Decision Tree • Feature Engineering • Grid Search (Ringan)

Fokus hari ini

- Intuisi & praktik **Decision Tree**
- **Feature Engineering** (angka, kategori, waktu, teks)
- **Grid Search** ringan (aman untuk pemula)

Tujuan Sesi

- Memahami cara pohon keputusan membuat **split**
- Bisa **menghitung Gini/Entropy** sederhana untuk satu split
- Menyusun **pipeline** dengan langkah **feature engineering** dasar
- Melakukan **grid search kecil** untuk memilih parameter tanpa overkill

Bagian A — Decision Tree (DT)

Intuisi Decision Tree

- Model berbentuk **pohon pertanyaan**
- Tiap **node** memilih fitur & nilai batas untuk **memisahkan kelas**
- Berhenti ketika **homogen**, **kedalaman maksimum**, atau **data terlalu sedikit**

Kelebihan

- Mudah dijelaskan (if–else)
- Tangani angka & kategori

Kekurangan

- **Overfitting** jika dibiarkan tumbuh tanpa batas
- Batas keputusan berbentuk **axis-aligned** (garis tegak/lurus)

Ukuran Ketidakmurnian (Impurity)

Gini:

$$\text{Gini}(S) = 1 - \sum_k p_k^2$$

Entropy:

$$H(S) = -\sum_k p_k \log_2 p_k$$

Gain (peningkatan kemurnian):

$$\text{Gain} = \text{Impurity}(S) - \sum_j \frac{|S_j|}{|S|} \text{Impurity}(S_j)$$

> Skikit-learn default **Gini** untuk klasifikasi, **MSE/MAE** untuk regresi.

Hitung Manual (Kecil & Jelas)

Dataset mini (10 baris), target: **Spam (1)** vs **Ham (0)**, kandidat split: fitur **has_gratis** (0/1).

Kelas	Count
Spam (1)	4
Ham (0)	6

Awal:

- $p_{spam} = 4/10, p_{ham} = 6/10$
- **Gini** awal $= 1 - (0.4^2 + 0.6^2) = 1 - (0.16 + 0.36) = 0.48$

Split by **has_gratis**:

- Left (has_gratis=1): 4 contoh → 3 Spam, 1 Ham
  $\text{Gini}_L = 1 - (0.75^2 + 0.25^2) = 1 - (0.5625 + 0.0625) = 0.375$
- Right (has_gratis=0): 6 contoh → 1 Spam, 5 Ham
  $\text{Gini}_R = 1 - (\frac{1}{6})^2 - (\frac{5}{6})^2 = 1 - (0.0278 + 0.6944) = 0.2778$

Gini setelah split
$= \frac{4}{10} \cdot 0.375 + \frac{6}{10} \cdot 0.2778 = 0.15 + 0.1667 = 0.3167$

Gain $= 0.48 - 0.3167 = 0.1633$ (bagus → impurity turun)

Kontrol Overfitting (Pruning)

Parameter penting (sklearn `DecisionTreeClassifier`):

- `max_depth` (batasi kedalaman)
- `min_samples_split`, `min_samples_leaf`
- `max_leaf_nodes`
- `ccp_alpha` (cost-complexity pruning)

Aturan ringkas

- Mulai dari **depth kecil** (mis. 3–6)
- Gunakan **min_samples_leaf** (mis. 2–5) agar daun tidak terlalu kecil
- Pakai **CV**/validasi untuk memilih parameter

Kode Minimal (Iris, klasifikasi)

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt

X, y = load_iris(return_X_y=True)
Xtr, Xte, ytr, yte = train_test_split(X, y, test_size=0.2, stratify=y, random_state=42)

dt = DecisionTreeClassifier(max_depth=4, random_state=42)
dt.fit(Xtr, ytr)
print(classification_report(yte, dt.predict(Xte)))

cm = confusion_matrix(yte, dt.predict(Xte))
plt.figure(); plt.imshow(cm); plt.title("CM – DT"); plt.xlabel("Pred"); plt.ylabel("True")
for (i,j),v in __import__("numpy").ndenumerate(cm): plt.text(j,i,str(v),ha="center",va="center")
plt.colorbar(); plt.show()

plt.figure(figsize=(8,5))
plot_tree(dt, feature_names=load_iris().feature_names, class_names=load_iris().target_names, filled=True)
plt.show()
```

Bagian B — Feature Engineering (FE)

Prinsip FE (Praktis)

- **Tujuan**: membuat fitur yang **memperjelas pola**
- **Sederhana dulu** → iterasi
- **Anti-leakage**: semua transformasi **fit di train** (pakai **Pipeline**)

Kategori umum

- **Numerik**: scaling, binning (quantile/bin), interaction ($x_1 \cdot x_2$), polynomial ringan
- **Kategori**: one-hot, target count (hati-hati leakage)
- **Waktu**: pecah ke `year, month, day, dow, hour`, bagian siklik (`sin/cos`)
- **Teks**: **TF-IDF** / n-gram pendek (baseline kuat)

Contoh FE — Numerik & Kategori

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler, PolynomialFeatures
from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split

# misal df: num_cols, cat_cols sudah didefinisikan
num_pipe = Pipeline([
  ("imp", SimpleImputer(strategy="median")),
  ("sc", StandardScaler()),
  ("poly", PolynomialFeatures(degree=2, include_bias=False))  # ringan; bisa dimatikan
])
cat_pipe = Pipeline([
  ("imp", SimpleImputer(strategy="most_frequent")),
  ("oh", OneHotEncoder(handle_unknown="ignore"))
])

pre = ColumnTransformer([
  ("num", num_pipe, num_cols),
  ("cat", cat_pipe, cat_cols),
])

clf = Pipeline([
  ("pre", pre),
  ("dt", DecisionTreeClassifier(max_depth=5, random_state=42))
])
clf.fit(X_train, y_train)
```

Contoh FE — Waktu & Teks (Ringkas)

Waktu → fitur siklik

- Jam 23 dan 00 itu **berdekatan**: gunakan

$$\sin\_hour = \sin\left(2\pi \cdot \frac{\text{hour}}{24}\right), \quad \cos\_hour = \cos\left(2\pi \cdot \frac{\text{hour}}{24}\right)$$

Teks → TF-IDF

```
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression

pipe_text = Pipeline([
  ("tfidf", TfidfVectorizer(min_df=3, ngram_range=(1,2))),
  ("clf", LogisticRegression(max_iter=1000))
])
```

Bagian C — Grid Search (Ringan)

Kenapa Grid Search?

- Cari kombinasi **hyperparameter** terbaik secara **terkontrol**
- Gunakan **grid kecil** + **CV** (mis. 3–5 fold)
- **Hindari** grid besar (waktu habis, overfitting ke val)

Tips

- Tuning yang **berpengaruh** dulu: `max_depth`, `min_samples_leaf` (DT)
- Simpan **seed** dan laporkan **mean ± std** skor

Contoh Grid Search Kecil (DT + FE)

```
from sklearn.model_selection import GridSearchCV, StratifiedKFold

cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
param_grid = {
  "dt__max_depth": [3, 5, 7],
  "dt__min_samples_leaf": [1, 3, 5]
}

grid = GridSearchCV(
  estimator=clf,           # Pipeline(pre -> dt)
  param_grid=param_grid,
  cv=cv,
  scoring="f1_macro",      # atau "accuracy" tergantung konteks
  n_jobs=-1
)
grid.fit(X_train, y_train)

print("Best params:", grid.best_params_)
print("CV best score:", grid.best_score_)
best = grid.best_estimator_
print("Test score:", best.score(X_test, y_test))
```

Catatan: gunakan **pipeline** sebagai estimator agar **transformasi FE** ikut tervalidasi di setiap fold (anti-leakage).

Membaca Hasil & Next Step

- Lihat **best_params_** → apakah masuk akal?
- Bandingkan **skor CV vs Test** → indikasi overfit jika drop besar
- Simpan model (opsional) dan tulis **3–5 rencana perbaikan**:
  - kurangi/ubah `max_depth`
  - ubah FE (hapus `poly`, tambah interaction tertentu)
  - coba model pembanding (LogReg, RandomForest baseline)

Kesalahan Umum

- **Leakage**: fit scaler/encoder **sebelum** split/CV → salah.
  ✓ Selalu bungkus di **Pipeline**.
- Grid terlalu besar → **waktu lama** dan rawan **overfit CV**.
  ✓ Mulai **kecil**; perluas bertahap.
- FE berlebihan (poly tinggi) saat data sedikit.
  ✓ Sederhana dulu, ukur dampak.

Checklist Praktis (Output Tugas)

- [ ] Jelaskan masalah & metrik (1–2 kalimat)
- [ ] DT baseline **jalan** + CM + report
- [ ] FE dasar **jalan** (impute, oh/scale, mungkin 1–2 fitur turunan)
- [ ] Grid search **kecil** (2×3 atau 3×3) + CV 5-fold
- [ ] Ringkas hasil: **best params, mean±std CV, skor test**, 3 rencana perbaikan

Tugas Mini

1. Buat **DT baseline** (tanpa FE), catat accuracy/F1.
2. Tambahkan **FE sederhana** (scaling + 1 fitur interaksi) → ukur perubahannya.
3. Jalankan **grid kecil** pada `max_depth` dan `min_samples_leaf`.
4. Tulis **2–3 kalimat** tentang overfitting/pruning yang kamu lihat.

Ringkasan

- **Decision Tree**: jelas & kuat, tapi **perlu kontrol** (depth/leaf/pruning).
- **Feature Engineering**: sederhana → iterasi; **anti-leakage** pakai **Pipeline**.
- **Grid Search ringan**: kecil tapi informatif; pakai **CV** dan **scoring** yang sesuai.