Pertemuan 1 — Machine Learning 101

Dari sejarah berpikir sampai efisiensi *machine learning*

Hendri Karisma

Semester Ganjil 2026/2027

Program Studi Teknik Informatika, STMIK Tazkia

Perkenalan



Hendri Karisma

VP Engineering, jejakin.com
Produk yang ditangani:

- Emission and ESG analytics and calculation
- MRV (*Measurement, Reporting, Verification*) untuk ESG dan proyek karbon
- *Marketplace* proyek ESG, karbon, air, dan REC

Lecturer, STMIK Tazkia
Mengajar: Machine Learning, Artificial Intelligence, Data Engineering, Distributed System, Research Methodology.

Research area
Secure and Sustainable Computation, Artificial Intelligence, Quantum Computing.

Situs hendrikarisma.my.id
GitHub github.com/situkangsayur
LinkedIn linkedin.com/in/hendri-karisma-736a0a39
X @infoHendri Instagram @karism4_

Alur pertemuan hari ini

1. Sejarah berpikir: dari Plato sampai LLM
2. Empat wilayah kerja AI
3. Dua cara belajar: induktif dan deduktif
4. Definisi belajar menurut Tom Mitchell
5. Mengapa kita belajar *machine learning*
6. Pondasi: dari matematika sampai LLM
7. Dua jenis solusi: analitik dan numerik
8. Kelas masalah: P, NP, NP-Complete, NP-Hard
9. Kompleksitas algoritma
10. Efektivitas dan efisiensi: logika deduktif vs ML

Sasaran pertemuan ini

Anda dapat menceritakan bagaimana gagasan “mesin yang belajar” sampai ke bentuknya hari ini, menempatkan ML di dalam peta AI, dan menjelaskan mengapa ML memilih jalan numerik yang iteratif, bukan jawaban pasti sekali hitung.

Sejarah berpikir: dari Plato sampai LLM

Pertanyaannya tua: apa itu berpikir?

- **Plato dan Aristoteles** — berpikir dipandang mengikuti pola yang dapat dituliskan. Aristoteles merumuskan silogisme: dari dua premis lahir satu kesimpulan. Inilah cikal bakal gagasan bahwa penalaran punya aturan, dan karena punya aturan, suatu saat mungkin dimesinkan.
- **Rene Descartes** (1637) — *cogito, ergo sum*, "aku berpikir, maka aku ada". Berpikir ditempatkan sebagai ciri yang menegaskan keberadaan manusia, dan dipandang milik manusia semata. Mesin, bagi Descartes, tidak mungkin berpikir.
- **Gottfried Leibniz** — bermimpi tentang bahasa dan kalkulus penalaran, sehingga perbedaan pendapat cukup diselesaikan dengan "*Calculemus*" (hitunglah).

Selama berabad-abad batasnya jelas: **menghitung** boleh dimesinkan, tetapi **berpikir** tidak. Sejarah berikutnya adalah kisah bergesernya batas itu.

Dari mesin hitung ke mesin berpikir

- **Charles Babbage** (1830-an) — merancang *Analytical Engine*: mesin yang dapat melakukan komputasi apa pun, tetapi langkahnya tetap harus disetel manusia.
- **Ada Lovelace** — menulis algoritma pertama untuk mesin itu, sekaligus menyatakan bahwa mesin "hanya melakukan apa yang kita perintahkan". Sanggahan atas pernyataan inilah yang kelak dibahas Turing.
- **Alan Turing** (1936, 1950) — merumuskan model komputasi universal, lalu mengganti pertanyaan "dapatkah mesin berpikir?" dengan permainan imitasi yang bisa diuji.
- **Dartmouth** (1956) — istilah *artificial intelligence* lahir, dengan keyakinan bahwa kecerdasan dapat diprogramkan.

Ringkasan

Pergeserannya halus tetapi menentukan: dari "mesin menjalankan perintah" menjadi "mesin memperbaiki perilakunya sendiri".

Sistem pakar, dan mengapa ia mentok

Tahun 1970–1980-an pendekatan yang dominan adalah menuliskan pengetahuan pakar menjadi ribuan aturan: DENDRAL untuk analisis senyawa, MYCIN untuk diagnosis infeksi.

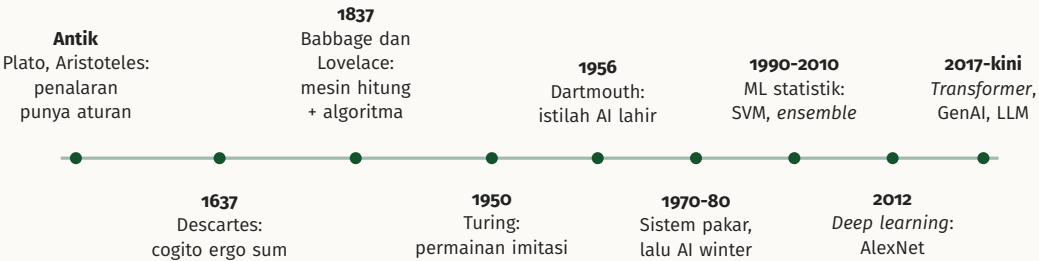
- Berhasil pada wilayah sempit dan stabil
- Aturannya statis: dunia berubah, aturannya tidak ikut berubah
- Menambah aturan baru bisa merusak yang lama
- Pakar sulit menjelaskan hal yang mereka lakukan secara intuitif

Ketika janji besar tidak tercapai, pendanaan surut. Masa ini dikenal sebagai *AI winter*.

Pelajaran yang terbawa

Pengetahuan yang **tidak dapat diucapkan** sulit ditulis sebagai aturan. Di sinilah pembelajaran dari contoh menjadi jalan keluar: biarkan sistem menyimpulkannya sendiri dari data.

Linimasa ringkas



Perhatikan polanya: setiap kali satu pendekatan mentok, penggantinya datang dari **cara merumuskan masalah yang berbeda**, jarang dari mesin yang lebih cepat.

Mengapa *deep learning* meledak setelah 2012

Tiga hal bertemu pada waktu yang sama:

1. **Data** — internet menyediakan citra dan teks berlabel dalam jumlah yang belum pernah ada, seperti ImageNet.
2. **Komputasi** — GPU membuat perkalian matriks besar menjadi murah.
3. **Algoritma** — ReLU, *dropout*, *initialization* yang lebih baik, dan *optimizer* yang lebih stabil membuat jaringan dalam bisa dilatih.

Gagasan *neural network* sendiri sudah ada sejak 1950-an. Yang berubah sejak itu adalah bahan bakunya: data dan daya komputasi. Dari sini jalannya lurus: jaringan makin dalam, lalu *transformer* (2017), lalu *generative AI* dan LLM.

Ringkasan

Sebuah ide bisa terlihat gagal hanya karena zamannya belum menyediakan data dan komputasi yang dibutuhkannya.

Empat wilayah kerja AI

Taksonomi AI: empat wilayah

Searching
Menelusuri kemungkinan untuk menemukan jalan: BFS, A*, minimax pada permainan

Planning
Menyusun urutan tindakan menuju tujuan: robot, penjadwalan, logistik

Learning
Memperbaiki kinerja dari pengalaman: inilah tempat *machine learning*

Reasoning
Menarik kesimpulan dari pengetahuan: logika, inferensi, basis pengetahuan

Sistem nyata menggabungkan keempatnya. Mobil otonom mencari jalur, merencanakan manuver, belajar mengenali objek, dan menalar aturan lalu lintas.

Dua cara belajar: induktif dan deduktif

Induktif dan deduktif

Induktif — *Machine learning*

Dari **contoh** menuju **aturan umum**.

- Diberi ribuan surel berlabel, simpulkan ciri surel *spam*
- Kesimpulannya bersifat kemungkinan, bukan kepastian
- Bisa salah, dan justru karena itu harus diukur

Deduktif — Sistem Pakar

Dari **aturan umum** menuju **kesimpulan khusus**.

- Jika demam dan batuk maka curiga infeksi
- Kesimpulannya pasti bila aturannya benar
- Aturan ditulis pakar, sehingga mahal dan kaku

Ringkasan

Mata kuliah ini menempati sisi kiri. Sisi kanan tidak mati: sistem modern sering menggabungkan keduanya, misalnya model yang belajar dari data tetapi dibatasi aturan yang tidak boleh dilanggar.

Definisi belajar menurut Tom Mitchell

Definisi yang akan kita pakai sepanjang semester

Tom M. Mitchell, 1997
Sebuah program komputer dikatakan **belajar** dari pengalaman *E* terhadap suatu kelas tugas *T* dan ukuran kinerja *P*, jika kinerjanya pada tugas *T*, yang diukur dengan *P*, **meningkat** seiring bertambahnya pengalaman *E*.

Definisi ini menuntut tiga hal disebutkan dengan jelas. Bila salah satunya kabur, klaim "sistem kami belajar" tidak dapat diuji.

- *T* — apa yang dikerjakan
- *P* — bagaimana keberhasilannya diukur
- *E* — dari pengalaman apa ia belajar

Latihan membaca: tiga contoh

Kasus	**Tugas** *T*	**Ukuran** *P*	**Pengalaman** *E*
Permainan dam (*checkers*)	Memainkan satu babak	Persentase kemenangan melawan lawan	Bermain melawan dirinya sendiri
Penyaring spam	Menandai surel sebagai spam atau bukan	*Accuracy*, khususnya berapa surel penting yang salah ditandai	Kumpulan surel yang sudah diberi *label* pengguna
Deteksi transaksi janggal	Menandai transaksi mencurigakan	Berapa penipuan tertangkap dibanding berapa alarm palsu	Riwayat transaksi berlabel dari bank

Perhatikan kolom *P*. Kalau hanya 1 dari 1000 transaksi adalah penipuan, model yang selalu menjawab “aman” sudah 99,9% akurat, dan sama sekali tidak berguna.

Tiga jenis pembelajaran dan alur kerjanya

Supervised

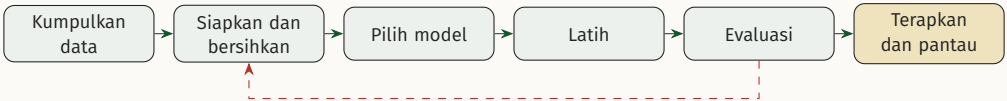
Data punya *label*. *Classification* dan *regression*. Fokus utama mata kuliah ini.

Unsupervised

Data tanpa *label*. Pengelompokan, reduksi dimensi, deteksi anomali.

Reinforcement

Belajar dari akibat: bertindak, menerima imbalan, memperbaiki kebijakan.



Alur ini berulang. Garis putus-putus itu yang paling sering terjadi di dunia nyata.

Mengapa kita belajar *machine learning*

Satu pertanyaan sederhana

Bagaimana Anda menulis program untuk mengenali kucing di dalam foto?

Bagaimana Anda menulis program untuk mengenali kucing di dalam foto?

Coba tuliskan aturannya:

- “Kalau ada dua telinga segitiga...” — bagaimana kalau telinganya terlipat?
- “Kalau berbulu abu-abu...” — bagaimana dengan kucing oranye, atau tanpa bulu?
- “Kalau ada kumis...” — bagaimana kalau fotonya gelap, terpotong, atau dari belakang?

Bagaimana Anda menulis program untuk mengenali kucing di dalam foto?

Coba tuliskan aturannya:

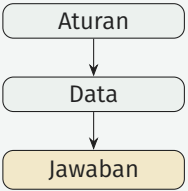
- “Kalau ada dua telinga segitiga...” — bagaimana kalau telinganya terlipat?
- “Kalau berbulu abu-abu...” — bagaimana dengan kucing oranye, atau tanpa bulu?
- “Kalau ada kumis...” — bagaimana kalau fotonya gelap, terpotong, atau dari belakang?

Ringkasan

Ada masalah yang **mudah dikerjakan manusia tetapi sulit dijelaskan sebagai aturan**. Untuk masalah semacam ini kita tidak menulis aturannya; kita menyiapkan contoh dan membiarkan program menemukan aturannya sendiri.

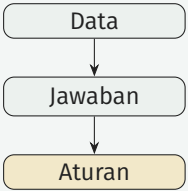
Dua cara membuat komputer menyelesaikan masalah

Pemrograman biasa



Manusia menyusun aturan. Komputer menjalankannya.

Machine learning



Manusia menyiapkan contoh. Komputer menyusun aturannya.

Aturan yang ditemukan mesin itulah yang kita sebut **model**.

Di mana ML dipakai, dan kapan sebaiknya tidak

Sudah dipakai setiap hari

- Penyaring *spam*, rekomendasi, deteksi penipuan
- Pengenalan wajah, suara, dan terjemahan
- Prakiraan permintaan, pembacaan citra medis
- Asisten berbasis LLM

Jangan pakai ML bila

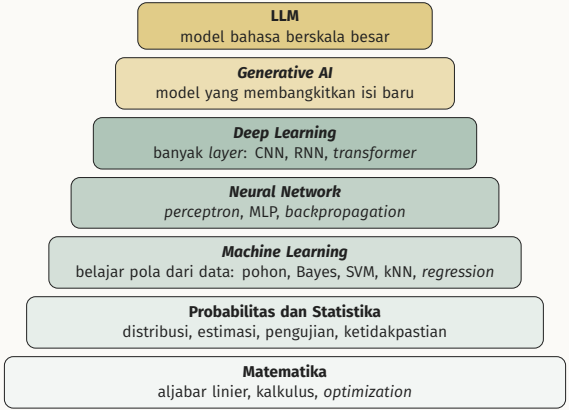
- Aturannya sudah jelas dan pasti (hitung pajak, validasi format)
- Datanya tidak ada atau tidak boleh dipakai
- Satu kesalahan pun berakibat fatal tanpa pengawasan
- Jawaban harus dipertanggungjawabkan langkah demi langkah

Ringkasan

Hampir semua sistem *software* modern punya satu bagian yang belajar dari data. Menolak memakai ML pada tempat yang salah adalah bagian dari keahlian, sama pentingnya dengan mampu melatih model.

Pondasi: dari matematika sampai LLM

Setiap *layer* berdiri di atas *layer* sebelumnya



Melompat ke *layer* paling atas tanpa *layer* bawahnya membuat kita bisa memakai, tetapi tidak bisa memperbaiki ketika hasilnya salah.

Membaca pondasi itu dari bawah

- **Matematika** memberi bahasa: vektor, matriks, *derivative*, dan cara mencari titik terbaik.
- **Probabilitas dan statistika** memberi cara menyatakan ketidakpastian dan menguji apakah temuan itu nyata.
- ***Machine learning*** memakai keduanya untuk menyusun aturan dari contoh.
- ***Neural network*** adalah satu keluarga model di dalam ML, bukan pengganti ML.
- ***Deep learning*** adalah *neural network* berlapis banyak; ia unggul ketika data dan komputasi berlimpah.
- ***Generative AI*** memakai model dalam untuk membangkitkan teks, gambar, atau suara baru.
- **LLM** adalah *generative AI* untuk bahasa, dilatih pada teks berskala sangat besar.

Ringkasan

Semua yang di atas tetap tunduk pada yang di bawah. LLM tidak menghapus statistika; ia justru contoh statistika yang dijalankan pada skala besar.

Dua jenis solusi: analitik dan numerik

Analitik dan numerik

Analitik (bentuk tertutup)

Menurunkan rumus, lalu menghitung sekali.

$$ax^2 + bx + c = 0 \Rightarrow x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Linear regression: $\hat{\beta} = (X^\top X)^{-1} X^\top y$

- Tepat, sekali jalan
- Perlu syarat tertentu, dan mahal bila matriksnya besar

Numerik (iteratif)

Menebak, mengukur kesalahan, memperbaiki, mengulang.

$$\theta_{t+1} = \theta_t - \eta \nabla L(\theta_t)$$

- Dipakai ketika rumus tertutupnya tidak ada
- Hasilnya hampiran; perlu *tuning learning rate* dan kriteria berhenti

Notasi: $\pm$: plus atau minus; $X^\top$: transpos X; $(\cdot)^{-1}$: invers matriks; $\hat{\beta}$ (dibaca *beta-hat*): hasil estimasi; θ (*theta*): parameter; ∇L (*nabla*): *gradient*; η (*eta*): besar langkah.

Letak *machine learning*

ML pada dasarnya **numerik**. Hanya sedikit model yang punya bentuk tertutup, itu pun hanya pada kasus kecil. Selebihnya kita menebak lalu memperbaiki, sampai cukup baik.

Mengapa ML memilih jalan numerik

- **Rumusnya sering tidak ada.** Untuk *neural network*, tidak ada persamaan yang langsung memberi *weight* terbaik. Yang ada hanya cara menghitung arah perbaikan.
- **Terlalu mahal walaupun ada.** Bentuk tertutup *linear regression* memuat $(X^\top X)^{-1}$, yang *cost*-nya tumbuh seperti d^3 terhadap jumlah *feature*. Pada data berdimensi besar, iterasi justru lebih murah.
- **Data datang bertahap.** Metode iteratif bisa memperbaiki model sedikit demi sedikit setiap kali data baru tiba; bentuk tertutup harus menghitung ulang dari awal.
- **Kita memang hanya butuh cukup baik.** Model dengan *error* 4,0% dan 4,05% biasanya sama bergunanya, sedangkan *cost* mengejar yang terakhir bisa berlipat.

Inilah yang akan kita temui pada pertemuan 9: *loss function*, *gradient descent*, dan SGD. Seluruh separuh kedua mata kuliah ini berdiri di atas gagasan tersebut.

Kelas masalah: P, NP, dan kerabatnya

Bentuk-bentuk persoalan

Bentuk pertanyaan

- **Keputusan**: jawabannya ya atau tidak
- **Pencarian**: temukan satu solusi yang sah
- ***Optimization***: temukan solusi terbaik menurut suatu ukuran
- **Pencacahan**: berapa banyak solusi yang ada

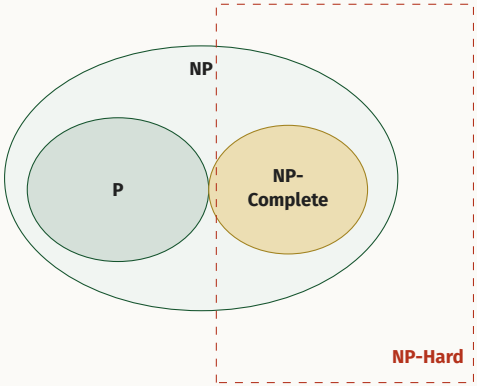
Pertanyaan yang menyertainya

- Berapa lama algoritmanya berjalan ketika *input* membesar?
- Berapa memori yang dibutuhkan?
- Apakah jawabannya harus tepat, atau cukup mendekati?

Machine learning hampir selalu berupa **masalah *optimization***: mencari model yang membuat kesalahan sekecil mungkin pada data.

P, NP, NP-Complete, NP-Hard

- **P**: dapat **diselesaikan** dalam waktu polinomial.
  Urutkan data, cari lintasan terpendek.
- **NP**: solusinya dapat **diperiksa** dalam waktu polinomial.
  Diberi satu jadwal, mudah dicek apakah sah.
- **NP-Complete**: yang paling sulit di dalam NP; semua masalah NP dapat direduksi kepadanya.
  SAT, pewarnaan graf, knapsack versi keputusan.
- **NP-Hard**: setidaknya sesulit NP-Complete, tetapi tidak harus berada di NP.
  TSP versi *optimization*, mencari *decision tree* terkecil.



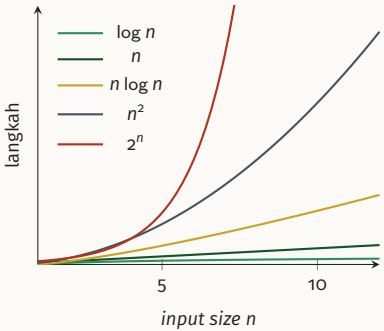
Gambar ini memakai dugaan $P \neq NP$, yang sampai hari ini belum dibuktikan maupun dibantah.

Kompleksitas algoritma

Tiga notasi pertumbuhan

- **Big O** $O(g)$ — batas **atas**.
  "Tidak lebih buruk dari."
- **Big Omega** $\Omega(g)$ — batas **bawah**.
  "Setidaknya sebesar ini."
- **Big Theta** $\Theta(g)$ — batas **atas dan bawah** sekaligus.
  "Persis tumbuh seperti ini."

Contoh: *merge sort* adalah $O(n \log n)$ sekaligus $\Omega(n \log n)$, sehingga $\Theta(n \log n)$. Sedangkan *quick sort* rata-rata $\Theta(n \log n)$ tetapi kasus terburuknya $O(n^2)$.



Mengapa perbedaan itu terasa

Pertumbuhan	$n = 10$	$n = 50$	$n = 100$	$n = 1000$
$\log_2 n$	3	6	7	10
n	10	50	100	1.000
$n \log_2 n$	33	282	664	9.966
n^2	100	2.500	10.000	1.000.000
2^n	1.024	$\approx 1{,}1 \times 10^{15}$	$\approx 1{,}3 \times 10^{30}$	tak terbayangkan

Pada $n = 50$, algoritma 2^n butuh sekitar 10^{15} langkah. Dengan satu miliar langkah per detik, itu lebih dari satu bulan. Pada $n = 100$, umur alam semesta tidak cukup.

Notasi: $\approx$ dibaca “kira-kira sama dengan”.

Ringkasan

Komputer yang lebih cepat tidak menyelamatkan algoritma eksponensial. Yang menyelamatkan adalah algoritma yang lebih baik, atau kesediaan menerima jawaban yang mendekati.

Kompleksitas yang menempel pada *machine learning*

- Mencari ***decision tree* terkecil** yang konsisten dengan data adalah NP-Hard. Karena itu ID3 dan C4.5 memakai pendekatan *greedy*: pilih atribut terbaik di setiap langkah, tanpa menjamin pohon paling ringkas.
- Melatih *neural network* berarti mencari minimum pada *error surface* yang tidak cembung. Kita tidak mengejar minimum global, melainkan titik yang cukup baik.
- *Feature selection* terbaik dan banyak persoalan *clustering* juga NP-Hard.

Ringkasan

Inilah alasan ML penuh kata **heuristik**, **aproksimasi**, dan **iterasi**. Penyebabnya: jawaban tepat sering mustahil dihitung dalam waktu yang masuk akal.

Efektivitas dan efisiensi: logika deduktif vs ML

Dua ukuran yang sering tertukar

Efektivitas

Seberapa **benar** hasilnya.

Accuracy, *precision*, *recall*, *error* pada data baru.
Pertanyaannya: apakah jawabannya tepat sasaran?

Efisiensi

Seberapa **murah** memperolehnya.

Training time, waktu jawab, memori, *computational cost*.
Pertanyaannya: berapa *cost* untuk sampai ke sana?

Ringkasan

Sistem yang efektif tetapi lambat tidak terpakai di produksi. Sistem yang cepat tetapi salah lebih berbahaya lagi, karena kesalahannya diproduksi dalam jumlah besar. Rekayasa yang baik selalu menyebut keduanya sekaligus.

Di mana *cost*-nya diletakkan

	Logika deduktif (sistem pakar)	***Machine learning* (induktif)**
Sumber aturan	Ditulis manusia; butuh pekan sampai bulan kerja pakar	Disimpulkan dari data oleh algoritma
Cost terbesar	Waktu manusia menyusun dan merawat aturan	Waktu mesin melatih model dan *cost* menyiapkan data
Waktu menjawab	Umumnya cepat, tetapi bisa meledak bila aturan saling berantai	Umumnya sangat cepat, karena model sudah jadi
Ketika dunia berubah	Aturan harus ditulis ulang satu per satu	Cukup melatih ulang dengan data baru
Jaminan	Kesimpulannya pasti bila aturannya benar; dapat ditelusuri	Bersifat peluang; benar rata-rata, bukan selalu
Kelemahan khas	Rapuh pada kasus yang tidak tertulis	Gagal diam-diam pada data yang berbeda dari *training data*

Perhatikan pergeserannya: ML memindahkan *cost* dari **waktu manusia** ke **waktu mesin**, dan waktu mesin jauh lebih mudah dibeli.

Time cost pada inferensi logika

- Pada logika proposisional, memeriksa apakah sebuah kesimpulan mengikuti dari sekumpulan aturan sudah termasuk masalah sulit: menguji kepuasan (SAT) adalah NP-Complete.
- Pada logika orde pertama, persoalannya lebih berat lagi: secara umum tidak ada prosedur yang selalu berhenti memberi jawaban untuk setiap pertanyaan.
- Mesin aturan praktis memakai siasat, misalnya algoritma pencocokan RETE, agar tidak mencocokkan semua aturan terhadap semua fakta setiap saat. *Cost*-nya tetap tumbuh bersama jumlah aturan dan fakta.
- Menambah satu aturan tidak berdiri sendiri: ia berinteraksi dengan aturan lain, sehingga perawatannya bertambah berat seiring waktu.

Ringkasan

Deduksi memberi kepastian, tetapi kepastian itu ada harganya: baik dalam waktu manusia menuliskannya, maupun dalam waktu mesin menalarinya.

Time cost pada *machine learning*

Model	***Training time* (kira-kira)**	**Waktu menjawab**
Linear regression, bentuk tertutup	$O(nd^2 + d^3)$	$O(d)$
Linear regression, *gradient descent*	$O(T\,n\,d)$, T = jumlah iterasi	$O(d)$
Naive Bayes	$O(nd)$	$O(d)$
Decision tree	$O(d\,n \log n)$	O(kedalaman)
Random forest (B pohon)	B kali pohon tunggal	$O(B \times$ kedalaman$)$
SVM dengan *kernel*	$O(n^2)$ sampai $O(n^3)$	$O(\lvert SV\rvert \times d)$
kNN	hampir nol, cukup menyimpan data	$O(nd)$ untuk setiap pertanyaan
Neural network	$O(E \times n \times P)$, P = jumlah parameter	$O(P)$

n = jumlah data, d = jumlah *feature*, E = jumlah *epoch*, $|SV|$ = jumlah *support vector*. Angka ini gambaran kasar, bukan jaminan.

Pola umumnya: **latih sekali, mahal; jawab berkali-kali, murah**. Pengecualiannya kNN, yang menunda semua pekerjaan sampai ada pertanyaan; karena itu ia disebut *lazy learner*.

Memilih dengan sadar

- **Mulai dari kendala waktu jawab.** Sistem *real-time* pada kasir atau gerbang tol punya anggaran milidetik. Model besar yang akurat tetapi lambat tidak lolos syarat itu, seberapa pun bagus *accuracy*-nya.
- **Ukur *retraining cost*.** Model yang harus dilatih ulang tiap hari punya *cost* berbeda dengan model yang cukup dilatih ulang tiap kuartal.
- **Pilih yang paling sederhana yang cukup.** Selisih *accuracy* satu persen sering tidak sebanding dengan *cost* sepuluh kali lipat dan perawatan yang jauh lebih rumit.
- **Gabungkan bila perlu.** Aturan deduktif untuk hal yang wajib pasti (misalnya batas hukum dan kebijakan), ML untuk hal yang berpola tetapi sulit dirumuskan.

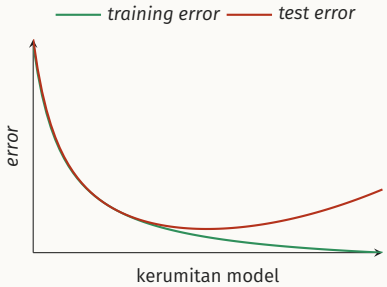
Ringkasan

Sepanjang semester ini kita akan terus menimbang dua hal sekaligus: apakah modelnya cukup tepat, dan apakah *cost*-nya masuk akal.

Satu peringatan yang akan kita ulang terus

Model yang menghafal *training data* belum tentu berguna. Yang kita kejar adalah ***generalization***: kinerja pada data yang belum pernah dilihat.

- ***Underfitting*** — model terlalu sederhana, salah di mana-mana
- ***Overfitting*** — model hafal *noise* pada *training data*, gagal pada data baru
- Karena itu data selalu dibagi: *training*, *validation*, dan *test*



Rencana satu semester

Peta mata kuliah

Sebelum UTS — fondasi dan model klasik

1. *Machine learning* 101 (hari ini)
2. *Concept learning*, Find-S, *version space*
3. *Decision tree*: ID3, C4.5/J48
4. *Ensemble*: *random forest*, XGBoost
5. Bayes, naive Bayes, GMM, dan EM
6. kNN dan K-Means
7. SVM dan HMM
8. **UTS — tugas besar**

Setelah UTS — *optimization* dan model dalam

9. *Linear regression*, evaluasi, *gradient descent*
10. *Perceptron*, MLP, *backpropagation*
11. Melatih jaringan secara stabil
12. *Deep neural network*
13. CNN
14. *Attention* dan *transformer*
15. LLM, *generative AI*, dan etikanya
16. **UAS — tugas besar**

Penilaian dan aturan main

Komponen nilai

Tugas dan kuis	20%
UTS — tugas besar	35%
UAS — tugas besar	45%

UTS dan UAS berbentuk **tugas besar**, tanpa ujian tertulis: kode, laporan, dan presentasi atas satu masalah nyata.

Aturan main

- Praktikum memakai Python: NumPy, pandas, scikit-learn, lalu PyTorch
- Boleh memakai alat bantu AI, asalkan **dituliskan** bagian mana yang dibantu dan Anda mampu menjelaskan kodenya
- Menyalin pekerjaan orang lain tanpa menyebut sumber dinilai nol
- Tugas besar dikerjakan 3–4 orang, memakai data publik atau data mitra dengan izin

Referensi

- **Tom M. Mitchell**, *Machine Learning*, McGraw-Hill, 1997.
  Acuan utama sampai UTS.
- **Stuart Russell dan Peter Norvig**, *Artificial Intelligence: A Modern Approach*, edisi ke-4, 2021.
  Peta besar AI dan sejarahnya.
- **Simon J.D. Prince**, *Understanding Deep Learning*, MIT Press, 2023.
  Terbuka beserta *notebook* di `udlbook.github.io/udlbook`. Acuan utama setelah UTS.
- **Andrew Ng**, catatan kuliah CS229 dan *Machine Learning Specialization*.
  Penjelasan *optimization* dan *neural network* yang runtut.
- **Ian Goodfellow dkk.**, *Deep Learning*, MIT Press, 2016.
  Rujukan teori.
- **Geoffrey Hinton**, kuliah *Neural Networks for Machine Learning* dan makalah terpilih.
  Hinton tidak menulis buku teks; yang dipakai materi kuliah dan makalahnya.

Tugas untuk pertemuan berikutnya

1. Baca Mitchell Bab 1 dan AIMA Bab 1–2.
2. Tulis ringkasan satu halaman berisi **tiga masalah nyata** di sekitar Anda yang dapat diselesaikan dengan ML. Untuk masing-masing sebutkan:
   - tugas T, ukuran kinerja P, dan pengalaman E;
   - dari mana datanya akan diperoleh;
   - satu alasan mengapa pendekatan berbasis aturan akan kesulitan.
3. Pastikan Python dan Jupyter berjalan di laptop Anda, atau siapkan akun Google Colab.

Pertemuan depan

Kita masuk ke inti *inductive learning*: *hypothesis space*, algoritma Find-S, dan mengapa tidak ada pembelajaran tanpa **bias**.

Terima kasih

Pertanyaan?