Pertemuan 6 — kNN dan *K-Means*

Dua algoritma yang hanya meminta satu hal dari data: ukuran jarak

Hendri Karisma

Semester Ganjil 2026/2027

Program Studi Teknik Informatika, STMIK Tazkia

Alur pertemuan hari ini

Bagian A — kNN (*supervised*)

1. *Eager learner* dan *lazy learner*
2. Algoritma k-nearest neighbour
3. *Distance metric* dan *scaling feature*
4. Memilih *k*; *weighting* jarak; LWR
5. Kutukan dimensi, *cost* menjawab, KD-tree

Bagian B — *K-Means* (*unsupervised*)

6. Belajar tanpa label: *clustering*
7. SSE dan algoritma Lloyd, contoh enam titik
8. Minimum lokal dan *k-means++*
9. Memilih *k*: *elbow* dan *silhouette*
10. Keterbatasan, dan kaitan dengan GMM

Penutup: kode NumPy dan scikit-learn, praktikum, rangkuman, kuis, dan tugas. Acuan: buku ajar Bab 8, Mitchell Bab 8, Hastie dkk. Bab 2 dan 14.3, Bishop Bab 9.

Tujuan pembelajaran

Setelah pertemuan ini Anda dapat:

1. membedakan *eager learner* dan *lazy learner*, termasuk konsekuensi *cost*-nya;
2. menjalankan kNN untuk *classification* dan *regression* dengan tangan, memilih *distance metric* dan *k*, serta menjelaskan mengapa *scaling feature* wajib;
3. menjelaskan *weighting* jarak dan *locally weighted regression*;
4. menjelaskan *curse of dimensionality* dan cara mempercepat pencarian tetangga;
5. menjalankan algoritma Lloyd untuk *K-Means* dengan tangan, dan menjelaskan peran *centroid* awal serta *k-means++*;
6. memilih jumlah kelompok dengan *elbow* dan *silhouette*, menyebut keterbatasan *K-Means*, dan membacanya sebagai GMM yang “keras”.

Dari eager ke lazy

Dua laporan lama yang tidak sedang memikirkan *machine learning*

1951 — Fix dan Hodges

Laporan teknis untuk Sekolah Kedokteran Penerbangan Angkatan Udara AS: cara mengelompokkan pengamatan tanpa menganggap datanya mengikuti distribusi tertentu. Salah satu aturannya: beri label pengamatan baru menurut mayoritas label pengamatan terdekatnya. Aturan itu kini disebut ***k-nearest neighbour*** (kNN).

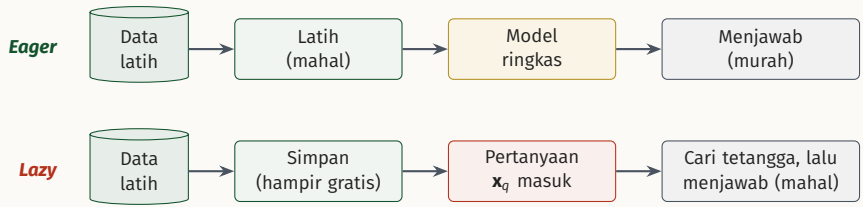
1957 — Lloyd, Bell Labs

Memadatkan sinyal suara menjadi sedikit level diskret untuk transmisi PCM dengan *error* kuadrat sekecil mungkin: tempatkan level, masukkan setiap nilai ke level terdekat, geser level ke rata-rata, ulangi. Terbit sebagai makalah baru pada 1982; MacQueen (1967) menamainya ***k-means***.

kNN memakai jarak untuk mencari contoh **berlabel** yang paling mirip dengan sebuah pertanyaan. *K-Means* memakai jarak untuk mengelompokkan data yang **tidak berlabel** sama sekali, dan dapat dibaca sebagai versi “keras” dari GMM yang kita bahas di Pertemuan 5.

Sumber: Fix dan Hodges (1951); Lloyd (1982); MacQueen (1967).

Eager learner dan *lazy learner*



	Eager	***Lazy***
Contoh	*Decision tree*, *naive Bayes*, GMM, SVM, *linear regression*	kNN, *locally weighted regression*, *case-based reasoning*
Hipotesis	Satu hipotesis global untuk seluruh ruang input	Hipotesis **lokal**, berbeda untuk setiap pertanyaan
Cost	Latih mahal, menjawab murah	Latih hampir gratis, menjawab mahal
Penjelasan	Dapat dibaca dari model tanpa melihat data	“Mirip dengan contoh-contoh ini”

Karena hipotesis dibentuk per pertanyaan, kNN melayani batas keputusan yang berliku tanpa usaha tambahan; model global harus menambah *feature* atau *kernel* lebih dulu.

k-nearest neighbour

Algoritmanya sesingkat itu

Training phase. Simpan seluruh pasangan $(\mathbf{x}_i, y_i)$. Selesai.

Tahap menjawab untuk satu pertanyaan $\mathbf{x}_q$:

1. Untuk $i = 1, \ldots, n$: hitung $d(\mathbf{x}_q, \mathbf{x}_i)$.
2. $N_k(\mathbf{x}_q) \leftarrow$ indeks k data dengan jarak terkecil.
3. Gabungkan label mereka menjadi satu jawaban.

Classification: suara terbanyak

$$\hat{y} = \arg\max_{c} \sum_{i \in N_k} \mathbb{1}[y_i = c]$$

Proporsi suara juga dapat dibaca sebagai taksiran peluang kelas.

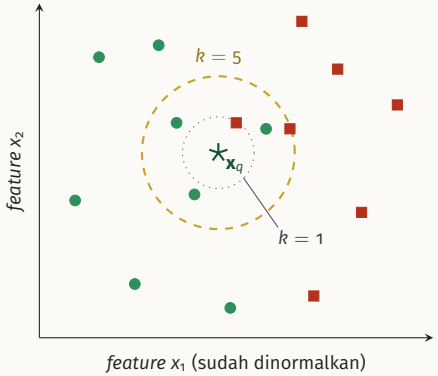
Regression: rata-rata tetangga

$$\hat{y} = \frac{1}{k} \sum_{i \in N_k} y_i$$

Output kontinu diperoleh dari rata-rata nilai tetangga.

Tidak ada parameter yang dilatih. k, *distance metric*, dan cara *weighting* adalah ***hyperparameter*** yang ditetapkan lewat validasi. Cover dan Hart (1967): bila data tak hingga banyak, *error* 1-NN paling buruk dua kali *error* Bayes optimal.

Satu pertanyaan, dua jawaban, tergantung k



Bulatan hijau dan kotak merah adalah dua kelas; bintang adalah pertanyaan $\mathbf{x}_q = (5, 5)$. Lima tetangga terdekat (Euclidean):

No.	Titik	Jarak	Kelas
1	(5,3; 5,5)	$\sqrt{0{,}09 + 0{,}25} = 0{,}58$	merah
2	(4,6; 4,3)	$\sqrt{0{,}16 + 0{,}49} = 0{,}81$	hijau
3	(4,3; 5,5)	$\sqrt{0{,}49 + 0{,}25} = 0{,}86$	hijau
4	(5,8; 5,4)	$\sqrt{0{,}64 + 0{,}16} = 0{,}89$	hijau
5	(6,2; 5,4)	$\sqrt{1{,}44 + 0{,}16} = 1{,}26$	merah

$k = 1$: jawabannya **merah**, karena tetangga terdekat kebetulan satu titik merah yang menyusup.

$k = 5$: suaranya 3 hijau lawan 2 merah, jawabannya **hijau**.

Satu titik dapat membalik keputusan, sehingga k perlu dipilih lewat validasi.

kNN dari nol dengan NumPy

```python
import numpy as np
from collections import Counter

def knn_prediksi(X_latih, y_latih, x_q, k=5, terbobot=False):
    d = np.sqrt(((X_latih - x_q) ** 2).sum(axis=1))   # jarak ke semua data
    idx = np.argpartition(d, k - 1)[:k]                # k terdekat, O(n)
    suara = Counter()
    for i in idx:
        w = 1.0 / max(d[i], 1e-12) ** 2 if terbobot else 1.0
        suara[y_latih[i]] += w
    return suara.most_common(1)[0][0]

# data pada gambar k=1 vs k=5: k=1 -> merah, k=5 -> hijau
```

- Baris `d = ...`: *broadcasting* menghitung n jarak sekaligus.
- `argpartition` mengambil k terkecil dalam $O(n)$ tanpa mengurutkan seluruh jarak.
- `terbobot=True` memberi *weight* $1/d^2$ (dibahas beberapa slide lagi); batas `1e-12` mencegah pembagian dengan nol.
- Untuk data nyata pakai `KNeighborsClassifier` dari scikit-learn.

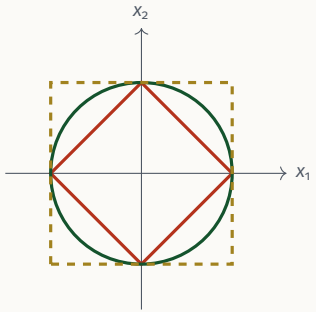
Kode lengkap: `bab08-knn-kmeans/knn_manual.py` di repositori kode buku.

Ukuran “mirip” harus dipilih lebih dulu

Metric	Rumus	Dipakai ketika
Euclidean (L_2)	$\sqrt{\sum_j (x_j - z_j)^2}$	Baku untuk *feature* numerik yang sebanding; satu selisih besar dihukum lebih keras daripada banyak selisih kecil
Manhattan (L_1)	$\sum_j \lvert x_j - z_j \rvert$	Lebih tahan *outlier*; lebih masuk akal pada data berdimensi tinggi
Minkowski (L_p)	$\left(\sum_j \lvert x_j - z_j \rvert^p \right)^{1/p}$	Bentuk umum: $p = 1$ Manhattan, $p = 2$ Euclidean, $p \to \infty$ hanya selisih terbesar
Cosine	$1 - \dfrac{\mathbf{x} \cdot \mathbf{z}}{\lVert\mathbf{x}\rVert \, \lVert\mathbf{z}\rVert}$	Ketika **arah** lebih berarti daripada besaran: teks TF-IDF, *embedding*
Hamming	$\sum_j \mathbb{1}[x_j \neq z_j]$	Data kategorikal atau biner: hitung berapa atribut yang berbeda
Gower	rata-rata ketidaksamaan per atribut, masing-masing di $[0, 1]$	Data campuran numerik dan kategorikal

kNN tidak punya *training phase*, sehingga *distance metric* menjadi satu-satunya tempat kita memberi tahu algoritma apa arti “mirip”. Di sinilah pengetahuan domain masuk.

Bola satuan, dan jebakan *one-hot*



L_1 (Manhattan): belah ketupat; L_2 (Euclidean): lingkaran; L_∞: persegi (garis putus-putus).

Setiap bentuk memuat semua titik berjarak 1 dari pusat. Titik yang dianggap “sama dekat” berbeda menurut metriknya.

Hati-hati: *one-hot encoding*

Bila atribut kategorikal diubah menjadi satu kolom biner per kategori, jarak Euclidean kuadrat antara dua baris yang kategorinya berbeda selalu 2, persis dua kali jarak Hamming.

Atribut dengan 50 kategori lalu menyumbang 50 kolom. Dipadukan dengan *feature* numerik yang belum diskalakan, sumbangannya bisa terlalu besar atau terlalu kecil. Untuk data campuran, pertimbangkan jarak Gower atau *weight* per kelompok *feature*.

Mengapa *scaling feature* wajib: satuan menentukan jawaban

Tiga pemohon pembiayaan dengan dua *feature*, gaji bulanan (rupiah) dan umur (tahun).

	Gaji (Rp)	Umur
P_1 (ditolak)	8.000.000	25
P_2 (diterima)	9.500.000	45
$\mathbf{x}_q$ (?)	8.200.000	44

Jarak Euclidean kuadrat pada satuan aslinya:

$$d(\mathbf{x}_q, P_1)^2 = (200.000)^2 + 19^2 = 4{,}0 \times 10^{10} + 361$$
$$d(\mathbf{x}_q, P_2)^2 = (1.300.000)^2 + 1^2 = 1{,}69 \times 10^{12} + 1$$

Jadi $d(\mathbf{x}_q, P_1) \approx 200.000$ dan $d(\mathbf{x}_q, P_2) \approx 1.300.000$. Tetangga terdekatnya P_1; dengan $k = 1$ jawabannya **ditolak**.

Lihat angka 361 dan 1

Sumbangan umur terhadap jarak ke P_1 hanya sekitar 10^{-8} dari total (361 dibanding 4×10^{10}). Umur secara efektif **tidak ikut dihitung**; yang kita jalankan sebenarnya kNN atas gaji saja.

Setelah diskalakan, jawabannya berbalik

Min–max dengan rentang yang masuk akal: gaji Rp5–20 juta (rentang 15 juta), umur 22–60 tahun (rentang 38).

	gaji′	umur′
P_1	0,200	0,079
P_2	0,300	0,605
$\mathbf{x}_q$	0,213	0,579

Contoh satu sel: gaji′ untuk $\mathbf{x}_q$ adalah $(8{,}2 - 5)/15 = 0{,}213$; umur′ adalah $(44 - 22)/38 = 0{,}579$.

$$d(\mathbf{x}_q, P_1) = \sqrt{0{,}013^2 + 0{,}500^2} = 0{,}500$$
$$d(\mathbf{x}_q, P_2) = \sqrt{0{,}087^2 + 0{,}026^2} = 0{,}091$$

P_2 kini lima kali lebih dekat, dan jawabannya menjadi **diterima**.

Yang berubah hanya satuan, dan keputusannya berbalik. ***Scaling*** karena itu dianggap bagian dari definisi model kNN.

- *Min–max*: $x' = (x - \min)/(\max - \min)$ ke $[0, 1]$; sederhana, peka nilai ekstrem.
- Standardisasi: $z = (x - \mu)/\sigma$; lebih tahan nilai ekstrem, pilihan bawaan.
- Hitung min, max, μ, σ **hanya dari *training data***, di dalam `Pipeline`, supaya informasi *test data* tidak bocor ke model.

Dampak *scaling* pada data Wine

Wine punya 13 *feature* bersatuan sangat berbeda: kadar prolin bernilai ratusan sampai lebih dari seribu, *feature* lain di bawah 10.

```
from sklearn.datasets import load_wine
from sklearn.model_selection import StratifiedKFold, cross_val_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler

X, y = load_wine(return_X_y=True)
cv = StratifiedKFold(10, shuffle=True, random_state=0)
for k in (1, 3, 5, 9, 15, 25):
    tanpa = cross_val_score(KNeighborsClassifier(k), X, y, cv=cv).mean()
    dengan = cross_val_score(make_pipeline(StandardScaler(),
                             KNeighborsClassifier(k)), X, y, cv=cv).mean()
    print(f"k={k:2d} tanpa={tanpa:.3f} dengan={dengan:.3f}")
```

Pada satu kali percobaan, *accuracy* 10-fold CV untuk $k = 5$ naik dari sekitar **0,68** tanpa *scaling* menjadi sekitar **0,96** dengan standardisasi. `make_pipeline` memasang ulang `StandardScaler` pada setiap *fold* latih.

Pengaruh k: dua kesalahan yang berlawanan

k kecil: *variance* tinggi

Keputusan bergantung pada sedikit titik, sehingga satu titik salah label atau satu *outlier* dapat membaliknya. Batas keputusannya berliku dan penuh pulau kecil.

Pada $k = 1$, *training error* selalu 0% (selama tidak ada dua titik identik berlabel beda), karena setiap titik adalah tetangga terdekat dirinya sendiri. Angka 0% itu tidak mengatakan apa pun tentang data baru.

k besar: bias tinggi

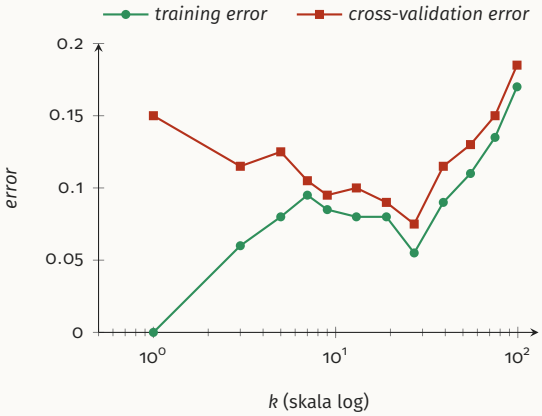
Suara diambil dari wilayah yang makin luas, sehingga batasnya makin halus dan struktur lokal tersapu.

Pada ekstremnya, $k = n$ membuat setiap pertanyaan dijawab dengan kelas mayoritas seluruh data: model tidak lagi melihat $\mathbf{x}_q$ sama sekali.

Inductive bias kNN

Titik yang berdekatan cenderung berlabel sama (asumsi *local smoothness*). Bila label ditentukan oleh XOR dua *feature* yang tenggelam di antara puluhan *feature* lain, asumsi itu runtuh dan kNN gagal berapa pun nilai k yang dipilih.

Error sebagai fungsi k, dan cara memilihnya



`make_moons`, 200 titik, *noise* tinggi, 10-fold CV (skrip `kurva_k.py`).

- $k = 1$: *training error* 0%, *cross-validation error* 15%, salah satu yang tertinggi. Jurang selebar itu tanda *overfitting*.
- *CV error* terendah 7,5% pada $k = 27$, lalu naik lagi ketika batas keputusan terlalu halus.
- Kurvanya bergelombang; selisih satu poin persen antara dua k berdekatan jangan dianggap kemenangan.

Memilih k

Coba deretan k, ukur dengan *cross-validation* pada *training data*, ambil yang terbaik (atau yang paling sederhana di antara yang setara), lalu laporkan kinerja pada *test data*. Dua kelas: k ganjil. c kelas: hindari kelipatan c. $k \approx \sqrt{n}$ cukup sebagai titik awal pencarian.

Weighting jarak: tetangga dekat bersuara lebih keras

Pada kNN biasa, tetangga di jarak 0,1 dan di jarak 1,2 punya suara yang sama besar. Beri *weight* $w_i = 1/d(\mathbf{x}_q, \mathbf{x}_i)^2$, lalu pakai suara atau *weighted average*:

$$\hat{y}_{classification} = \arg\max_c \sum_{i \in N_k} w_i \mathbb{1}[y_i = c], \qquad \hat{y}_{regression} = \frac{\sum_{i \in N_k} w_i y_i}{\sum_{i \in N_k} w_i}$$

Contoh, $k = 3$**.**

Tetangga	d_i	$w_i = 1/d_i^2$	Kelas
1	0,5	4,000	A
2	1,0	1,000	B
3	1,2	0,694	B

Suara biasa: B menang 2 lawan 1. *Weighted vote*: A memperoleh 4,000, B memperoleh $1{,}000 + 0{,}694 = 1{,}694$. **A menang.**

Versi *regression* dengan tetangga bernilai $y = 10,\ 12,\ 14$ pada jarak yang sama:

$$\hat{y} = \frac{4(10) + 1(12) + 0{,}694(14)}{4 + 1 + 0{,}694} = \frac{61{,}72}{5{,}694} = 10{,}84$$

Rata-rata biasa memberi 12,00; *weighting* menarik jawaban mendekati tetangga terdekat.

Bila $d = 0$, jawaban diambil langsung dari data itu. Dengan *weighting*, k yang terlalu besar menjadi kurang merusak. Shepard (1968) memakai *semua* data dengan *weight* kebalikan jarak: *inverse distance weighting*.

Locally weighted regression (LWR)

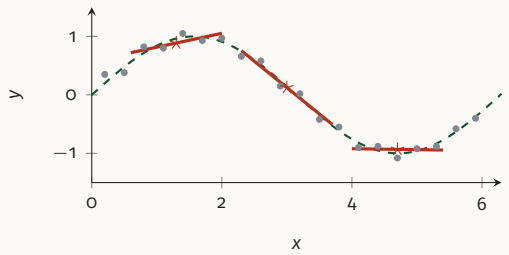
kNN *regression* menjawab dengan rata-rata yang datar, sehingga kurvanya bertangga. LWR **mencocokkan sebuah garis** di sekitar setiap pertanyaan dengan *weighted least squares*:

$$\min_{\beta} \sum_i w_i \big(y_i - \beta_0 - \beta_1 x_{i1} - \dots\big)^2,$$

$$w_i = K\left(\frac{d(\mathbf{x}_q, \mathbf{x}_i)}{h}\right), \quad K(u) = e^{-u^2/2}.$$

Garis dipakai sekali untuk menjawab $\mathbf{x}_q$, lalu dibuang.

Dua puluh titik, $h = 0{,}4$: untuk $x_q = 1{,}3$, $\hat{y} = 0{,}887 + 0{,}237(x - 1{,}3)$; untuk $x_q = 3$, $\hat{y} = 0{,}124 - 0{,}902(x - 3)$; untuk $x_q = 4{,}7$, $\hat{y} = -0{,}932 - 0{,}016(x - 4{,}7)$.



Putus-putus hijau: fungsi sebenarnya $\sin x$; bintang: pertanyaan.

Ringkasan

h berperan seperti *k*: *h* kecil berarti *variance* tinggi, *h* besar bias tinggi. Bila $h \to \infty$, LWR menjadi satu garis *regression* global.

Kutukan dimensi dan *cost* menjawab

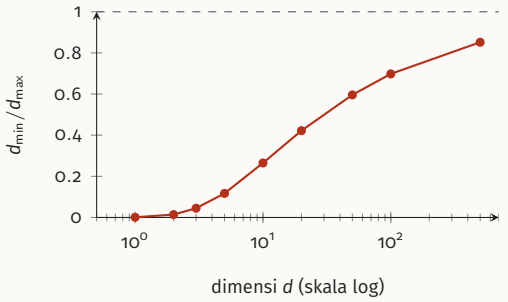
Pada dimensi tinggi, semua titik nyaris sama jauh

Istilah ***curse of dimensionality*** berasal dari Bellman (1961). Dua ilustrasi pada kubus satuan $[0, 1]^d$:

d	1	2	10	50	100
kulit 0,05: $1 - 0{,}9^d$	10%	19%	65%	99,5%	$\approx$ 100%
sisi 10% data: $0{,}1^{1/d}$	0,10	0,32	0,79	0,95	0,98

Baris pertama: bagian volume yang berada di *layer* setebal 0,05 dari permukaan. Pada $d = 100$ hampir tidak ada "bagian dalam".

Baris kedua: sisi kubus kecil yang memuat 10% data seragam. Pada $d = 10$, "sepuluh persen tetangga terdekat" sudah merentang 79% rentang setiap *feature*.



Rasio jarak terdekat terhadap terjauh untuk 1.000 titik acak seragam, rata-rata 200 percobaan (`kutukan_dimensi.py`). Pada $d = 2$ rasionya sekitar 0,014; pada $d = 100$ sekitar 0,70. Ketika rasio mendekati 1, "tetangga terdekat" dan "titik terjauh" hampir berjarak sama (Beyer dkk., 1999).

Menghadapi kutukan dimensi

- **Buang *feature* yang tidak relevan.** Setiap *feature* tak relevan menambah komponen acak pada jarak dan menenggelamkan *feature* yang berguna. Efeknya lebih terasa pada kNN daripada pada *decision tree*, yang mengabaikan *feature* tak berguna dengan sendirinya.
- **Reduksi dimensi.** PCA atau *embedding* memadatkan puluhan *feature* menjadi beberapa arah yang membawa *variance*.
- ***Weight* per *feature*.** Beri setiap *feature weight* α_j di dalam jarak, setel dengan *cross-validation*. Gagasan ini berkembang menjadi *metric learning*.

Kaitan dengan Pertemuan 2

Makin banyak *feature*, makin longgar ruang hipotesis, dan makin banyak data yang dibutuhkan untuk mengikatnya. Untuk mempertahankan kerapatan titik yang sama, jumlah data harus tumbuh eksponensial terhadap dimensi.

Data nyata sering terletak pada "permukaan" berdimensi jauh lebih rendah daripada jumlah kolomnya, sehingga kutukannya tidak selalu sekeras simulasi seragam. kNN tetap kuat bila dimensinya sedang, datanya cukup, dan *feature*-nya terpilih baik.

Latih hampir gratis, menjawab mahal

	Latih	Satu pertanyaan
Waktu	$O(1)$	$O(nd)$
Memori	$O(nd)$	$O(nd)$

Mencari k terkecil setelah jarak dihitung hanya menambah $O(n \log k)$ dengan *heap* atau $O(n)$ dengan *partial sort*. *Cost*-nya didominasi penghitungan n jarak.

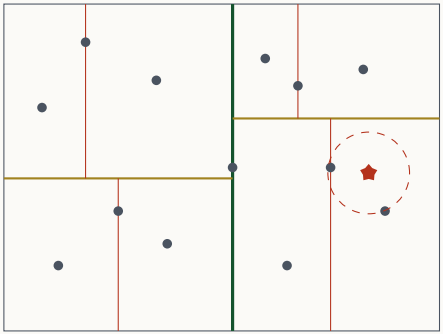
Dengan $n = 1.000.000$ dan $d = 100$, satu pertanyaan memerlukan sekitar 10^8 operasi dasar. Pada seribu permintaan per detik, sistem itu tidak dapat dilayani apa adanya.

Kembali ke Pertemuan 1

Pola umum ML: latih sekali dengan mahal, menjawab berkali-kali dengan murah. kNN adalah pengecualian yang paling terang.

Ia sering efektif, tanpa asumsi bentuk fungsi dan tanpa *tuning* panjang, tetapi lambat saat menjawab, dan seluruh *training data* ikut dibawa ke produksi. Pada sistem *real-time* dengan anggaran milidetik, itu sering menjadi alasan penolakan yang tidak berhubungan dengan *accuracy*-nya.

KD-tree: membelah ruang supaya sebagian besar titik dilewati



hijau: belahan pertama (x_1); kuning: belahan kedua (x_2); merah: belahan ketiga (x_1). Bintang: pertanyaan.

KD-tree (Bentley, 1975) membelah ruang bergantian menurut sumbu *feature*. Pencarian dimulai dari sel tempat pertanyaan berada, lalu hanya memeriksa sel lain yang mungkin memuat titik lebih dekat dari lingkaran putus-putus.

Sangat membantu pada dimensi rendah, kira-kira sampai belasan atau dua puluhan dimensi. Di atas itu kinerjanya merosot mendekati pencarian menyeluruh, dan penyebabnya kutukan dimensi tadi.

Ball-tree membelah dengan bola bersarang; lebih tahan pada dimensi menengah dan menerima *metric* selain Euclidean. Keduanya dipilih lewat parameter `algorithm` di scikit-learn.

Dari pencarian tepat ke pencarian hampiran

- ***Approximate nearest neighbour*** (ANN) melepas jaminan ketepatan demi kecepatan.
  - LSH (*locality-sensitive hashing*): fungsi *hash* yang sengaja menabrakkan titik-titik yang berdekatan (Indyk dan Motwani, 1998).
  - HNSW (*hierarchical navigable small world*): graf berlapis yang ditelusuri dalam waktu kira-kira logaritmik (Malkov dan Yashunin, 2020).
  - FAISS menjalankan indeks semacam ini pada miliaran vektor (Johnson dkk., 2021).
- **Kurangi datanya.** Buang contoh yang tidak pernah mengubah keputusan, atau ganti sekelompok titik berdekatan dengan satu wakil (prototipe).

Kebutuhan	**Pilihan**	**Catatan**
Data ribuan–ratusan ribu	scikit-learn `KNeighbors*`	Tepat, KD-tree/Ball-tree bawaan
Library ANN	FAISS, hnswlib	HNSW, IVF, kuantisasi; GPU
Vector database	pgvector, Qdrant, Milvus	Filter atribut; dasar sistem RAG (Pertemuan 15)
Evaluasi indeks	*recall@k*	Bandingkan dengan pencarian tepat pada sampel *query*

“Produk serupa” di *marketplace* dan “nasabah berprofil mirip” di bank adalah kNN berskala besar. Vektor nasabah adalah data pribadi; kendali aksesnya sama ketatnya dengan *database* biasa.

Tanpa label: *clustering*

Belajar tanpa label

Semua yang dibahas sejauh ini memerlukan label: kNN mencari tetangga lalu menyalin label mereka. Banyak data tidak pernah diberi label. Riwayat transaksi jutaan nasabah tidak datang dengan keterangan “nasabah ini termasuk segmen mana”.

Unsupervised learning mencari struktur pada data $\{\mathbf{x}_i\}$ tanpa y. Bentuk yang paling sering dipakai adalah ***clustering***: membagi data ke kelompok yang anggotanya saling mirip.

Pada Pertemuan 5, GMM melakukannya dengan menganggap data berasal dari campuran Gaussian lalu menaksir parameternya dengan EM. *K-Means* memakai alat yang jauh lebih sederhana: jarak Euclidean yang sama dengan kNN.

Contoh pemakaian *clustering*

- Segmentasi nasabah menurut saldo dan frekuensi transaksi
- Mengelompokkan dokumen atau berita menurut topik
- Kuantisasi warna citra: ganti jutaan warna dengan k warna wakil
- Kuantisasi vektor, tujuan awal Lloyd pada sinyal suara
- Menyiapkan rata-rata awal bagi EM pada GMM

Hati-hati dengan nama

k pada kNN adalah jumlah tetangga yang ditanya untuk satu pertanyaan. k pada *K-Means* adalah jumlah kelompok yang dicari di seluruh data.

kNN, *K-Means*, dan GMM

	kNN	***K-Means***	**GMM (Pertemuan 5)**
Jenis	*Supervised*, *lazy*	*Unsupervised*	*Unsupervised*, probabilistik
Arti *k*	Jumlah tetangga	Jumlah kelompok	Jumlah komponen Gaussian
Yang disimpan	Seluruh *training data*	*k centroid*	*k* rata-rata, *covariance*, dan *weight* campuran
Keanggotaan	—	Tepat satu kelompok (*hard*)	Peluang di setiap kelompok (*soft*)
Dinilai dengan	*Accuracy* atau RMSE pada *test data*	SSE, *silhouette*	*Log-likelihood*, BIC, *silhouette*

Baris “keanggotaan” dipakai lagi di akhir: *K-Means* dapat diturunkan dari GMM.

K-Means

Fungsi tujuan dan algoritma Lloyd

Cari k ***centroid*** $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k$ yang meminimalkan

$$\mathrm{SSE} = \sum_{j=1}^{k} \sum_{\mathbf{x}_i \in S_j} \|\mathbf{x}_i - \boldsymbol{\mu}_j\|^2,$$

dengan S_j himpunan titik di kelompok ke-j. Di scikit-learn besaran ini disebut ***inertia***.

Minimum global SSE adalah persoalan NP-*hard* bahkan untuk $k = 2$ (Aloise dkk., 2009). Praktiknya memakai prosedur berulang Lloyd, yang hanya menjamin minimum lokal.

K-Means (algoritma Lloyd)

1. Pilih k *centroid* awal (acak atau *k-means++*).
2. ***Assignment***: untuk setiap i, $c_i \leftarrow \arg\min_j \|\mathbf{x}_i - \boldsymbol{\mu}_j\|^2$.

   Notasi: $\arg\min_j$: indeks j yang membuat ekspresi paling kecil; $\leftarrow$: pemberian nilai; $|S_j|$: banyak anggota S_j.
3. ***Update***: untuk setiap j, $\boldsymbol{\mu}_j \leftarrow \frac{1}{|S_j|} \sum_{i:\, c_i = j} \mathbf{x}_i$.
4. Ulangi langkah 2–3 sampai tidak ada label yang berubah.

Satu iterasi berbiaya $O(nkd)$: matriks jarak kuadrat $n \times k$ dihitung sekaligus.

Mengapa SSE tidak pernah naik

Langkah *assignment* memindahkan setiap titik ke pusat terdekat, jadi suku milik titik itu hanya bisa mengecil.

Langkah *update*. Turunkan jumlah kuadrat jarak satu kelompok terhadap pusatnya:

$$\frac{\partial}{\partial \boldsymbol{\mu}_j} \sum_{\mathbf{x}_i \in S_j} \|\mathbf{x}_i - \boldsymbol{\mu}_j\|^2 = -2 \sum_{\mathbf{x}_i \in S_j} (\mathbf{x}_i - \boldsymbol{\mu}_j) = \mathbf{0} \quad \Longrightarrow \quad \boldsymbol{\mu}_j = \frac{1}{|S_j|} \sum_{\mathbf{x}_i \in S_j} \mathbf{x}_i .$$

Rata-rata adalah titik yang meminimalkan jumlah kuadrat jarak ke anggota kelompoknya, sehingga memindahkan pusat ke sana juga tidak menaikkan SSE.

Akibatnya

Jumlah cara membagi n titik ke k kelompok berhingga dan SSE tidak pernah naik, sehingga algoritma pasti berhenti. Tempat berhentinya adalah **minimum lokal** yang bergantung pada *centroid* awal.

Contoh enam titik, $k = 2$: iterasi 1

Titik $A(1;1)$, $B(1{,}5;2)$, $C(3;4)$, $D(5;7)$, $E(3{,}5;5)$, $F(4{,}5;5)$. Sengaja dipilih awal yang buruk: $\boldsymbol{\mu}_1 = A$ dan $\boldsymbol{\mu}_2 = B$, dua titik yang sebenarnya satu kelompok. SSE dihitung setelah langkah *assignment*, dengan pusat dari iterasi sebelumnya.

Iterasi 1. Jarak kuadrat ke $\boldsymbol{\mu}_1$ dan $\boldsymbol{\mu}_2$:

	A	B	C	D	E	F
ke $\boldsymbol{\mu}_1 = (1;1)$	0	1,25	13,00	52,00	22,25	28,25
ke $\boldsymbol{\mu}_2 = (1{,}5;2)$	1,25	0	6,25	37,25	13,00	18,00
kelompok	1	2	2	2	2	2

$\text{SSE} = 0 + 0 + 6{,}25 + 37{,}25 + 13{,}00 + 18{,}00 = 74{,}5.$

Pusat baru: $\boldsymbol{\mu}_1 = (1;1)$ dan $\boldsymbol{\mu}_2 = \left(\frac{1{,}5+3+5+3{,}5+4{,}5}{5};\ \frac{2+4+7+5+5}{5}\right) = (3{,}5;\ 4{,}6).$

Iterasi 2 dan 3: algoritma pulih dari awal yang buruk

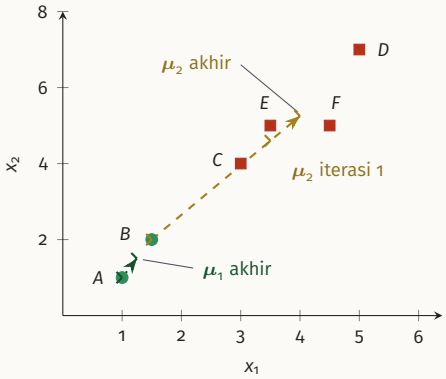
Iterasi 2. B berjarak kuadrat 1,25 ke $\boldsymbol{\mu}_1$ dan $2^2 + 2{,}6^2 = 10{,}76$ ke $\boldsymbol{\mu}_2$, sehingga B pindah ke kelompok 1. Titik lain tetap.

$$\mathrm{SSE} = 0 + 1{,}25 + 0{,}61 + 8{,}01 + 0{,}16 + 1{,}16 = 11{,}19$$

Pusat baru: $\boldsymbol{\mu}_1 = (1{,}25;\ 1{,}5)$ dan $\boldsymbol{\mu}_2 = (4;\ 5{,}25)$.

Iterasi 3. Tidak ada titik yang berpindah; SSE $= 7{,}875$ dan pusat tidak bergerak lagi. Algoritma berhenti dengan kelompok $\{A, B\}$ dan $\{C, D, E, F\}$.

Jejak SSE: $74{,}5 \to 11{,}19 \to 7{,}875$, turun di setiap iterasi seperti yang dijamin.



Tanda silang dan panah putus-putus: lintasan *centroid*.

Algoritma Lloyd dari nol dengan NumPy

```python
import numpy as np

def kmeans_lloyd(X, C, maks_iter=100):
    C = C.astype(float).copy()
    for it in range(maks_iter):
        d2 = ((X[:, None, :] - C[None, :, :]) ** 2).sum(axis=2)
        label = d2.argmin(axis=1)                    # assignment
        sse = d2[np.arange(len(X)), label].sum()
        C_baru = np.array([X[label == j].mean(axis=0)  # update
                           for j in range(len(C))])
        if np.allclose(C_baru, C):
            break
        C = C_baru
    return C, label, sse

X = np.array([[1, 1], [1.5, 2], [3, 4], [5, 7], [3.5, 5], [4.5, 5]])
print(kmeans_lloyd(X, X[[0, 1]]))   # SSE 74.5 -> 11.19 -> 7.875
```

- `X[:, None, :] - C[None, :, :]` berukuran $n \times k \times d$; setelah `sum(axis=2)` menjadi matriks jarak kuadrat $n \times k$.
- `argmin(axis=1)` adalah langkah *assignment*; rata-rata per kelompok adalah langkah *update*.
- SSE dicatat setelah *assignment* dengan pusat lama, sama seperti hitungan tangan.
- Berhenti ketika pusat tidak bergerak lagi.
- Kelompok kosong tidak ditangani di sini; `KMeans` di scikit-learn menanganinya.

Kode lengkap:
`babo8-knn-kmeans/kmeans.py`.

Minimum lokal: awal yang lain, jawaban yang lain

Jalankan Lloyd pada enam titik yang sama, tetapi mulai dari $\boldsymbol{\mu}_1 = C$ dan $\boldsymbol{\mu}_2 = D$.

Iterasi	Kelompok	SSE
1	$\{A, B, C, E, F\}, \{D\}$	23,75
2	$\{A, B, C, E\}, \{D, F\}$	19,95
3	$\{A, B, C\}, \{D, E, F\}$	13,375
4	$\{A, B, C\}, \{D, E, F\}$	10,67

Setelah empat iterasi tidak ada titik yang ingin berpindah, jadi Lloyd menganggap tugasnya selesai. Pembagian $\{A, B\}$ dan $\{C, D, E, F\}$ dari slide sebelumnya punya SSE **7,875**, lebih kecil.

Pada data besar

Minimum lokal biasanya berbentuk satu kelompok nyata yang dibelah dua sementara dua kelompok lain digabung menjadi satu.

Dua penangkal

- Awal yang lebih cerdas: *k-means++*.
- Banyak awal: jalankan beberapa kali (`n_init`) dan ambil hasil dengan SSE terkecil.

Initialization k-means++

k-means++ (Arthur dan Vassilvitskii, 2007)

1. $\boldsymbol{\mu}_1 \leftarrow$ satu titik data, dipilih acak seragam.
2. Untuk $j = 2, \ldots, k$:
   - $D(\mathbf{x}_i) \leftarrow \min_{l<j} \|\mathbf{x}_i - \boldsymbol{\mu}_l\|$, jarak ke pusat terdekat;
   - $\boldsymbol{\mu}_j \leftarrow \mathbf{x}_i$ dengan peluang $D(\mathbf{x}_i)^2 / \sum_m D(\mathbf{x}_m)^2$.
3. Jalankan algoritma Lloyd dari pusat-pusat itu.

SSE harapan hasil *initialization* ini paling buruk $O(\log k)$ kali SSE optimal, bahkan sebelum iterasi Lloyd dijalankan.

Contoh. Misalkan undian pertama memilih $\boldsymbol{\mu}_1 = F(4{,}5; 5)$. Jarak kuadrat ke F menjadi *weight* undian kedua:

	A	B	C	D	E	F
$D(\mathbf{x})^2$	28,25	18,00	3,25	4,25	1,00	0
peluang	0,516	0,329	0,059	0,078	0,018	0

Jumlah *weight* 54,75. Peluang pusat kedua jatuh di A atau B, kelompok yang berbeda dari F: $0{,}516 + 0{,}329 = 0{,}845$. Undian seragam hanya memberi $2/5 = 0{,}4$. Dari $\boldsymbol{\mu}_1 = F$ dan $\boldsymbol{\mu}_2 = A$, Lloyd sampai di SSE 7,875 dalam dua iterasi.

k-means++ dengan NumPy

```python
def kmeanspp_awal(X, k, rng):
    C = [X[rng.integers(len(X))]]                          # pusat pertama: acak
    for _ in range(1, k):
        D2 = ((X[:, None, :] - np.array(C)[None]) ** 2).sum(axis=2).min(axis=1)
        C.append(X[rng.choice(len(X), p=D2 / D2.sum())])  # titik jauh lebih mungkin
    return np.array(C)

C0 = kmeanspp_awal(X, 2, np.random.default_rng(0))       # F dan A
print(kmeans_lloyd(X, C0))                               # SSE 9.75 -> 7.875
```

- `D2` adalah jarak kuadrat setiap titik ke pusat terdekat yang sudah terpilih.
- `rng.choice(..., p=D2/D2.sum())` adalah undian dengan *weight* $D(\mathbf{x})^2$; titik yang sudah menjadi pusat mendapat *weight* 0.
- Dengan *seed* 0 undian memilih *F* lalu *A*, persis contoh di slide sebelumnya; Lloyd lalu mencatat SSE 9,75 $\rightarrow$ 7,875.

Kode lengkap: `bab08-knn-kmeans/kmeans.py`; `X` dan `kmeans_lloyd` dari slide kode Lloyd.

KMeans di scikit-learn

```
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
from sklearn.metrics import silhouette_score

X, _ = make_blobs(n_samples=300, centers=4, cluster_std=0.8, random_state=0)
for k in range(2, 9):
    km = KMeans(n_clusters=k, init="k-means++", n_init=10,
                random_state=0).fit(X)
    print(k, round(km.inertia_, 1), round(silhouette_score(X, km.labels_), 3))
```

- `KMeans` memakai *k-means++* sebagai bawaan; `n_init` mengulang seluruh prosedur dan mengambil SSE terkecil.
- Nilai bawaan `n_init` pernah berubah antarversi, jadi tuliskan secara eksplisit.
- `km.inertia_` adalah SSE; *output*-nya dipakai pada slide *elbow*.
- Data `make_blobs` sudah berskala sebanding. Pada data nyata (saldo, frekuensi transaksi), pasang `StandardScaler` sebelum `KMeans` dalam satu `Pipeline`, dengan alasan yang sama seperti kNN.

Memilih k: SSE saja tidak cukup

SSE selalu turun bila k ditambah, dan menjadi nol pada $k = n$ ketika setiap titik menjadi pusatnya sendiri.

Metode *elbow*

Gambarkan SSE terhadap k, lalu cari "siku": k setelah mana penurunan SSE melandai.

Silhouette (Rousseeuw, 1987)

Untuk setiap titik: a = rata-rata jarak ke anggota lain kelompoknya, b = rata-rata jarak ke anggota kelompok terdekat lainnya.

$$s = \frac{b - a}{\max(a, b)}, \qquad -1 \le s \le 1$$

Dekat 1: jauh lebih dekat ke kelompoknya sendiri. Dekat 0: di perbatasan. Negatif: mungkin salah kelompok.

Contoh, kelompok $\{A, B\}$ dan $\{C, D, E, F\}$.

Titik A: $a = \|A - B\| = \sqrt{1{,}25} = 1{,}118$, dan

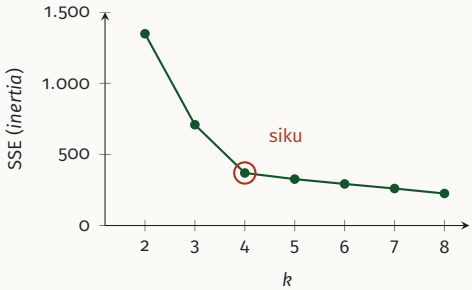
$$b = \tfrac{1}{4}(3{,}606 + 7{,}211 + 4{,}717 + 5{,}315) = 5{,}212,$$

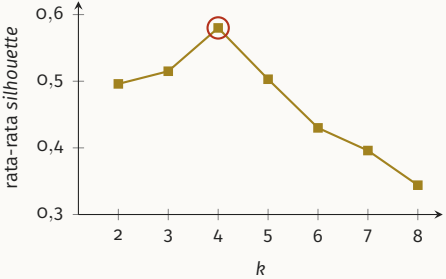
sehingga $s_A = (5{,}212 - 1{,}118)/5{,}212 = 0{,}785$.

Titik C: $a = \tfrac{1}{3}(3{,}606 + 1{,}118 + 1{,}803) = 2{,}176$ dan $b = \tfrac{1}{2}(3{,}606 + 2{,}500) = 3{,}053$, sehingga $s_C = (3{,}053 - 2{,}176)/3{,}053 = 0{,}287$.

C terletak di antara kedua kelompok, dan skornya yang rendah memperlihatkan itu. Rata-rata keenam skor adalah **0,614**.

Elbow dan *silhouette* pada 300 titik sintetis



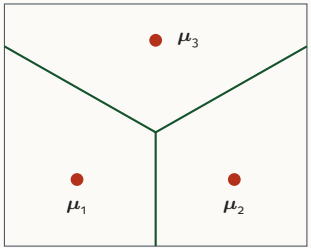


k	2	3	4	5	6	7	8
SSE	1.348,9	708,8	369,5	326,3	292,4	260,2	225,2
Silhouette	0,496	0,515	**0,580**	0,503	0,430	0,396	0,344

`make_blobs`, empat kelompok. SSE turun tajam sampai $k = 4$ lalu melandai, dan *silhouette* memuncak di $k = 4$. Pada data nyata kedua ukuran jarang sepakat setegas ini; pilihan k ikut ditentukan kebutuhan, misalnya berapa segmen yang sanggup dilayani tim pemasaran.

Keterbatasan *K-Means*

- **Kelompok dianggap bulat dan seukuran.** Batas antara dua kelompok selalu garis lurus (bidang) yang tegak lurus ruas penghubung kedua pusat; ruang terbagi menjadi sel-sel **Voronoi**. Kelompok memanjang, bersarang, atau berbentuk bulan sabit seperti `make_moons` dipotong di tempat yang salah.
- **Peka terhadap skala *feature*.** Sama dengan kNN; standardisasi lebih dulu.
- **Peka terhadap *outlier*.** Rata-rata ditarik nilai ekstrem. ***K-medoids*** memakai titik data sebagai pusat dan jumlah jarak (tanpa kuadrat), sehingga lebih tahan.
- k **harus ditentukan** dan hasilnya bergantung pada awal: pakai beberapa `n_init`, ukuran di atas, dan pertimbangan domain.



Tiga *centroid* membagi bidang menjadi tiga sel Voronoi. Setiap titik di dalam satu sel masuk ke kelompok pusat sel itu, apa pun bentuk kelompok aslinya.

K-Means sebagai GMM yang "keras"

GMM (Pertemuan 5) memberi setiap komponen *covariance* sendiri dan setiap titik peluang keanggotaan (*responsibility*). *K-Means* diperoleh dengan tiga penyederhanaan (Bishop, 2006): *weight* campuran sama, semua *covariance* $\sigma^2 I$, dan $\sigma \to 0$. Dengan dua penyederhanaan pertama, langkah E menjadi

$$r_{ij} = \frac{\exp\!\left(-\|\mathbf{x}_i - \boldsymbol{\mu}_j\|^2 / 2\sigma^2\right)}{\sum_{l=1}^{k} \exp\!\left(-\|\mathbf{x}_i - \boldsymbol{\mu}_l\|^2 / 2\sigma^2\right)}.$$

Saat σ mengecil, pusat terdekat mendominasi: $r_{ij} \to 1$ untuk pusat terdekat dan $\to 0$ untuk lainnya. Itulah langkah *assignment*. Langkah M, *weighted average* r_{ij}, menjadi rata-rata biasa: langkah *update*.

Titik C pada contoh enam titik. Jarak kuadrat ke $\boldsymbol{\mu}_1 = (1{,}25;\ 1{,}5)$ adalah 9,3125 dan ke $\boldsymbol{\mu}_2 = (4;\ 5{,}25)$ adalah 2,5625.

σ^2	peluang C di kelompok 2
4	0,699
1	0,967
0,25	$> 0{,}9999$

GMM dengan σ besar mengakui C berada di perbatasan, sesuai *silhouette*-nya yang hanya 0,287. *K-Means* langsung memasukkannya ke kelompok 2. Karena murah, *K-Means* juga dipakai menyiapkan rata-rata awal EM; `GaussianMixture` di scikit-learn melakukannya secara bawaan.

Praktikum, rangkuman, dan tugas

Praktikum: kNN dan *K-Means* pada data yang sama

1. **kNN pada Wine.** Jalankan kode *scaling* tadi untuk $k \in \{1, 3, 5, 9, 15, 25\}$. Ganti `StandardScaler` dengan `MinMaxScaler` dan bandingkan. Pilih k dengan `GridSearchCV` di dalam `Pipeline`, lalu laporkan *accuracy* pada *test data* yang disisihkan di awal.
2. ***Weighting.*** Ulangi dengan `weights="distance"`. Pada k berapa selisihnya terbesar?
3. **Lloyd dengan tangan dan kode.** Cocokkan *output* `kmeans_lloyd` dengan hitungan tangan untuk awal (A, B) dan (C, D).
4. **Memilih k.** Jalankan kode `make_blobs`, gambar *elbow* dan *silhouette*. Ulangi dengan `cluster_std=2.0`: apakah kedua ukuran masih sepakat?
5. **Batas *K-Means*.** `make_moons(n_samples=300, noise=0.08)`: jalankan `KMeans` dan `GaussianMixture` dengan $k = 2$, gambar hasilnya, lalu ukur *accuracy* kNN (10-fold CV) dengan label aslinya.

Yang dinilai

- *Scaling* dipasang di dalam `Pipeline`, tanpa kebocoran
- Tabel *accuracy* terhadap k, dengan dan tanpa *scaling*
- Hitungan tangan Lloyd cocok dengan *output* kode
- Grafik *elbow* dan *silhouette* beserta k pilihan dan alasannya
- Satu paragraf: mengapa *K-Means* gagal pada `make_moons` sedangkan kNN berhasil

Kumpulkan *notebook* yang dapat dijalankan ulang dari awal sampai akhir dengan *seed* tercatat.

Rangkuman

kNN

- Tanpa *training phase*: simpan data, jawab dengan suara terbanyak atau rata-rata k tetangga, bisa diberi *weight* $1/d^2$
- Keputusan bergantung pada *distance metric*; *scaling* dari *training data* adalah bagian dari model
- k (atau h pada LWR) kecil: *variance* tinggi; besar: bias tinggi; pilih dengan *cross-validation*
- Menjawab berbiaya $O(nd)$; pada dimensi tinggi jarak terdekat dan terjauh hampir sama
- KD-tree dan Ball-tree untuk dimensi rendah; ANN (HNSW) di balik *vector database*

K-Means

- Mengelompokkan data tanpa label dengan meminimalkan SSE
- Lloyd: *assignment* ke pusat terdekat, *update* ke rata-rata; SSE tidak pernah naik
- Hasilnya minimum lokal (awal C, D: SSE 10,67 lawan 7,875); *k-means++* dan `n_init` mengurangi risiko itu
- k dipilih dengan *elbow*, *silhouette*, dan kebutuhan domain
- GMM dengan *covariance* $\sigma^2 I$ yang sama dan $\sigma \to 0$: mewarisi asumsi kelompok bulat, peka skala dan *outlier*

Kuis singkat dan tugas

Kuis (5 menit)

1. Mengapa *training error* kNN dengan $k = 1$ selalu nol, dan mengapa angka itu tidak berguna untuk memilih k?
2. Pada contoh $\mathbf{x}_q = (5, 5)$, apa keputusan kNN dengan *weight* $1/d^2$ untuk $k = 5$?
3. Mengapa SSE terkecil tidak dapat dipakai untuk memilih k pada *K-Means*?
4. Sebut satu perbedaan arti k pada kNN dan pada *K-Means*.

Tugas

1. Baca buku ajar Bab 8 dan Mitchell Bab 8; Bishop 9.1 sebagai pendalaman.
2. Ulangi contoh pemohon pembiayaan dengan standardisasi z: rata-rata gaji Rp9 juta (simpangan baku Rp3 juta), rata-rata umur 38 (simpangan baku 10). Siapa tetangga terdekat $\mathbf{x}_q$?
3. Jalankan Lloyd dengan tangan dari $\boldsymbol{\mu}_1 = C$, $\boldsymbol{\mu}_2 = D$; tulis label dan SSE tiap iterasi.
4. Bila undian pertama *k-means++* memilih A, hitung peluang setiap titik menjadi pusat kedua dan peluang pusat kedua jatuh di $\{C, D, E, F\}$.
5. Hitung *silhouette* titik D dan F; bandingkan dengan $s_C = 0{,}287$.
6. Selesaikan praktikum dan kumpulkan *notebook*-nya.

Pertemuan depan

SVM mencari *hyperplane* bermargin terbesar dan hanya menyimpan *support vector*; HMM memodelkan urutan lewat keadaan tersembunyi.

Terima kasih

Pertanyaan?