Pertemuan 3 — *Decision Tree*

Entropy, *information gain*, dan jalan dari ID3 ke C4.5/J48

Hendri Karisma

Semester Ganjil 2026/2027

Program Studi Teknik Informatika, STMIK Tazkia

Alur pertemuan hari ini

1. Mengapa *decision tree*: model yang bisa dibaca
2. *Dataset* PlayTennis milik Mitchell
3. *Entropy*: mengukur ketidakmurnian
4. *Information gain*, dihitung dengan tangan
5. Algoritma ID3 dan pohon PlayTennis
6. *Hypothesis space* dan *inductive bias* ID3
7. Kelemahan gain, lalu C4.5/J48
8. *Overfitting* dan *pruning*
9. *Time cost* latih dan waktu jawab

Sasaran pertemuan ini

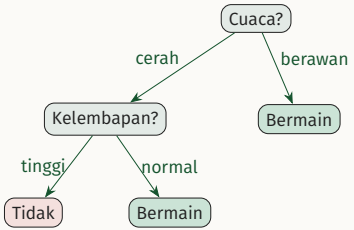
Pertemuan lalu kita melihat bahwa setiap pembelajar punya ***hypothesis space*** dan tidak dapat belajar tanpa ***inductive bias***. Hari ini kita memakai pembelajar dengan bias yang berbeda dan jauh lebih praktis: *decision tree* lebih menyukai **pohon yang pendek**. Anda akan mampu menghitung *entropy* dan *information gain* dengan tangan, menjalankan ID3 sampai pohonnya jadi, dan menjelaskan mengapa pohon perlu dipangkas.

Mengapa *decision tree*

Model yang bisa dibaca manusia

Sebagian besar model ML menjawab tanpa mau bercerita. *Decision tree* justru sebaliknya: seluruh pengetahuannya terbaca sebagai rangkaian pertanyaan. Setiap *internal node* bertanya tentang satu atribut, setiap cabang adalah salah satu jawabannya, dan setiap *leaf* memberi keputusan.

Satu lintasan dari *root* ke *leaf* dapat dibaca langsung menjadi satu aturan *if-then*, sehingga pohon utuh setara dengan sehimpunan aturan yang saling melengkapi.



Dibaca menjadi aturan:

- `JIKA` cuaca = cerah `DAN` kelembapan = tinggi `MAKA` tidak bermain
- `JIKA` cuaca = berawan `MAKA` bermain

Karena itu *decision tree* sering dipakai justru di tempat yang menuntut pertanggungjawaban: penilaian kredit, triase medis, dan audit kepatuhan.

Dataset PlayTennis: contoh yang akan kita pakai seharian

Hari	Outlook	Temp.	Humidity	Wind	Play
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Empat belas baris, empat atribut kategorikal, satu target biner. Kecil, tetapi cukup untuk dihitung dengan tangan sampai selesai.

Nilai tiap atribut:

- Outlook: Sunny, Overcast, Rain
- Temperature: Hot, Mild, Cool
- Humidity: High, Normal
- Wind: Weak, Strong

Sasarannya: 9 Yes dan 5 No.

Dua keputusan algoritma

Atribut mana yang ditanyakan lebih dulu, dan kapan berhenti bertanya.

Sumber: Mitchell, *Machine Learning*, 1997, Tabel 3.2

Entropy dan *information gain*

Entropy: seberapa campur aduk sebuah himpunan

Kita butuh satu angka yang mengatakan seberapa **tidak murni** sekumpulan contoh. Untuk himpunan S dengan c kelas dan proporsi p_i pada kelas ke-i:

$$H(S) \;=\; -\sum_{i=1}^{c} p_i \log_2 p_i$$

Notasi: $\sum_{i=1}^{c}$ (sigma kapital): jumlahkan suku di kanannya untuk $i = 1, \ldots, c$; $\log_2$: logaritma basis 2.

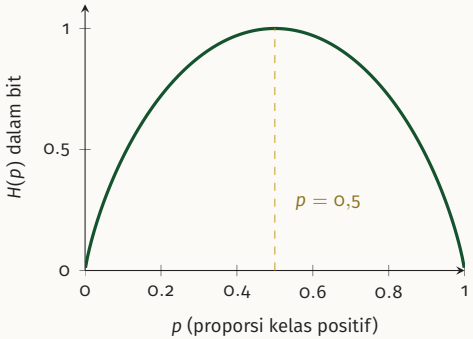
Satuannya **bit**, dan tafsirannya harfiah: rata-rata jumlah bit yang dibutuhkan untuk menyampaikan kelas satu contoh yang diambil acak dari S.

- Semua contoh satu kelas $\Rightarrow H = 0$ (tidak ada yang perlu diberitahukan)
- Dua kelas seimbang $\Rightarrow H = 1$ (paling tidak pasti)
- Kesepakatan: $0 \log_2 0$ dianggap 0

Ringkasan

Entropy tidak mengukur benar atau salah. Ia mengukur **ketidakpastian**. Itulah sebabnya ia cocok dipakai sebagai kompas: kita mencari pertanyaan yang paling banyak mengurangi ketidakpastian.

Entropy pada kasus dua kelas



Kurvanya cekung, simetris di $p = 0{,}5$, dan menyentuh nol di kedua ujung.

Komposisi	p	H
$[9+,\ 0-]$	1,000	0,000
$[6+,\ 1-]$	0,857	0,592
$[6+,\ 2-]$	0,750	0,811
$[2+,\ 3-]$	0,400	0,971
$[3+,\ 4-]$	0,429	0,985
$[3+,\ 3-]$	0,500	1,000

Perhatikan bahwa kurvanya datar di sekitar puncak: perbaikan dari $[3+, 4-]$ ke $[2+, 3-]$ kecil, sedangkan menjelang ujung penurunannya tajam.

Entropy PlayTennis sebelum satu pertanyaan pun diajukan

Himpunan lengkap S berisi 14 contoh: 9 Yes dan 5 No, jadi $[9+, 5-]$.

$$\begin{aligned} H(S) &= -\frac{9}{14}\log_2\frac{9}{14} \;-\; \frac{5}{14}\log_2\frac{5}{14} \\ &= -0{,}643 \times (-0{,}637) \;-\; 0{,}357 \times (-1{,}485) \\ &= 0{,}410 + 0{,}530 \;=\; \mathbf{0{,}940}\ \text{bit} \end{aligned}$$

Angka 0,940 inilah titik tolaknya. Menebak “Yes” terus menerus memberi 9 dari 14 benar, dan sisa ketidakpastiannya sebesar 0,940 bit itulah yang hendak kita kikis dengan bertanya.

Ringkasan

Kalau $H(S)$ sudah nol, tidak ada yang perlu dipelajari: satu *leaf* bertuliskan satu kelas sudah cukup. Semua pekerjaan berikutnya adalah usaha menurunkan angka ini secepat mungkin.

Information gain: berapa bit yang dihemat oleh satu pertanyaan

Bila S dipecah menurut atribut A, tiap nilai v melahirkan satu himpunan anak S_v. Ketidakpastian yang tersisa adalah rata-rata *entropy* anak, ditimbang ukurannya:

$$Gain(S, A) \;=\; H(S) \;-\; \underbrace{\sum_{v \in \mathrm{Nilai}(A)} \frac{|S_v|}{|S|}\, H(S_v)}_{\text{\textit{entropy} sisa setelah bertanya } A}$$

- Selalu ≥ 0: memecah tidak pernah menambah ketidakpastian rata-rata
- $Gain = 0$ berarti atribut itu tidak memberi keterangan apa pun
- Penimbang $|S_v|/|S|$ penting: cabang yang hampir kosong tidak boleh ikut menentukan

Ringkasan

Bacalah rumus ini sebagai satu pertanyaan praktis: **kalau saya bertanya tentang A sekarang, seberapa banyak kebingungan saya berkurang?** Atribut dengan jawaban terbesar itulah yang dipasang di *node* ini.

Hitung rinci: $Gain(S, \textbf{Humidity})$

Humidity punya dua nilai. Kita pisahkan keempat belas contohnya:

High (7 contoh): D1, D2, D3, D4, D8, D12, D14
Yes: D3, D4, D12 No: D1, D2, D8, D14 $\Rightarrow [3+, 4-]$

$$H(S_{\text{High}}) = -\tfrac{3}{7}\log_2\tfrac{3}{7} - \tfrac{4}{7}\log_2\tfrac{4}{7}$$
$$= 0{,}524 + 0{,}461 = 0{,}985$$

Normal (7 contoh): D5, D6, D7, D9, D10, D11, D13
Yes: D5, D7, D9, D10, D11, D13 No: D6 $\Rightarrow [6+, 1-]$

$$H(S_{\text{Normal}}) = -\tfrac{6}{7}\log_2\tfrac{6}{7} - \tfrac{1}{7}\log_2\tfrac{1}{7}$$
$$= 0{,}191 + 0{,}401 = 0{,}592$$

$$Gain(S, \text{Humidity}) = 0{,}940 - \tfrac{7}{14}(0{,}985) - \tfrac{7}{14}(0{,}592) = 0{,}940 - 0{,}493 - 0{,}296 = \mathbf{0{,}151}$$

Cabang Normal hampir murni, cabang High masih separuh-separuh. Rata-ratanya tetap jauh di bawah 0,940, jadi bertanya soal kelembapan memang berguna.

Hitung rinci: $Gain(S, \textbf{Wind})$

Weak (8 contoh): D1, D3, D4, D5, D8, D9, D10, D13
Yes: D3, D4, D5, D9, D10, D13 No: D1, D8 $\Rightarrow [6+, 2-]$

$$H(S_{\text{Weak}}) = -\tfrac{6}{8}\log_2\tfrac{6}{8} - \tfrac{2}{8}\log_2\tfrac{2}{8} = 0{,}311 + 0{,}500 = 0{,}811$$

Strong (6 contoh): D2, D6, D7, D11, D12, D14
Yes: D7, D11, D12 No: D2, D6, D14 $\Rightarrow [3+, 3-]$

$$H(S_{\text{Strong}}) = -\tfrac{3}{6}\log_2\tfrac{3}{6} - \tfrac{3}{6}\log_2\tfrac{3}{6} = 0{,}500 + 0{,}500 = 1{,}000$$

$$Gain(S, \text{Wind}) = 0{,}940 - \tfrac{8}{14}(0{,}811) - \tfrac{6}{14}(1{,}000) = 0{,}940 - 0{,}463 - 0{,}429 = \mathbf{0{,}048}$$

Ringkasan

Cabang Strong terbelah persis separuh, sehingga entropy-nya maksimal dan hampir tidak menyumbang apa pun. Inilah wujud atribut yang lemah: memecah data tetapi tidak memperjelas jawabannya.

Keempat atribut dibandingkan pada akar

Atribut	Pecahannya	*Entropy* anak	*Entropy* sisa	Gain
Outlook	Sunny $[2+, 3-]$, Overcast $[4+, 0-]$, Rain $[3+, 2-]$	0,971 / 0,000 / 0,971	0,694	**0,246**
Humidity	High $[3+, 4-]$, Normal $[6+, 1-]$	0,985 / 0,592	0,789	0,151
Wind	Weak $[6+, 2-]$, Strong $[3+, 3-]$	0,811 / 1,000	0,892	0,048
Temperature	Hot $[2+, 2-]$, Mild $[4+, 2-]$, Cool $[3+, 1-]$	1,000 / 0,918 / 0,811	0,911	0,029

Outlook menang karena satu cabangnya, Overcast, langsung murni: empat contoh dan semuanya Yes. Satu cabang yang selesai seketika menurunkan rata-rata jauh lebih efektif daripada tiga cabang yang masing-masing sedikit membaik.

Ringkasan

Temperature nyaris tidak berguna di sini, padahal intuisi kita mungkin menduga sebaliknya. Gain yang dihitung dari data menunjukkan hal lain.

Algoritma ID3

ID3: rekursi yang sangat singkat

ID3(S, target, Atribut)

```
buat node akar
jika semua contoh di S berkelas sama c
  kembalikan leaf berlabel c
jika Atribut kosong
  kembalikan leaf kelas terbanyak
A ← arg max_{a∈Atribut} Gain(S, a)
beri label A pada akar
untuk setiap nilai v dari A:
  S_v ← {x ∈ S : x.A = v}
  jika S_v kosong
    tambahkan leaf kelas terbanyak
  selain itu
    tambahkan cabang v menuju
    ID3(S_v, target, Atribut \{A})
kembalikan akar
```

Tiga hal yang perlu diperhatikan:

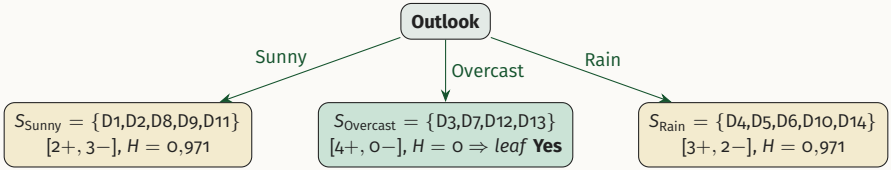
- Atribut yang sudah dipakai **dibuang** dari daftar pada cabang di bawahnya, karena nilainya sudah tetap di situ.
- Tiga kondisi berhenti: kelas seragam, atribut habis, atau cabang kosong.
- Pemilihan A bersifat *greedy*: dipilih yang terbaik saat itu, dan **tidak pernah ditinjau ulang**.

Ringkasan

Mencari pohon terkecil yang konsisten dengan data adalah NP-Hard, seperti yang kita sebut pada Pertemuan 1. ID3 tidak mencarinya; ia menempuh satu jalan yang masuk akal dan berhenti di situ.

Langkah 1 — akar: Outlook

Gain terbesar ada pada Outlook (0,246), sehingga ia menempati akar dan melahirkan tiga cabang:



Cabang Overcast selesai seketika: entropy-nya nol, jadi rekursi berhenti dan *node*-nya menjadi *leaf* **Yes**. Dua cabang lain masih campur, dan masing-masing diproses ulang oleh ID3 dengan sisa atribut {Temperature, Humidity, Wind}.

Langkah 2 — cabang Sunny: Humidity

Pada $S_{\text{Sunny}} = \{\text{D1, D2, D8, D9, D11}\}$ dengan $[2+, 3-]$ dan $H = 0{,}971$, kita hitung ulang gain untuk tiga atribut yang tersisa. Perhatikan: Outlook tidak lagi ikut dinilai.

Atribut	Pecahannya di cabang Sunny	*Entropy* sisa	Gain
Humidity	High {D1,D2,D8} $[0+, 3-]$, Normal {D9,D11} $[2+, 0-]$	0,000	**0,971**
Temperature	Hot $[0+, 2-]$, Mild $[1+, 1-]$, Cool $[1+, 0-]$	0,400	0,571
Wind	Weak {D1,D8,D9} $[1+, 2-]$, Strong {D2,D11} $[1+, 1-]$	0,951	0,020

Humidity memberi gain 0,971, yaitu *seluruh entropy* yang tersisa. Kedua cabangnya murni, sehingga keduanya langsung menjadi *leaf*: High $\rightarrow$ **No**, Normal $\rightarrow$ **Yes**.

Ringkasan

Bandingkan dengan akar: di sana Humidity hanya bernilai 0,151, di sini 0,971. Nilai sebuah atribut selalu **bergantung pada konteks** himpunan yang sedang dipecah, karena itu gain harus dihitung ulang di setiap *node*.

Langkah 3 — cabang Rain: Wind

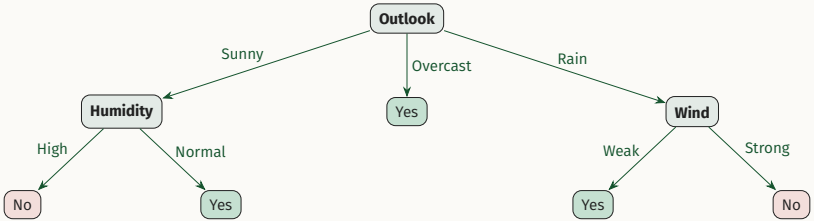
Pada $S_{\text{Rain}} = \{\text{D4, D5, D6, D10, D14}\}$ dengan $[3+, 2-]$ dan $H = 0{,}971$:

Atribut	Pecahannya di cabang Rain	*Entropy* sisa	Gain
Wind	Weak {D4,D5,D10} $[3+, 0-]$, Strong {D6,D14} $[0+, 2-]$	0,000	**0,971**
Temperature	Mild $[2+, 1-]$, Cool $[1+, 1-]$	0,951	0,020
Humidity	High {D4,D14} $[1+, 1-]$, Normal {D5,D6,D10} $[2+, 1-]$	0,951	0,020

Wind memisahkan dengan bersih: angin lemah selalu bermain, angin kencang selalu tidak. Kedua cabangnya murni, jadi rekursi berhenti dan pohon selesai tanpa perlu memakai Temperature sama sekali.

Dua hal yang layak dicatat. Pertama, Temperature tidak pernah terpilih di *node* mana pun, padahal ia tersedia; pohon yang dipelajari memakai tiga dari empat atribut. Kedua, seluruh 14 contoh terklasifikasi benar, sehingga *error* pada *training data* nol — dan justru itulah yang nanti perlu kita curigai.

Pohon akhir PlayTennis, dan aturan yang terbaca darinya



- JIKA Outlook=Sunny DAN Humidity=High MAKA No
- JIKA Outlook=Sunny DAN Humidity=Normal MAKA Yes
- JIKA Outlook=Overcast MAKA Yes
- JIKA Outlook=Rain DAN Wind=Weak MAKA Yes
- JIKA Outlook=Rain DAN Wind=Strong MAKA No

Lima *leaf*, lima aturan: saling lepas dan menutupi seluruh kemungkinan.

Hypothesis space dan *inductive bias*

ID3 dibandingkan Candidate-Elimination

	Candidate-Elimination	**ID3**
Hypothesis space	Hanya konjungsi atribut; banyak fungsi target tidak termuat di dalamnya	**Semua** fungsi diskret atas atribut yang ada; pohon dapat menyatakan disjungsi
Cara mencari	Menyimpan *seluruh* hipotesis yang konsisten (*version space*)	Menelusuri satu jalan saja, dari pohon kosong ke pohon yang makin rinci
Peninjauan ulang	Lengkap: hipotesis yang gugur benar-benar tidak konsisten	*Greedy* tanpa *backtracking*: pilihan di akar tidak pernah dibatalkan
Sikap pada *noise*	Rapuh; satu contoh salah *label* dapat mengosongkan *version space*	Cukup tahan; keputusan tiap *node* berdasar statistik banyak contoh

Ringkasan

Pertukarannya terbalik dan menarik. *Hypothesis space* ID3 **lengkap** sehingga fungsi target pasti ada di dalamnya, tetapi **pencariannya tidak lengkap**. Candidate-Elimination sebaliknya: pencariannya lengkap, ruangnya yang terbatas.

Inductive bias ID3: preferensi, bukan restriksi

Karena *hypothesis space*-nya memuat semua fungsi diskret, ID3 tidak dibatasi oleh bentuk hipotesisnya. Yang membuatnya mampu menggeneralisasi adalah **urutan pencariannya**.

Inductive bias ID3, kira-kira

Di antara semua pohon yang konsisten dengan *training data*, ID3 cenderung memilih pohon yang **lebih pendek**, dan menempatkan atribut ber-gain tinggi **lebih dekat ke akar**.

Preference bias (ID3)

Seluruh *hypothesis space* boleh dijelajahi; algoritma hanya *mengurutkan* mana yang dicoba lebih dulu. Fungsi target tidak akan tersingkir oleh bentuknya.

Restriction bias (*Candidate-Elimination*)

Hypothesis space-nya sendiri dipersempit. Kalau fungsi target berada di luar himpunan konjungsi, ia tidak akan pernah ditemukan, sebanyak apa pun datanya.

Umumnya *preference bias* lebih disukai, karena kegagalannya bersifat sementara: tambah data atau ubah heuristiknya, hipotesis yang benar masih mungkin terjangkau. Pembenaran atas "lebih pendek lebih baik" bersandar pada ***Occam's razor***: hipotesis sederhana lebih sedikit, jadi kalau ia kebetulan cocok dengan banyak data, kecocokan itu kurang mungkin kebetulan.

Dari ID3 ke C4.5/J48

Kelemahan *information gain*: silau pada atribut bernilai banyak

Tambahkan satu kolom **Tanggal** ke PlayTennis, berisi 14 nilai yang semuanya berbeda. Memecah menurut Tanggal menghasilkan 14 cabang, masing-masing berisi satu contoh, sehingga *entropy* setiap anak nol:

$$\mathit{Gain}(S, \text{Tanggal}) = 0{,}940 - 0 = 0{,}940$$

Gain maksimal, dan ID3 akan memilihnya di akar. Hasilnya pohon berkedalaman satu dengan 14 *leaf*: sempurna pada *training data*, dan sama sekali tidak berguna pada data baru.

Pola yang perlu dikenali

Gain memberi hadiah kepada atribut yang **memecah halus**, bukan yang **menjelaskan**. Nomor induk, nomor transaksi, nama pelanggan, dan *timestamp* adalah kandidat langganan masalah ini.

Gejalanya khas di lapangan: *training accuracy* hampir 100%, *test accuracy* runtuh, dan *root node*-nya berupa kolom identitas.

Gain ratio dan *split information* pada C4.5

Quinlan menormalkan gain dengan *entropy* dari *pemecahannya sendiri*, tanpa melihat kelas:

$$SplitInfo(S, A) = -\sum_{v} \frac{|S_v|}{|S|} \log_2 \frac{|S_v|}{|S|}$$

$$GainRatio(S, A) = \frac{Gain(S, A)}{SplitInfo(S, A)}$$

SplitInfo besar bila atribut memecah menjadi banyak bagian yang berukuran mirip; kecil bila hanya dua bagian. Atribut yang memecah halus otomatis terkena pembagi besar.

Atribut	Gain	SplitInfo	Ratio
Tanggal	0,940	3,807	0,247
Outlook	0,246	1,577	0,156
Humidity	0,151	1,000	0,151
Wind	0,048	0,985	0,049
Temperature	0,029	1,557	0,019

Jaraknya menyusut drastis: dari 0,940 lawan 0,246 menjadi 0,247 lawan 0,156.

Ringkasan

Pada data sekecil ini Tanggal *masih* unggul. C4.5 memang menambah penjaga: hanya atribut dengan gain setidaknya **sebesar rata-rata** yang dinilai rasionya, dan *SplitInfo* yang mendekati nol diperlakukan khusus. Rata-rata gain di sini 0,283, jadi yang lolos justru hanya Tanggal; penjaga ini tidak menyingkirkan kolom identitas. Yang menolong adalah membuang kolom identitas di tahap penyiapan data, sebelum melatih.

Atribut kontinu: memilih *threshold*

ID3 asli hanya menerima atribut kategorikal. C4.5 menanganinya dengan mengubah atribut kontinu A menjadi uji biner $A < c$, lalu mencari c terbaik langsung dari data.

1. Urutkan nilai A yang muncul pada contoh di *node* itu.
2. Ambil titik tengah setiap pasangan nilai berurutan yang **berbeda kelasnya** sebagai calon *threshold*.
3. Hitung gain untuk setiap calon, pilih yang terbesar.
4. Uji $A < c$ diperlakukan seperti atribut biner biasa, dan boleh dipakai ulang di cabang bawah dengan *threshold* berbeda.

Temperature	40	48	60	72	80	90
PlayTennis	No	No	Yes	Yes	Yes	No

Kelas berubah dua kali, jadi calon *threshold*-nya hanya dua: $c = (48{+}60)/2 = 54$ dan $c = (80{+}90)/2 = 85$. Di sini *threshold* $c = 54$ memberi gain terbaik.

Hanya mengurutkan sekali per *node*; inilah sumber faktor $\log n$ pada *training cost*.

Konsekuensi bentuknya: setiap uji berupa satu *threshold* pada satu atribut, sehingga batas keputusan pohon selalu berupa kotak-kotak sejajar sumbu. Batas diagonal hanya bisa didekati bertangga, dan itu butuh banyak *node*.

Nilai hilang, dan ringkasan perbedaan ID3 dengan C4.5/J48

Ketika nilai atribut tidak ada

- Cara termudah: isi dengan nilai **paling sering** pada atribut itu, atau nilai paling sering di antara contoh sekelas pada *node* yang sama.
- Cara C4.5: bagi contohnya menjadi **pecahan**. Bila 6 dari 10 contoh di *node* itu bernilai High, contoh yang hilang nilainya dihitung sebagai *weight* 0,6 di cabang High dan 0,4 di cabang Normal.
- Pada waktu menjawab, contoh dengan nilai hilang ditelusuri ke semua cabang, dan jawabannya diambil dari kelas dengan *weight* terbesar.

	ID3	C4.5 / J48
Kriteria pilih	*gain*	*gain ratio*
Atribut kontinu	tidak ada	*threshold* $A < c$
Nilai hilang	tidak ada	*weight* pecahan
Pruning	tidak ada	*post-pruning*
Output	pohon	pohon dan aturan

J48 adalah nama implementasi C4.5 di perangkat Weka, jadi keduanya merujuk algoritma yang sama. Di scikit-learn yang tersedia adalah varian CART, yang memakai *Gini impurity* dan selalu membentuk pohon biner.

Overfitting, *pruning*, dan *cost*

Overfitting, dirumuskan dengan tepat

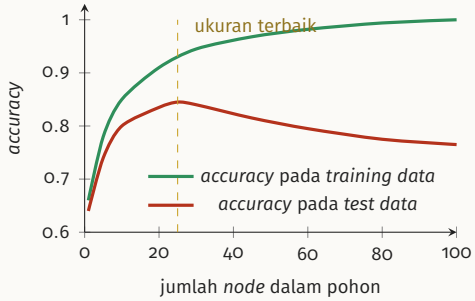
Mitchell, 1997
Suatu hipotesis h dikatakan ***overfit*** terhadap *training data* apabila terdapat hipotesis lain h' sedemikian sehingga h punya *error* lebih kecil daripada h' pada ***training data***, tetapi h' punya *error* lebih kecil daripada h pada **seluruh distribusi data**:

$$error_{train}(h) < error_{train}(h') \quad \text{tetapi} \quad error_{\mathcal{D}}(h) > error_{\mathcal{D}}(h')$$

Definisi ini menuntut **dua** pengukuran. Menyatakan sebuah model *overfit* hanya dengan melihat *training error*-nya tidak sah; yang menjadi bukti adalah selisihnya terhadap data yang belum pernah dilihat.

Decision tree sangat rentan, dan alasannya melekat pada cara kerjanya: rekursinya akan terus memecah sampai setiap *leaf* murni, kalau tidak dilarang. Pohon PlayTennis kita punya *training error* nol, dan angka itu justru tanda bahwa tidak ada yang menahannya berhenti.

Bentuk kurvanya, dan dari mana masalahnya datang



Training accuracy naik terus, karena setiap *node* tambahan selalu bisa dipakai menghafal satu contoh lagi. *Test accuracy* naik sampai satu titik, lalu **turun**. Jarak antara kedua kurva itulah ukuran *overfitting*.

Dua sebab utama:

- ***Noise.*** Satu contoh salah *label* memaksa pohon menumbuhkan cabang khusus untuknya, dan cabang itu salah bagi semua contoh lain yang mirip.
- ***Training data* sedikit.** Di kedalaman besar, sebuah *node* mungkin hanya memegang dua atau tiga contoh; pola yang terlihat di situ bisa muncul karena kebetulan, bukan karena ada keteraturan.

Dua waktu untuk menahan pohon: sebelum atau sesudah tumbuh

Pre-pruning — berhenti lebih awal

Rekursi dihentikan ketika syarat tertentu tercapai:

- batas **kedalaman** maksimum
- **minimum sampel** untuk boleh memecah, dan minimum sampel per *leaf*
- ***threshold* gain**: berhenti bila gain terbaik terlalu kecil
- uji keberartian statistik, misalnya χ^2 (dibaca *chi-square*)

Murah, satu kali jalan. Tetapi keputusannya diambil **tanpa tahu** apa yang akan terjadi di bawah.

Post-pruning — tumbuhkan penuh, lalu pangkas

Pohon dibiarkan tumbuh sampai habis, lalu cabang yang tidak terbukti berguna dibuang:

- *reduced-error pruning*
- *rule post-pruning* pada C4.5
- *cost-complexity pruning* pada CART, lewat parameter α (*alpha*)

Lebih mahal, tetapi memangkas dengan pengetahuan lengkap tentang isi cabang yang dibuang.

Mengapa *post-pruning* umumnya lebih baik

Ada pemecahan yang **sendirian** tampak tidak berguna tetapi membuka jalan bagi pemecahan berikutnya yang sangat berguna — gejala yang dikenal sebagai efek XOR. *Pre-pruning* memotong jalan itu sebelum manfaatnya terlihat; *post-pruning* menilai satu cabang setelah mengetahui seluruh isinya. Karena itu *pre-pruning* cenderung menghasilkan pohon yang *underfit*, sedangkan *post-pruning* memangkas berdasarkan bukti.

Dua cara memangkas yang perlu Anda kenal

Reduced-error pruning

Data dibagi tiga: *training*, ***validation***, dan *test*. Setelah pohon tumbuh penuh dari *training data*:

1. Untuk setiap *internal node*, bayangkan ia diganti *leaf* berlabel kelas terbanyak di bawahnya.
2. Ukur *accuracy* pada data **validasi**.
3. Lakukan *pruning* yang paling menaikkan *validation accuracy*.
4. Ulangi sampai tidak ada *pruning* yang memperbaikinya lagi.

Sederhana dan efektif. Kelemahannya jelas: ia memakan data, sehingga sulit dipakai ketika datanya sedikit — justru keadaan yang paling rawan *overfit*.

Rule *post-pruning* (C4.5)

1. Tumbuhkan pohon penuh, lalu ubah **setiap lintasan *root* ke *leaf*** menjadi satu aturan *if-then*.
2. Untuk setiap aturan, coba buang satu demi satu prasyaratnya; pertahankan pembuangan yang tidak memperburuk taksiran *accuracy*-nya.
3. Urutkan aturan yang tersisa menurut *accuracy*-nya, dan pakai urutan itu saat menjawab.

Keunggulannya: prasyarat dapat dibuang **di tengah** lintasan, sesuatu yang tidak mungkin dilakukan pada pohon karena memangkas *node* atas berarti membuang seluruh subpohon. C4.5 menaksir *accuracy* dari *training data* itu sendiri dengan koreksi pesimistik, sehingga tidak perlu menyisihkan *validation data*.

Cost: *training time* dan waktu jawab

Training time $O(d\, n \log n)$

Pada satu *node*, setiap atribut dari d atribut perlu ditinjau atas n contoh. Untuk atribut kontinu, contohnya harus **diurut** lebih dulu, dan di situlah faktor $\log n$ muncul. Karena setiap tingkat pohon membagi data tanpa menduplikasinya, total *cost* per tingkat tetap $O(d\, n)$, dan kedalaman yang khas berada pada orde $\log n$.

Waktu jawab $O($**kedalaman**$)$

Menjawab satu contoh hanya berarti menuruni satu lintasan: beberapa perbandingan skalar, tanpa perkalian matriks, tanpa menyimpan *training data*.

Cocokkan ini dengan tabel *cost* pada Pertemuan 1. *Decision tree* menempati sudut yang menguntungkan: latihnya hampir linier, jawabannya di antara yang termurah. Itulah sebabnya pohon tetap dipakai di sistem dengan anggaran milidetik, dan menjadi balok penyusun *ensemble* seperti *random forest* dan XGBoost pada pertemuan depan.

Ringkasan

Selain memperbaiki *accuracy*, *pruning* membuat pohon lebih cepat menjawab, lebih hemat memori, dan lebih mudah dibaca manusia.

Tugas untuk pertemuan berikutnya (1): hitung dengan tangan

1. Baca **Mitchell Bab 3** seluruhnya, terutama 3.4 (*information gain*), 3.6 (*inductive bias*), dan 3.7 (*overfitting* dan *pruning*).
2. Kerjakan tiga perhitungan berikut pada *dataset* PlayTennis. Tuliskan langkahnya, jangan hanya hasil akhirnya, dan bulatkan ke tiga angka di belakang koma.
   - *Gain*$(S, \text{Outlook})$ dan *Gain*$(S, \text{Temperature})$ pada himpunan lengkap, lengkap dengan *entropy* setiap anaknya.
   - *SplitInfo* dan *GainRatio* untuk Outlook dan Temperature, lalu jelaskan apakah urutan pemenangnya berubah dibanding memakai gain biasa.
   - Ganti *label* D11 dari Yes menjadi No, lalu hitung ulang gain keempat atribut di akar. Apakah Outlook masih menang? Apa yang pengamatan itu katakan tentang kepekaan ID3 terhadap *noise*?

Kerjakan sendiri lebih dulu tanpa kalkulator otomatis atau alat bantu AI. Menghitung $\log_2$ dengan tangan sekali saja membuat rumus pada slide tadi berhenti terasa sebagai simbol, dan mulai terasa sebagai keputusan.

Tugas untuk pertemuan berikutnya (2): praktikum

3. **Praktikum scikit-learn**, memakai data `load_wine` atau `load_breast_cancer`:
   - bagi *training data* dan *test data*, latih `DecisionTreeClassifier` tanpa batasan apa pun, lalu laporkan *training accuracy* dan *test accuracy*;
   - pangkas dengan `max_depth`, `min_samples_leaf`, dan `ccp_alpha`, lalu bandingkan sebelum dan sesudah *pruning* dalam satu tabel berisi jumlah *node*, *training accuracy*, dan *test accuracy*;
   - gambar pohon hasil *pruning* dengan `plot_tree`, lalu tuliskan **tiga aturan** *if-then* yang Anda baca darinya;
   - tutup dengan satu paragraf: apakah *pruning* menurunkan *training accuracy*, dan apakah *test accuracy*-nya membaik? Hubungkan jawabannya dengan definisi *overfitting* pada slide tadi.

Pertemuan depan

Kalau satu pohon mudah *overfit*, bagaimana kalau kita menanam banyak pohon lalu memungutkan suaranya? Dari pertanyaan itu lahir *random forest* dan *gradient boosting*.

Terima kasih

Pertanyaan?