Pertemuan 4 — *Ensemble Learning*

Banyak pohon yang lemah, satu keputusan yang kuat

Hendri Karisma

Semester Ganjil 2026/2027

Program Studi Teknik Informatika, STMIK Tazkia

Alur pertemuan hari ini

1. Dari satu pohon ke banyak pohon
2. Bias dan *variance*: membongkar *error*
3. *Bootstrap* dan *bagging*
4. *Random forest*: OOB dan *feature importance*
5. *Boosting*: dari AdaBoost ke *gradient boosting*
6. XGBoost dan kerabatnya
7. Memilih dengan sadar

Sasaran pertemuan ini

Anda dapat menjelaskan mengapa sekumpulan model lemah bisa mengalahkan satu model kuat, membedakan mana teknik yang menurunkan ***variance*** dan mana yang menurunkan **bias**, menghitung sendiri satu iterasi AdaBoost, serta memilih antara pohon tunggal, *random forest*, dan XGBoost dengan alasan yang dapat dipertanggungjawabkan.

Dari satu pohon ke banyak pohon

Di mana kita berhenti pertemuan lalu

Decision tree adalah model yang paling mudah dibaca manusia: jalurnya dapat diucapkan sebagai kalimat. Tetapi ia juga model yang paling **tidak stabil**.

- Ganti 5% *training data*, dan atribut di akar bisa berubah. Ketika akar berubah, seluruh pohon di bawahnya berubah, dan aturan yang tadi kita jelaskan tidak lagi berlaku.
- Pohon yang tumbuh penuh mencapai *error* nol pada *training data*, lalu jatuh pada data baru.
- *Pruning* menolong, tetapi menukar satu masalah dengan masalah lain: pohon menjadi lebih stabil sekaligus lebih tumpul.

Ringkasan

Gagasan hari ini sederhana dan tidak intuitif sekaligus. Alih-alih membuat satu pohon menjadi sempurna, kita tanam **banyak** pohon yang masing-masing biasa-biasa saja, lalu menggabungkan pendapat mereka. Ketidakstabilan yang tadi jadi kelemahan justru menjadi bahan baku yang kita butuhkan.

Kebijaksanaan orang banyak, dihitung

Lima pemilih. Masing-masing benar berpeluang $p = 0{,}6$, dan kesalahan mereka **saling independen**. Keputusan diambil dengan suara terbanyak, jadi kita butuh sedikitnya 3 jawaban benar.

$$P(k \text{ benar}) = \binom{5}{k} 0{,}6^{k}\, 0{,}4^{5-k}$$

k	Peluang	Mayoritas?
0	0,0102	tidak
1	0,0768	tidak
2	0,2304	tidak
3	**0,3456**	**ya**
4	**0,2592**	**ya**
5	**0,0778**	**ya**
Jumlah tiga terakhir		**0,6826**

Satu pemilih benar 60% dari waktu; lima pemilih benar 68,3%. Tidak ada pemilih yang menjadi lebih pandai. Yang berubah hanya cara bertanyanya.

Rumus umumnya

Untuk B pemilih, B ganjil:

$$P_{\text{may}} = \sum_{k=\lceil B/2 \rceil}^{B} \binom{B}{k} p^{k} (1-p)^{B-k}$$

Notasi: $\binom{B}{k}$: banyak cara memilih k dari B; $\lceil \cdot \rceil$: pembulatan ke atas.

Menambah pemilih, dan dua syarat yang tidak boleh dilanggar

Satu pemilih benar	$B = 1$	$B = 3$	$B = 5$	$B = 11$	$B = 25$	$B = 51$	$B = 101$
$p = 0{,}70$	0,700	0,784	0,837	0,922	0,983	0,999	1,000
$p = 0{,}60$	0,600	0,648	0,683	0,753	**0,846**	0,926	0,979
$p = 0{,}55$	0,550	0,575	0,593	0,633	0,694	0,764	0,844
$p = 0{,}45$	0,450	0,425	0,407	0,367	0,306	0,236	0,156

Ke kanan pada tiga baris atas, peluang terdorong menuju 1. Ke kanan pada baris terakhir, terdorong menuju **0**. *Ensemble* memperkuat kecenderungan, apa pun arahnya.

Dua syarat

(1) Lebih baik daripada tebakan acak. Setiap anggota butuh $p > 0{,}5$, walau sedikit. Itulah arti *weak learner*: lemah, tetapi tidak nol. **(2) *Error*-nya tidak seragam.** Bila semua anggota salah pada kasus yang sama persis, kolom $B = 101$ akan sama dengan kolom $B = 1$. Sisa pertemuan ini sebenarnya tentang cara **merekayasa keberagaman** itu.

Bias dan *variance*

Membongkar *error* menjadi tiga bagian

Dekomposisi *bias–variance* untuk *squared error*

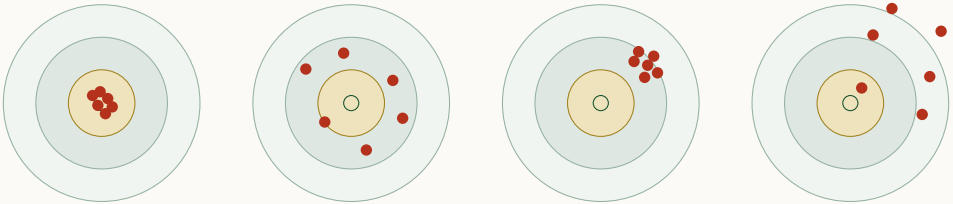
$$\mathbb{E}\left[(y - \hat{f}(x))^2\right] = \underbrace{\left(\mathbb{E}[\hat{f}(x)] - f(x)\right)^2}_{\text{bias}^2} + \underbrace{\mathbb{E}\left[(\hat{f}(x) - \mathbb{E}[\hat{f}(x)])^2\right]}_{\textit{variance}} + \underbrace{\sigma^2}_{\textit{noise}}$$

Notasi: σ^2 (*sigma* kuadrat): *variance noise*; $\mathbb{E}[\cdot]$: nilai harapan (*expected value*), rata-rata teoretis atas banyak *training data* yang berbeda.

Harapan $\mathbb{E}[\cdot]$ diambil atas **kemungkinan *training data* yang berbeda**. Bayangkan model yang sama dilatih pada seratus sampel data, lalu kita amati seratus *prediction*-nya pada satu titik uji.

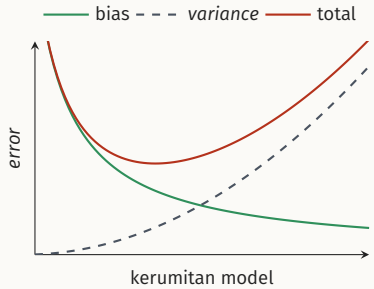
- **Bias** — seberapa jauh **rata-rata** seratus *prediction* itu dari kebenaran. Model yang terlalu sederhana salah secara sistematis.
- ***Variance*** — seberapa **menyebar** *prediction* itu satu terhadap yang lain. Pohon penuh: ganti data sedikit, *prediction* berubah banyak.
- ***Noise*** σ^2 — bagian yang tidak dapat dihilangkan siapa pun.

Empat kuadran sasaran



Titik merah adalah *prediction* model yang dilatih pada sampel data berbeda-beda; pusat sasaran adalah kebenaran. Perhatikan kuadran ketiga: ia **konsisten, tetapi konsisten salah**. Inilah tipe kesalahan yang paling sering lolos dari pemeriksaan, karena hasilnya terlihat stabil dan meyakinkan.

Siapa mengurangi apa



Titik terbaik ada di tengah, bukan di ujung mana pun.

Bagging menyerang *variance*

Melatih banyak model **kuat dan tidak stabil** secara sejajar, lalu merata-ratakan. Rata-rata banyak tebakan yang menyebar jauh lebih tenang daripada satu tebakan. Biasnya hampir tidak berubah.

Boosting menyerang bias

Menyusun banyak model **lemah dan kaku** secara berurutan, masing-masing memperbaiki sisa kesalahan pendahulunya. Satu tunggul pohon berbias besar; tiga ratus tunggul yang saling melengkapi tidak.

Akibatnya langsung terasa: pada *bagging*, menambah pohon nyaris tidak pernah merugikan. Pada *boosting*, menambah pohon terus-menerus akhirnya *overfit*, karena modelnya memang sedang dibuat semakin lentur.

Bootstrap dan *bagging*

Bootstrap: dari satu *training data* menjadi banyak

Kita hanya punya satu data berukuran n, tetapi butuh banyak yang berbeda. ***Bootstrap*** mengambil n baris secara acak **dengan pengembalian**: satu baris boleh terambil dua kali, sebagian tidak terambil sama sekali.

Peluang satu baris **tidak** terambil pada satu pengambilan adalah $1 - \frac{1}{n}$. Karena n pengambilan saling bebas,

$$P(\text{tak pernah terambil}) = \left(1 - \tfrac{1}{n}\right)^n$$

sehingga peluang sebuah baris **muncul** adalah $1 - \left(1 - \frac{1}{n}\right)^n$. Untuk n besar,

$$\left(1 - \tfrac{1}{n}\right)^n \xrightarrow[n\to\infty]{} e^{-1} \;\Rightarrow\; 1 - e^{-1} \approx 0{,}632$$

n	$(1 - 1/n)^n$	Baris unik
5	0,3277	67,2%
10	0,3487	65,1%
100	0,3660	63,4%
1.000	0,3677	63,2%
∞	$e^{-1} = 0{,}3679$	**63,2%**

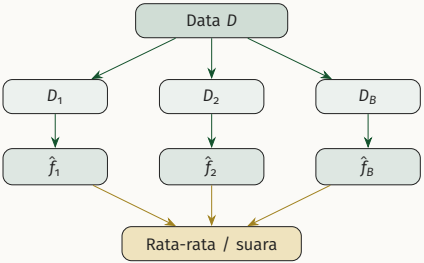
Ringkasan

Tiap sampel *bootstrap* memuat sekitar **63,2%** baris unik dan menyisakan **36,8%** yang tak pernah dilihat. Sisa itu bukan limbah — ia akan menjadi *validation data* gratis.

Bagging: bootstrap aggregating

1. Untuk $b = 1, \ldots, B$: ambil sampel *bootstrap* D_b berukuran n, lalu latih $\hat{f}_b$ pada D_b sampai tumbuh penuh, **tanpa** *pruning*.
2. Gabungkan:
   - *regression* — rata-rata, $\hat{f}_{\text{bag}}(x) = \frac{1}{B}\sum_b \hat{f}_b(x)$;
   - *classification* — suara terbanyak, atau rata-rata peluang kelas (*soft voting*), yang biasanya sedikit lebih baik karena memakai derajat keyakinan, bukan hanya putusannya.

Tiap pohon dibiarkan *overfit* dengan sengaja. Itu bukan kelalaian: *bagging* butuh anggota berbias rendah, dan *variance* besar yang menyertainya akan dibereskan oleh perata-rataan.



Semua cabang berjalan **sejajar** dan saling bebas, sehingga mudah diparalelkan ke banyak inti prosesor.

Mengapa perata-rataan menurunkan *variance*, dan sampai batas mana

Misalkan B model punya *variance* sama σ^2 dan korelasi antar pasangan ρ (*rho*).

Ringkasan

$$\operatorname{Var}\left(\frac{1}{B}\sum_{b=1}^{B}\hat{f}_b(x)\right) = \underbrace{\rho\,\sigma^2}_{\text{tak bisa dihilangkan}} + \underbrace{\frac{1-\rho}{B}\,\sigma^2}_{\text{lenyap bila } B \text{ besar}}$$

- Suku kedua turun seperti $1/B$: menambah pohon murah dan aman, tetapi melandai cepat. Dari 10 ke 100 pohon terasa; dari 500 ke 5.000 hampir tidak.
- Suku pertama **tidak peduli** berapa pun B. Bila $\rho = 0{,}8$, 80% *variance* asal tetap tinggal walau kita menanam sejuta pohon — dan sampel *bootstrap* bertumpang tindih sekitar 63%, jadi ρ memang tinggi.
- Untuk maju lebih jauh, yang dibutuhkan adalah pohon yang lebih **berbeda**, yaitu ρ yang lebih kecil. Menambah jumlah pohon saja tidak cukup.

Random forest

Bagging, ditambah keacakan di setiap *node*

Random forest (Breiman, 2001) menambah satu langkah **di dalam** pohon: pada setiap *node*, pilih dulu secara acak m dari d *feature*, lalu cari *best split* **hanya** di antara m *feature* itu.

- *Classification*: $m \approx \sqrt{d}$. Untuk $d = 100$, berarti 10 *feature* per *node*.
- *Regression*: $m \approx d/3$.
- Angka itu sekadar titik awal. m adalah *hyperparameter* paling berpengaruh pada *random forest* dan layak disetel.

Keacakannya diambil ulang di **setiap** *node*, bukan sekali per pohon.

Mengapa ini bekerja

Bila ada satu *feature* yang sangat kuat, *bagging* murni akan memilihnya sebagai akar pada hampir semua pohon. Hasilnya seratus pohon yang nyaris kembar: ρ tinggi, *variance* tidak banyak turun.

Dengan *feature selection* acak, *feature* kuat itu bahkan tidak selalu tersedia untuk dipertimbangkan. Pohon terpaksa menemukan jalan lain, dan *feature* menengah yang selalu tertutupi akhirnya mendapat kesempatan.

Harganya: tiap pohon menjadi sedikit **lebih lemah** sendiri-sendiri. Yang dibeli dengan pelemahan itu adalah ρ yang jauh lebih kecil.

Out-of-bag error: validasi yang tidak perlu dibeli

Setiap pohon tidak pernah melihat sekitar 36,8% baris. Baris itu *out-of-bag* (OOB) bagi pohon tersebut, dan bagi pohon itu ia sama asingnya dengan *test data*.

Prosedurnya

1. Untuk setiap baris i, kumpulkan pohon yang **tidak** memakai baris i saat dilatih — rata-rata sekitar $0{,}368\,B$ pohon.
2. Gabungkan *prediction* pohon-pohon itu untuk baris i.
3. Bandingkan dengan *label* sebenarnya, lalu rata-ratakan atas seluruh baris. Itulah **OOB *error***.

Nilainya sebanding dengan *cross-validation error*, tetapi diperoleh sebagai efek samping dari satu kali *training* — tanpa memisahkan data, tanpa melatih k kali.

Kegunaan praktisnya

- Menentukan kapan jumlah pohon cukup: gambar OOB *error* terhadap B, berhenti saat kurvanya mendatar.
- Menyetel m tanpa menyentuh *test data*.

Catatan kejujuran. OOB *error* agak pesimistis, karena tiap baris hanya dinilai oleh sepertiga hutan. Ia juga tidak sah bila data berurutan waktu atau berkelompok; untuk itu tetap pakai pembagian berbasis waktu atau *group-aware split*.

Feature importance: dua cara, dan dua peringatan

Penurunan *impurity* (MDI)

Jumlahkan penurunan *impurity* (Gini atau *entropy*) pada semua *node* yang memakai *feature* itu, dibobot jumlah baris yang melewatinya, lalu rata-ratakan atas seluruh pohon. Gratis: angkanya sisa perhitungan *training*. Inilah `feature_importances_`.

Permutasi

Acak ulang isi satu kolom pada *validation data*, lalu ukur berapa kinerja turun. Lebih mahal (*d* kali evaluasi ulang) tetapi lebih jujur, karena mengukur pengaruh pada **kinerja**, bukan pada struktur pohon.

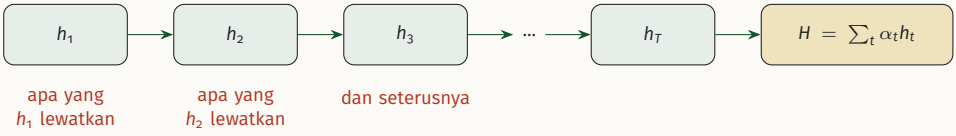
Dua peringatan

- **MDI berpihak pada kardinalitas tinggi.** Kolom dengan banyak nilai berbeda — ID pelanggan, kode pos, *timestamp* — punya lebih banyak titik pisah untuk dicoba, sehingga tampak penting walau isinya *noise*.
- ***Feature* berkorelasi saling mencuri kredit.** Bila dua kolom nyaris sama, keduanya tampak sedang-sedang saja. Permutasi pun tertipu di sini: membuang satu kolom tidak terasa karena kembarannya masih ada.

Feature importance adalah **petunjuk untuk diselidiki** dan tidak membuktikan sebab-akibat. *Feature* di puncak sering perlu diperiksa dulu: jangan-jangan ia bocoran dari *label* (*data leakage*).

Boosting: dari AdaBoost ke *gradient boosting*

Berurutan, bukan sejajar



Pada *bagging*, semua anggota menghadapi masalah yang **sama** dengan data berbeda, sehingga urutannya tidak penting dan semuanya bisa dilatih serentak. Pada *boosting*, setiap anggota menghadapi masalah yang **sudah berubah** — apa yang masih salah setelah pendahulunya bekerja — sehingga urutannya menentukan dan *training* antar pohon tidak bisa diparalelkan.

Ringkasan

Ada dua cara memberi tahu model berikutnya “apa yang masih salah”. **AdaBoost** memperbesar ***weight*** baris yang salah. ***Gradient boosting*** menyerahkan ***residual*** sebagai target baru. Keduanya dua wajah dari gagasan yang sama.

AdaBoost: algoritmanya

Label $y_i \in \{-1, +1\}$ dan $h_t(x) \in \{-1, +1\}$, sehingga $y_i h_t(x_i) = +1$ bila benar dan -1 bila salah. Itulah trik yang membuat rumusnya ringkas.

1. *Weight* awal seragam: $w_i^{(1)} = 1/n$.
2. Untuk $t = 1, \ldots, T$:
   - 2.1 latih h_t pada data yang diberi ***weight*** $w^{(t)}$ (biasanya satu *decision stump*);
   - 2.2 $\epsilon_t = \sum_{i:\, h_t(x_i) \neq y_i} w_i^{(t)}$ (ϵ: *epsilon*);
   - 2.3 $\alpha_t = \frac{1}{2} \ln \frac{1 - \epsilon_t}{\epsilon_t}$;
   - 2.4 $w_i^{(t+1)} = \frac{w_i^{(t)} \exp\big(-\alpha_t y_i h_t(x_i)\big)}{Z_t}$.
3. $H(x) = \operatorname{sign}\Big(\sum_{t=1}^{T} \alpha_t h_t(x)\Big)$.

Baris yang salah dikalikan $e^{+\alpha_t} > 1$ sehingga *weight*-nya naik; yang benar dikalikan $e^{-\alpha_t} < 1$ sehingga turun.

Membaca α_t

ϵ_t	α_t	arti
0,01	2,30	hampir sempurna
0,10	1,10	sangat berguna
0,30	0,42	berguna
0,50	0,00	tak bersuara
0,70	−0,42	dipakai terbalik

Model seburuk tebakan acak mendapat suara nol. Model yang **lebih buruk** daripada acak mendapat α negatif, artinya jawabannya dipakai secara terbalik — dan itu masih berguna.

Satu iterasi AdaBoost, dihitung sampai angkanya

$n = 10$ baris, *weight* awal $w_i^{(1)} = 0{,}1$ semuanya. Tunggul h_1 salah pada 3 baris.

(a) *Weighted error* $\epsilon_1 = 3 \times 0{,}1 = 0{,}3$

(b) $\alpha_1 = \frac{1}{2} \ln \frac{0{,}7}{0{,}3} = \frac{1}{2} \ln 2{,}3333 = \mathbf{0{,}4236}$

(c) Pengali *weight*

$$e^{+\alpha_1} = 1{,}5275 \qquad e^{-\alpha_1} = 0{,}6547$$

(d) Sebelum dinormalkan

	cacah	per baris	total
Salah	3	$0{,}1 \times 1{,}5275 = 0{,}15275$	0,45826
Benar	7	$0{,}1 \times 0{,}6547 = 0{,}06547$	0,45826
Z_1 = jumlah keduanya			**0,91652**

(e) Setelah dibagi Z_1

	per baris	jumlah kelompok
3 baris salah	$0{,}15275/0{,}91652 = 0{,}1667$	**0,5**
7 baris benar	$0{,}06547/0{,}91652 = 0{,}0714$	**0,5**

Ringkasan

Angka 0,5 dan 0,5 bukan kebetulan: dengan α_t yang dipilih demikian, *weight* total selalu terbagi **tepat separuh** antara baris yang salah dan yang benar.

Iterasi kedua, dan bentuk akhirnya

Tunggul h_2 dilatih pada *weight* baru. Katakanlah ia membenarkan ketiga baris sulit itu, tetapi tersandung pada 2 baris yang tadi benar.

$$\epsilon_2 = 2 \times 0{,}0714 = 0{,}1429$$

$$\alpha_2 = \tfrac{1}{2} \ln \frac{0{,}8571}{0{,}1429} = \tfrac{1}{2} \ln 6 = \mathbf{0{,}8959}$$

$$H(x) = \operatorname{sign}\left(0{,}4236\, h_1 + 0{,}8959\, h_2\right)$$

h_2 mendapat suara dua kali lebih besar — bukan karena ia model yang lebih baik secara umum, melainkan karena ia unggul pada soal yang **saat itu** dianggap penting.

Tiga hal yang perlu diingat

- *Training error ensemble* turun secara eksponensial terhadap T selama setiap $\epsilon_t < 0{,}5$; ini dijamin teorema.
- Bila sebuah tunggul mencapai $\epsilon_t = 0$ maka $\alpha_t \to \infty$; implementasi nyata berhenti di titik itu.
- AdaBoost meminimumkan *exponential loss* $\sum_i e^{-y_i F(x_i)}$ dengan skor $F(x) = \sum_t \alpha_t h_t(x)$, yang menghukum kesalahan sangat keras. Dari sinilah kepekaannya pada *outlier*: satu *label* salah tulis akan terus diperbesar *weight*-nya sampai mendominasi.

Gradient boosting: menuruni *gradient* di ruang fungsi

Pada pertemuan 9 kita memperbaiki **parameter** dengan $\theta \leftarrow \theta - \eta \nabla_\theta L$. *Gradient boosting* (Friedman, 2001) memperbaiki **fungsinya sendiri**:

Ringkasan

$$F_m(x) = F_{m-1}(x) + \nu \cdot h_m(x), \qquad h_m \approx -\left.\frac{\partial L}{\partial F}\right|_{F=F_{m-1}}$$

Notasi: ν (dibaca *nu*); $\partial L/\partial F$: *partial derivative* L terhadap F.

Satu pohon menggantikan satu langkah *gradient*. Untuk $L = \frac{1}{2}(y - F)^2$, arah negatif *gradient*-nya persis *residual*: $-\partial L/\partial F = y - F_{m-1}(x) = r$.

- Karena itu resepnya sesederhana ini: hitung *residual*, latih pohon untuk **memprediksi *residual***, tambahkan sebagiannya ke model, ulangi.
- Keunggulan cara pandang *gradient* adalah keleluasaannya. Ganti L dengan *log loss* dan yang diprediksi pohon menjadi $y_i - p_i$: kita dapat *boosting* untuk *classification*. Ganti dengan *Huber loss* untuk data penuh *outlier*, atau *pinball loss* untuk *regression* kuantil. Kerangkanya tidak berubah.

Residual sebagai target: satu langkah dengan angka nyata

Satu *feature* berurutan, target $y = [12,\ 15,\ 21,\ 30,\ 34]$, $\nu = 0{,}1$, pohon dengan satu pemisah saja.

i	y_i	$F_0 = \bar{y}$	$r_i = y_i - F_0$	F_1
1	12	22,4	−10,4	21,76
2	15	22,4	−7,4	21,76
3	21	22,4	−1,4	21,76
4	30	22,4	7,6	23,36
5	34	22,4	11,6	23,36

h_1 memisah antara baris 3 dan 4, lalu mengisi tiap *leaf* dengan rata-rata *residual* di *leaf* itu: $-6{,}4$ dan $+9{,}6$. Pembaruannya $F_1 = F_0 + 0{,}1 \times h_1$, jadi $22{,}4 - 0{,}64 = 21{,}76$ dan $22{,}4 + 0{,}96 = 23{,}36$.

Apakah kita maju?

Jumlah kuadrat *residual* turun dari **357,20** (F_0) menjadi **298,83** (F_1) — 16% dari satu pohon yang isinya hanya dua angka. *Residual* baru menjadi $[-9{,}76,\ -6{,}76,\ -0{,}76,\ 6{,}64,\ 10{,}64]$, dan pohon kedua dilatih pada deret itu.

Langkahnya **disengaja kecil**. Tanpa faktor 0,1, *leaf* kiri langsung menembak $22{,}4 - 6{,}4 = 16$ dan pohon pertama sudah menghabiskan hampir seluruh sinyal — termasuk *noise*-nya.

Learning rate, dan mengapa kita menahan diri

ν (`learning_rate` atau `eta`) mengecilkan kontribusi setiap pohon. Efeknya seperti *regularization*: tidak ada satu pohon pun yang boleh menentukan hasil sendirian, sehingga *noise* yang tertangkap satu pohon masih bisa dikoreksi pohon berikutnya.

- Nilai lazim: 0,01 sampai 0,3.
- ν dan jumlah pohon M saling menukar: setengahkan ν, dan M perlu kira-kira dua kali lipat.
- ν kecil dengan M besar biasanya lebih baik, dengan *time cost* latih lebih panjang — pertukaran **efektivitas–efisiensi** dari Pertemuan 1, kali ini sebagai satu baris *configuration*.

Cara menyetelnya yang praktis

1. Tetapkan ν kecil, misalnya 0,05.
2. Pasang jumlah pohon besar, misalnya 5.000.
3. Nyalakan *early stopping* pada satu himpunan validasi: berhenti bila tidak membaik selama 50 putaran.
4. Biarkan data yang memutuskan berapa pohon yang dibutuhkan.

Ringkasan

Tanpa *early stopping*, jumlah pohon pada *boosting* adalah *hyperparameter* yang berbahaya: menambahnya terus menurunkan *training error* sampai nol sekaligus menaikkan *test error*. Pada *random forest*, kesalahan semacam ini praktis tidak terjadi (RF tetap bisa *overfit* lewat pohon yang terlalu dalam).

XGBoost dan kerabatnya

Objektif yang menyebut ukuran pohon, dan pendekatan orde dua

Gradient boosting klasik mengendalikan kerumitan dari luar. XGBoost (Chen dan Guestrin, 2016) memasukkannya **ke dalam objektif**, lalu menguraikan rugi dengan Taylor sampai orde kedua ($g_i = \partial_{\hat{y}} l$, $h_i = \partial^2_{\hat{y}} l$):

$$\Omega(f) = \gamma T + \tfrac{1}{2}\lambda \|w\|^2 \qquad \mathcal{L}^{(t)} \approx \sum_i \left[g_i f_t(x_i) + \tfrac{1}{2} h_i f_t(x_i)^2 \right] + \gamma T + \tfrac{1}{2}\lambda \sum_j w_j^2$$

Notasi: Ω (*omega* kapital), γ (*gamma*), λ (*lambda*); $\|w\|$: panjang (norma) vektor w.

Dengan $G_j = \sum_{i \in I_j} g_i$ dan $H_j = \sum_{i \in I_j} h_i$, untuk struktur pohon yang tetap ini kuadratik sederhana pada w_j, sehingga optimumnya dapat ditulis langsung:

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

- T = jumlah *leaf*, γ = *cost* per *leaf* tambahan; λ menarik nilai *leaf* ke nol, sehingga *leaf* berpenghuni sedikit tidak boleh memberi *prediction* ekstrem.
- Memakai h_i berarti langkahnya tahu **kelengkungan** rugi, bukan hanya arahnya; karena itu pohon yang dibutuhkan lebih sedikit. Dan $\mathcal{L}^*$ memberi **skor mutu** sebuah struktur pohon.

Split gain, dihitung dua kali

$$\text{Gain} = \frac{1}{2}\left[\frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{(G_L+G_R)^2}{H_L+H_R+\lambda}\right] - \gamma$$

Kasus A — *split* diterima

$G_L = -6,\ H_L = 3,\ G_R = 8,\ H_R = 4,\ \lambda = 1,\ \gamma = 0{,}5$

$$\text{kiri} = 36/4 = 9, \quad \text{kanan} = 64/5 = 12{,}8$$
$$\text{induk} = 2^2/8 = 0{,}5$$
$$\text{Gain} = \tfrac{1}{2}(9 + 12{,}8 - 0{,}5) - 0{,}5 = \mathbf{10{,}15}$$

Kedua sisi menarik ke arah berlawanan; memisahkan jelas menguntungkan.

Kasus B — *split* ditolak

$G_L = -1,\ H_L = 3,\ G_R = 1,\ H_R = 3,\ \lambda = 1,\ \gamma = 0{,}5$

$$\text{kiri} = 1/4 = 0{,}25, \quad \text{kanan} = 1/4 = 0{,}25$$
$$\text{induk} = 0^2/7 = 0$$
$$\text{Gain} = \tfrac{1}{2}(0{,}25 + 0{,}25 - 0) - 0{,}5 = \mathbf{-0{,}25}$$

Perbaikannya 0,25, harga satu *leaf* 0,5. Cabang tidak dibuat.

Naikkan γ dan pohon memendek dengan sendirinya — *pruning* menjadi akibat langsung dari objektif, tanpa langkah terpisah. Naikkan λ dan nilai *leaf* mengerut, terutama pada *leaf* berpenghuni sedikit, karena H_j-nya kecil sehingga λ mendominasi penyebutnya.

Mengapa cepat, dan bagaimana ia memperlakukan nilai hilang

Yang membuatnya cepat

- **Blok terurut.** Data disusun per kolom dalam blok terkompresi yang sudah terurut, sehingga tidak perlu mengurutkan ulang di setiap *node*.
- **Histogram.** Nilai dikelompokkan ke beberapa ratus *bin* (`tree_method=hist`), sehingga calon pemisah berkurang dari n menjadi jumlah *bin*.
- **Paralel dan sadar *cache*.** Pohonnya berurutan, tetapi pencarian pemisah di dalam satu pohon dibagi ke banyak inti.

Yang menahan *overfit*

`eta` (shrinkage) pada setiap pohon; `subsample` untuk mengambil sebagian baris per pohon, meminjam gagasan *bagging*; `colsample_bytree` dan `colsample_bylevel` untuk mengambil sebagian kolom, meminjam gagasan *random forest*.

Missing value

Tidak ada imputasi. Pada setiap *node*, XGBoost mencoba mengirim seluruh baris kosong ke kiri, lalu ke kanan, dan menyimpan pilihan dengan gain lebih besar sebagai **arah bawaan** *node* itu. "Kosong" diperlakukan sebagai informasi yang dipelajari, bukan lubang yang harus ditambal.

Tiga kerabat yang akan Anda temui

XGBoost

Tumbuh per tingkat (*level-wise*), *regularization* eksplisit, matang dan portabel. Pilihan bawaan yang aman, dan yang kita pakai di praktikum.

LightGBM

Tumbuh per *leaf* (*leaf-wise*): selalu memecah *leaf* dengan gain terbesar. Lebih cepat pada data besar, tetapi lebih mudah *overfit* pada data kecil sehingga `num_leaves` harus dijaga.

CatBoost

Menangani *feature* kategorikal secara bawaan dengan *target statistics* yang diurutkan, dan memakai *ordered boosting* untuk menekan kebocoran *prediction*.

Ringkasan

Ketiganya *gradient boosted trees* yang berbeda pada rekayasanya; algoritma intinya sama. Semua yang Anda pelajari hari ini — *residual*, *shrinkage*, *subsampling*, *early stopping* — berlaku untuk ketiganya. Yang berbeda hanya nama *hyperparameter*-nya.

Sampai hari ini, pada data **tabular** berukuran menengah, keluarga inilah yang umumnya masih mengalahkan *neural network*. Gambar, teks, dan audio adalah cerita lain, dan itu bagian setelah UTS.

Memilih dengan sadar

Tabel perbandingan

	Pohon tunggal	***Random forest***	***Gradient boosting* / XGBoost**
Accuracy khas	Paling rendah	Baik, sering cukup	Terbaik pada data tabular
Kestabilan	Rendah; ganti sedikit data, berubah banyak	Tinggi; diredam oleh perata-rataan	Menengah; peka pada ν, kedalaman, jumlah pohon
Keterbacaan	Penuh; jalurnya dapat diucapkan sebagai kalimat	Hilang; tersisa *feature importance*	Hilang; butuh SHAP atau *partial dependence*
Training cost	Paling murah	B kali pohon tunggal, tetapi sejajar	Mahal dan berurutan, ditambah pencarian *hyperparameter*
Inference cost	O(kedalaman)	$O(B \times$ kedalaman$)$	$O(M \times$ kedalaman$)$, pohon lebih pendek
Hyperparameter utama	kedalaman, `min_samples_leaf`	B, `max_features` (m)	`eta`, `max_depth`, γ, λ, `subsample`, jumlah pohon
Kapan dipilih	Ketika aturannya harus dapat dijelaskan dan diaudit	*Baseline* pertama, atau data *noisy*	*Accuracy* dikejar dan Anda punya waktu menyetel

Peringatan praktis, dan *cost* yang lupa dihitung

- ***Boosting* lebih rentan pada data *noisy*.** Ia dirancang mengejar baris yang sulit. *Label* yang salah tulis juga baris yang sulit, dan akan dikejar dengan sama tekunnya sampai dihafal.
- ***Random forest* hampir selalu aman.** Jalankan dengan pengaturan bawaan sebagai *baseline*. Bila XGBoost yang disetel berhari-hari hanya menang setengah persen, perbaikan berikutnya lebih mungkin datang dari data dan *feature*-nya.
- ***Ensemble* menukar keterbacaan dengan *accuracy*.** Pada wilayah yang diatur regulasi — pemberian kredit, keputusan medis — pertukaran itu mungkin tidak boleh dilakukan, seberapa pun bagus angkanya.

Kembali ke Pertemuan 1

Kita sudah membedakan **efektivitas** dari **efisiensi**. *Ensemble* adalah contoh paling jelas dari pertukaran itu, dan *cost*-nya ada di tiga tempat:

- ***training time*** — *B* atau *M* kali satu pohon;
- ***inference time*** — setiap *prediction* menyusuri ratusan pohon, dan inilah yang menabrak anggaran milidetik di produksi;
- **waktu manusia** — menyetel, menjelaskan, dan merawat model yang tidak bisa dibaca langsung.

Karena itu “pakai XGBoost saja” bukan jawaban rekayasa. Jawaban rekayasa menyebut angka: berapa *accuracy* yang dibutuhkan, berapa milidetik yang tersedia, dan seberapa jauh keputusannya harus dapat dipertanggungjawabkan.

Tugas untuk pertemuan berikutnya

1. Pilih satu *dataset* tabular berlabel (*Adult Income*, *Telco Churn*, atau data mitra dengan izin). Bagi menjadi *training* dan *test*, dan jangan sentuh *test data* sampai langkah terakhir.
2. Latih **tiga** model: *decision tree* tunggal, *random forest*, dan XGBoost. Laporkan *accuracy*, *precision*, *recall*, F1, AUC, serta *training time* dan *inference time* masing-masing.
3. Untuk *random forest*, laporkan **OOB *error***, bandingkan dengan *test error*, dan gambar OOB *error* terhadap jumlah pohon. Untuk XGBoost, pakai *early stopping* dan sebutkan berapa pohon akhirnya terpakai.
4. Bandingkan *feature importance* MDI dengan *permutation importance*. Tunjukkan satu *feature* yang peringkatnya berbeda, lalu jelaskan dugaan penyebabnya.
5. Tulis analisis **satu halaman**: mengapa hasil ketiga model berbeda, dan mana yang Anda pilih bila anggaran inferensinya 10 ms per permintaan.

Bacaan

- Hastie, Tibshirani, Friedman, *The Elements of Statistical Learning*, Bab 10 dan 15.
- Chen dan Guestrin, *XGBoost: A Scalable Tree Boosting System*, KDD 2016.
- Géron, *Hands-On Machine Learning*, Bab 7, untuk sisi kodenya.

Pertemuan depan

Kita masuk ke *Bayesian learning*: teorema Bayes sebagai cara memperbarui keyakinan dari data, naive Bayes sebagai *classifier* yang murah, lalu GMM dan algoritma EM untuk data tanpa label.

Terima kasih

Pertanyaan?