Learning-To-Rank

Alfan F. Wicaksono

Learning-To-Rank (LTR / LETOR)

Machine learning models to solve a ranking problem.

Given a query q and top-n documents (retrieved by TF-IDF or BM25) $D = d_1, d_2, \ldots, d_n$, 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:

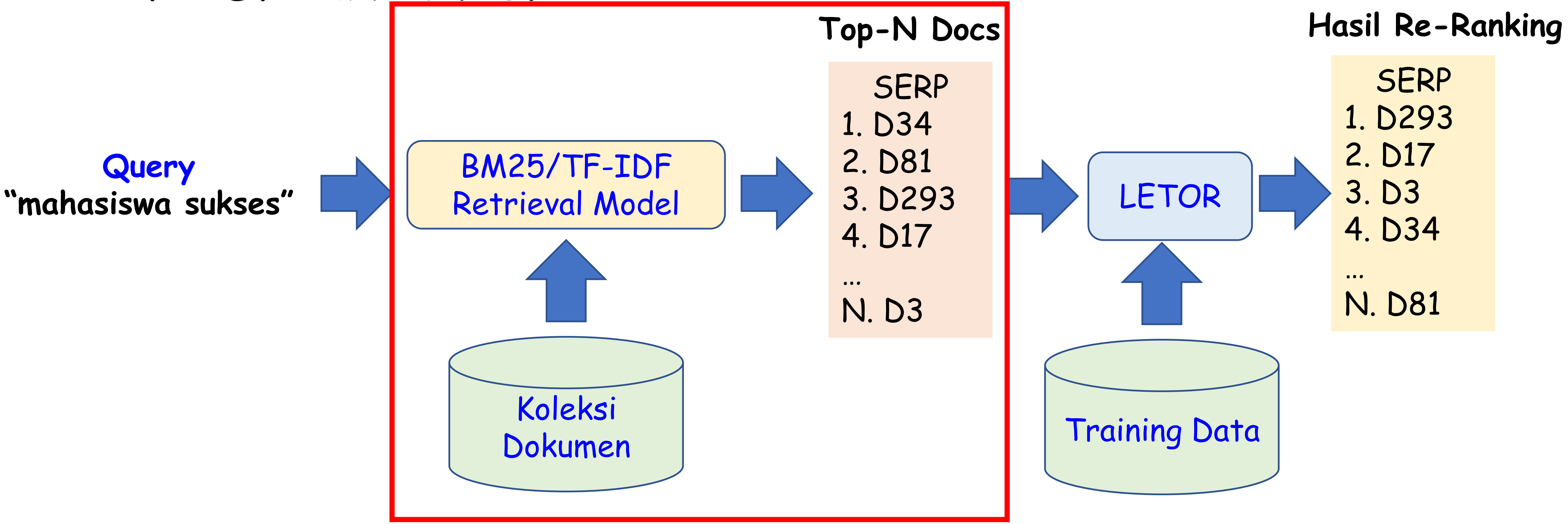
- Log frequency of query word
- # images on page
- # out links on page
- # clicks on page
- # query reformulation
- "Scroll-down" & "scroll-up" actions
- Dwell time

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

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



SERP
1. D34
2. D81
3. D293
4. D17
…
N. D3

SERP
1. D293
2. D17
3. D3
4. D34
…
N. D81

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**.

x1: q1, d1	Y1: 3 (fully relevant)
x2: q1, d5	Y2: 1 (marginally relevant)
x3: q1, d9	Y3: 2 (somewhat relevant)
x4: q2, d2	Y4: 1
x5: q2, d13	Y5: 3
x6: q2, d18	y6: 2
…	…

Diberikan q dan sekumpulan dokumen d1, d2, …, fungsi **f(q, di)** 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)**.

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)
…	…

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

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

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:

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

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

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

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

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?

$$v_m = argmin_v G(v)$$

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

Binary Regression Trees

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

$$w = argmin_w \sum_{x_i \in leaf} L(y_i, w)$$

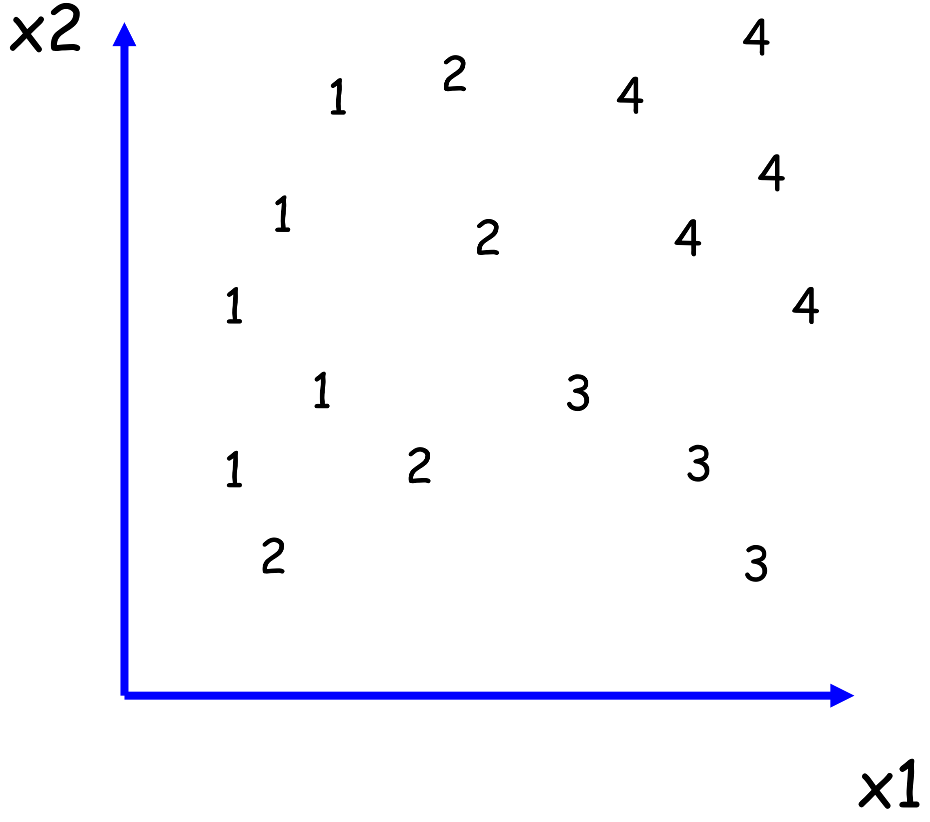
- Dapat dibuktikan bahwa jika **L** adalah MSE (mean squared error), predicted value di sebuah leaf/terminal node adalah **mean**.

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

Training Regression Tree

Loss-function = MSE

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

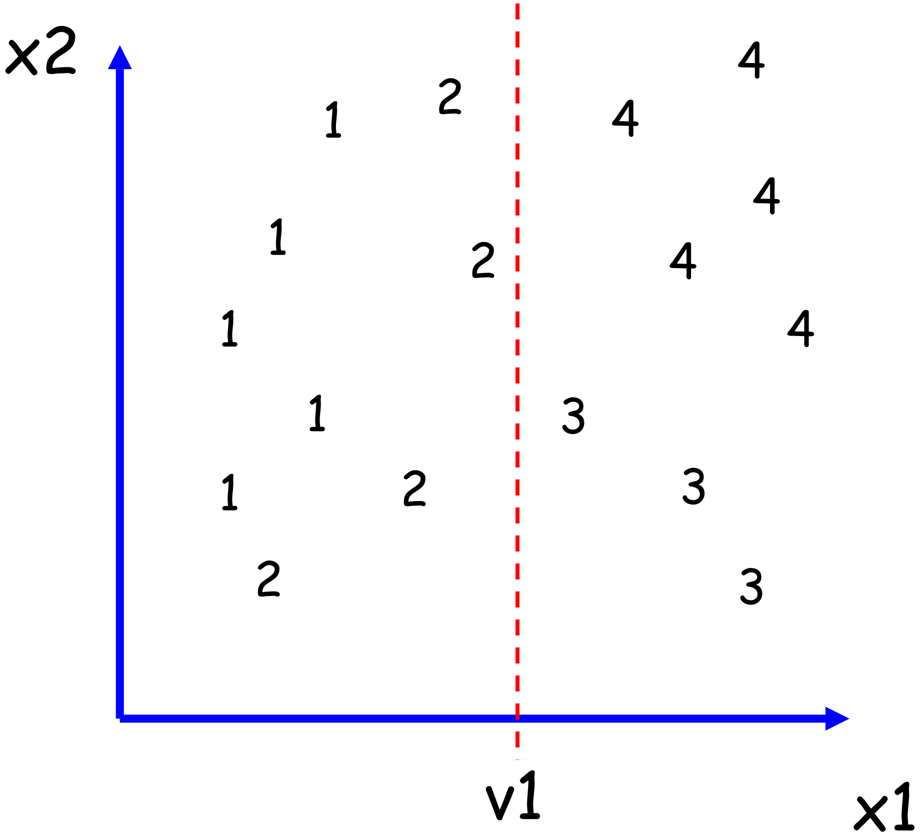


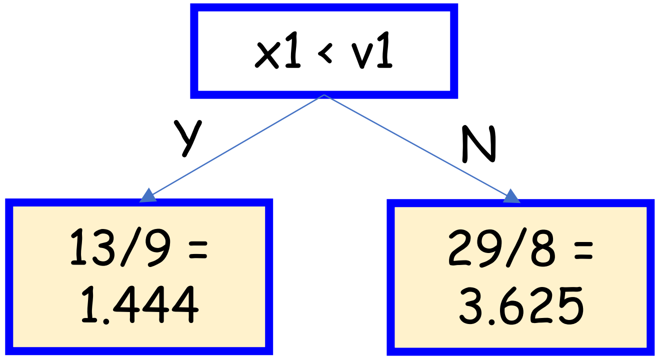
Misal, decision tree regressor dalam bentuk **f(x1, x2)** dan kita menerapkan binary splitting.

Training Regression Tree

Loss-function = MSE

Misal, pada iterasi pertama, **split variable x1** dengan **split value v1** memberikan predicted error paling minimal.





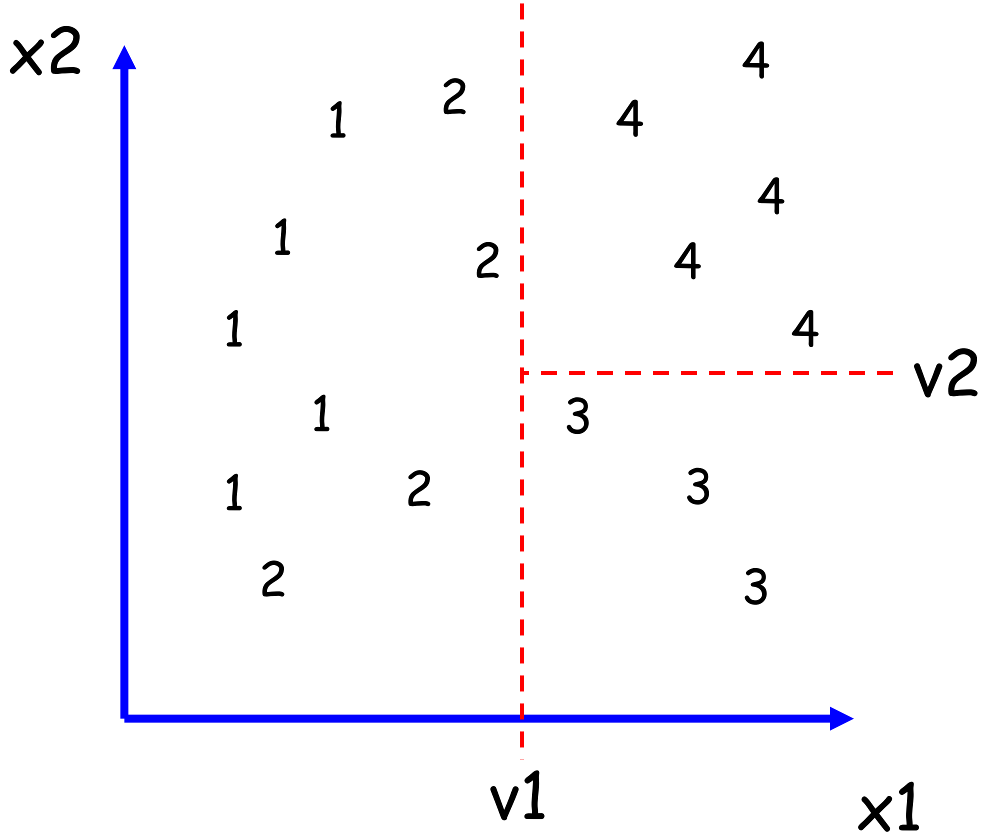
$$G(v_1) = \frac{9}{17}\left[\frac{5}{9}.\frac{1}{2}.(1-1.444)^2 + \frac{4}{9}.\frac{1}{2}.(2-1.444)^2\right] + \frac{8}{17}\left[\frac{3}{8}.\frac{1}{2}.(3-3.625)^2 + \frac{5}{8}.\frac{1}{2}.(4-3.625)^2\right] = 0.12$$

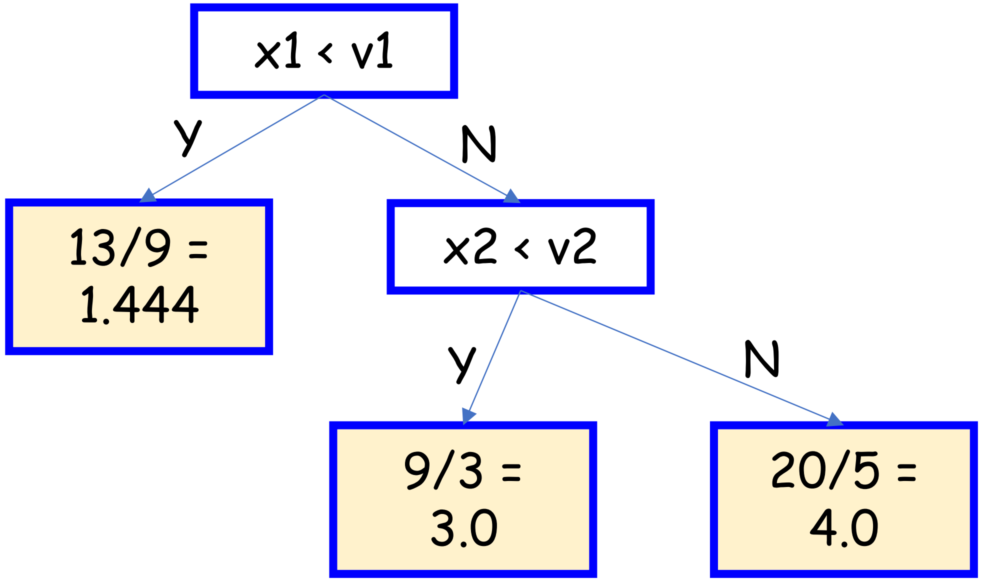
Latihan: coba hitung G untuk split value lain; dan bahkan untuk split variable x2! Apakah ada yang < 0.12?

Training Regression Tree

Loss-function = MSE

Misal, pada iterasi pertama, **split variable x1** dengan **split value v1** memberikan predicted error paling minimal.

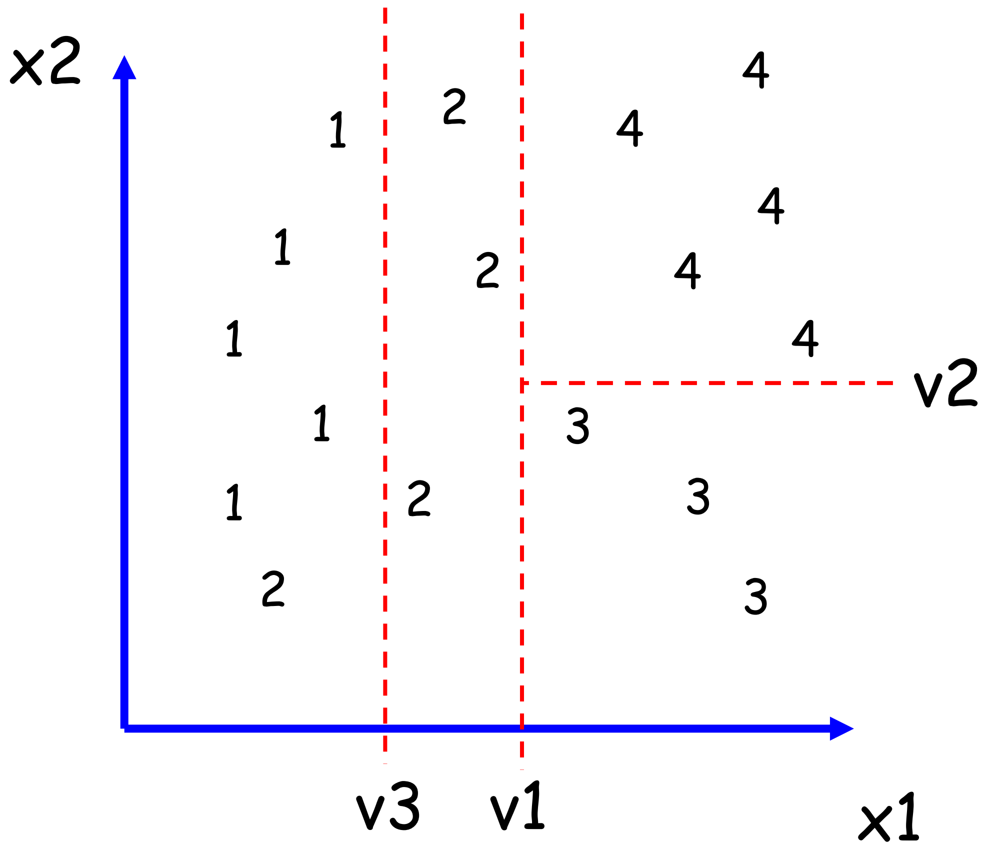


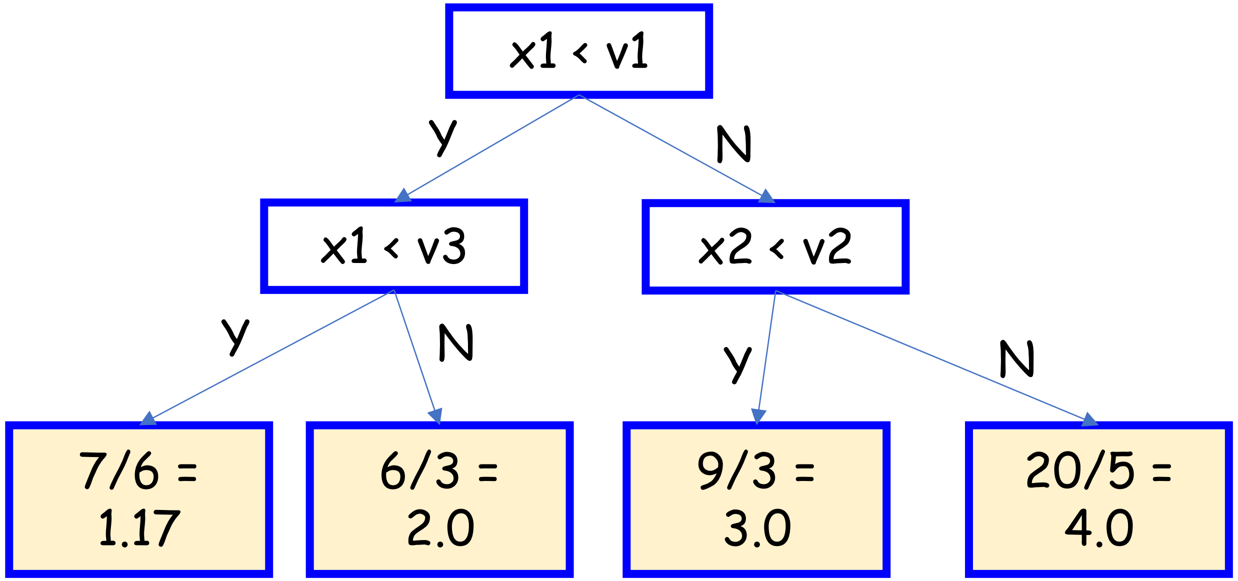


Training Regression Tree

Loss-function = MSE

Misal, pada iterasi pertama, **split variable x1** dengan **split value v1** memberikan predicted error paling minimal.

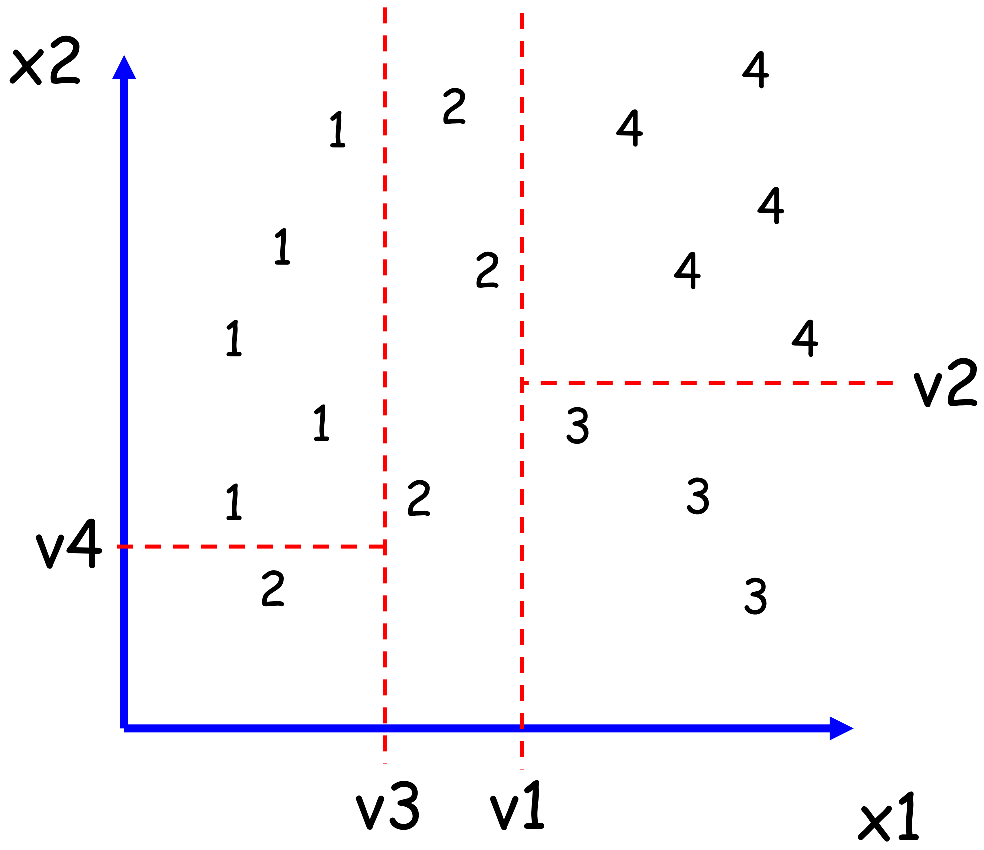


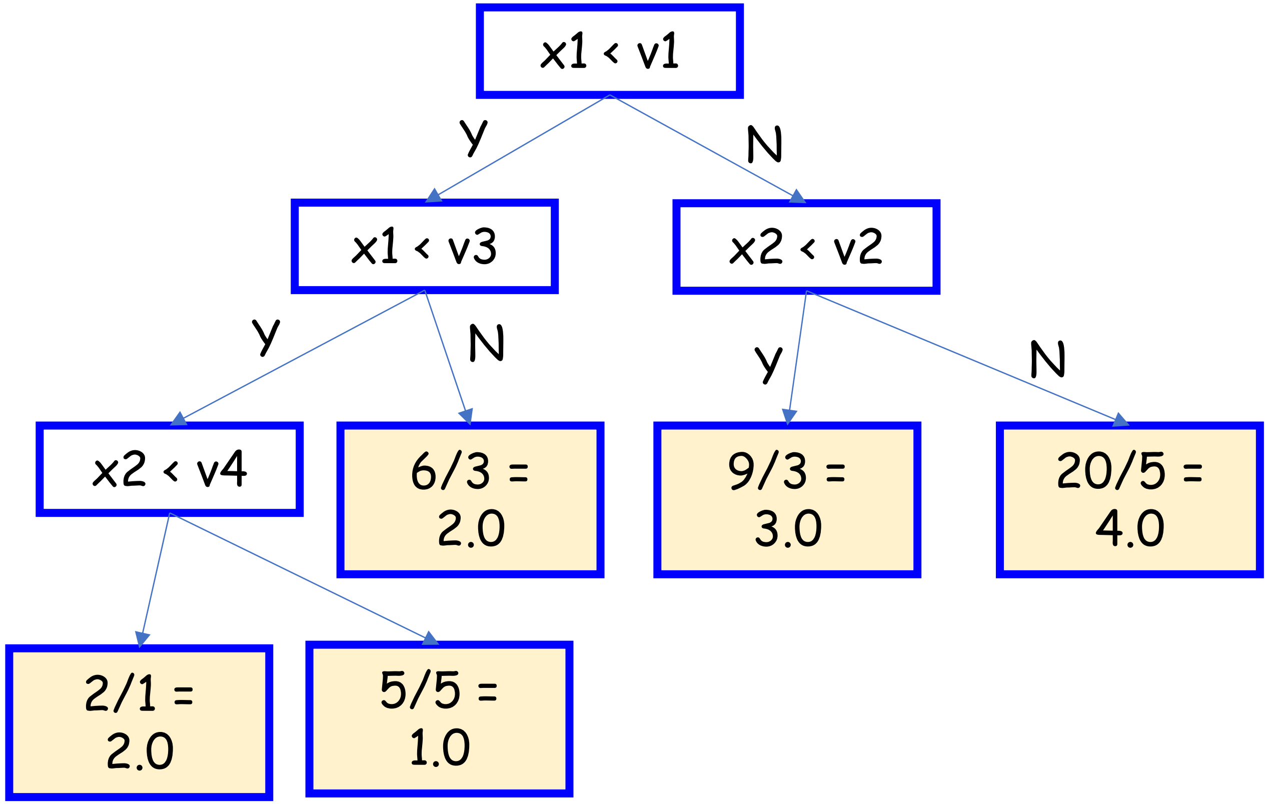


Training Regression Tree

Loss-function = MSE

Misal, pada iterasi pertama, **split variable x1** dengan **split value v1** memberikan predicted error paling minimal.





Training Regression Tree

Kapan berhenti splitting?

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

Konsep Gradient Boosting

Boosting

- Bagaimana membuat highly-effective model dengan menggabungkan banyak model-model yang “lemah”?

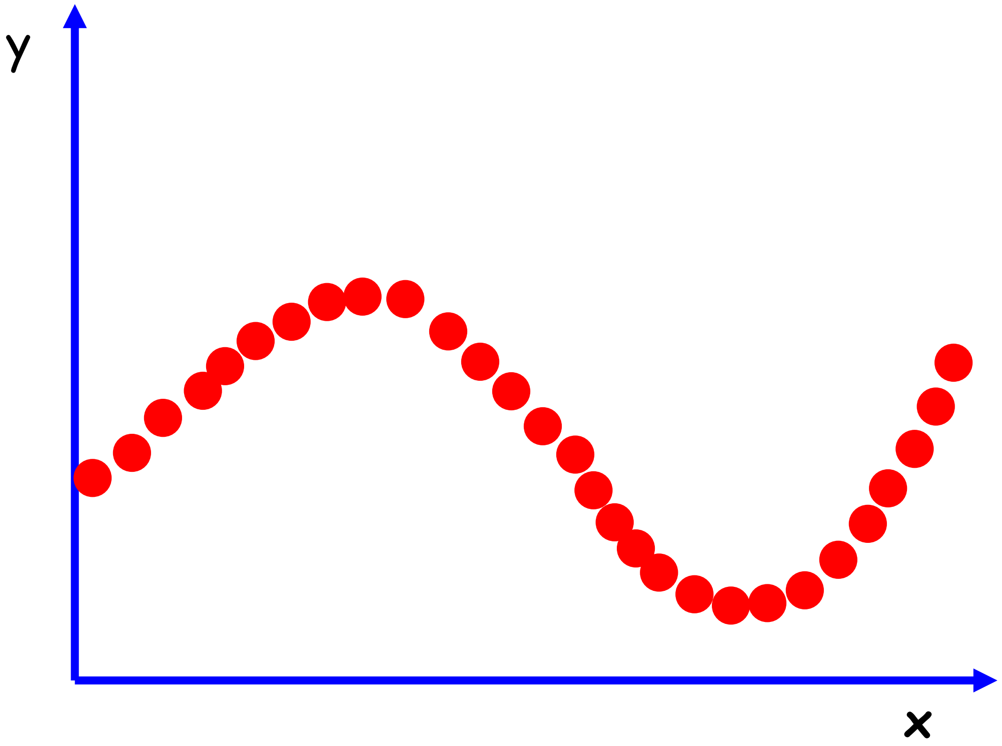
- Sebuah **Ensemble**, misal dengan additive model:

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

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

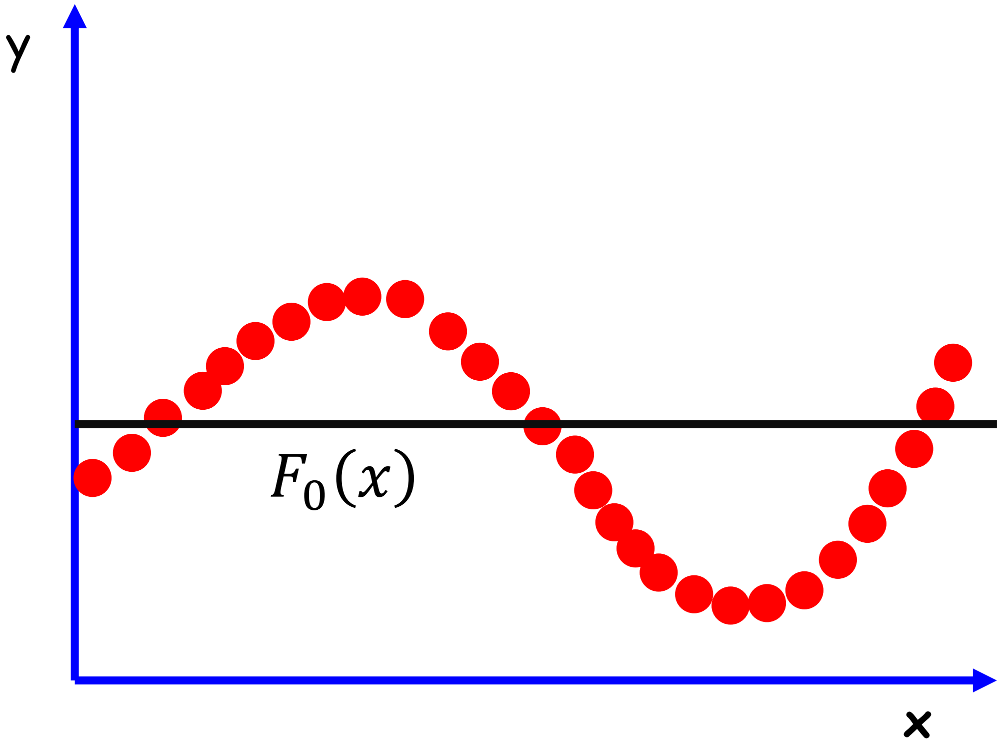
Boosting

Misal, {(x1, y1), (x2, y2), …, (xn, yn)} adalah training data



Boosting

Misal, {(x1, y1), (x2, y2), …, (xn, yn)} adalah training data



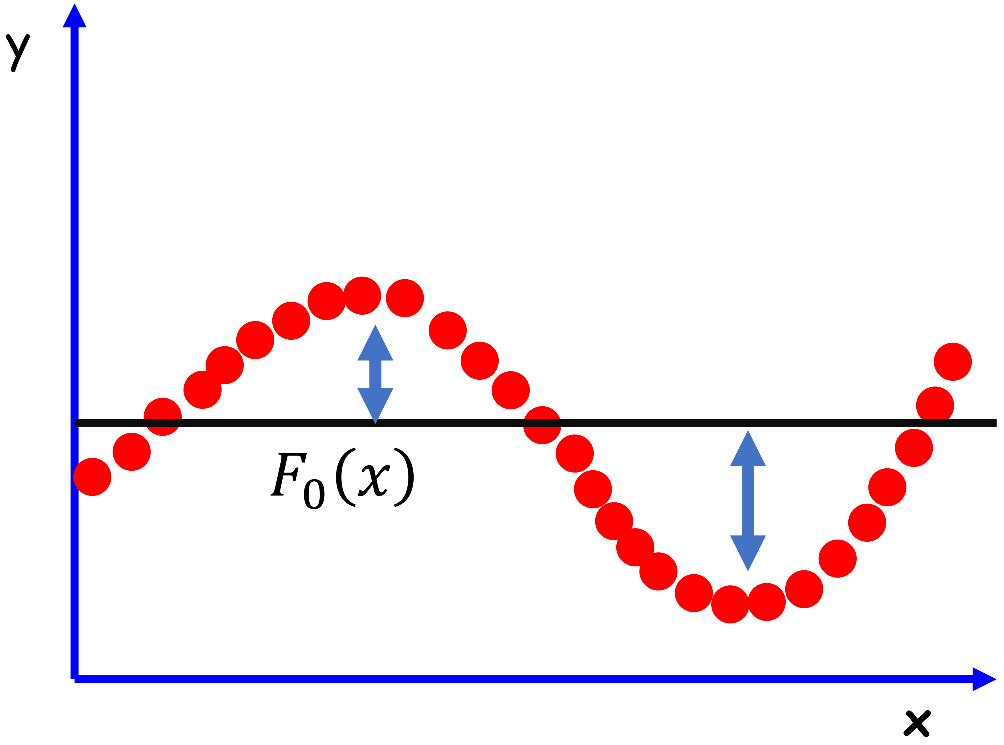
Dimulai dari sebuah "weak learner" simple:

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

Boosting

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

Misal, {(x1, y1), (x2, y2), …, (xn, yn)} adalah training data



Pseudo-Residual

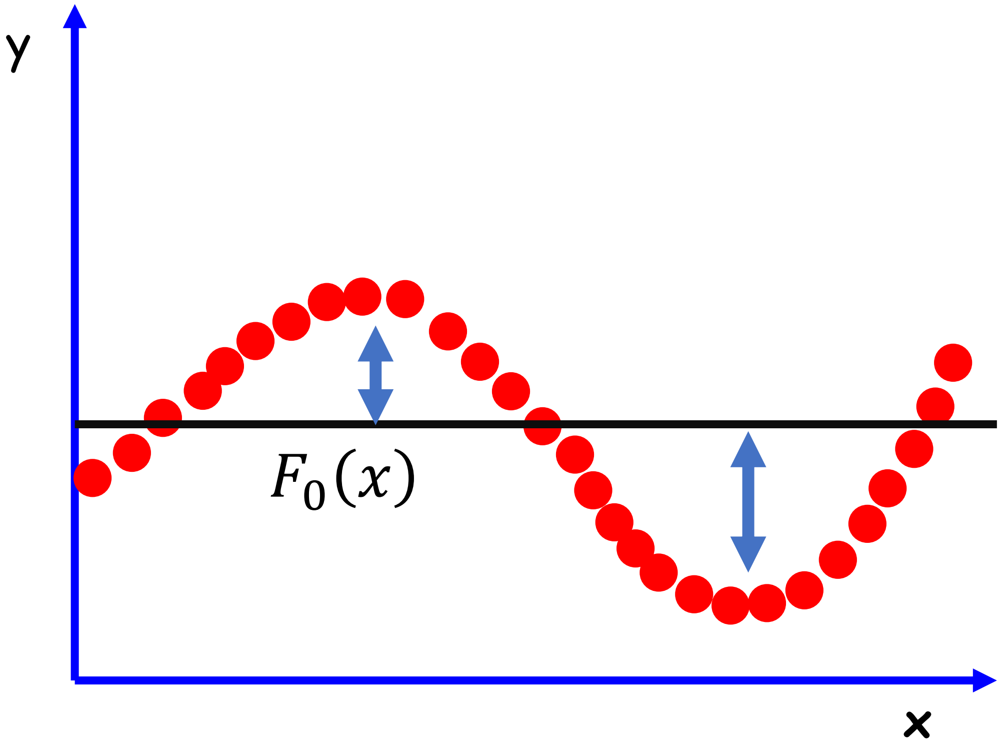
x	Psedo-Res.
x1	y1 - F_0(x1)
x2	y2 - F_0(x2)
…	…
xn	yn - F_0(xn)

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

Boosting

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

Misal, {(x1, y1), (x2, y2), …, (xn, yn)} adalah training data



Pseudo-Residual

x	Psedo-Res.
x1	y1 - F_0(x1)
x2	y2 - F_0(x2)
…	…
xn	yn - F_0(xn)

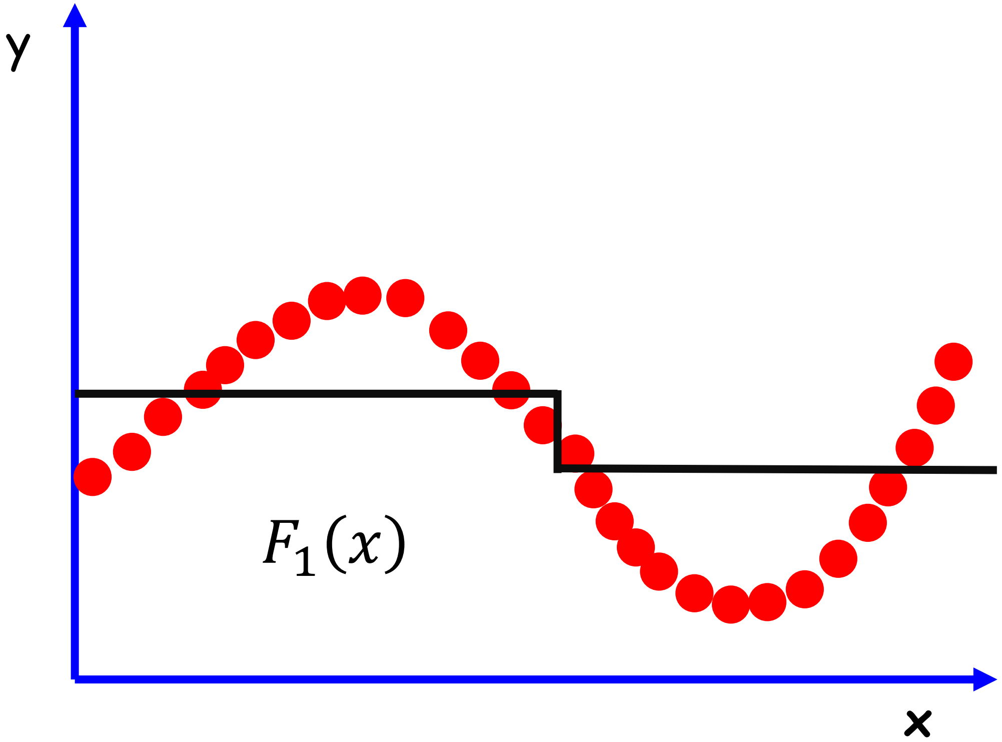
$$\boldsymbol{y} = F_0(\boldsymbol{x}) + \boldsymbol{\epsilon}^{(1)}$$

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

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

Boosting

Misal, {(x1, y1), (x2, y2), …, (xn, yn)} adalah training data



Pseudo-Residual

x	**Psedo-Res.**
x1	y1 - F_0(x1)
x2	y2 - F_0(x2)
…	…
xn	yn - F_0(xn)

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

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

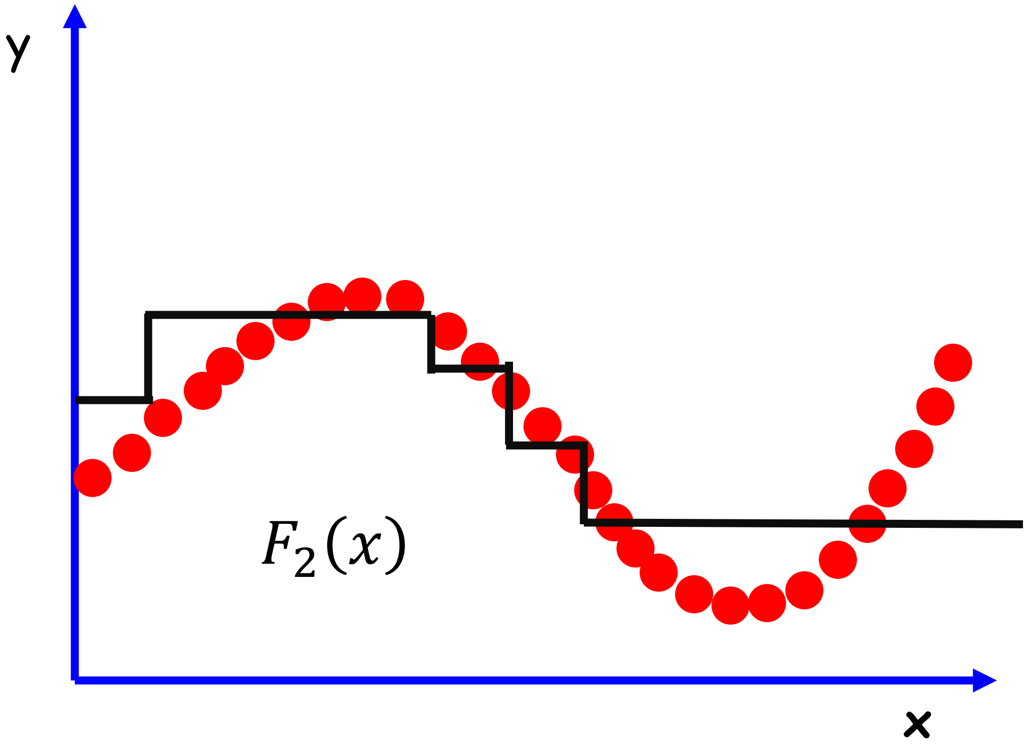
$$h_1(x) = argmin_h \left[\sum_{i=1}^{n} L(y_i - F_0(x_i), h(x_i))\right]$$

Model baru yang lebih baik dari sebelumnya: →

$$\boldsymbol{F(x) = F_1(x) = F_0(x) + h_1(x)}$$

Boosting

Misal, {(x1, y1), (x2, y2), …, (xn, yn)} adalah training data



Pseudo-Residual

x	Psedo-Res.
x1	y1 - F_1(x1)
x2	y2 - F_1(x2)
…	…
xn	yn - F_1(xn)

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

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

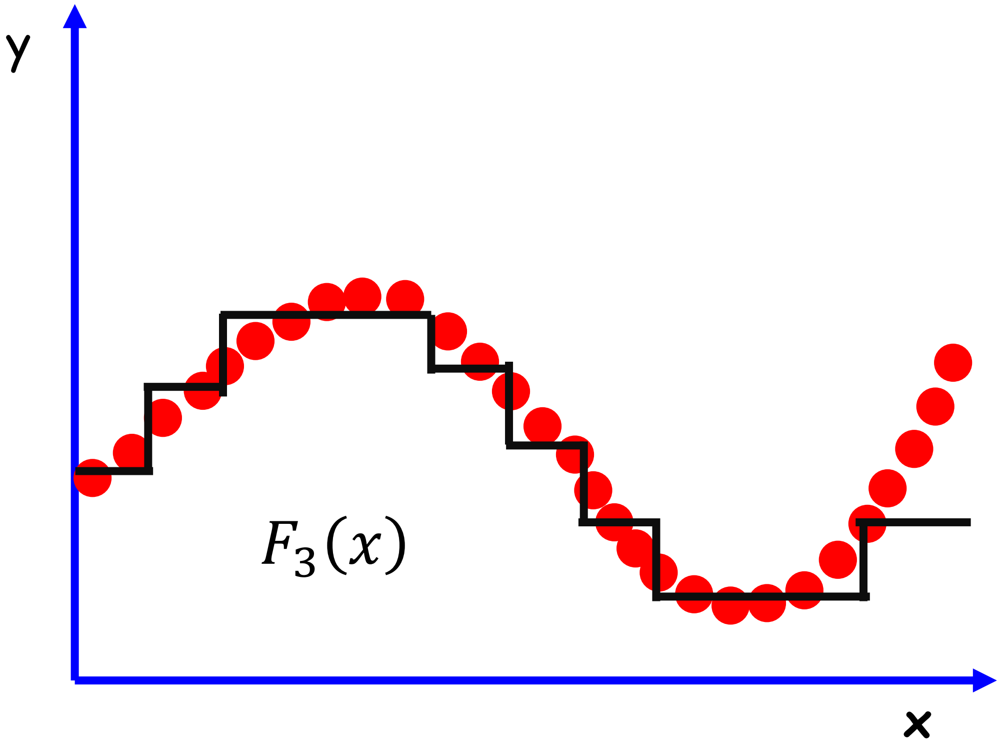
$$h_2(x) = argmin_h \left[\sum_{i=1}^{n} L(y_i - F_1(x_i), h(x_i)) \right]$$

Model baru yang lebih baik dari sebelumnya: →

$$\boldsymbol{F(x) = F_2(x) = F_0(x) + h_1(x) + h_2(x)}$$

Boosting

Misal, {(x1, y1), (x2, y2), …, (xn, yn)} adalah training data



Pseudo-Residual

x	Psedo-Res.
x1	y1 - F_2(x1)
x2	y2 - F_2(x2)
…	…
xn	yn - F_2(xn)

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

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

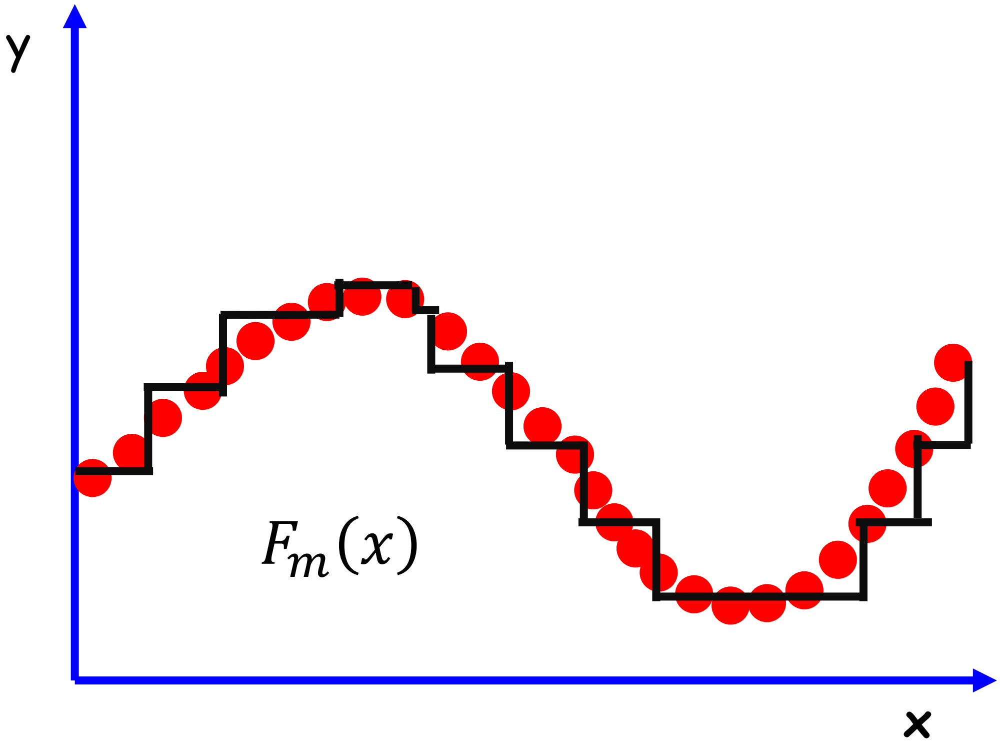
$$h_3(x) = argmin_h\left[\sum_{i=1}^{n} L(y_i - F_2(x_i), h(x_i))\right]$$

Model baru yang lebih baik dari sebelumnya:

$$\boldsymbol{F(x) = F_3(x) = F_0(x) + h_1(x) + h_2(x) + h_3(x)}$$

Boosting

Misal, {(x1, y1), (x2, y2), …, (xn, yn)} adalah training data

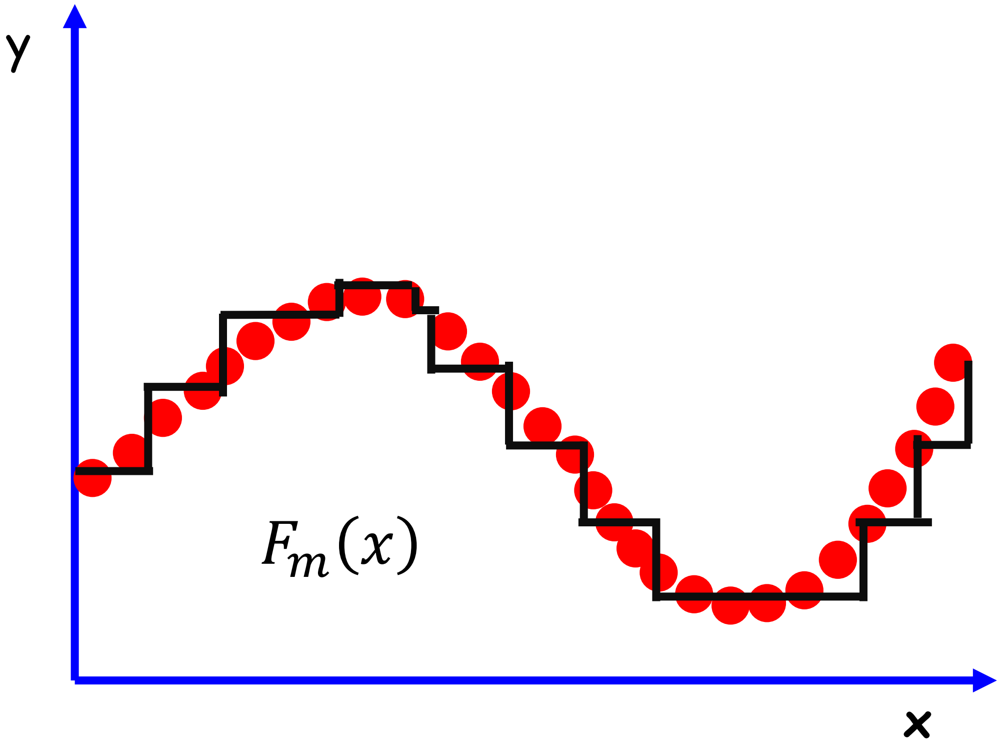


Dan seterusnya …

Model baru yang lebih baik dari sebelumnya: → $\boldsymbol{F(x) = F_m(x) = F_0(x) + h_1(x) + \cdots + h_m(x)}$

Boosting

Misal, {(x1, y1), (x2, y2), …, (xn, yn)} adalah training data



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

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

$$\Downarrow$$

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

$$\Downarrow$$

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

Model baru yang lebih baik dari sebelumnya: →

$$\boldsymbol{F(x) = F_m(x) = F_0(x) + h_1(x) + \cdots + h_m(x)}$$

Boosting

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

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

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

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

Boosting

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

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

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

Gradient Boosting

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

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”:

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

Cari fungsi $\boldsymbol{h_m(x)}$ yang memprediksi negative gradient dari L!

Bukti: https://explained.ai/gradient-boosting/descent.html

Gradient Boosting

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

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

$$-\frac{\partial L}{\partial F_{m-1}(x_i)} = y_i - F_{m-1}(x_i) \;\; \overset{\text{"proportional"}}{\sim} \;\; h_m(x_i)$$

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

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

Gradient Boosting Framework

Inisilisasi model dengan nilai konstan:

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

For m = 1 to M :

1. For i = 1 to n:

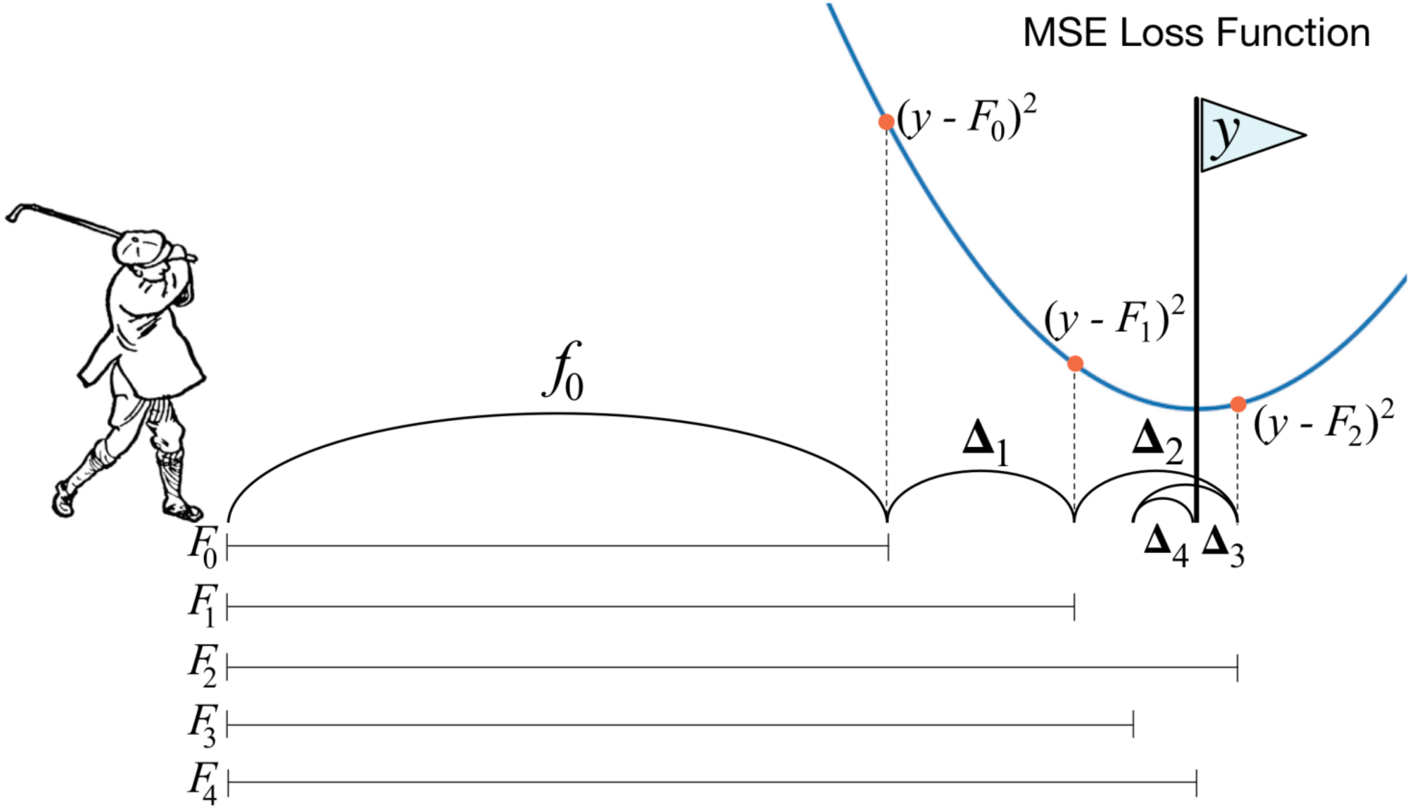
   Hitung pseudo-residual: $r_{im} = -\left[\frac{\partial L(y_i, F_{m-1}(x_i))}{\partial F_{m-1}(x_i)}\right]$

2. Fit weak learner, $\boldsymbol{h_m(x)}$, dengan pseudo-residual {(x1, r_{1m}), …, (x1, r_{nm})}
3. Cari weight coefficient, $\boldsymbol{\beta_m}$, dengan optimization berikut:

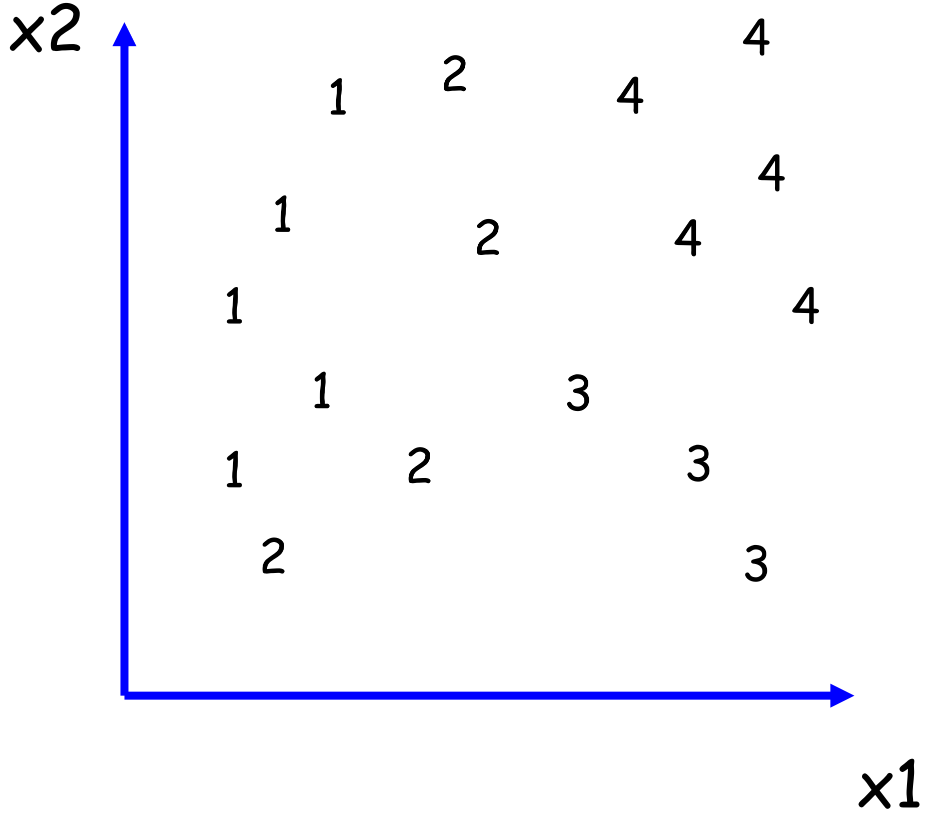
$$\beta_m = argmin_\beta \sum_{i=1}^{n} L(y_i, F_{m-1}(x_i) + \beta h_m(x_i))$$

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)



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

Misal, loss-function:

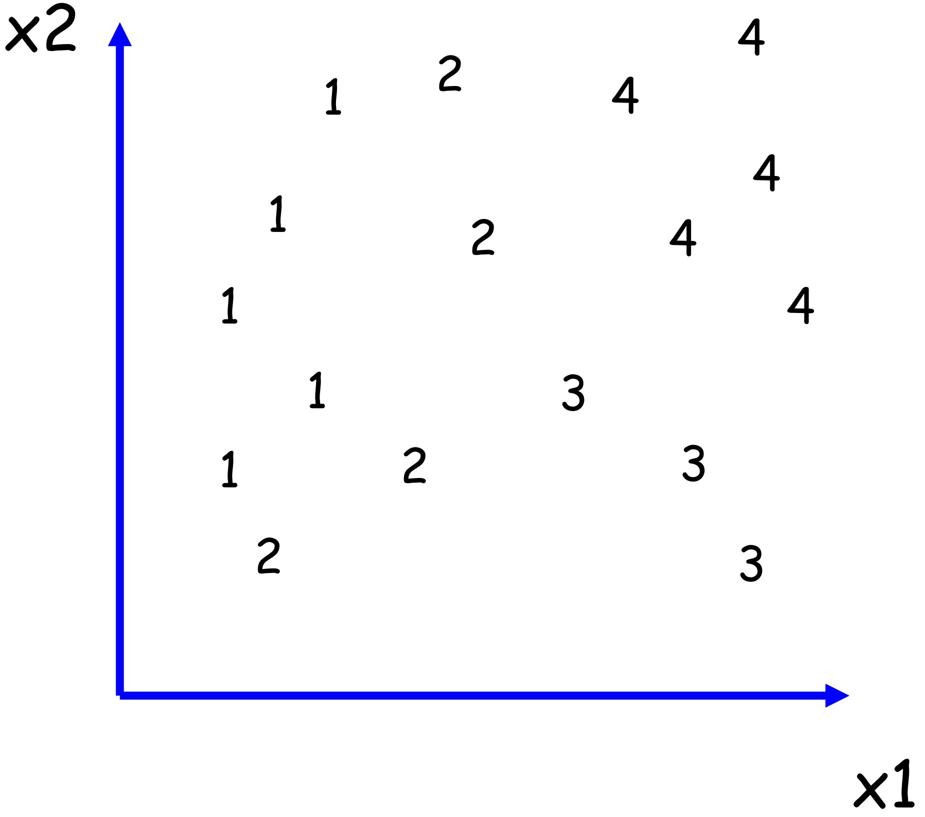
$$L = \sum_{x=1}^{n} \frac{1}{2}\left(y_i - f(\boldsymbol{x})\right)^2$$

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

Set dL/dx = 0 akan menemukan L optimum di γ = 2.471

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

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

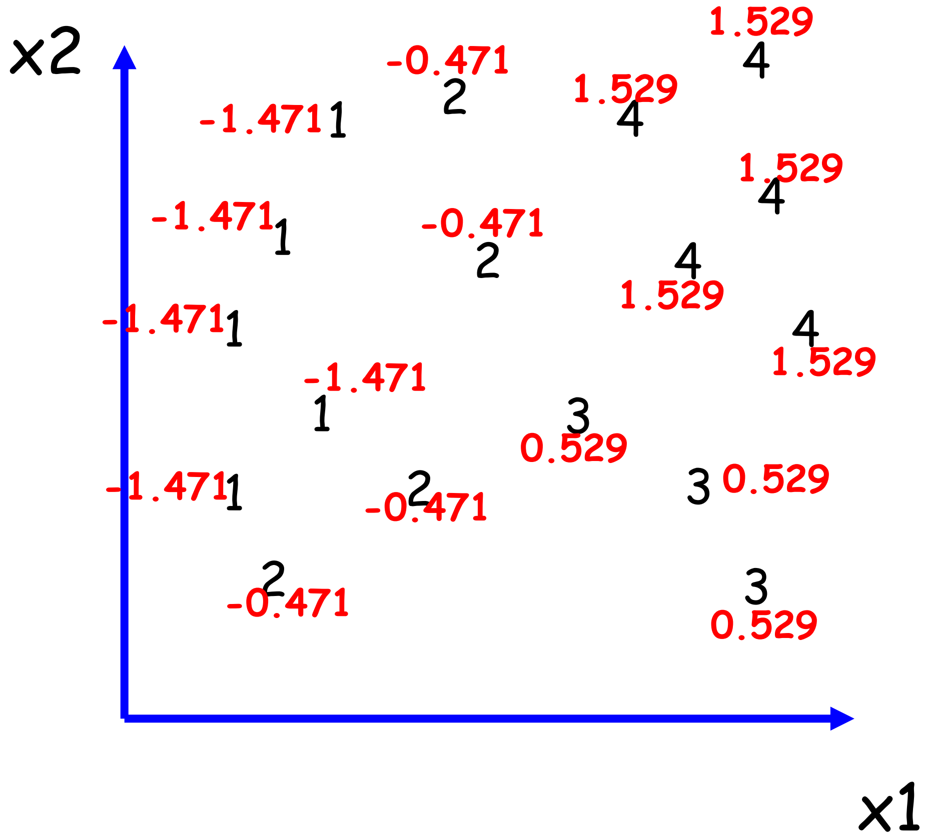


$F_0(x)$

$\boxed{2.471}$

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

Merah: pseudo-residual



$F_0(x)$

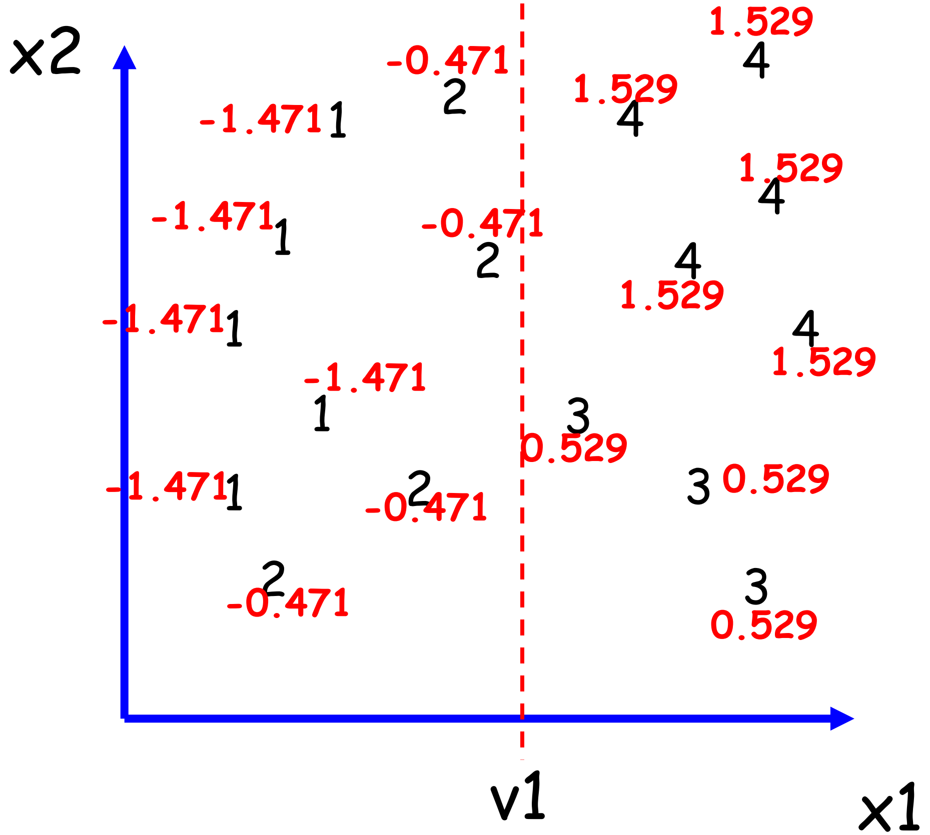
$\boxed{2.471}$

Asumsi Tree: hanya sampai depth = 1

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

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

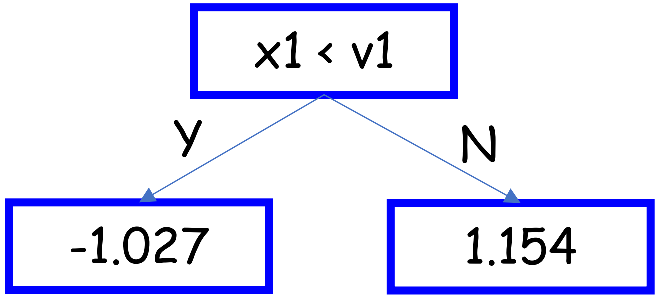
Merah: pseudo-residual



$F_0(x)$

2.471

$h_1(x)$



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

dL/dx = 0
=> x = -1.027

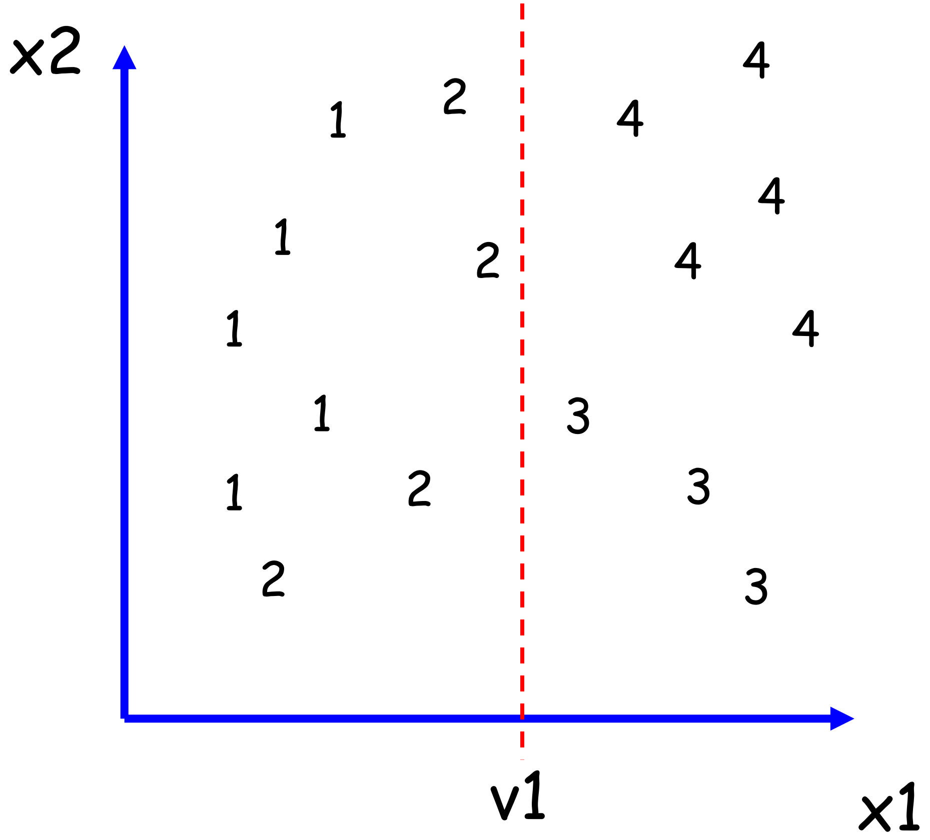
L = (5/2)(4 – (2.471 + x))^2
+ (3/2)(3 – (2.471 + x))^2

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

dL/dx = 0
=> x = 1.154

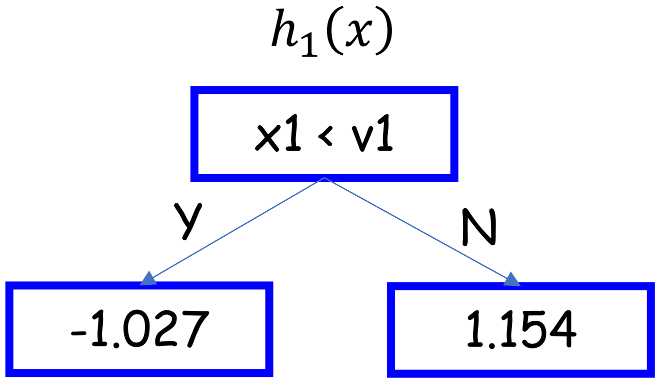
MART: Multiple Additive Regression Trees (Gradient-Boosted Regression Trees)

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

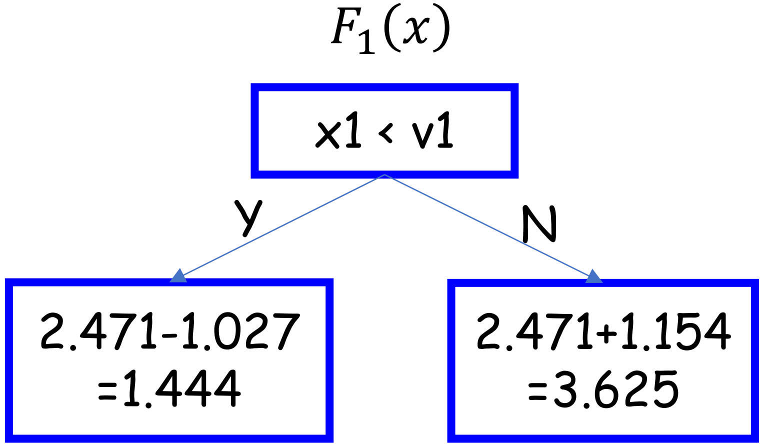


$F_0(x)$

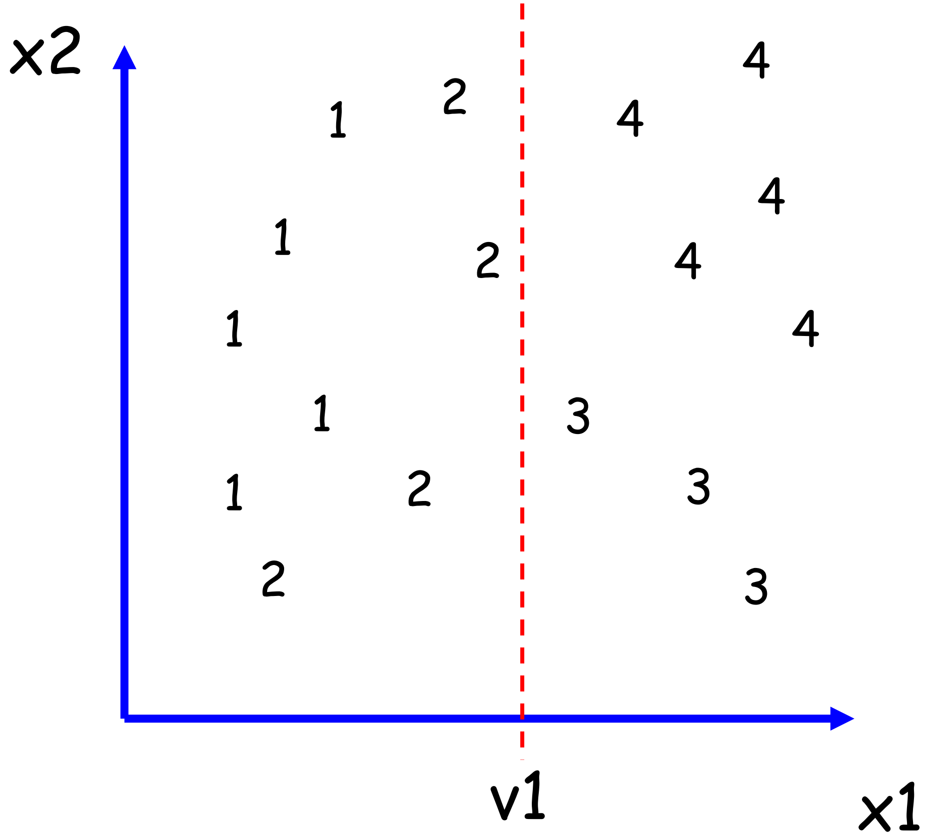
2.471



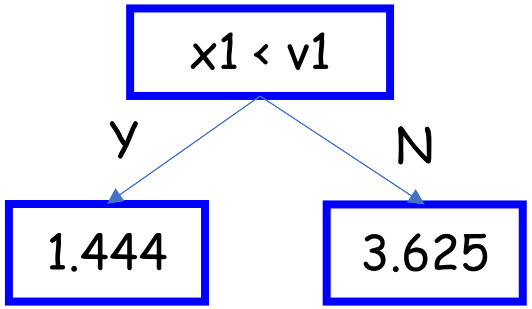
$$F_1(x) = F_0(x) + h_1(x)$$



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



$F_1(x)$



x1 < v1

Y

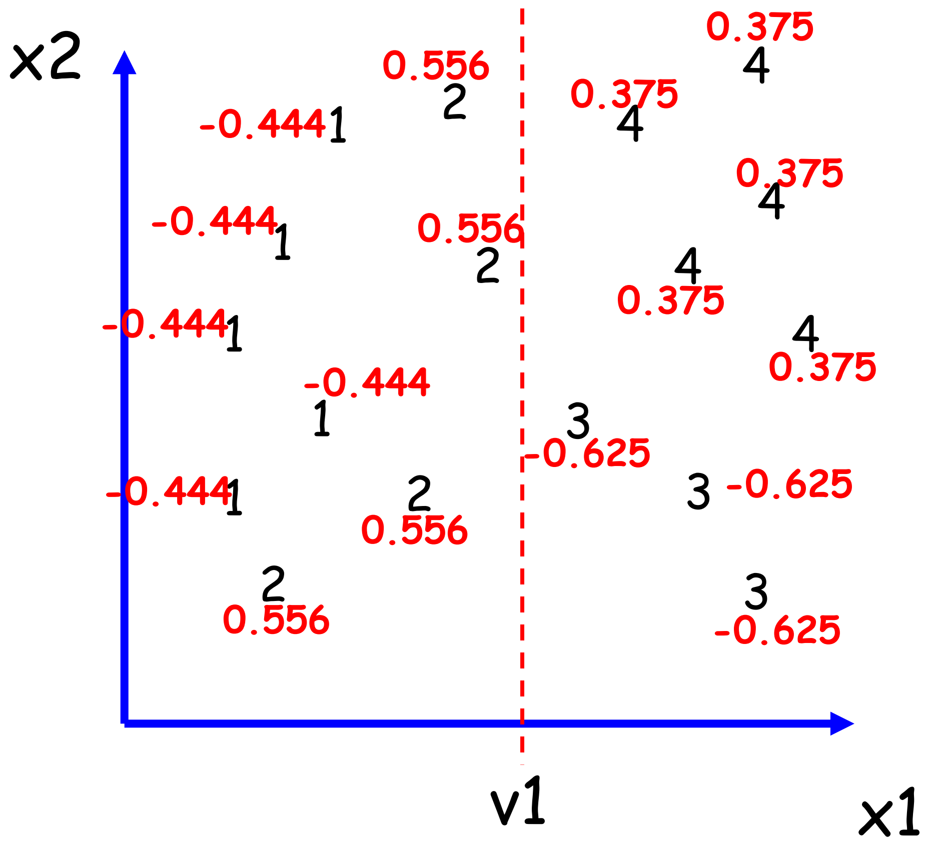
N

1.444

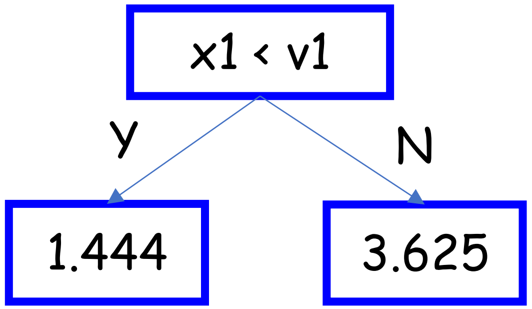
3.625

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

Merah: pseudo-residual



$F_1(x)$



x1 < v1

Y

N

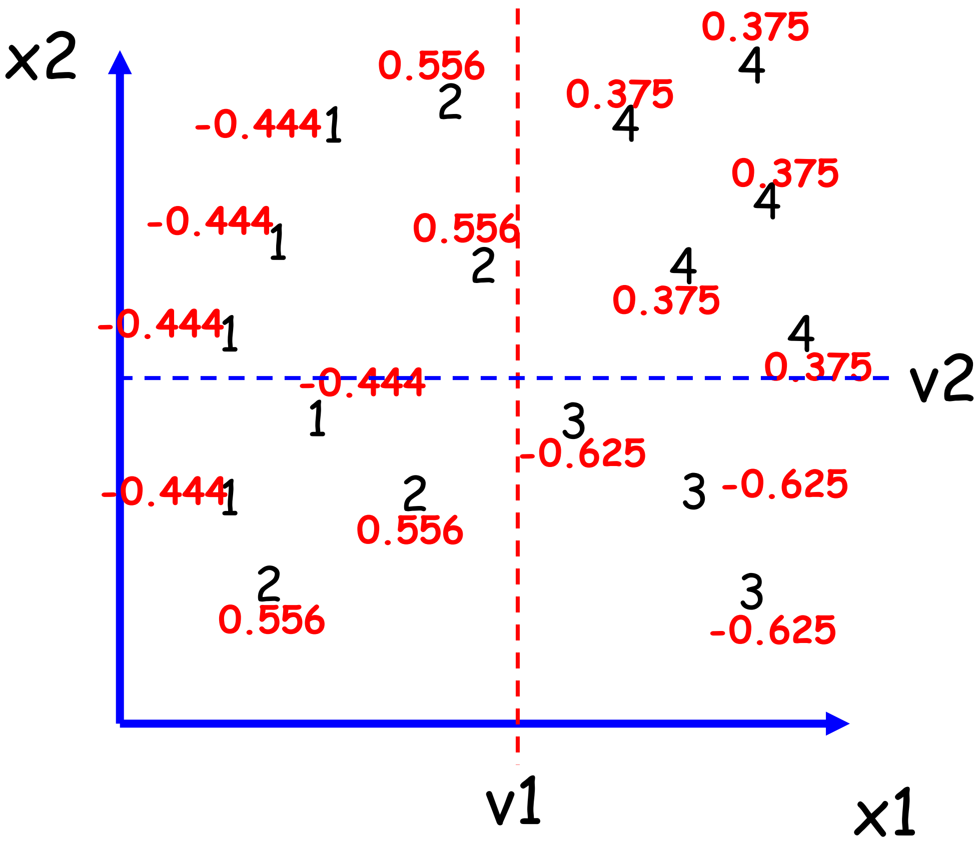
1.444

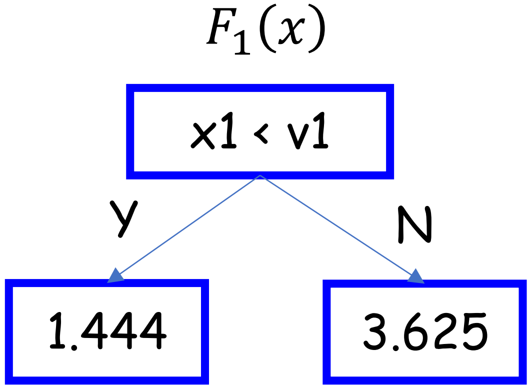
3.625

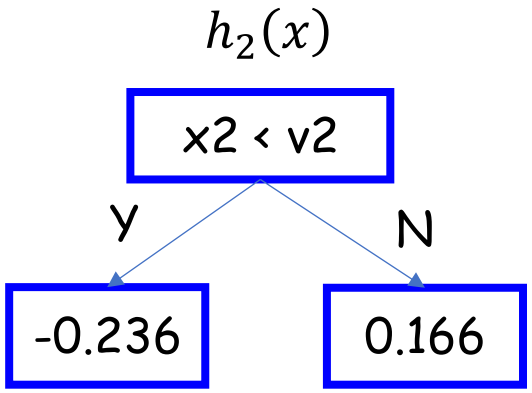
MART: Multiple Additive Regression Trees (Gradient-Boosted Regression Trees)

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

Merah: pseudo-residual







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

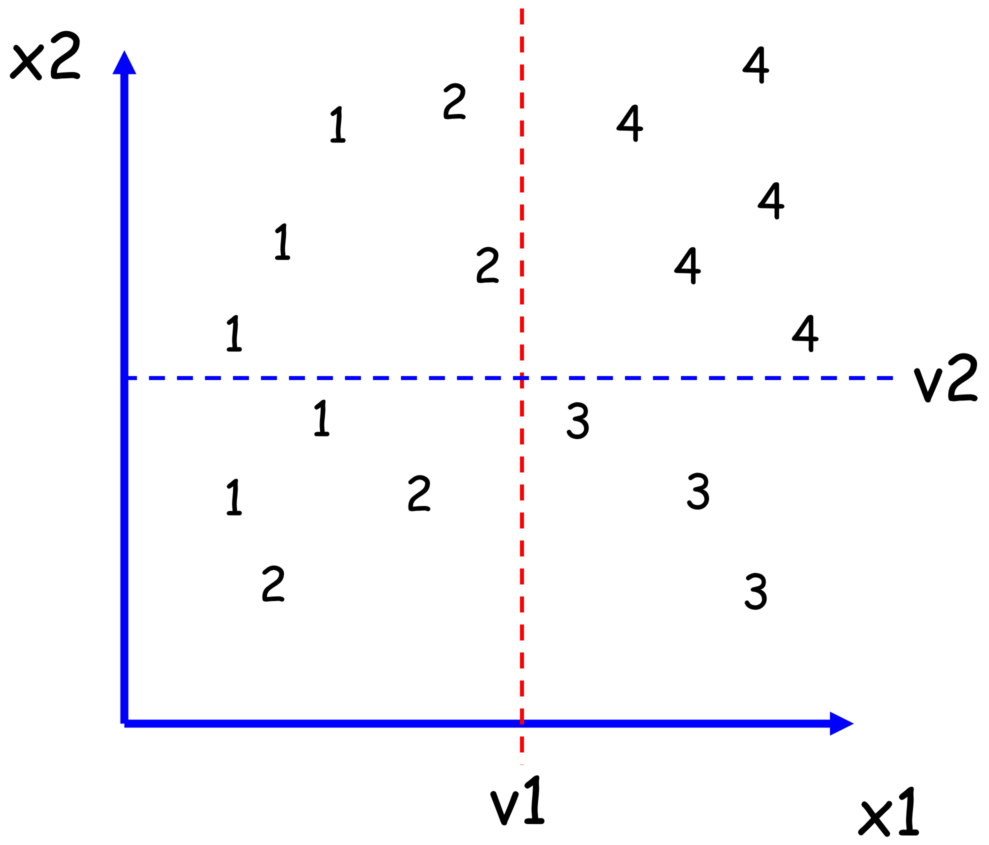
dL/dx = 0
=> x = -0.236

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

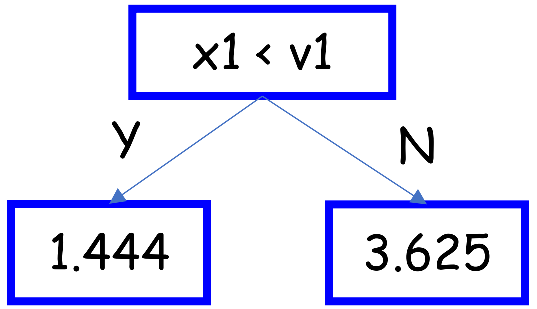
dL/dx = 0
=> x = 0.166

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

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



$F_1(x)$


x1 < v1

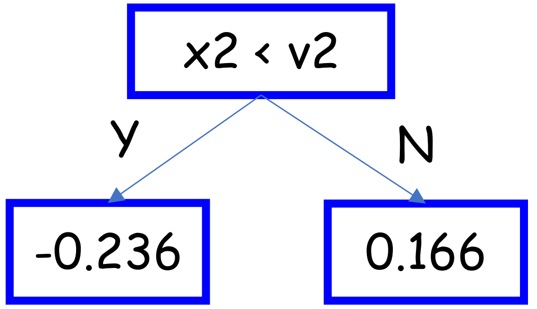
Y

N

1.444

3.625

$h_2(x)$


x2 < v2

Y

N

-0.236

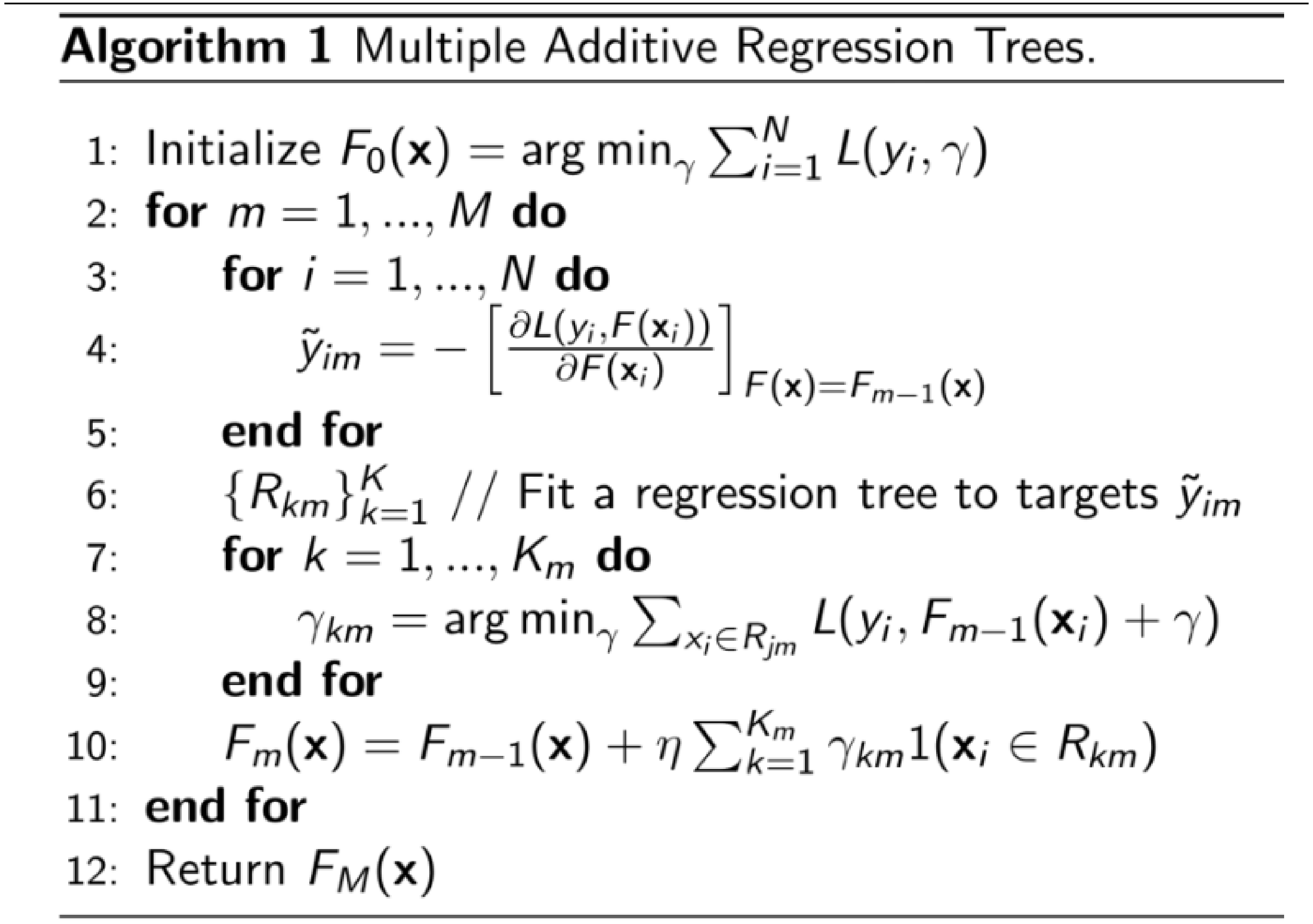
0.166

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

x1 < v1 & x2 < v2	= 1.444 – 0.236
x1 < v1 & x2 >= v2	= 1.444 + 0.166
x1 >= v1 & x2 < v2	= 3.625 – 0.236
x1 >= v1 & x2 >= v2	= 3.625 + 0.166

[Friedman 1999]

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

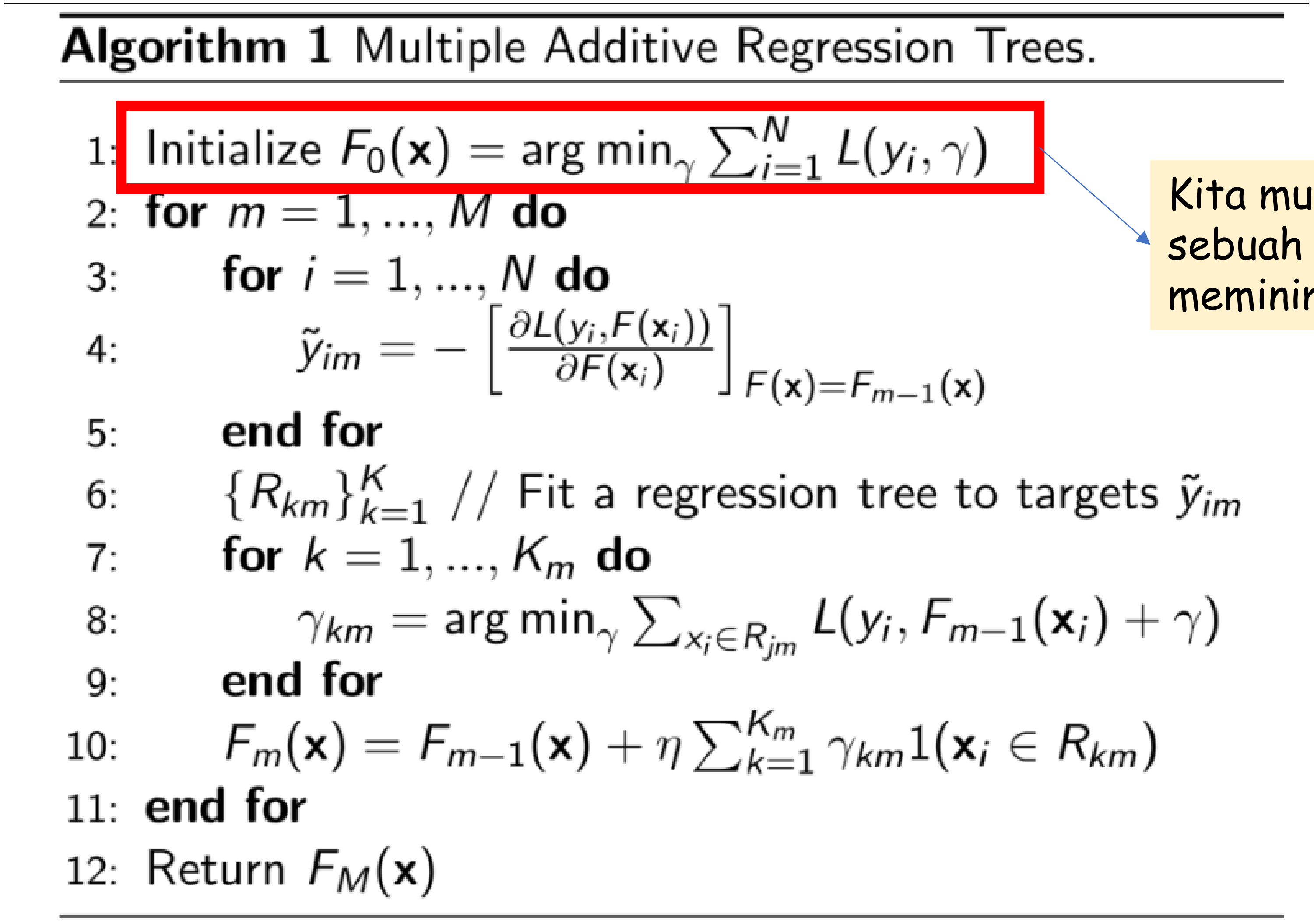


Algorithm 1 Multiple Additive Regression Trees.

```
Initialize $F_0(\mathbf{x}) = \arg\min_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$
for $m = 1, \ldots, M$ do
    for $i = 1, \ldots, N$ do
        $\tilde{y}_{im} = -\left[\frac{\partial L(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)}\right]_{F(\mathbf{x}) = F_{m-1}(\mathbf{x})}$
    end for
    $\{R_{km}\}_{k=1}^{K}$ // Fit a regression tree to targets $\tilde{y}_{im}$
    for $k = 1, \ldots, K_m$ do
        $\gamma_{km} = \arg\min_\gamma \sum_{x_i \in R_{jm}} L(y_i, F_{m-1}(\mathbf{x}_i) + \gamma)$
    end for
    $F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \eta \sum_{k=1}^{K_m} \gamma_{km} 1(\mathbf{x}_i \in R_{km})$
end for
Return $F_M(\mathbf{x})$
```

[Friedman 1999]

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



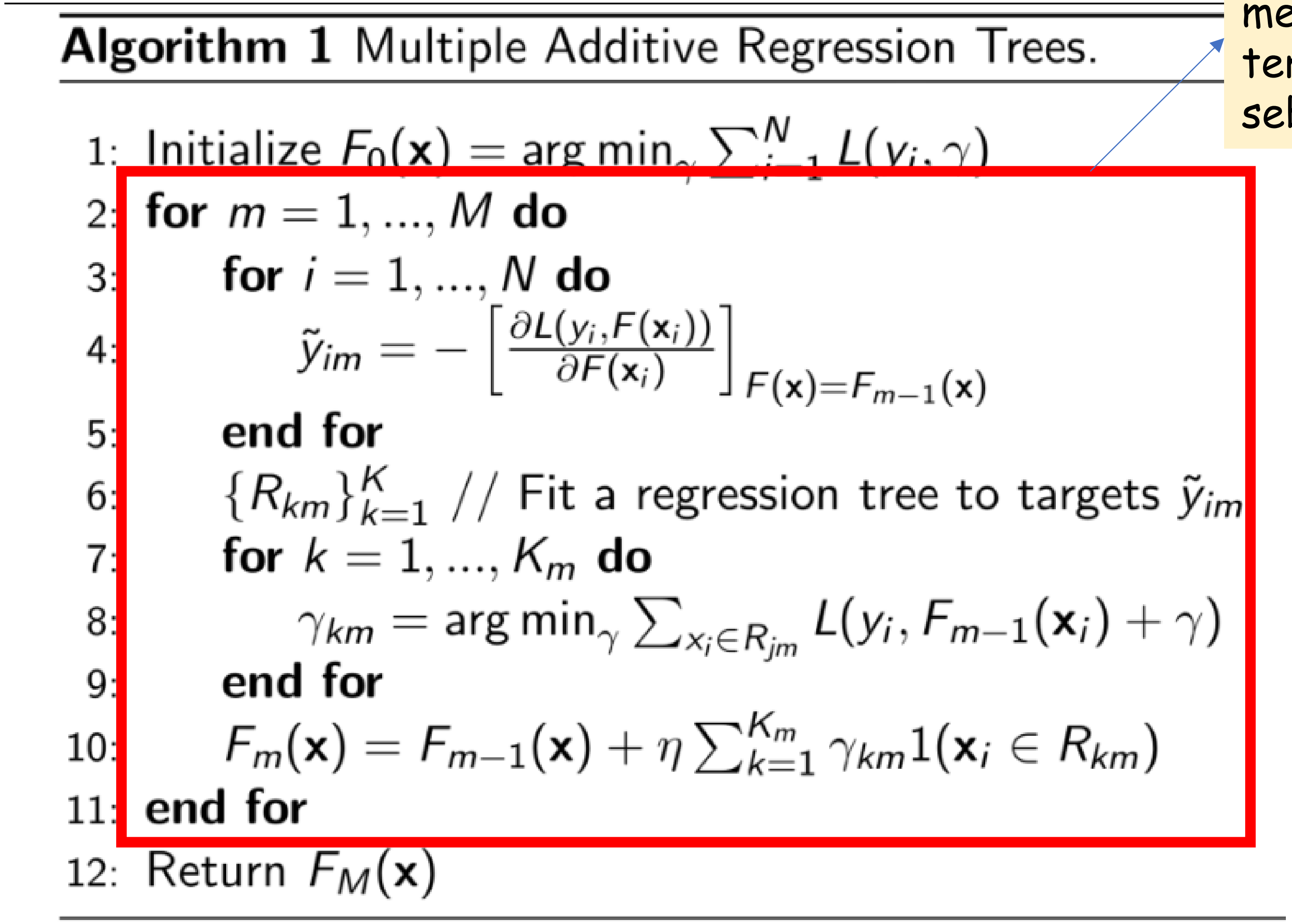
Algorithm 1 Multiple Additive Regression Trees.

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

Kita mulai F(x) dengan sebuah nilai konstan yang meminimalkan error.

[Friedman 1999]

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



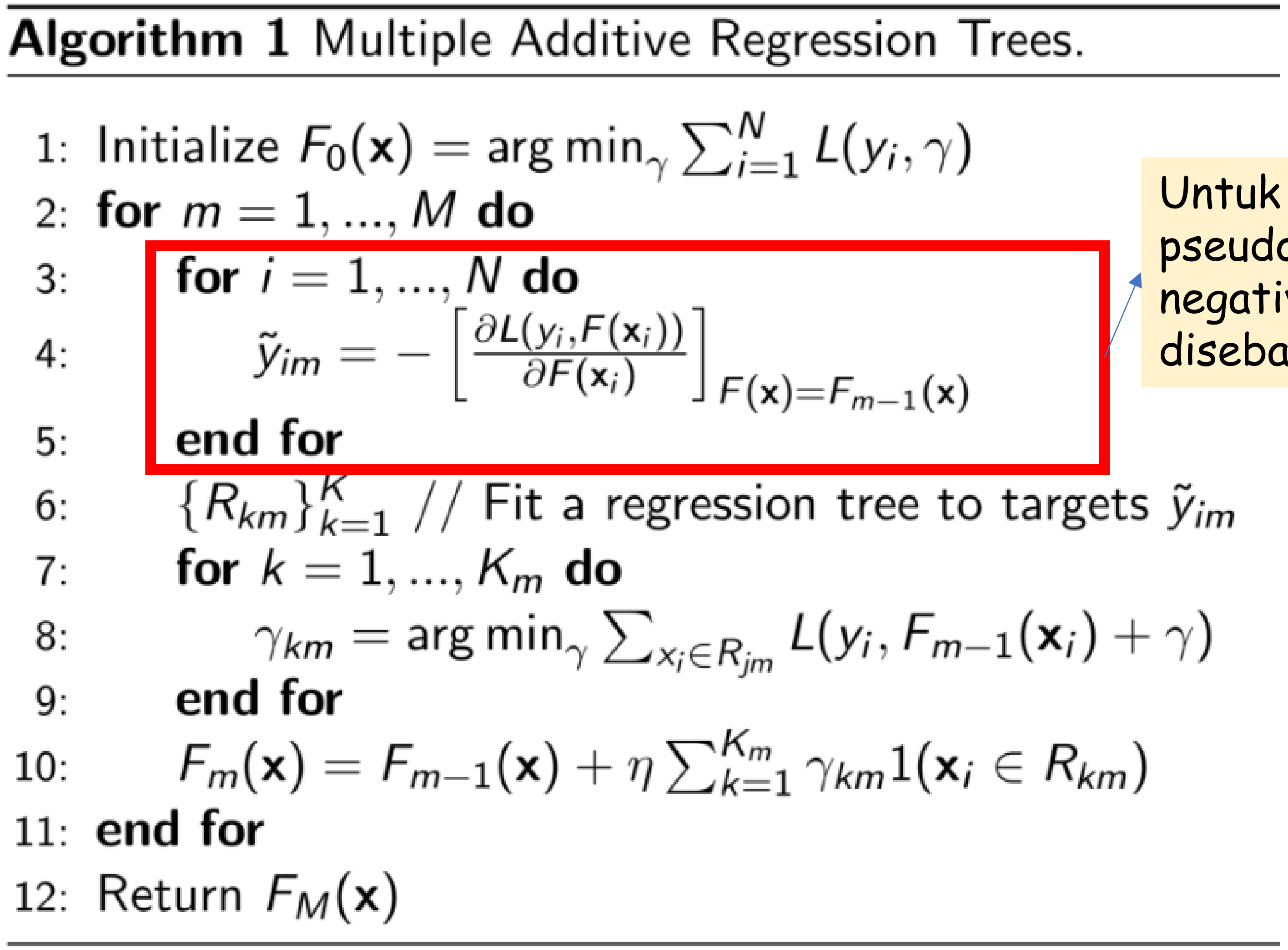
Algorithm 1 Multiple Additive Regression Trees.

```
Initialize $F_0(\mathbf{x}) = \arg\min_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$
for $m = 1, ..., M$ do
    for $i = 1, ..., N$ do
        $\tilde{y}_{im} = -\left[\frac{\partial L(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)}\right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})}$
    end for
    $\{R_{km}\}_{k=1}^{K}$ // Fit a regression tree to targets $\tilde{y}_{im}$
    for $k = 1, ..., K_m$ do
        $\gamma_{km} = \arg\min_\gamma \sum_{x_i \in R_{jm}} L(y_i, F_{m-1}(\mathbf{x}_i) + \gamma)$
    end for
    $F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \eta \sum_{k=1}^{K_m} \gamma_{km} 1(\mathbf{x}_i \in R_{km})$
end for
Return $F_M(\mathbf{x})$
```

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

[Friedman 1999]

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



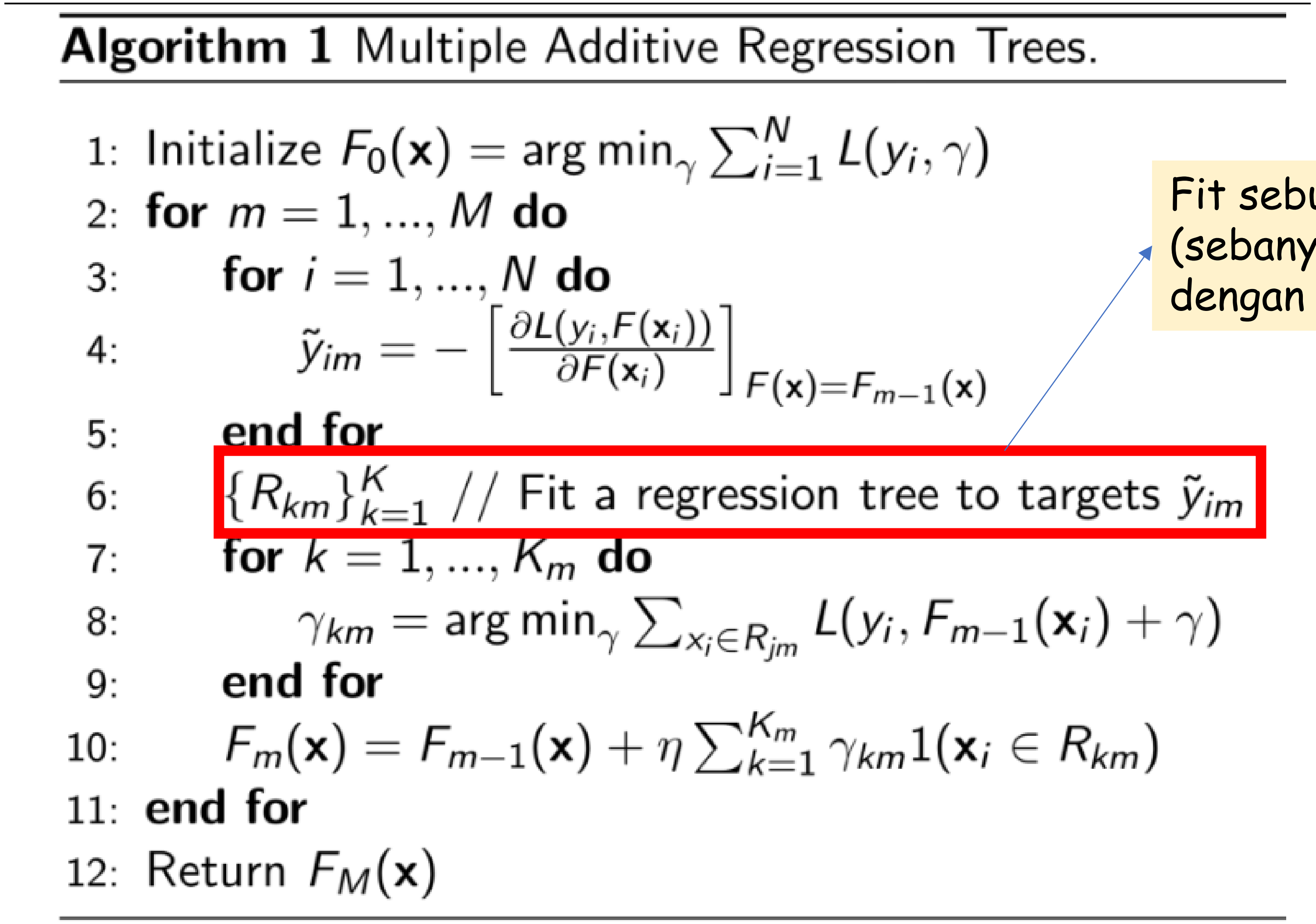
Algorithm 1 Multiple Additive Regression Trees.

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

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

[Friedman 1999]

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



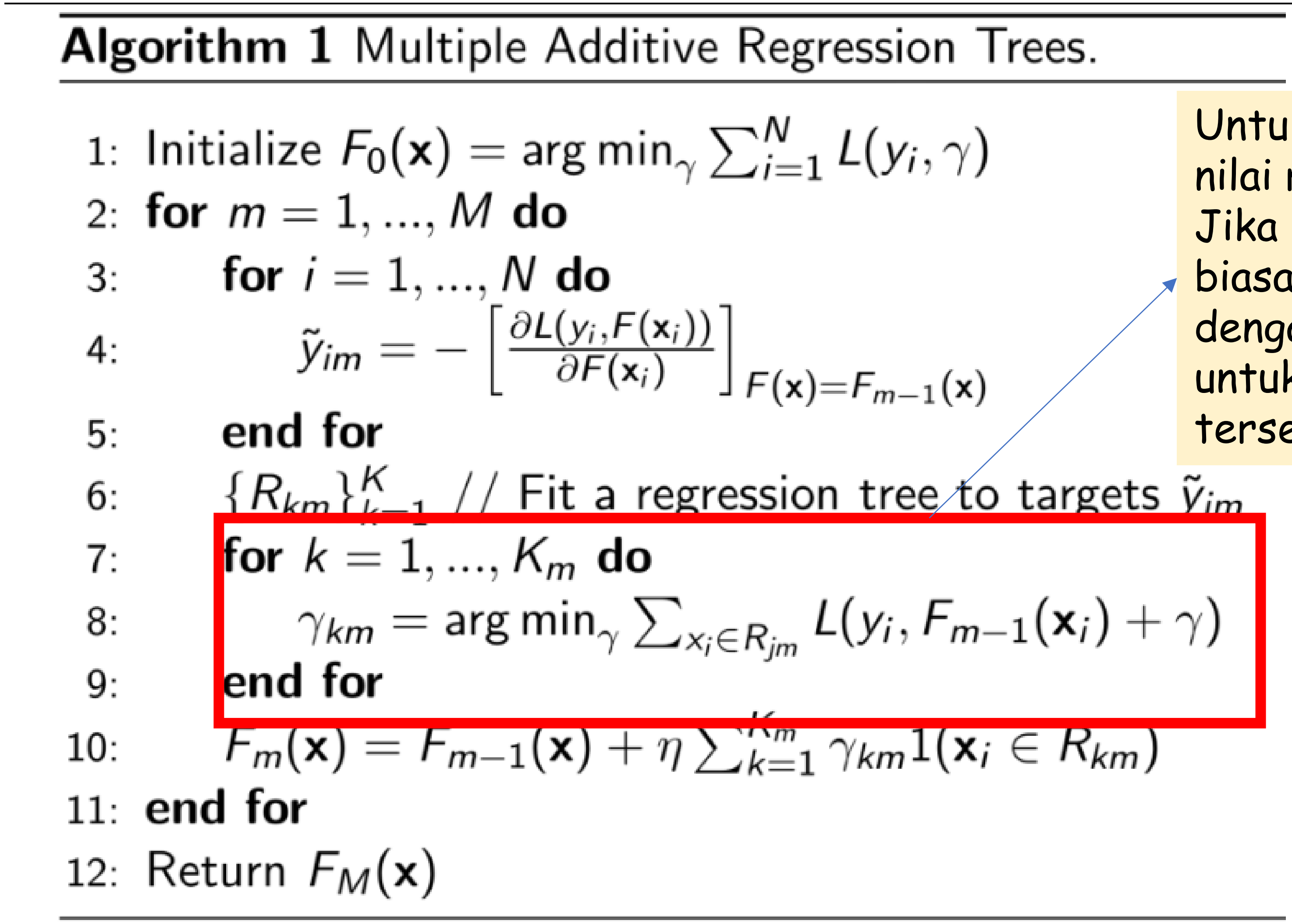
Algorithm 1 Multiple Additive Regression Trees.

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

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

[Friedman 1999]

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



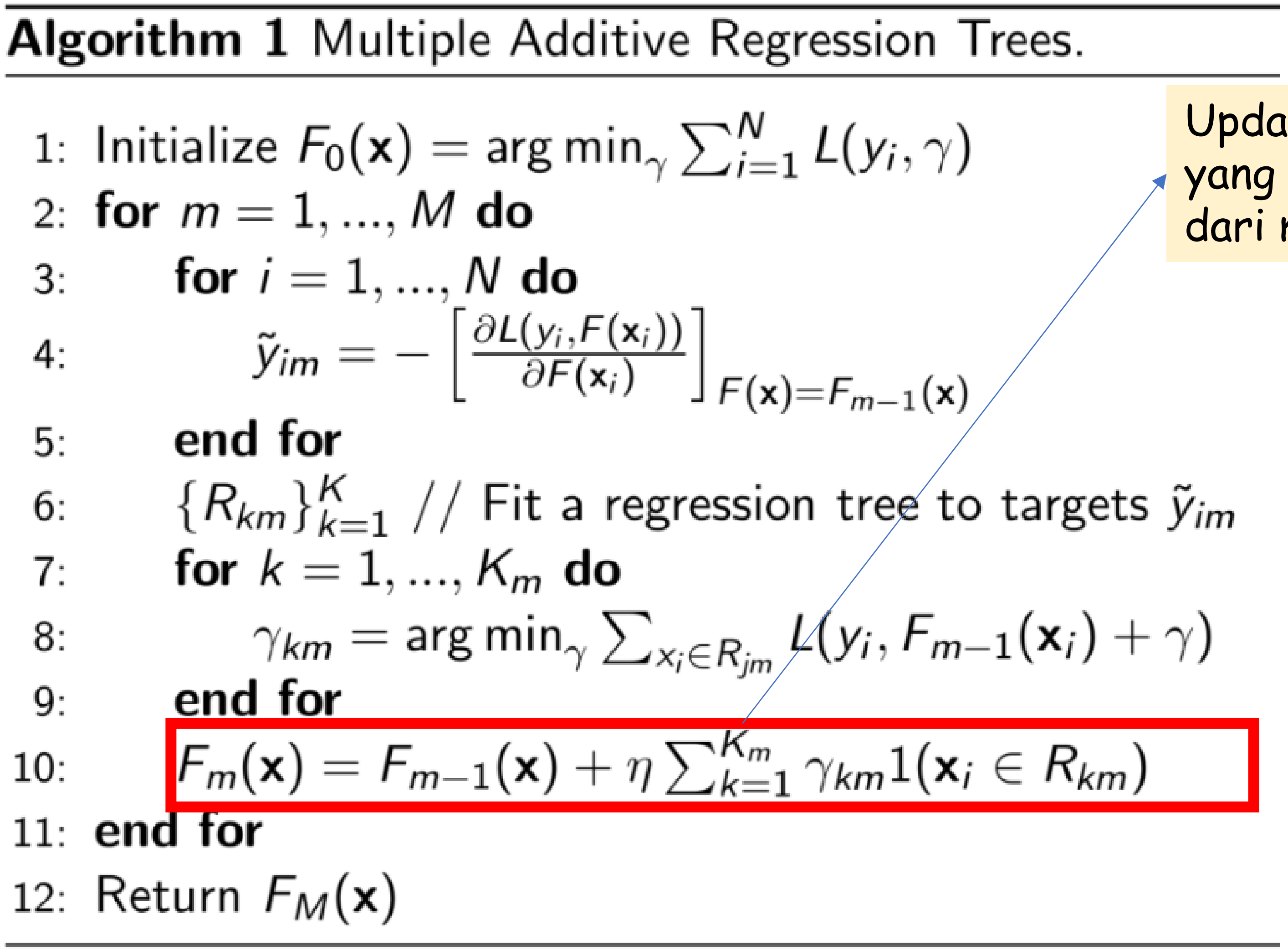
Algorithm 1 Multiple Additive Regression Trees.

```
Initialize $F_0(\mathbf{x}) = \arg\min_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$
for $m = 1, ..., M$ do
    for $i = 1, ..., N$ do
        $\tilde{y}_{im} = -\left[\frac{\partial L(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)}\right]_{F(\mathbf{x}) = F_{m-1}(\mathbf{x})}$
    end for
    $\{R_{km}\}_{k=1}^{K}$ // Fit a regression tree to targets $\tilde{y}_{im}$
    for $k = 1, ..., K_m$ do
        $\gamma_{km} = \arg\min_\gamma \sum_{x_i \in R_{jm}} L(y_i, F_{m-1}(\mathbf{x}_i) + \gamma)$
    end for
    $F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \eta \sum_{k=1}^{K_m} \gamma_{km} 1(\mathbf{x}_i \in R_{km})$
end for
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.

[Friedman 1999]

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



Algorithm 1 Multiple Additive Regression Trees.

```
Initialize $F_0(\mathbf{x}) = \arg\min_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$
for $m = 1, \ldots, M$ do
    for $i = 1, \ldots, N$ do
        $\tilde{y}_{im} = -\left[\frac{\partial L(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)}\right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})}$
    end for
    $\{R_{km}\}_{k=1}^{K}$ // Fit a regression tree to targets $\tilde{y}_{im}$
    for $k = 1, \ldots, K_m$ do
        $\gamma_{km} = \arg\min_\gamma \sum_{x_i \in R_{jm}} L(y_i, F_{m-1}(\mathbf{x}_i) + \gamma)$
    end for
    $\boxed{F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \eta \sum_{k=1}^{K_m} \gamma_{km} 1(\mathbf{x}_i \in R_{km})}$
end for
Return $F_M(\mathbf{x})$
```

Update model terbaru **F**$_m$**(x)** yang merupakan hasil boosting dari model **F**$_{m-1}$**(x)**

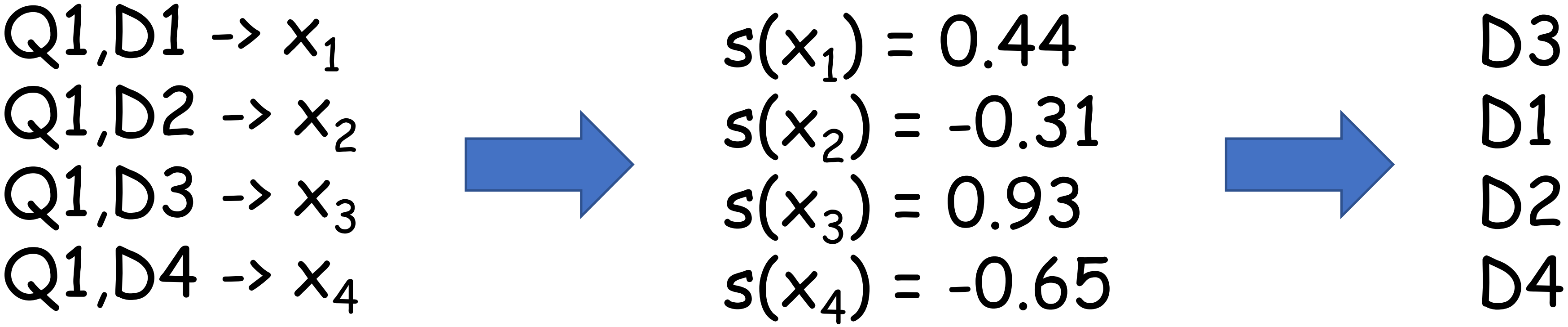
RankNet (Burges, 2010)

Ide

Misal, **x** adalah **representasi vektor** dari pasangan **<query, dokumen>**.

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

Misal:



Q1,D1 -> x_1
Q1,D2 -> x_2
Q1,D3 -> x_3
Q1,D4 -> x_4

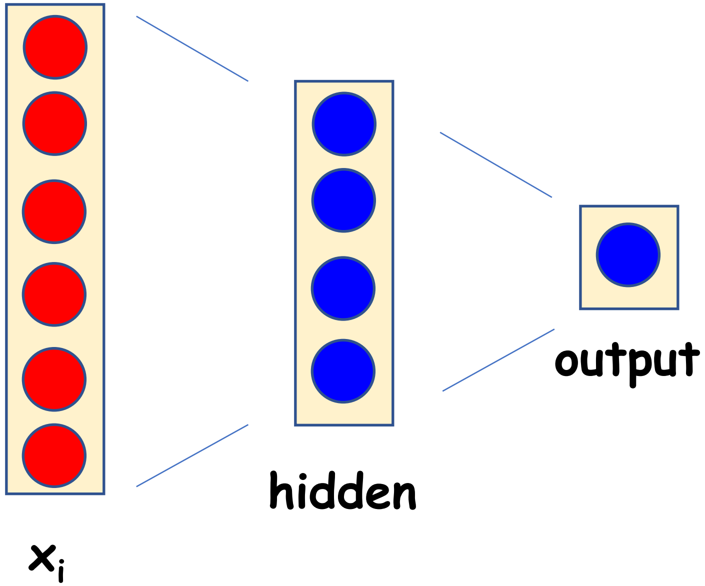
$s(x_1) = 0.44$
$s(x_2) = -0.31$
$s(x_3) = 0.93$
$s(x_4) = -0.65$

D3
D1
D2
D4

$s(x_i)$

- Bisa berupa beberapa feed-forward neural networks dengan **output berdimensi 1**.

- Misal, **$s(x_i;\ W1,\ W2)$** dengan 2-layer neural networks:



(num_hidden_units x 1)

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

(num_features x num_hidden_units)

(num_features x 1)

$\boldsymbol{P_{ij}}$: Probabilitas bahwa **doc i** lebih relevan dibandingkan **doc j**.

RankNet dilatih secara “pairwise”

Contoh format training data:

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

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
…	…

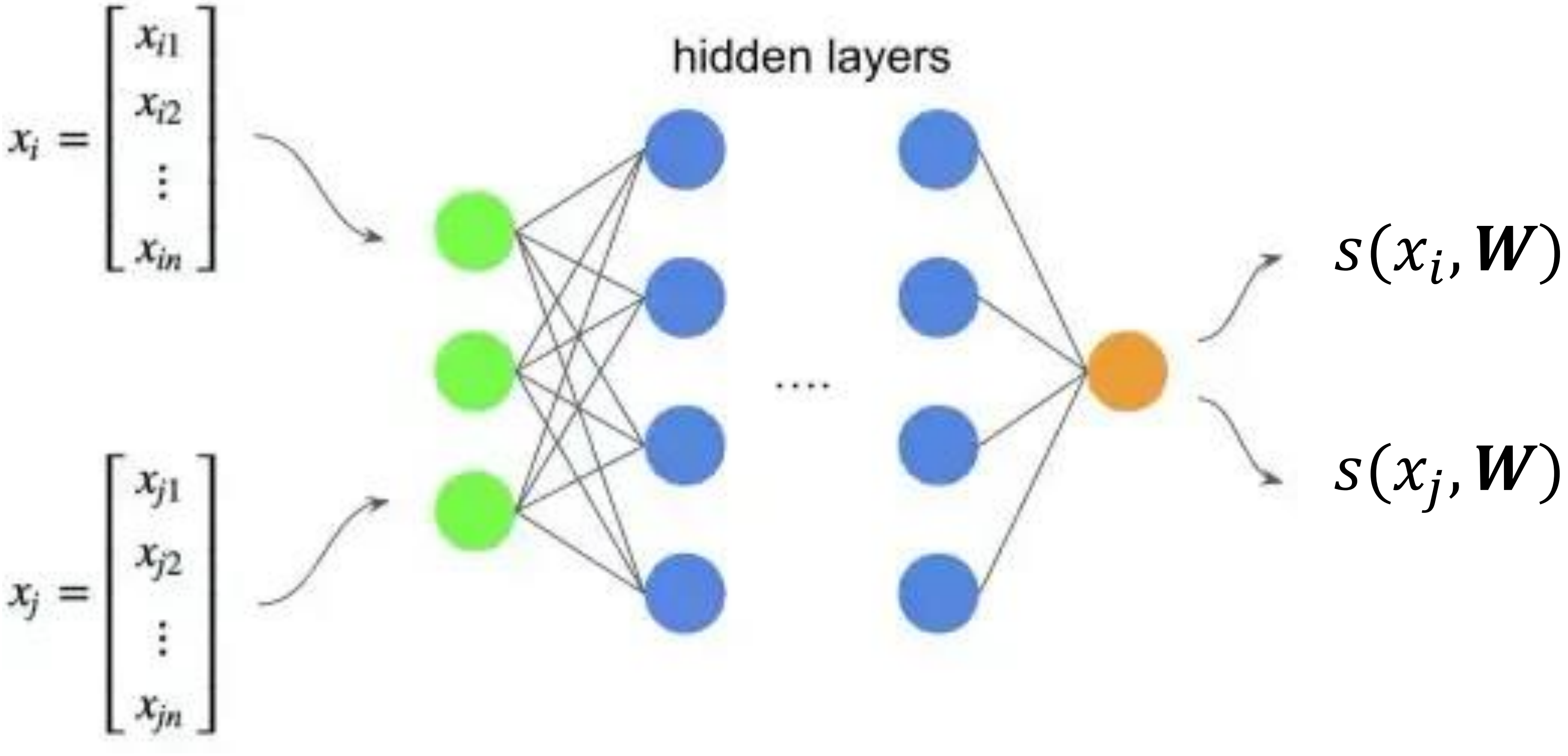
$$\hat{P}_{i,j} = \frac{1}{1 + \exp(-[s(x_i, \boldsymbol{W}) - s(x_j, \boldsymbol{W})])}$$

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”


$x_i = \begin{bmatrix} x_{i1} \\ x_{i2} \\ \vdots \\ x_{in} \end{bmatrix}$
$x_j = \begin{bmatrix} x_{j1} \\ x_{j2} \\ \vdots \\ x_{jn} \end{bmatrix}$
hidden layers
....
$s(x_i, \boldsymbol{W})$
$s(x_j, \boldsymbol{W})$

https://medium.com/swlh/ranknet-factorised-ranknet-lambdarank-explained-implementation-via-tensorflow-2-0-part-i-1e71d8923132

RankNet dilatih secara “pairwise”

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

$$= -P_{i,j}\left(s(x_i) - s\left(x_j\right)\right) + \log\left(1 + \exp\left(s(x_i) - s\left(x_j\right)\right)\right)$$

Dapat ditunjukkan bahwa:

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

Berlanjut ke halaman berikutnya …

RankNet's "Lambda" λ_{ij}

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

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

Predicted Prob – True Prob

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

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

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

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

RankNet's "Lambda" λ_{ij}

$$\boldsymbol{\lambda_{ij}}$$

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(xi)) 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 $\hat{p}_{11,12} = 0.8$

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

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

Training RankNet

Update Parameter dengan Stochastic Gradient Descent

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

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

λ_i = **Gradient**

"Lambda" w.r.t doc i -> λ_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**.

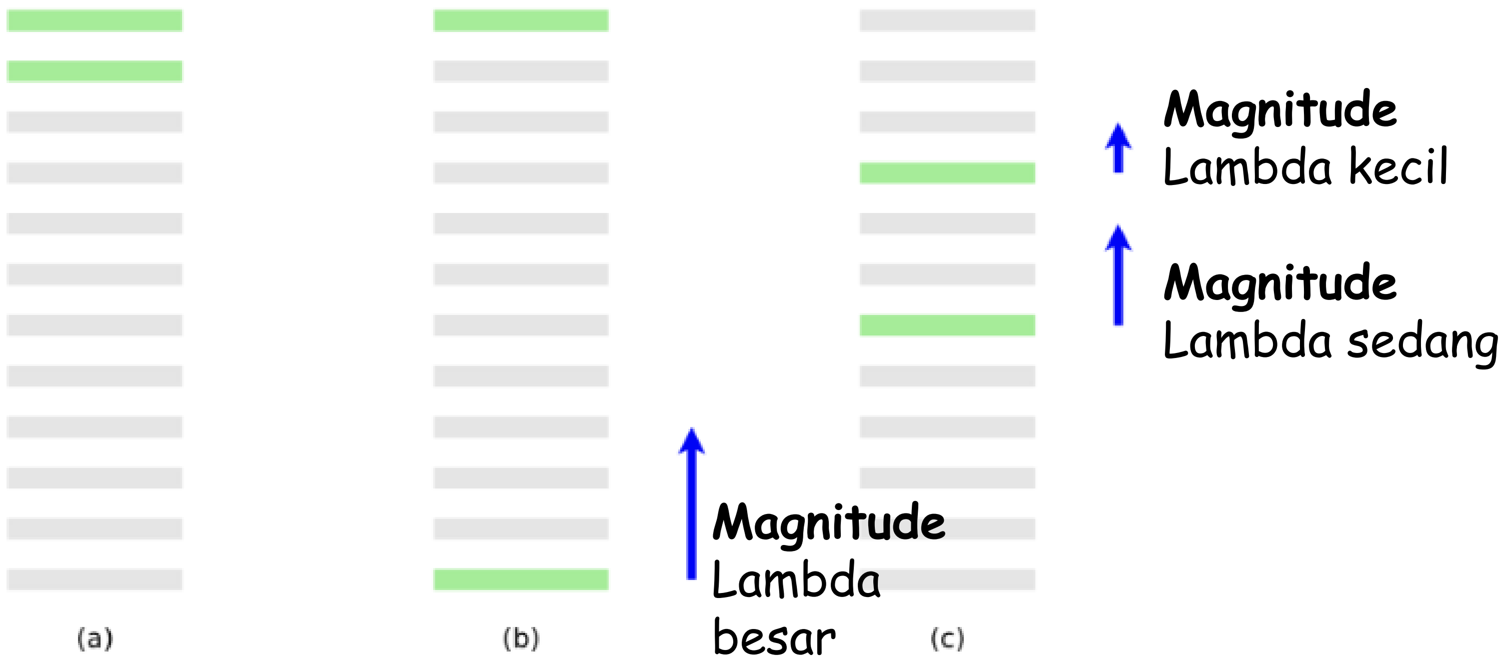
$$\lambda_i = \sum_{j:(i,j)\in I} \lambda_{ij} - \sum_{j:(j,i)\in I} \lambda_{ij}$$

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

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

“Lambda” w.r.t doc i -> λ_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



Magnitude Lambda kecil

Magnitude Lambda sedang

Magnitude Lambda besar

Inversion / pairwise error: sebuah pasangan “tak terurut”

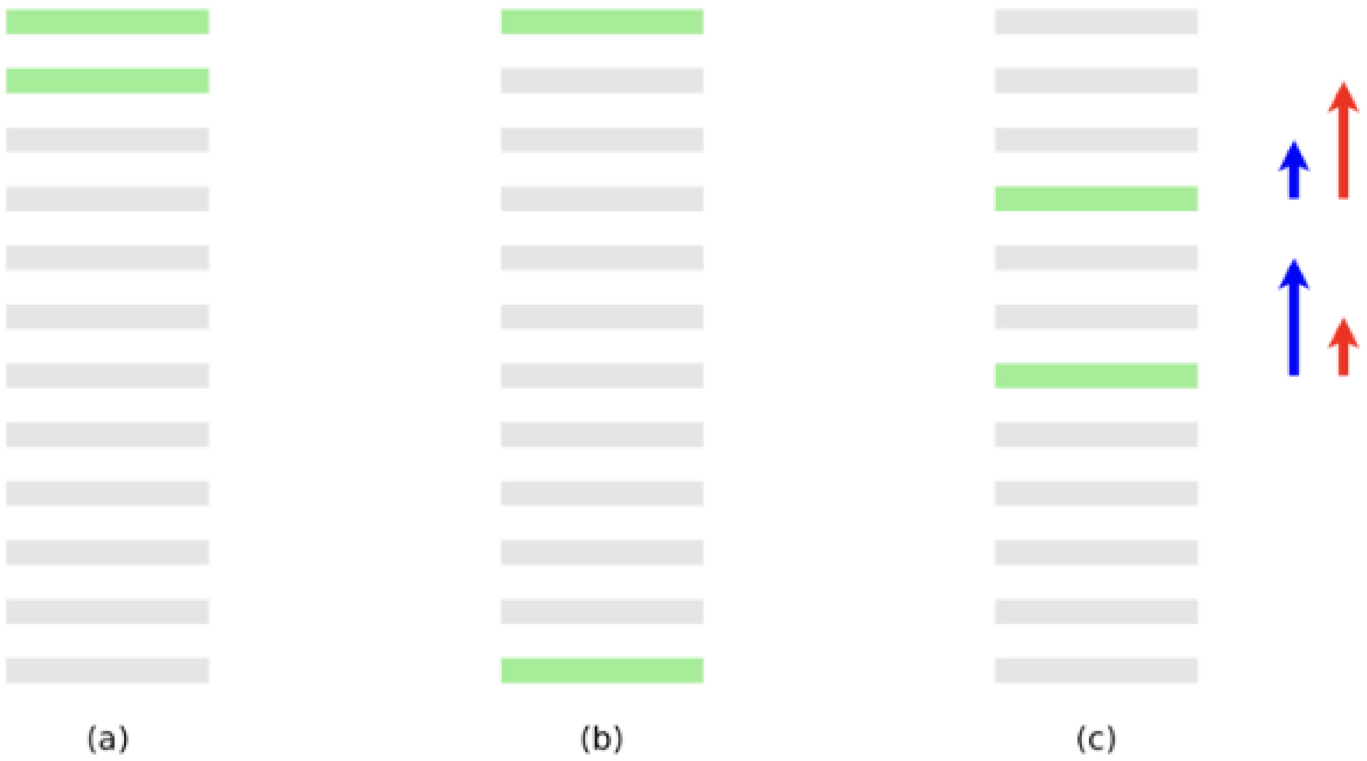
Prinsip training RankNet adalah meminimalkan *the number of inversions* pada ranking.

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

LambdaRank

Problem with RankNet's Lambda

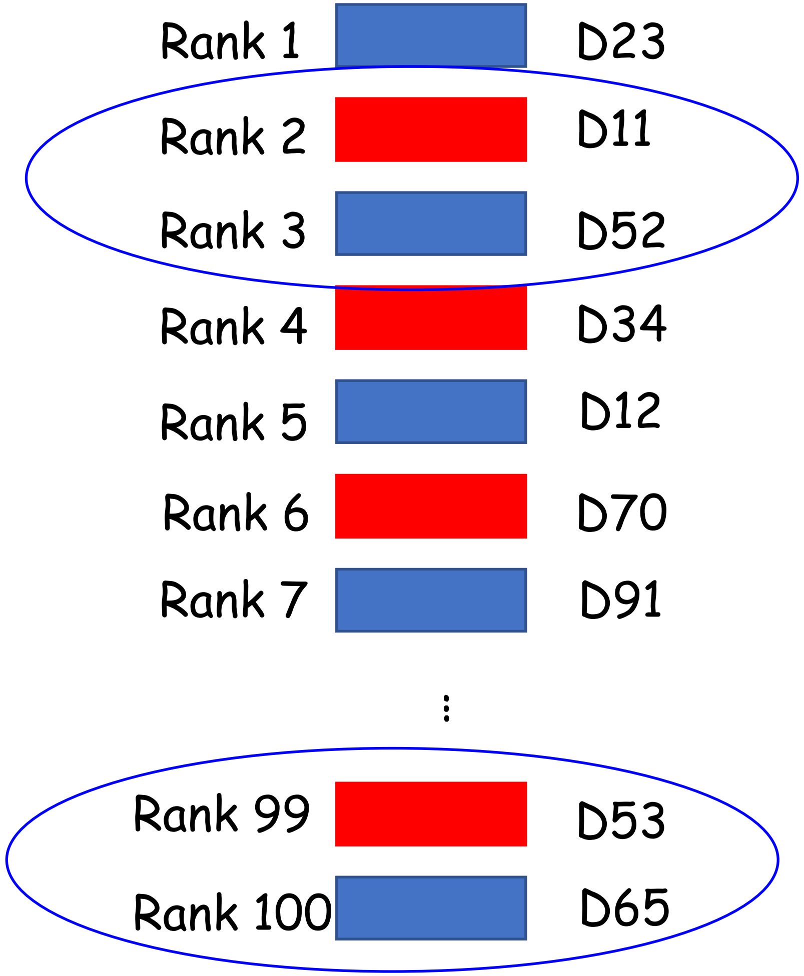
Problem: RankNet is based on pairwise error, while modern IR measures emphasize higher ranking positions. Red arrows show better λ's for modern IR, esp. web search.



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".

Dari Slide Chris Manning & Pandu Nayak, Learning-to-Rank, KuliahWeb 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.

$\mathbf{r}_{ideal}$ = [1, 1, 1, 0, 0]

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

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

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

ΔNDCG ketika dua dokumen di-swap

r = [0, 1, 0, 1, 1] DCG = 1.45 NDCG = 0.68

r = [1, 0, 0, 1, 1] DCG = 1.82 NDCG = 0.85

ΔNDCG = 0.17

r = [0, 1, 0, 1, 1] DCG = 1.45 NDCG = 0.68

r = [0, 1, 1, 0, 1] DCG = 1.52 NDCG = 0.71

ΔNDCG = 0.03

Semuanya mempunyai ranking ideal yang sama:

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

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

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

Lambda “RankNet“ vs Lambda “LambdaRank“

Lambda-nya RankNet

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

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

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

Agar “positionally aware”, dikali nilai **ΔNDCG** ketika **rank position doc i dan doc j di-swap**

Lambda “RankNet“ vs Lambda “LambdaRank“

Lambda-nya LambdaRank

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

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**:

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

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**.

Yang diambil dari LambdaRank & MART

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

$$\boldsymbol{L\left(y_i, F(x_i)\right) = \log(1 + \exp(-2y_iF(x_i)))}$$

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

Yang diambil dari LambdaRank & MART

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

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

Yang diambil dari LambdaRank & MART

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

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

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

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**.

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

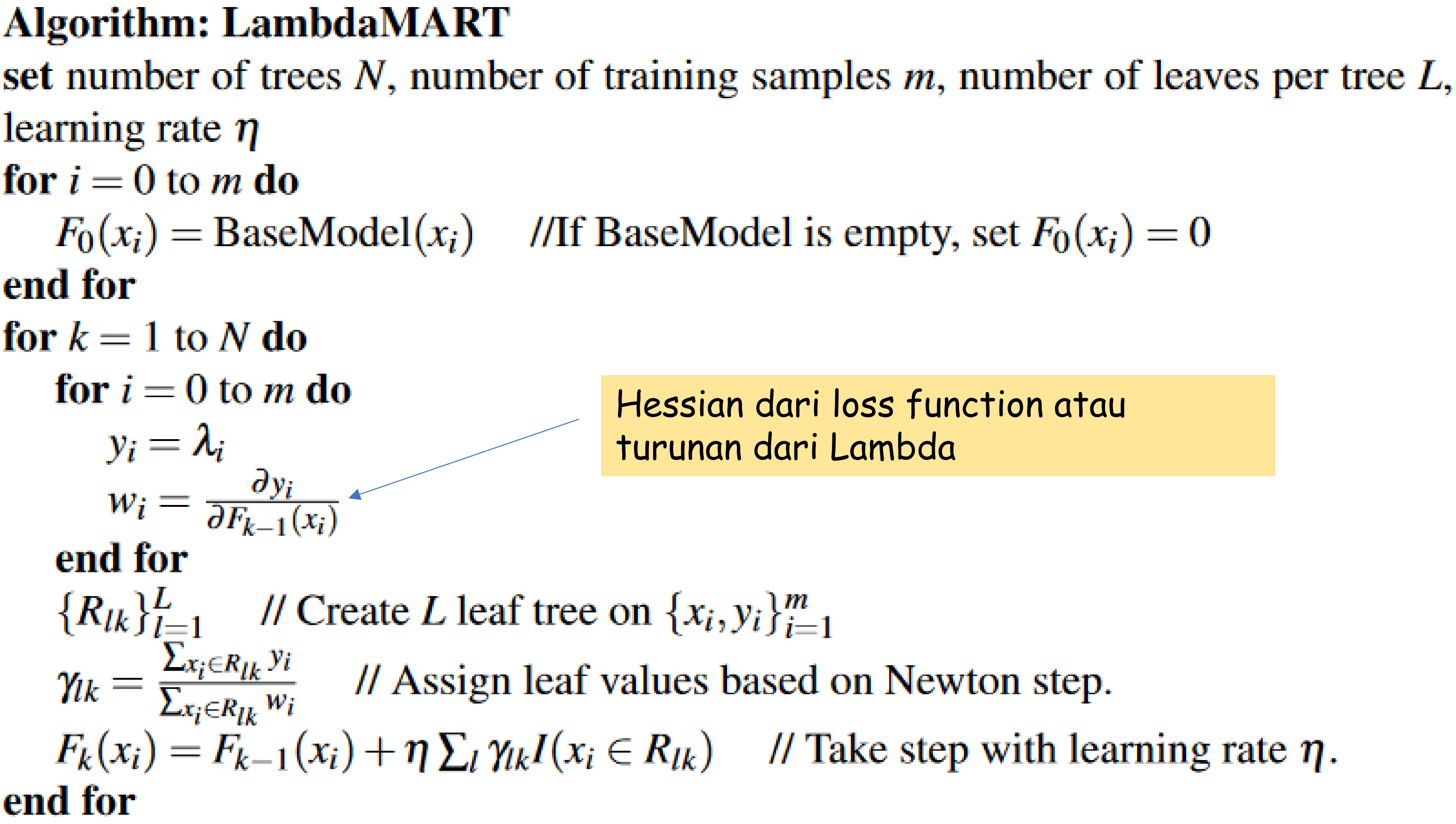
Gradient Descent

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

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 $\eta$
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
        $y_i = \lambda_i$
        $w_i = \frac{\partial y_i}{\partial F_{k-1}(x_i)}$
    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.
    $F_k(x_i) = F_{k-1}(x_i) + \eta \sum_l \gamma_{lk} I(x_i \in R_{lk})$    // Take step with learning rate $\eta$.
end for
```

Hessian dari loss function atau turunan dari Lambda