In [ ]:

```
# ===== 0) 데이터 예시 =====
import numpy as np
import pandas as pd
from sklearn.datasets import make_classification
import warnings
import warnings
warnings.filterwarnings("ignore", category=UserWarning, module="sklearn.model_selection")

# 예시용: 숫자 + 범주 혼합 데이터 구성
X_num, y = make_classification(n_samples=2000, n_features=6, n_informative=4, random_state=42)
df_num = pd.DataFrame(X_num, columns=[f"num{i}" for i in range(X_num.shape[1])])

rng = np.random.default_rng(42)
df_cat = pd.DataFrame({
    "cat0": rng.choice(["A","B","C","D","E"], size=len(df_num)),
    # "cat1": rng.choice(["X","Y","Z"], size=len(df_num)),
})
X_df = pd.concat([df_num, df_cat], axis=1)

num_cols = [c for c in X_df.columns if c.startswith("num")]
cat_cols = [c for c in X_df.columns if c.startswith("cat")]

# ===== 1) 전처리 파이프라인 =====
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder

preprocess = ColumnTransformer(
    transformers=[
        ("num", StandardScaler(with_mean=True, with_std=True), num_cols),
        ("cat", OneHotEncoder(handle_unknown="ignore", sparse_output=False), cat_cols),
    ],
    remainder="drop",
    verbose_feature_names_out=False
)

# ===== 2) 앙상블(스태킹) 정의 =====
from sklearn.pipeline import Pipeline
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier, HistGradientBoostingClassifier, StackingClassifier
from sklearn.linear_model import LogisticRegression

base_estimators = [
    ("svm", SVC(probability=True, kernel="rbf", random_state=42)),
    ("rf", RandomForestClassifier(random_state=42)),
    ("hgb", HistGradientBoostingClassifier(random_state=42)),
]

final_estimator = LogisticRegression(max_iter=1000, n_jobs=-1, random_state=42)
# StackingClassifier는 순차적 연결이 아니라, 병렬 base 모델들의 예측을 meta 모델 입력으로 쓰는 구조
stacking = StackingClassifier(
    estimators=base_estimators,
    final_estimator=final_estimator,
    passthrough=False,           # 원본 특성까지 최종기에 넣으려면 True
    stack_method="auto",
    n_jobs=-1
)

pipe = Pipeline([
    ("prep", preprocess),
    ("clf", stacking),
])

# ===== 3) 하이퍼파라미터 탐색 공간 =====
from sklearn.experimental import enable_halving_search_cv  # noqa: F401
from sklearn.model_selection import HalvingRandomSearchCV, StratifiedKFold, KFold
from scipy.stats import loguniform, randint

param_dist = {
    # SVM (RBF)
    "clf__svm__C": loguniform(1e-2, 1e2),
    "clf__svm__gamma": loguniform(1e-4, 1e0),

    # RandomForest
    "clf__rf__n_estimators": randint(100, 600),
    "clf__rf__max_depth": randint(3, 20),
    "clf__rf__min_samples_split": randint(2, 10),

    # HistGradientBoosting
    "clf__hgb__learning_rate": loguniform(1e-2, 3e-1),
    "clf__hgb__max_depth": randint(3, 20),
    "clf__hgb__l2_regularization": loguniform(1e-6, 1e-1),

    # 최종 로지스틱
    "clf__final_estimator__C": loguniform(1e-2, 1e2),
}

cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
# cv = KFold(n_splits=5, shuffle=True, random_state=42)
search = HalvingRandomSearchCV(
    estimator=pipe,
    param_distributions=param_dist,
    factor=3,                   # 샘플/파라미터 줄여가는 속도
    random_state=42,
    cv=cv,
    scoring="roc_auc_ovo_weighted",
    n_jobs=-1,
    verbose=0
)

search.fit(X_df, y)

print("best score:", search.best_score_)
print("best params:")
for k, v in search.best_params_.items():
    print("  ", k, "=", v)

best_model = search.best_estimator_
```