

Learning-To-Rank

Alfan F. Wicaksono

Learning-To-Rank (LTR / LETOR)

Machine learning models to solve a ranking problem.

q1 dilempar ke sparse retrieval kita (BM25/TFIDF)

Jadi Q1 -> Sparse Retrieval -> terus dari semua dokumen di cek kalau Q1 relevan apa engga terhadap dokumen-dokumen (nah ada 3 kemungkinan 1/0/?) ? artinya itu gak tau (di infer tidak relevan)

Q2 juga dilatih, sama perci. Nah total kalau ada 5 dokumen, ada 5 dari Q1, 5 Dari Q2 maka ada 10 dokumen yang di latih

Given a query q and top- n documents (retrieved by TF-IDF or BM25) $D = d_1, d_2, \dots, d_n$, we would like to learn a function $f(q, D)$ that returns an ordered list of documents D^* ranked from the most to the least relevant to the query q .

Learning-To-Rank is different from Classification & Regression.

Mengapa IR tidak menggunakan ML dari dulu?

- Modern supervised ML has been around for about **25 years**.
- TF-IDF and OKAPI BM25 has been around for about **50 years**.
- Naïve Bayes has been around for about **60 years**.
- **Dahulu:** poor machine learning techniques.
- **Dahulu:** Limited training data and not enough features for ML to show value.

Why is ML needed now?

Modern Web search systems records a great number of features:

- Log frequency of query word
- # images on page
- # out links on page
- # clicks on page
- # query reformulation
- "Scroll-down" & "scroll-up" actions
- Dwell time

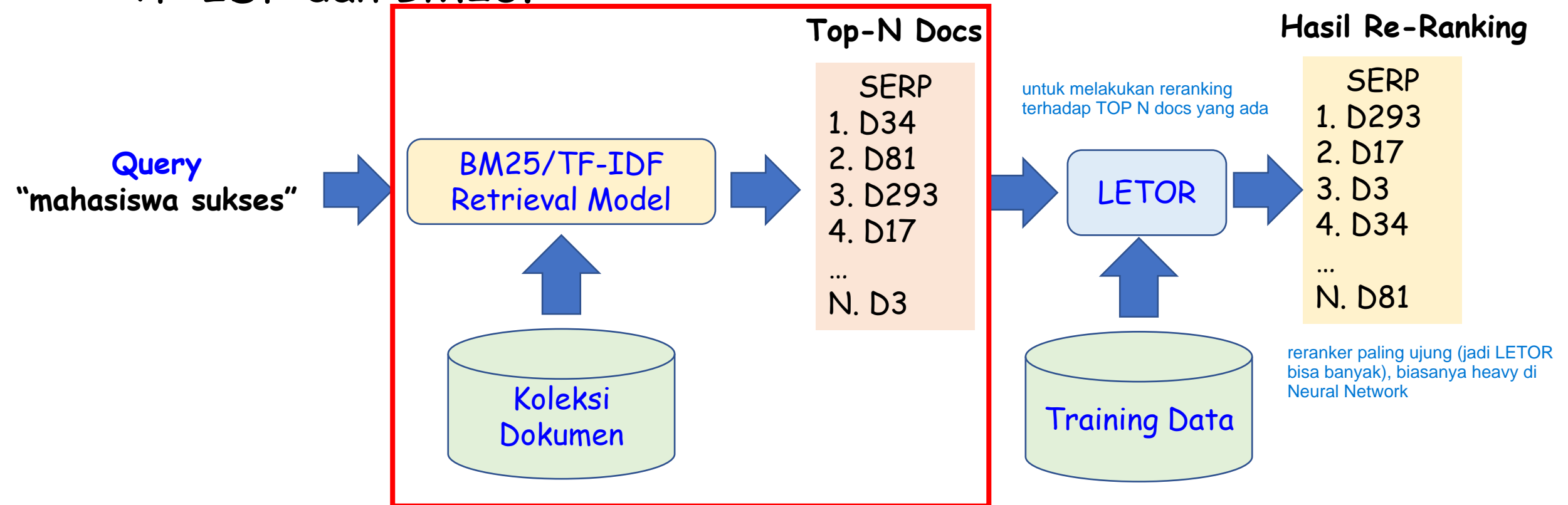
The New York Times in 2008-06-03 says that **Google** was using over 200 such features.

What about today? over 500-800 features?

LETOR for Re-Ranking

namanya: arsitektur cascade model

Di kebanyakan kasus, LETOR digunakan untuk re-ranking hasil retrieval yang dilakukan oleh sparse retrieval models seperti TF-IDF dan BM25.



Training Data Format

Pointwise LTR

Assuming training data is available consisting of query-document pairs (q, d) represented as feature vectors x with relevance ranking c .

$x_1: q_1, d_1$	$y_1: 3$ (fully relevant)
$x_2: q_1, d_5$	$y_2: 1$ (marginally relevant)
$x_3: q_1, d_9$	$y_3: 2$ (somewhat relevant)
$x_4: q_2, d_2$	$y_4: 1$
$x_5: q_2, d_{13}$	$y_5: 3$
$x_6: q_2, d_{18}$	$y_6: 2$
...	...

Diberikan q dan sekumpulan dokumen d_1, d_2, \dots , fungsi $f(q, d_i)$ digunakan untuk scoring masing-masing dokumen, dan ranking dibentuk dengan sorting dokumen secara menurun.

Training Data Format

Pairwise LTR

Assuming training data is available consisting of pairs of documents for each query **(q, d1, d2)**.

masih di train menggunakan pairwise cuman kalau udah fit (ada parameter thetanya, maka kalau kita scoring, kita tinggal panggil F nya aja (jadi kalau semua parameter (theta) udah diketahui, itu jadi point wise)

kalau document 1 lebih relevan dari dokument 5 (d1 > d5)
maka 1, kalau sama 0.5 else 0

x1: q1, (d1,d5)	Y1: 1 (d1 > d5)
x2: q1, (d5,d9)	Y2: 0 (d5 < d9)
x3: q1, (d4, d10)	Y3: 0.5 (d4 = d10)
x4: q2, (d2, d18)	Y4: 0 (d2 < d18)
x5: q2, (d13,d18)	Y5: 1 (d13 > d18)

...

...

^ ini data digenerate oleh humans

Catat bahwa **f(q, di)** masih pointwise! Namun **f(q, di)** di-train dengan cara pairwise. Yang di-fit dengan **y** adalah fungsi berikut:

$$P(d_i > d_j | \theta) = \frac{\text{sigmoid}}{1 + \exp(-(f(q, d_i | \theta) - f(q, d_j | \theta)))}$$

ini sigmoid

jadi nanti di pair kan dengan binary cross entropy

Yahoo! Learning-To-Rank Challenge

- Tahun 2011
- 36,251 queries, 883,000 documents, 700 features, 5 relevance levels
- Winner (Burges et al.)
 - 8 Tree Ensembles (LambdaMART)
 - 2 LambdaRank Neural Nets
 - 2 Logistic Regression Models

Model LTR yang dipelajari

- RankNet
- LambdaRank
- LambdaMART
- Untuk bisa memahami 3 model fondasi LTR di atas, perlu belajar:
 - Regression Trees
 - Gradient-Boosting Framework

Regression Trees

Regression Trees

- Decision Tree untuk prediksi real value (regression problem)
- Splitting criterion:
 - Choose split values that minimizes the **impurity or loss-function** of the values in each subset S_i of S :

kita mau minimize loss dari jumlah semua $L(S_i)$ agar minimal

$|S|$ total banyaknya titik (koordinat)

$$G = \sum_i \frac{|S_i|}{|S|} L(S_i)$$

dengan $L(S_i)$ adalah node-level impurity or loss function.

Asumsi koordinat di split equally pakai divider di $x = 4$ (bebas), $y = 0$ (jadi garis vertical)

S Left: ada 4

S Right: ada 4 -> jadi ngitungnya $4/8 * \text{Loss}(S \text{ Left}) + 4/8 * \text{Loss}(S \text{ Right})$

If $L(S_i)$ is **Mean Squared Error**, then predicted value of a leaf node is the **mean of all instances**.

$$L(S_i) = \frac{1}{|S_i|} \sum_{x_j \in S_i} \frac{1}{2} (y_j - f(x_j))^2$$

Binary Regression Trees

- Cari sebuah split v_m yang memisahkan:

$$S_{left}(v_m) = \{(x, y) | x_j < v_m\} \quad \text{dan} \quad S_{right}(v_m) = S - S_{left}$$

- Bagaimana mencari split value v_m ?

$$v_m = \operatorname{argmin}_v G(v)$$

$$G(v) = \frac{|S_{left}(v)|}{|S|} L(S_{left}(v)) + \frac{|S_{right}(v)|}{|S|} L(S_{right}(v))$$

Binary Regression Trees

- Penentuan nilai “wakil” (predicted value) w di setiap leaf node bergantung loss function L , yaitu:

$$w = \underset{w}{\operatorname{argmin}} \sum_{x_i \in \text{leaf}} L(y_i, w)$$

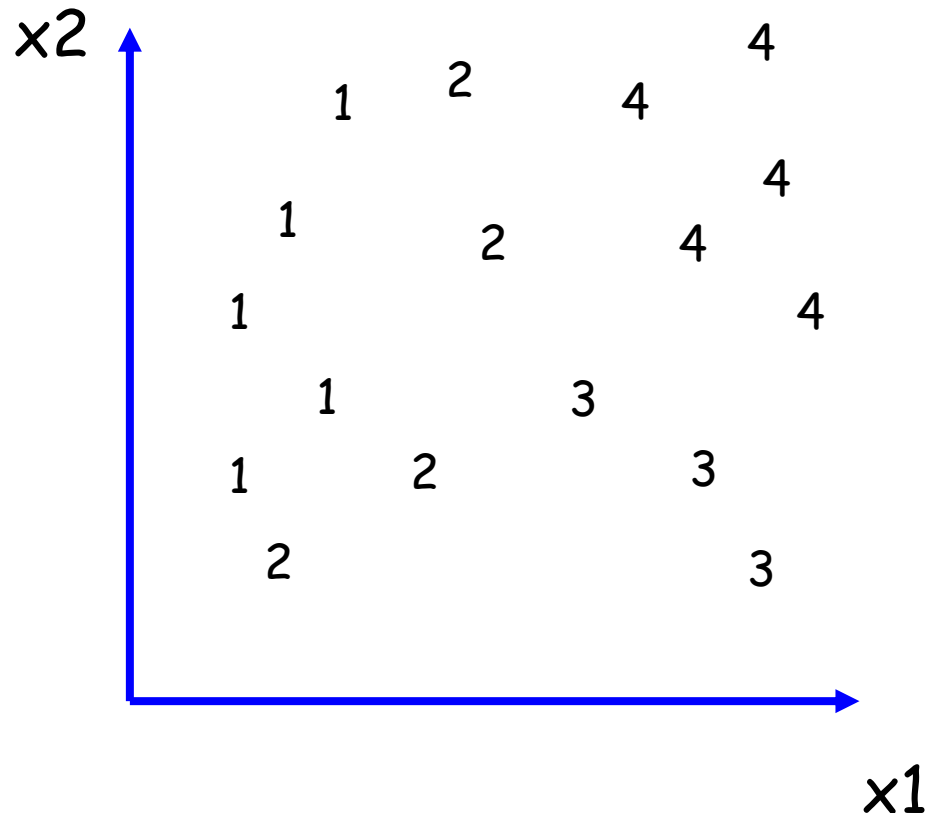
- Dapat dibuktikan bahwa jika L adalah MSE (mean squared error), predicted value di sebuah leaf/terminal node adalah **mean**.

$$w = \operatorname{mean}(x | x \in \text{leaf}) \quad \text{jika} \quad L(S_i) = \frac{1}{|S_i|} \sum_{x_j \in S_i} \frac{1}{2} (y_j - f(x_j))^2$$

Training Regression Tree

Loss-function = MSE

Cari *split variable* dan *split value* yang meminimalkan predicted error!

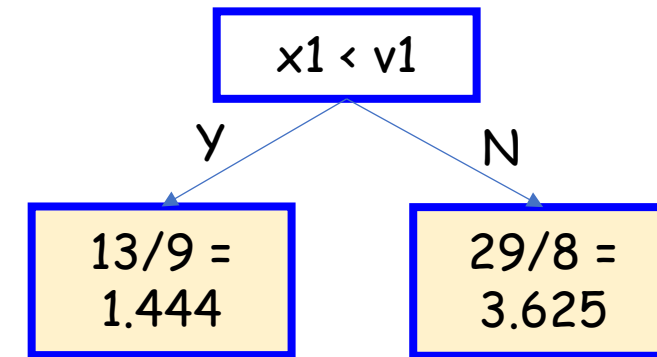
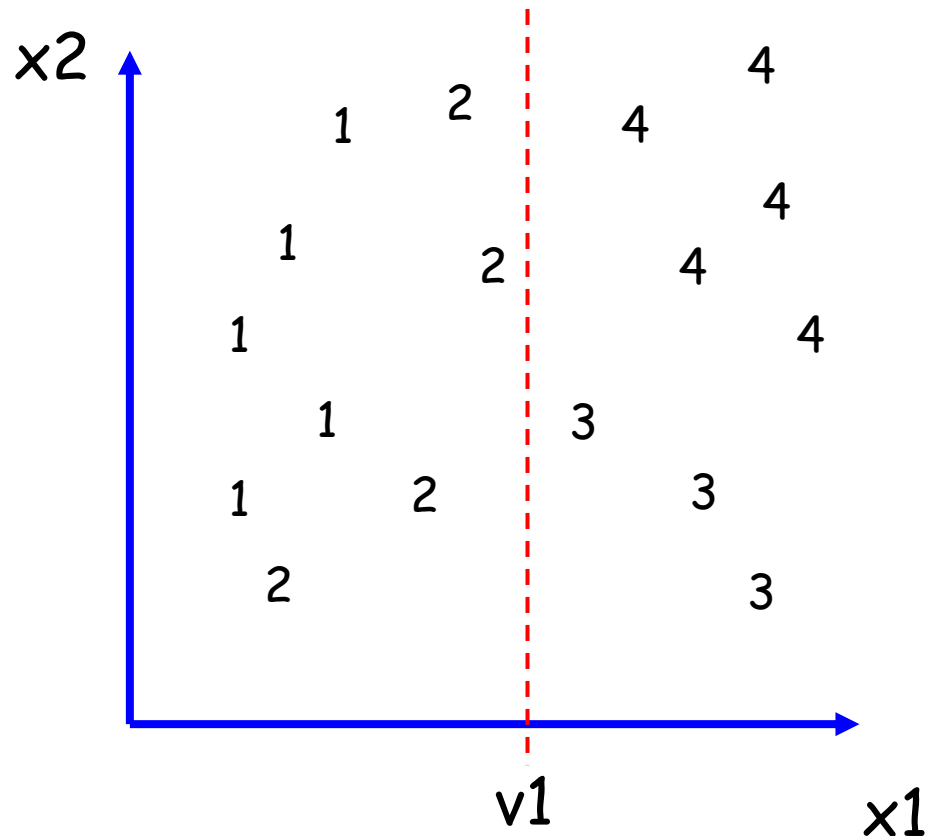


Misal, decision tree regressor dalam bentuk $f(x_1, x_2)$ dan kita menerapkan binary splitting.

Training Regression Tree

Loss-function = MSE

Misal, pada iterasi pertama, **split variable** x_1 dengan **split value** v_1 memberikan predicted error paling minimal.



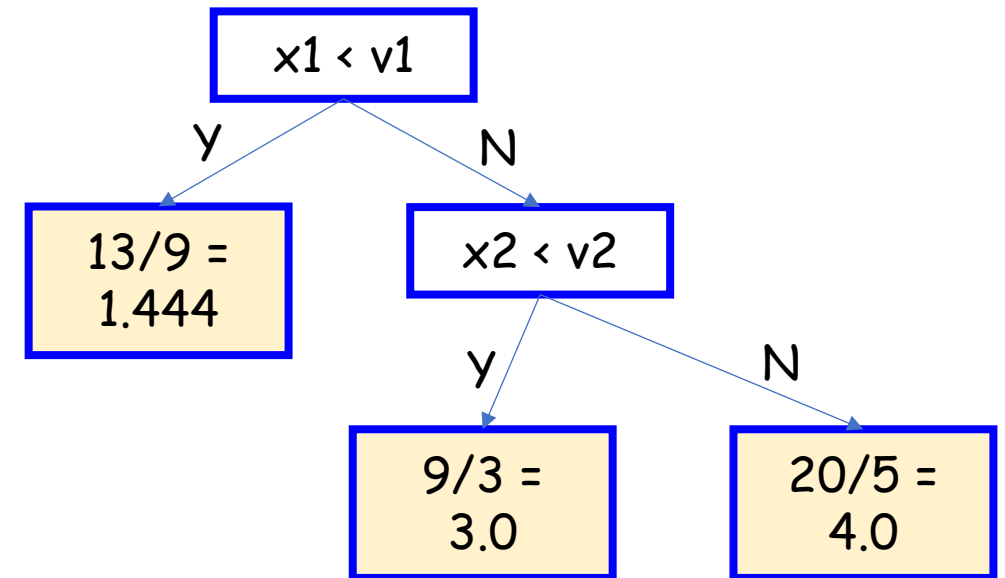
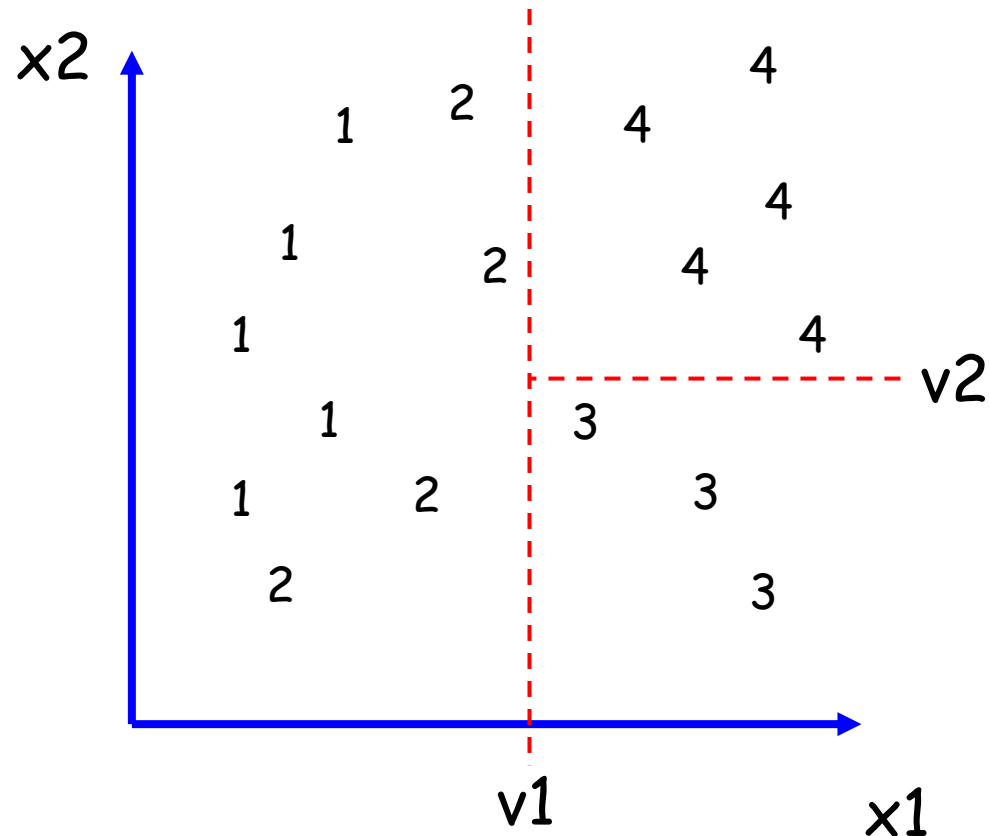
$$G(v_1) = \frac{9}{17} \left[\frac{5}{9} \cdot \frac{1}{2} \cdot (1 - 1.444)^2 + \frac{4}{9} \cdot \frac{1}{2} \cdot (2 - 1.444)^2 \right] + \frac{8}{17} \left[\frac{3}{8} \cdot \frac{1}{2} \cdot (3 - 3.625)^2 + \frac{5}{8} \cdot \frac{1}{2} \cdot (4 - 3.625)^2 \right] = 0.12$$

Latihan: coba hitung G untuk split value lain; dan bahkan untuk split variable x_2 ! Apakah ada yang < 0.12 ?

Training Regression Tree

Loss-function = MSE

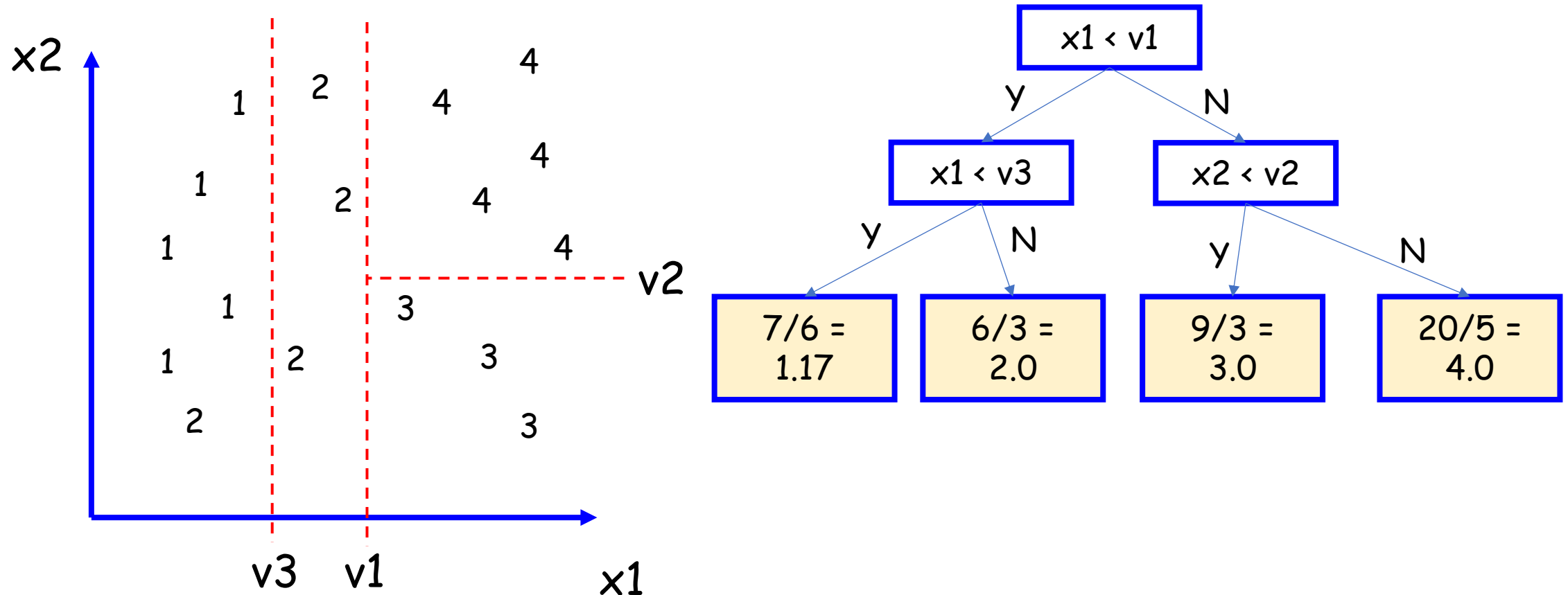
Misal, pada iterasi pertama, **split variable** x_1 dengan **split value** v_1 memberikan predicted error paling minimal.



Training Regression Tree

Loss-function = MSE

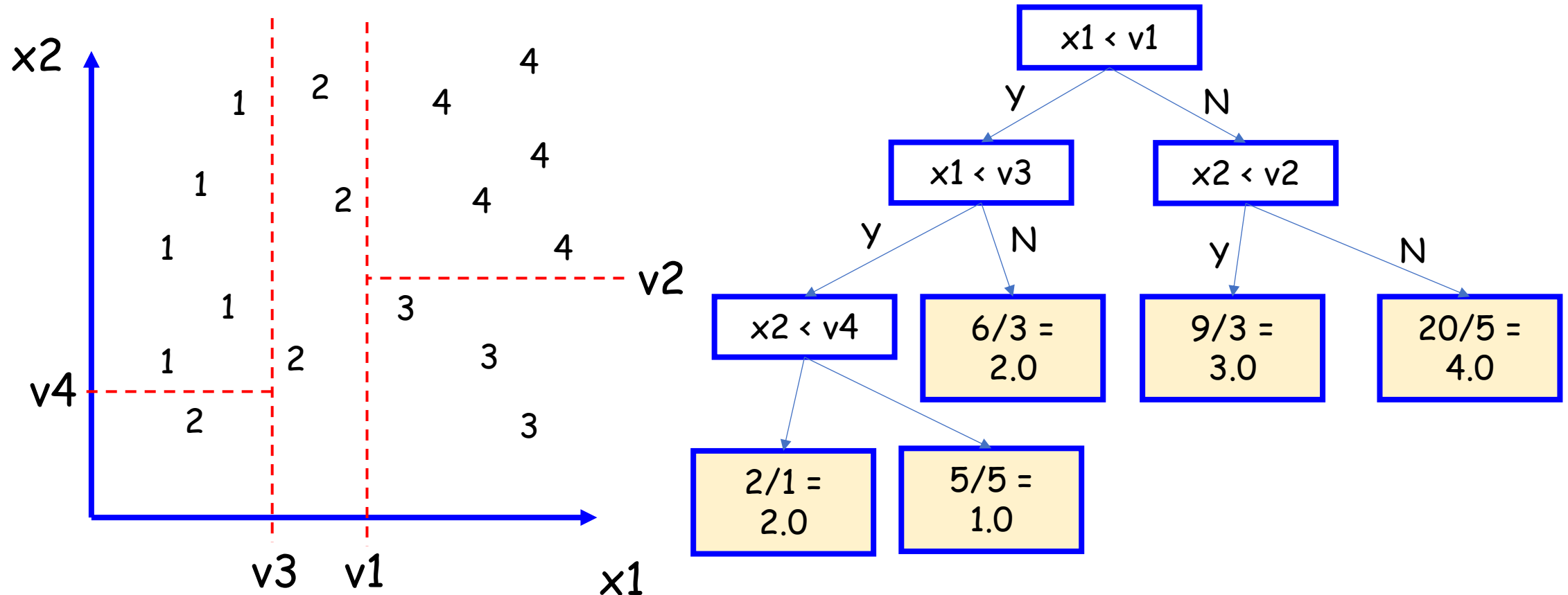
Misal, pada iterasi pertama, **split variable** x_1 dengan **split value** v_1 memberikan predicted error paling minimal.



Training Regression Tree

Loss-function = MSE

Misal, pada iterasi pertama, **split variable** x_1 dengan **split value** v_1 memberikan predicted error paling minimal.



Training Regression Tree

Kapan berhenti splitting?

- Cutoff pada nilai G
- Tree depth
- # leaf nodes

Konsep Gradient Boosting

ensemble Learning:

1. Bagging:

- > D01 ada fitur X1 X2 X3, hasilnya M

D02 ada fitur X1 X2 X3, hasilnya M

D03 ada fitur X1 X2 X3, hasilnya M

Lalu M itu di combine

2. Stacking: ada M model yang dilatih dengan dataset yang sama. Lalu M1 , M2, Mn -> Hasil prediksi semua modelnya di combine, merupakan meta features . Nah meta features ini membuat meta model lain memprediksi

3. Boosting: additive model, mulai dari classifier/regressor yang paling simple, nah errornya ini ditambah dengan H1, hasilnya di evaluasi lagi errornya dan di tambah oleh H2, dst.

Boosting

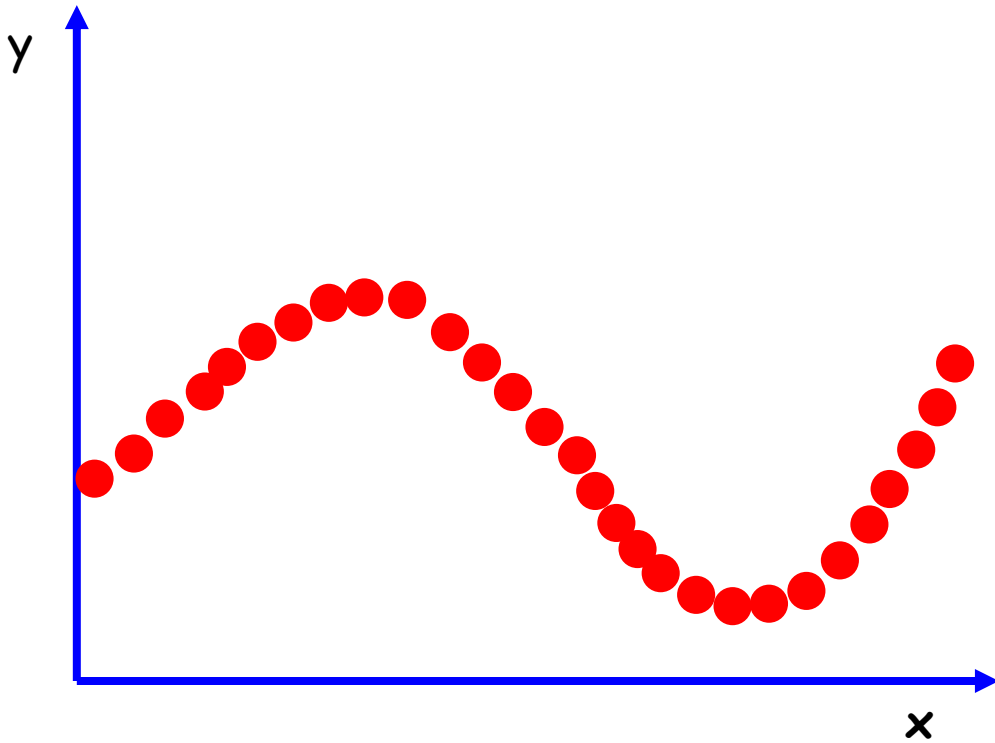
- Bagaimana membuat highly-effective model dengan menggabungkan banyak model-model yang "lemah"?
- Sebuah **Ensemble**, misal dengan additive model:

Y_{true} diprediksi dengan $F_m(X) = F_0(X) + h_1(X|\theta_1) + h_2(X|\theta_2) + \dots + h_m(X|\theta_m)$

Di setiap stage, $F_i(X)$ dilatih untuk memprediksi error (**pseudo-residual**) yang dihasilkan model pada stage sebelumnya, yaitu $Y_{true} - F_{i-1}(X)$

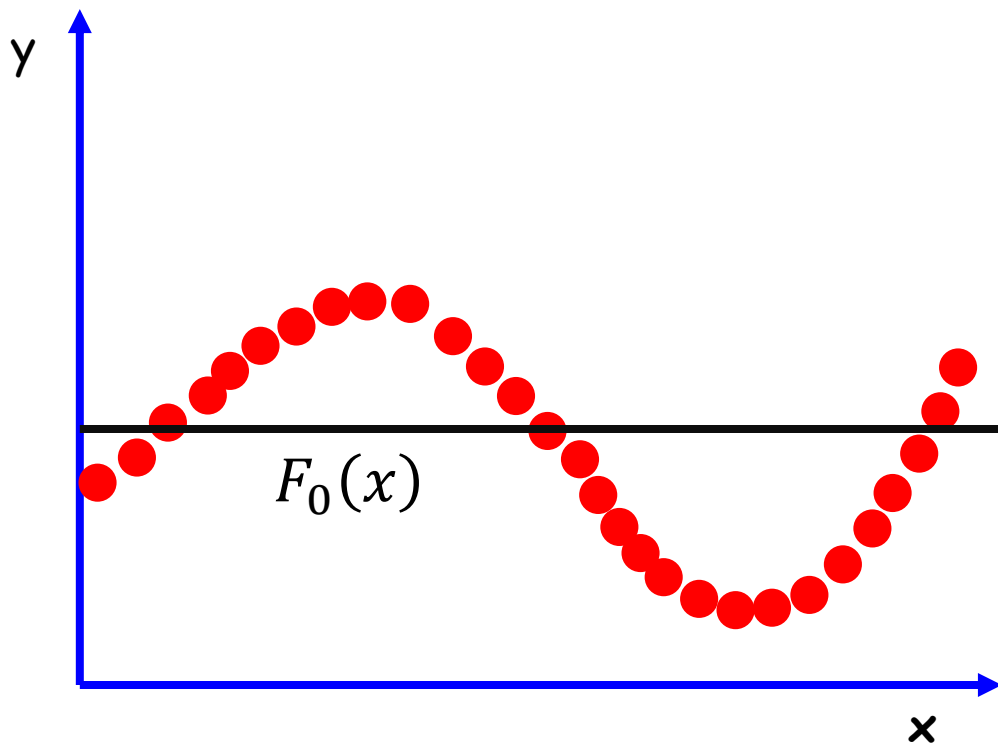
Boosting

Misal, $\{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$ adalah training data



Boosting

Misal, $\{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$ adalah training data

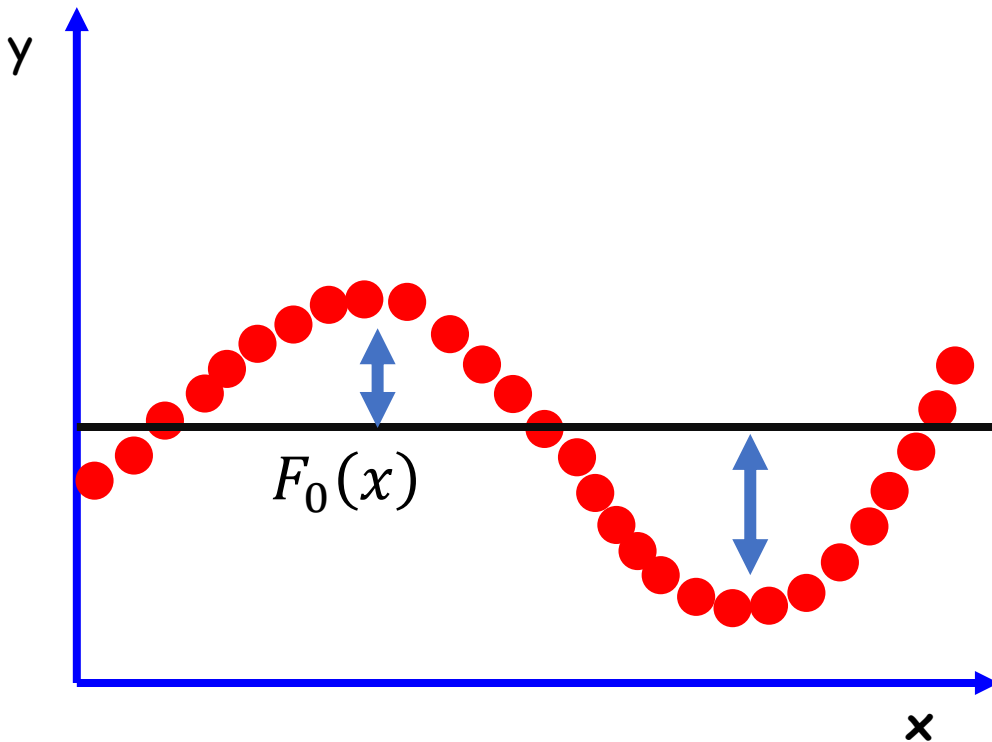


Dimulai dari sebuah "weak learner" simple:

$$F(x) = F_0(x) = \operatorname{argmin}_{\gamma} \sum_{i=1}^n L(y_i, \gamma)$$

Boosting

Misal, $\{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$ adalah training data



$$F(x) = F_0(x) = \operatorname{argmin}_{\gamma} \sum_{i=1}^n L(y_i, \gamma)$$

Pseudo-Residual

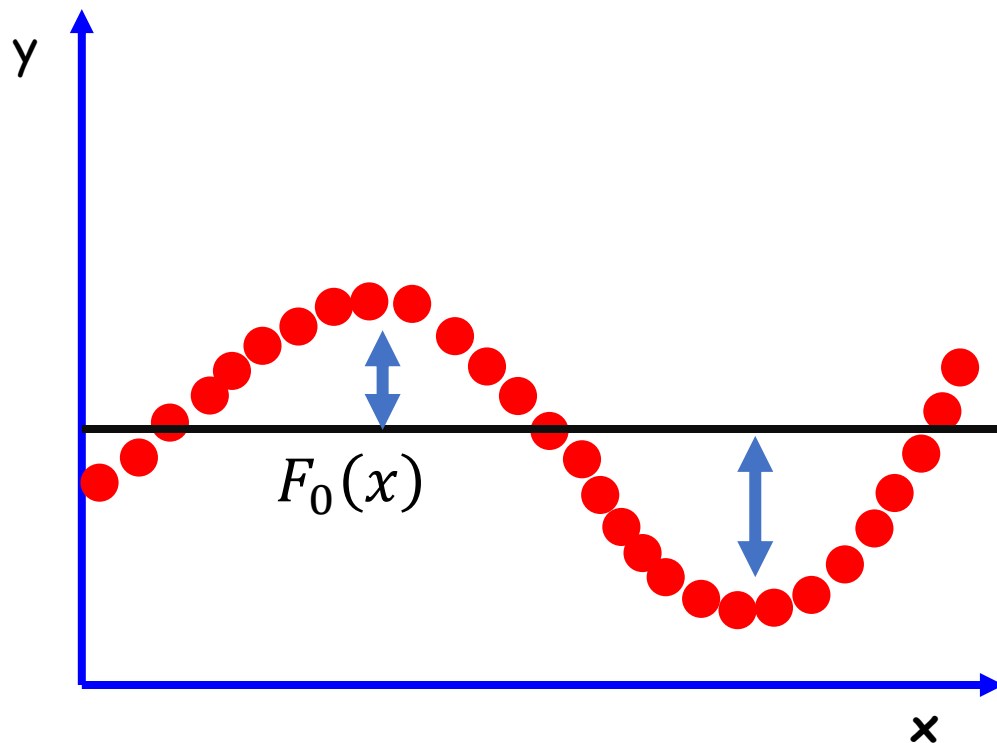
x	Pseudo-Res.
x1	$y_1 - F_0(x_1)$
x2	$y_2 - F_0(x_2)$
...	...
xn	$y_n - F_0(x_n)$

$$y = F_0(x) + \epsilon^{(1)}$$

awal-awal predicted value ($F_0(x)$) tidak sempurna ada error, nah kalo kita tambahkan errornya kan jadi Y (stay true to label), jadi sama persis

Boosting

Misal, $\{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$ adalah training data



$$F(x) = F_0(x) = \operatorname{argmin}_{\gamma} \sum_{i=1}^n L(y_i, \gamma)$$

Pseudo-Residual

x	Pseudo-Res.
x1	$y_1 - F_0(x_1)$
x2	$y_2 - F_0(x_2)$
...	...
xn	$y_n - F_0(x_n)$

$$y = F_0(x) + \epsilon^{(1)}$$

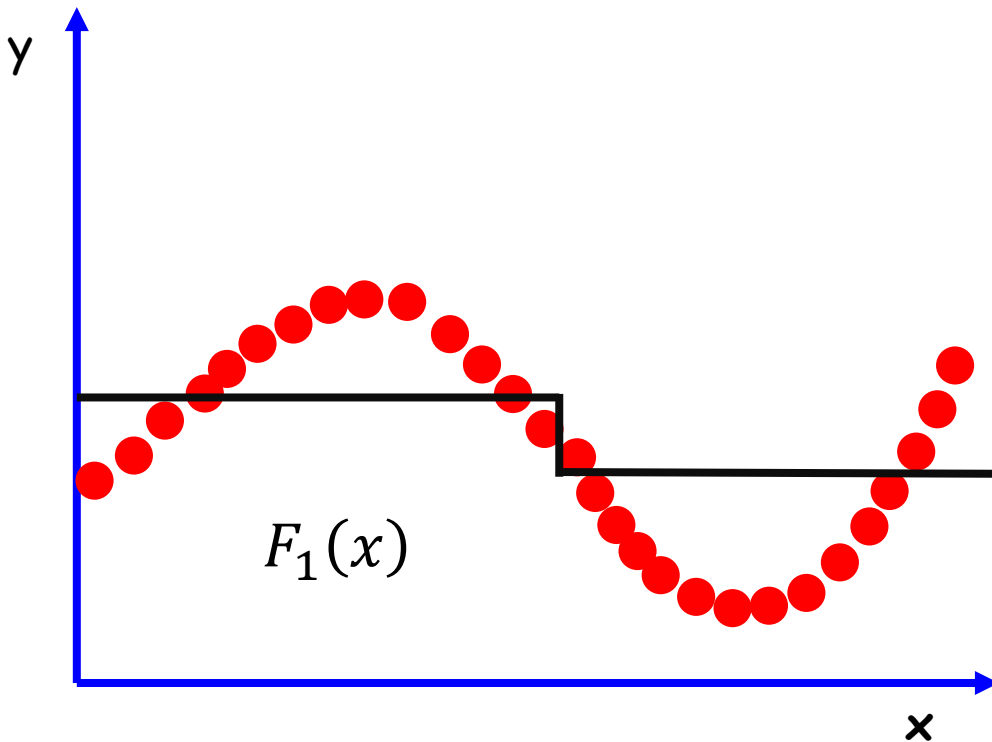
Nah lalu yang kita predict sendiri ada ERROR itu sendiri

Latih weak learner ke-2, $h_1(x)$, agar fit dengan **pseudo-residual**, yaitu dengan:

$$h_1(x) = \operatorname{argmin}_h \left[\sum_{i=1}^n L(y_i - F_0(x_i), h(x_i)) \right]$$

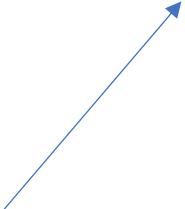
Boosting

Misal, $\{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$ adalah training data



Pseudo-Residual

x	Psedo-Res.
x1	$y_1 - F_0(x_1)$
x2	$y_2 - F_0(x_2)$
...	...
xn	$y_n - F_0(x_n)$

$$y = F_0(x) + \epsilon^{(1)}$$


Latih weak learner ke-2, $h_1(x)$, agar fit dengan **pseudo-residual**, yaitu dengan:

$$h_1(x) = \operatorname{argmin}_h \left[\sum_{i=1}^n L(y_i - F_0(x_i), h(x_i)) \right]$$

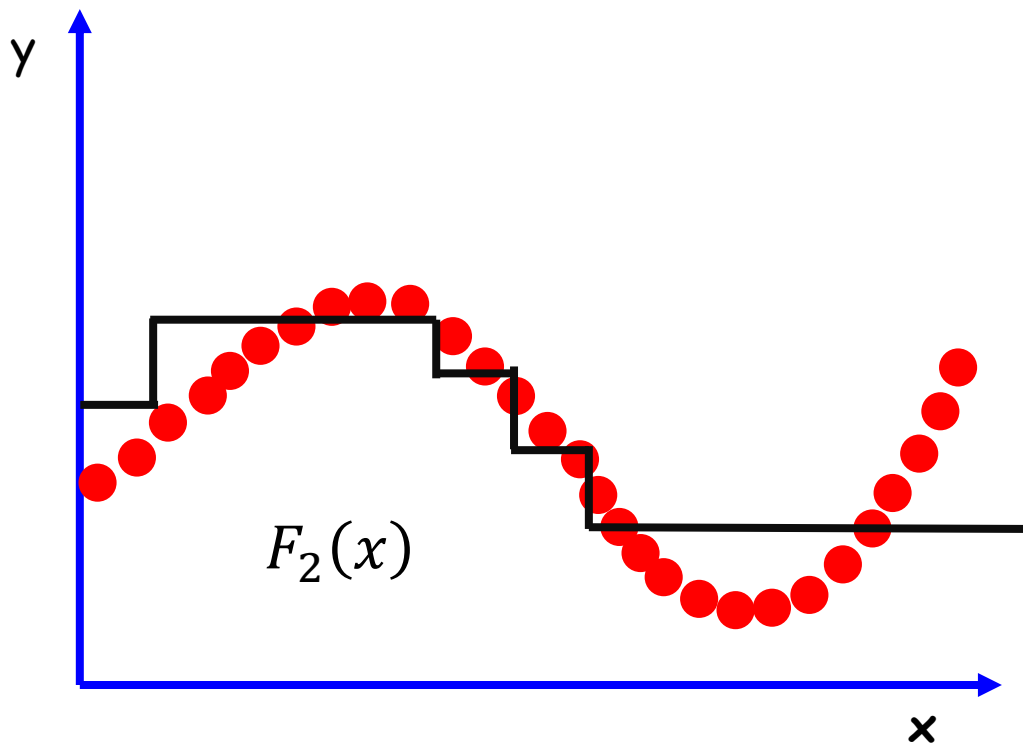
hasil 1 kali boosting

Model baru yang lebih baik dari sebelumnya:

$$\longrightarrow F(x) = F_1(x) = F_0(x) + h_1(x)$$

Boosting

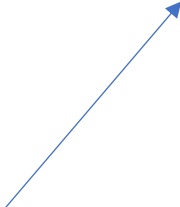
Misal, $\{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$ adalah training data



dihitung lagi pseudo residual dari $F_1(x)$, lalu ditambah lagi

Pseudo-Residual

x	Psedo-Res.
x_1	$y_1 - F_1(x_1)$
x_2	$y_2 - F_1(x_2)$
...	...
x_n	$y_n - F_1(x_n)$

$$y = F_1(x) + \epsilon^{(2)}$$


Latih weak learner ke-3, $h_2(x)$, agar fit dengan **pseudo-residual**, yaitu dengan:

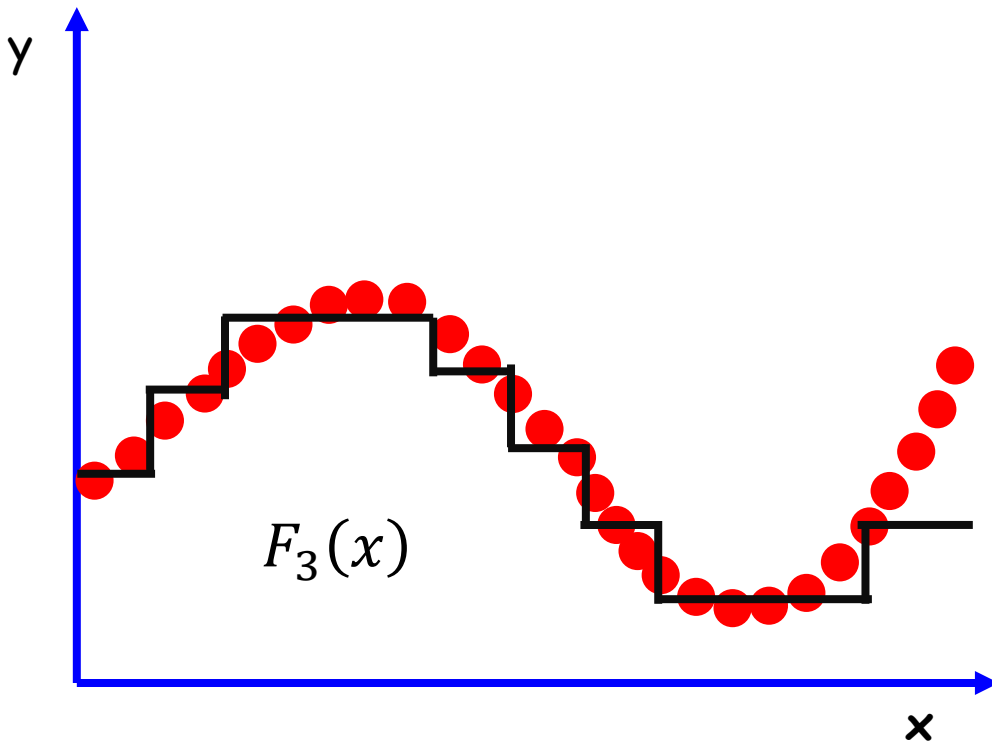
$$h_2(x) = \operatorname{argmin}_h \left[\sum_{i=1}^n L(y_i - F_1(x_i), h(x_i)) \right]$$

Model baru yang lebih baik dari sebelumnya:

$$\longrightarrow F(x) = F_2(x) = F_0(x) + h_1(x) + h_2(x)$$

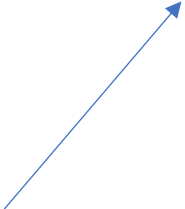
Boosting

Misal, $\{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$ adalah training data



Pseudo-Residual

x	Psedo-Res.
x1	$y_1 - F_2(x_1)$
x2	$y_2 - F_2(x_2)$
...	...
xn	$y_n - F_2(x_n)$

$$y = F_2(x) + \epsilon^{(3)}$$


Latih weak learner ke-4, $h_3(x)$, agar fit dengan **pseudo-residual**, yaitu dengan:

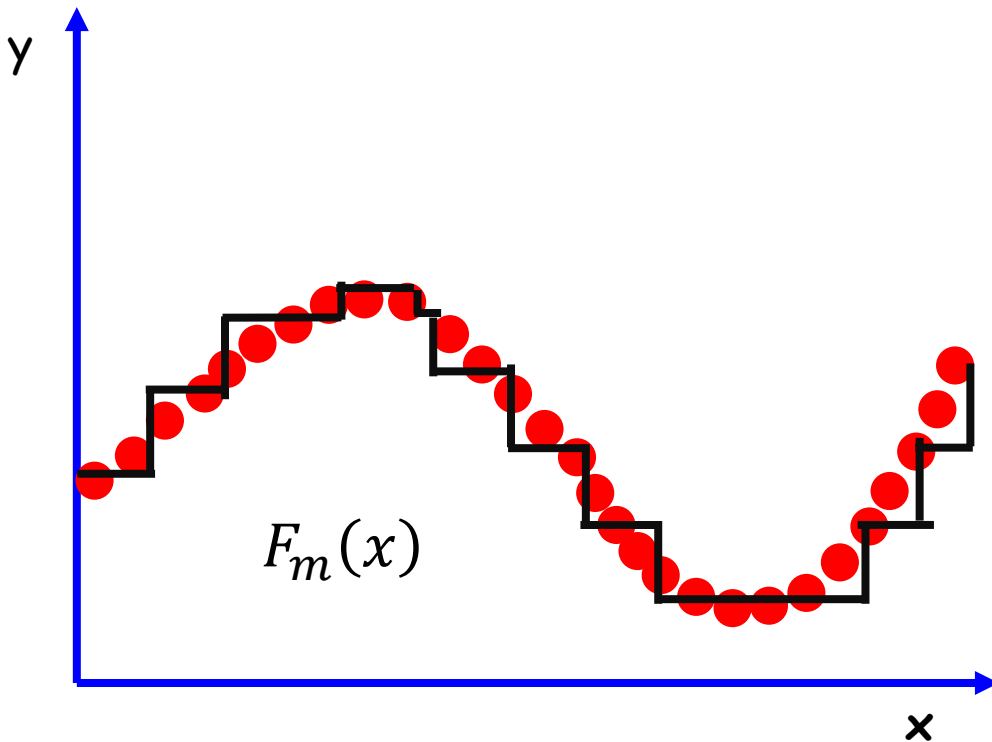
$$h_3(x) = \operatorname{argmin}_h \left[\sum_{i=1}^n L(y_i - F_2(x_i), h(x_i)) \right]$$

Model baru yang lebih baik dari sebelumnya:

$$\longrightarrow F(x) = F_3(x) = F_0(x) + h_1(x) + h_2(x) + h_3(x)$$

Boosting

Misal, $\{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$ adalah training data



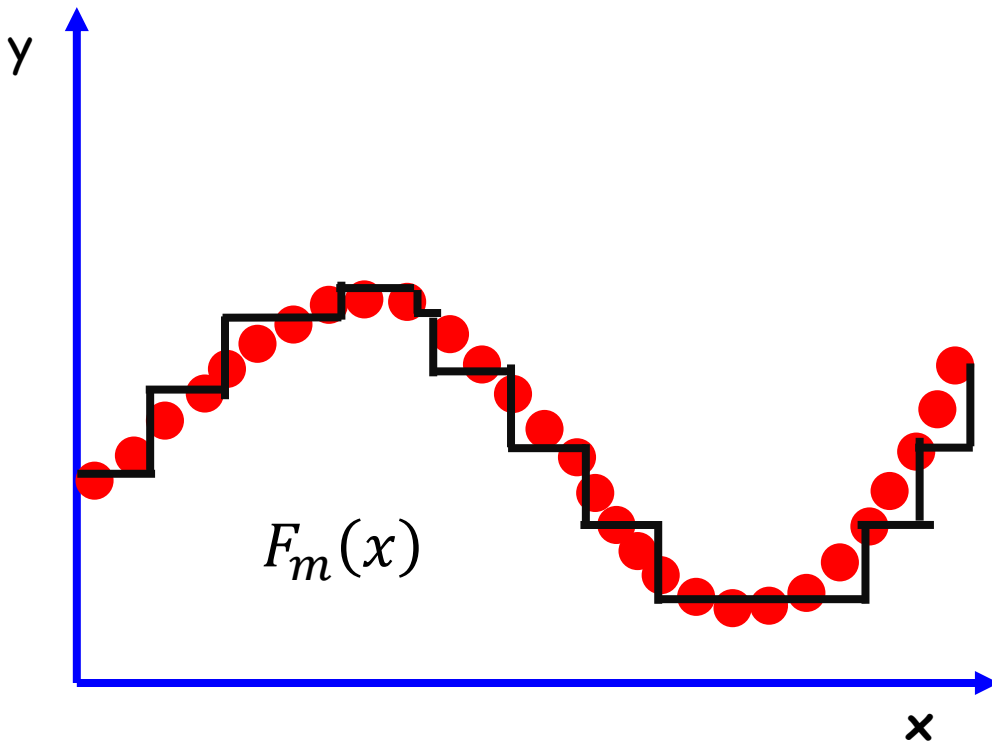
Dan seterusnya ...

Model baru yang lebih baik dari sebelumnya:

$$\longrightarrow F(x) = F_m(x) = F_0(x) + h_1(x) + \dots + h_m(x)$$

Boosting

Misal, $\{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$ adalah training data



Di setiap iterasi, kita fit weak learner dengan pseudo-residual:

$$h_m(x) = \operatorname{argmin}_h \left[\sum_{i=1}^n L(\epsilon_i^{(m)}, h(x_i)) \right]$$



$$h_m(x) = \operatorname{argmin}_h \left[\sum_{i=1}^n L(y_i - F_{m-1}(x_i), h(x_i)) \right]$$



$$h_m(x) = \operatorname{argmin}_h \left[\sum_{i=1}^n L(y_i, F_{m-1}(x_i) + h(x_i)) \right]$$

Model baru yang lebih baik dari sebelumnya:

$$\longrightarrow F(x) = F_m(x) = F_0(x) + h_1(x) + \dots + h_m(x)$$

Boosting

$$F_m(x) = F_0(x) + h_1(x) + h_2(x) + \cdots + h_m(x)$$



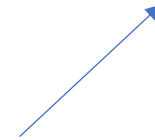
$$F_m(x) = F_{m-1}(x) + h_m(x)$$

Cari fungsi $h_m(x)$ yang memprediksi pseudo-error yang dihasilkan oleh $F_{m-1}(x)$.



jadi prediksi pseudo-errornya

$$F_m(x) = F_{m-1}(x) + \underset{h \in H}{\operatorname{argmin}} \left[\sum_{i=1}^n L(y_i, F_{m-1}(x_i) + h(x_i)) \right]$$



Boosting

$$F_m(x) = F_0(x) + \beta_1 h_1(x) + \beta_2 h_2(x) + \cdots + \beta_m h_m(x)$$



$$F_m(x) = F_{m-1}(x) + \beta_m h_m(x)$$



$$F_m(x) = F_{m-1}(x) + \beta_m \operatorname{argmin}_{h \in H} \left[\sum_{i=1}^n L(y_i, F_{m-1}(x_i) + h(x_i)) \right]$$

Gradient Boosting

tiba-tiba ada kata gradien boosting.

Karena pada saat mencari pseudoresidual loss = $(y - F_{m-1}(X)) = h_m(x)$

$F_m(x) = F_{m-1}(x) + h_m(x)$, Karena proporsional loss nya dengan $h_m(X)$

H itu adalah loss model dari boosting. Maka bisa dipakai gradien

$$F_m(x) = F_{m-1}(x) + \beta_m \operatorname{argmin}_{h \in H} \left[\sum_{i=1}^n L(y_i, F_{m-1}(x_i) + h(x_i)) \right]$$

Sudut pandang lain dalam melihat masalah ini adalah dengan melihat **pseudo-residual sebagai negative gradient** dari loss function dan kemudian melakukan update seperti "gradient-descent algorithm":

$$F_m(x) = F_{m-1}(x) + \beta_m \sum_{i=1}^n - \frac{\partial L(y_i, F_{m-1}(x_i))}{\partial F_{m-1}(x_i)}$$

Cari fungsi $h_m(x)$ yang memprediksi negative gradient dari L!

Gradient Boosting

Mengapa? Pertimbangkan kasus khusus jika **Loss** adalah berbentuk **Squared Error**:

$$L = \sum_{i=1}^n \frac{1}{2} (y_i - F_{m-1}(x_i))^2$$

$$-\frac{\partial L}{\partial F_{m-1}(x_i)} = y_i - F_{m-1}(x_i) \sim h_m(x_i)$$

“proportional”

Observasi ini menyarankan bahwa Gradient Boosting bisa **diperumum** dengan **Loss Function** apapun dengan:

$$F_m(x) = F_{m-1}(x) + \beta_m \sum_{i=1}^n -\frac{\partial L(y_i, F_{m-1}(x_i))}{\partial F_{m-1}(x_i)}$$

Gradient Boosting Framework

Inisialisasi model dengan nilai konstan:

$$F_0(x) = \operatorname{argmin}_{\gamma} \sum_{i=1}^n L(y_i, \gamma)$$

For $m = 1$ to M :

1. For $i = 1$ to n :

Hitung pseudo-residual: $r_{im} = - \left[\frac{\partial L(y_i, F_{m-1}(x_i))}{\partial F_{m-1}(x_i)} \right]$

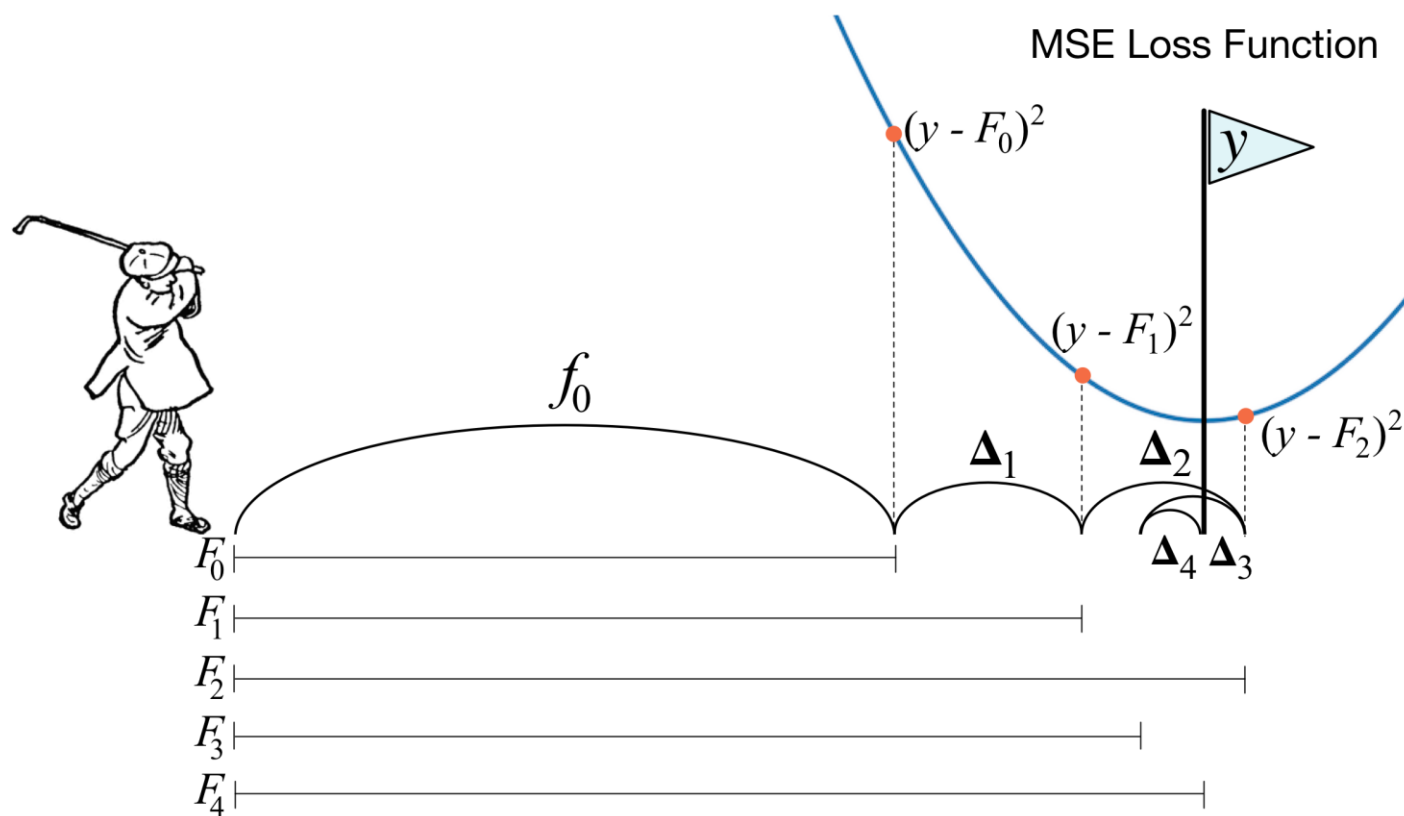
2. Fit weak learner, $h_m(x)$, dengan pseudo-residual $\{(x_1, r_{1m}), \dots, (x_n, r_{nm})\}$

3. Cari weight coefficient, β_m , dengan optimization berikut:

$$\beta_m = \operatorname{argmin}_{\beta} \sum_{i=1}^n L(y_i, F_{m-1}(x_i) + \beta h_m(x_i))$$

4. Update model : $F_m(x) = F_{m-1}(x) + \beta_m h_m(x)$

<https://explained.ai/gradient-boosting/descent.html>



MART: Multiple Additive Regression Trees (Gradient-Boosted Regression Trees)

penggabungan regresi tree and boosting

h_m di boosting diganti jadi regression tree.

Initial step: cari sebuah nilai konstan yang meminimalkan loss function.

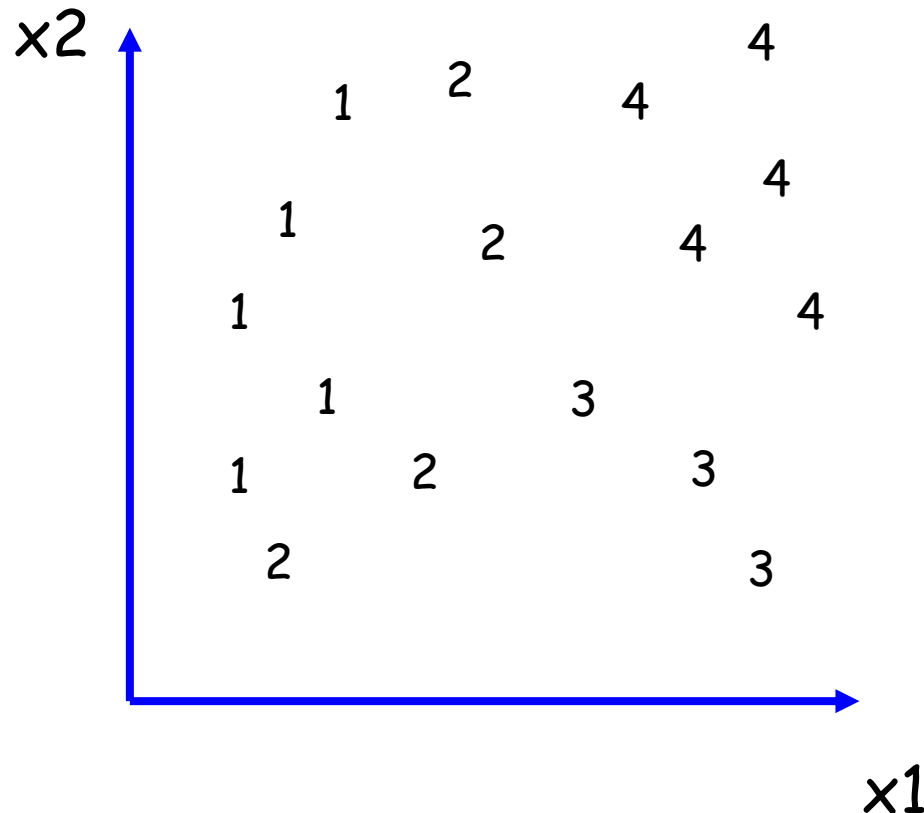
Misal, loss-function:

$$L = \sum_{i=1}^n \frac{1}{2} (y_i - f(\mathbf{x}))^2$$

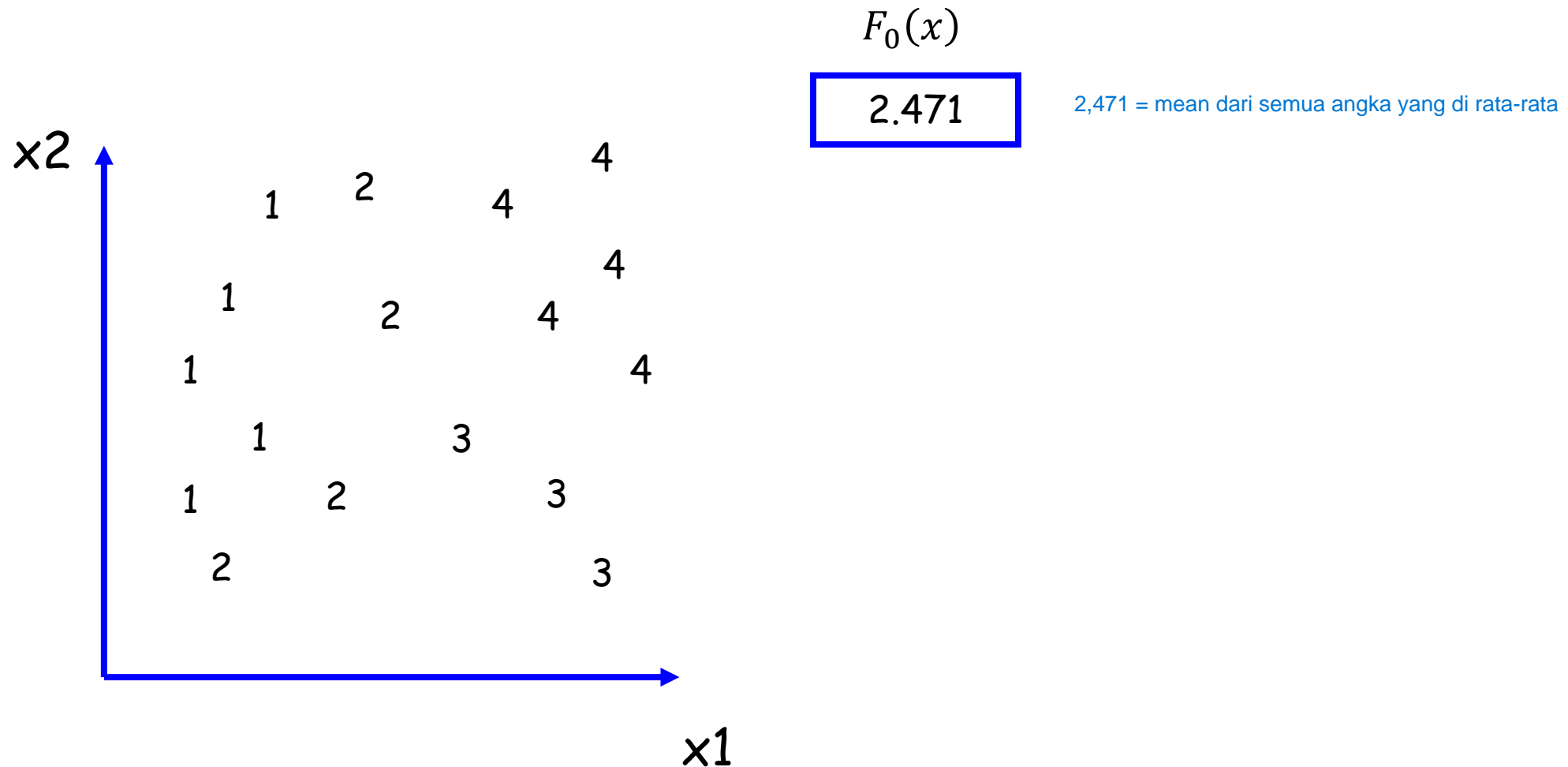
$$L = \frac{5}{2} \cdot (1 - \gamma)^2 + \frac{4}{2} \cdot (2 - \gamma)^2 + \frac{3}{2} \cdot (3 - \gamma)^2 + \frac{5}{2} \cdot (4 - \gamma)^2$$

Set $dL/dx = 0$ akan menemukan L optimum di $\gamma = 2.471$

Jadi, $f(\mathbf{x}) = \gamma = 2.471$

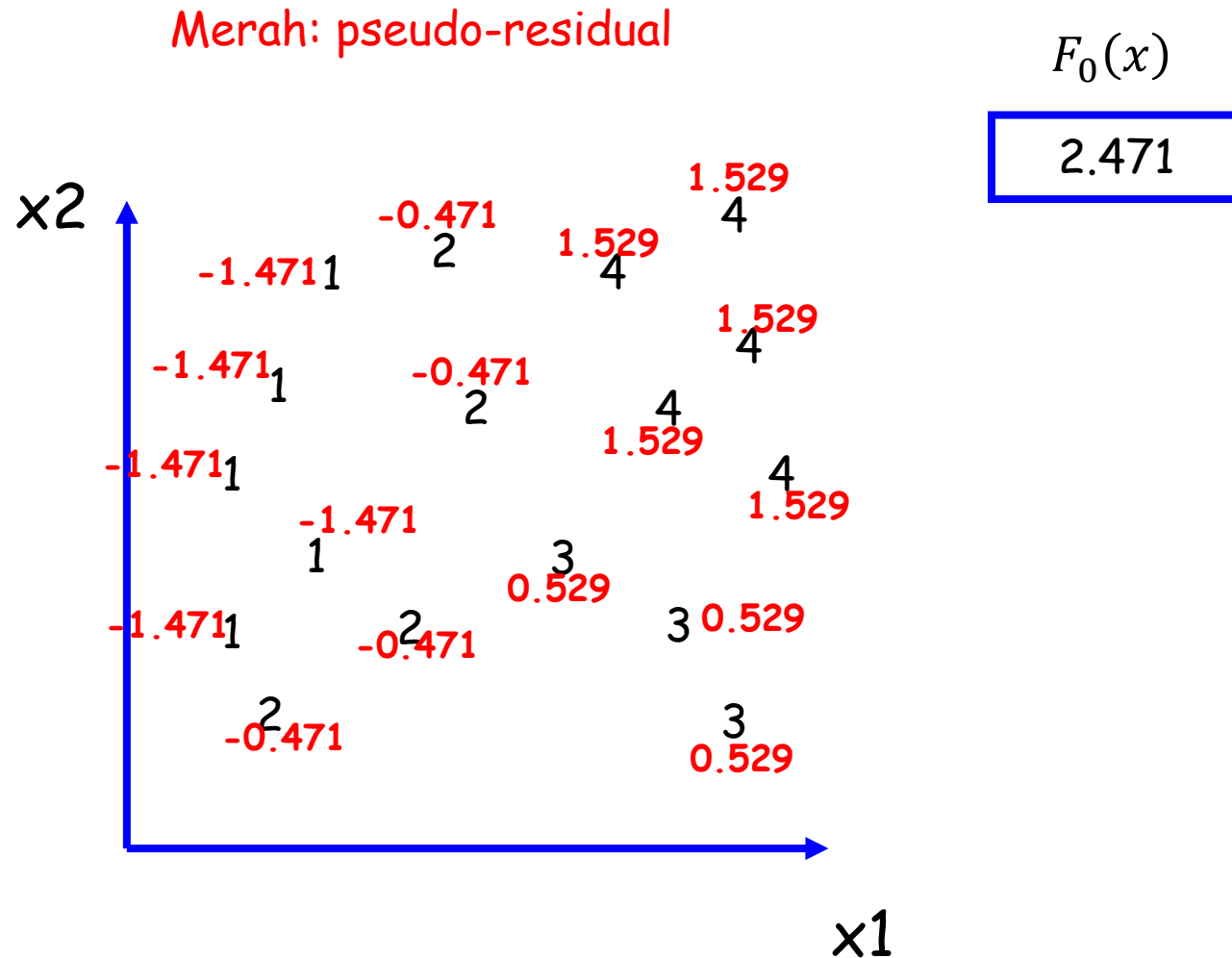


MART: Multiple Additive Regression Trees (Gradient-Boosted Regression Trees)



MART: Multiple Additive Regression Trees (Gradient-Boosted Regression Trees)

hitung negative gradient

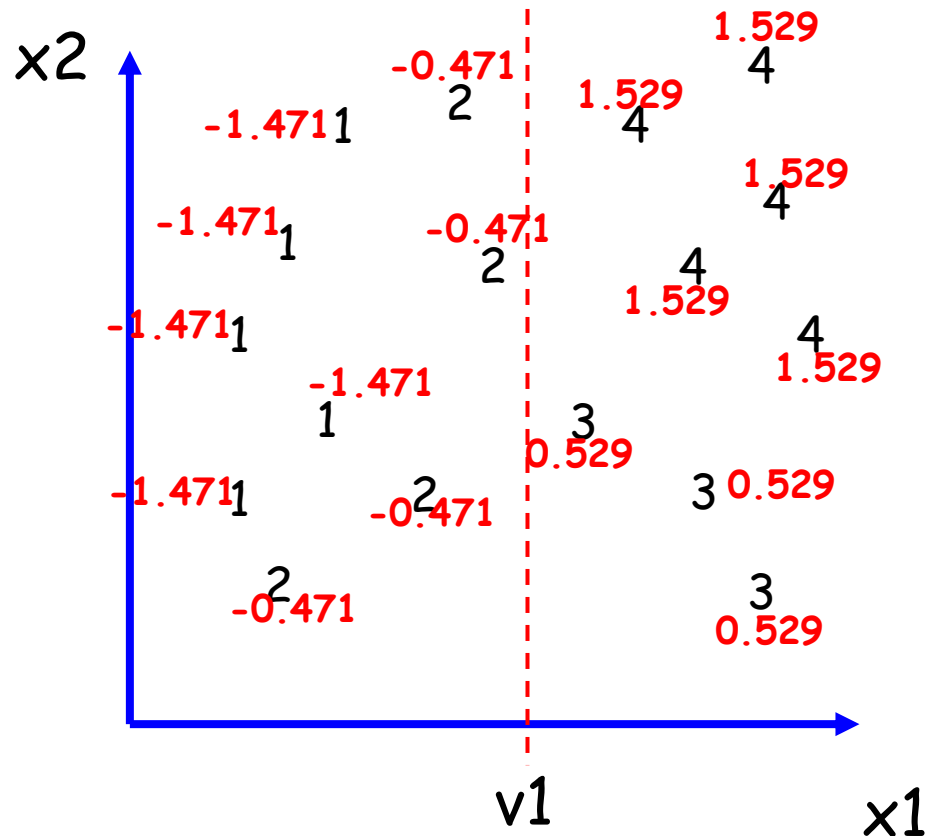


Asumsi Tree: hanya sampai depth = 1

MART: Multiple Additive Regression Trees (Gradient-Boosted Regression Trees)

Buat regression tree yang memprediksi pseudo-residual (negative gradient)

Merah: pseudo-residual



$F_0(x)$

2.471

kita memprediksi loss nya
(yang warna merah).
Jadi input

$$L = (5/2)(1 - (2.471 + x))^2 + (4/2)(2 - (2.471 + x))^2$$

$$L = (5/2)(-1.471 - x)^2 + (4/2)(-0.471 - x)^2$$

$$\frac{dL}{dx} = 0 \\ \Rightarrow x = -1.027$$

$h_1(x)$

$x_1 < v_1$

rata-rata merah di sebelah kanan

-1.027

rata-rata merah di sebelah kiri

1.154

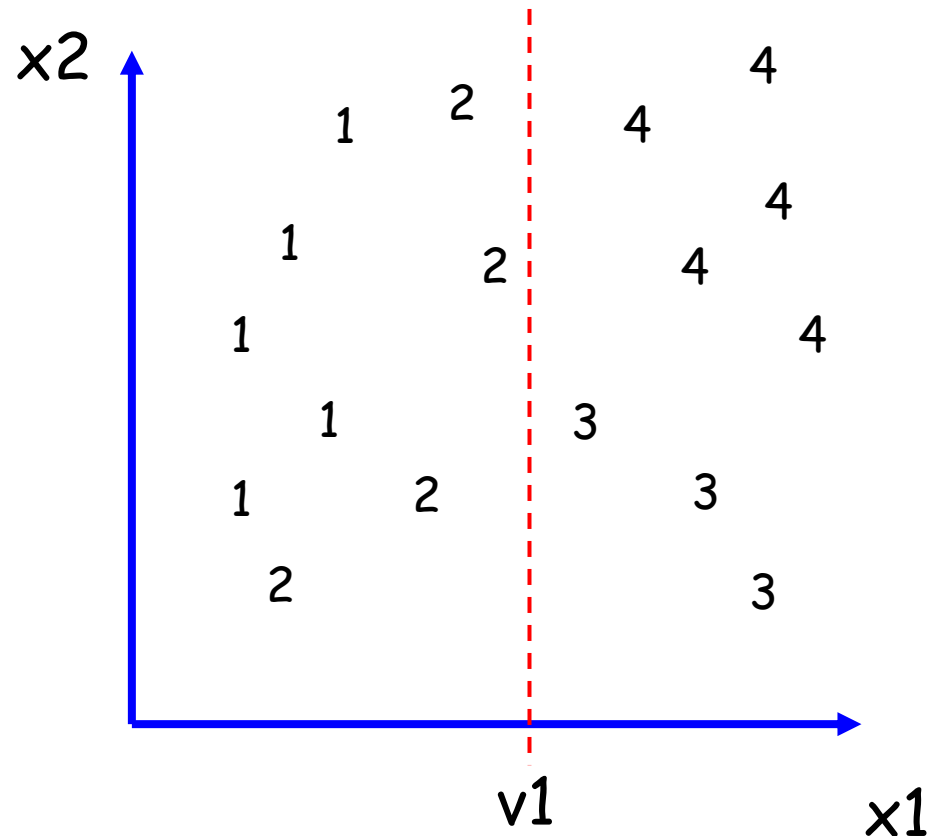
$$L = (5/2)(4 - (2.471 + x))^2 + (3/2)(3 - (2.471 + x))^2$$

$$L = (5/2)(1.529 - x)^2 + (3/2)(0.529 - x)^2$$

$$\frac{dL}{dx} = 0 \\ \Rightarrow x = 1.154$$

MART: Multiple Additive Regression Trees (Gradient-Boosted Regression Trees)

Buat regression tree yang memprediksi pseudo-residual (negative gradient)



$F_0(x)$

2.471

$h_1(x)$

$x_1 < v_1$

Y

N

-1.027

1.154

ditambahkan karena boosting ($2.471 + (-1.027)$ dan ($2.471 + 1.154$)

$$F_1(x) = F_0(x) + h_1(x)$$

$F_1(x)$

$x_1 < v_1$

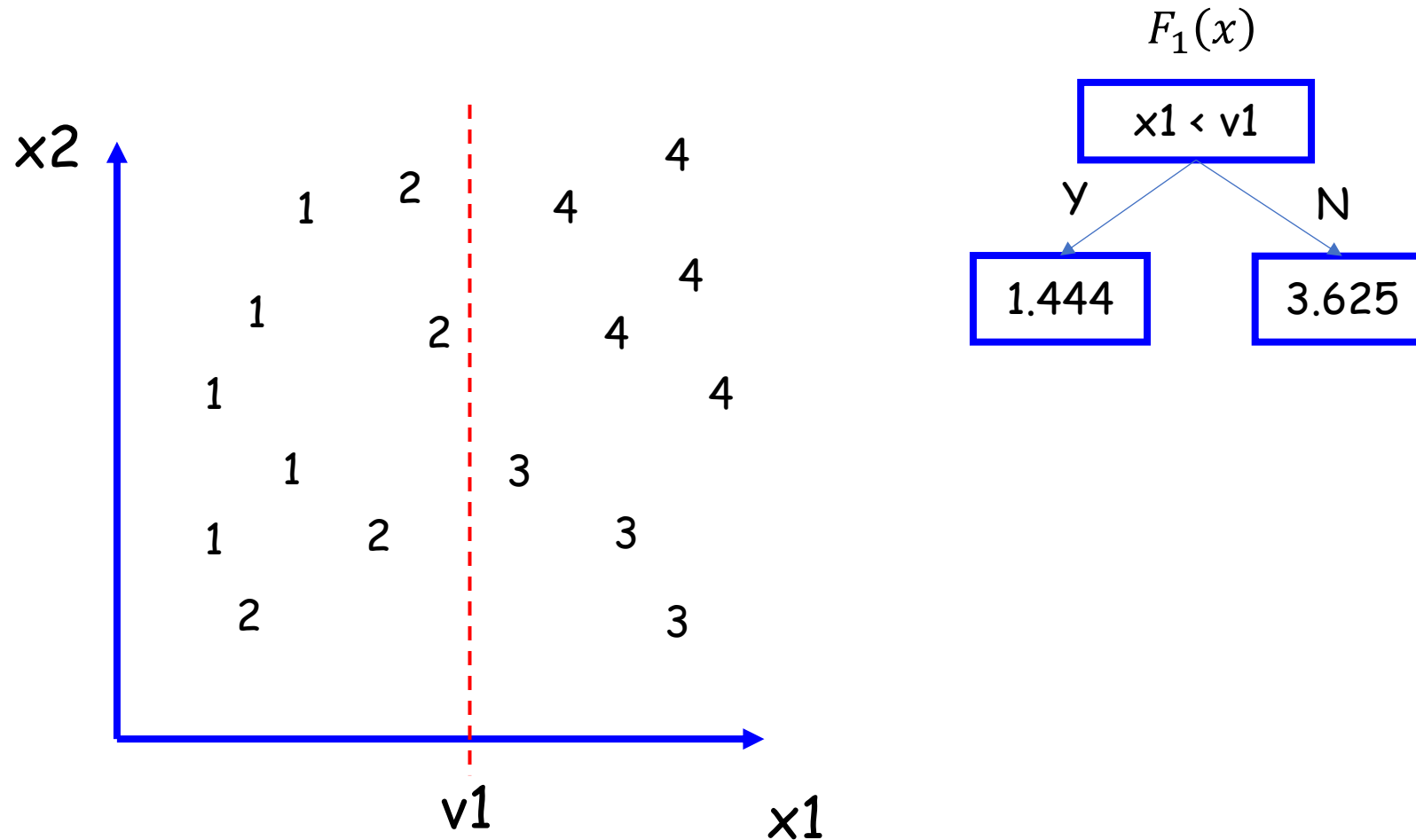
Y

N

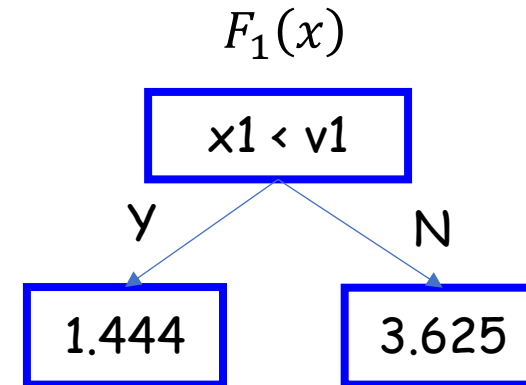
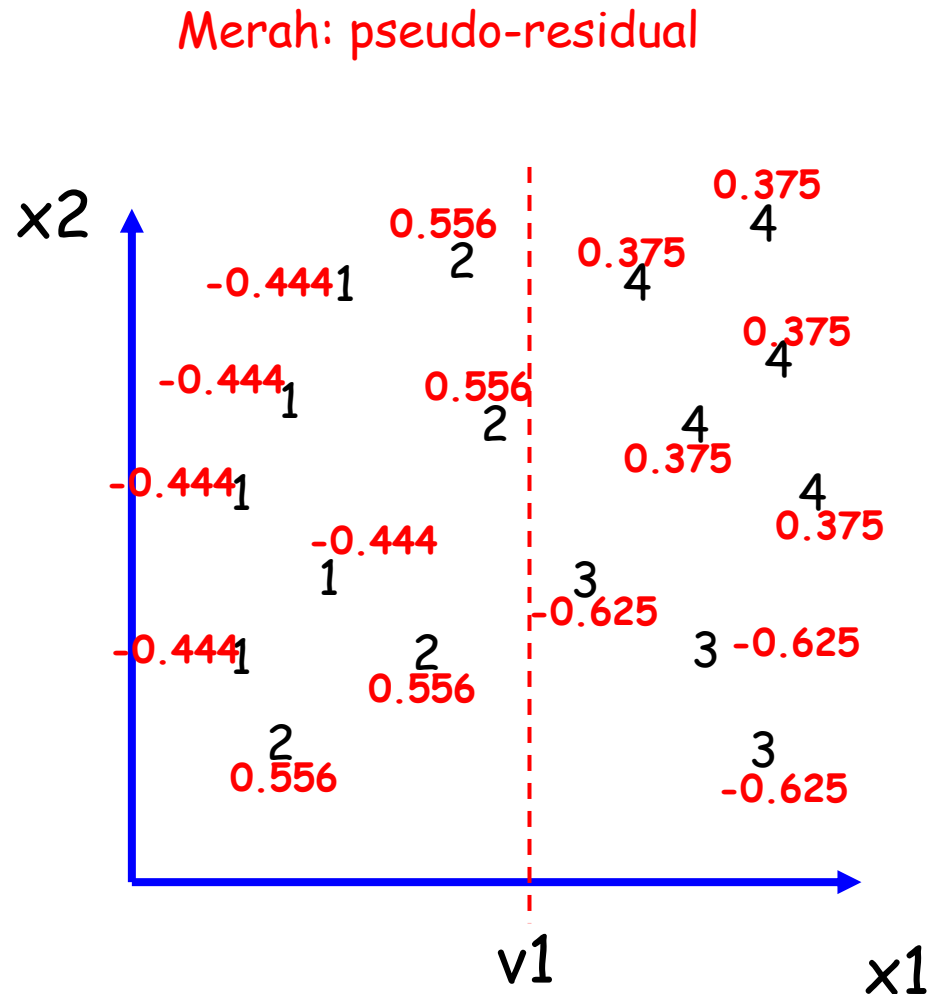
$2.471 - 1.027$
 $= 1.444$

$2.471 + 1.154$
 $= 3.625$

MART: Multiple Additive Regression Trees (Gradient-Boosted Regression Trees)

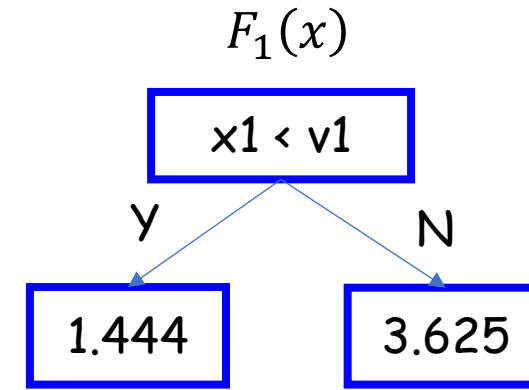
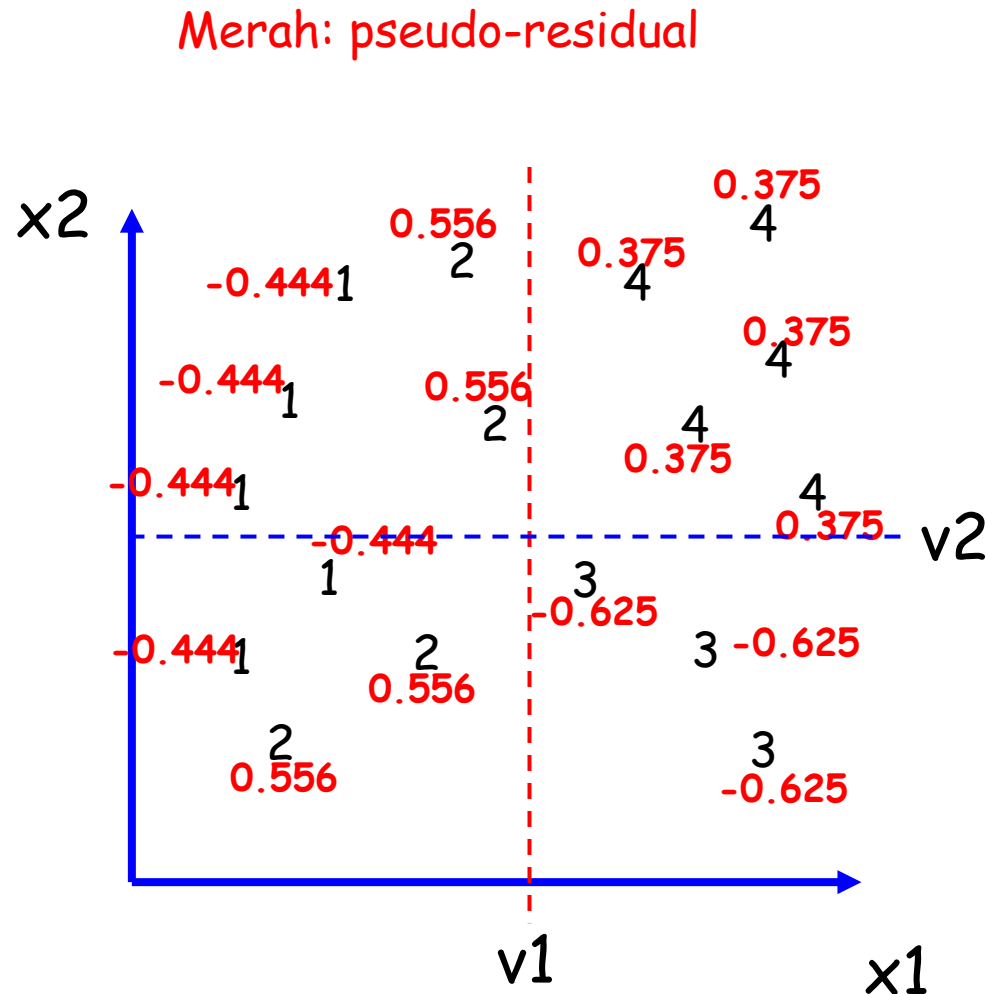


MART: Multiple Additive Regression Trees (Gradient-Boosted Regression Trees)



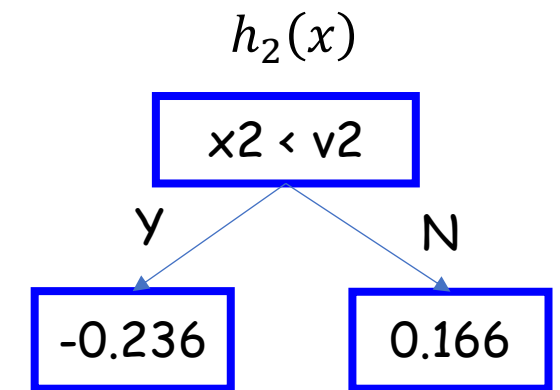
MART: Multiple Additive Regression Trees (Gradient-Boosted Regression Trees)

Buat regression tree yang memprediksi pseudo-residual (negative gradient)



$$L = (2/2)(1 - (1.444 + x))^2 + (2/2)(2 - (1.444 + x))^2 + (3/2)(3 - (3.625 + x))^2$$

$$dL/dx = 0 \\ \Rightarrow x = -0.236$$



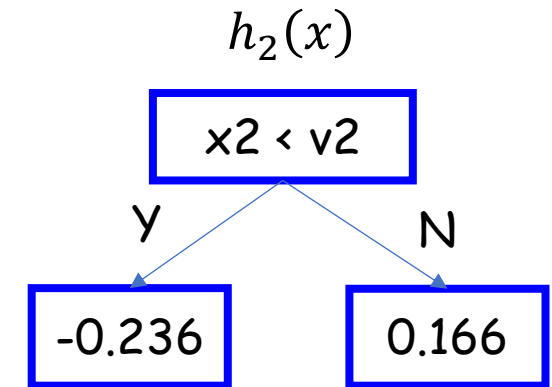
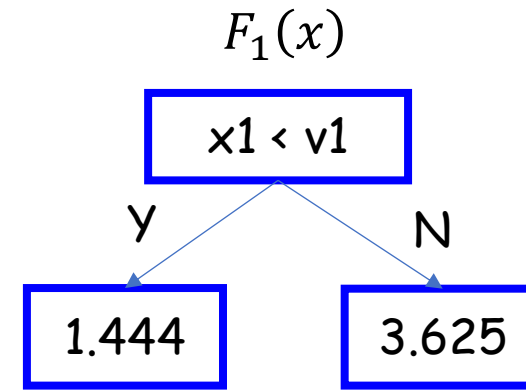
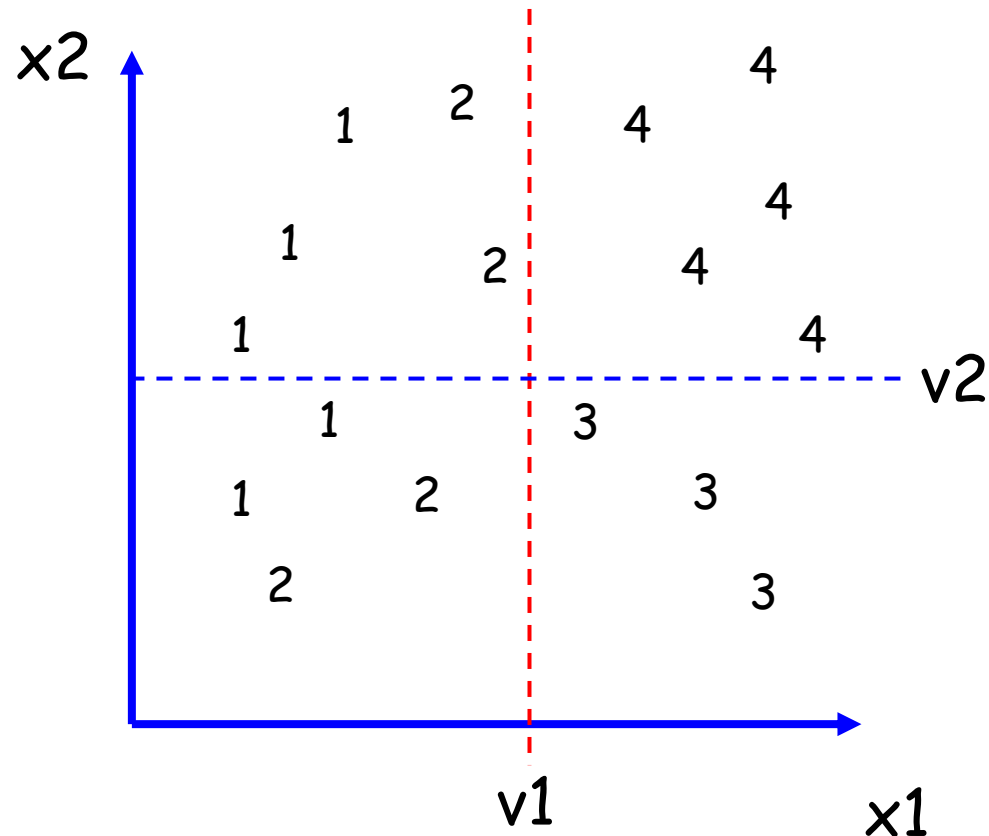
karena $x_2 < v_2$, berarti ini rata-rata yang dibawah)

$$L = (3/2)(1 - (1.444 + x))^2 + (2/2)(2 - (1.444 + x))^2 + (5/2)(4 - (3.625 + x))^2$$

$$dL/dx = 0 \\ \Rightarrow x = 0.166$$

MART: Multiple Additive Regression Trees (Gradient-Boosted Regression Trees)

Buat regression tree yang memprediksi pseudo-residual (negative gradient)



$$F_2(x) = F_1(x) + h_2(x)$$

$x1 < v1 \ \& \ x2 < v2$
 $x1 < v1 \ \& \ x2 \geq v2$
 $x1 \geq v1 \ \& \ x2 < v2$
 $x1 \geq v1 \ \& \ x2 \geq v2$

$= 1.444 - 0.236$
 $= 1.444 + 0.166$
 $= 3.625 - 0.236$
 $= 3.625 + 0.166$

MART: Multiple Additive Regression Trees (Gradient-Boosted Regression Trees)

Algorithm 1 Multiple Additive Regression Trees.

- 1: Initialize $F_0(\mathbf{x}) = \arg \min_{\gamma} \sum_{i=1}^N L(y_i, \gamma)$ argmin itu adalah mengembalikan nilai gamma yang paling kecil
 - 2: **for** $m = 1, \dots, M$ **do** bukan nilai loss paling kecil
 - 3: **for** $i = 1, \dots, N$ **do**
 - 4: $\tilde{y}_{im} = - \left[\frac{\partial L(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)} \right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})}$
 - 5: **end for**
 - 6: $\{R_{km}\}_{k=1}^{K_m}$ // Fit a regression tree to targets \tilde{y}_{im}
 - 7: **for** $k = 1, \dots, K_m$ **do**
 - 8: $\gamma_{km} = \arg \min_{\gamma} \sum_{\mathbf{x}_i \in R_{jm}} L(y_i, F_{m-1}(\mathbf{x}_i) + \gamma)$ menghitung wakil dari leaf node
 - 9: **end for**
 - 10: $F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \eta \sum_{k=1}^{K_m} \gamma_{km} 1(\mathbf{x}_i \in R_{km})$
 - 11: **end for**
 - 12: Return $F_M(\mathbf{x})$
-

MART: Multiple Additive Regression Trees (Gradient-Boosted Regression Trees)

Algorithm 1 Multiple Additive Regression Trees.

```

1: Initialize  $F_0(\mathbf{x}) = \arg \min_{\gamma} \sum_{i=1}^N L(y_i, \gamma)$ 
2: for  $m = 1, \dots, M$  do
3:   for  $i = 1, \dots, N$  do
4:      $\tilde{y}_{im} = - \left[ \frac{\partial L(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)} \right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})}$ 
5:   end for
6:    $\{R_{km}\}_{k=1}^{K_m}$  // Fit a regression tree to targets  $\tilde{y}_{im}$ 
7:   for  $k = 1, \dots, K_m$  do
8:      $\gamma_{km} = \arg \min_{\gamma} \sum_{\mathbf{x}_i \in R_{jm}} L(y_i, F_{m-1}(\mathbf{x}_i) + \gamma)$ 
9:   end for
10:   $F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \eta \sum_{k=1}^{K_m} \gamma_{km} 1(\mathbf{x}_i \in R_{km})$ 
11: end for
12: Return  $F_M(\mathbf{x})$ 

```

Kita mulai $F(\mathbf{x})$ dengan sebuah nilai konstan yang meminimalkan error.

MART: Multiple Additive Regression Trees (Gradient-Boosted Regression Trees)

Algorithm 1 Multiple Additive Regression Trees.

```

1: Initialize  $F_0(\mathbf{x}) = \arg \min_{\gamma} \sum_{i=1}^N L(y_i, \gamma)$ 
2: for  $m = 1, \dots, M$  do
3:   for  $i = 1, \dots, N$  do
4:      $\tilde{y}_{im} = - \left[ \frac{\partial L(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)} \right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})}$ 
5:   end for
6:    $\{R_{km}\}_{k=1}^{K_m}$  // Fit a regression tree to targets  $\tilde{y}_{im}$ 
7:   for  $k = 1, \dots, K_m$  do
8:      $\gamma_{km} = \arg \min_{\gamma} \sum_{\mathbf{x}_i \in R_{jm}} L(y_i, F_{m-1}(\mathbf{x}_i) + \gamma)$ 
9:   end for
10:   $F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \eta \sum_{k=1}^{K_m} \gamma_{km} 1(\mathbf{x}_i \in R_{km})$ 
11: end for
12: Return  $F_M(\mathbf{x})$ 

```

Loop sebanyak M = kita melakukan boosting terhadap fungsi $F_0(\mathbf{x})$ sebanyak M kali.

MART: Multiple Additive Regression Trees (Gradient-Boosted Regression Trees)

Algorithm 1 Multiple Additive Regression Trees.

```
1: Initialize  $F_0(\mathbf{x}) = \arg \min_{\gamma} \sum_{i=1}^N L(y_i, \gamma)$ 
2: for  $m = 1, \dots, M$  do
3:   for  $i = 1, \dots, N$  do
4:      $\tilde{y}_{im} = - \left[ \frac{\partial L(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)} \right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})}$ 
5:   end for
6:    $\{R_{km}\}_{k=1}^{K_m}$  // Fit a regression tree to targets  $\tilde{y}_{im}$ 
7:   for  $k = 1, \dots, K_m$  do
8:      $\gamma_{km} = \arg \min_{\gamma} \sum_{\mathbf{x}_i \in R_{jm}} L(y_i, F_{m-1}(\mathbf{x}_i) + \gamma)$ 
9:   end for
10:   $F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \eta \sum_{k=1}^{K_m} \gamma_{km} \mathbf{1}(\mathbf{x}_i \in R_{km})$ 
11: end for
12: Return  $F_M(\mathbf{x})$ 
```

Untuk setiap instance, hitung pseudo-residual (alias negative gradient) yang disebabkan oleh $F_{m-1}(\mathbf{x})$

MART: Multiple Additive Regression Trees (Gradient-Boosted Regression Trees)

Algorithm 1 Multiple Additive Regression Trees.

```

1: Initialize  $F_0(\mathbf{x}) = \arg \min_{\gamma} \sum_{i=1}^N L(y_i, \gamma)$ 
2: for  $m = 1, \dots, M$  do
3:   for  $i = 1, \dots, N$  do
4:      $\tilde{y}_{im} = - \left[ \frac{\partial L(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)} \right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})}$ 
5:   end for
6:    $\{R_{km}\}_{k=1}^{K_m}$  // Fit a regression tree to targets  $\tilde{y}_{im}$ 
7:   for  $k = 1, \dots, K_m$  do
8:      $\gamma_{km} = \arg \min_{\gamma} \sum_{\mathbf{x}_i \in R_{km}} L(y_i, F_{m-1}(\mathbf{x}_i) + \gamma)$ 
9:   end for
10:   $F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \eta \sum_{k=1}^{K_m} \gamma_{km} 1(\mathbf{x}_i \in R_{km})$ 
11: end for
12: Return  $F_M(\mathbf{x})$ 

```

Fit sebuah regression tree
(sebanyak K leaf nodes)
dengan pseudo-residuals.

MART: Multiple Additive Regression Trees (Gradient-Boosted Regression Trees)

Algorithm 1 Multiple Additive Regression Trees.

```

1: Initialize  $F_0(\mathbf{x}) = \arg \min_{\gamma} \sum_{i=1}^N L(y_i, \gamma)$ 
2: for  $m = 1, \dots, M$  do
3:   for  $i = 1, \dots, N$  do
4:      $\tilde{y}_{im} = - \left[ \frac{\partial L(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)} \right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})}$ 
5:   end for
6:    $\{R_{km}\}_{k=1}^{K_m}$  // Fit a regression tree to targets  $\tilde{y}_{im}$ 
7:   for  $k = 1, \dots, K_m$  do
8:      $\gamma_{km} = \arg \min_{\gamma} \sum_{\mathbf{x}_i \in R_{jm}} L(y_i, F_{m-1}(\mathbf{x}_i) + \gamma)$ 
9:   end for
10:   $F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \eta \sum_{k=1}^{K_m} \gamma_{km} \mathbf{1}(\mathbf{x}_i \in R_{km})$ 
11: end for
12: Return  $F_M(\mathbf{x})$ 

```

Untuk setiap leaf node, hitung nilai representative/wakil. Jika loss adalah squared error biasa, ini sebenarnya sama dengan menghitung mean untuk semua instance di leaf tersebut.

MART: Multiple Additive Regression Trees (Gradient-Boosted Regression Trees)

Algorithm 1 Multiple Additive Regression Trees.

```
1: Initialize  $F_0(\mathbf{x}) = \arg \min_{\gamma} \sum_{i=1}^N L(y_i, \gamma)$ 
2: for  $m = 1, \dots, M$  do
3:   for  $i = 1, \dots, N$  do
4:      $\tilde{y}_{im} = - \left[ \frac{\partial L(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)} \right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})}$ 
5:   end for
6:    $\{R_{km}\}_{k=1}^{K_m}$  // Fit a regression tree to targets  $\tilde{y}_{im}$ 
7:   for  $k = 1, \dots, K_m$  do
8:      $\gamma_{km} = \arg \min_{\gamma} \sum_{\mathbf{x}_i \in R_{jm}} L(y_i, F_{m-1}(\mathbf{x}_i) + \gamma)$ 
9:   end for
10:   $F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \eta \sum_{k=1}^{K_m} \gamma_{km} 1(\mathbf{x}_i \in R_{km})$ 
11: end for
12: Return  $F_M(\mathbf{x})$ 
```

Update model terbaru $F_m(\mathbf{x})$
yang merupakan hasil boosting
dari model $F_{m-1}(\mathbf{x})$

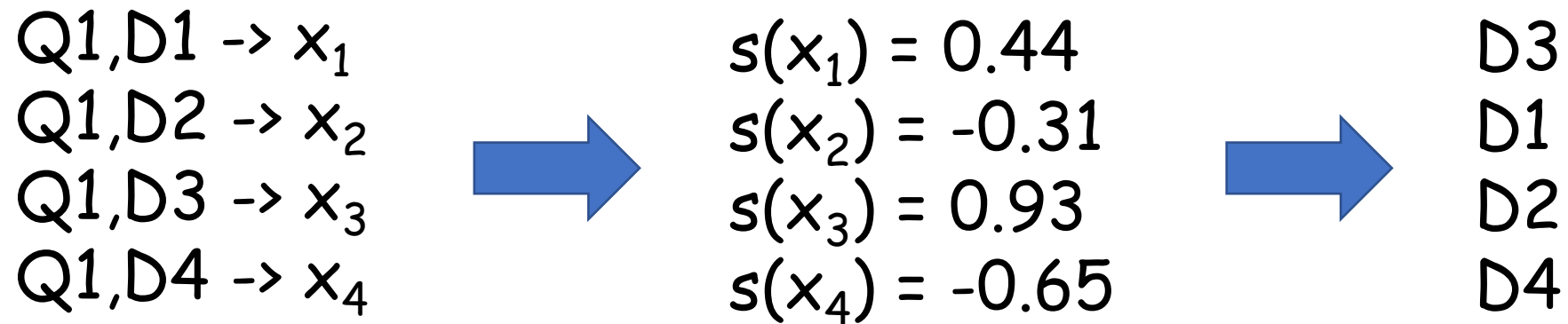
RankNet (Burges, 2010)

Ide

Misal, x adalah **representasi vektor** dari pasangan $\langle \text{query}, \text{dokumen} \rangle$.

Kita ingin mempunyai sebuah fungsi skor $s(x_i)$ yang dapat digunakan untuk ranking dokumen terhadap suatu query.

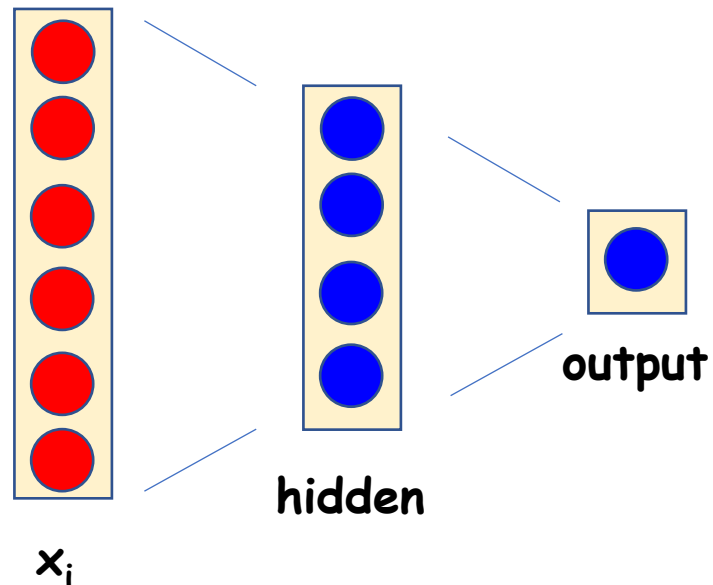
Misal:



$s(x_i)$

- Bisa berupa beberapa feed-forward neural networks dengan **output berdimensi 1**.
- Misal, $s(x_i; W_1, W_2)$ dengan 2-layer neural networks:

x_1 = pasangan <Query, document>



$$s(x; W_1, W_2) = (x^T \times W_1) \times W_2$$

Diagram illustrating the dimensions of the weights and input/output vectors in the equation:

- $(\text{num_features} \times 1)$ points to x^T
- $(\text{num_features} \times \text{num_hidden_units})$ points to W_1
- $(\text{num_hidden_units} \times 1)$ points to W_2

P_{ij} : Probabilitas bahwa doc i lebih relevan dibandingkan doc j.

RankNet dilatih secara "pairwise"

Contoh format training data:

x1: q1, (d1, d5)	$P_{1,5}$: 1
x2: q1, (d5, d9)	$P_{5,9}$: 0
x3: q1, (d4, d10)	$P_{4,10}$: 0.5
x4: q2, (d2, d18)	$P_{2,18}$: 0
x5: q2, (d13, d18)	$P_{13,18}$: 1
...	...

Actual probability = P_{ij} , 1 jika doc i lebih relevan dari doc j; 0 jika sebaliknya; dan 0.5 jika sama relevansinya.

p kalo gak ada topi, itu artinya true probability (yang di anotasi oleh manusia)

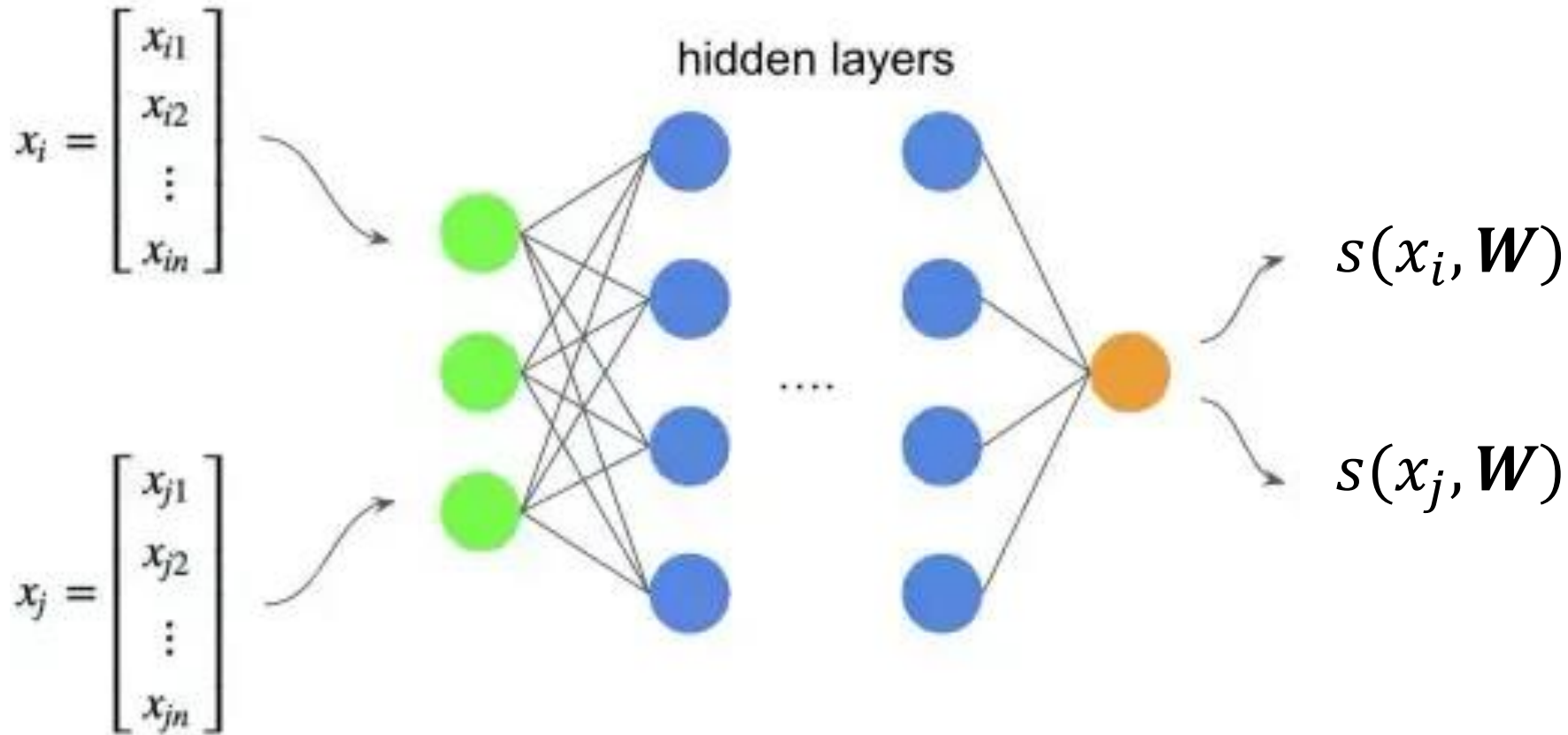
$$\hat{P}_{i,j} = \frac{1}{1 + \exp(-[s(x_i, \mathbf{W}) - s(x_j, \mathbf{W})])}$$

Model probability

Loss function -> binary cross entropy:

$$L = -P_{i,j} \log(\hat{P}_{i,j}) - (1 - P_{i,j}) \log(1 - \hat{P}_{i,j})$$

RankNet dilatih secara "pairwise"



RankNet dilatih secara "pairwise"

$$\begin{aligned} L &= -P_{i,j} \log(\hat{P}_{i,j}) - (1 - P_{i,j}) \log(1 - \hat{P}_{i,j}) \\ &= -P_{i,j} \left(s(x_i) - s(x_j) \right) + \log \left(1 + \exp \left(s(x_i) - s(x_j) \right) \right) \end{aligned}$$

Dapat ditunjukkan bahwa:

turunan loss terhadap score document i

$$\frac{\partial L}{\partial s(x_i)} = - \frac{\partial L}{\partial s(x_j)}$$

turunan loss terhadap score document j

$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial s(x_i)} \frac{\partial s(x_i)}{\partial W} + \frac{\partial L}{\partial s(x_j)} \frac{\partial s(x_j)}{\partial W}$$

Berlanjut ke halaman berikutnya ...

RankNet's "Lambda" λ_{ij}

The desired change of scores for the pair of doc i and doc j

$$\frac{\partial L}{\partial s(x_i)} = -\frac{\partial L}{\partial s(x_j)} = \frac{\exp(s(x_i) - s(x_j))}{1 + \exp(s(x_i) - s(x_j))} - P_{ij} = \hat{P}_{ij} - P_{ij} = \lambda_{ij}$$

predicted prob - actual prob

$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial s(x_i)} \frac{\partial s(x_i)}{\partial W} - \frac{\partial L}{\partial s(x_j)} \frac{\partial s(x_j)}{\partial W}$$

$$= \frac{\partial L}{\partial s(x_i)} \left(\frac{\partial s(x_i)}{\partial W} - \frac{\partial s(x_j)}{\partial W} \right)$$

$$= \lambda_{ij} \left(\frac{\partial s(x_i)}{\partial W} - \frac{\partial s(x_j)}{\partial W} \right)$$

turunan loss terhadap w, proporsional terhadap lambda.

Atau bisa anggap bahwa lambda itu part of gradient

Predicted Prob - True Prob

Untuk update parameter **W** dalam sekali loop gradient descent, perlu menghitung nilai **lambda** ini.

RankNet's "Lambda" λ_{ij}

lamba proposional dengan loss terhadap Weight

λ_{ij}

ekspektasinya 3 biru diatas

2 merah diatas

Untuk sebuah pasangan doc i dan doc j, λ_{ij} merepresentasikan laju perubahan pada loss/error yang disebabkan oleh perubahan pada skor untuk doc i (kontribusi ke error yang disebabkan $s(x_i)$) atau doc j (untuk arah berlawanan)

	D23
	D11
	D52
	D34
	D12

Misal, gold standard mengatakan bahwa D12 lebih relevan dibandingkan D11, yaitu $P_{12,11} = 1$ atau $P_{11,12} = 0$.

Namun model mengatakan $\hat{p}_{12,11} = 0.2$ atau $\hat{p}_{11,12} = 0.8$

$$\lambda_{12,11} = -0.8$$

Artinya, loss akan turun, jika skor D12 "dinaikkan" (ranking D12 "dinaikkan ke atas")

Karena arah loss itu berarah belakang artinya kalau D12 nya itu naik, maka loss nya turun

Training RankNet

Update Parameter dengan Stochastic Gradient Descent

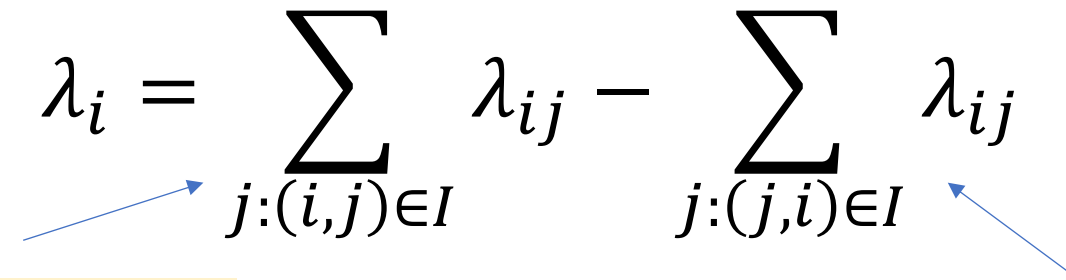
Untuk setiap pasangan **doc i** dan **doc j**:

$$W := W - \alpha \cdot \lambda_{ij} \left(\frac{\partial s(x_i)}{\partial W} - \frac{\partial s(x_j)}{\partial W} \right)$$

$$\lambda_i = \text{Gradient}$$

"Lambda" w.r.t doc i $\rightarrow \lambda_i$

- λ_i : Total kontribusi doc i terhadap error/loss
- Misal, I adalah himpunan pasangan (i, j) dimana doc i **lebih relevan** dibandingkan doc j.

$$\lambda_i = \sum_{j:(i,j) \in I} \lambda_{ij} - \sum_{j:(j,i) \in I} \lambda_{ij}$$


Total ketika doc i **lebih relevan** dibandingkan pasangannya.

Total ketika doc i **kurang relevan** dibandingkan pasangannya.

"Lambda" w.r.t doc $i \rightarrow \lambda_i$

(a) is the perfect ranking, (b) is a ranking with 10 pairwise errors, (c) is a ranking with 8 pairwise errors. Each blue arrow represents the λ_i for each query-document vector x_i

memaknai lambda i:



(a)

harapan ideal



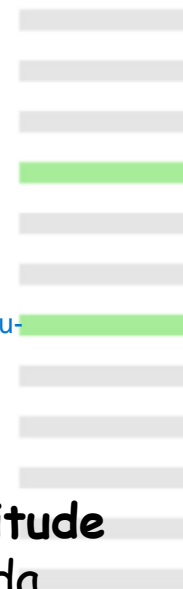
(b)

ada 10 pairwise error

(ada 10 kesalahan urutan) (bar abu-abu ini)



Magnitude
Lambda
besar



(c)



Magnitude
Lambda kecil



Magnitude
Lambda sedang

lambda_i artinya sudah terasosiasi dengan dokumen_i

Inversion / pairwise error:
sebuah pasangan "tak terurut"

Prinsip training RankNet adalah meminimalkan *the number of inversions* pada ranking.

LambdaRank

Problem with RankNet's Lambda

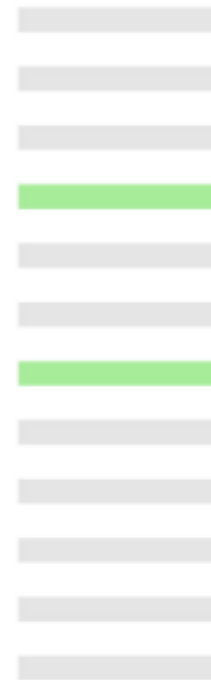
Problem: RankNet is based on pairwise error, while modern IR measures emphasize higher ranking positions. Red arrows show better λ 's for modern IR, esp. web search.



(a)



(b)



(c)









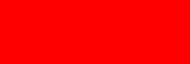

Ingat di kuliah topik IR evaluation, bahwa **user lebih sering melihat top rank positions dibandingkan yang di bawah**. Jadi, perubahan di top rank positions seharusnya mempunyai "bobot lebih".

lambdarank itu yang merah

ranknet yang biru

Dari Slide Chris Manning & Pandu Nayak, Learning-to-Rank, Kuliah Web Search & IR, Stanford U.

Problem with RankNet's Lambda

Rank 1		D23
Rank 2		D11
Rank 3		D52
Rank 4		D34
Rank 5		D12
Rank 6		D70
Rank 7		D91
⋮		
Rank 99		D53
Rank 100		D65

Misal, $\hat{P}_{52,11} = \hat{P}_{65,53}$.

Dengan informasi ini, RankNet melihat bahwa:

$$\lambda_{52,11} = \lambda_{65,53}$$

Menurut Anda apakah hal ini "make sense"?

Normalized DCG (NDCG)

$$NDCG@K = \frac{DCG@K}{IDCG@K}$$

DCG@K dibagi dengan DCG@K **ketika "ranking ideal"**

Contoh:

sebuah ranking $r = [0, 1, 0, 1, 1]$,

dengan asumsi hanya ada 3 relevant documents di koleksi.

$$r_{\text{ideal}} = [1, 1, 1, 0, 0]$$

$$DCG@5(r) = \frac{0}{\log_2(2)} + \frac{1}{\log_2(3)} + \frac{0}{\log_2(4)} + \frac{1}{\log_2(5)} + \frac{1}{\log_2(6)} = 1.45$$

$$IDCG@5(r) = \frac{1}{\log_2(2)} + \frac{1}{\log_2(3)} + \frac{1}{\log_2(4)} + \frac{0}{\log_2(5)} + \frac{0}{\log_2(6)} = 2.13$$

$$NDCG@5(r) = \frac{1.45}{2.13} = 0.68$$

$\Delta NDCG$ ketika dua dokumen di-swap

$$r = [0, 1, 0, 1, 1] \quad DCG = 1.45 \quad NDCG = 0.68$$

$$r = [1, 0, 0, 1, 1] \quad DCG = 1.82 \quad NDCG = 0.85$$

$$\Delta NDCG = 0.17$$

yang warna biru di flip, (yang atas: sebelumnya), yang baru: setelahnya yang baru.

$$r = [0, 1, 0, 1, 1] \quad DCG = 1.45 \quad NDCG = 0.68$$

$$r = [0, 1, 1, 0, 1] \quad DCG = 1.52 \quad NDCG = 0.71$$

$$\Delta NDCG = 0.03$$

nanti $F * \Delta NDCG$

Semuanya mempunyai ranking ideal yang sama:

$$r_{ideal} = [1, 1, 1, 0, 0] \quad DCG = 2.13$$

$$IDCG@5(r) = \frac{1}{\log_2(2)} + \frac{1}{\log_2(3)} + \frac{1}{\log_2(4)} + \frac{0}{\log_2(5)} + \frac{0}{\log_2(6)} = 2.13$$

Artinya, swap dua dokumen di posisi rank tinggi lebih memberikan dampak

Lambda "RankNet" vs Lambda "LambdaRank"

Lambda-nya RankNet

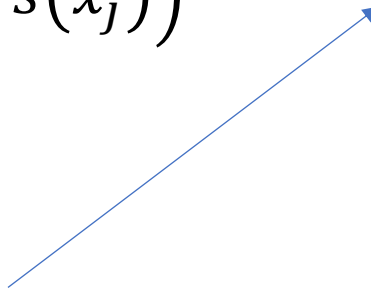
$$\lambda_{ij} = \frac{\exp(s(x_i) - s(x_j))}{1 + \exp(s(x_i) - s(x_j))} - P_{ij} = \hat{P}_{ij} - P_{ij}$$

Lambda pada RankNet **tidak peduli** dengan posisi ranking!
Lambda pasangan dok rank 1 & 2 bisa saja **sama dengan**
Lambda pasangan dok rank 500 & 501.

Lambda "RankNet" vs Lambda "LambdaRank"

Lambda-nya LambdaRank

ditambahin ini biar, dia positionally aware, karena kalau lambda biasa, dia gak aware terhadap posisi-posisi kata-katanya.

$$\lambda_{ij} = \frac{-1}{1 + \exp(s(x_i) - s(x_j))} \cdot |\Delta NDCG_{ij}|$$


Agar "positionally aware", dikali nilai $\Delta NDCG$ ketika rank position doc i dan doc j di-swap

Lambda "RankNet" vs Lambda "LambdaRank"

Lambda-nya LambdaRank

METRIC yang penting jangan statis kayak precision biasa. Average precision masih boleh

$$\lambda_{ij} = \frac{-1}{1 + \exp(s(x_i) - s(x_j))} \cdot |\Delta METRIC_{ij}|$$

Bisa diperumum dengan Top-Weighted metric lain seperti RBP dan Average Precision (AP).

Training LambdaRank

Sama saja seperti RankNet, namun Lambda sudah berubah!

Untuk setiap pasangan **doc i** dan **doc j**:

$$W := W - \alpha \cdot \lambda_{ij} \left(\frac{\partial s(x_i)}{\partial W} - \frac{\partial s(x_j)}{\partial W} \right)$$

LambdaMART

"Lambda-Boosted Regression Trees"

Yang diambil dari LambdaRank & MART

- P_{ij} pada RankNet dan LambdaRank sebenarnya dilatih dengan gaya “binary classification” via “logistic regression”.
 - 1 jika doc i lebih relevan dibandingkan doc j
 - 0 jika sebaliknya.
- Kita perlu cari loss function yang cocok untuk binary classification.

Lambda Mart

Dari

Mart (Boosting), tinggal ubah gradien jadi lambda

Yang diambil dari LambdaRank & MART

Singkat cerita, Burges et al., menggunakan loss function berikut untuk MART jika digunakan untuk binary classification:

$$L(y_i, F(x_i)) = \log(1 + \exp(-2y_i F(x_i)))$$

Dengan catatan $y_i \in \{-1, +1\}$ dan **BUKAN** $\{0, 1\}$

Yang diambil dari LambdaRank & MART

Sehingga **negative gradient** dari loss function tersebut w.r.t $F_{m-1}(\mathbf{x})$ adalah:

$$-\frac{\partial L(y_i, F_{m-1}(x_i))}{\partial F_{m-1}(x_i)} = \frac{2 \cdot y_i}{1 + \exp(2 \cdot y_i \cdot F_{m-1}(x_i))}$$

Yang diambil dari LambdaRank & MART

Sehingga **negative gradient** dari loss function tersebut w.r.t $F_{m-1}(\mathbf{x})$ adalah:

$$-\frac{\partial L(y_i, F_{m-1}(x_i))}{\partial F_{m-1}(x_i)} = \frac{2 \cdot y_i}{1 + \exp(2 \cdot y_i \cdot F_{m-1}(x_i))}$$



Sekilas mirip **Lambda** di **LambdaRank**.

Yang diambil dari LambdaRank & MART

Tetapi tetap saja, pada LambdaMART, **pseudo-residuals (negative gradient)** yang digunakan adalah:

$$\lambda_i = \sum_{j:(i,j) \in I} \lambda_{ij} - \sum_{j:(j,i) \in I} \lambda_{ij}$$

dimana
$$\lambda_{ij} = \frac{-1}{1 + \exp(s(x_i) - s(x_j))} \cdot |\Delta METRIC_{ij}|$$

Metric bisa diganti dengan NDCG, DCG, RBP, AP, atau yang lainnya.

Gradient Descent vs Newton's Method

Daripada menggunakan **Gradient Descent Step** (seperti MART biasa), Burges et al. memilih untuk menggunakan **Newton's Step**.

biar lebih cepet menuju goal

cuman lama komputasinya

Newton's Step membutuhkan perhitungan **hessian**.

Gradient Descent

$$\theta_{t+1} = \theta_t - \alpha \nabla L(y, f(x|\theta_t))$$

Newton's Method

$$\theta_{t+1} = \theta_t - \alpha \frac{\nabla L(y, f(x|\theta_t))}{\nabla^2 L(y, f(x|\theta_t))}$$

LambdaRank + MART

Algorithm: LambdaMART

set number of trees N , number of training samples m , number of leaves per tree L ,
learning rate η

for $i = 0$ to m **do**

$F_0(x_i) = \text{BaseModel}(x_i)$ //If BaseModel is empty, set $F_0(x_i) = 0$

end for

for $k = 1$ to N **do**

for $i = 0$ to m **do**

$y_i = \lambda_i$

$w_i = \frac{\partial y_i}{\partial F_{k-1}(x_i)}$

Hessian dari loss function atau
turunan dari Lambda

end for

$\{R_{lk}\}_{l=1}^L$ // Create L leaf tree on $\{x_i, y_i\}_{i=1}^m$

$\gamma_{lk} = \frac{\sum_{x_i \in R_{lk}} y_i}{\sum_{x_i \in R_{lk}} w_i}$ // Assign leaf values based on Newton step. sum gradien/sum hessian

$F_k(x_i) = F_{k-1}(x_i) + \eta \sum_l \gamma_{lk} I(x_i \in R_{lk})$ // Take step with learning rate η .

end for