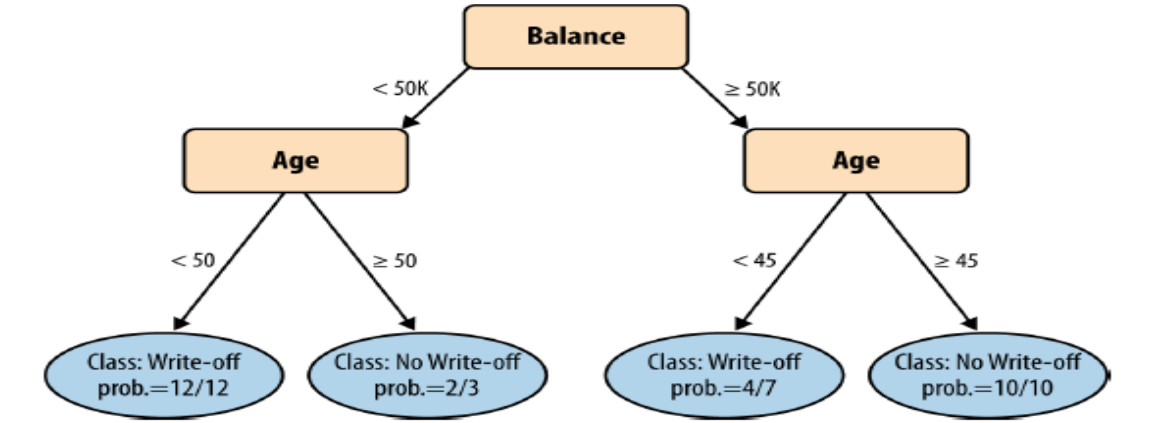
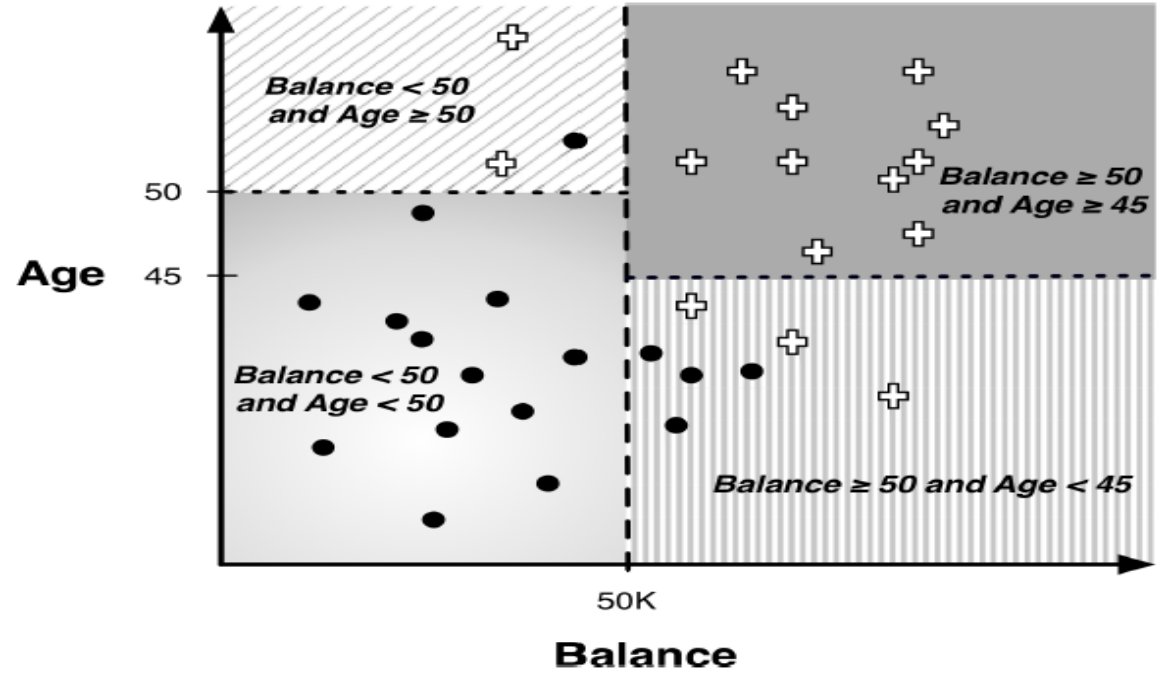
# ML Algorithms
# for
# Shallow Learning Problems

# Decision Tree

# Supervised Segment with Tree-Structured Models




*Source: Data Science for Business by Provost & Fawcett*

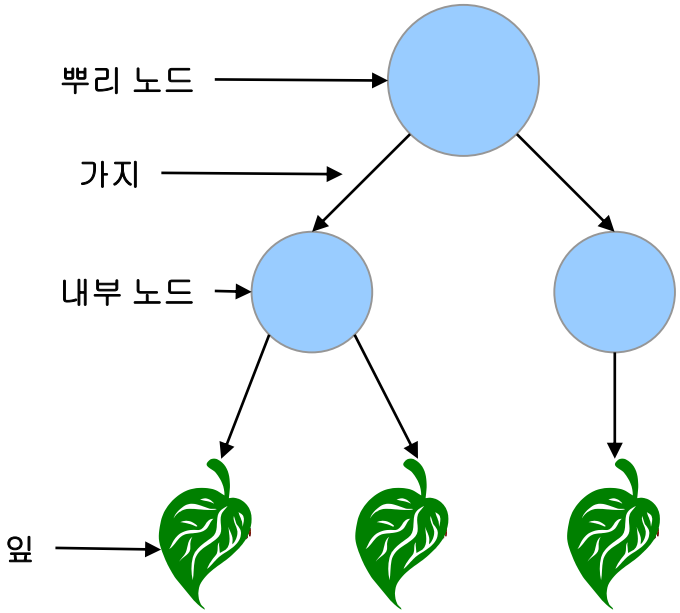
# 의사결정나무(Decision Tree)의 개요

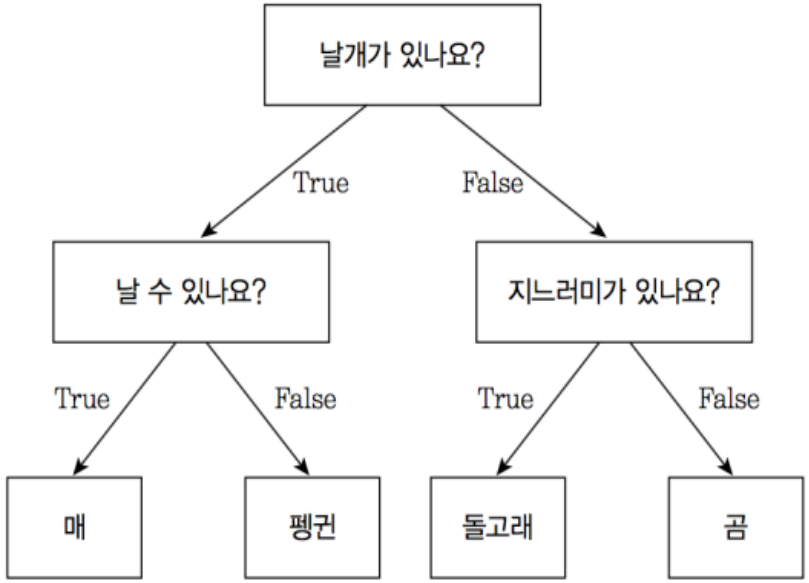
- 분류와 예측에서 자주 사용되는 강력하고 인기 있는 도구
- 분류 결과를 인간이 쉽게 이해할 수 있는 규칙으로 표현
  - 대출 상담자에게 대출을 해 줄 경우, 정확한 근거를 제시하여 대출을 거부할 자료를 제시할 수 있음
  - If 연간 수입 $20,000이상 and 관련 계좌가 3개 이상 then 대출
- 다수의 feature와 class 간의 관계에 대한 통찰을 얻을 수 있기 때문에 데이터 탐색 과정에서도 널리 활용
- 주요 의사결정나무 알고리즘
  - Categorical class만 가능: C5.0, QUEST
  - Numeric & categorical class 모두 가능: CART, CHAID

*# scikit-learn의 DecisionTreeClassifier는 CART를 기반으로 구현*

# 의사결정나무의 구성

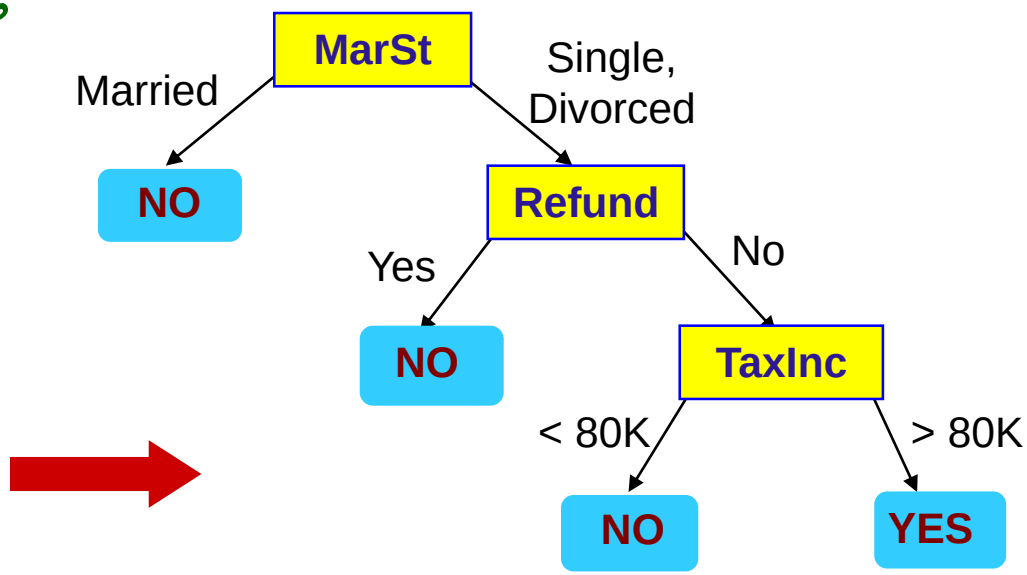
- Root node – 최상단에 위치 (스무고개의 첫 번째 질문)
- Internal node – 속성의 분리 기준을 포함 (중간 질문)
- Link – 마디와 마디를 이어줌 (질문에 대한 답)
- Leaf – 최종 분류 값을 가짐 (결정)



# 의사결정나무: 예시#1

| Tid | Refund (categorical) | Marital Status (categorical) | Taxable Income (continuous) | Cheat (class) |
|---|---|---|---|---|
| 1 | Yes | Single | 125K | No |
| 2 | No | Married | 100K | No |
| 3 | No | Single | 70K | No |
| 4 | Yes | Married | 120K | No |
| 5 | No | Divorced | 95K | Yes |
| 6 | No | Married | 60K | No |
| 7 | Yes | Divorced | 220K | No |
| 8 | No | Single | 85K | Yes |
| 9 | No | Married | 75K | No |
| 10 | No | Single | 90K | Yes |

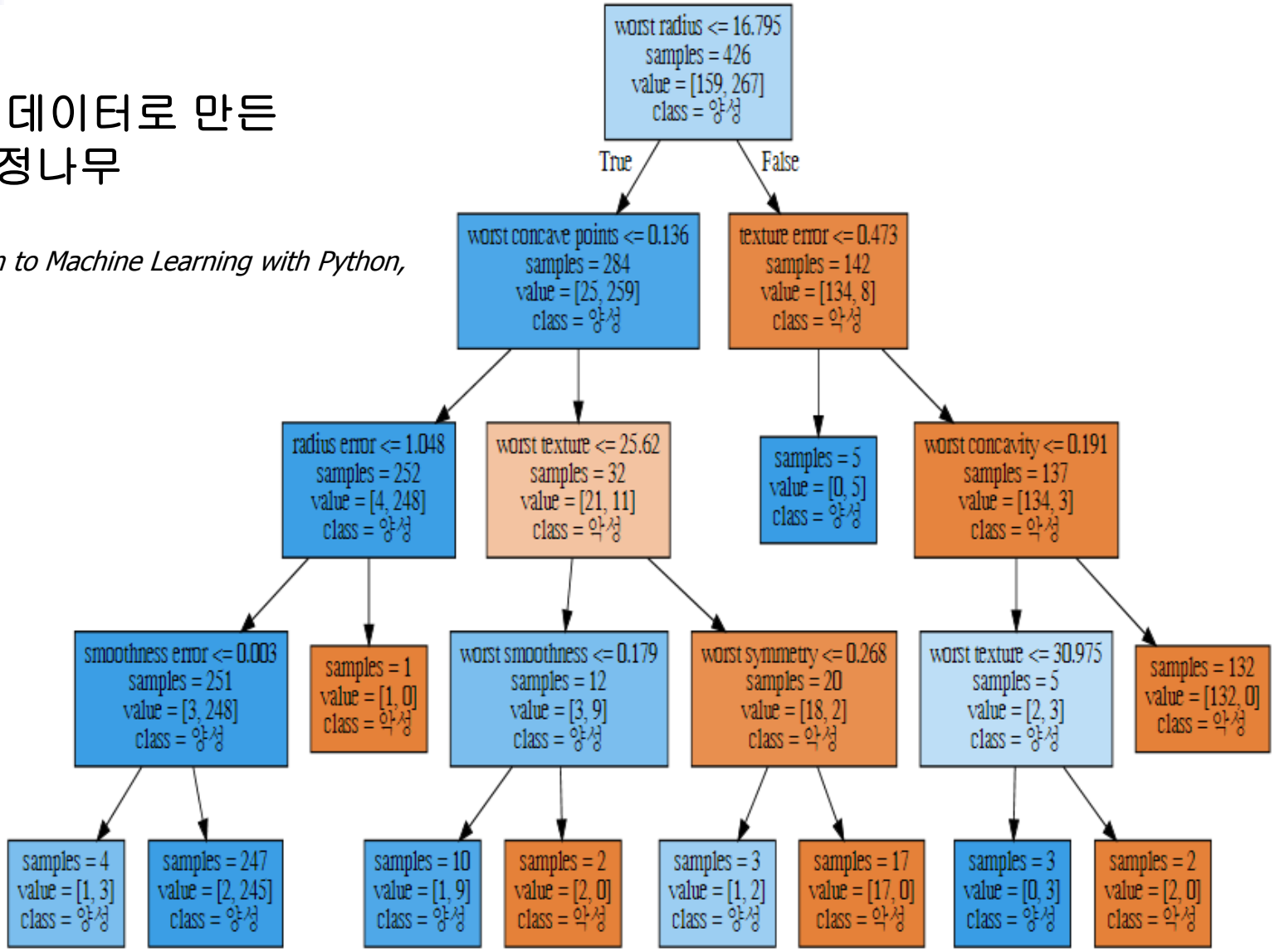



**There could be more than one tree that fits the same data!**

# 의사결정나무: 예시#2

유방암 데이터로 만든
의사결정나무

*Introduction to Machine Learning with Python, pp.108*



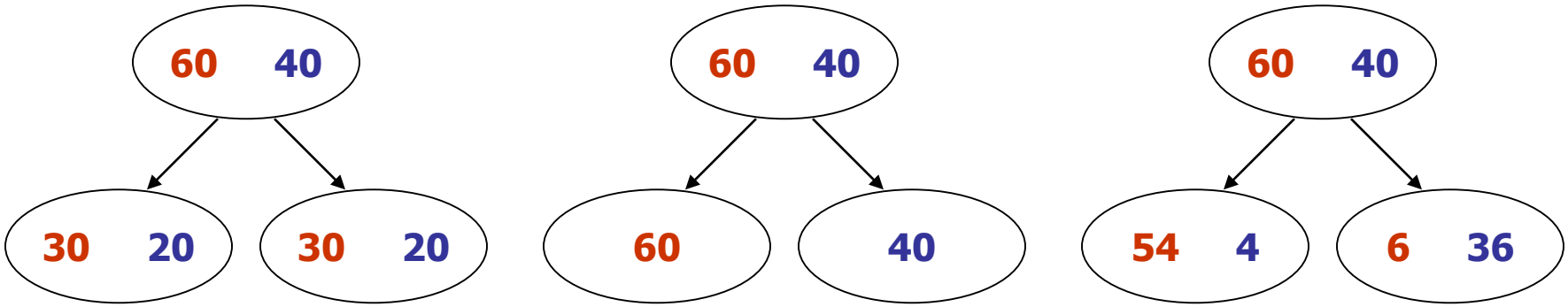
# 어떻게 의사결정나무가 생성되나

- 모든 의사결정나무 알고리즘들은 기본절차에 있어서 다음과 같은 공통점을 가지고 있음

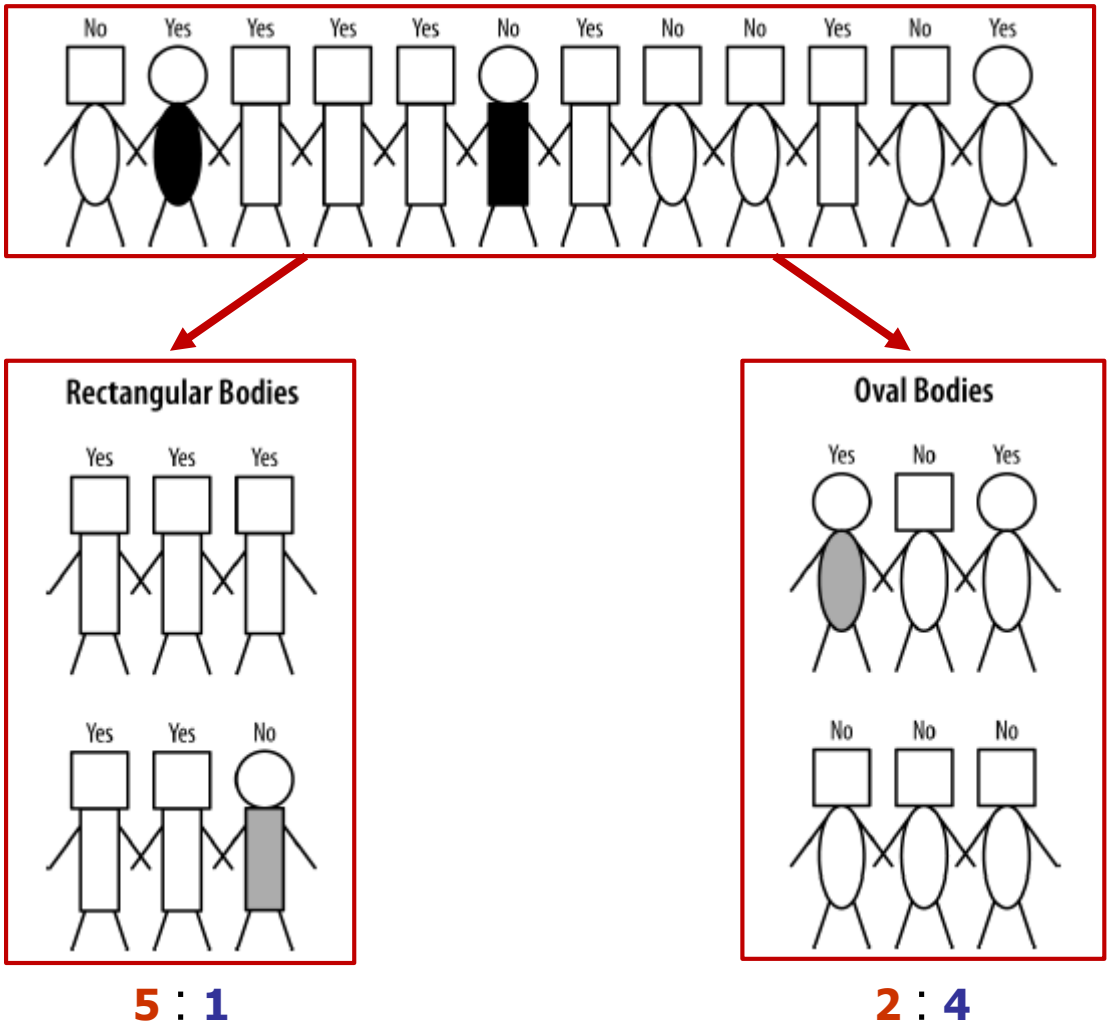
목표변수 측면에서 부모노드보다 더 순수도(purity)가 높은 자식노드들이 되도록, 데이터를 반복적으로 더 작은 집단으로 나눈다(repeatedly split)

# 순수도와 분리기준

- 순수도(Purity)
  - class의 특정 범주에 개체들이 포함되는 정도
- 분리기준(Splitting Criteria)
  - 하나의 부모노드로부터 자식노드들이 형성될 때 feature의 선택과 선택된 feature에 따른 범주를 선택할 때의 기준
  - 어떤 feature를 이용하여 분리하는 것이 class의 분포를 가장 잘 구별해 주는지를 파악하여 자식노드가 생성되는데, 부모노드의 순수도에 비해서 자식노드들의 순수도가 증가하도록 자식노드를 형성



# 나무 분리에 따른 순수도의 변화



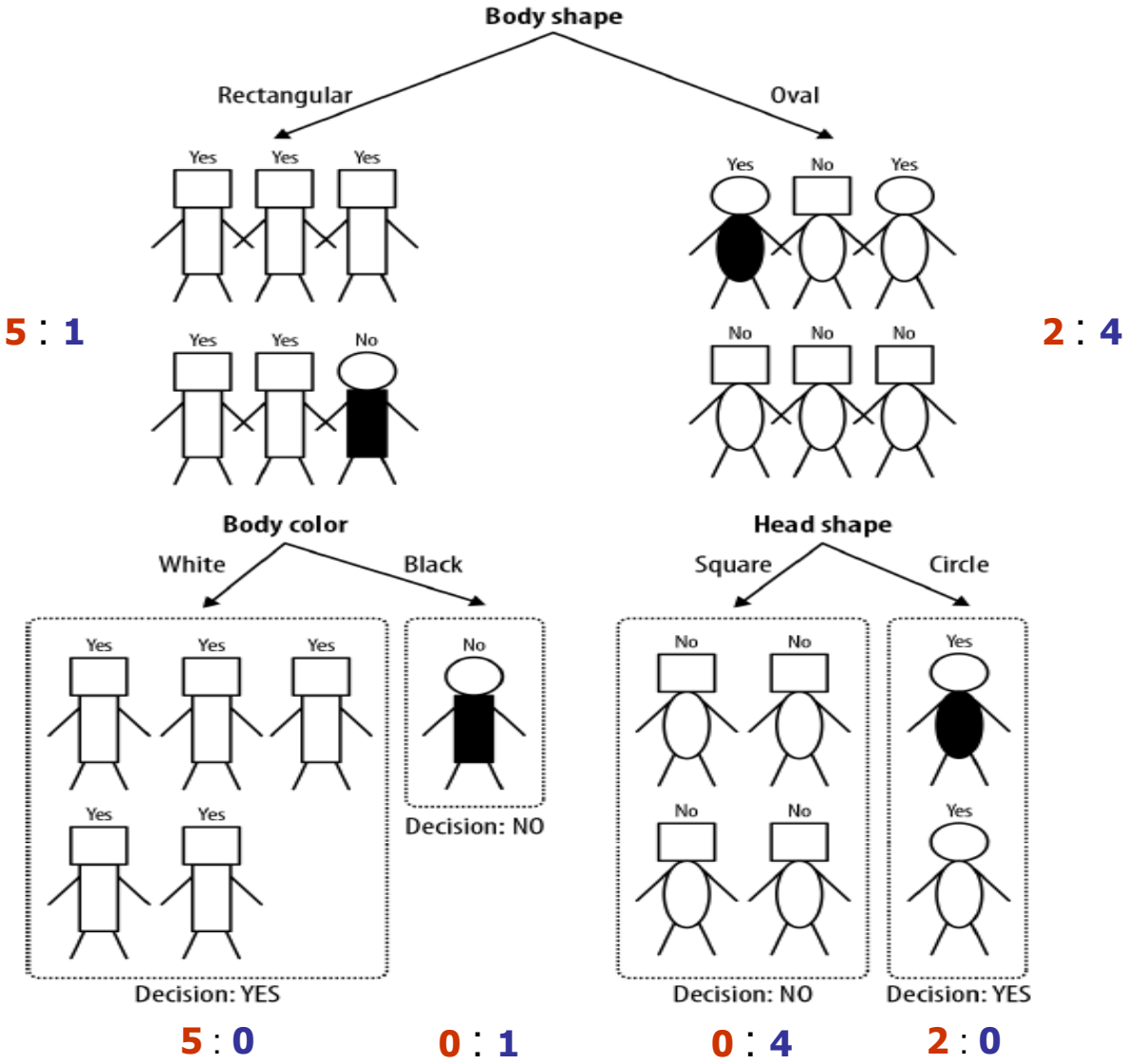

**7(Yes) : 5(No)**

**5 : 1**

**2 : 4**

*Source: Data Science for Business by Provost & Fawcett*

# 나무 분리에 따른 순수도의 변화




Source: Data Science for Business by Provost & Fawcett

# 순수도 척도

- 범주형 클래스에 대한 분할을 평가하는 순수도 척도
  - 지니(Gini)
  - 엔트로피(Entropy, 정보 이익(information gain))
  - 정보 이익 비율(Information gain ratio)
  - 카이제곱 검정(Chi-square test)

  *# scikit-learn에서는 Gini와 Entropy를 지원*

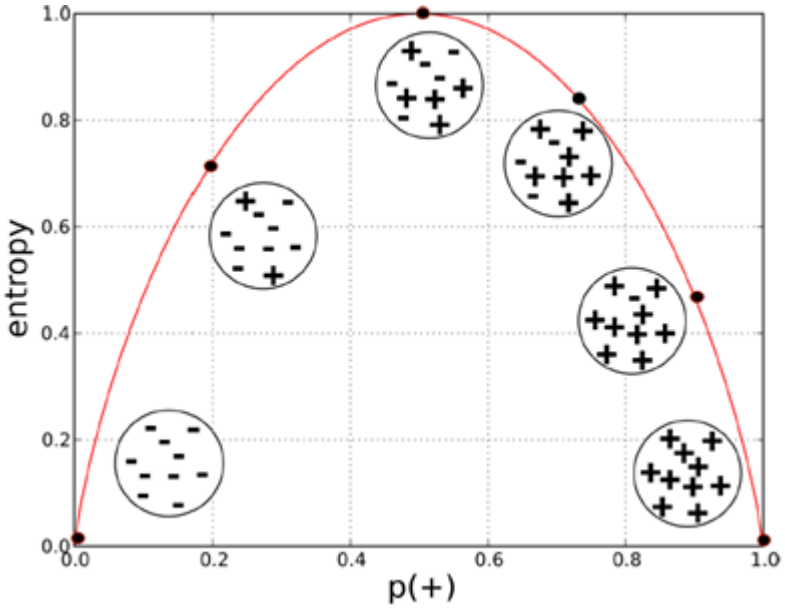
- 수치형 클래스의 경우
  - 분산의 감소(reduction in variance)
  - F 검정

# 엔트로피(Entropy)

- 결정규칙을 분리할 수 있는 기준으로 데이터의 무질서 정도를 측정할 수 있는 방법
  - 엔트로피는 물리학 용어로서 자연 상태의 무질서 정도를 나타내는 척도
  - 의사결정나무에서는 엔트로피가 높을 수록 class 구분을 잘 못해주는 feature가 되며, 낮을 수록 구분을 잘 해주는 유의한 feature가 된다.

$$Entropy = -\sum_{j=1}^{n} p_j \log p_j$$

($p$는 각 클래스의 비율)

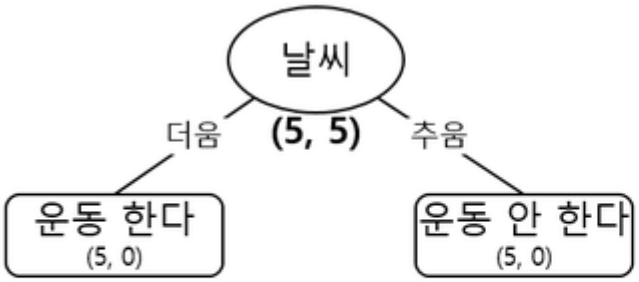



*Entropy of a two-class as a function of p(+)*

# 정보이득(Information Gain)

- 특정 feature를 선택했을 때 얻게되는 정보량이 얼마인지를 측정하는 지수
  - 정보이득을 계산하는데 엔트로피가 사용됨
  - 모든 feature의 정보이득을 계산한 후 정보이득이 가장 많은 feature를 선택하여 나무를 분리

$$Gain(S, f) = Entropy(S) - \sum_{v \in values(f)} \frac{|S_v|}{|S|} Entropy(S_v)$$



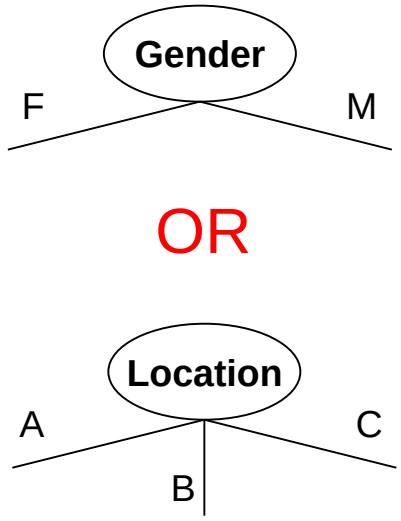

$$E(5, 5) = -\frac{5}{10} log_2 \frac{5}{10} - \frac{5}{10} log_2 \frac{5}{10} = 1.0$$

$$E(5, 0) = -\frac{5}{5} log_2 \frac{5}{5} - \frac{0}{5} log_2 \frac{0}{5} = 0.0$$

**날씨** 속성을 선택했을 때의 정보이득 양 = 1.0 – [0.5*0.0 +0.5*0.0] = **1.0**

# 의사결정나무 생성과정: 예시

| 성별(Gender) | 거주지역(Location) | 응답여부(Respond) |
|---|---|---|
| M | A | Y |
| M | B | Y |
| M | A | Y |
| M | C | Y |
| F | B | N |
| F | A | N |
| F | B | N |
| M | C | N |
| M | A | N |
| M | A | Y |




| | | Respond | | Total |
|---|---|---|---|---|
| | | Y | N | |
| Gender | F | 0 | 3 | 3 |
| | M | 5 | 2 | 7 |
| Total | | 5 | 5 | 10 |

| | | Respond | | Total |
|---|---|---|---|---|
| | | Y | N | |
| Location | A | 3 | 2 | 5 |
| | B | 1 | 2 | 3 |
| | C | 1 | 1 | 2 |
| Total | | 5 | 5 | 10 |

# 의사결정나무 생성과정: 예시

## Information Gain

- Gender

| | | Respond | | Total |
|---|---|---|---|---|
| | | Y | N | |
| Gender | F | 0 | 3 | 3 |
| | M | 5 | 2 | 7 |
| Total | | 5 | 5 | 10 |

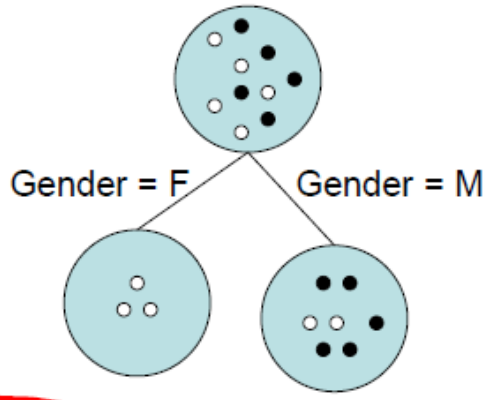
$$E_{before} = -\frac{5}{10}\log_2\frac{5}{10} - \frac{5}{10}\log_2\frac{5}{10} = 1$$

$$E_{left} = -0 - \frac{3}{3}\log_2\frac{3}{3} = 0$$

$$E_{right} = -\frac{5}{7}\log_2\frac{5}{7} - \frac{2}{7}\log_2\frac{2}{7} = 0.863121$$

$$E_{after} = \frac{3}{10}\times 0 + \frac{7}{10}\times 0.863121 = 0.604185$$

$$IG_{Gender} = E_{before} - E_{after} = 1 - 0.604185 = 0.395815$$



# 의사결정나무 생성과정: 예시

## Information Gain

- Location

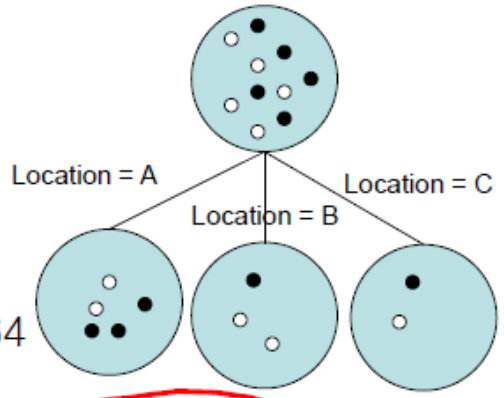
$$E_{before} = -\frac{5}{10}\log_2\frac{5}{10} - \frac{5}{10}\log_2\frac{5}{10} = 1$$

$$E_{left} = -\frac{3}{5}\log_2\frac{3}{5} - \frac{2}{5}\log_2\frac{2}{5} = 0.970951$$

| | | Respond | | Total |
|---|---|---|---|---|
| | | Y | N | |
| Location | A | 3 | 2 | 5 |
| | B | 1 | 2 | 3 |
| | C | 1 | 1 | 2 |
| Total | | 5 | 5 | 10 |

$$E_{middle} = -\frac{1}{3}\log_2\frac{1}{3} - \frac{2}{3}\log_2\frac{2}{3} = 0.918296$$

$$E_{right} = -\frac{1}{2}\log_2\frac{1}{2} - \frac{1}{2}\log_2\frac{1}{2} = 1$$




$$E_{after} = \frac{5}{10}\times 0.970951 + \frac{3}{10}\times 0.918296 + \frac{2}{10}\times 1 = 0.960964$$

$$IG_{location} = E_{before} - E_{after} = 1 - 0.960964 = 0.039036$$

IG값이 큰 Gender가 분기 기준으로 선택!

# *Gini* Index

- If a data set *T* contains examples from *n* classes, gini index, *gini*(*T*) is defined as $gini\ (T) = 1 - \sum_{j=1}^{n} p_j^2$

  where $p_j$ is the relative frequency of class *j* in *T*.
- If a data set *T* is split into two subsets $T_1$ and $T_2$ with sizes $N_1$ and $N_2$ respectively, the *gini* index of the split data contains examples from *n* classes, the *gini* index *gini*(*T*) is defined as

$$gini_{split}(T) = \frac{N_1}{N} gini(T_1) + \frac{N_2}{N} gini(T_2)$$

- The attribute provides the smallest $gini_{split}(T)$ is chosen to split the node

# Numeric Features

- Standard method: binary splits

| 64 | 65 | 68 | 69 | 70 | 71 | 72 | 72 | 75 | 75 | 80 | 81 | 83 | 85 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Yes | No | Yes | Yes | Yes | No | No | Yes | Yes | Yes | No | Yes | Yes | No |

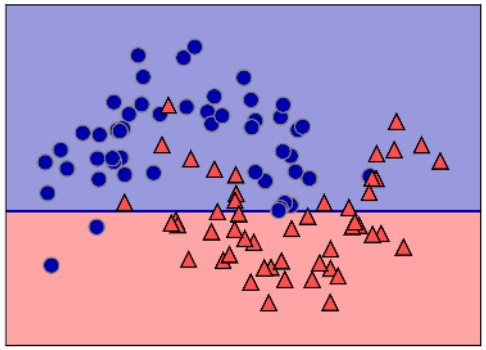
**X < 71.5** | **X >= 71.5**

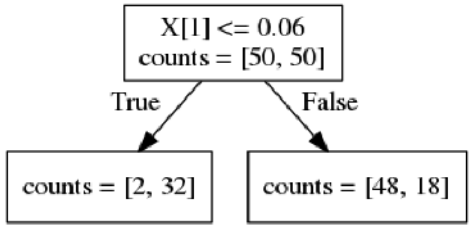
- Unlike nominal features, every feature has many possible split points
- Solution is straightforward extension:
  - Evaluate info gain for every possible split point of feature
  - Choose "best" split point
  - Info gain for best split point is info gain for feature
- Computationally more demanding
  - Sort instances by the values of the numeric feature

*Source: http://www.sts.tu-harburg.de/teaching/ss-09/ml-sose-09/03-Decision-Tree-c45.pdf*

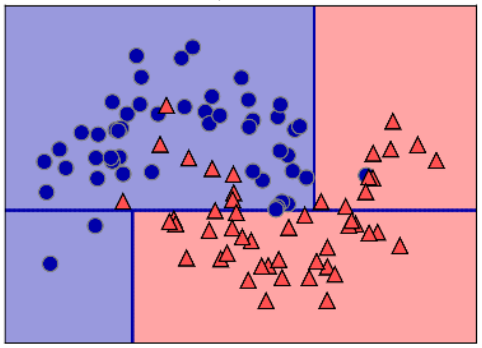
# 나무 깊이에 따른 결정 경계의 변화

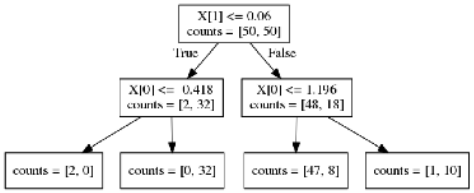
depth = 1



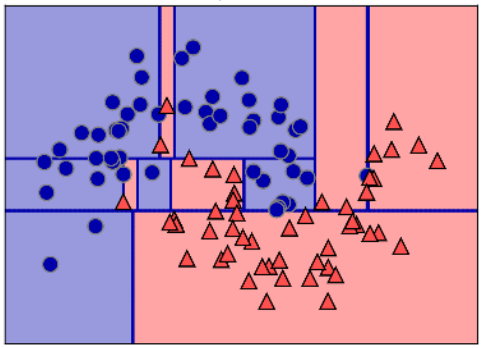


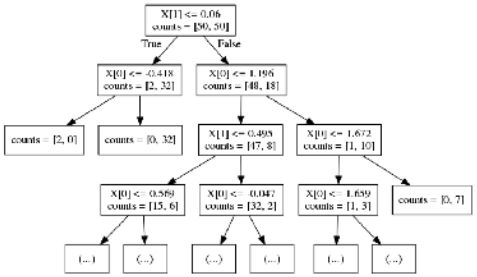

depth = 2





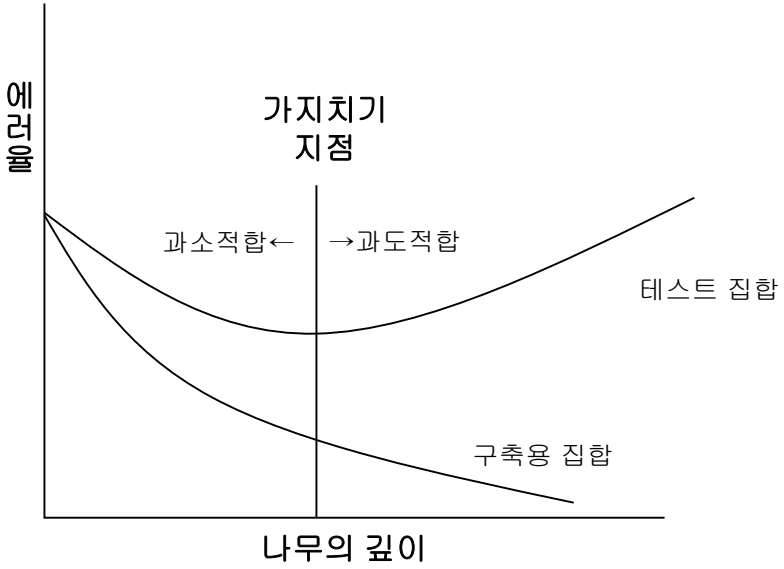

depth = 9





# 가지치기(Pruning)

- 최대나무(full tree)는 학습 데이터에 최적화된다.
  (Why? 리프노드의 소멸은 나무의 에러율을 증가시키기 때문)
- 최대나무가 새로운 데이터에 대해서도 가장 잘 분류하는 나무일까?
  **No! Overfitting이 발생한다!**
- Overfitting을 방지하기 위해 불필요하게 복잡해진 나무의 의미 없는 가지를 제거하는 작업을 가지치기(Pruning)라고 한다.



# 가지치기 전략

- Two strategies for "pruning" the decision tree:
  - Pre-pruning - stop growing a branch when information becomes unreliable
  - Post-pruning - take a fully-grown decision tree and discard unreliable parts

*# scikit-learn에서는 pre-pruning만 지원*

- Post-pruning preferred in practice
  - Pre-pruning can "stop too early"

Source: http://www.sts.tu-harburg.de/teaching/ss-09/ml-sose-09/03-Decision-Tree-c45.pdf

# 사전 가지치기(Pre-pruning)

- Stopping Rule

  현재의 노드에서 더 이상 분리가 일어나지 못하게 하는 규칙

  - 모든 자료가 한 그룹에 속할 때
  - 노드에 속하는 자료가 일정한 수 이하일 때: *min_samples_split in scikit-learn*
  - 불순도의 감소량이 아주 적을 때: *min_impurity_decrease*
  - 뿌리노드로부터 깊이가 일정 수 이상일 때: *max_depth*
  - 리프의 수가 일정 수 이상일 때: *max_leaf_nodes*

# 사후 가지치기(Post-pruning)

- Grow decision tree to its entirety
- Trim the nodes of the decision tree in a bottom-up fashion
- If generalization error improves after trimming, replace sub-tree by a leaf node.
- Class label of leaf node is determined from majority class of instances in the sub-tree

# 사후 가지치기: 예시

| | |
|---|---|
| Class = Yes | 20 |
| Class = No | 10 |
| Error = 10/30 | |

$$Error(t) = 1 - \max_i P(i \mid t)$$

**Training Error (Before splitting) = 10/30**

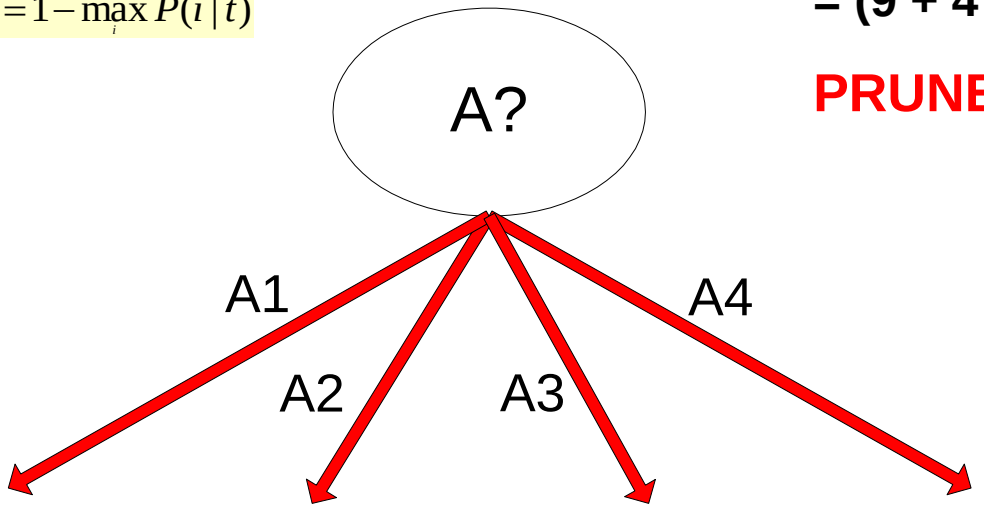
**Pessimistic error = (10 + 0.5)/30 = 10.5/30**

**Training Error (After splitting) = 9/30**

**Pessimistic error (After splitting)**

**= (9 + 4 × 0.5)/30 = 11/30**

**PRUNE!**



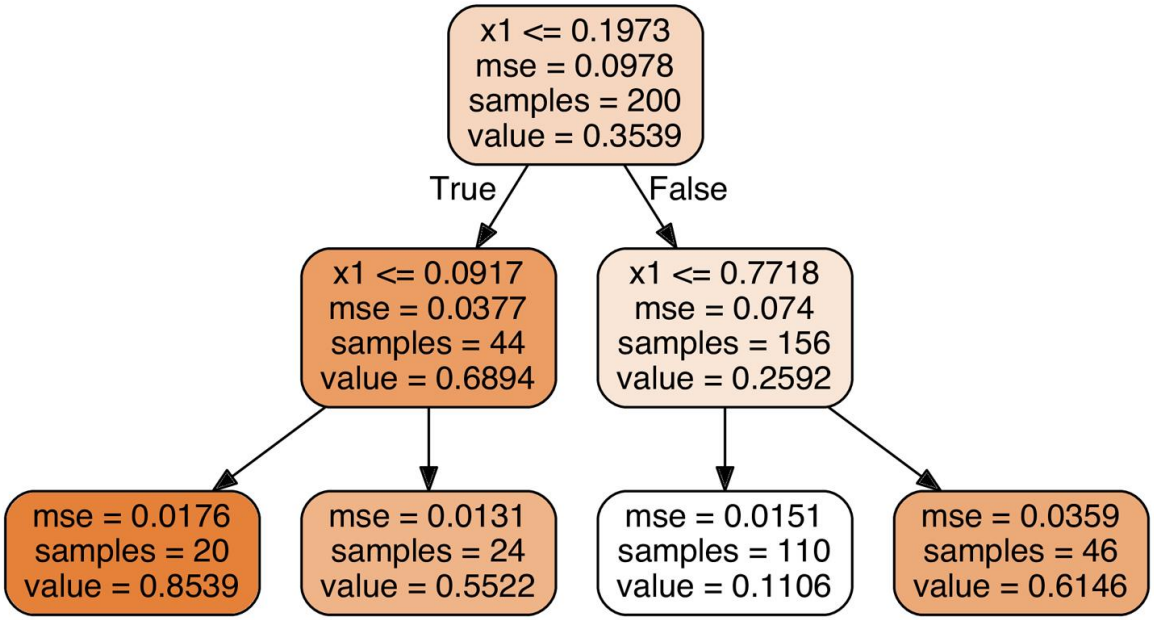

| | |
|---|---|
| Class = Yes | 8 |
| Class = No | 4 |

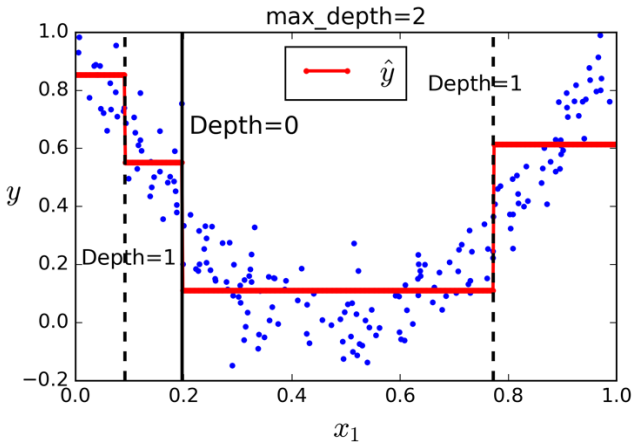
| | |
|---|---|
| Class = Yes | 3 |
| Class = No | 4 |

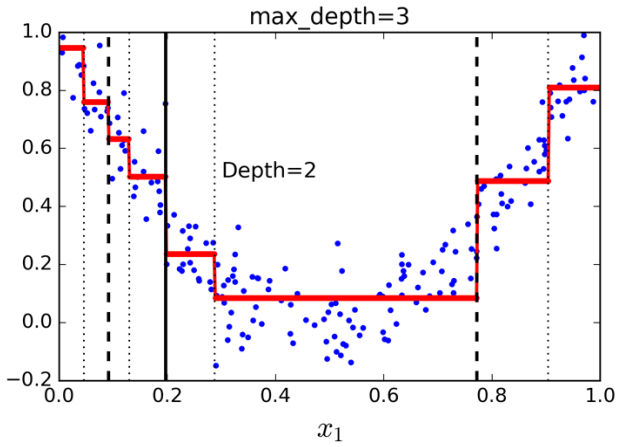
| | |
|---|---|
| Class = Yes | 4 |
| Class = No | 1 |

| | |
|---|---|
| Class = Yes | 5 |
| Class = No | 1 |

# Decision Tree for Regression



# Decision Tree for Regression

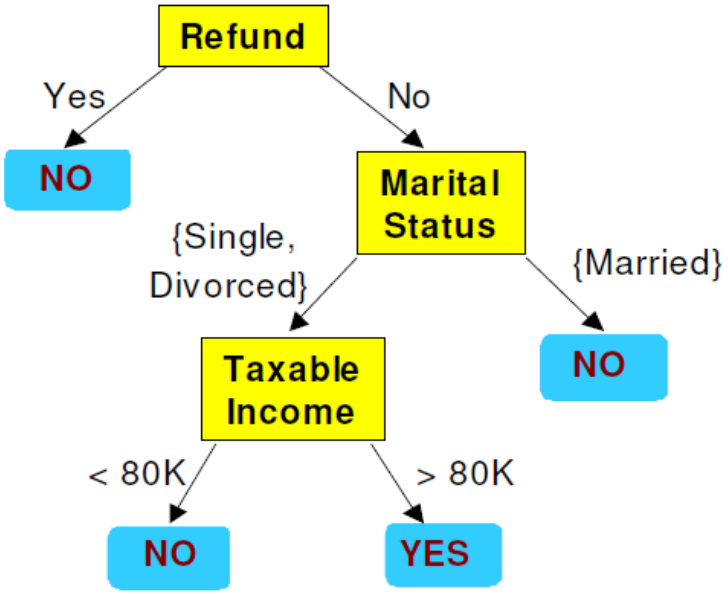
- CART cost function for classification

$$J(k, t_k) = \frac{m_{\text{left}}}{m} G_{\text{left}} + \frac{m_{\text{right}}}{m} G_{\text{right}}$$

- CART cost function for regression

$$J(k, t_k) = \frac{m_{\text{left}}}{m} \text{MSE}_{\text{left}} + \frac{m_{\text{right}}}{m} \text{MSE}_{\text{right}}$$

# From Decision Trees To Rules




**Classification Rules**

(Refund=Yes) ==> No

(Refund=No, Marital Status={Single,Divorced}, Taxable Income<80K) ==> No

(Refund=No, Marital Status={Single,Divorced}, Taxable Income>80K) ==> Yes

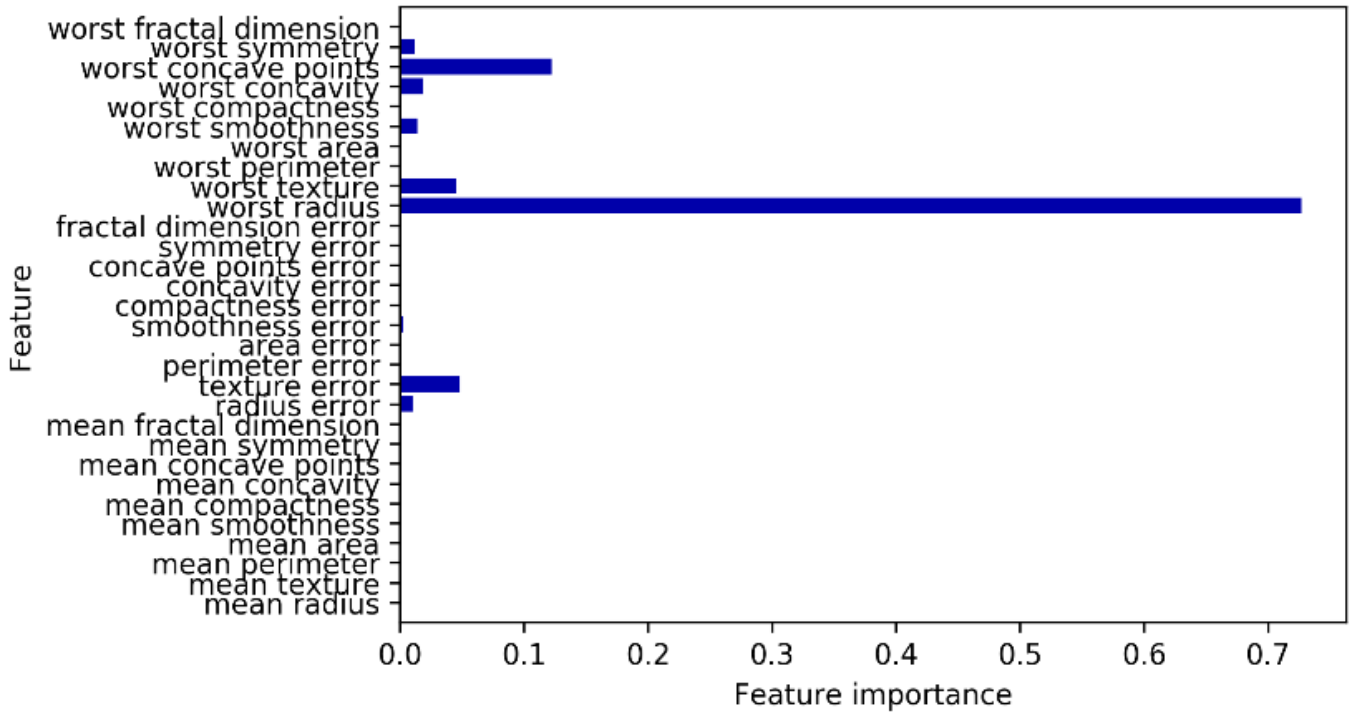
(Refund=No, Marital Status={Married}) ==> No

**Rules are mutually exclusive and exhaustive**

**Rule set contains as much information as the tree**

# Feature Importance

- 나무를 생성하는 과정에서 각 feature가 얼마나 중요하게 사용되었는지를 0과 1 사이의 값으로 수치화한 것
  - 각 feature에 대해, 0은 전혀 사용되지 않았다는 의미
  - 1은 완벽하게 target class를 예측했다는 의미이며, 전체 합은 1이 됨
- 이를 기반으로 한 Feature Selection이 많이 활용됨

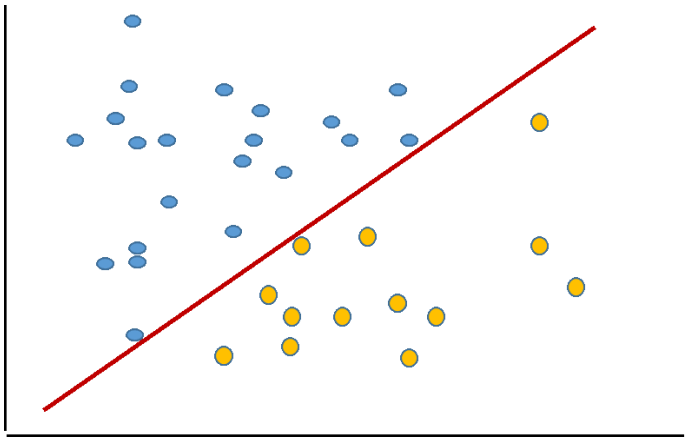


# Pros.

- 결과를 쉽게 이해하고 설명할 수 있으며 의사결정을 하는데 직접적으로 사용할 수 있는 장점이 있기 때문에 Base Model로 널리 사용되고 있다.
- Tree를 분리하는 과정에서 Feature Selection이 자동으로 이루어진다.
- Tree의 상단에 위치한 변수들이 중요한 예측변수이기 때문에 다른 알고리즘을 위한 Feature Selection 방법으로도 사용된다.
- 연속형과 범주형 변수를 모두 다루기 때문에 사전 데이터 준비가 많이 필요하지 않다.
- 분할에 대한 선택이 관측치의 크기에 의해서가 아닌 순서에 의해 결정되기 때문에 Outlier로부터 영향을 거의 받지 않는다.
- 결측 값을 하나의 가지로 다룰 수 있기 때문에 이를 예측에 활용할 수 있다.
  - 예) 가구 수준의 인구통계학적 데이터 내의 비결측값의 수가 정기 생명 보험 제안에 대한 응답률과 양의 상관관계를 보임

# Cons.

- 레코드의 개수의 약간의 차이에 따라 Tree의 모양이 많이 달라질 수 있다. 두 변수가 비슷한 수준의 정보력을 갖는다고 했을 때, 약간의 차이에 의해 다른 변수가 선택되면 이 후의 Tree 구성이 크게 달라질 수 있다.
- 데이터의 특성이 특정 변수에 수직/수평적으로 구분되지 못할 때 분류율이 떨어지고, Tree가 복잡해지는 문제가 발생한다. SVM, Logistic Regression 등의 알고리즘이 여러 변수를 동시에 고려하지만 Decision Tree는 한 개의 변수만을 고려하기 때문에 발생하는 당연한 문제이다.



보충자료

# Decision Tree Plotting

1. graphviz-2.38.msi를 다운받아 설치

2. 아래와 같이 PATH 환경변수를 설정
   ① 바탕화면에서 컴퓨터 아이콘을 마우스 오른쪽 단추로 클릭한 후 속성 선택
   ② 고급 시스템 설정을 선택한 후 환경변수 버튼을 클릭
   ③ 시스템 변수 섹션에서 PATH 환경 변수를 찾아 선택한 후 편집을 클릭
   ④ 변수 값 입력창의 맨 뒤에 ;C:₩Program Files (x86)₩Graphviz2.38₩bin을 추가한 후 확인을 클릭. 확인을 눌러 나머지 창을 모두 닫습니다.
   ⑤ 윈도우를 다시 시작

3. pydotplus 패키지 설치
   ① 윈도우 명령창에서 pip install pydotplus 실행
   ② 또는 Jupyter Notebook에서 !pip install pydotplus 실행

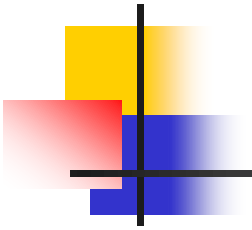
보충자료

# Decision Tree Plotting (Cont.)

```
from sklearn.tree import export_graphviz
from IPython.display import Image
from pydotplus import graph_from_dot_data

dot_data = export_graphviz(tree_model, out_file=None,
                           feature_names=df.columns[1:-1],
                           class_names=['NO','YES'],
                           filled=True, rounded=True,
                           special_characters=True)
graph = graph_from_dot_data(dot_data)
graph.write_pdf("tree_model.pdf")
Image(graph.create_png())
```

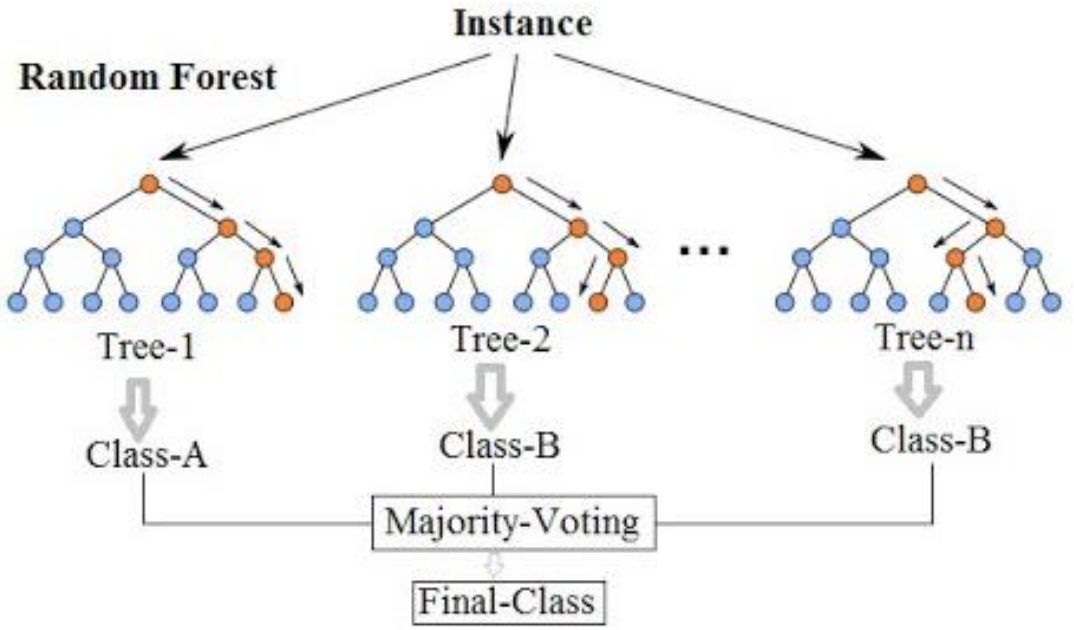
# Random Forest



# Ensemble learning or Classifier ensemble

- Classifier ensemble can be:
  - Different learning algorithms
  - Algorithms with different parameters values
  - Dataset with different features
  - Dataset with different subsets (e.g. bagging)

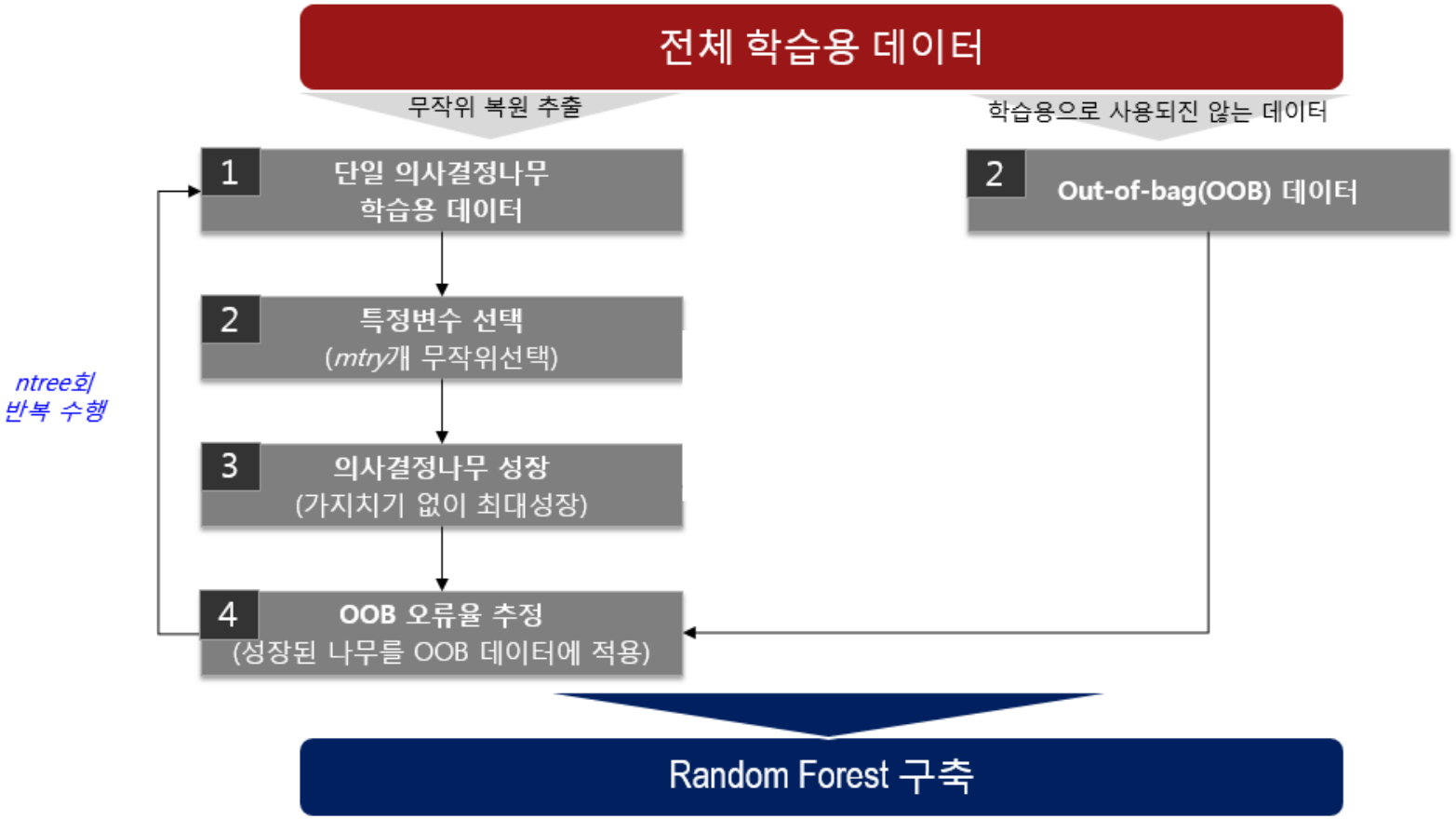
# Random Forest – concept




- Breiman이 2001년에 개발
- 하나가 아닌 여러 개의 나무로 확장시킨 의사결정나무 기반의 Ensemble 기법

Randomization
- Bootstrap samples **(bagging)**
- Random selection of $K \leqslant p$ split variables **(random input selection)**

# Random Forest – algorithm




*Source: 김성진 & 안현철, "랜덤 포레스트를 활용한 기업채권등급평가 모형", 2014.*

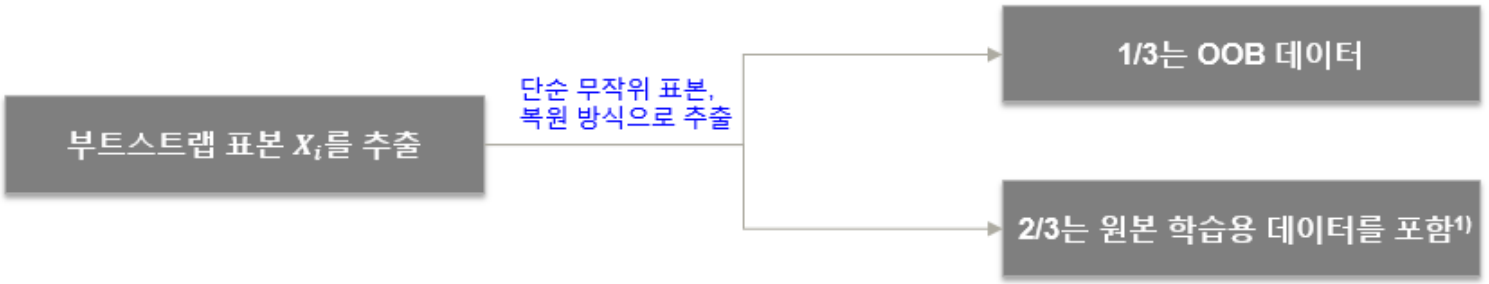
# Random Forest – algorithm

**1** **RF에 적용될 기본 모수에 대한 값을 정한다.**

| 모수 | 설명 |
|---|---|
| ***ntree*** | • RF를 구성할 전체 의사결정나무의 개수를 의미함<br>• 대수의 효과를 얻을 수 있도록 큰 상수 값으로 설정하며, 기본 값으로 500 정도 설정 |
| ***mtry*** | • RF를 구성하는 의사결정나무에서 사용될 특징변수의 개수를 의미함<br>• 전체 특징변수의 개수가 $M$이라고 할 때<br>- 분류를 위한 의사결정나무 $mtry = \sqrt{M}$<br>- 예측을 위한 의사결정나무 $mtry = M/3$ |

# Random Forest – algorithm

## 2 특징 변수 선택




- 부트스트랩 표본 $X_i$($i$=부트스트랩 반복 횟수)를 추출
  (부트스트랩: 주어진 학습 데이터에서 중복을 허용하여 원 데이터와 같은 크기의 데이터를 만드는 과정)
- $X_i$는 전체 학습용 데이터(X)에서 단순 무작위 표본, 복원 방식으로 추출한다.

## 3 의사결정나무 성장(가지치기 없이 최대 성장)

- 부트스트랩 표본 $X_i$를 이용하여 이진 의사결정나무를 성장 시킨다.
- 가지를 분할하기 위한 최적 분할 결정 시 무작위로 선택된 *mtry*개의 특정변수만 사용한다.
- 최적의 분할 시 분할 기준은 Gini[2)] 계수를 사용한다.
- 나무는 가지치기를 하지 않고 최대한 성장시킨다.

1) n개의 데이터가 있을 때 n개의 표본을 단순확률로 반복 추출할 경우 각 데이터 다시 뽑힐 확률( 1-1/e = 0.632)
2) 지니계수가 0에 가까우면 소득분배가 균등하게, 1에 가까우면 불균등하게 이루어졌다는 뜻이다

# Random Forest – algorithm

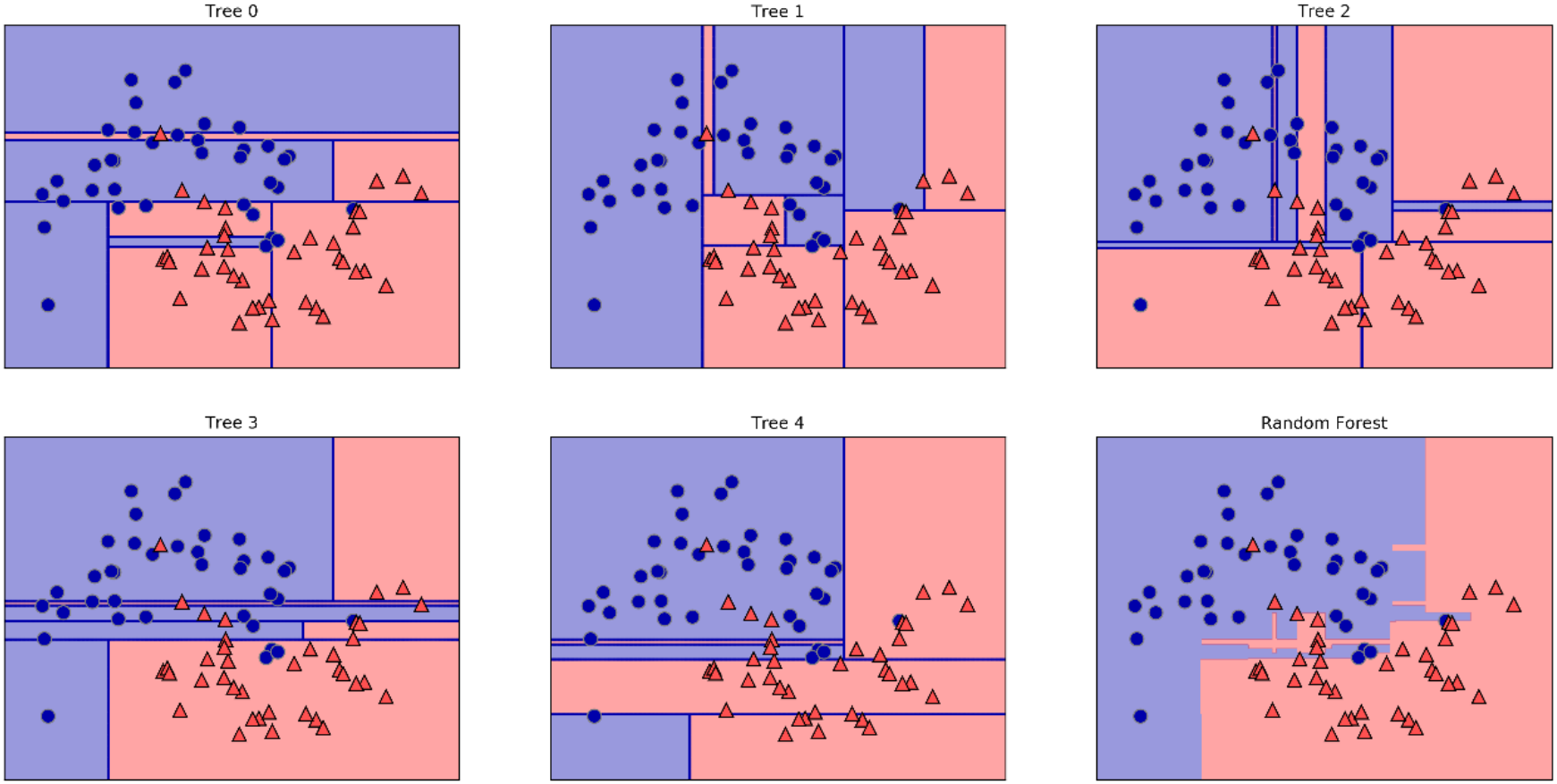
**4 OOB 오류를 추정(성장된 나무를 OOB 데이터에 적용**

- 특정변수 선택 단계에서 추출된 OOB 데이터들에 앞서 구축된 의사결정나무에 적용하여 예측결과를 산출
- 예측결과와 실제결과를 비교하여, OOB 데이터에 대한 오류율을 산출

**5 Random Forest 구축**

- 최종적으로 1~ 4 단계를 *ntree*회 반복하여, ntree개의 개별 의사결정나무로 구성된 RF를 최종 구성
- 개별 의사결정나무결정 나무의 예측 값을 종합하여 산출
  - 종속변수가 연속형인 경우: 평균
  - 종속변수가 범주형인 경우: 투표
- *mtry*와 *ntree*의 적정성: $ERR_{OOB}$를 종합

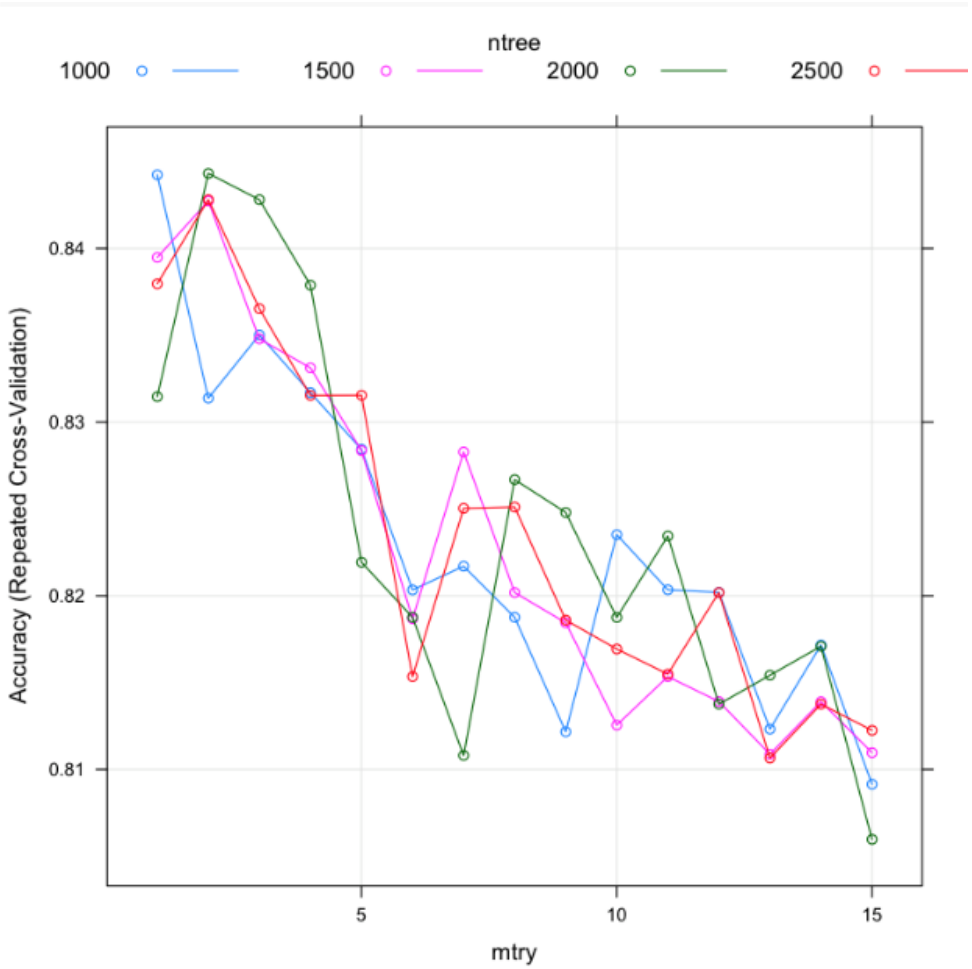
# 앙상블 효과에 따른 결정 경계의 변화



# Pros. & Cons.

- 숲의 크기(나무의 수)가 커질수록 일반화 오류가 특정 값으로 수렴하게 되어 over-fitting을 피할 수 있다.
- 전체 학습용 데이터에서 무작위로 복원 추출된 데이터를 사용함으로써 잡음이나 outlier로부터 크게 영향을 받지 않는다.
- 분석가가 feature 선정으로부터 자유로울 수 있다.
- Class의 빈도가 불균형일 경우 타 기법에 비해 우수한 예측력을 보인다.
- 최종 결과에 대한 해석이 어렵다.
  ⇒ 시각화를 통한 해석을 위해서는 단일 의사결정나무를 사용
- 대량의 데이터를 분석할 경우 상당한 시간이 소요된다.
  ⇒ 멀티 코어 CPU인 경우에 scikit-learn에서는 n_jobs 파라미터를 이용하여 병렬처리가 가능
- feature 차원이 높고 sparse한 데이터에는 잘 작동하지 않는다.
  ⇒ 이런 경우에는 선형모형이 더 적합

# Random Forest – considered heuristics




mtry: Number of variables randomly sampled as candidates at each split.

ntree: Number of trees to grow.

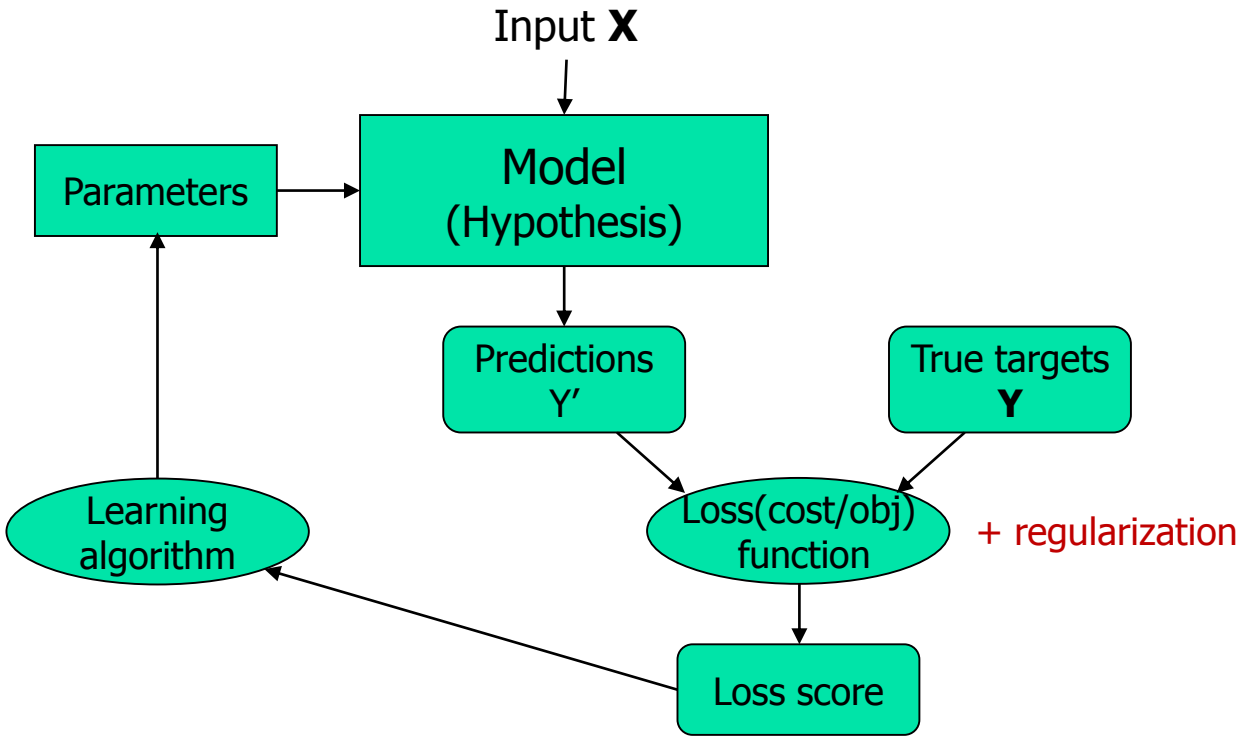
*# scikit-learn에서의 주요 파라미터:*

- *n_estimators: ntree에 해당*
- *max_features: mtry에 해당*
- *max_depth*
- *max_leaf_nodes*

*Source: http://machinelearningmastery.com/tune-machine-learning-algorithms-in-r/*

# Linear Models

# How machine learning works



# Hypothesis

- Hypothesis

$$\hat{y} = h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n = \boldsymbol{\theta}^T \boldsymbol{x}$$

  - $\hat{y}$: 예측값
  - $x_j$ : j번째 feature 값
  - $\theta_j$: j번째 parameter 값
  - n: feature의 개수

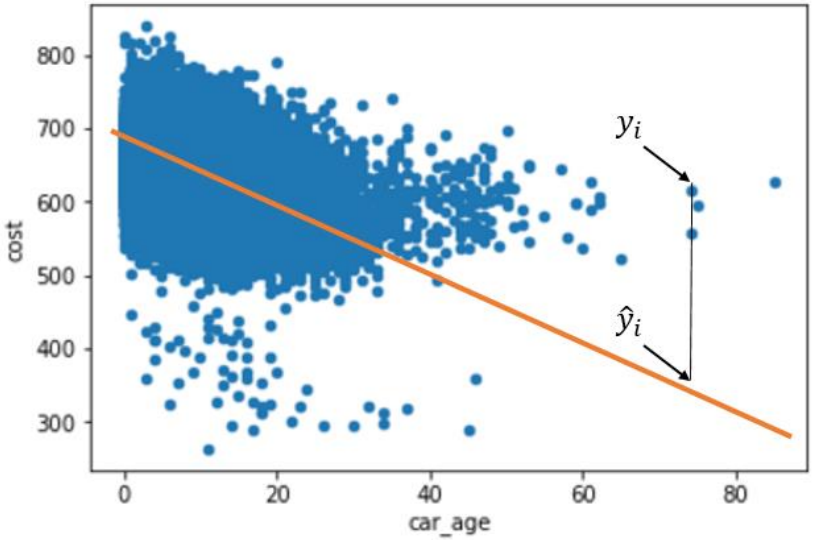
- m개의 training example (관측치)를 사용하여 hypothesis를 학습
  - $(\boldsymbol{x}^{(1)}, \boldsymbol{y}^{(1)}), (\boldsymbol{x}^{(m)}, \boldsymbol{y}^{(m)}), \cdots, (\boldsymbol{x}^{(m)}, \boldsymbol{y}^{(m)})$

# Cost Function

- Cost function

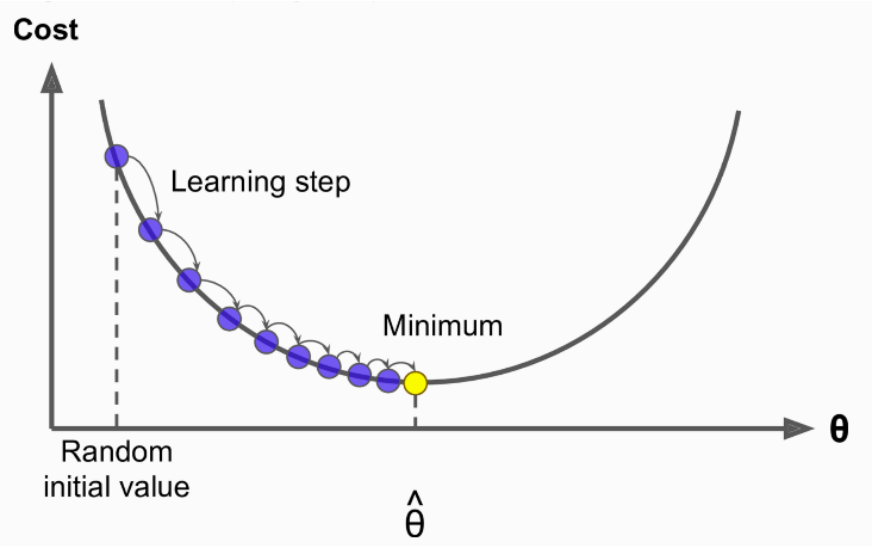
$$J(\theta) = MSE(\boldsymbol{x}, h_\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)} - h_\theta\left(\boldsymbol{x}^{(i)}\right)\right)^2$$

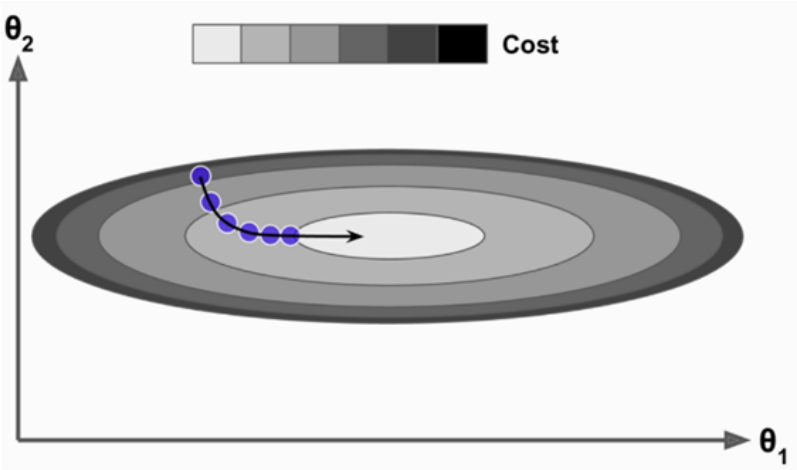
- Cost function을 최소화 하는 $\theta_j$ 들을 어떻게 찾는가?
  - Least square estimation (LSE)
  - Gradient descent 알고리즘



# Learning algorithm

- Gradient Descent
  - Cost 함수의 최소값을 찾아가는 numerical optimization 방법
  - initial value에서 시작하여 가장 기울기가 가파른 방향으로 매 step 이동
  - $\theta^{(t+1)} = \theta^{(t)} - \eta \frac{\partial J(\theta^{(t)})}{\partial \theta^{(t)}}$ $\eta$: learning rate $(0 \sim 1)$




*Source: Figure 4-3 in Geron(2017)*

# Regularized Linear Models

- Linear model의 overfitting을 방지하는 다른 방법
- 모형의 복잡도가 증가하는 것에 대한 penalty 고려

Ridge regression: $l_2$ penalty 사용
Lasso regression: $l_1$ penalty 사용
Elastic Net: ridge와 lasso의 결합

# Ridge Regression

- Cost function

$$J(\theta) = MSE(\theta) + \boldsymbol{\alpha} \cdot \frac{1}{2}\sum_{i=1}^{n} \theta_i^2$$

$\theta$ 벡터의 $l_2$-norm
➔ $l_2$ penalty 라고 부름

- $\alpha$: regularization parameter (hyperparameter, tuning parameter)
  - Parameter가 가능한 작아지도록 penalty 부여
  - $\alpha = 0$: 일반 linear regression과 동일

- $\alpha$의 선택
  - 여러 개의 $\boldsymbol{\alpha}$ 값에 대해 model fitting 후 CV cost $J_{cv}(\hat{\theta})$가 가장 작은 model 선택
  - 어떤 $\alpha$값을 try 해볼 것인가? 2배씩 증가

    $\alpha = 0,\ \alpha = 0.01,\ \alpha = 0.02,\ \alpha = 0.04,\ \alpha = 0.08,\ \cdots,\ \alpha = 10$

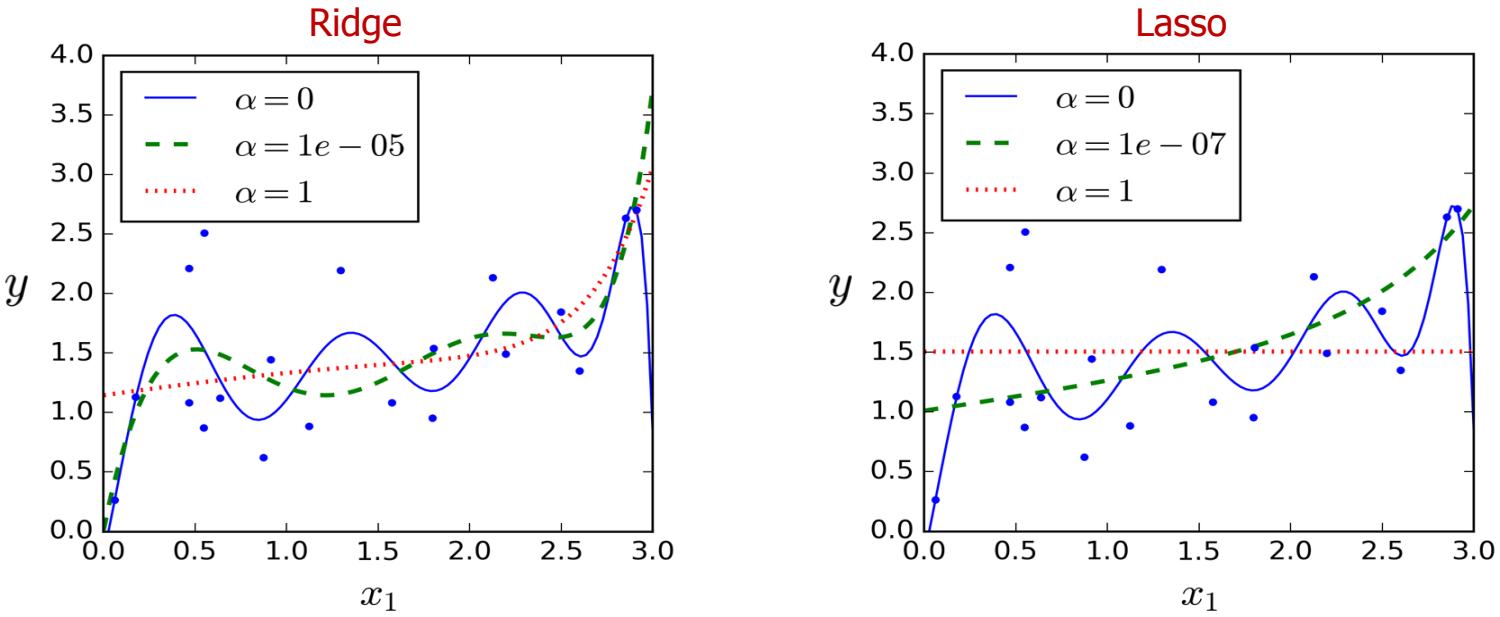
# Lasso Regression

- Least Absolute Shrinkage and Selection Operator Regression (LASSO)
- Cost function

$$J(\theta) = MSE(\theta) + \boldsymbol{\alpha} \cdot \boxed{\sum_{i=1}^{n} |\theta_i|}$$

$l_1$ penalty

# Ridge vs. Lasso




*Source: Figure 4-17,18 in Geron(2017)*

- Ridge는 $\theta_i$를 전체적으로 축소시키는 효과
- Lasso는 덜 중요한 feature에 대해 $\theta_i = 0$으로 만드는 효과
  - 자동적인 feature selection 과정
- Lasso가 대체적으로 더 작은 최적 $\alpha$를 가지는 경향

# Elastic Net

- Ridge와 Lasso의 중간
- Cost function

$$J(\theta) = MSE(\theta) + r\alpha \cdot \sum_{i=1}^{n} |\theta_i| + \frac{1-r}{2}\alpha \cdot \sum_{i=1}^{n} \theta_i^2$$

- $r$: $l_1$ **penalty**의 비율
  - $r = 0$: Ridge regression
  - $r = 1$: Lasso regression
- **Ridge, Lasso & Elastic net**
  - Ridge는 좋은 default
  - 소수의 feature만 남기길 바라면 Lasso나 Elastic net 사용
  - Lasso는 특성수 > 샘플수 또는 feature 간의 강한 상관관계가 있을 때 error 발생 가능

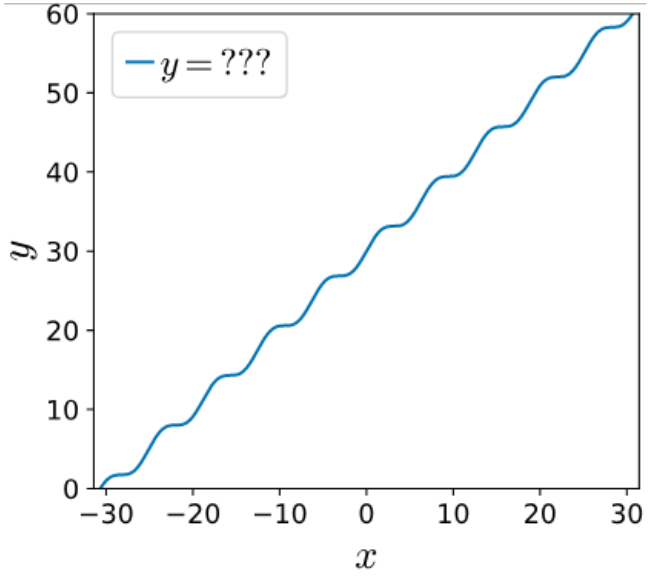
# Gradient Boosting Machines

*(Source) "How to explain gradient boosting" by Terence Parr & Jeremy Howard*

# What is Boosting?

- Ensemble meta-algorithm used to convert many weak learners into a strong learner
- The general idea of most boosting methods is to train predictors sequentially, each trying to correct its predecessor.
- There are many boosting methods available, but by far the most popular is Gradient Boosting
- Gradient Boosting works by sequentially adding predictors to an ensemble, each one correcting its predecessor. This method tries to fit the new predictor to the residual errors made by the previous predictor.

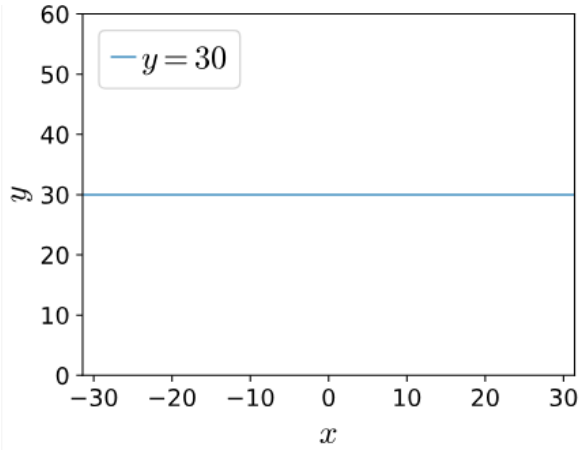
# An introduction to additive modeling

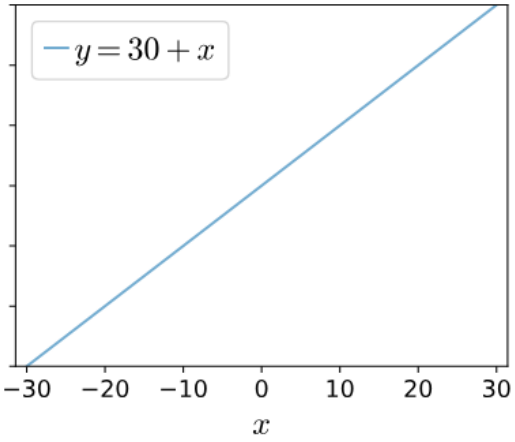


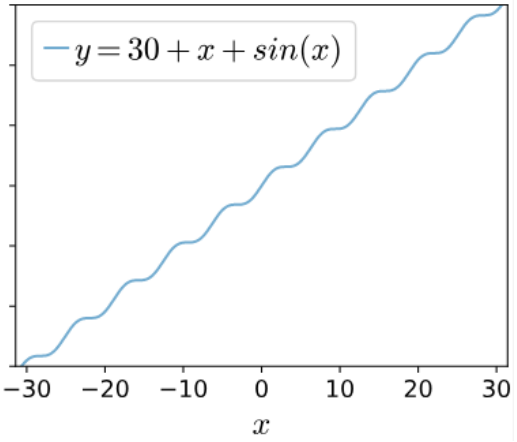

$$F(x) = f_1(x) + f_2(x) + f_3(x)$$

where:

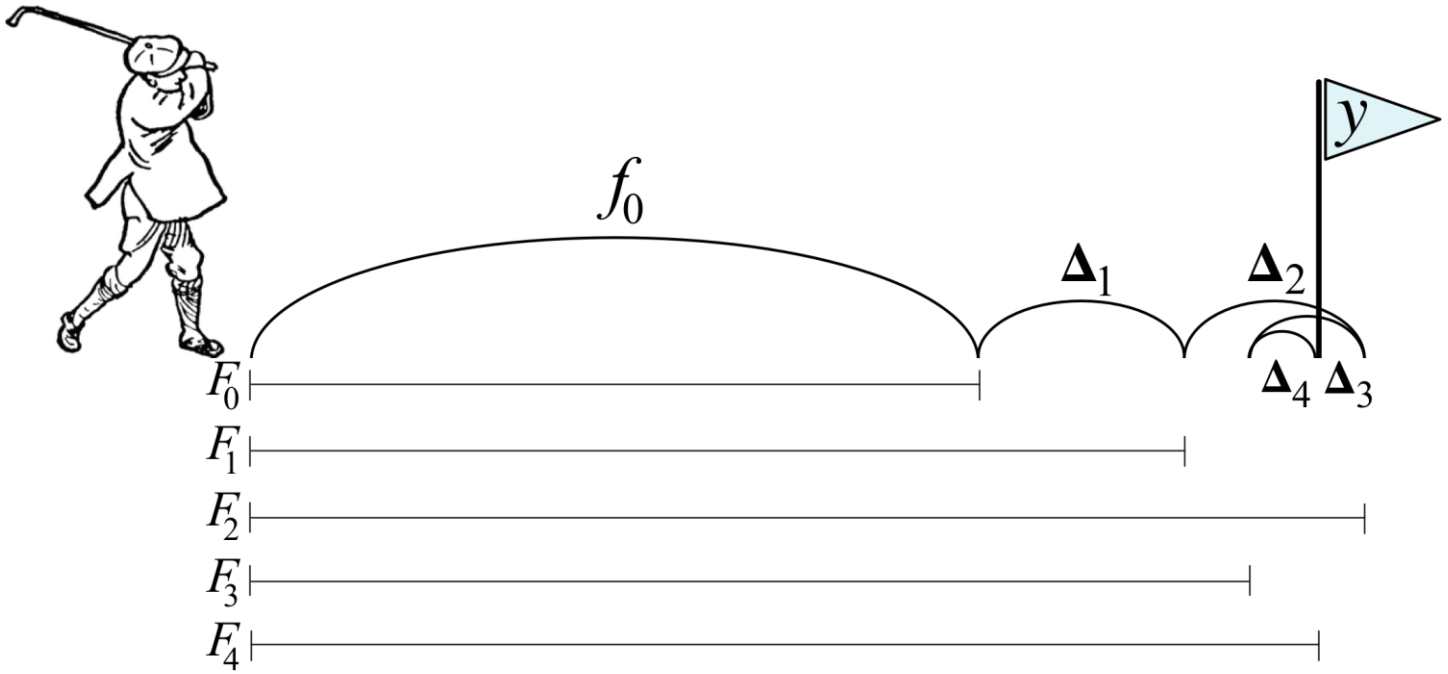
$$f_1(x) = 30$$
$$f_2(x) = x$$
$$f_3(x) = sin(x)$$



# The intuition behind gradient boosting



# The intuition behind gradient boosting

$$\begin{aligned}\hat{y} &= f_0(\mathbf{x}) + \Delta_1(\mathbf{x}) + \Delta_2(\mathbf{x}) + \ldots + \Delta_M(\mathbf{x}) \\ &= f_0(\mathbf{x}) + \sum_{m=1}^{M} \Delta_m(\mathbf{x}) \\ &= F_M(\mathbf{x})\end{aligned}$$

Or

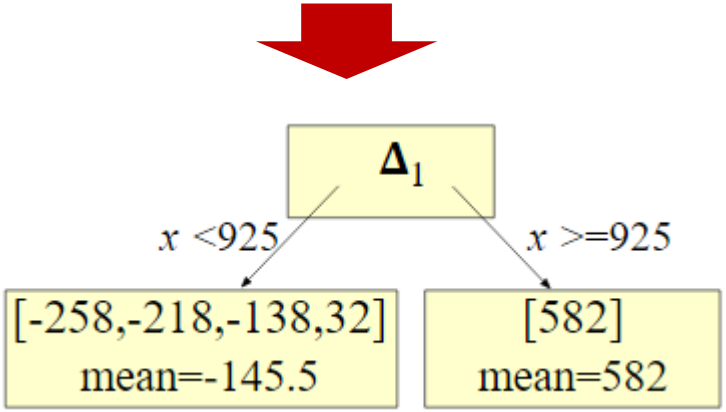
$$\begin{aligned}F_0(\mathbf{x}) &= f_0(\mathbf{x}) \\ F_m(\mathbf{x}) &= F_{m-1}(\mathbf{x}) + \Delta_m(\mathbf{x})\end{aligned}$$

*We can add incremental shrinkage (learning rate) $\eta$ :*

$$F_m(X) = F_{m-1}(X) + \eta\Delta_m(X)$$

# Gradient boosting by example

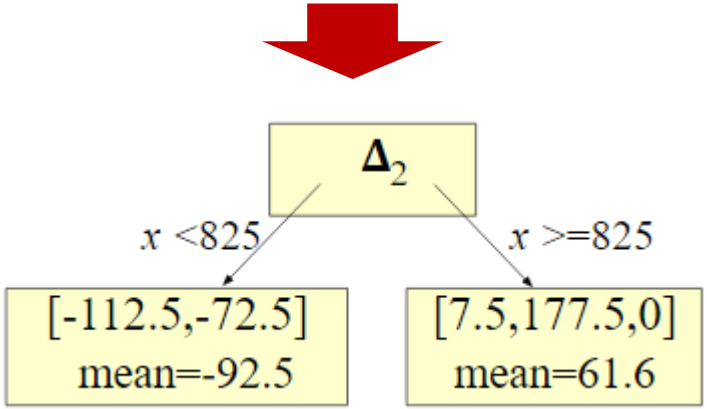
| sqfeet | rent | $F_0$ | y $- F_0$ |
|---|---|---|---|
| 750 | 1160 | 1418 | -258 |
| 800 | 1200 | 1418 | -218 |
| 850 | 1280 | 1418 | -138 |
| 900 | 1450 | 1418 | 32 |
| 950 | 2000 | 1418 | 582 |




| $\Delta_1$ | $F_1$ | y-$F_1$ |
|---|---|---|
| -145.5 | 1272.5 | -112.5 |
| -145.5 | 1272.5 | -72.5 |
| -145.5 | 1272.5 | 7.5 |
| -145.5 | 1272.5 | 177.5 |
| 582 | 2000 | 0 |

# Gradient boosting by example

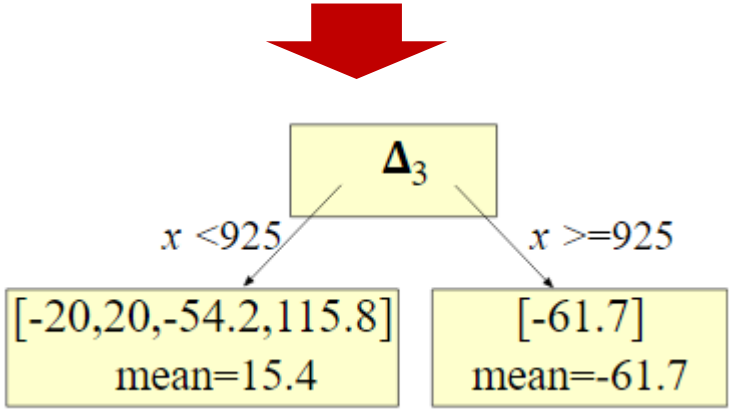
| $\Delta_1$ | $F_1$ | y-$F_1$ |
|---|---|---|
| -145.5 | 1272.5 | -112.5 |
| -145.5 | 1272.5 | -72.5 |
| -145.5 | 1272.5 | 7.5 |
| -145.5 | 1272.5 | 177.5 |
| 582 | 2000 | 0 |




| $\Delta_2$ | $F_2$ | y - $F_2$ |
|---|---|---|
| -92.5 | 1180 | -20 |
| -92.5 | 1180 | 20 |
| 61.7 | 1334.2 | -54.2 |
| 61.7 | 1334.2 | 115.8 |
| 61.7 | 2061.7 | -61.7 |

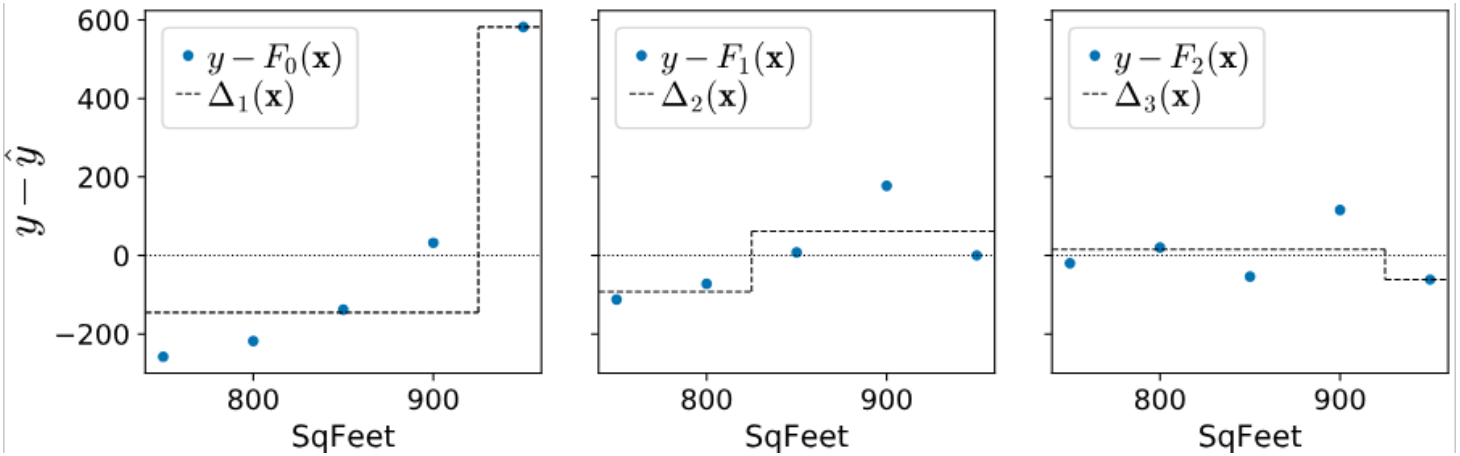
# Gradient boosting by example

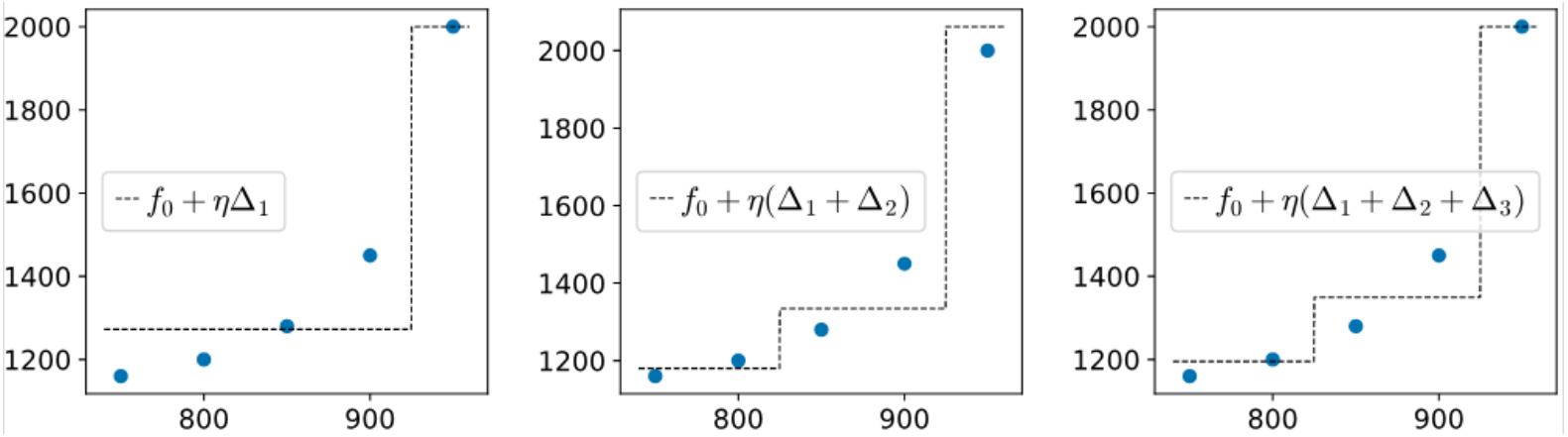
| $\Delta_2$ | $F_2$ | $\mathbf{y} - F_2$ |
|---|---|---|
| -92.5 | 1180 | -20 |
| -92.5 | 1180 | 20 |
| 61.7 | 1334.2 | -54.2 |
| 61.7 | 1334.2 | 115.8 |
| 61.7 | 2061.7 | -61.7 |




| $\Delta_3$ | $F_3$ |
|---|---|
| 15.4 | 1195.4 |
| 15.4 | 1195.4 |
| 15.4 | 1349.6 |
| 15.4 | 1349.6 |
| -61.7 | 2000 |

# Gradient boosting by example



# GBM algorithm

**Algorithm:** *l2boost*($X$,$\mathbf{y}$,$M$,$\eta$) **returns** model $F_M$

Let $F_0(X) = \frac{1}{N}\sum_{i=1}^{N} y_i$, mean of target $\mathbf{y}$ across all observations

**for** $m = 1$ **to** $M$ **do**

  Let $\mathbf{r}_{m-1} = \mathbf{y} - F_{m-1}(X)$ be the residual direction vector

  Train regression tree $\Delta_m$ on $\mathbf{r}_{m-1}$, minimizing squared error

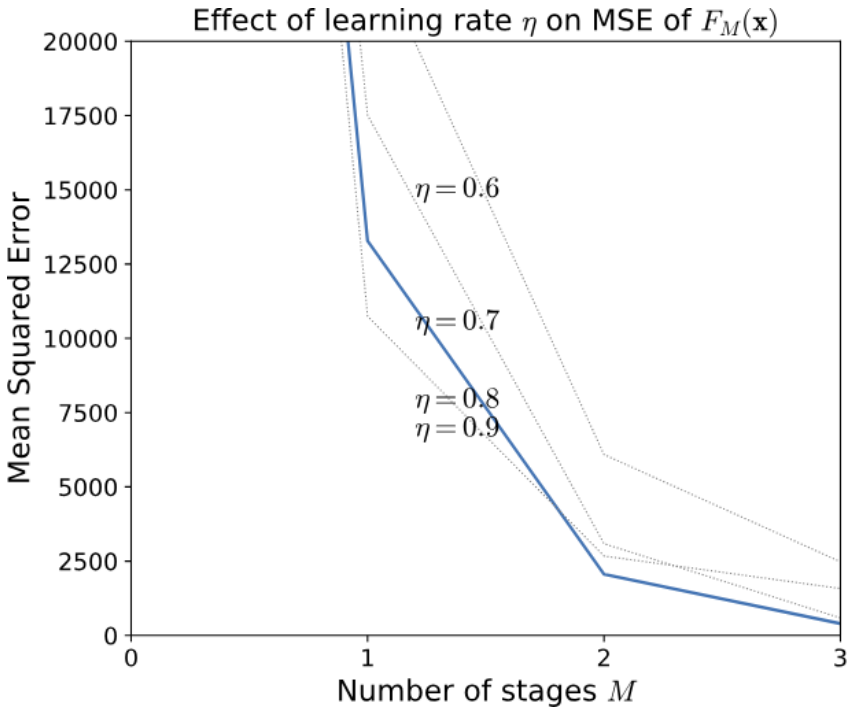
  $F_m(X) = F_{m-1}(X) + \eta\Delta_m(X)$

**end**

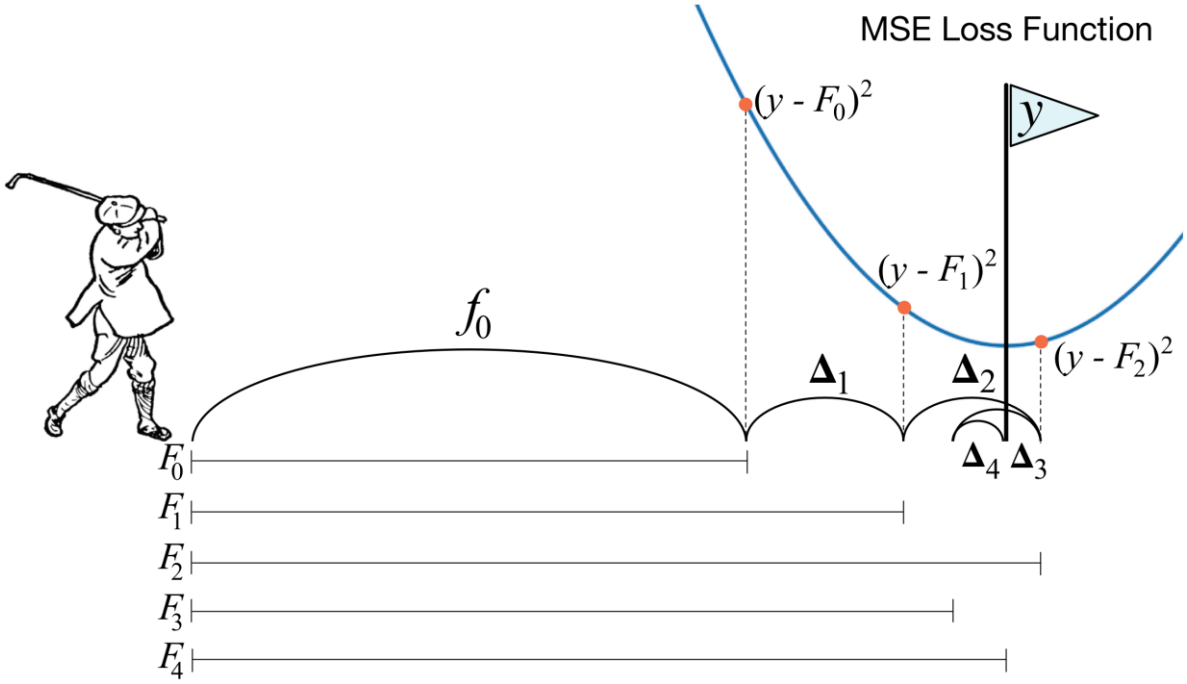
**return** $F_M$

# Choosing hyperparameters

- ✓ *The number of stages and the learning rate affect model accuracy.*
- ✓ *The more stages we use, the more accurate the model, but the more likely we are to be overfitting.*
- ✓ *The primary value of the learning rate, or "shrinkage" is to reduce overfitting of the overall model.*



# Gradient boosting performs gradient descent




**Gradient descent**

$$\mathbf{x}_t = \mathbf{x}_{t-1} - \eta \nabla f(\mathbf{x}_{t-1})$$

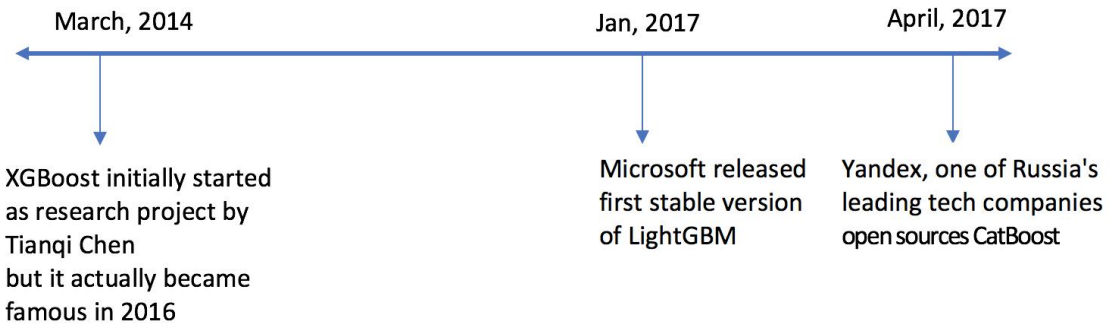
**Gradient boosting**

$$\hat{\mathbf{y}}_m = \hat{\mathbf{y}}_{m-1} + \eta(-\nabla L(\mathbf{y}, \hat{\mathbf{y}}_{m-1}))$$

$\boldsymbol{y - \hat{y}_{m-1}}$ => residual

# What is XGBoost?

- Optimized gradient-boosting machine learning library
- Originally written in C++
- Has APIs in several languages:
  - Python
  - R
  - Scala
  - Julia
  - Java




*Source: http://www.datacamp.com/*

# What makes XGBoost so popular?

- Speed and performance
- Core algorithm is parallelizable
- Consistently outperforms single-algorithm methods
- State-of-the-art performance in many ML tasks

*Source: http://www.datacamp.com/*

# Using XGBoost: A Quick Example

```
In [1]: import xgboost as xgb

In [2]: import pandas as pd

In [3]: import numpy as np

In [4]: from sklearn.model_selection import train_test_split

In [5]: class_data = pd.read_csv("classification_data.csv")

In [6]: X, y = class_data.iloc[:,:-1], class_data.iloc[:,-1]

In [7]: X_train, X_test, y_train, y_test= train_test_split(X, y,
        test_size=0.2, random_state=123)

In [8]: xg_cl = xgb.XGBClassifier(objective='binary:logistic',
        n_estimators=10, seed=123)

In [9]: xg_cl.fit(X_train, y_train)

In [10]: preds = xg_cl.predict(X_test)

In [11]: accuracy = float(np.sum(preds==y_test))/y_test.shape[0]
```

# When should I use XGBoost?

## When to use XGBoost

- You have a large number of training samples
  - Greater than 1000 training samples and less 100 features
  - The number of features < number of training samples
- You have a mixture of categorical and numeric features
  - Or just numeric features

## When to NOT use XGBoost

- Image recognition & Computer vision
- Natural language processing and understanding problems
- When the number of training samples is significantly smaller than the number of features

*Source: http://www.datacamp.com/*

# Tunable parameters

- learning rate: learning rate/eta
  - This scales the contribution of each tree. If you set it to a low value, you will need more trees in the ensemble to fit the training set
- gamma: min loss reduction to create new tree split
- lambda: L2 reg on leaf weights
- alpha: L1 reg on leaf weights
- max_depth: max depth per tree
- subsample: % samples used per tree
- colsample_bytree: % features used per tree

$$Obj = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k)$$

Training loss — Complexity of the Trees

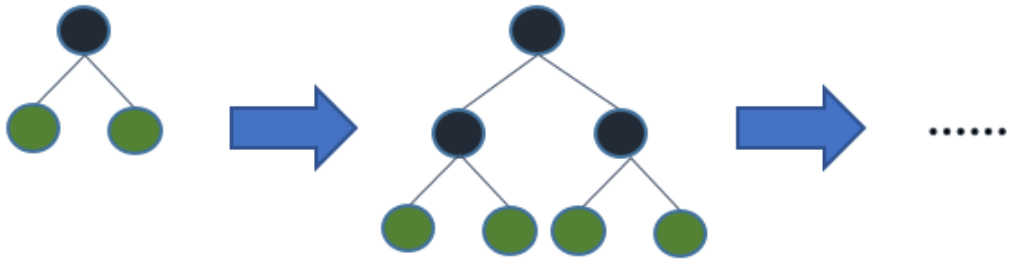
$$\Omega(f_t) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

Number of leaves — L2 norm of leaf scores

Source: http://www.datacamp.com/

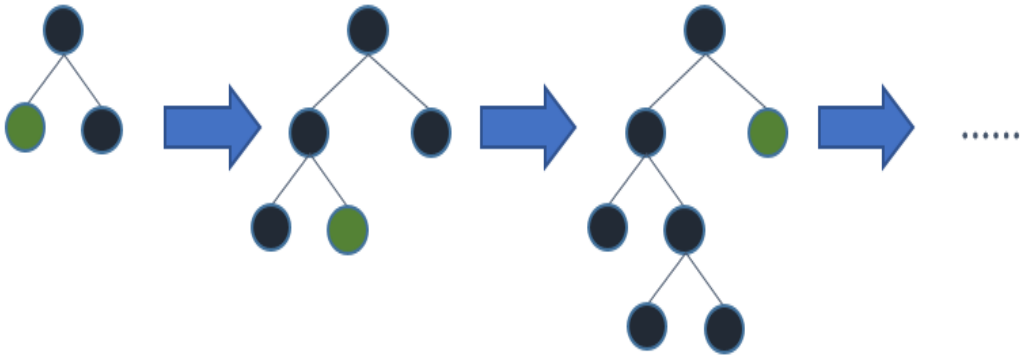
# XGBoost vs. Light GBM

- How to create the trees
  - Light GBM splits the tree leaf wise with the best fit whereas other boosting algorithms split the tree depth wise




Level-wise tree growth

- Advantages of Light GBM
  - Faster training speed and higher efficiency
  - Lower memory usage
  - Better accuracy than any other boosting algorithm
  - Parallel learning supported




Leaf-wise tree growth

Source: https://www.analyticsvidhya.com/blog/2017/06/which-algorithm-takes-the-crown-light-gbm-vs-xgboost/

| Function | XGBoost | Light GBM |
|---|---|---|
| Important parameters which control overfitting | 1. **learning_rate or eta** – optimal values lie between 0.01-0.2<br>2. **max_depth**<br>3. **min_child_weight:** similar to min_child leaf; default is 1 | 1. **learning_rate**<br>2. **max_depth**: default is 20. Important to note that tree still grows leaf-wise. Hence it is important to tune **num_leaves** (number of leaves in a tree) which should be smaller than 2^(max_depth). It is a very important parameter for LGBM<br>3. **min_data_in_leaf**: default=20, alias= min_data, min_child_samples |
| Parameters for categorical values | Not Available | 1. **categorical_feature:** specify the categorical features we want to use for training our model |
| Parameters for controlling speed | 1. **colsample_bytree**: subsample ratio of columns<br>2. **subsample**: subsample ratio of the training instance<br>3. **n_estimators**: maximum number of decision trees; high value can lead to overfitting | 1. **feature_fraction:** fraction of features to be taken for each iteration<br>2. **bagging_fraction:** data to be used for each iteration and is generally used to speed up the training and avoid overfitting<br>3. **num_iterations:** number of boosting iterations to be performed; default=100 |