ML_4

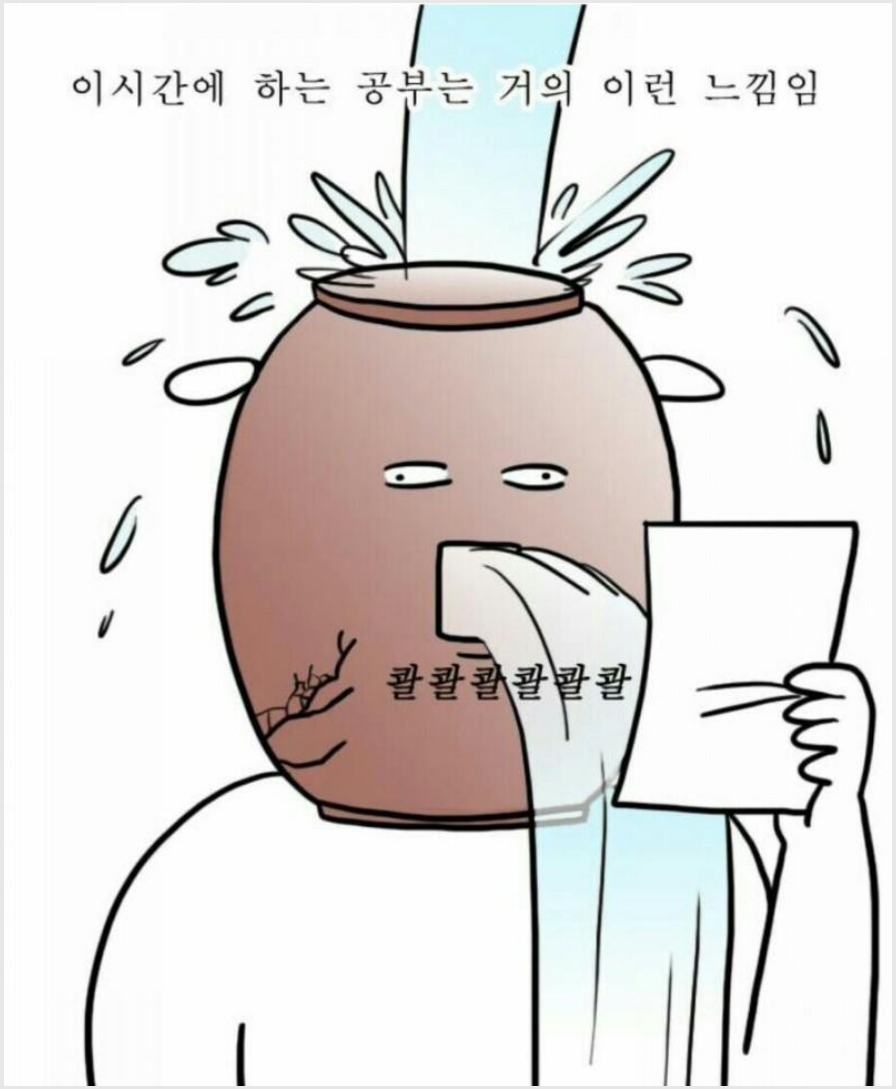
Team BMS

INDEX

ML Study
4 Week

Ice Breaking

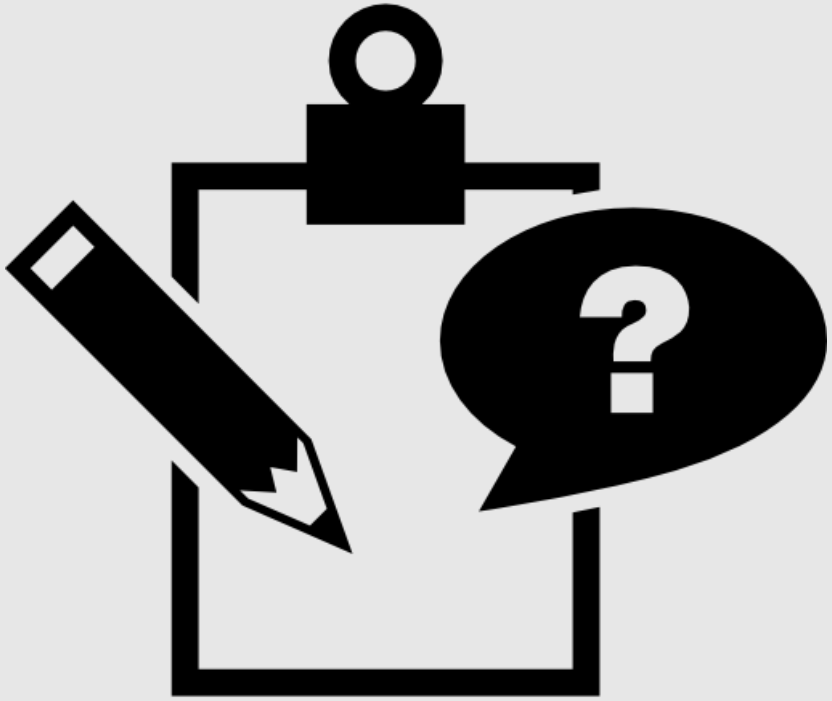
Ice Breaking



Ice Breaking

01 I. Ice Breaking

Before Start

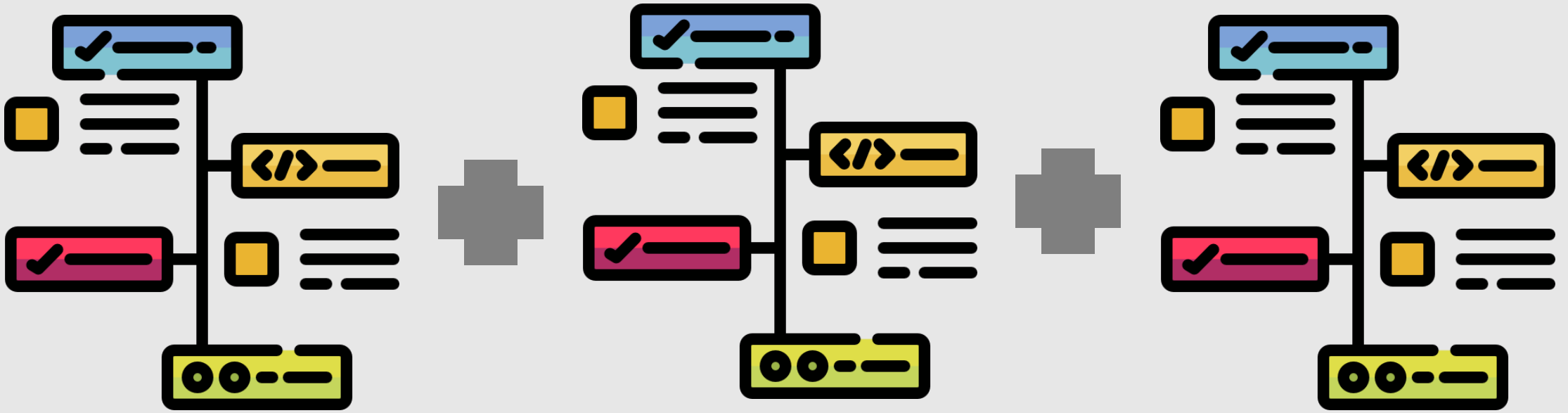


Ensemble Learning and Random Forests

Ensemble Learning

Ensemble?

Ensemble Learning



여러 가지 동일한 종류의 혹은 서로 상이한 모형들의 예측/분류 결과를 종합하여
최종적인 의사결정에 활용하는 방법론

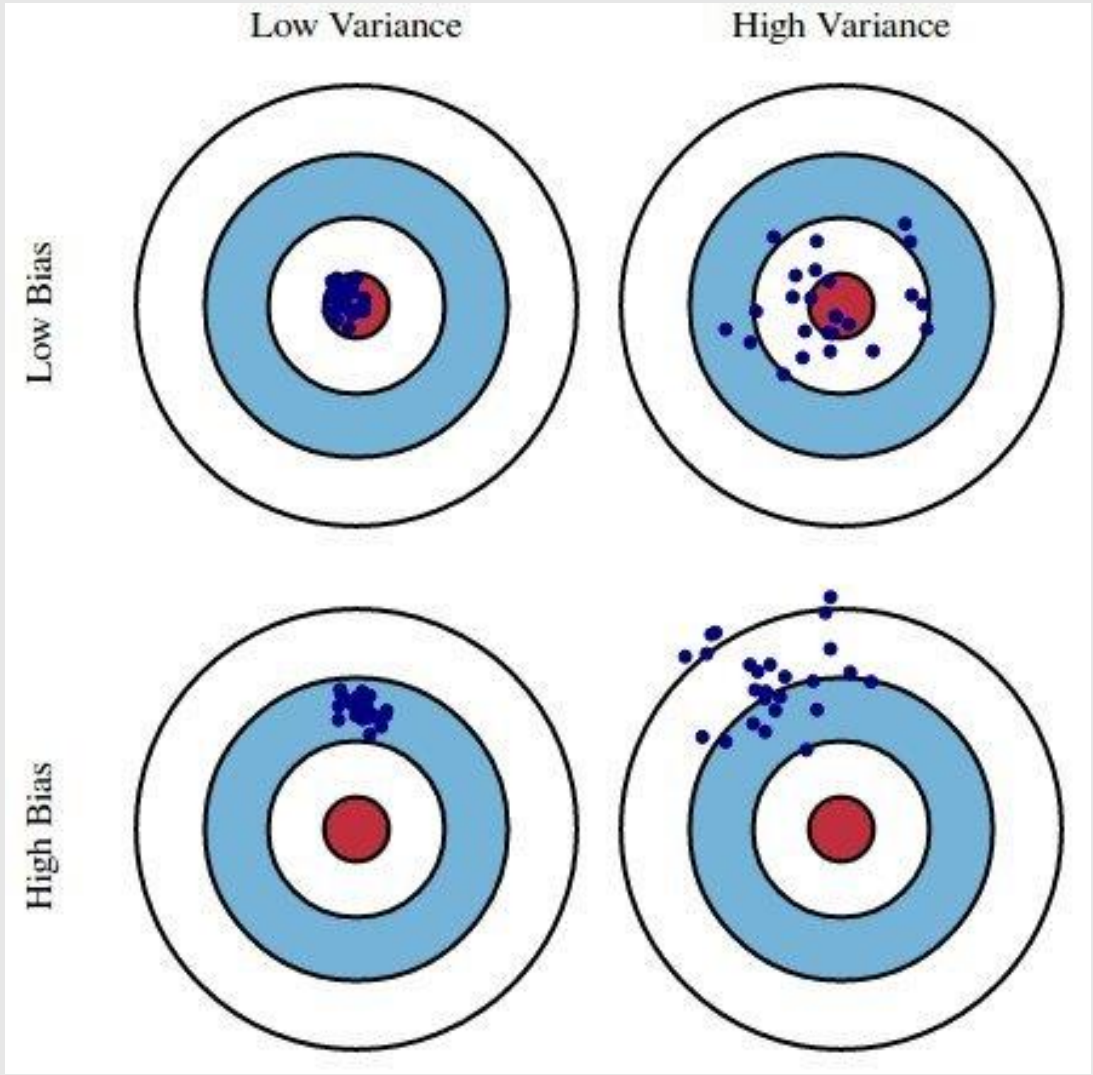
Ensemble Learning

- 예를 들어 두 집단을 분류하는 분류기가 5개 있고,
  각각의 오분류율이 5%라고 가정할 때, 만약 해당 모형들이 모두 동일한 결정을 내린다고 한다면 모형의 오분류율은 5%가 됨
- 하지만, 각각의 분류기가 상호독립적이어서 전체 분류기의 절반 이상이 오분류를 하는 경우에 앙상블 모형의 오분류율은 0.01%로 떨어지게 됨

$$E = \sum_{i=3}^{5} (0.05)^i (1 - 0.05)^{5-i} = 0.0001$$

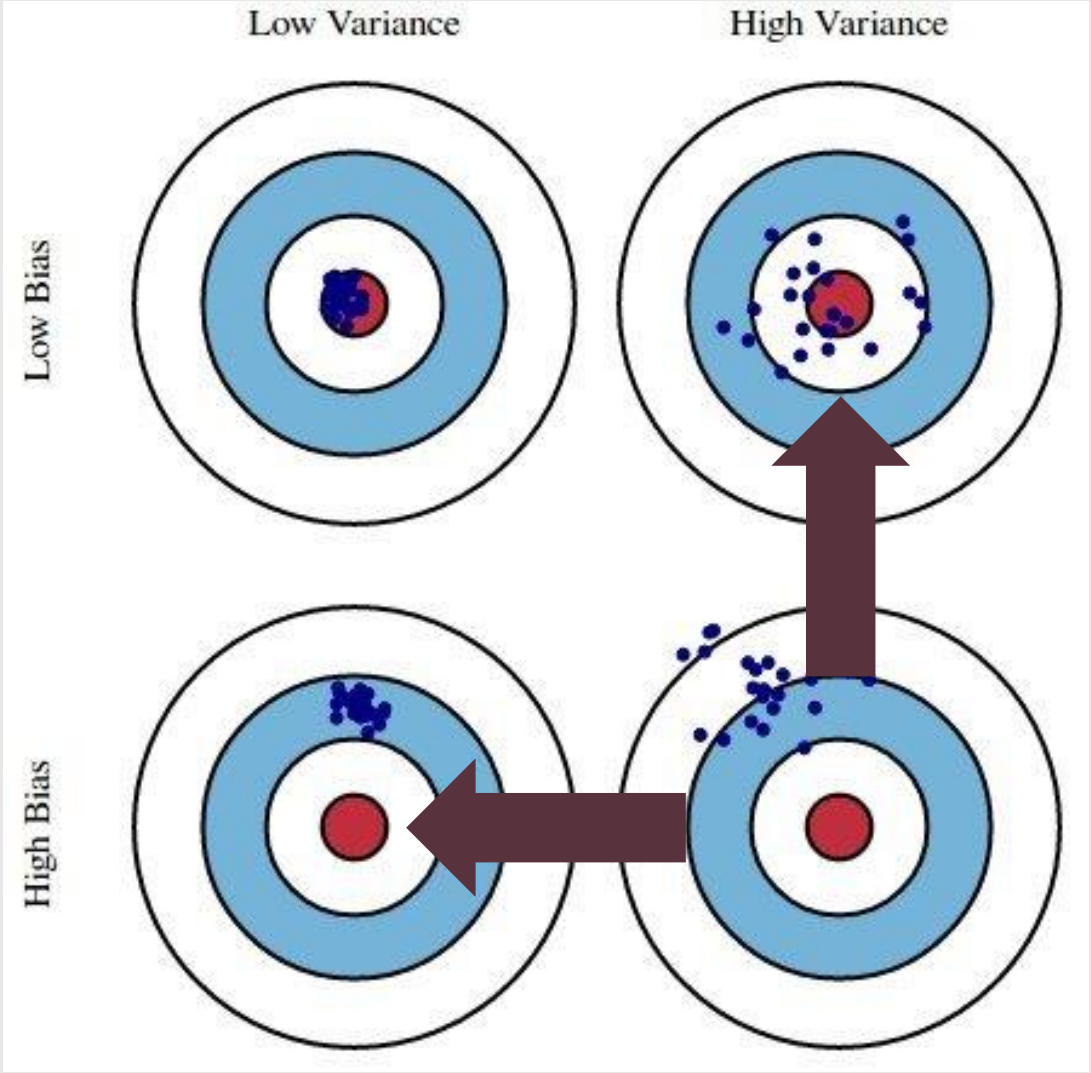
- 이는 이상치에 대한 대응력을 높이고 전체적인 분산을 감소시키기 때문에 정분류율이 높아지는 것으로 알려짐
- 단점으로는 모형이 블랙박스가 된다는 것이 있다.

Ensemble Learning

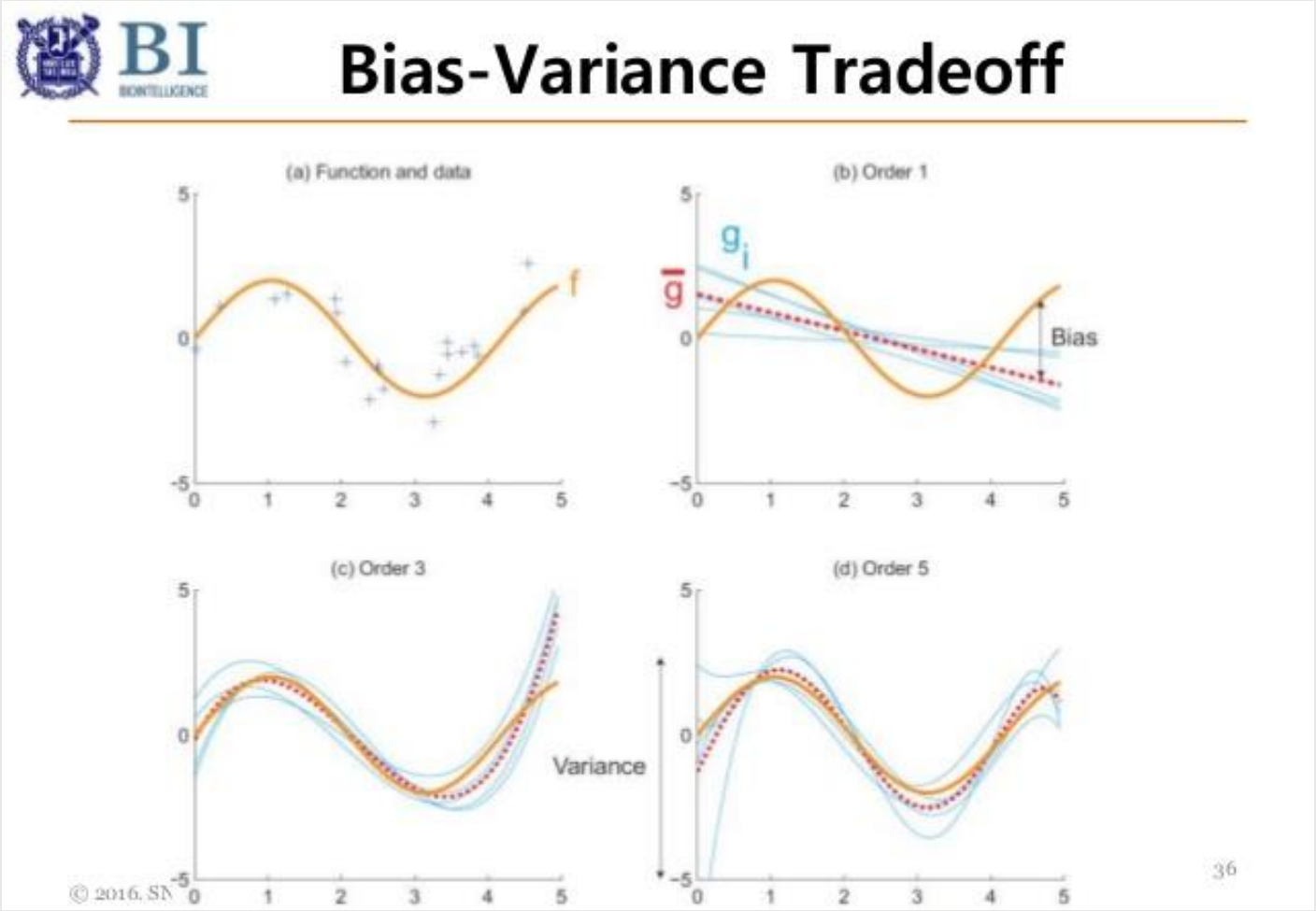


〈https://www.kdnuggets.com/2016/08/bias-variance-tradeoff-overview.html〉

Ensemble Learning

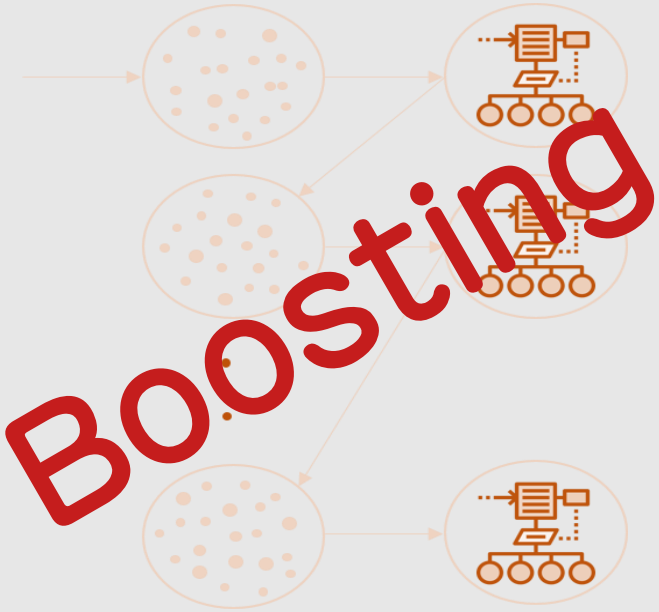


Ensemble Learning

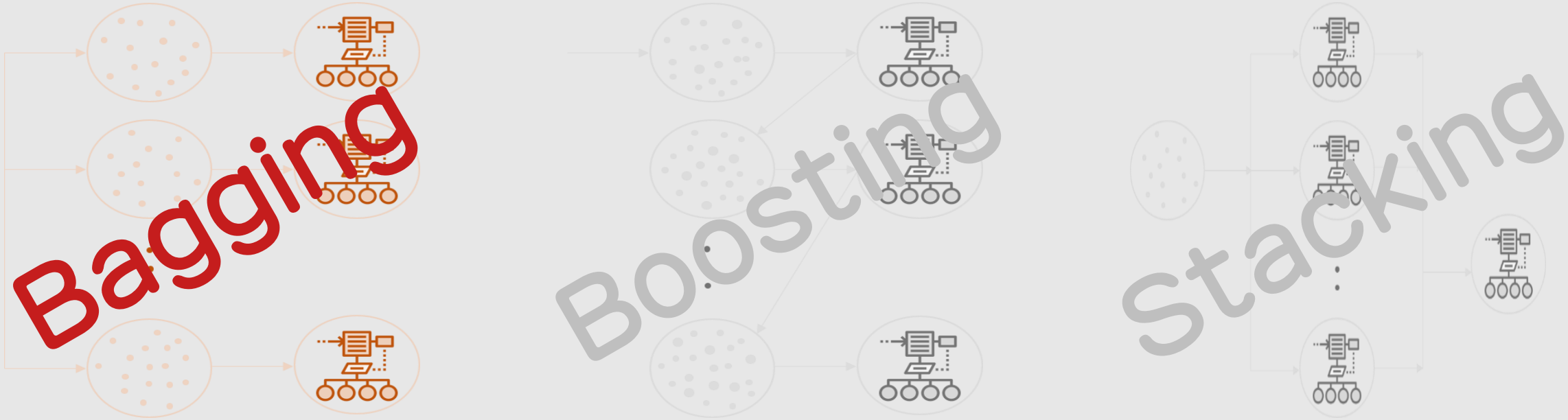


〈https://www.slideshare.net/donghyunkwak90/prml-3132-linear-regression-biasvariance-decomposition〉

Ensemble Learning



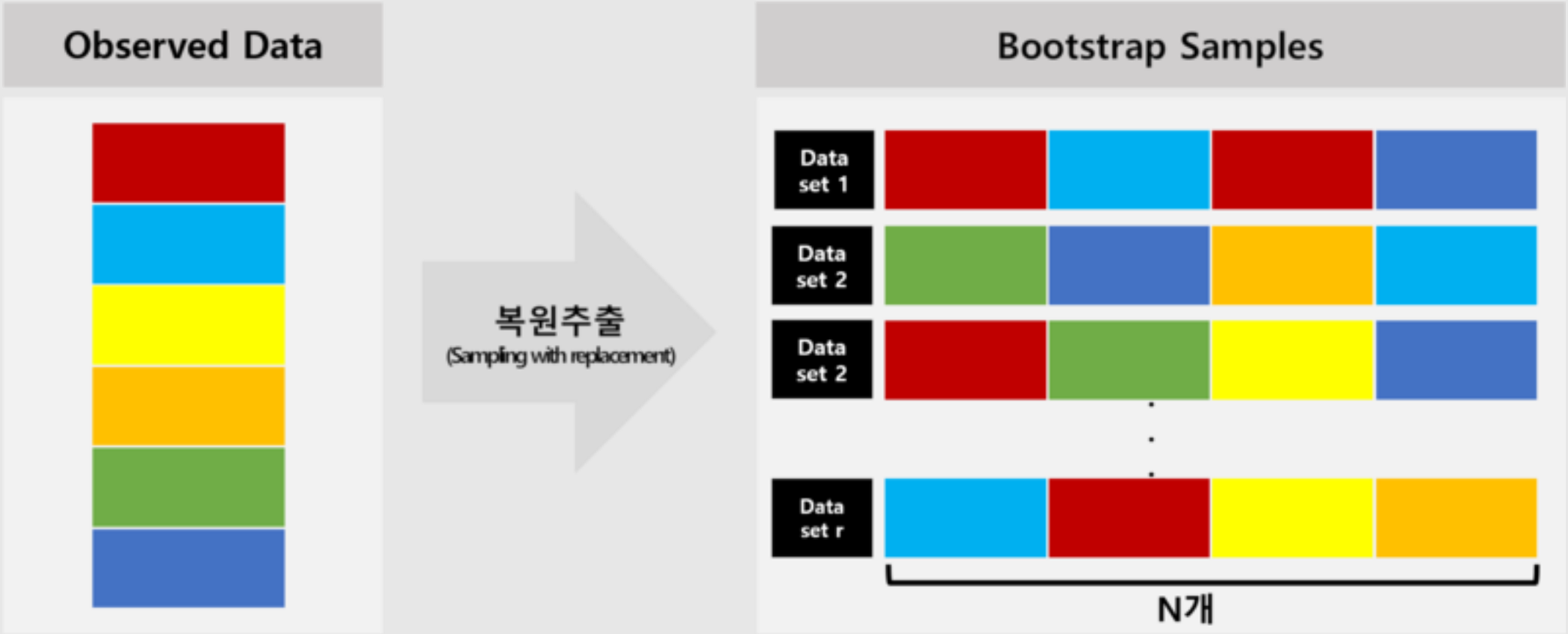
Ensemble Learning



Bootstrap Sample

Bootstrap CSS Framework?

Bootstrap Sample



〈https://m.blog.naver.com/PostView.nhn?blogId=gksshdk8003&logNo=220862646751&proxyReferer=https%3A%2F%2Fwww.google.com%2F〉

Montecarlo Cross-validation

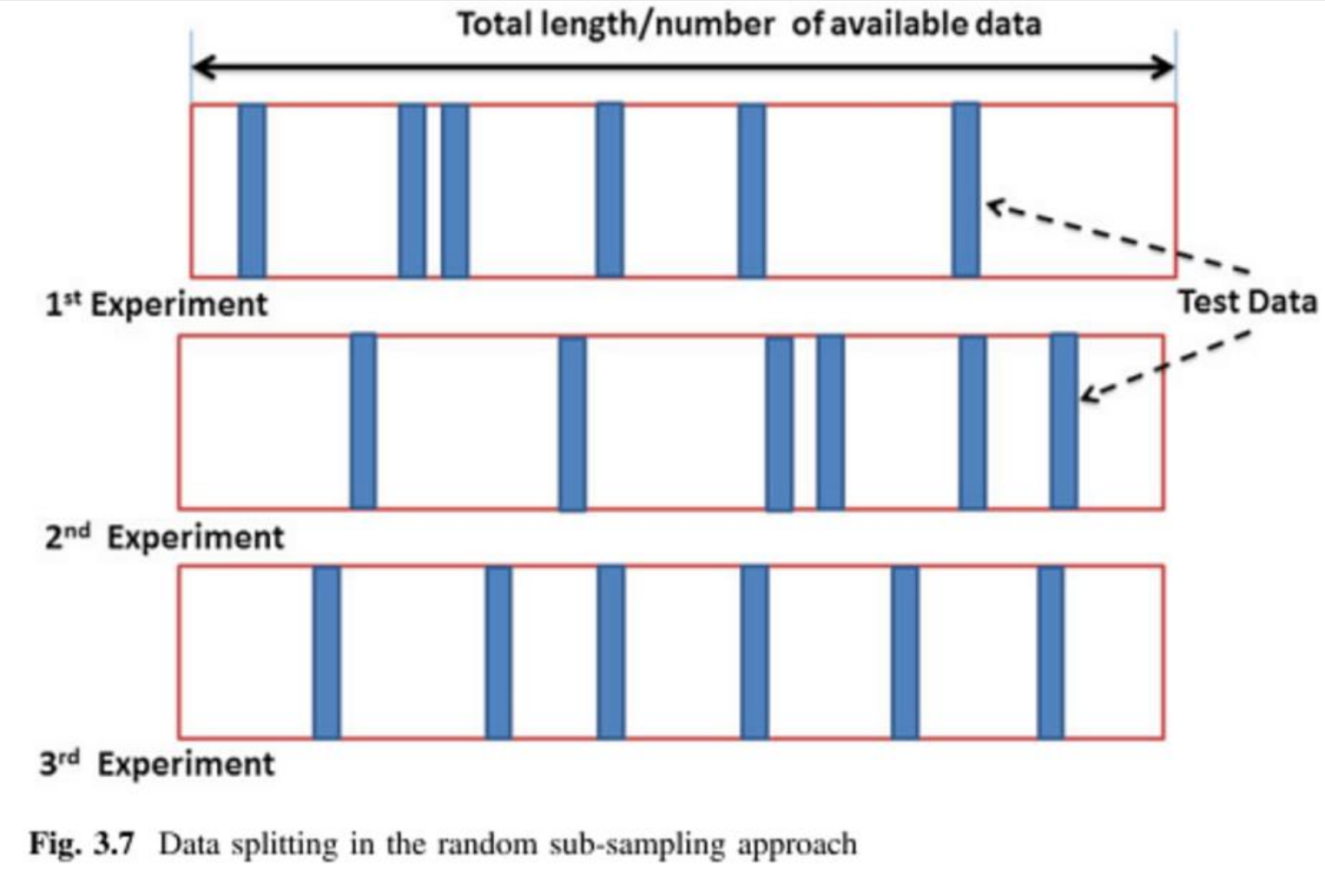


Fig. 3.7 Data splitting in the random sub-sampling approach

〈https://stats.stackexchange.com/questions/51416/k-fold-vs-monte-carlo-cross-validation〉

Montecarlo

〈https://www.silversea.com/destinations/mediterranean-cruise/monte-carlo-to-venice-3916.html〉

Montecarlo

물리적, 수학적 시스템의 행동을 시뮬레이션 하기 위한 계산 알고리즘이다.
다른 알고리즘과는 달리 통계학적이고, 일반적으로 무작위의 숫자를 사용한 비결정적인 방법이다.
〈위키피디아〉

동명의 카지노에서 따온 이름을 가진, 무작위 추출된 난수를 이용하여
함수의 값을 계산하는 통계학의 방법.
최적화, 수치적분, 확률분포로부터의 추출 등에 쓰인다.
무작위 추출된 난수를 이용하여 원하는 함수의 값을 계산하기 위한 시뮬레이션 방법이다.
자유도가 높거나 닫힌 꼴(closed form)의 해가 없는 문제들에 널리 쓰이는 방법이지만,
어느 정도의 오차를 감안해야만 하는 특징이 있다.
〈나무위키〉

어떤 값을 계산할 때 난수를 이용해 확률적인 계산을 하는 것
〈zzing0907 블로그〉

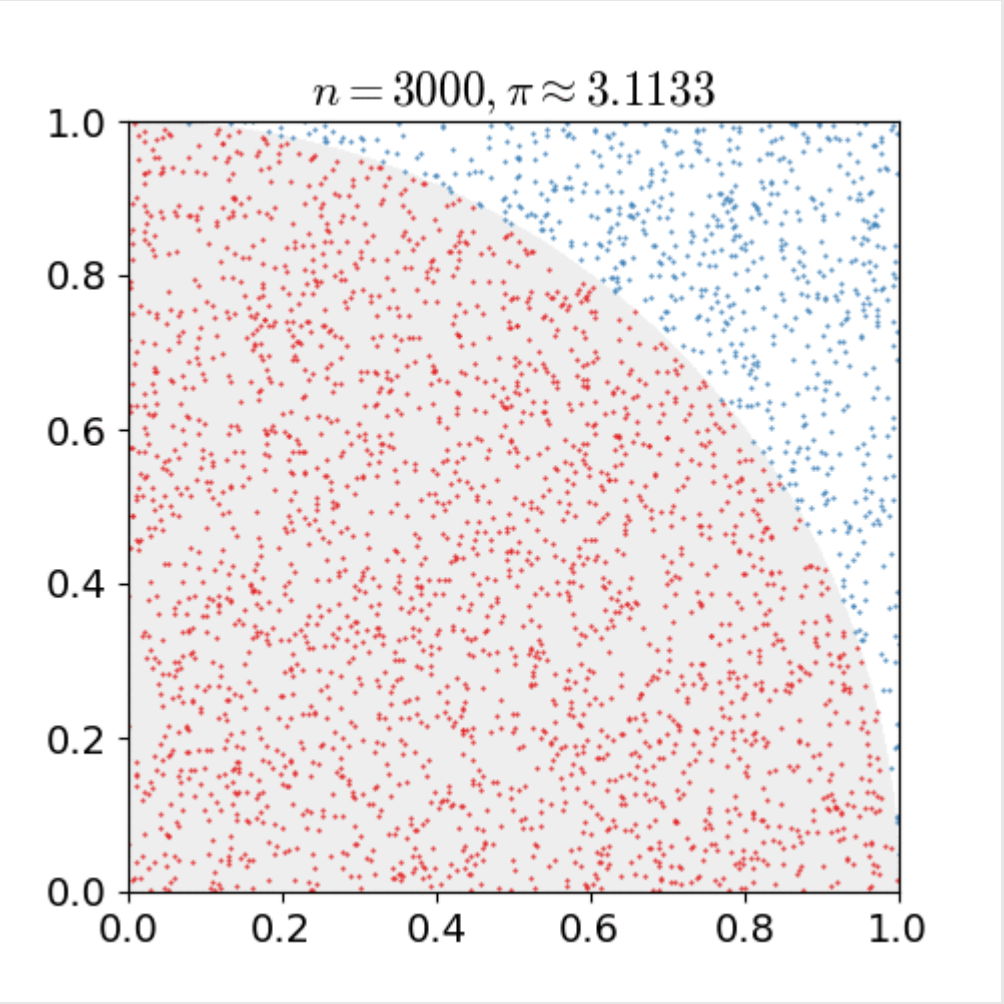
Montecarlo

흥미롭게도 Monte Carlo method 은 진짜 임의의 수를 사용할 필요는 없다.
대부분의 유용한 기술은 deterministic, pseudo-random sequences를 사용하며
test and re-run simulations 를 쉽게 만든다. 좋은 시뮬레이션을 만드는데 필요한
진짜 중요한 것은 pseudo-random sequence가 "충분히 임의적인 것으로 (random
enough)" 보이게 하는 것이다. 즉 충분히 많은 수의 요소들의 순서가 고려될 때
균일 분포 (uniformly distributed)하거나 다른 바람직한 분포를 따라야 한다는 것이다.

알고리즘의 반복과 많은 수의 계산이 포함되기 때문에, Monte Carlo는 컴퓨터를 사용하여 계산하기에 적당한 방법이며, 컴퓨터 시뮬레이션의 많은 기술을 사용하기에 적당한 방법이다.

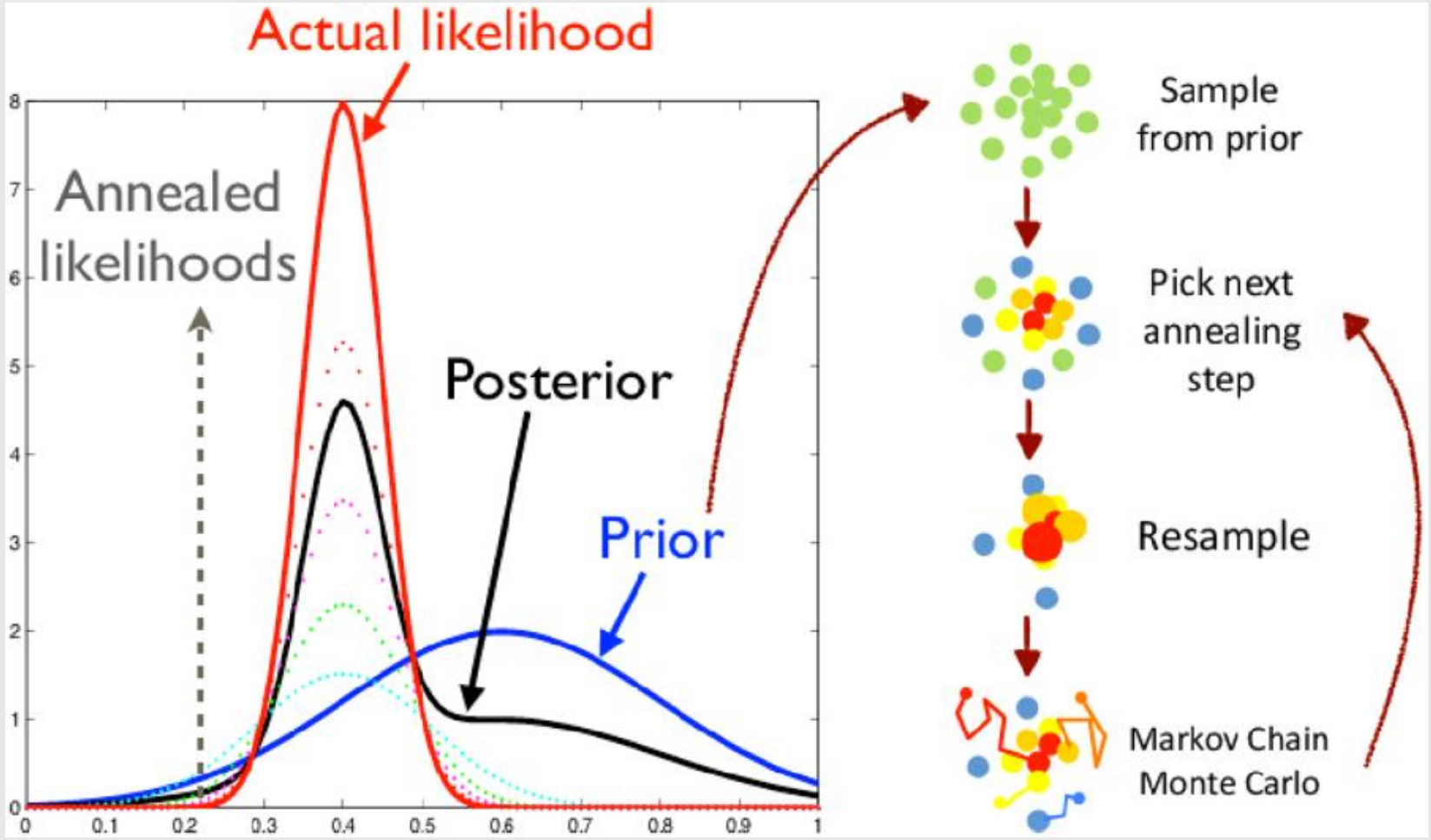
〈AI Study〉

Montecarlo Method Example



〈http://codingdojang.com/scode/507〉
(gif 파일)

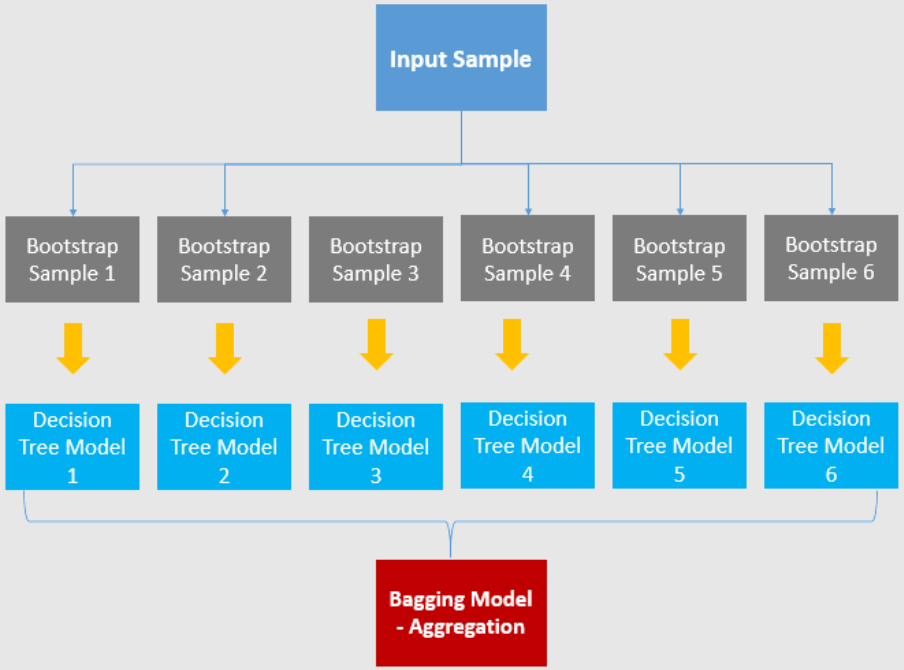
Montecarlo Method Example



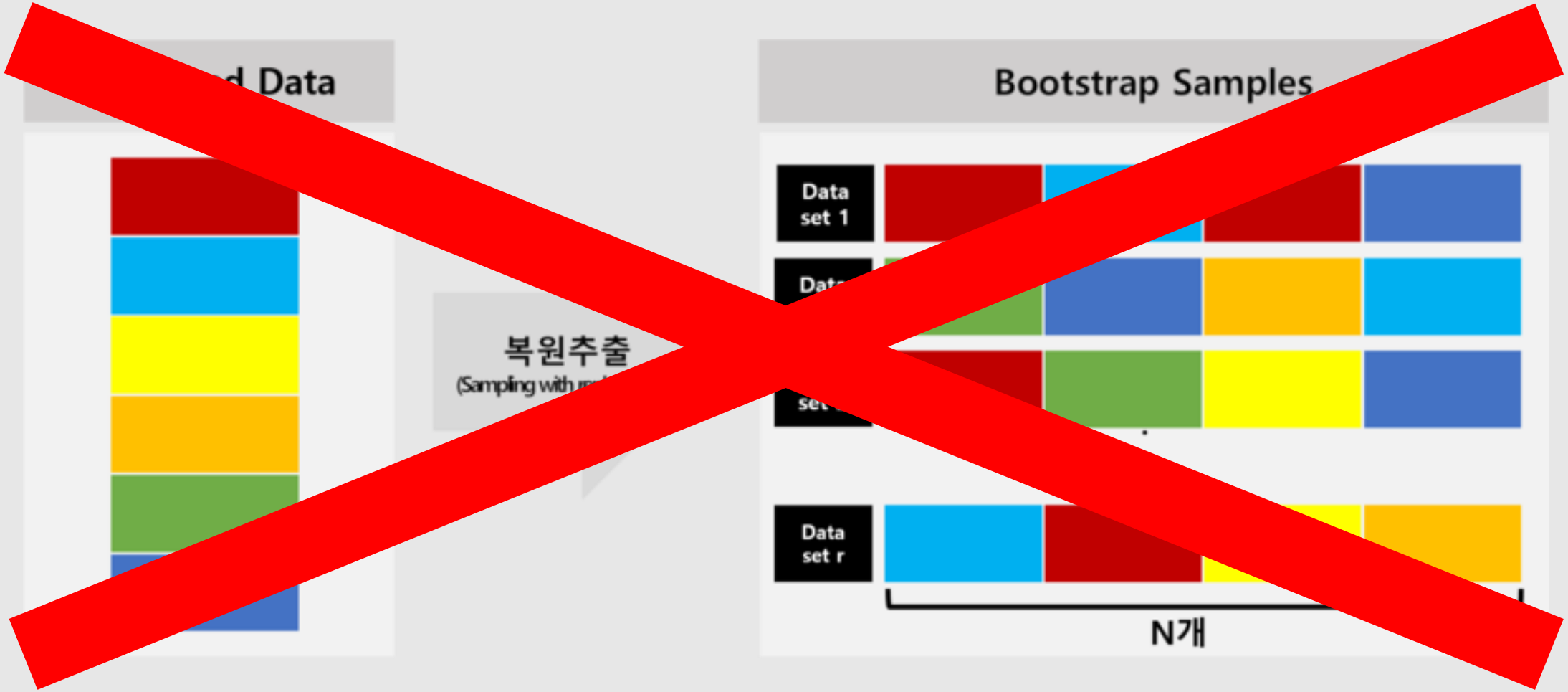
〈http://risk.asmedigitalcollection.asme.org/article.aspx?articleid=2647605〉

Bagging

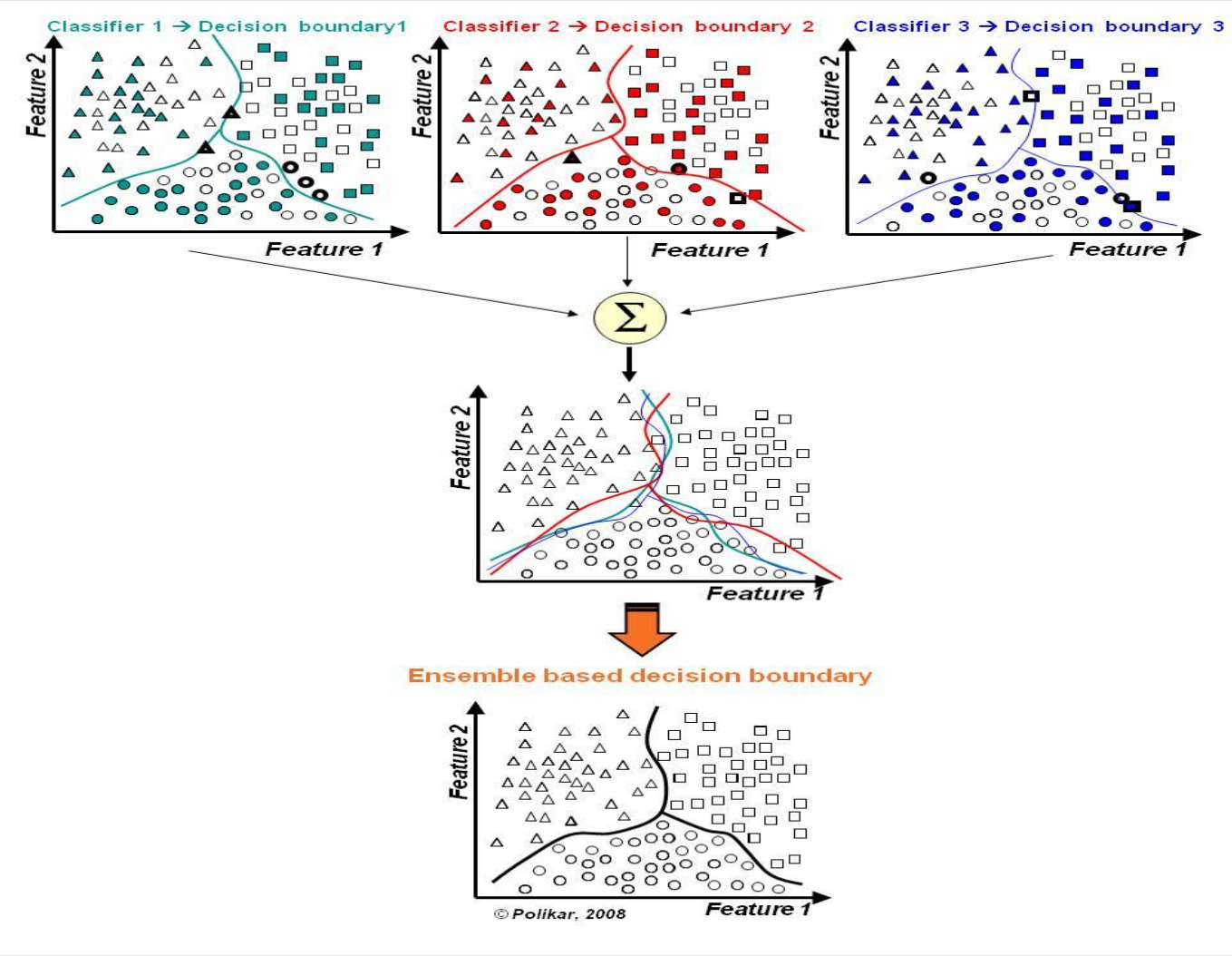
Bagging =
Bootstrap Aggregating
(부트스트랩 샘플링 + 집계)



Pasting

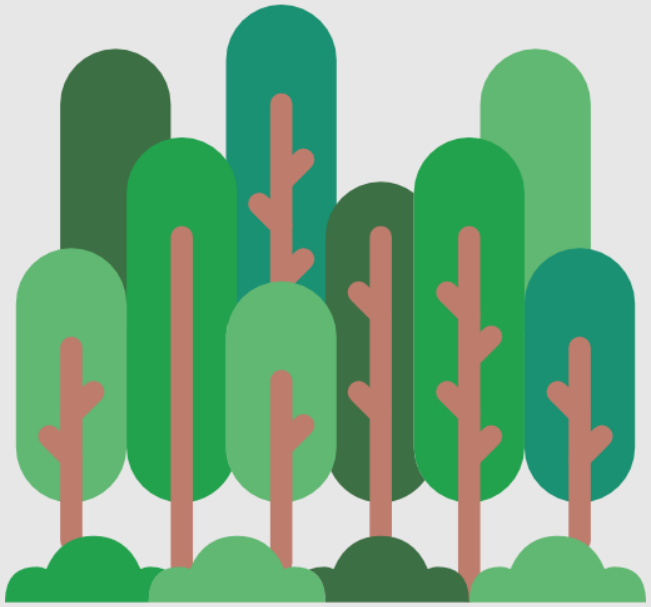


Random Forest



〈https://swalloow.github.io/bagging-boosting〉

Random Forest vs Extra Trees

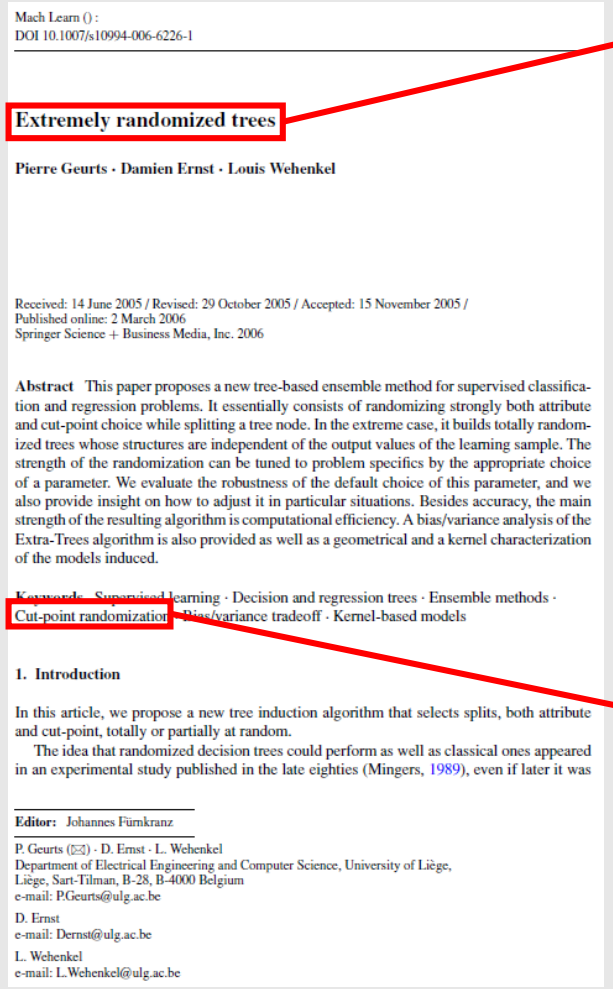


VS

Random Forest?

Extra Trees?

Extra Trees



Mach Learn () :
DOI 10.1007/s10994-006-6226-1

Extremely randomized trees

Pierre Geurts · Damien Ernst · Louis Wehenkel

Received: 14 June 2005 / Revised: 29 October 2005 / Accepted: 15 November 2005 /
Published online: 2 March 2006
Springer Science + Business Media, Inc. 2006

Abstract This paper proposes a new tree-based ensemble method for supervised classification and regression problems. It essentially consists of randomizing strongly both attribute and cut-point choice while splitting a tree node. In the extreme case, it builds totally randomized trees whose structures are independent of the output values of the learning sample. The strength of the randomization can be tuned to problem specifics by the appropriate choice of a parameter. We evaluate the robustness of the default choice of this parameter, and we also provide insight on how to adjust it in particular situations. Besides accuracy, the main strength of the resulting algorithm is computational efficiency. A bias/variance analysis of the Extra-Trees algorithm is also provided as well as a geometrical and a kernel characterization of the models induced.

Keywords Supervised learning · Decision and regression trees · Ensemble methods · Cut-point randomization · Bias/variance tradeoff · Kernel-based models

1. Introduction

In this article, we propose a new tree induction algorithm that selects splits, both attribute and cut-point, totally or partially at random.

The idea that randomized decision trees could perform as well as classical ones appeared in an experimental study published in the late eighties (Mingers, 1989), even if later it was

Editor: Johannes Fürnkranz

P. Geurts (✉) · D. Ernst · L. Wehenkel
Department of Electrical Engineering and Computer Science, University of Liège,
Liège, Sart-Tilman, B-28, B-4000 Belgium
e-mail: P.Geurts@ulg.ac.be

D. Ernst
e-mail: Dernst@ulg.ac.be

L. Wehenkel
e-mail: L.Wehenkel@ulg.ac.be

〈Pierre Geurts et al. (2006)〉

Extremely Randomized Trees

Random Forest Paper:
Leo Breiman et al. **2001**
Extra Trees Paper:
Pierre Geurts et al. **2006**

Keywords: Cut-point Randomization
➔극단적인 랜덤 분할
➔Bias를 낮춰줌

Boosting

Boosting

가령 경마의 초보자가 경마의 승률을 높이기 위해 경마의 고수들을 찾아가서 조언을 듣는 경우를 가정해보자. 초보자가 경마의 고수들에게 어떤 경주마를 선택해야 할 것인지 질문을 하면, 고수들도 언제든 통할 수 있는 경주마를 고를 수 있는 golden rule은 별로 없다고 할 것이다. (혹시 있더라도 말을 해주지 않을지도 모르겠지만…)

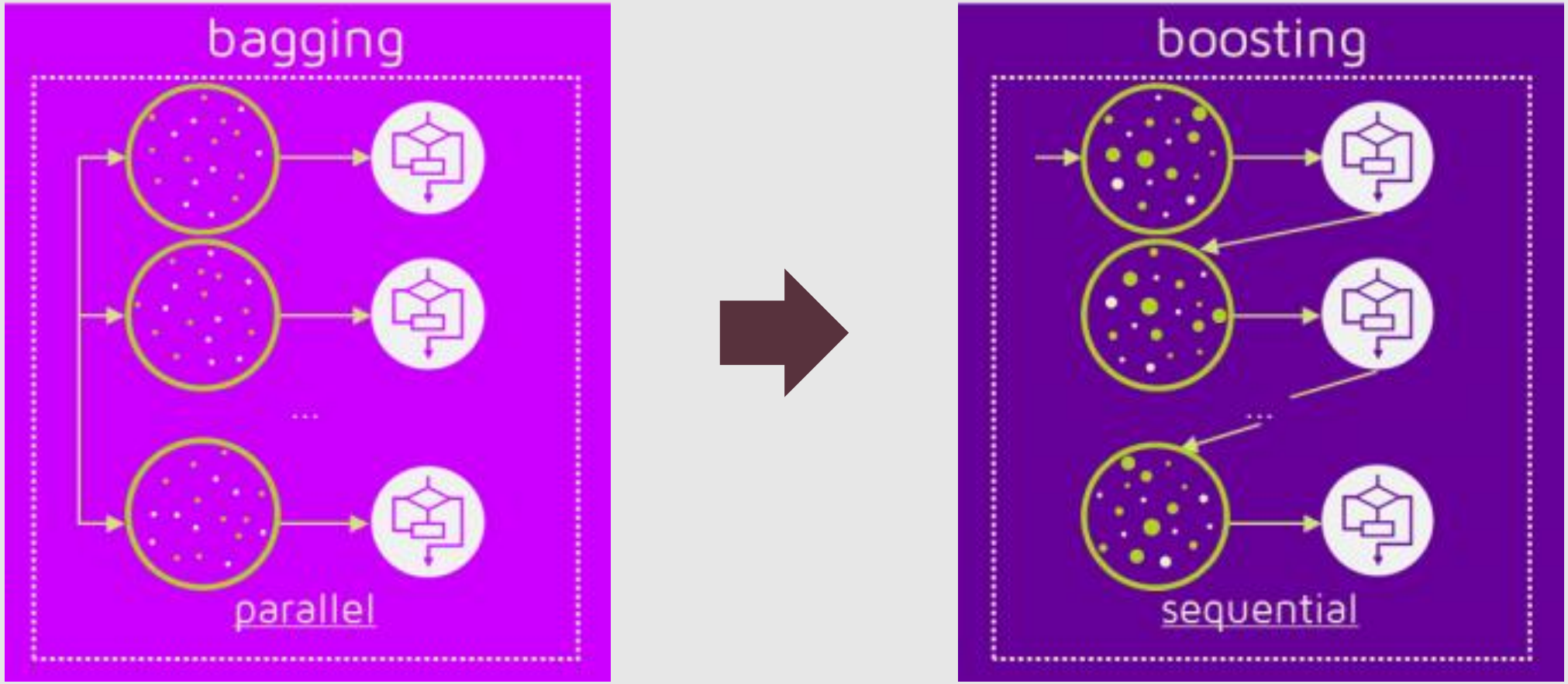
어느 경우든 적용이 가능한 golden rule을 말하는 것은 어렵겠지만, 팁을 한가지씩 알려달라고 하면 나름대로의 규칙(weak rule)을 말해줄 것이다. 가령, "최근 경주에서 우승을 많이 한 말을 골라라", "승률의 가능성이 높은 말을 골라라", "랩타임(lap time)이 가장 짧은 말을 골라라" 등등의 조언을 듣게 될 것이다. 여러 전문가로부터 들은 조언에 따라 말을 선택하면 다양한 경우가 발생하겠지만, 경우의 수가 겹치는 쪽을 택하여 경주마 선택의 범위를 좁히면, 승률이 올라갈 것이다.

여러 전문가로부터 조언을 들었고 의견도 분분하다면, 자신만의 경주마 선정 규칙을 정해야 한다. 경기가 달라지면 적용되는 규칙도 달라져야 할 것이고, 특정 경기에서는 다양한 조언을 어떻게 통합해서 그것을 적용해야 할까?

이 때 쓸 수 있는 방법이 바로 "Boosting"이다.

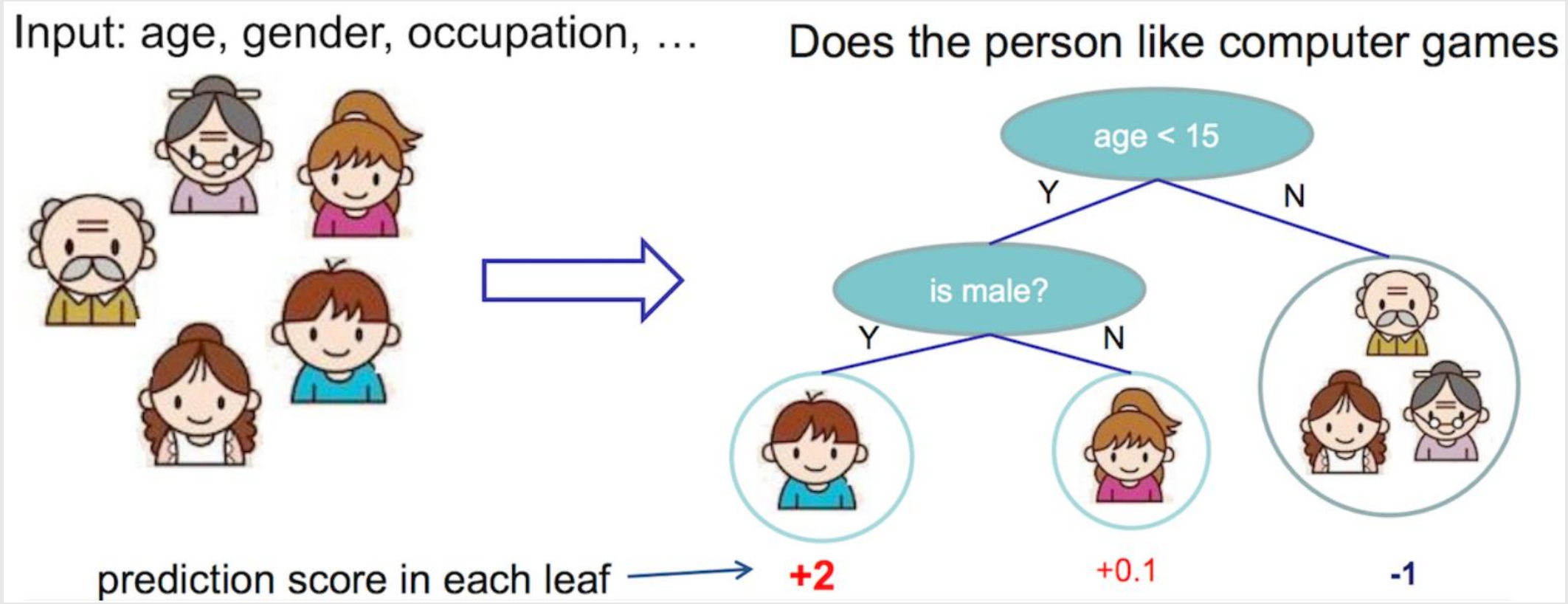
〈http://blog.naver.com/laonple/220834569716〉

Boosting

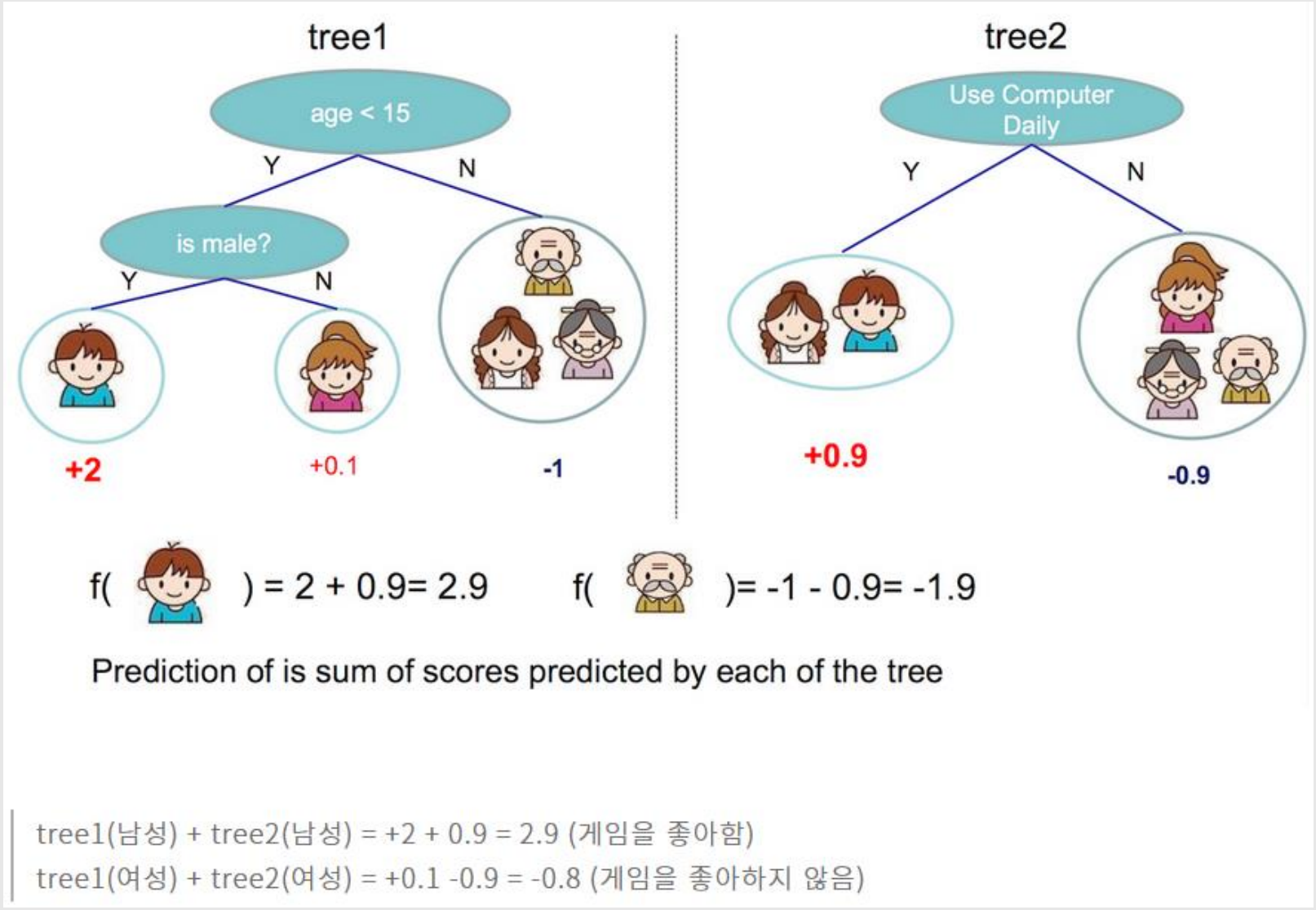


〈https://medium.com/greyatom/a-quick-guide-to-boosting-in-ml-acf7c1585cb5〉

Tree Structure Boosting



Tree Structure Boosting



tree1(남성) + tree2(남성) = +2 + 0.9 = 2.9 (게임을 좋아함)
tree1(여성) + tree2(여성) = +0.1 -0.9 = -0.8 (게임을 좋아하지 않음)

Adaboost

Given: $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in \mathcal{X}$, $y_i \in \{-1, +1\}$.

training sample

target label (no, yes)

Initialize: $D_1(i) = 1/m$ for $i = 1, \ldots, m$.

D : weight of training samples (the probability of being selected for training the component classifier)

For $t = 1, \ldots, T$:

T : iteration round

- Train weak learner using distribution D_t.
- Get weak hypothesis $h_t : \mathcal{X} \to \{-1, +1\}$.
- Aim: select h_t with low weighted error:

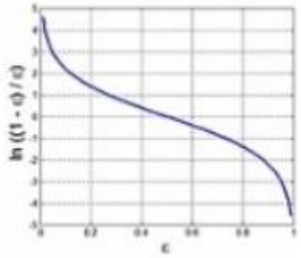
ε : error rate of weak classifier

$$\varepsilon_t = \Pr_{i \sim D_t}[h_t(x_i) \neq y_i] = \sum_{i : h_t(x_i) \neq y_i} D_t(i)$$

1 if misclassified, 0 if properly classified

- Choose $\alpha_t = \frac{1}{2} \ln\left(\frac{1 - \varepsilon_t}{\varepsilon_t}\right)$.

α : weight (importance) of weak classifier



- Update, for $i = 1, \ldots, m$:

update weights of training samples

$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

where Z_t is a normalization factor (chosen so that D_{t+1} will be a distribution).

$$\sum_i D_{t+1}(i) = 1$$

Output the final hypothesis:

H : strong classifier

$$H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right).$$

Adaboost

Algorithm 2 AdaBoost

Input: Give sample $(x_1, y_1), ..., (x_n, y_n)$ where $y \in \{-1, +1\}$

Output: G is the strong classifier which produce by this algorithm.

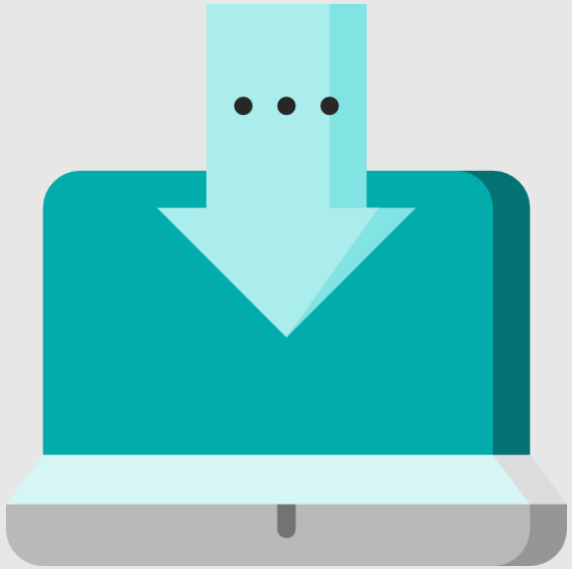
Steps of algorithm

- Initialize weights $w_{1,i} = \frac{1}{2m}, \frac{1}{2l}$ for $y_i = 0, 1$ respectively.
- For t = 1, ..., T
  1. normalize the weight $w_{t,i} = \frac{w_{t,i}}{\sum_{i=1}^{n} w_{t,i}}$
  2. select the best weak classifier with respect to the weighted error rate.
  3. Define $g_t(x) = g(x, f_t, p_t, \theta_t)$ where $f_t, p_t, and \theta_t$ are the minimizers of ε_t
  4. Update the weights
- The final strong classifier is:

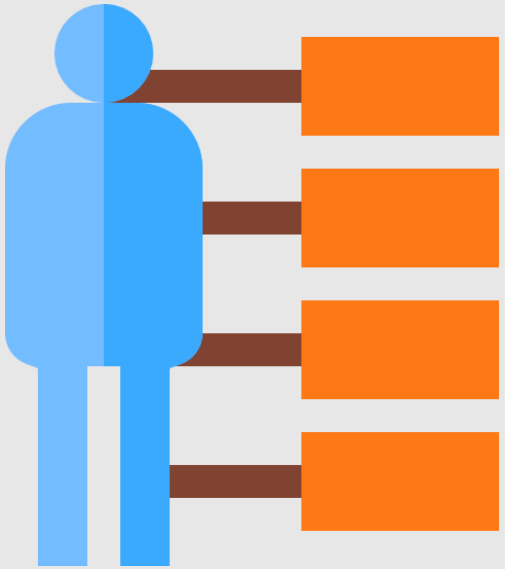
$$G(x) = \begin{cases} 1 \text{ if } \sum_{t=1}^{T} \alpha_t g_t(x) >= \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 \text{ if } otherwise \end{cases} \quad (7)$$

〈https://steemit.com/kr-dev/@steemonen1/adaboost〉

Adaboost



Input



Feature

Base Classifier이자
Week Classifier의 후보

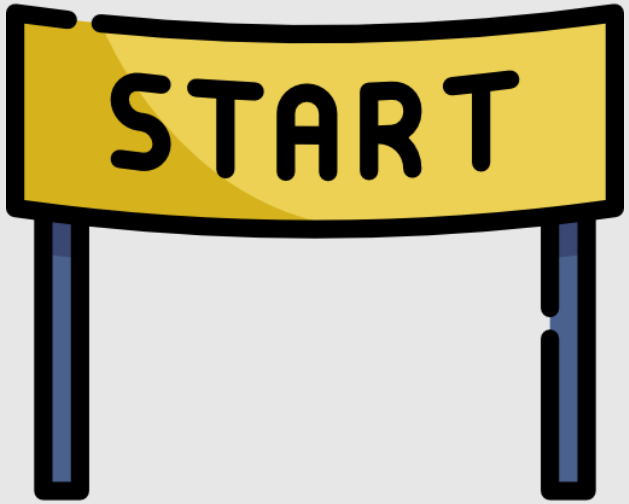
Iteration

Week Classifier의 개수

Adaboost

초기화 (Initialization)

: 모든 Training Sample들의 Weight(가중치, 중요도)를 동일하게 초기화



Adaboost

아래 과정을 **T회 반복**한다 (t = 1 ,2, ... , T) :

① **모든 feature에 대해** training sample들을 얼마나 잘 분류하는지 **성능 평가**한다.
어떻게? **각 feature마다 weighted error를 계산**한다.

▶ positive를 negative로 분류하거나, negative를 positive로 분류 → 잘못된 분류. error.
얼마나 많은 training sample들을 각자의 class에 맞게 제대로 분류하지 못했는지?

▶ weighted error = sample들의 weight를 고려하여 각 feature의 error를 구한다.

② **분류 성능이 가장 좋은 하나의 feature**를 해당 round t 에서의 **weak classifier**로 한다.

▶ 분류 성능이 가장 좋은 = weighted error가 가장 작은.

③ 해당 **weak classifier의 weight(가중치, 중요도)**를 구한다.

▶ weighted error를 이용하여 계산한다.

- 에러가 크면 중요도는 작아지고, 에러가 작으면 중요도는 커지도록 한다.
= 성능이 좋은 weak classifier의 판단을 더 신뢰하여 점수를 더 쳐준다..

④ **training sample들의 weight를 업데이트**한다.

▶ 잘못 분류된 sample의 weight는 증가, 잘 분류된 sample의 weight는 감소시킨다.

오분류된 sample의 weight이 커지는 건 해당 sample의 중요도가 커지는 것과 같다.
다음 iteration에서 해당 sample을 잘못 분류할 때 weighted error가 커지게 되고,
weighted error가 큰 feature는 weak classifier가 될 수 없다.
→ 이전 단계에서 잘못 분류된 sample을 제대로 분류할 수 있는 feature가
다음 단계에서 weak classifier로 될 수 있다.

⑤ t = t+1 (① 번으로 돌아가 **알고리즘을 반복**한다.)

[정리] 매 iteration round마다 분류 성능이 가장 좋은 하나의 feature를 선택하여
해당 iteration round에서의 weak classifier로 한다.
(T번 반복 → T개의 weak classifiers)

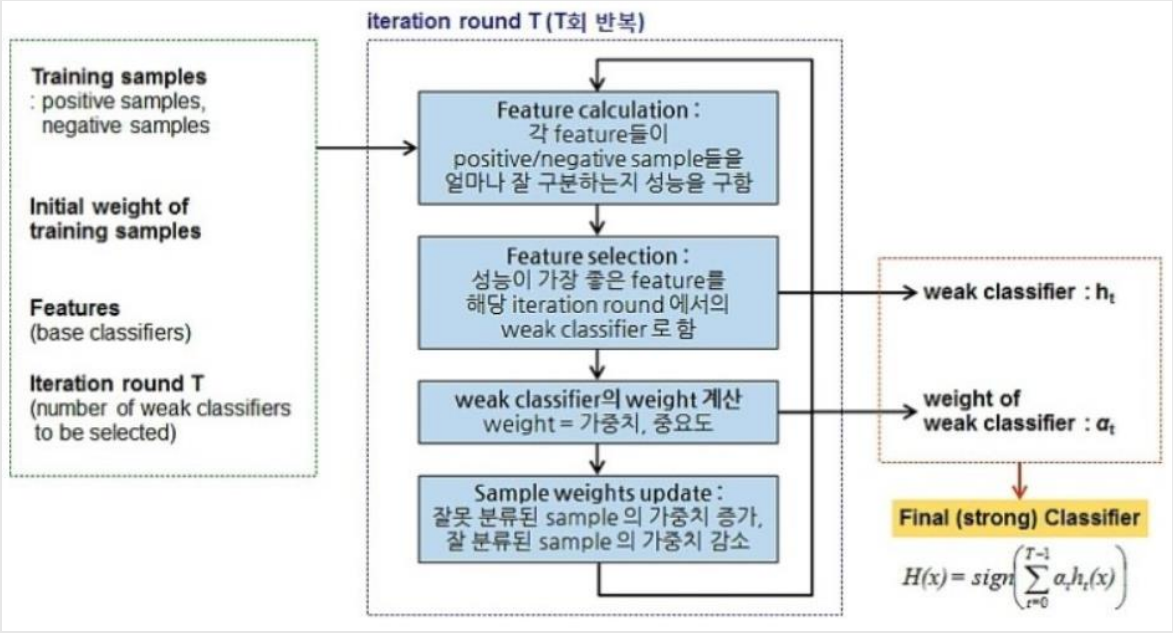
〈http://blog.naver.com/PostView.nhn?blogId=dic1224&logNo=220989033563〉

Adaboost

T개의 Weak classifier를 Weighted Linear Combination하여 최종 Strong classifier를 얻는다.

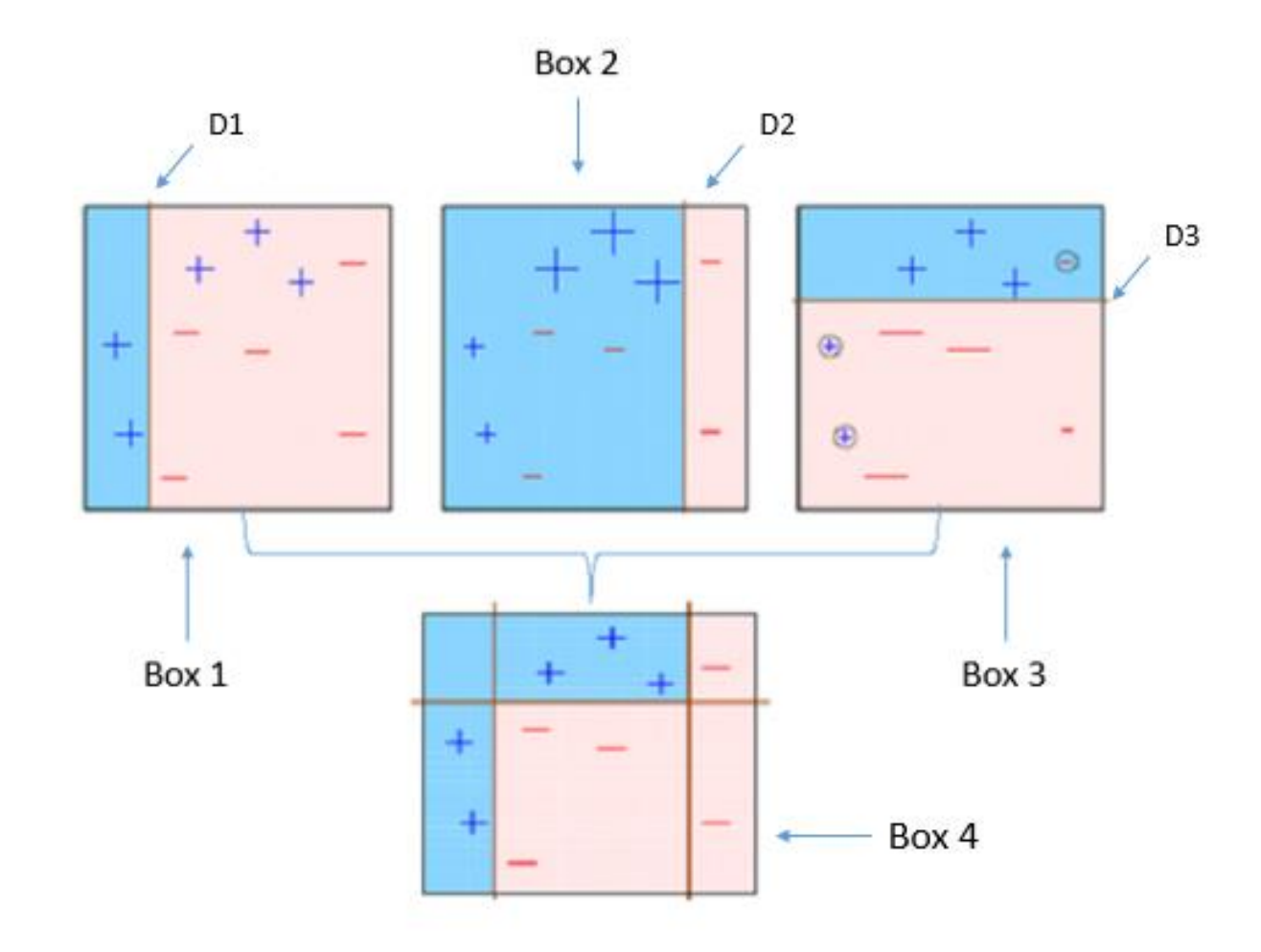
$$H(x) = \alpha_1 h_1(x) + \alpha_2 h_2(x) + \cdots + \alpha_T h_T(x) = \sum_{t=1}^{T} \alpha_t h_t(x)$$

H: Final strong classifier
h = Weak classifier
α = Weight of weak classifier
T = Iteration round



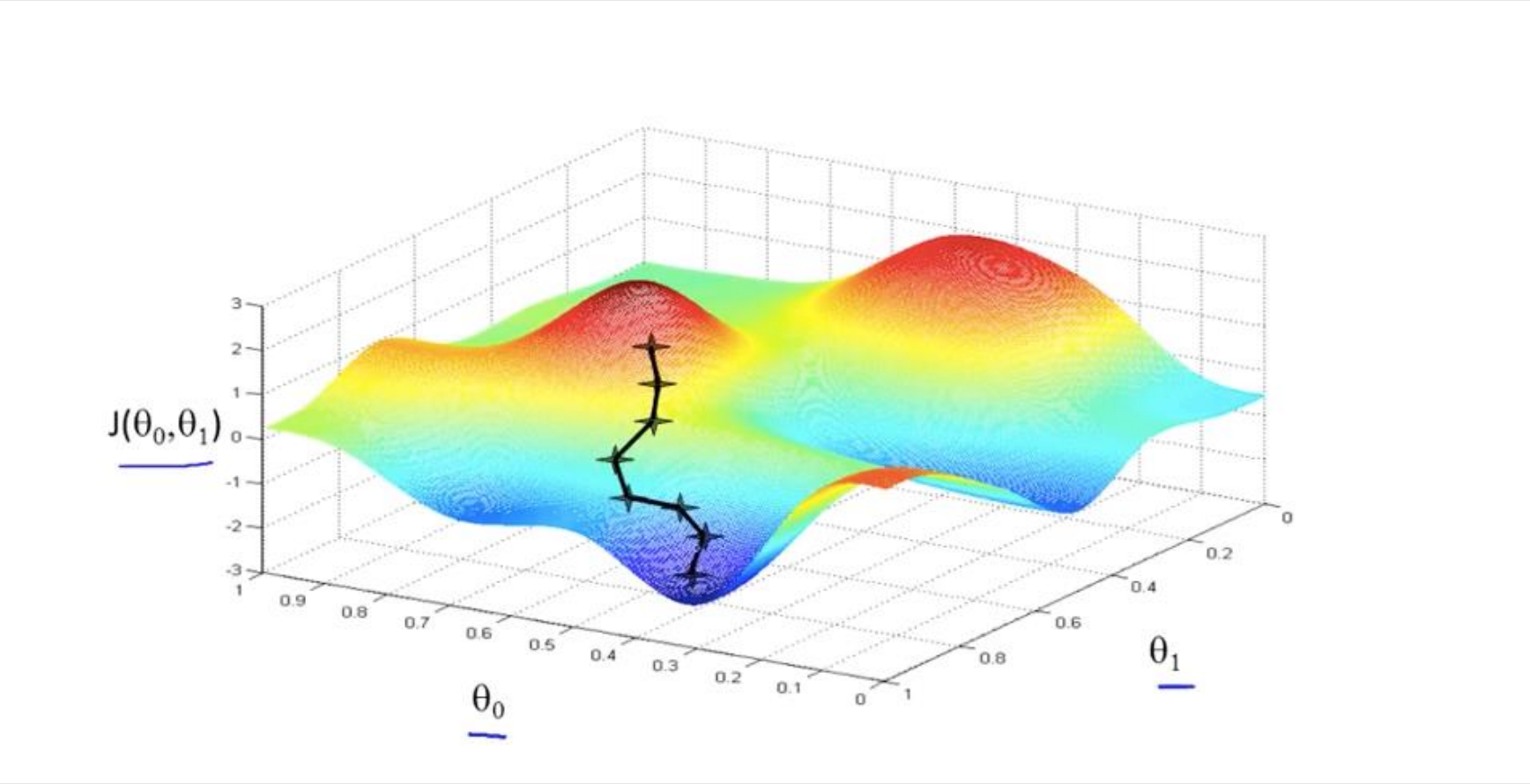
〈http://blog.naver.com/PostView.nhn?blogId=dic1224&logNo=220989033563〉

Adaboost



〈https://medium.com/greyatom/a-quick-guide-to-boosting-in-ml-acf7c1585cb5〉

Gradient Boosting Visualization



〈https://stats.stackexchange.com/questions/241715/gradient-descent-and-cost-function-trouble〉

Adaboost vs Gradient Boosting

$$err_m = \sum_{Y_i \neq T_m(x_i)} W_i$$

$$\alpha_m = \beta \cdot \ln \frac{1 - err_m}{err_m} \qquad (\beta\text{: constant})$$

$$W_i \rightarrow W_i \cdot e^{\alpha_m}$$

$$\theta_i := \theta_i - \rho \frac{\partial J}{\partial \theta_i}$$

Gradient Boosting

정리하면 Gradient Boosting에서는 Gradient가 현재까지 학습된 모델의 약점을 드러내는 역할을 하고, 다른 모델이 그걸 중점적으로 보완해서 성능을 Boosting한다. 위에서는 L2 손실함수를 썼지만 미분만 가능하다면 다른 손실함수도 얼마든지 쓸 수 있다는 것이 장점이다. 부스팅 알고리즘의 특성상 계속 약점(오분류/잔차)을 보완하려고 하기 때문에 잘못된 레이블링이나 아웃라이어에 필요 이상으로 민감할 수 있다. 이런 문제에 강인한 L1 Loss나 Huber Loss를 쓰고자 한다면 그냥 손실함수만 교체하면 된다. 손실함수의 특성은 Gradient를 통해 자연스럽게 학습에 반영된다.

〈http://4four.us/article/2017/05/gradient-boosting-simply〉

Gradient Boosting

* Gradient Boost 개선 - subsampling

Gradient Boost 알고리즘의 발표 후에 작은 개선 사항이 있었다.
하나의 트리를 학습할 때 모든 학습 데이터를 사용하는 것이 아니라, 학습 데이터 중에서 랜덤하게 추출한 일부 데이터만 사용하는 것이다. 이것은 overfitting을 감소시켜서 성능을 향상시킨다. 대략 이 수치는 0.5 에서 0.8 사이의 값을 사용하고, 일반적으로 기본 값인 0.5를 사용한다.

* Gradient Boost 개선 - min child weight

또다른 성능 향상 방법은 하나의 트리에서 말단 노드가 가져야 할 데이터수(observations)의 최소값을 제한하는 것이다. 이 수보다 적은 수를 가지는 말단 노드가 생성되는 분할은 무시가 된다. 이렇게 함으로써 특이값 때문에 발생하는 변동폭을 줄여서 성능을 향상시킨다.

〈http://jetzt.tistory.com/1041〉

XGBoost

dmlc

XGBoost

XGBoost

〈https://brunch.co.kr/@snobberys/137〉

XGBoost

- Model: assuming we have K trees

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i), \quad f_k \in \mathcal{F}$$

- Objective

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

Training loss

Complexity of the Trees

XGBoost

- The prediction at round t is $\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + f_t(x_i)$

This is what we need to decide in round t

$$
\begin{aligned}
Obj^{(t)} &= \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t)}) + \sum_{i=1}^{t} \Omega(f_i) \\
&= \sum_{i=1}^{n} l\left(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)\right) + \Omega(f_t) + constant
\end{aligned}
$$

Goal: find f_t to minimize this

- Consider square loss

$$
\begin{aligned}
Obj^{(t)} &= \sum_{i=1}^{n} \left(y_i - (\hat{y}_i^{(t-1)} + f_t(x_i))\right)^2 + \Omega(f_t) + const \\
&= \sum_{i=1}^{n} \left[2(\hat{y}_i^{(t-1)} - y_i) f_t(x_i) + f_t(x_i)^2\right] + \Omega(f_t) + const
\end{aligned}
$$

This is usually called residual from previous round

XGBoost

미적분학에서,
테일러 급수는 도함수들의 한 점에서의 값으로 계산된 항의 무한합으로 해석함수를 나타내는 방법

$$
\begin{aligned}
&T_f(x_1, \ldots, x_d) \\
&= \sum_{n_1=0}^{\infty} \sum_{n_2=0}^{\infty} \cdots \sum_{n_d=0}^{\infty} \frac{(x_1 - a_1)^{n_1} \cdots (x_d - a_d)^{n_d}}{n_1! \cdots n_d!} \left(\frac{\partial^{n_1 + \cdots + n_d} f}{\partial x_1^{n_1} \cdots \partial x_d^{n_d}} \right) (a_1, \ldots, a_d) \\
&= f(a_1, \ldots, a_d) + \sum_{j=1}^{d} \frac{\partial f(a_1, \ldots, a_d)}{\partial x_j} (x_j - a_j) \\
&\quad + \frac{1}{2!} \sum_{j=1}^{d} \sum_{k=1}^{d} \frac{\partial^2 f(a_1, \ldots, a_d)}{\partial x_j \partial x_k} (x_j - a_j)(x_k - a_k) \\
&\quad + \frac{1}{3!} \sum_{j=1}^{d} \sum_{k=1}^{d} \sum_{l=1}^{d} \frac{\partial^3 f(a_1, \ldots, a_d)}{\partial x_j \partial x_k \partial x_l} (x_j - a_j)(x_k - a_k)(x_l - a_l) + \cdots
\end{aligned}
$$

〈https://ko.wikipedia.org/wiki/%ED%85%8C%EC%9D%BC%EB%9F%AC_%EA%B8%89%EC%88%98〉

XGBoost

- Goal $Obj^{(t)} = \sum_{i=1}^{n} l\left(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)\right) + \Omega(f_t) + constant$
  - Seems still complicated except for the case of square loss
- Take Taylor expansion of the objective
  - Recall $f(x + \Delta x) \simeq f(x) + f'(x)\Delta x + \frac{1}{2} f''(x)\Delta x^2$
  - Define $g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)}), \quad h_i = \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$

$$Obj^{(t)} \simeq \sum_{i=1}^{n} \left[l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t) + constant$$

XGBoost

$$\sum_{i=1}^{n} \left[g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t)$$

- where $g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)}), \quad h_i = \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$

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

Number of leaves **L2 norm of leaf scores**

- Define the instance set in leaf j as $I_j = \{i | q(x_i) = j\}$
- Regroup the objective by each leaf

$$\begin{aligned} Obj^{(t)} &\simeq \sum_{i=1}^{n} \left[g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t) \\ &= \sum_{i=1}^{n} \left[g_i w_{q(x_i)} + \frac{1}{2} h_i w^2_{q(x_i)} \right] + \gamma T + \lambda \frac{1}{2} \sum_{j=1}^{T} w_j^2 \\ &= \sum_{j=1}^{T} \left[(\sum_{i \in I_j} g_i) w_j + \frac{1}{2} (\sum_{i \in I_j} h_i + \lambda) w_j^2 \right] + \gamma T \end{aligned}$$

- This is sum of T independent quadratic functions

XGBoost

$$argmin_x \; Gx + \tfrac{1}{2}Hx^2 = -\tfrac{G}{H}, \; H > 0 \qquad \min_x \; Gx + \tfrac{1}{2}Hx^2 = -\tfrac{1}{2}\tfrac{G^2}{H}$$

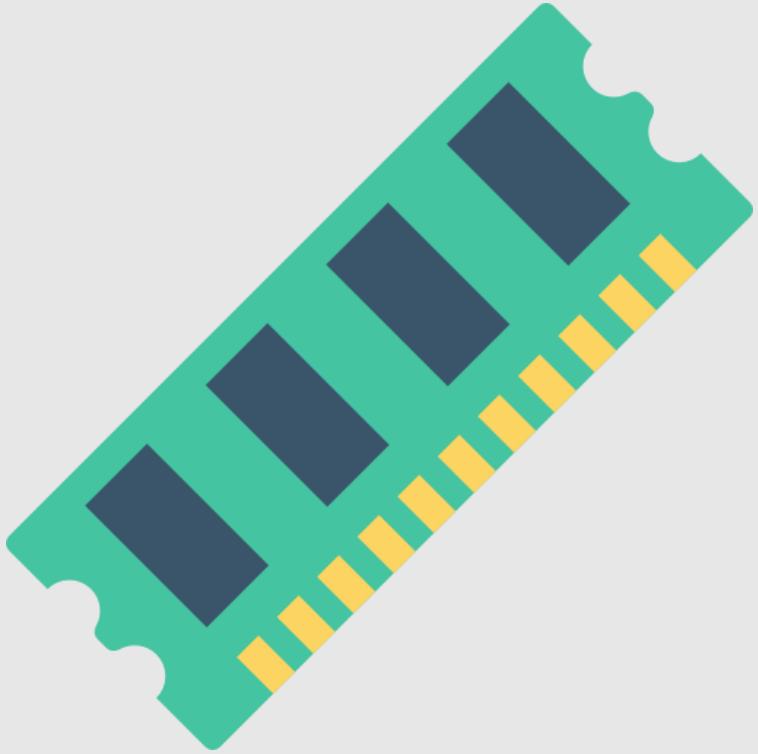
$$w_j^* = -\frac{G_j}{H_j+\lambda} \qquad Obj = -\tfrac{1}{2}\sum_{j=1}^{T} \frac{G_j^2}{H_j+\lambda} + \gamma T$$

This measures how good a tree structure is!

$$Gain = \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{(G_L+G_R)^2}{H_L+H_R+\lambda} - \gamma$$

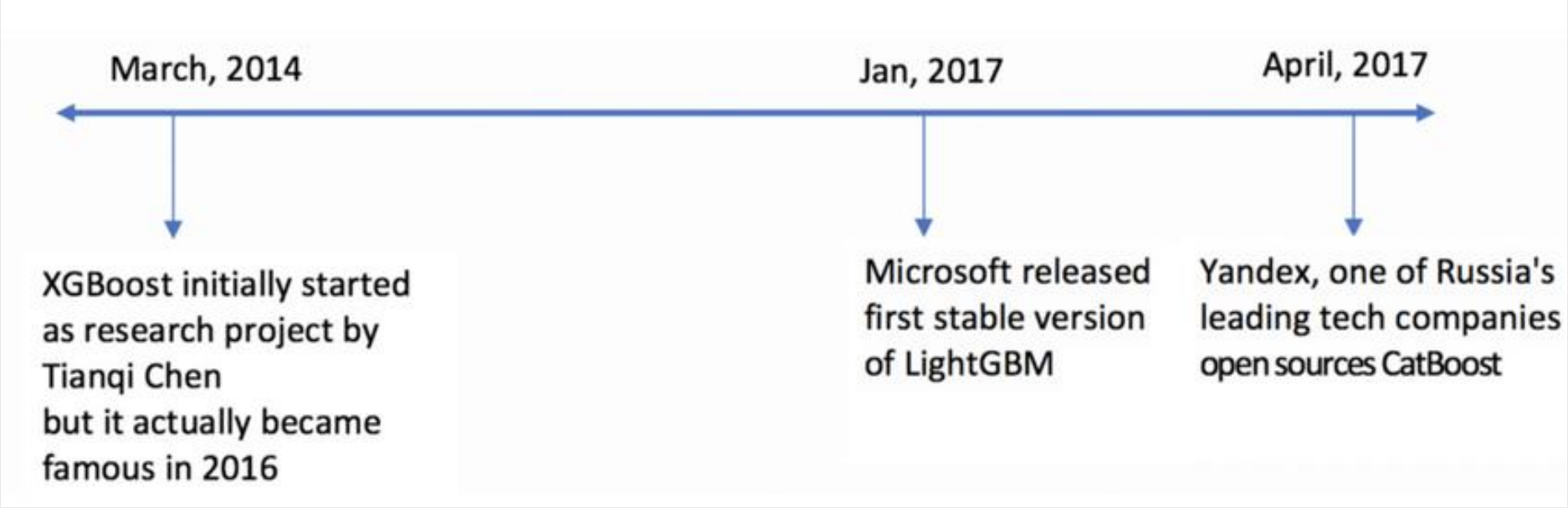
XGBoost

Regularization

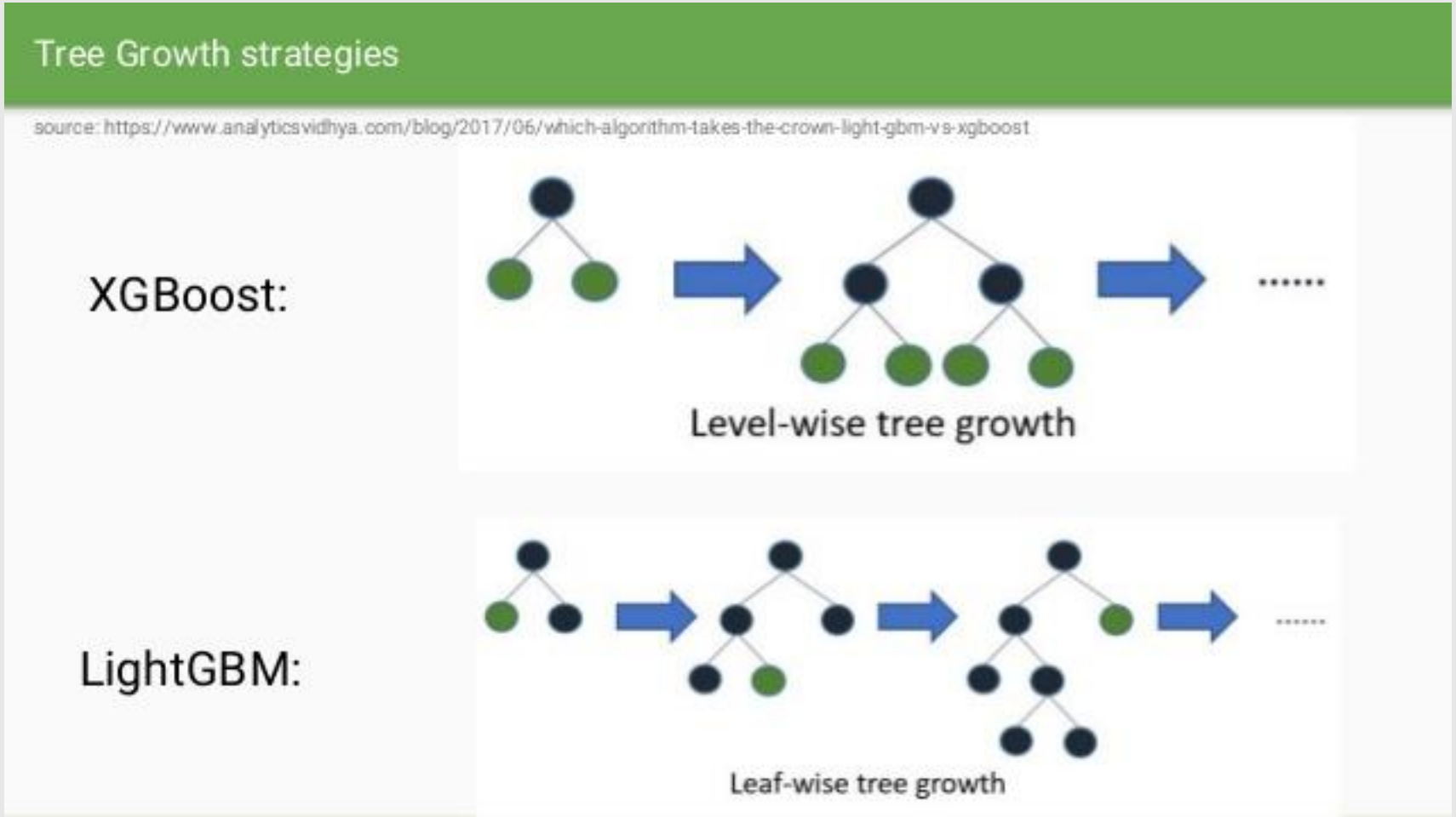


Parallel Computing

XGBoost → LGBM → CatBoost

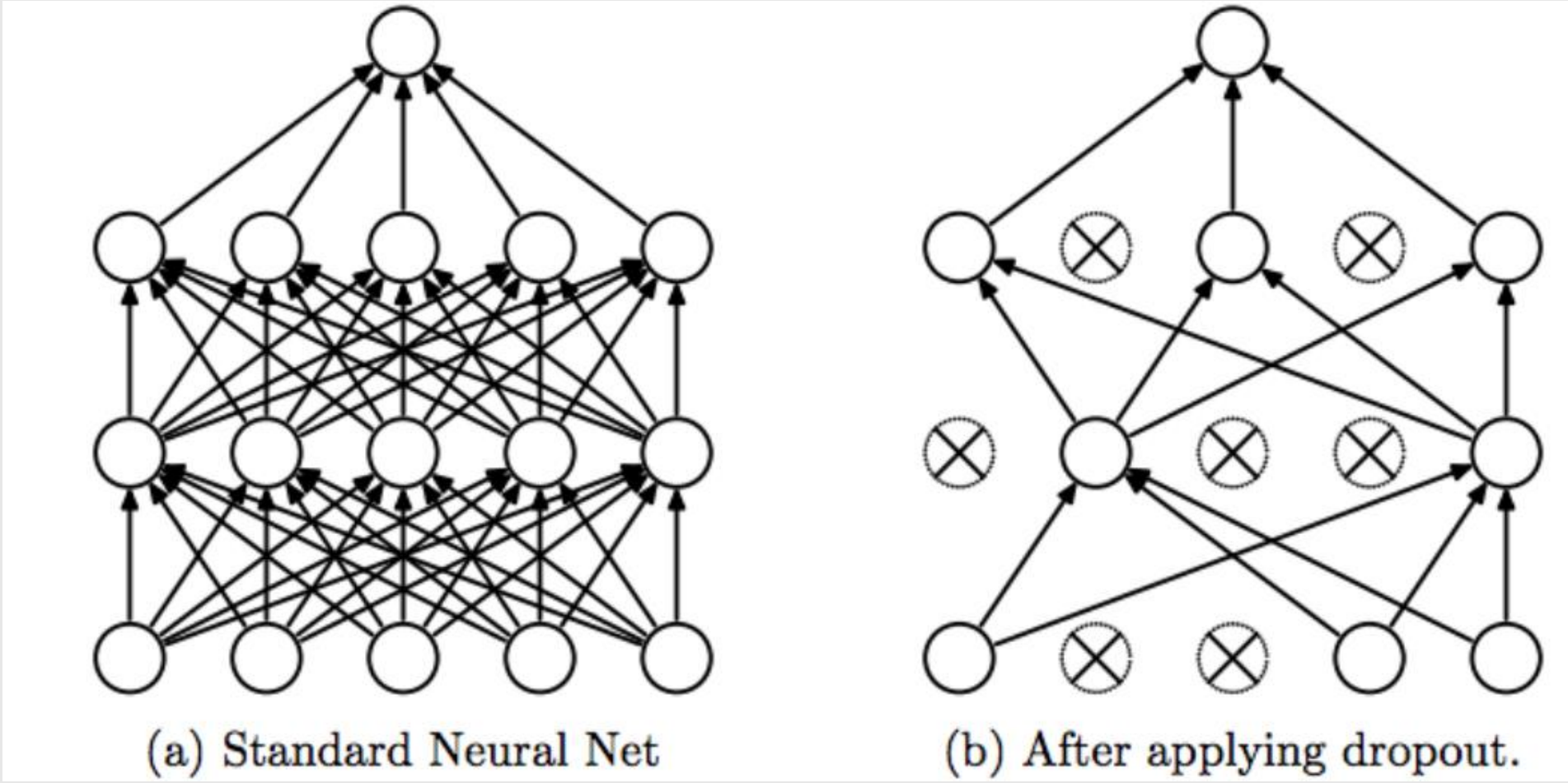


LightGBM



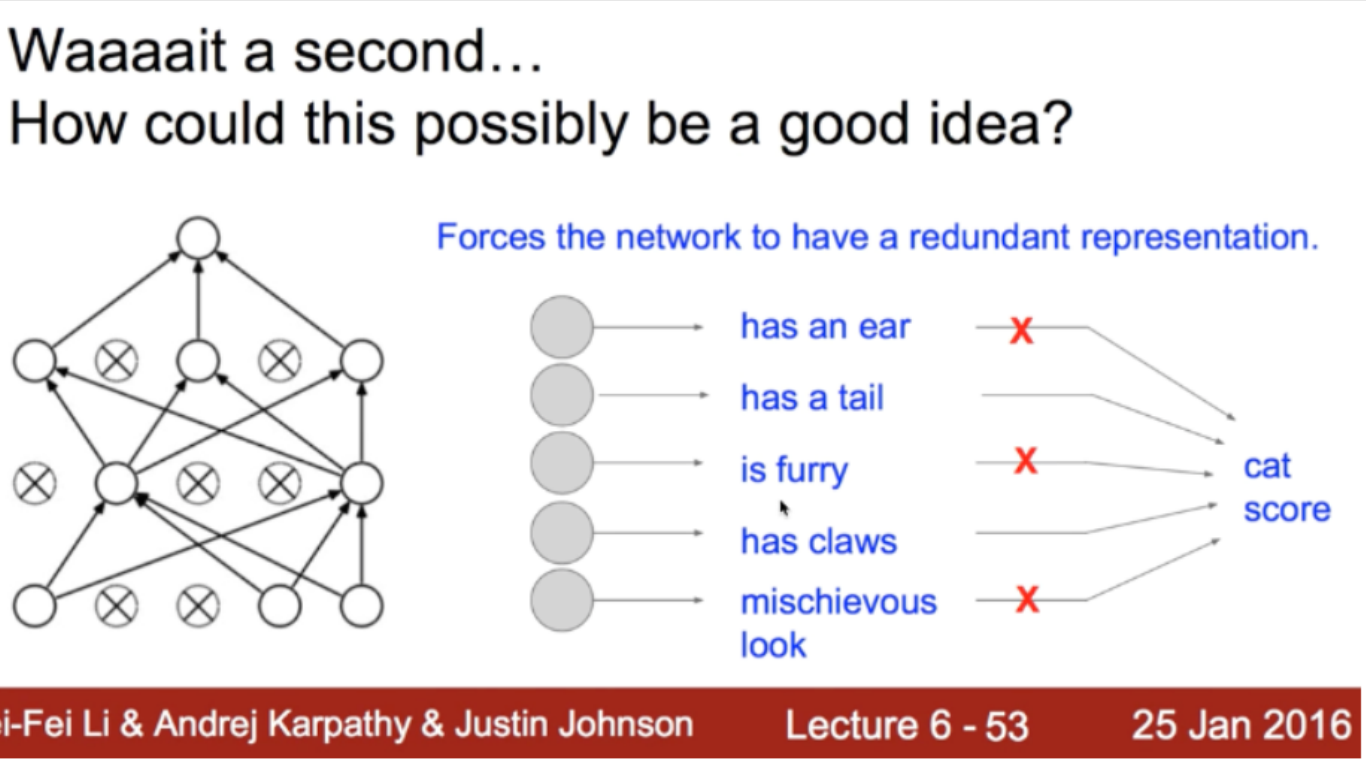
〈https://www.slideshare.net/GabrielCyprianoSaca/xgboost-lightgbm〉

LightGBM



〈http://pythonkim.tistory.com/42〉

LightGBM



제목을 보면.. 글쓴 이도 놀라고 있다. 어떻게 이게 좋은 생각일 수 있을까?

전문가들이 너무 많다고 가정하자. 귀만 판단하는 전문가, 꼬리만 판단하는 전문가 등등 너무 많은 weight이 있다면, 이들 중 일부만 사용해도 충분히 결과를 낼 수 있다. 오히려 이들 중에서 충분할 만큼의 전문가만 선출해서 반복적으로 결과를 낸다면, 오히려 균형 잡힌 훌륭한 결과가 나올 수도 있다.

한국 속담에 딱 맞는 말이 있다. "사공이 많으면 배가 산으로 간다". 머신러닝을 공부하는 과정에서 많이 듣게 되는 용어가 균형(balance)일 수 있다. 여러 가지 방법 또는 시도를 통한 균형을 잡을 때, 좋은 성능이 난다고 알려져 있다.

〈http://pythonkim.tistory.com/42〉

LightGBM

The good

- Often yields good results
- Reduced need for feature engineering
- Fast to train a single model
- Good choice if all you have is 1 shot at the problem
- GPU support
- Scikit-learn API
- Great to ensemble and optimize for multiple metrics

The bad

- Too many parameters
- Slow to tune parameters
- GPU config can be tough (try Docker)
- No GPU support on scikit-learn API (XGBoost)

〈https://www.slideshare.net/GabrielCyprianoSaca/xgboost-lightgbm〉

XGBoost & LightGBM

3. Boosting

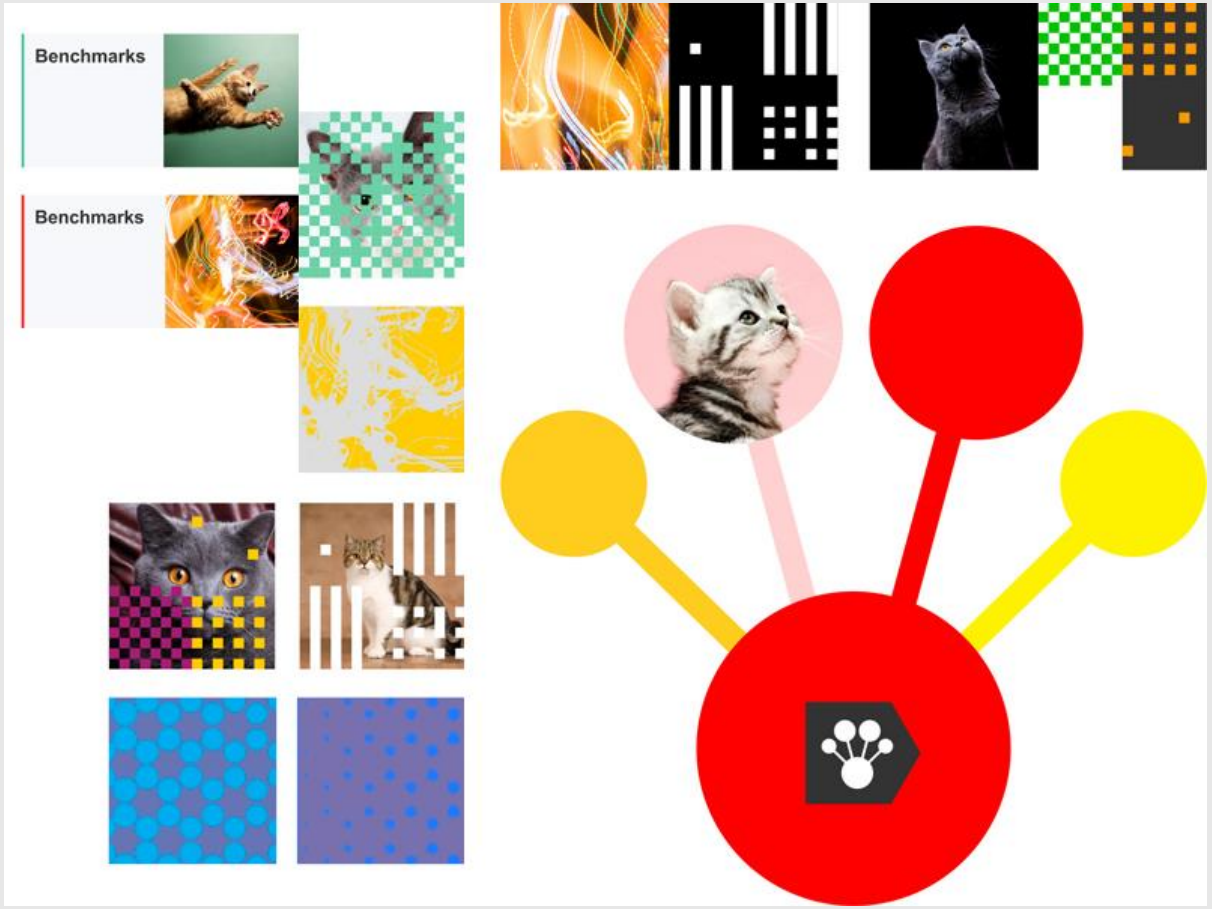
Boosting 알고리즘

알고리즘	특징	비고
AdaBoost	• 다수결을 통한 정답 분류 및 오답에 가중치 부여	
GBM	• Loss Function의 gradient를 통해 오답에 가중치 부여	gradient_boosting.pdf
Xgboost	• GBM 대비 성능향상 • 시스템 자원 효율적 활용 (CPU, Mem) • Kaggle을 통한 성능 검증 (많은 상위 랭커가 사용)	2014년 공개 boosting-algorithm-xgboost
Light GBM	• Xgboost 대비 성능향상 및 자원소모 최소화 • Xgboost가 처리하지 못하는 대용량 데이터 학습 가능 • Approximates the split (근사치의 분할)을 통한 성능 향상	2016년 공개 light-gbm-vs-xgboost

〈https://www.slideshare.net/freepsw/boosting-bagging-vs-boosting〉

Catboost

Yandex
CatBoost



<https://dribbble.com/shots/3798213-Yandex-CatBoost>

Catboost

CatBoost: unbiased boosting with categorical features

Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, Andrey Gulin
Yandex, Moscow, Russia
{ostroumova-la, gleb57, alvor88, annaveronika, gulin}@yandex-team.ru

Abstract

This paper presents the key algorithmic techniques behind CatBoost, a new gradient boosting toolkit. Their combination leads to CatBoost outperforming other publicly available boosting implementations in terms of quality on a variety of datasets. Two critical algorithmic advances introduced in CatBoost are the implementation of *ordered* boosting, a permutation-driven alternative to the classic algorithm, and an innovative algorithm for processing categorical features. Both techniques were created to fight a *prediction shift* caused by a special kind of target leakage present in all currently existing implementations of gradient boosting algorithms. In this paper, we provide a detailed analysis of this problem and demonstrate that proposed algorithms solve it effectively, leading to excellent empirical results.

1 Introduction

Gradient boosting is a powerful machine-learning technique that achieves state-of-the-art results in a variety of practical tasks. For many years, it has remained the primary method for learning problems with heterogeneous features, noisy data, and complex dependencies: web search, recommendation systems, weather forecasting, and many others [5, 25, 27, 30]. Gradient boosting is essentially a process of constructing an ensemble predictor by performing gradient descent in a functional space. It is backed by solid theoretical results that explain how strong predictors can be built by iteratively combining weaker models (*base predictors*) in a greedy manner [17].

We show in this paper that all existing implementations of gradient boosting face the following statistical issue. A prediction model F obtained after several steps of boosting relies on the targets of all training examples. We demonstrate that this actually leads to a shift of the distribution of $F(\mathbf{x}_k) \mid \mathbf{x}_k$ for a training example $\mathbf{x}_k$ from the distribution of $F(\mathbf{x}) \mid \mathbf{x}$ for a test example $\mathbf{x}$. This finally leads to a *prediction shift* of the learned model. We identify this problem as a special kind of target leakage in Section 4. There is a similar issue in standard algorithms of preprocessing categorical features. One of the most effective ways [6, 24] to use them in gradient boosting is converting categories to their target statistics. A target statistic is a simple statistical model itself, and it can also cause target leakage and a prediction shift. We analyze this in Section 3.

In this paper, we propose an *ordering principle* to solve both problems. Relying on it, we derive *ordered boosting*, a modification of standard gradient boosting algorithm, which avoids target leakage (Section 4), and a new algorithm for processing categorical features (Section 5). Their combination is implemented as an open-source library[1] called CatBoost (for "Categorical Boosting"), which outperforms the existing state-of-the-art implementations of gradient boosted decision trees — XGBoost [8] and LightGBM [16] — on a diverse set of popular machine learning tasks (see Section 8).

[1] https://github.com/catboost/catboost

Preprint. Work in progress.

Liudmila Prokhorenkova et al. 2017

- Ensemble모형은 Bias가 너무 크다.
- 같은 데이터를 활용해서 데이터에 의존성을 야기한다.

➔ **잔차 최소화를 위해 Dynamic Boosting을 쓰자**

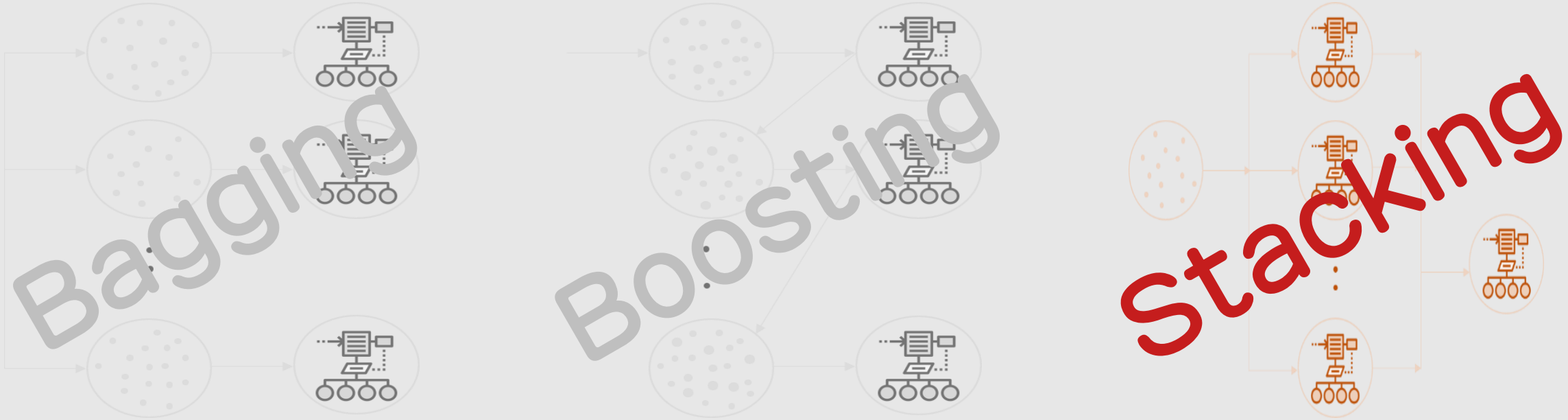
- Dynamic Boosting
  - LOO(Leave-one-out)을 생각하면 편하다.
  - A데이터로 가중치 계산 후
    B데이터로 다음 연산을 진행하는 것
  - 이를 통한 unbiased residual을 추정하고
    그걸로 학습하는 것.

➔ **바이어스 잔차가 일으키는 과적합을 해결하기 위해 굉장히 복잡도가 높은 모델을 사용해야 하는데 이 복잡도를 해결한 사례**

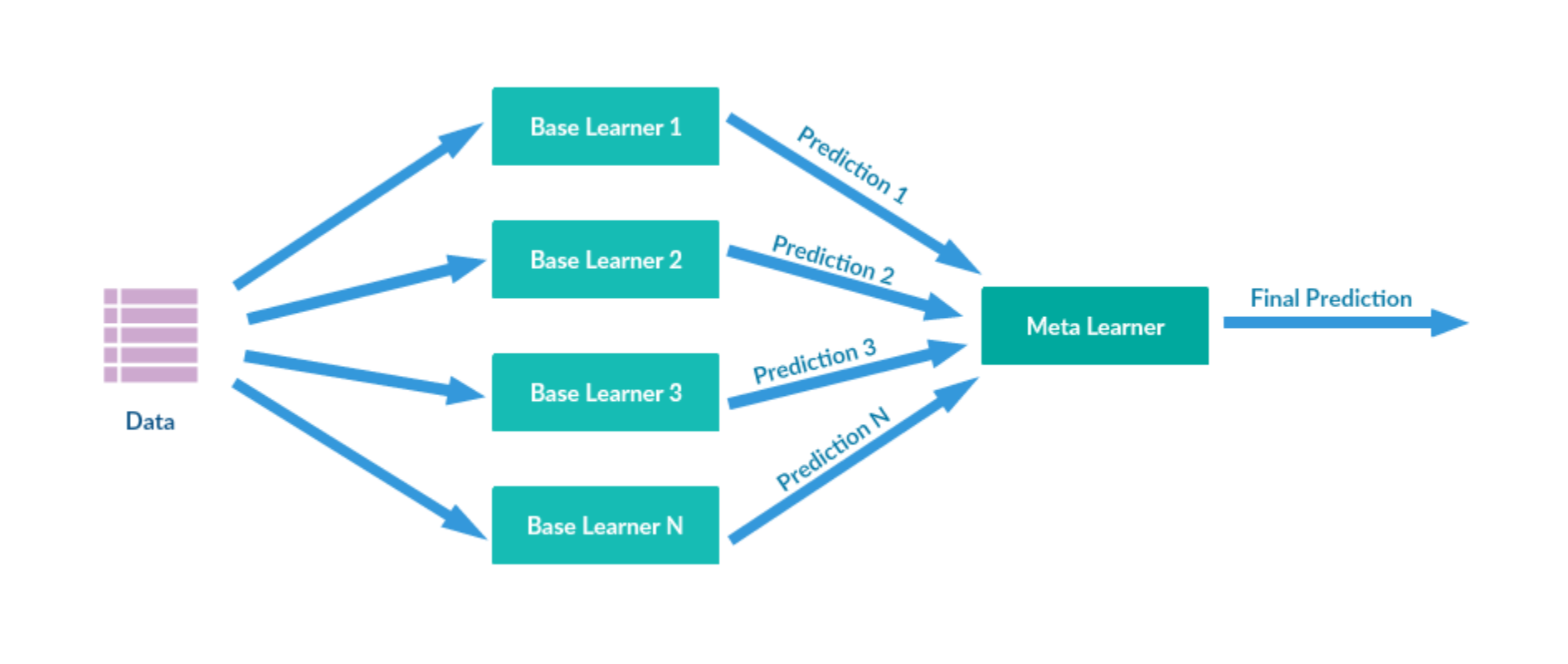
제대로 잘 이해못했을 수도 있고
아예 잘못 이해했을 수도 있다.

<http://ishuca.tistory.com/418>

Stacking



Stacking



〈http://supunsetunga.blogspot.com/2016/06/stacking-in-machine-learning.html〉

Stacking

Combining One-Class Classifiers Using Meta Learning

Eitan Menahem
Deutsche Telekom Laboratories
Department of Information Engineering
Ben-Gurion University of the Negev
Be'er Sheva, 84105, Israel
eitanme@post.bgu.ac.il

Lior Rokach
Deutsche Telekom Laboratories
Department of Information Engineering
Ben-Gurion University of the Negev
Be'er Sheva, 84105, Israel
liorr@post.bgu.ac.il

Yuval Elovici
Deutsche Telekom Laboratories
Department of Information Engineering
Ben-Gurion University of the Negev
Be'er Sheva, 84105, Israel
elovici@post.bgu.ac.il

ABSTRACT

Selecting the best classifier among the available ones is a difficult task, especially when only instances of one class exist. In this work we examine the notion of combining one-class classifiers as an alternative for selecting the best classifier. In particular, we propose two new one-class classification performance measures to weigh classifiers and show that a simple ensemble that implements these measures can outperform the most popular one-class ensembles. Furthermore, we propose a new one-class ensemble scheme, TUPSO, which uses meta-learning to combine one-class classifiers. Our experiments demonstrate the superiority of TUPSO over all other tested ensembles and show that the TUPSO performance is statistically indistinguishable from that of the hypothetical best classifier.

1. INTRODUCTION AND BACKGROUND

In regular classification tasks we aim to classify an unknown instance into one class from a predefined set of classes. One-class classification aims to differentiate between instances of class of interest and all other instances. The one-class classification task is of particular importance to information retrieval tasks [17]. Consider, for example, trying to identify documents of âĂIJinterestâĂİ to a user, where the only information available is the previous documents that this user has read (i.e. positive examples), yet another example is citation recommendation, in which the system helps authors in selecting the most relevant papers to cite, from a potentially overwhelming number of references [1]. Again, one can obtain representative positive examples by simply going over the references, however it would be hard to identify typical negative examples (the fact that a certain paper is not cited by another paper does not necessarily indicate it is irrelevant). Many one-class classification algorithms have been investigated [2, 25, 14]. While there are plenty of learning algorithms to choose from, identifying the one that performs best in relation to the problem at hand is difficult. This is because evaluating a one-class classifier's performance is problematic. By definition, the data collections only contain one-class examples and thus, performance metrics, such as false-positive (FP), and true negative (TN), cannot be computed. In the absence of FP and TN, derived performance metrics, such as classification accuracy, precision, among others, cannot be computed. Moreover, prior knowledge concerning the classification performance on some previous tasks may not be very useful for a new classification task because classifiers can excel in one dataset and fail in another, i.e., there is no consistent winning algorithm.

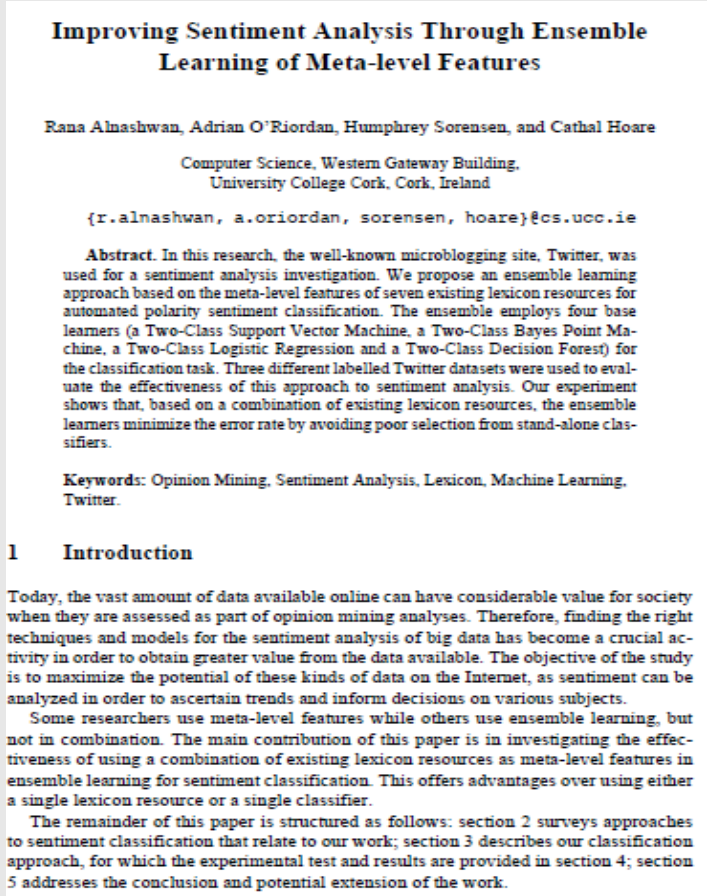
This difficulty can be addressed in two ways. The first option is to select the classifier assumed to perform best according to some heuristic estimate based on the available positive examples (i.e., TP and FN). The second option is to train an ensemble from the available classifiers. To the best of our knowledge, no previous work on selecting the best classifier in the one-class domain has been published and the only available one-class ensemble technique for diverse learning algorithms is the fixed-rule ensemble, which in many cases, as we later show, makes more classification errors when compared to a random-selected classifier.

In this paper we search for a new method for combining one-class classifiers. We begin by presenting two heuristic methods to evaluate the classification performance of one-class classifiers. We then introduce a simple heuristic ensemble that uses these heuristic methods to select a single base-classifier. Later, we present TUPSO, a general meta-learning based ensemble, roughly based on the Stacking technique [29] and incorporates the two classification performance evaluators. We then experiment with the discussed ensemble techniques on forty different datasets. The experiments show that TUPSO is by far the best option to use when multiple one-class classifiers exist. Furthermore, we show that TUPSO's classification performance is strongly correlated with that of the actual best ensemble-member.

1.1 One-Class Ensemble

The main motivation behind the ensemble methodology is to weigh several individual classifiers and combine them to obtain a classifier that outperforms them all. Indeed, previous

Eitan Menahem et al. 2013



Improving Sentiment Analysis Through Ensemble Learning of Meta-level Features

Rana Alnashwan, Adrian O'Riordan, Humphrey Sorensen, and Cathal Hoare

Computer Science, Western Gateway Building,
University College Cork, Cork, Ireland

{r.alnashwan, a.oriordan, sorensen, hoare}@cs.ucc.ie

Abstract. In this research, the well-known microblogging site, Twitter, was used for a sentiment analysis investigation. We propose an ensemble learning approach based on the meta-level features of seven existing lexicon resources for automated polarity sentiment classification. The ensemble employs four base learners (a Two-Class Support Vector Machine, a Two-Class Bayes Point Machine, a Two-Class Logistic Regression and a Two-Class Decision Forest) for the classification task. Three different labelled Twitter datasets were used to evaluate the effectiveness of this approach to sentiment analysis. Our experiment shows that, based on a combination of existing lexicon resources, the ensemble learners minimize the error rate by avoiding poor selection from stand-alone classifiers.

Keywords: Opinion Mining, Sentiment Analysis, Lexicon, Machine Learning, Twitter.

1 Introduction

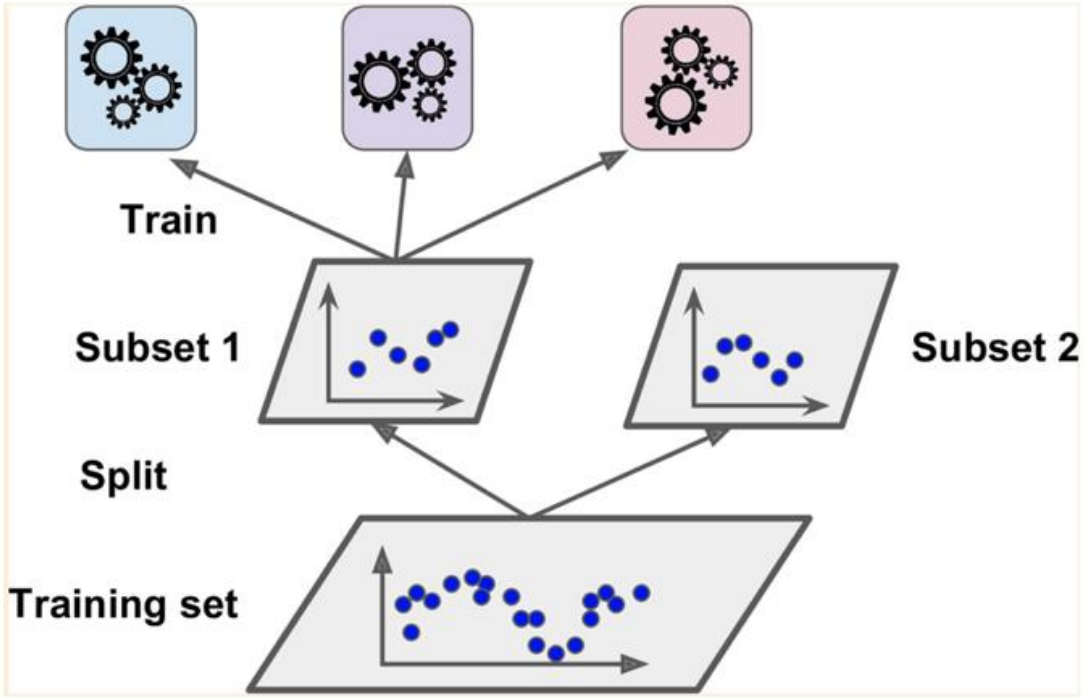
Today, the vast amount of data available online can have considerable value for society when they are assessed as part of opinion mining analyses. Therefore, finding the right techniques and models for the sentiment analysis of big data has become a crucial activity in order to obtain greater value from the data available. The objective of the study is to maximize the potential of these kinds of data on the Internet, as sentiment can be analyzed in order to ascertain trends and inform decisions on various subjects.

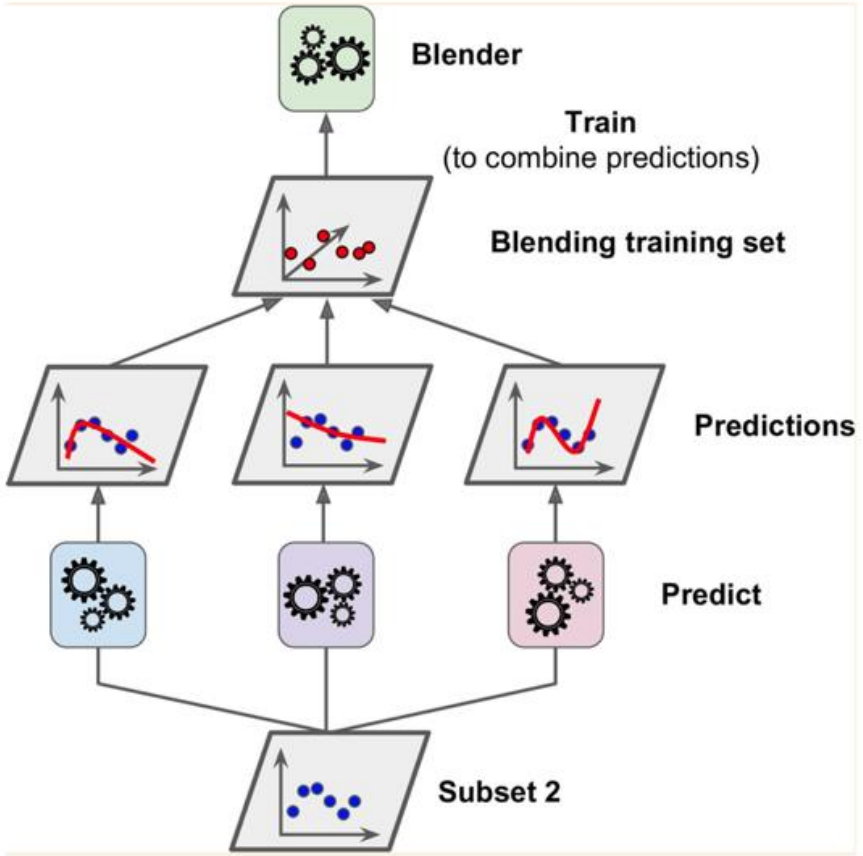
Some researchers use meta-level features while others use ensemble learning, but not in combination. The main contribution of this paper is in investigating the effectiveness of using a combination of existing lexicon resources as meta-level features in ensemble learning for sentiment classification. This offers advantages over using either a single lexicon resource or a single classifier.

The remainder of this paper is structured as follows: section 2 surveys approaches to sentiment classification that relate to our work; section 3 describes our classification approach, for which the experimental test and results are provided in section 4; section 5 addresses the conclusion and potential extension of the work.

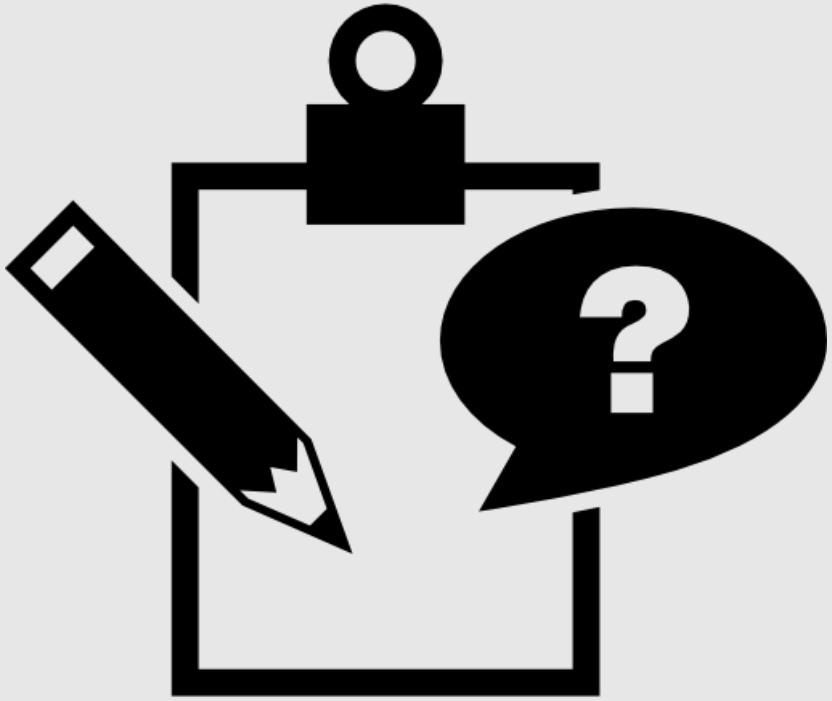
Rana Alnashwan et al. 2003

Stacking





〈https://stats.stackexchange.com/questions/350897/stacking-without-splitting-data〉



Before Finish Class

Hearthstone

Google Deepmind AI tries it hand at creating Hearthstone a…

Google Deepmind tasks a machine learning system with recreating c…

www.techrepublic.com

알파고로 바둑에서 엄청난 모습을 보여준 구글 딥마인드 AI가
하스스톤과 매더게(MTG) 카드 연구를 시작했습니다

Hearthstone


```
class MadderBomber(MinionCard):          BLEU = 100.0
  def __init__(self):
    super().__init__("Madder Bomber", 5,
    CHARACTER_CLASS.ALL, CARD_RARITY.RARE,
    battlecry=Battlecry(Damage(1),
    CharacterSelector(players=BothPlayer(),
       picker= RandomPicker(6))))

def create_minion(self, player):§
  return Minion(5, 4)§
```


```
class Preparation(SpellCard):            BLEU = 64.2
  def __init__(self):
    super().__init__("Preparation", 0,
    CHARACTER_CLASS.ROGUE, CARD_RARITY.EPIC,
    target_func=hearthbreaker.targeting.find_minion_spell_target)

  def use(self, player, game):
    super().use(player, game)
    self.target.change_attack(3)
                          player.add_aura(AuraUntil(ManaChange(-3),
                          CardSelector(condition=IsSpell()), SpellCast()))
```

Figure 5: Examples of decoded cards from HS. Copied segments are marked in green and incorrect segments are marked in red.

Time Table

10월 26일: 시험기간
11월 02일: 8,9장 (차원축소, 텐서플로 시작하기)
11월 09일: 10장 (인공신경망 소개)
11월 11일: 진교훈 생일
11월 16일: 11장 (심층 신경망 훈련)

12월에 끝냅시다.


Ideas worth spreading
– TED Talks

고생하셨습니다.