Tree-Based Methods and Random Forests

Tadao Hoshino （星野匡郎）

Econometrics II: ver. 2019 Spring Semester

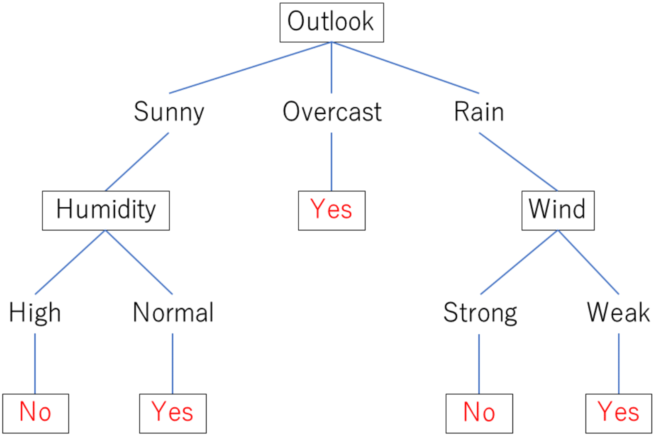
Introduction

"Play Tennis" example:

[Data]

Outlook	**Humidity**	**Wind**	**Play Tennis**
Overcast	High	Strong	Yes
Overcast	Normal	Strong	Yes
Sunny	Normal	Weak	Yes
Rain	Normal	Strong	No
Overcast	Normal	Weak	Yes
Rain	High	Weak	Yes
Sunny	High	Weak	No

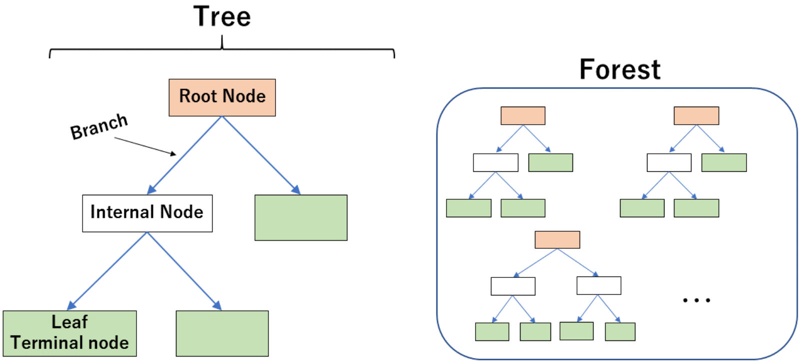
[Decision Tree]



Introduction

- The above figure is called a decision tree.
- The machine learning algorithm for creating a decision tree from data is called decision tree learning.
  - Note that there may be many possible decision trees for a given dataset.
- Typical problems solved with decision trees:
  - Medical diagnosis
  - Robot movements
  - Image recognition
  - and many more.

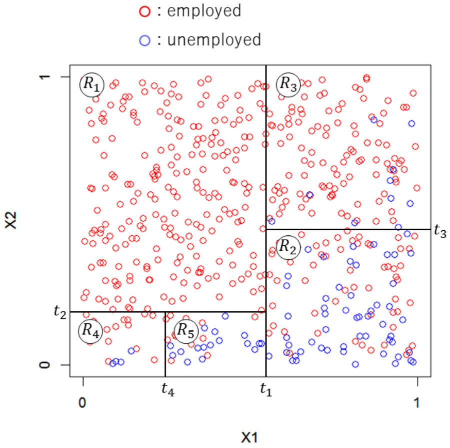
Terminology

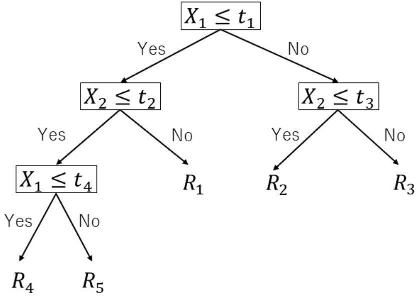


- Classification and regression tree (CART):

CART is a "nonparametric" machine learning method, which utilizes a decision-tree structure to solve classification and regression problems.

- In many classification and regression problems, some of input variables are not discrete but continuous.
- For continuous variables, we introduce a threshold value t and consider a bifurcation rule $\{X \leq t\}$ *or* $\{X > t\}$.





Fit a simple model (constant-only model) in each leaf R_m.

Regression Trees

Nonparametric Regression

- Consider a general regression model

$$Y = g(\mathbf{X}) + \varepsilon,$$

  where Y is a response variable, $\mathbf{X} = (X_1, \ldots, X_k)^\top$ is a vector of input variables, and ε is an error term.
- We assume that $E[\varepsilon|\mathbf{X}] = 0$; this is equivalent to assume that

$$g(\mathbf{x}) = E[Y|\mathbf{X} = \mathbf{x}].$$

- The "linear" regression model is a special case with

$$g(\mathbf{X}) = \beta_0 + X_1\beta_1 + \cdots + X_k\beta_k$$

  The function $g(\mathbf{X})$ can be estimated by estimating the parameters $(\beta_0, \beta_1, \ldots, \beta_k)$. $=>$ Parametric regression

- One can directly estimate $g(\cdot)$ without assuming any specific functional form on it.
- For simplicity, assume that $\mathbf{X} = (X_1, X_2)^\top$ and $X_1, X_2 \in \{0, 1\}$. Then, $g(\mathbf{X})$ has four values:

$$g(x_1, x_2) = E[Y|X_1 = x_1, X_2 = x_2] \text{ for } x_1, x_2 \in \{0, 1\}$$

- Each $g(x_1, x_2)$ can be estimated simply by the conditional sample mean

$$\widehat{g}_n(x_1, x_2) = \frac{\sum_{i=1}^n \mathbf{1}\{X_{1i} = x_1, X_{2i} = x_2\} Y_i}{\sum_{i=1}^n \mathbf{1}\{X_{1i} = x_1, X_{2i} = x_2\}}$$

$=>$ <u>Nonparametric regression with discrete variables</u>

- When X_1 and X_2 are continuous, since

$$\Pr(X_1 = x_1, X_2 = x_2) = 0$$

for any x_1 and x_2, the above nonparametric regression is infeasible.

- However, for a sufficiently small rectangle $(x_1, x_2) \in R_{\mathbf{x}}$, we can approximate the event $\{X_1 = x_1, X_2 = x_2\}$ by $\{(X_1, X_2) \in R_{\mathbf{x}}\}$, i.e.,

$$E[Y|X_1 = x_1, X_2 = x_2] \approx E[Y|(X_1, X_2) \in R_{\mathbf{x}}]$$

- Thus, for continuous (X_1, X_2), we can estimate $g(x_1, x_2)$ by

$$\widehat{g}_n(x_1, x_2) = \frac{\sum_{i=1}^{n} \mathbf{1}\{(X_{1i}, X_{2i}) \in R_{\mathbf{x}}\} Y_i}{\sum_{i=1}^{n} \mathbf{1}\{(X_{1i}, X_{2i}) \in R_{\mathbf{x}}\}}$$

$=>$ Nonparametric regression with continuous variables

- Data: $\{(Y_i, \mathbf{X}_i) : 1 \leq i \leq n\}$, where
  - Y = continuous response variable
  - $\mathbf{X} = (X_1, \ldots, X_k)^\top = k$ input variables.
- Suppose that the space of $\mathbf{X}$ (the so-called feature space) is partitioned into M leaves: $R_1, \ldots, R_M$ (how to form the partition will be described later).
- Predict the value of Y by

$$\widehat{Y} = \sum_{m=1}^{M} \widehat{c}_m \mathbf{1}\{\mathbf{X} \in R_m\}$$

where

$$\widehat{c}_m = \frac{1}{N_m} \sum_{\mathbf{X}_i \in R_m} Y_i, \quad \text{and} \quad N_m = \sum_{i=1}^{n} \mathbf{1}\{\mathbf{X}_i \in R_m\}.$$

- Finding the best partition is intractable in general, since there are too many possible partitions to consider in the feature space.

A "greedy" algorithm

1. Starting with all of the data, consider a variable j and a threshold t, and define the pair of half-spaces:

$$R_1(j,t) = \{\mathbf{X} : X_j \leq t\} \text{ and } R_2(j,t) = \{\mathbf{X} : X_j > t\}.$$

Then, we determine the splitting variable j^* and threshold t^* by solving

$$\min_{j,t} \left[\sum_{i=1:\ \mathbf{X}_i \in R_1(j,t)}^{n} (Y_i - \widehat{c}_1(j,t))^2 + \sum_{i=1:\ \mathbf{X}_i \in R_2(j,t)}^{n} (Y_i - \widehat{c}_2(j,t))^2 \right],$$

where

$$\widehat{c}_m(j,t) = \frac{\sum_{i=1}^{n} \mathbf{1}\{\mathbf{X}_i \in R_m(j,t)\} Y_i}{\sum_{i=1}^{n} \mathbf{1}\{\mathbf{X}_i \in R_m(j,t)\}} \text{ for } m = 1,2.$$

Regression Trees

A "greedy" algorithm (cont.)

2. Having found the best initial split (j^*, t^*), partition the data into the two resulting regions (leaves) and repeat the splitting process on each of the two regions.
3. Repeat Steps 1 and 2 ("grow" the tree) until some convergence criterion is met.

* If the tree is grown too large with many leaves, an overfitting problem arises; it is necessary to prune the tree. $=>$ cross-validation.

- To implement Regression Trees in **R**, we can use the **tree** package.
- Install and load the package by

```
install.packages("tree")
library(tree)
```

(This package is very new, and **R** version 3.6 or later is recommended.)

- Import the apartment data:

```
data <- read.csv("apartments.csv")
```

Regression Trees

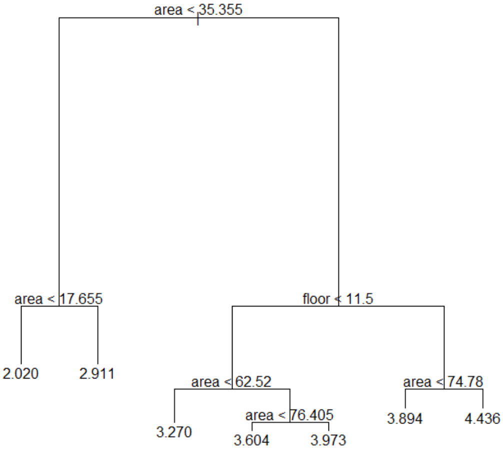
```
>  library(tree)
>  data <- read.csv("apartments.csv")
>  head(data)
   price  area floor renov    stdist commercial industrial
1 20.038 19.70     3     1 0.3123682          1          0
2 96.300 91.24    23     0 0.3116436          0          1
3 39.300 42.08    13     0 0.2460939          1          0
4 85.600 74.36    15     0 0.4952629          0          0
5  5.700 17.89     6     0 0.8047969          0          0
6 25.200 32.37     8     0 0.5117592          1          0
>  dim(data)
[1] 500   7
```

- We define the response variable as the log of apartment price.
- Fit an unpruned regression tree using the function `tree()`:[1]

```
up_tree <- tree(log(price) ~ ., data)
plot(up_tree)
text(up_tree)
```

[1] By default, this function avoids generating a too large tree by restricting several parameters, including the minimum number of observations in each leaf. The default setup can be customized using `tree.control()` command.

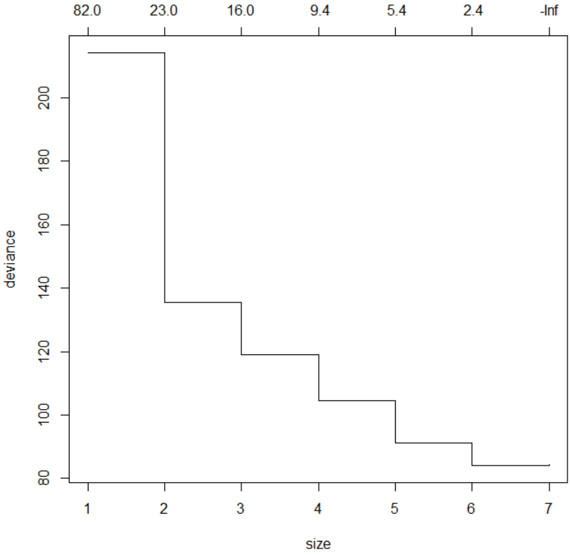
Regression Trees



- Prune the tree using cross-validation with the `cv.tree()` function:

```
cv_tree <- cv.tree(up_tree)
plot(cv_tree)
```

- As the result of this, we can find that the tree with 6 leaves has almost equal (or slightly better) performance to the one with 7 leaves.


82.0
23.0
16.0
9.4
5.4
2.4
-Inf
deviance
200
180
160
140
120
100
80
1
2
3
4
5
6
7
size

Regression Trees

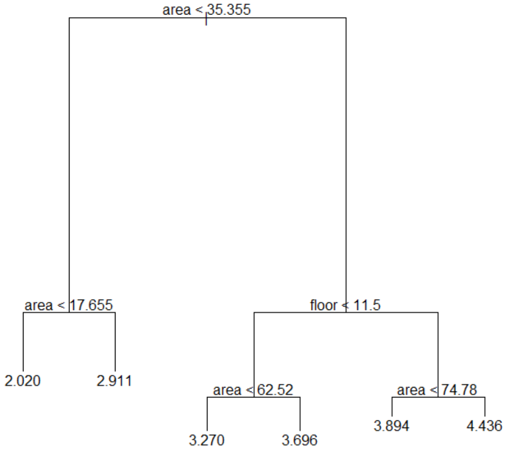
- Fit an optimal pruned regression tree:

```
opt_tree <- prune.tree(up_tree, best = 6)
                                # of leaves
plot(opt_tree)
text(opt_tree)
```

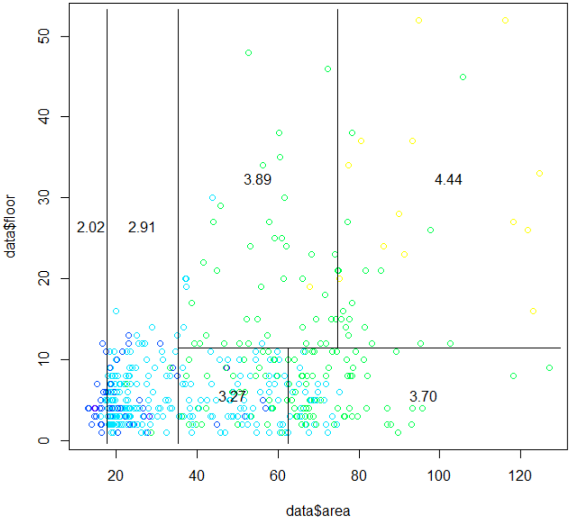
- Visualize the partition:

```
r <- round(log(data$price))
plot(data$area, data$floor, col = topo.colors(5)[r])
partition.tree(opt_tree, add = TRUE)
```

Regression Trees



Regression Trees



Binary Classification Trees

- The above approach can be used for binary classification problems, where Y is a dummy variable.
- For $Y \in \{0, 1\}$, it holds that

$$0 \leq \widehat{c}_m \leq 1 \quad \text{where recall that} \quad \widehat{c}_m = \frac{1}{N_m} \sum_{\mathbf{X}_i \in R_m} Y_i.$$

  Thus, the predicted value of Y can be interpreted as the predicted probability of $Y = 1$.
- For classification problems, we can consider various types of error metrics other than the mean squared error loss.

Examples of impurity measures

- Gini index (squared error loss):

$$\sum_{m=1}^{M} \sum_{\mathbf{X}_i \in R_m} (Y_i - \widehat{c}_m)^2 \quad \left(= \sum_{m=1}^{M} N_m \widehat{c}_m (1 - \widehat{c}_m) \right)$$

- Cross entropy (negative log-likelihood, deviance):

$$- \sum_{m=1}^{M} \sum_{\mathbf{X}_i \in R_m} \left[Y_i \log \widehat{c}_m + (1 - Y_i) \log(1 - \widehat{c}_m) \right]$$

$$\left(= - \sum_{m=1}^{M} N_m \left[\widehat{c}_m \log \widehat{c}_m + (1 - \widehat{c}_m) \log(1 - \widehat{c}_m) \right] \right)$$

- Then, based on some impurity measure, binary classification trees can be grown (and pruned) in the same way as regression trees.
- Classification trees can be implemented in **R** with the package **tree**.
- Load the packages **ROCR** and **tree**, and import the Titanic survival data:

```
library(ROCR)
library(tree)
train <- read.csv("Titanic_train.csv")
test  <- read.csv("Titanic_test.csv")
```

Binary Classification Trees

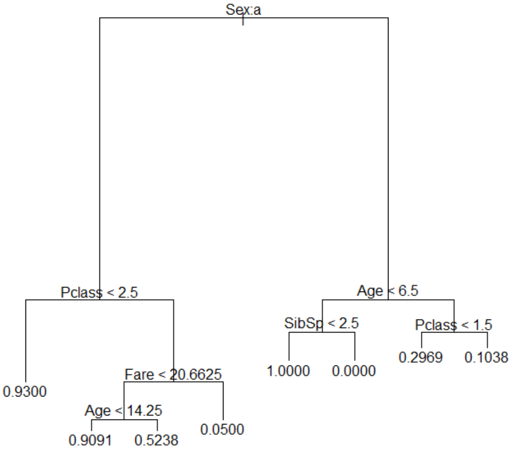
```
> library(ROCR)
> library(tree)
> train <- read.csv("Titanic_train.csv")
> test  <- read.csv("Titanic_test.csv")
>
> head(train)
                           Passenger Survived Pclass    Sex Age SibSp Parch    Fare Embarked
1               Turja, Miss. Anna Sofia        1      3 female  18     0     0  9.8417        S
2          Francatelli, Miss. Laura Mabel        1      1 female  30     0     0 56.9292        C
3 Bishop, Mrs. Dickinson H (Helen Walton)        1      1 female  19     1     0 91.0792        C
4            Crosby, Capt. Edward Gifford        0      1   male  70     1     1 71.0000        S
5          Coelho, Mr. Domingos Fernandeo        0      3   male  20     0     0  7.0500        S
6                  Hunt, Mr. George Henry        0      2   male  33     0     0 12.2750        S
> dim(train)
[1] 464   9
> dim(test)
[1] 250   9
```

Binary Classification Trees

- The objective of this exercise is to predict the survival status of the passengers in the test data set.
- We first fit an unpruned classification tree to the training data set:[2]

```
up_tree <- tree(Survived ~ Pclass + Sex + Age + SibSp
           + Fare, train)
plot(up_tree)
text(up_tree)
```

[2]The default impurity measure is the *cross entropy*. You can change this with the `split` option.


Sex:a
Pclass < 2.5
Age < 6.5
SibSp < 2.5
Pclass < 1.5
0.2969
0.1038
1.0000
0.0000
Fare < 20.6625
0.9300
Age < 14.25
0.0500
0.9091
0.5238

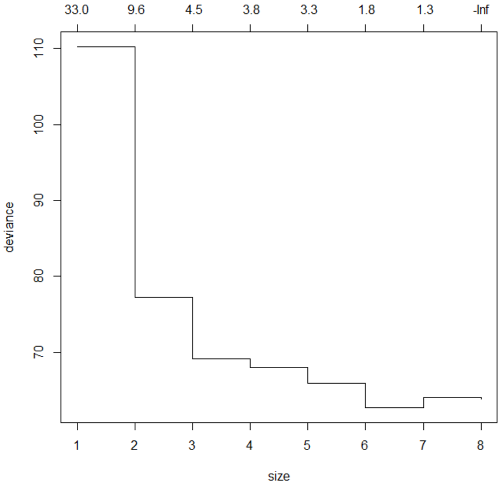
- Find an optimum subtree using cross-validation:

```
cv_tree <- cv.tree(up_tree)
plot(cv_tree)
```

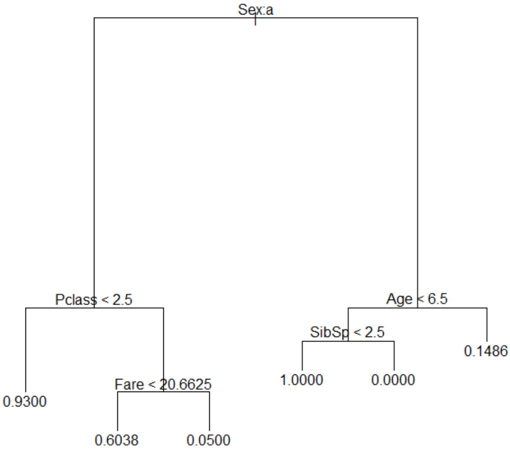
- As the result of this, we can find that the tree with 6 leaves performs the best.
- Fit an optimal pruned classification tree:

```
opt_tree <- prune.tree(up_tree, best = 6)
plot(opt_tree)
text(opt_tree)
```

Binary Classification Trees



Binary Classification Trees



- Compute the survival probability of passengers in the test data set:

```
s <- predict(opt_tree, newdata = test)
```

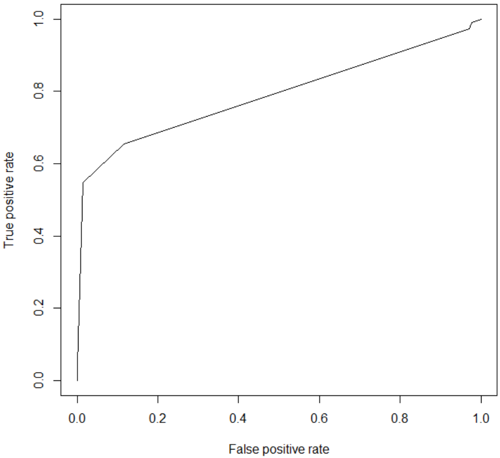
- Compute the AUC score, and plot the ROC curve

```
pred <- prediction(s, test$Survived)
auc  <- performance(pred, "auc")
auc@y.values[[1]]
```

```
>  auc@y.values[[1]]
[1] 0.7900975
```

```
roc <- performance(pred, "tpr", "fpr")
plot(roc)
```

Binary Classification Trees



Random Forests

Random Forests

- It is often the case that simple classification trees perform worse than the other methods.[3]
- The poor predictive performance of classification trees is mostly due to its nonparametric nature.
  - The performance of nonparametric methods quickly deteriorates as the dimension of feature increases (i.e., the curse of dimensionality).
- To improve the practical performance: an ensemble method
  1. Select a subset of the input variables, and create a classification tree T_1 based on the selected inputs.
  2. Select another subset of the input variables, and similarly create a new classification tree, say T_2.
  3. Repeat the above step many times, and combine (take the average of) the trees T_1, T_2, ...

[3] In the above Titanic example, the AUC score of the linear classifier and that of the logistic classifier were about 0.84; see the lecture note #7.

Random Forest algorithm

1. For $b = 1, \ldots, B$,
   - (Bootstrapping) Draw a random sample of size n with replacement from the original data set.[4]
   - Randomly select $q < k$ input variables, where q is normally 2 or 3, and k is the total number of input variables.
   - Grow a regression/classification tree, and compute the predicted value $\widehat{Y}_b$ of the response variable Y.
2. Take the average over the B predictors:

$$\widehat{Y}^{\text{random forest}} = \frac{1}{B} \sum_{b=1}^{B} \widehat{Y}_b.$$

[4] By doing this, we can avoid the overfitting problem.

Random Forests

- We can easily implement the random forest algorithm in **R** with the package **randomForest**.

```
install.packages("randomForest")
library(randomForest)
```

- We again use the Titanic data.
- In order to perform a random forest classification with this package, the response variable needs to be a "factor" class object.
- We can use the function `randomForest()` to learn a random forest classifier ($B = 500$ by default).

```
train$Survived <- as.factor(train$Survived)
rf <- randomForest(Survived ~ Pclass + Sex + Age
      + SibSp + Fare, train)
```

Random Forests

AUC score and ROC curve.

```
srf  <- predict(rf, newdata = test, "prob")[,2]
pred <- prediction(srf, test$Survived)
auc  <- performance(pred, "auc")
auc@y.values[[1]]
```

```
>  auc@y.values[[1]]
[1] 0.8705833
```

The random forest performs very well, outperforming the linear and logistic classifier.

```
roc <- performance(pred, "tpr", "fpr")
plot(roc)
```

Random Forests