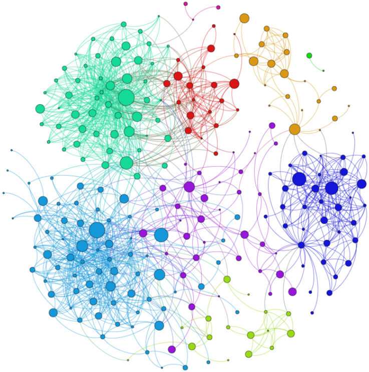
Tree-based Methods

Jiaming Mao

Xiamen University



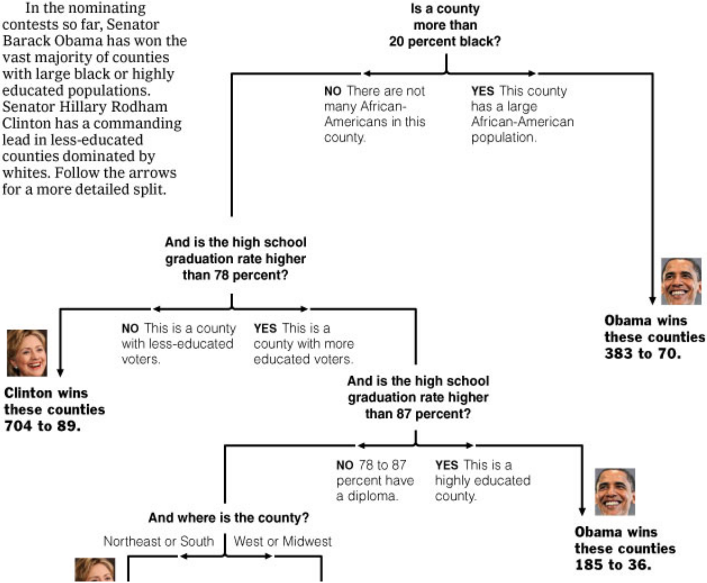
This version: Spring 2019

Contact: jmao@xmu.edu.cn

Course homepage: jiamingmao.github.io/data-analysis

Decision Tree: The Obama-Clinton Divide

In the nominating contests so far, Senator Barack Obama has won the vast majority of counties with large black or highly educated populations. Senator Hillary Rodham Clinton has a commanding lead in less-educated counties dominated by whites. Follow the arrows for a more detailed split.



Is a county more than 20 percent black?

NO There are not many African-Americans in this county.

YES This county has a large African-American population.

Obama wins these counties 383 to 70.

And is the high school graduation rate higher than 78 percent?

NO This is a county with less-educated voters.

YES This is a county with more educated voters.

Clinton wins these counties 704 to 89.

And is the high school graduation rate higher than 87 percent?

NO 78 to 87 percent have a diploma.

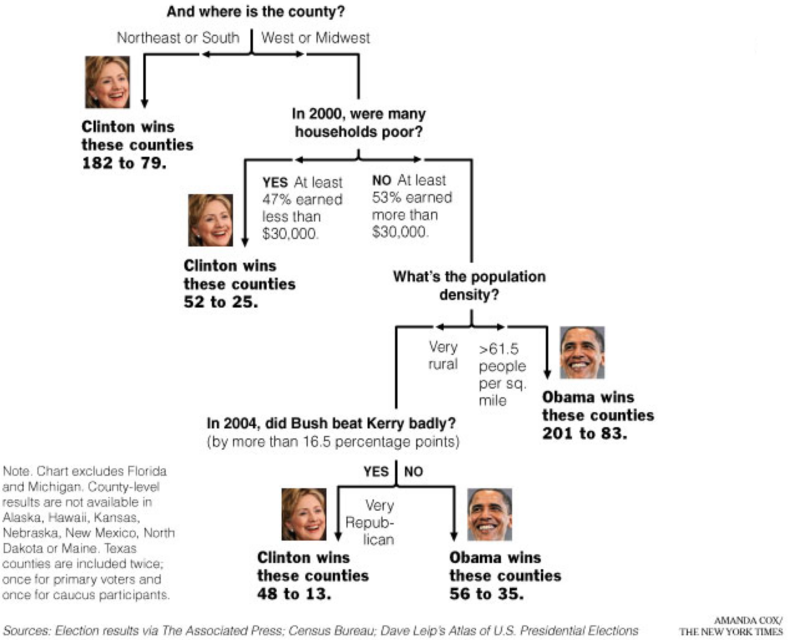
YES This is a highly educated county.

Obama wins these counties 185 to 36.

And where is the county?

Northeast or South

West or Midwest



And where is the county?

Northeast or South | West or Midwest

Clinton wins these counties 182 to 79.

In 2000, were many households poor?

YES At least 47% earned less than $30,000.

NO At least 53% earned more than $30,000.

Clinton wins these counties 52 to 25.

What's the population density?

Very rural | >61.5 people per sq. mile

Obama wins these counties 201 to 83.

In 2004, did Bush beat Kerry badly? (by more than 16.5 percentage points)

YES | NO

Very Repub-lican

Clinton wins these counties 48 to 13.

Obama wins these counties 56 to 35.

Note. Chart excludes Florida and Michigan. County-level results are not available in Alaska, Hawaii, Kansas, Nebraska, New Mexico, North Dakota or Maine. Texas counties are included twice; once for primary voters and once for caucus participants.

Sources: Election results via The Associated Press; Census Bureau; Dave Leip's Atlas of U.S. Presidential Elections

AMANDA COX/
THE NEW YORK TIMES

Decision Trees

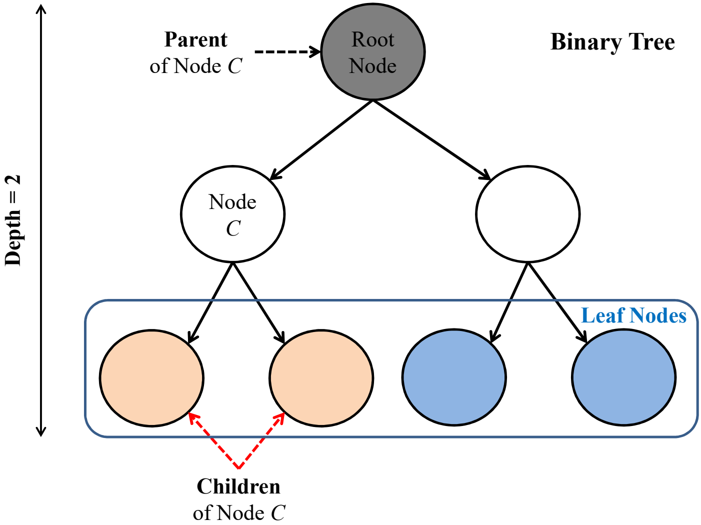
- Tree-based based methods divide the predictor space into different regions, and then fit a model in each region.

- The predictor space can be divided by making successive binary splits on the predictor variables $x_1, \dots, x_p$: we choose a variable x_j, $j = 1, \dots, p$, divide up the predictor space according to $x_j \leq c$ and $x_j > c$, and then proceed on each half.

- A decision tree thus constructed consists of **internal nodes** and **terminal nodes** (**leaves**). Each internal node is a decision to split the predictor space, and the leaves are final predictions. The **size** of a tree is the number of its leaves.

- Decision trees can be applied to both regression and classification problems.

Decision Trees



Regression Tree

Let $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$ be the training data. A regression tree of M leaves can be thought of as dividing the predictor space into M regions $R_1, \ldots, R_M$, each corresponding to a leaf of the tree.

For each R_m , $m = 1, \ldots, M$, let

$$\overline{y}_m = \frac{1}{n_m} \sum_{x_i \in R_m} y_i \tag{1}$$

, where n_m is the number of observations in R_m.

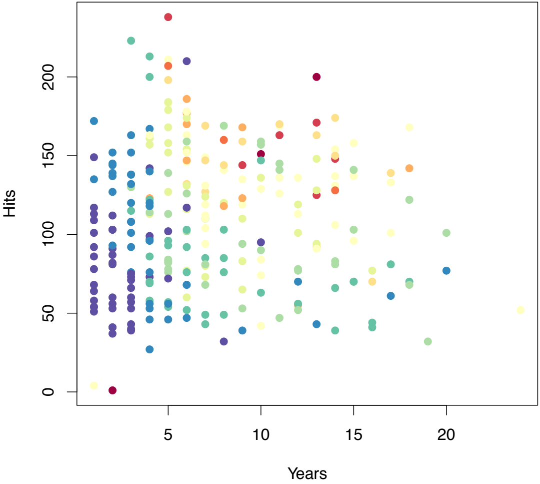
Regression Tree

The estimated regression function is:

$$\widehat{f}(x) = \sum_{m=1}^{M} \overline{y}_m \cdot \mathcal{I}\{x \in R_m\} \tag{2}$$

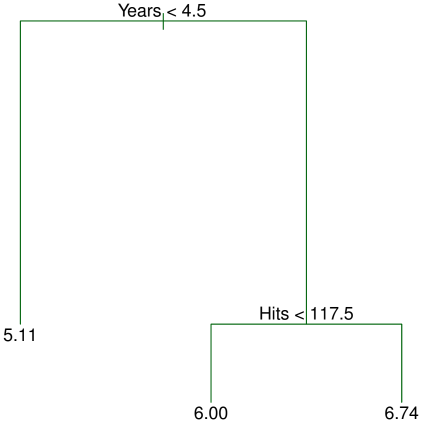
- For every observation that falls into the region R_m, we make the same prediction, which is simply the mean of the response values for the training observations in R_m.
- The result is a piecewise constant model.

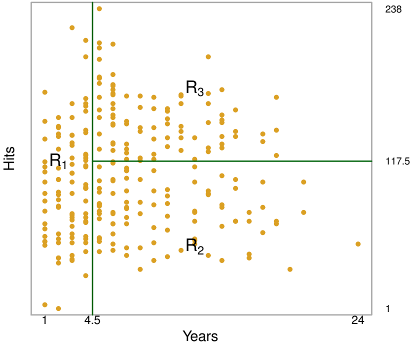
Baseball Salary



Color-coded from low (blue, green) to high (yellow,red)

Baseball Salary





The results suggest that years is the most important factor in determining salary. Among players who have played for more than 4.5 years, the number of hits made (in the previous year) affects salary.

Building a Regression Tree

- The goal is to find $R_1, \ldots, R_M$ that minimize E_{in}[1]. In the regression setting, using the L2 loss, we minimize the RSS:

$$\text{RSS} = \sum_{m=1}^{M} \sum_{x_i \in R_m} (y_i - \overline{y}_m)^2$$

- Unfortunately, it is computationally infeasible to consider every possible partition of the predictor space.

[1] Here we minimize E_{in} first and regularize it later.

The CART Algorithm

The CART algorithm adopts a topdown, greedy approach and divides the predictor space by making successive binary splits. At each step, it chooses the split to achieve the biggest drop in RSS.

- The algorithm begins by dividing the predictor space into two regions. Then it divides one of the two regions[2] further into two regions ... The process continues until a stopping criterion is reached[3].
- This is a greedy approach because at each step of the tree-building process, the optimal split is made at that particular step, rather than looking ahead and picking a split that will lead to a better tree in some future step.

[2] Depending on splitting which would result in the biggest drop in RSS.

[3] For instance, we may continue until no region contains more than 10 observations.

The CART Algorithm

Given a region $R \subseteq \mathbb{R}^p$, the optimal split is choosing an x_j and a split point s, so that the two sub-regions

$$R_{\text{left}} = \{x \in R : x_j < s\} \text{ and } R_{\text{right}} = \{x \in R : x_j \geq s\}$$

are as homogenous in response y as possible.

For regression trees, this means choosing (j, s) to minimize:

$$\sum_{x_i \in R_{\text{left}}} (y_i - \overline{y}_{\text{left}})^2 + \sum_{x_i \in R_{\text{right}}} \left(y_i - \overline{y}_{\text{right}}\right)^2$$

The CART Algorithm

The process generates a large tree T_0. We then prune the tree by choosing, for each value of $\alpha \geq 0$, a subtree $T \subset T_0$ that minimizes

$$\sum_{m=1}^{|T|} \sum_{x_i \in R_m} (y_i - \overline{y}_m)^2 + \alpha |T|$$

, where $|T|$ is the size of T and α is a tuning parameter that controls the tradeoff between the subtree's complexity and its fit to the training data.

- $\alpha = 0 \Rightarrow T = T_0$
- $\alpha \uparrow \Rightarrow |T| \downarrow$
- The optimal α can be selected using cross-validation.

The CART Algorithm

- Intuitively, a small tree might be too simple, while a large tree might overfit the data.

- The strategy of growing a large tree and then pruning it is used because any stopping rule may be short-sighted, in that a split may look bad but it may lead to a good split below it.

The CART Algorithm

Algorithm 8.1 *Building a Regression Tree*

1. Use recursive binary splitting to grow a large tree on the training data, stopping only when each terminal node has fewer than some minimum number of observations.

2. Apply cost complexity pruning to the large tree in order to obtain a sequence of best subtrees, as a function of α.

3. Use K-fold cross-validation to choose α. That is, divide the training observations into K folds. For each $k = 1, \ldots, K$:

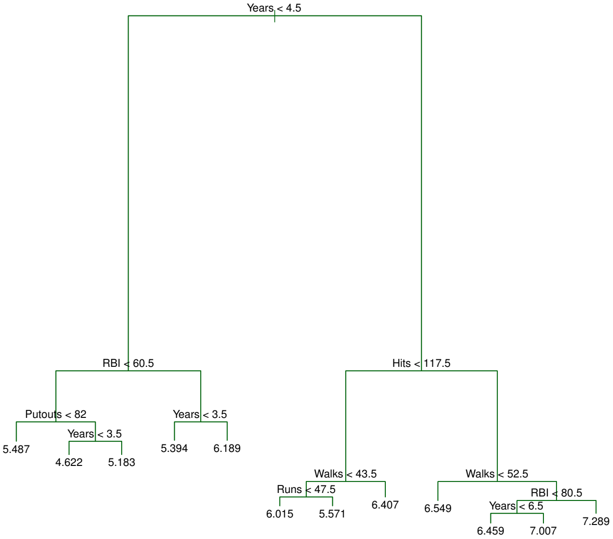
   (a) Repeat Steps 1 and 2 on all but the kth fold of the training data.

   (b) Evaluate the mean squared prediction error on the data in the left-out kth fold, as a function of α.

   Average the results for each value of α, and pick α to minimize the average error.

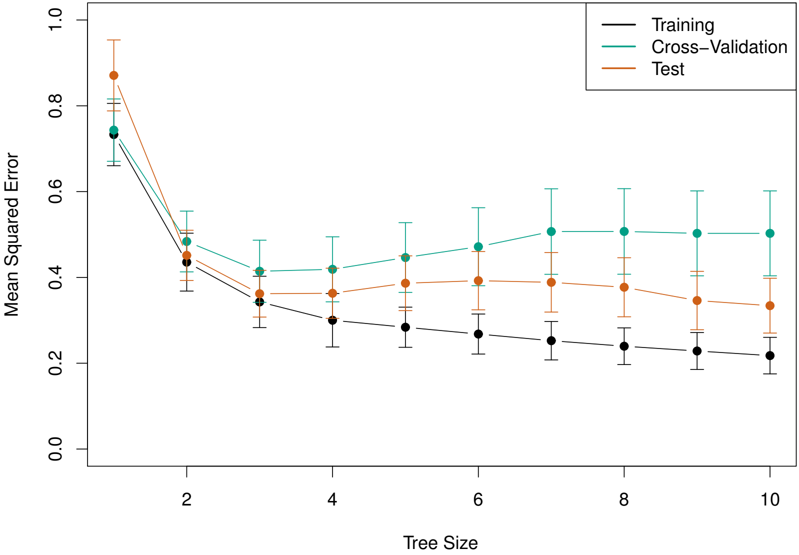
4. Return the subtree from Step 2 that corresponds to the chosen value of α.

Baseball Salary



The unpruned tree

Baseball Salary



Classification Tree

For classification trees, the leaves are class predictions.

Let $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$ be the training data, where $y \in \{1, \ldots, J\}$. For each R_m , $m = 1, \ldots, M$, we assign a class label c_m.

We then classify a new point $x \in \mathbb{R}^p$ by

$$\widehat{f}(x) = \sum_{m=1}^{M} c_m \cdot \mathcal{I}\{x \in R_m\} \tag{3}$$

Classification Tree

Let $p_j^m \equiv \Pr\left(y = j \middle| x \in R_m \right)$ be the class probabilities in R_m. p_j^m can be estimated by

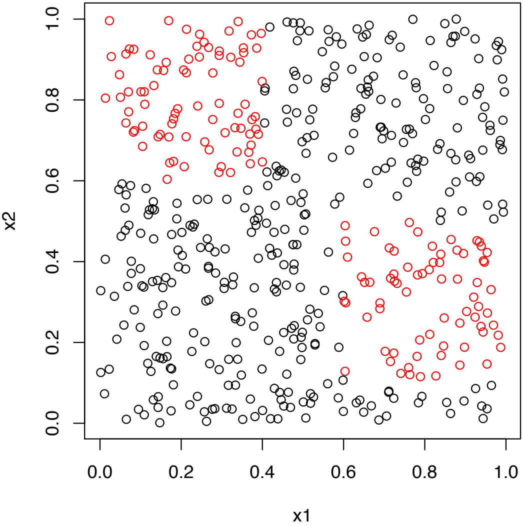
$$\widehat{p}_j^m = \frac{1}{n_m} \sum_{x_i \in R_m} \mathcal{I}\left\{ y_i = j \right\} \tag{4}$$

Then

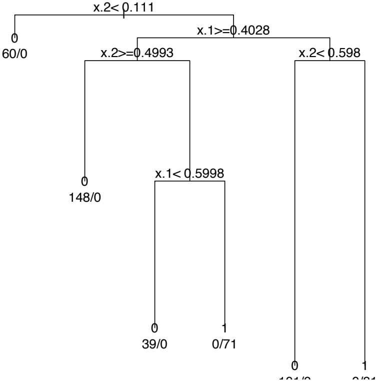
$$c_m = \arg\max_j \widehat{p}_j^m \tag{5}$$

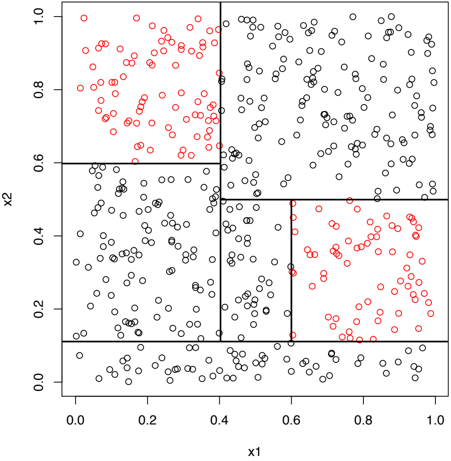
, i.e., c_m is the most common occurring class in R_m.

Classification Tree



Classification Tree





Building a Classification Tree

As in building regression trees, we first find $R_1, \ldots, R_M$ that minimize E_{in}. Let $Q_m = E_{in}(R_m)$. Then at each step, the CART algorithm chooses the optimal split to minimize:

$$n_{\text{left}} Q_{\text{left}} + n_{\text{right}} Q_{\text{right}}$$

For classification trees, Q_m is also called the **node impurity** measure.

Node Impurity Measures

Commonly used node impurity measures are:

Misclassification rate:

$$Q_m = 1 - \widehat{p}^m_{c_m}$$

, where $\widehat{p}^m_{c_m}$ is the proportion of observations in R_m that belong to class c_m.

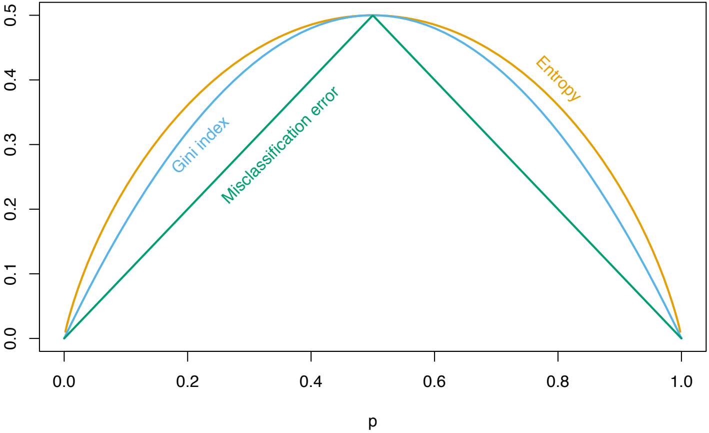
Gini index:

$$Q_m = \sum_{j=1}^{J} \widehat{p}^m_j \left(1 - \widehat{p}^m_j\right)$$

Cross-entropy:

$$Q_m = -\sum_{j=1}^{J} \widehat{p}^m_j \log \widehat{p}^m_j$$

Node Impurity Measures



Node impurity measures for binary classification (cross-entropy has been scaled to pass through $(0.5, 0.5)$). Note that the Gini index and cross-entropy are differentiable and thus easier to optimize numerically.

NBC Shows

Data from NBC on 40 TV pilots. Data on each show include its genre, ratings, and viewer demographics. The demographics information contain over 50 variables including viewer location, household size, race, income, occupation, home ownership, etc.

```r
nbc <- read.csv("nbc_showdetails.csv")
demo <- read.csv("nbc_demographics.csv")
```

NBC Shows

There are three genres: {Drama/Adventure, Reality, Situation Comedy}. To predict genre based on demographics:

```
###################################
# Multinomial Logistic Regression #
###################################
require(nnet)
require(AER)
y <- nbc$Genre
demo <- demo[,-1]
logitfit <- multinom(y~.,data=demo)
result <- coeftest(logitfit)
```

NBC Shows

```
result[1:5,]
```

```
##                                   Estimate Std. Error   z value Pr(>|z|)
## Reality:(Intercept)                 0.0143       5.85  0.002447    0.998
## Reality:TERRITORY.EAST.CENTRAL      6.6040    2235.83  0.002954    0.998
## Reality:TERRITORY.NORTHEAST         1.5089    1404.82  0.001074    0.999
## Reality:TERRITORY.PACIFIC          -2.1562    1776.01 -0.001214    0.999
## Reality:TERRITORY.SOUTHEAST         0.6641    1910.97  0.000348    1.000
```

None of the coefficient estimates are significant ...

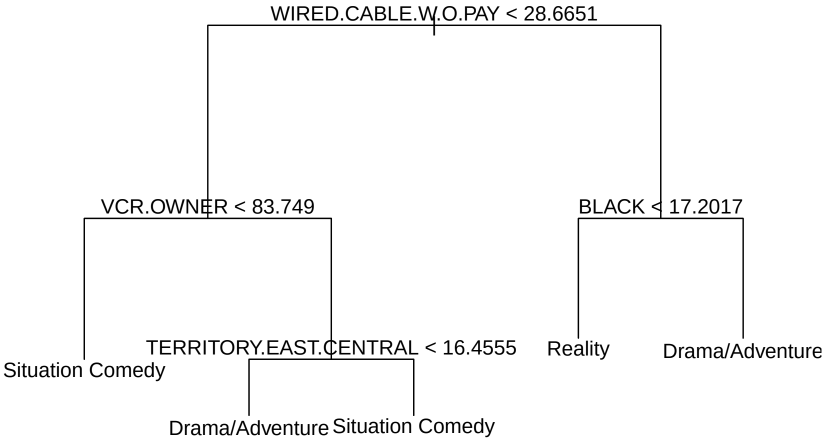
```
result[result[,4]<=0.05,]
```

```
##      Estimate Std. Error z value Pr(>|z|)
```

NBC Shows

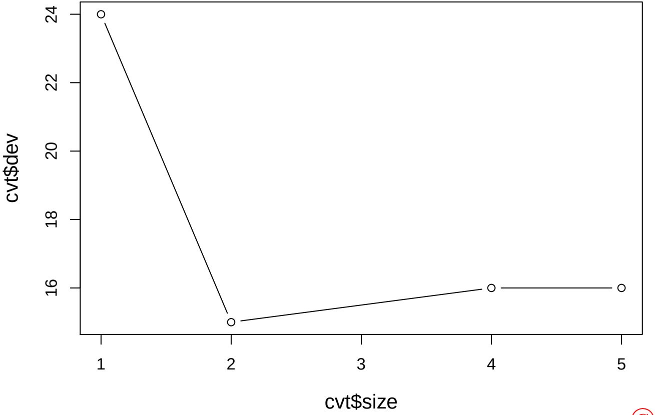
```
#######################
# Classification Tree #
#######################
require(tree)
t0 <- tree(y~.,data=demo,mincut=1) #mincut: minimum size for child nodes
summary(t0)

##
## Classification tree:
## tree(formula = y ~ ., data = demo, mincut = 1)
## Variables actually used in tree construction:
## [1] "WIRED.CABLE.W.O.PAY"    "VCR.OWNER"
## [3] "TERRITORY.EAST.CENTRAL" "BLACK"
## Number of terminal nodes:  5
## Residual mean deviance:  0.192 = 6.73 / 35
## Misclassification error rate: 0.05 = 2 / 40
```

NBC Shows

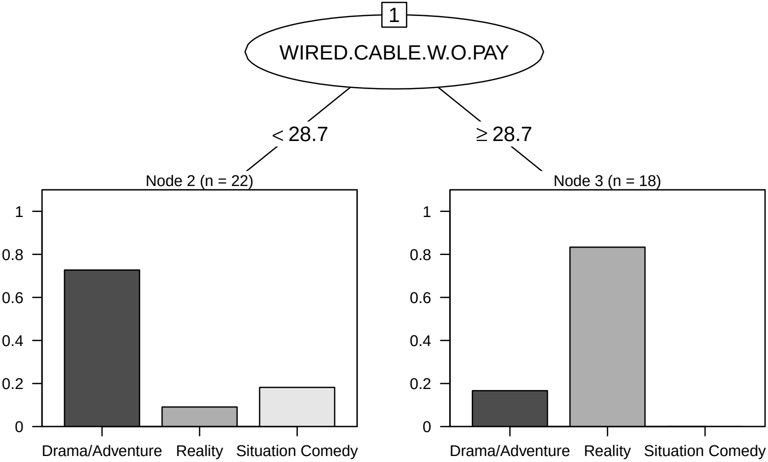

WIRED.CABLE.W.O.PAY < 28.6651
VCR.OWNER < 83.749
BLACK < 17.2017
Situation Comedy
TERRITORY.EAST.CENTRAL < 16.4555
Reality
Drama/Adventure
Drama/Adventure
Situation Comedy

NBC Shows

```
# To prune the tree, use CV to select the best tuning parameter
cvt <- cv.tree(t0,FUN=prune.misclass) #use misclassification rate
                                      #for pruning
t1 = prune.misclass(t0,best=cvt$size[which.min(cvt$dev)])
```



NBC Shows


1
WIRED.CABLE.W.O.PAY
< 28.7
≥ 28.7
Node 2 (n = 22)
Node 3 (n = 18)
1
0.8
0.6
0.4
0.2
0
Drama/Adventure
Reality
Situation Comedy

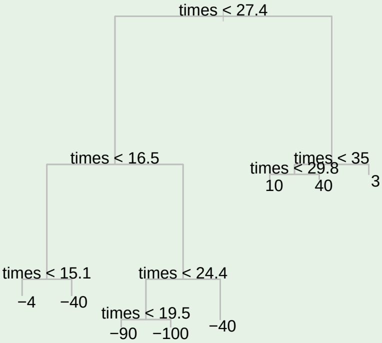
Advantages and Disadvantages of Trees

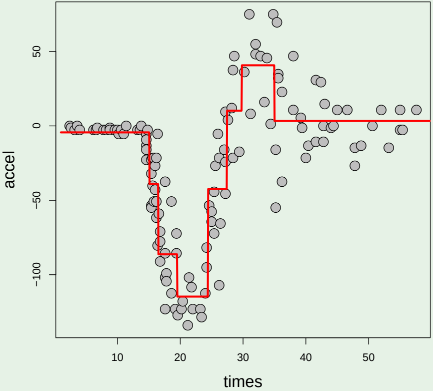
Pros:

- Highly interpretable
- Automatically detecting nonlinear relationships
- Automatically modeling interactions

Advantages and Disadvantages of Trees

Motorcycle crash test dummy data





x : time from impact; y : head acceleration

Advantages and Disadvantages of Trees

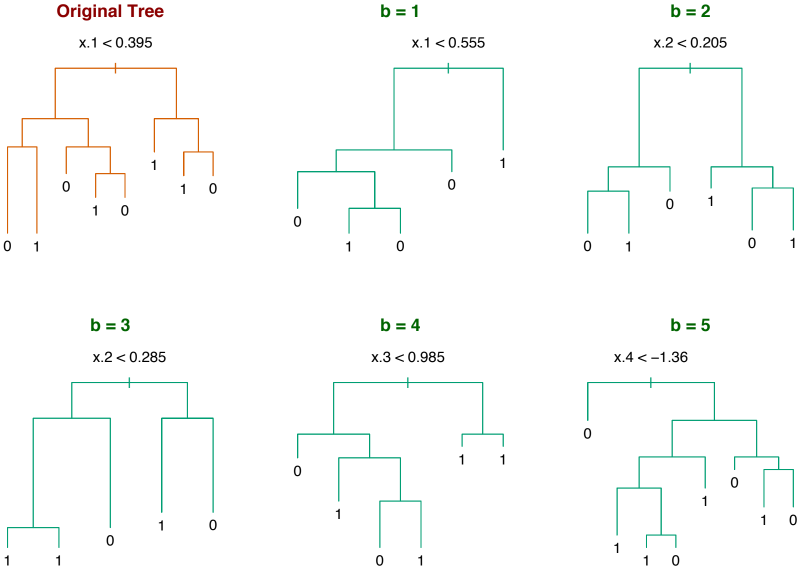
Cons:

- Relatively poor predictive performance
  - Trees generally suffer from high variance and are unstable: a small change in the observed data often results in a completely different tree.
  - This is due to their hierarchical nature: once a split is made, all the splits under it are changed as well.
- Difficulty in capturing simple relationships
  - Trees require a large number of parameters (splits) to capture simple linear and additive relationships.
  - In other words, the weaknesses of tree-based methods are the strengths of linear models, and vice versa.

Bagging

- **Bootstrap aggregation**, or **bagging**, is a general-purpose procedure for reducing the variance of a statistical learning method.
- We know that given a set of N independent random variables each with variance σ^2, the variance of the mean is $\frac{\sigma^2}{N}$.
- Hence if we can average the predictions made on many independent training data sets drawn from the population, then we can reduce the variance and hence increase the prediction accuracy of our method.
- In practice, we do not have access to multiple independent training data sets from the population. Instead, we can draw multiple samples from the empirical distribution using bootstrap.

Bagging



Original Tree

b = 1

b = 2

b = 3

b = 4

b = 5

Bagging for Regression

Let $b = 1, \ldots, B$ index the bootstrap samples drawn from the training data. We fit our model on each bootstrap sample.

For regression problems, let $\widehat{f}^{(b)}(x)$ be the model's prediction trained on sample b. We then average all the predictions to obtain

$$\widehat{f}(x) = \frac{1}{B} \sum_{b=1}^{B} \widehat{f}^{(b)}(x) \tag{6}$$

Boston Housing Price

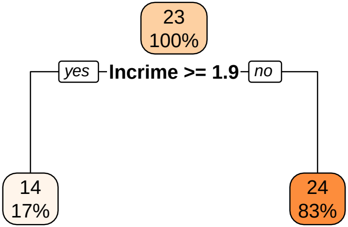
Data: median value of owner-occupied homes in 506 census tracts in the Boston area from the 1970 census. Other variables include average rooms per house, distance to employment centers, crime rate, percentage of population with lower socioeconomic status, etc.

```
require(MASS) # constains the data set 'Boston'
require(rpart)
head(Boston,3)

##      crim zn indus chas   nox   rm  age  dis rad tax ptratio black lstat
## 1 0.00632 18  2.31    0 0.538 6.58 65.2 4.09   1 296    15.3   397  4.98
## 2 0.02731  0  7.07    0 0.469 6.42 78.9 4.97   2 242    17.8   397  9.14
## 3 0.02729  0  7.07    0 0.469 7.18 61.1 4.97   2 242    17.8   393  4.03
##   medv
## 1 24.0
## 2 21.6
## 3 34.7
```

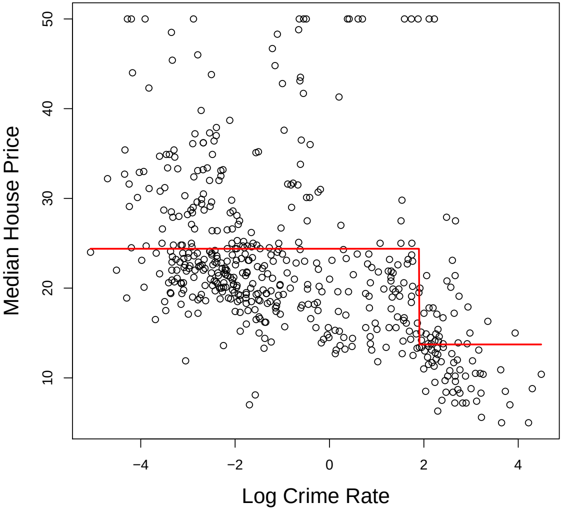
Boston Housing Price

Let's first predict median house prices based on crime rates only, using a single decision stump – the simplest tree with a single split at root.

```
# Fitting a single strump
n = nrow(Boston)
Boston$lncrime = log(Boston$crim)
fit = rpart(medv~lncrime,data=Boston,control=rpart.control(maxdepth=1))
```



Boston Housing Price

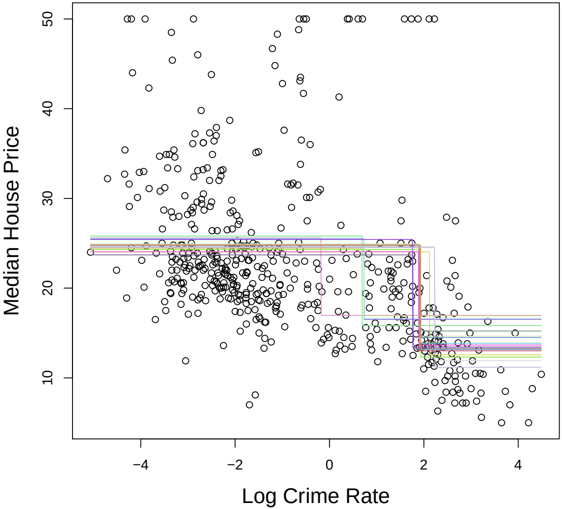


Boston Housing Price

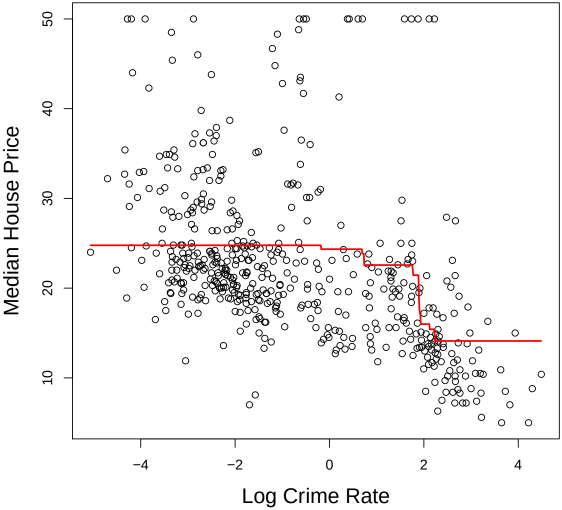
Now let's perform bagging using $B = 20$ bootstrap samples and make predictions of housing price on a range of crime rate values:

```r
crimelims = range(Boston$lncrime)
crime.grid = seq(crimelims[1],crimelims[2],0.01)
#
# Function to fit a single stump
stump = function(index,b){
  fit = rpart(medv~lncrime,data=Boston,subset=indx,
              control=rpart.control(maxdepth=1))
  return(predict(fit,newdata=list(lncrime=crime.grid)))
}
# Bagging
B = 20
yhat_bag = 0
for (b in 1:B){
  bootsample = sample(n,n,replace=T)
  yhat_bag = yhat_bag + stump(bootsample,b)/B
}
```

Boston Housing Price



Boston Housing Price



Bagging for Classification

For classification problems, let $\widehat{p}_j^{(b)}(x)$ be the class probabilities estimated on bootstrap sample b. We have:

$$\widehat{f}(x) = \underset{j \in \{1,\ldots,J\}}{\arg\max}\, \widehat{p}_j(x) \tag{7}$$

, where

$$\widehat{p}_j(x) = \frac{1}{B} \sum_{b=1}^{B} \widehat{p}_j^{(b)}(x)$$

This is the *probability approach*.

Bagging for Classification

Alternatively, for classification problems, let

$$\widehat{f}(x) = \underset{j \in \{1,\ldots,J\}}{\arg\max} \sum_{b=1}^{B} \mathcal{I}\left(\widehat{f}^{(b)}(x) = j\right) \tag{8}$$

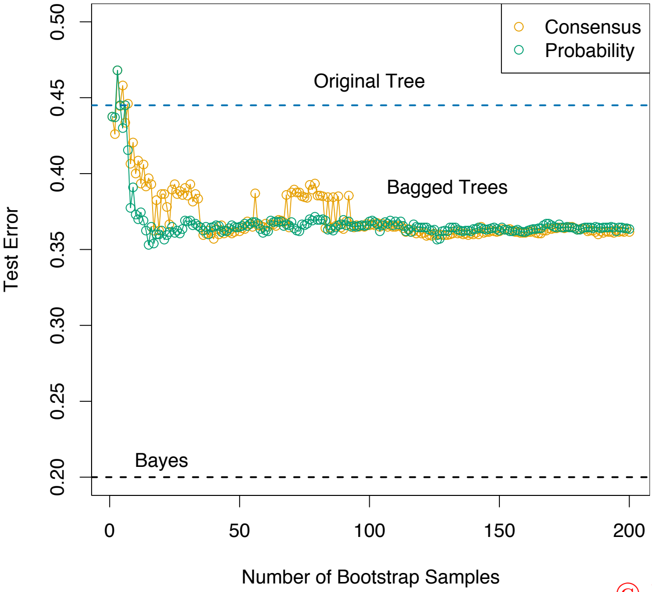
This is the *consensus approach* – decision by “majority vote”.

Bagging for Classification

Simulation: $N = 30, p = 5, J = 2$. Each predictor has a standard Gaussian distribution with pairwise correlation 0.95. y is generated according to $p\left(y = 1 \,|x_1 \leq 0.5\right) = 0.2$ and $p\left(y = 1 \,|x_1 > 0.5\right) = 0.8$.

Bagging is performed by fitting classification trees (with no pruning) to 200 bootstrap samples.

Bagging for Classification



The Wisdom of Crowds

Consider a binary classification problem where $y \in \{\mathsf{F}, \mathsf{T}\}$. Suppose that for a given input x, the true value of y is T. We have B independent classifiers, $f^{(b)}(x)$, $b = 1, \dots, B$, and each has a misclassification rate of 0.4, i.e., $\Pr\left(f^{(b)}(x) = \mathsf{F}\right) = 0.4$.

Let $\text{✌} = \sum_{b=1}^{B} \mathcal{I}\left(f^{(b)}(x) = \mathsf{T}\right)$ be the total number of "votes" for $y = \mathsf{T}$ among the B classifiers. Then

$$\text{✌} \sim \text{Binomial}\,(B, 0.6)$$

The bagged classifier (consensus approach) is

$$f^{\text{bag}}(x) = \underset{j \in \{\mathsf{F},\mathsf{T}\}}{\arg\max} \sum_{b=1}^{B} \mathcal{I}\left(f^{(b)}(x) = j\right)$$

The Wisdom of Crowds

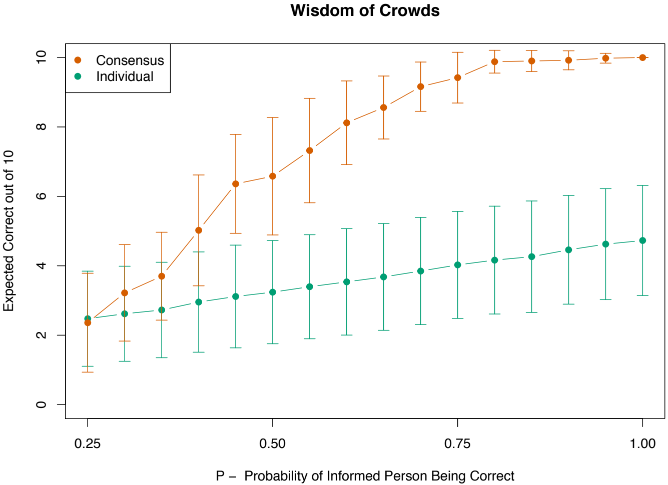
Hence the misclassification rate of the bagged classifier is

$$\Pr\left(f^{\text{bag}}(x) = \text{F}\right) = \Pr\left(\text{✌} < \frac{B}{2}\right) \to 0 \quad \text{as } B \to \infty \tag{9}$$

- (9) is true in this example as long as the misclassification rate of the individual classifier is less than 0.5. If $\Pr\left(f^{(b)}(x) = \text{F}\right) > 0.5$, then $f^{\text{bag}}(x)$ will become perfectly inaccurate as $B \to \infty$.

- Bagging a good classifier can improve predictive accuracy, but bagging a bad one hurts.

The Wisdom of Crowds



50 members vote in 10 categories, each with 4 nominations. For each category, only 15 (randomly selected) voters are informed (probability of selecting the "correct" candidate > 0.25).

Bagging

- Bagging can dramatically reduce the variance and stabilize the predictions of unstable procedures, and is therefore particularly useful for high-variance, low-bias procedures.

- Trees are ideal candidates for bagging, since they can capture complex interaction structures in the data, and if grown sufficiently deep, have relatively low bias.

- Real structure that persists across data sets shows up in the average. Noisy useless signals will average out to have no effect.

- This technique of model averaging is central to many advanced nonparametric learning algorithms.

- Downside? Loss of interpretability. *A bagged tree is no longer a tree*. Bagging improves prediction accuracy at the expense of interpretability.

Random Forests

- **Random forests** improve on bagging by reducing the correlation between the sampled trees.

- We know that given a set of N identically distributed random variables with variance σ^2 and positive pairwise correlation ρ, the variance of the mean is $\rho\sigma^2 + (1-\rho)\frac{\sigma^2}{N}$.

- Hence, in the context of decision trees, we can improve the variance reduction of bagging by reducing the correlation between the trees, without increasing the variance too much.

- This is achieved in random forests through random selection of the input variables in the tree-growing process.

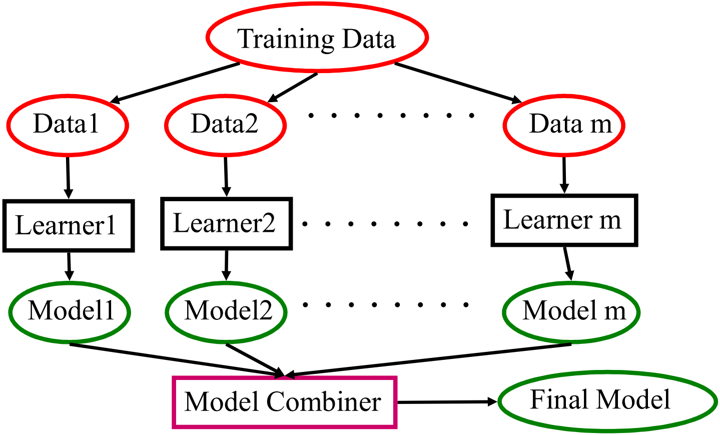
Random Forests

- As in bagging, random forests build a number of trees on bootstrapped samples.
- When building these trees, each time a split in a tree is considered, a random selection of m ($m < p$) predictors is chosen as split candidates from the full set of p predictors. The split is allowed to use only one of these m predictors.
- A fresh selection of m predictors is taken at each split.
  - Typical choice of m: $m = \sqrt{p}$.

Ensemble Learning

- Bagging and random forests are examples of **ensemble methods**.
- Instead of learning a single hypothesis from a hypothesis set, ensemble methods select a collection (ensemble) of hypotheses and combine their predictions.
- Doing so often improves prediction accuracy at the expense of interpretability.

Ensemble Learning

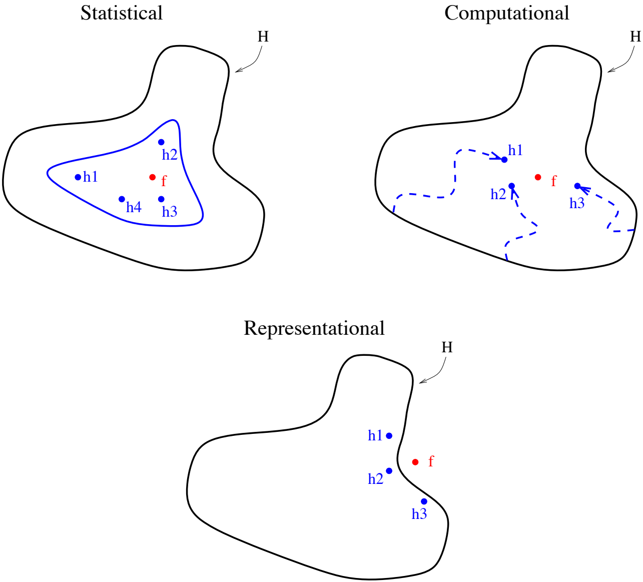


Ensemble Learning

Three reasons why ensemble methods may work well:

1. Statistical: for a given hypothesis set $\mathcal{H}$, when sample size is small, many different hypotheses in $\mathcal{H}$ can all give the same accuracy on the training data. Combining their predictions helps lower the variance and improve prediction accuracy.
2. Computational: combining local optima produced by local search
3. Representational: by forming weighted sums of hypotheses drawn from $\mathcal{H}$, it might be possible to expand $\mathcal{H}$ and produce a better approximation to target function f.

Ensemble Learning


Statistical

Computational

Representational

Boosting

- Like bagging, boosting is a general and powerful tool that can be applied to many statistical learning methods for regression or classification.
- While bagging fits a model (**base learner**) independently to multiple bootstrap samples, boosting fits a simple model (**weak learner**) to the training data sequentially to construct a more complex model (**strong learner**).
- Basic idea: in each round, the training data are re-weighted according to the performance of the model in the previous round. Observations that were not accurately predicted see their weights increase. The model is then fit to the re-weighted data. In this way, we slowly improve the performance of the model in areas where it does not perform well.

L2 Boost (Regression)

1. Set $\widehat{f}(x) = 0$ and $r_i = y_i \; \forall i$
2. For $b = 1, \ldots, B$:
   1. Fit a regression model $\widehat{f}^{(b)}$ to the training data $\{(x_1, r_1), \ldots, (x_N, r_N)\}$.
   2. Update $\widehat{f}$ by adding in a *shrunken* version of $\widehat{f}^{(b)}$:

      $$\widehat{f}(x) \leftarrow \widehat{f}(x) + \lambda \widehat{f}^{(b)}(x)$$

   3. Update the residuals:

      $$r_i \leftarrow r_i - \lambda \widehat{f}^{(b)}(x_i)$$

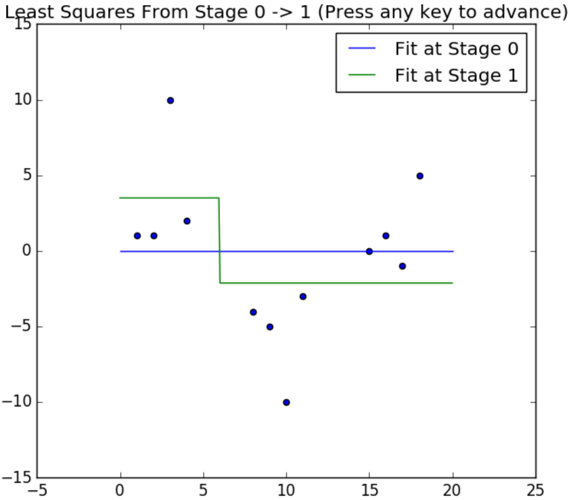
3. Output

$$\widehat{f}(x) = \sum_{b=1}^{B} \lambda \widehat{f}^{(b)}(x)$$

Boosting for Regression

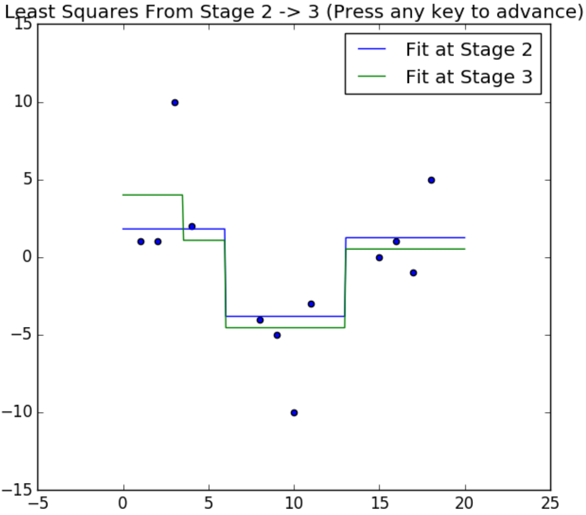
- When applying boosting to regression algorithms, in each round b, we simply fit the base learner, $\widehat{f}^{(b)}$, to the residuals of $\widehat{f}^{(b-1)}$. This is akin to re-weighting the training data: observations that were fit well in the previous round see their importance diminished (r_i small), while those that were not fit well become more important (r_i large) in this round.
- The base learner $\widehat{f}^{(b)}$ is usually a *weak learner*, e.g., a stump.
- The shrinkage parameter λ is a small positive number (typical values: 0.01, 0.001, etc.). The boosting approach is to learn *slowly* and λ controls the rate of learning.

Boosting for Regression



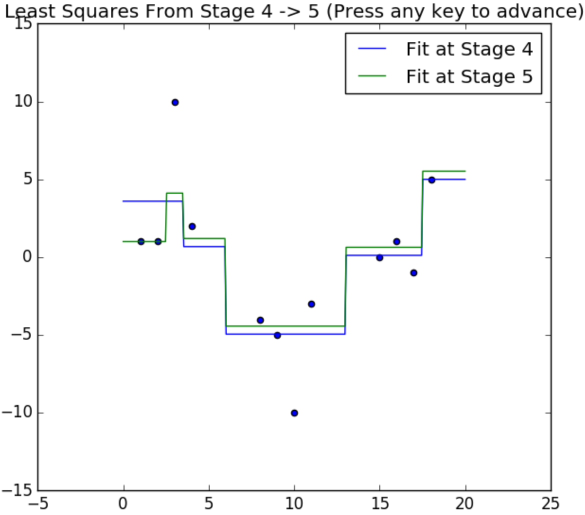
L2 Boosting with Stumps

Boosting for Regression



L2 Boosting with Stumps

Boosting for Regression



L2 Boosting with Stumps

AdaBoost (Binary Classification)

1. Let $y \in \{-1, 1\}$. Initialize $w_i = \frac{1}{N}$ $\forall i$.

2. For $b = 1, \dots, B$:

    1. Fit a classifier $\widehat{f}^{(b)}$ to the training data by minimizing the weighted error

    $$\frac{\sum_{i=1}^{N} w_i \mathcal{I}\left(y_i \neq f^{(b)}\left(x_i\right)\right)}{\sum_{i=1}^{N} w_i}$$

    2. Let $\alpha_b = \log\left(\left(1 - \epsilon_b\right) / \epsilon_b\right)$, where ϵ_b is the weighted error of $\widehat{f}^{(b)}$, and update w_i as follows:

    $$w_i \leftarrow w_i \exp\left(\alpha_b \mathcal{I}\left(y_i \neq \widehat{f}^{(b)}\left(x_i\right)\right)\right)$$

3. Output

$$\widehat{f}\left(x\right) = \operatorname{sign}\left(\sum_{b=1}^{B} \alpha_b \widehat{f}^{(b)}\left(x\right)\right)$$

Boosting for Classification

- For classification problems, the boosted classifier is a weighted sum of individual classifiers, with weights proportional to each classifier's accuracy on the training set (good classifiers get more weight).

- In AdaBoost, If an individual classifier has accuracy $< 50\%$ ($\epsilon_b > 0.5$), we flip the sign of its predictions and turn it into a classifier with accuracy $> 50\%$. This is achieved by making $\alpha_b < 0$ so that the classifier enters negatively into the final hypothesis.

Boosting for Classification

- In each round of the boosting process, we re-weight the observations in the training data: those that were misclassified in the previous round[4] see their weights increase relative to those that were correctly classified. In this way, successive classifiers are forced to place greater emphasis on points that have been misclassified by previous classifiers, and data points that continue to be misclassified by successive classifiers receive ever greater weight[5].

[4] After classifiers with $\epsilon_b > 0.5$ are "flipped".

[5] This adaptive data weighting scheme gives the algorithm its name: AdaBoost stands for "Adaptive Boosting".

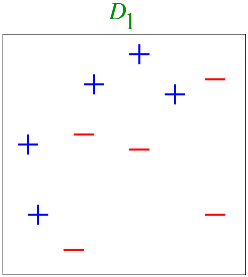
Boosting for Classification

Now suppose we use decision tree as the base classifier. To fit classification trees to weighted data, we calculate the weighted $\widehat{p}_j^m$ for each region:

$$\widehat{p}_j^m = \frac{\sum_{x_i \in R_m} w_i \mathcal{I}\left(y_i = j\right)}{\sum_{x_i \in R_m} w_i} \tag{10}$$

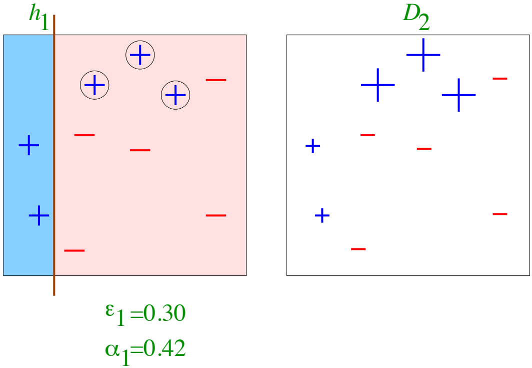
and let (10) replace (4) in the CART algorithm.

Boosting for Classification



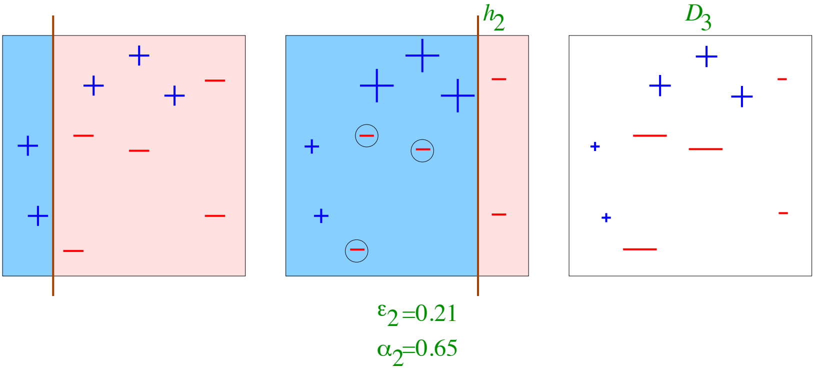
Weak classifiers = vertical or horizontal half-planes

Boosting for Classification



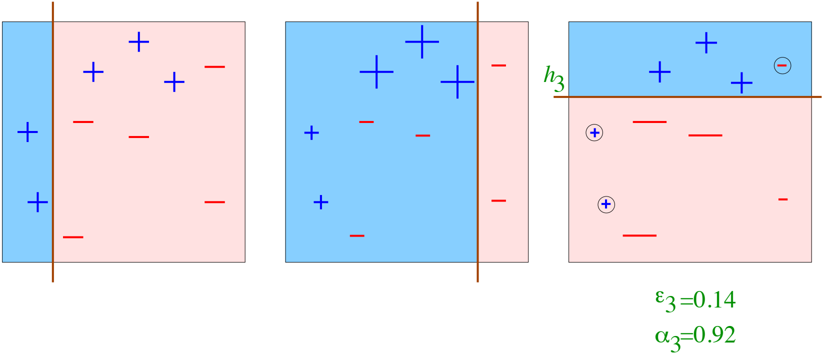
Round 1

Boosting for Classification



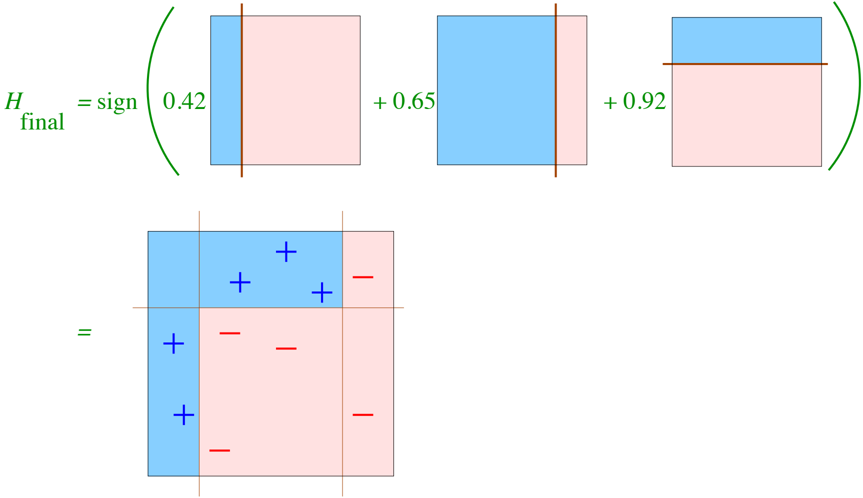
Round 2

Boosting for Classification



Round 3

Boosting for Classification



Final Classifier

Boosting

In essence, boosting is a forward stepwise algorithm to fit an adaptive additive model:

$$f(x) = \alpha_1 \phi(x; \beta_1) + \cdots + \alpha_M \phi(x; \beta_M) \tag{11}$$

(11) is equivalent in functional form to a linear basis function model. The difference is that here the basis functions are *base learners* (e.g., decision trees) whose functional form are not chosen before seeing the data but are *learned* from the data – they are called **adaptive basis functions**.

Boosting

Fitting (11) requires minimizing the in-sample error:

$$E_{in}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \ell(y_i, f(x_i))$$

, where $\theta = \{\alpha_m, \beta_m\}_{m=1}^{M}$.

- L2 loss: $\ell(y, f(x)) = (y - f(x))^2 \Rightarrow$ L2 Boost
- Exponential loss: $\ell(y, f(x)) = \exp(-y \cdot f(x)) \Rightarrow$ AdaBoost[6]

[6] See Appendix for why AdaBoost corresponds to forward stepwise additive modeling with exponential loss.

Boosting

Instead of doing a global minimization, the boosting strategy is to follow a forward stepwise procedure by adding basis functions *one by one*[7]:

- Choose the $\phi(x;\beta_1)$ and α_1 that gives the smallest in-sample error. Let $\widehat{f}(x) = \widehat{\alpha}_1 \phi\left(x; \widehat{\beta}_1\right)$.
- Choose the $\phi(x;\beta_2)$ and α_2 that gives the largest *additional* reduction in in-sample error:

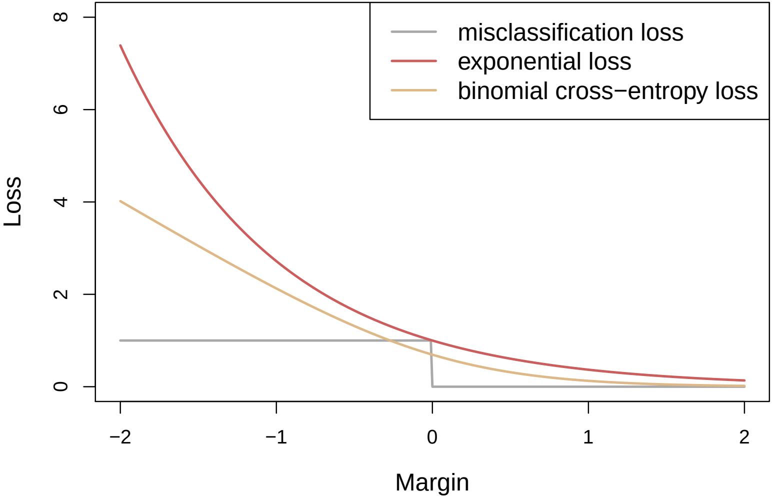
$$\left(\widehat{\alpha}_2, \widehat{\beta}_2\right) = \underset{\alpha_2,\beta_2}{\arg\min} \sum_{i=1}^{N} \ell\left(y_i, \widehat{f}(x_i) + \alpha_2 \phi(x_i;\beta_2)\right)$$

  Let $\widehat{f}(x) = \widehat{\alpha}_1 \phi\left(x; \widehat{\beta}_1\right) + \widehat{\alpha}_2 \phi\left(x; \widehat{\beta}_2\right)$.

- And continue ...

[7] Similar to forward stepwise linear regression.

Boosting


misclassification loss
exponential loss
binomial cross−entropy loss
Loss
0
2
4
6
8
−2
−1
0
1
2
Margin

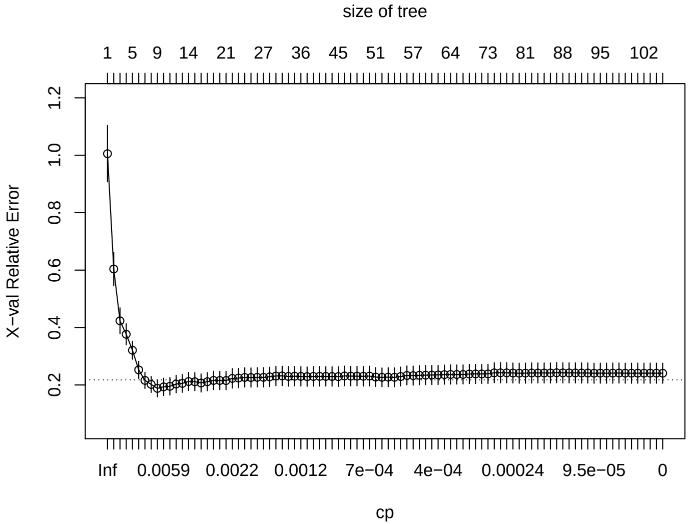
Boston Housing Price

Now let's predict median house prices based on all variables in the data set:

```r
train = sample(nrow(Boston),nrow(Boston)*0.7) # create training & test set

#################
# Decision Tree #
#################
# Step 1: grow tree until stopping criteria:
#         cp: cost complexity parameter; any split must lower in-sample
#             error by a factor of cp in order to continue grow
#         minbucket: minimum number of observations in any terminal node
fit0 = rpart(medv~.,data=Boston,subset=train,
             control=rpart.control(cp=0,minbucket=2))
```

Boston Housing Price

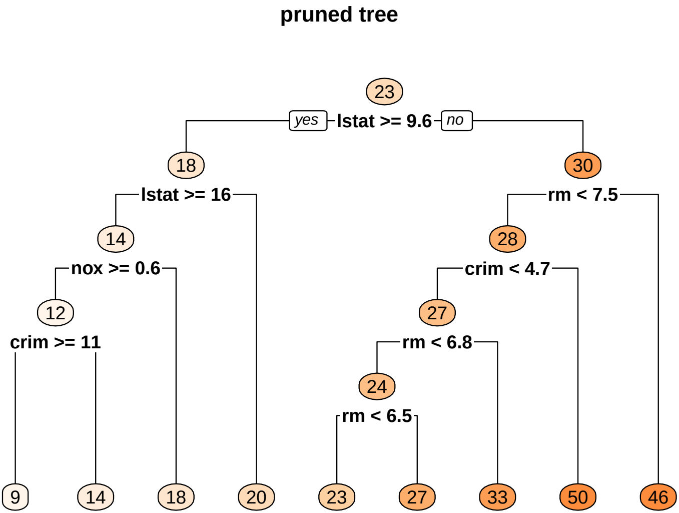
```
plotcp(fit0)
```



Boston Housing Price

```r
# Step 2: prune the tree
# select the cost-complexity parameter ("cp") with the smallest
# (relative) cross-validation error ("xerror")
fit = prune(fit0,cp=fit0$cptable[which.min(fit0$cptable[,"xerror"]),"CP"])
#
# Compute test error
y = Boston[-train,"medv"] #true medv on test data
yhat = predict(fit,newdata=Boston[-train,])
testErr = mean((yhat-y)^2)
testErr

## [1] 24.7
```

Boston Housing Price



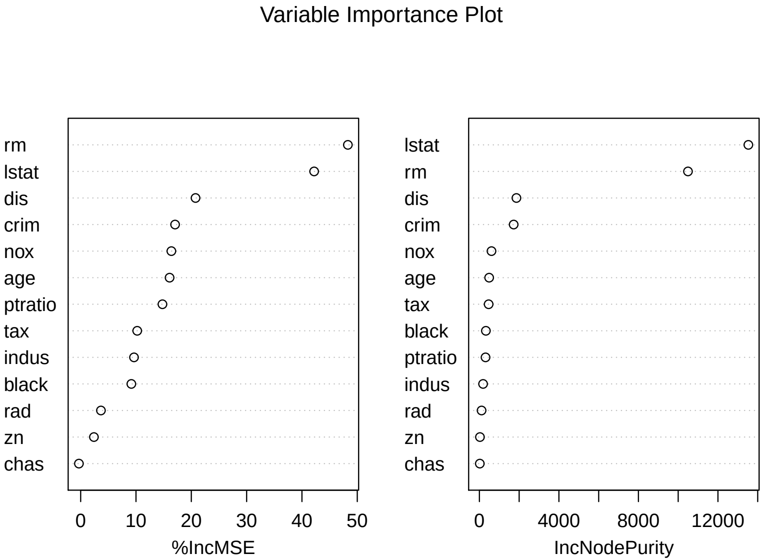
Boston Housing Price

```
###########
# Bagging #
###########
require(randomForest)
# mtry: number of  predictors considered for each split
# bagging is a special case of a random forest with mtry = all predictors
fit = randomForest(medv~.,data=Boston,subset=train,
                   mtry=13,importance=TRUE)
fit

##
## Call:
##  randomForest(formula = medv ~ ., data = Boston, mtry = 13, importance =
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 13
##
##           Mean of squared residuals: 10.2
##                     % Var explained: 88.2
```

Boston Housing Price

```r
# Compute test error
y = Boston[-train,"medv"]
yhat = predict(fit,newdata=Boston[-train,])
testErr = mean((yhat-y)^2)
testErr

## [1] 11.3
```

Boston Housing Price



Boston Housing Price

```r
#################
# Random Forest #
#################
fit = randomForest(medv~.,data=Boston,subset=train,mtry=5,importance=TRUE)
fit

##
## Call:
##  randomForest(formula = medv ~ ., data = Boston, mtry = 5, importance =
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 5
##
##           Mean of squared residuals: 11.1
##                     % Var explained: 87.2
```

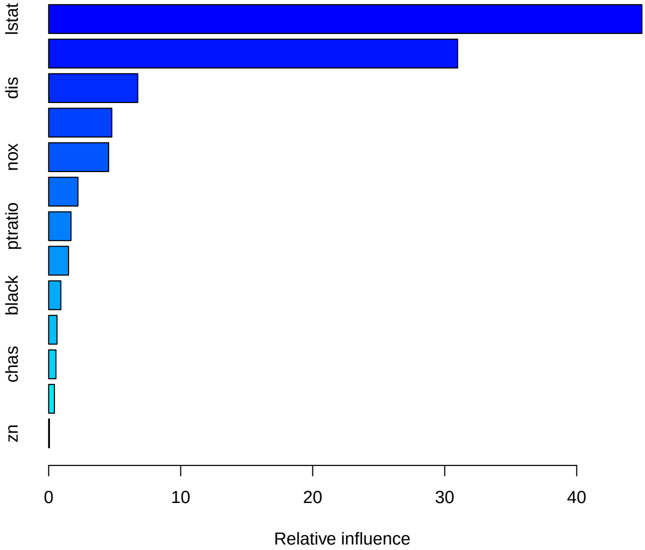
Boston Housing Price

```
# Compute test error
y = Boston[-train,"medv"]
yhat = predict(fit,newdata=Boston[-train,])
testErr = mean((yhat-y)^2)
testErr

## [1] 7.88
```

Boston Housing Price

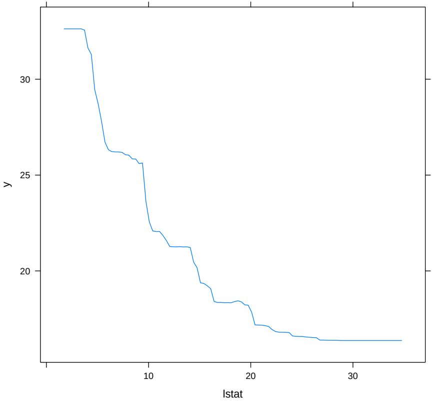
```r
############
# Boosting #
############
require(gbm)
fit = gbm(medv~.,data=Boston[train,],distribution="gaussian",
          n.trees=5000,interaction.depth=5,shrinkage=0.001)
#
# Compute test error
y = Boston[-train,"medv"]
yhat = predict(fit,newdata=Boston[-train,],n.trees=5000)
testErr = mean((yhat-y)^2)
testErr

## [1] 9.57
```

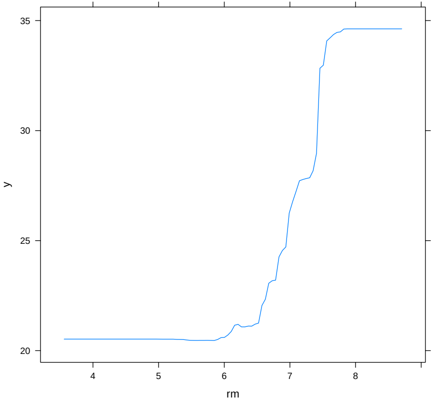
Boston Housing Price



Boston Housing Price

Partial dependence plots illustrate the marginal effect of the selected variables after integrating out the other variables:





Appendix: AdaBoost as Forward Stepwise Modeling

Let $y \in \{-1, 1\}$. Start with the forward stepwise procedure outlined on page 76, using the exponential loss function $\ell(y, f(x)) = \exp(-yf(x))$.

In step m, given $\widehat{f}^{(m-1)}(x)$ – the result of the previous $m-1$ steps, the goal is find

$$\left(\widehat{\alpha}_m, \widehat{\phi}_m\right) = \arg\min_{\alpha,\phi} \sum_{i=1}^{N} \exp\left[-y_i\left(\widehat{f}^{(m-1)}(x) + \alpha\phi(x_i)\right)\right] \quad (12)$$

$$= \arg\min_{\alpha,\phi} \sum_{i=1}^{N} w_i^{(m)} \exp\left[-\alpha y_i \phi(x_i)\right] \quad (13)$$

, where $w_i^{(m)} = \exp\left(-y_i \widehat{f}^{(m-1)}(x)\right)$.

Appendix: AdaBoost as Forward Stepwise Modeling

Minimizing (13) $\Rightarrow$

$$\widehat{\alpha}_m = \frac{1}{2}\log\frac{1-\epsilon_m}{\epsilon_m}$$

, where

$$\epsilon_m = \frac{\sum_{i=1}^N w_i^{(m)}\mathcal{I}\left(y_i \neq \widehat{\phi}_m\left(x_i\right)\right)}{\sum_{i=1}^N w_i^{(m)}}$$

, and

$$\widehat{\phi}_m = \underset{\alpha,\phi}{\arg\min}\sum_{i=1}^N w_i^{(m)}\mathcal{I}\left(y_i \neq \phi\left(x_i\right)\right)$$

, i.e. $\widehat{\phi}_m$ is the function that minimizes the weighted misclassification error ϵ_m.

Appendix: AdaBoost as Forward Stepwise Modeling

We can then update $\widehat{f}^{(m)}(x) = \widehat{f}^{(m-1)}(x) + \widehat{\alpha}_m \widehat{\phi}_m(x)$, and

$$\begin{aligned}
w_i^{(m+1)} &= \exp\left(-y_i \widehat{f}^{(m)}(x_i)\right) \\
&= \exp\left(-y_i\left[\widehat{f}^{(m-1)}(x_i) + \widehat{\alpha}_m \widehat{\phi}_m(x_i)\right]\right) \\
&= w_i^{(m)} \exp\left(-y_i \widehat{\alpha}_m \widehat{\phi}_m(x_i)\right) \\
&= w_i^{(m)} \exp\left(2\widehat{\alpha}_m \mathcal{I}(y_i \neq \phi(x_i))\right) \cdot \exp(-\widehat{\alpha}_m)
\end{aligned}$$

, where we use the fact that $-y_i \widehat{\phi}_m(x_i) = 2\mathcal{I}(y_i \neq \phi(x_i)) - 1$.

Therefore, we can multiply $\widehat{\alpha}_m$ by 2, i.e., let $\widehat{\alpha}_m = \log \frac{1-\epsilon_m}{\epsilon_m}$ and update w_i as

$$w_i^{(m+1)} = w_i^{(m)} \exp\left(\widehat{\alpha}_m \mathcal{I}(y_i \neq \phi(x_i))\right)$$

This gives us the AdaBoost algorithm.

Acknowledgement I

Part of this lecture is adapted from the following sources:

- Hastie, T., R. Tibshirani, and J. Friedmand. 2008. *The Elements of Statistical Learning* (2^{nd} ed.). Springer.
- James, G., D. Witten, T. Hastie, and R. Tibshirani. 2013. *An Introduction to Statistical Learning: with Applications in R*. Springer.
- Rosenberg, D. S. *Machine Learning and Computational Statistics*, Lecture at NYU Center for Data Science, retrieved on 2019.01.01. [link]
- Sontag, D. *Introduction To Machine Learning*. Lecture at NYU, retrieved on 2018.01.01. [link]
- Taddy, M. *Big Data*. Lecture at the University of Chicago Booth School of Business, retrieved on 2017.01.01. [link]

Acknowledgement II

- Tibshirani, R. *Data Mining*, Lecture at Carnegie Mellon University, retrieved on 2017.01.01. [link]
- Van der Schaar, M. and S. Flaxman. *Statistical Machine Learning*. Lecture at Oxford University, retrieved on 2018.01.01. [link]

Reference

Dietterich, T. G. 2000. “Ensemble Methods in Machine Learning,” In: Multiple Classifier Systems. MCS 2000. Lecture Notes in Computer Science, vol 1857. Springer.