```
# -*- coding: utf-8 -*-

### SCRIPT 14 - PYTHON
# COMPARADOR DE MODELOS PRECICCION CLUSTER
#
# =================================================================


✅ Celda 1 — Instalación de librerías (solo una vez)
"""

# ¡Ejecuta esta celda primero!
!pip -q install pandas numpy scikit-learn matplotlib joblib xgboost lightgbm

"""✅ Celda 2 — Imports + utilidades de gráficos"""

import numpy as np, pandas as pd, matplotlib.pyplot as plt, joblib
from pathlib import Path

from sklearn.model_selection import StratifiedKFold, cross_validate
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder, StandardScaler, MinMaxScaler
from sklearn.impute import SimpleImputer
from sklearn.metrics import (
    accuracy_score, f1_score, log_loss, confusion_matrix, top_k_accuracy_score
)

from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import RandomForestClassifier, HistGradientBoostingClassifier
# Opcionales (instalados en Celda 1)
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier

RANDOM_STATE = 42

def plot_confusion_matrix(classes_, y_true, y_pred, out_path):
    cm = confusion_matrix(y_true, y_pred, labels=classes_)
    plt.figure(figsize=(6,5))
    plt.imshow(cm, interpolation='nearest')
    plt.title("Matriz de confusión")
    plt.colorbar()
    ticks = np.arange(len(classes_))
    plt.xticks(ticks, classes_, rotation=45, ha='right')
    plt.yticks(ticks, classes_)
    # anotar celdas
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            plt.text(j, i, format(cm[i, j], 'd'), ha="center", va="center")
    plt.ylabel('Real')
    plt.xlabel('Predicho')
    plt.tight_layout()
    plt.savefig(out_path, dpi=160)
    plt.close()

def plot_scatter_real_vs_pred(classes_, y_true, y_pred, out_path):
    label_to_idx = {label: idx for idx, label in enumerate(classes_)}
    yt = np.array([label_to_idx.get(v, np.nan) for v in y_true])
    yp = np.array([label_to_idx.get(v, np.nan) for v in y_pred])
    x = np.arange(len(yt))
    plt.figure(figsize=(8,4))
    plt.scatter(x, yt, alpha=0.6, label="Real")
    plt.scatter(x, yp, alpha=0.6, marker="x", label="Predicho")
    plt.yticks(np.arange(len(classes_)), classes_)
    plt.title("Real vs Predicho (scatter)")
    plt.xlabel("Índice de muestra (test)")
    plt.ylabel("Clase")
    plt.legend()
    plt.tight_layout()
    plt.savefig(out_path, dpi=160)
    plt.close()

"""✅ Celda 3 — Carga de datos (SUBIR archivos)"""

from google.colab import files
uploaded = files.upload()  # Selecciona: df_train_Clusterk6_v8.csv y df_test_Clusterk6_v8.csv
list(uploaded.keys())

"""✅ Celda 4 — Entrenamiento (CV 5-fold) + Evaluación en TEST + Gráficos"""

# Rutas (si usaste Opción A, se guardan en /content)
train_path = "df_train_Clusterk6_v8.csv"
test_path  = "df_test_Clusterk6_v8.csv"

ARTIF_DIR = Path("/content/cluster6_artifacts")
ARTIF_DIR.mkdir(parents=True, exist_ok=True)

TARGET = "Cluster_6"
ID_COL = "Identificador"  # si no existe no pasa nada

# --- Carga ---
df_train = pd.read_csv(train_path)
df_test  = pd.read_csv(test_path)
assert TARGET in df_train.columns and TARGET in df_test.columns, "Falta Cluster_6 en algún CSV"

# --- X/y (quitamos objetivo e Identificador) ---
X_train = df_train.drop(columns=[c for c in [TARGET, ID_COL] if c in df_train.columns])
y_train = df_train[TARGET].astype("category")
X_test  = df_test.drop(columns=[c for c in [TARGET, ID_COL] if c in df_test.columns])
y_test  = df_test[TARGET].astype("category")

# --- Columnas num/cat desde TRAIN ---
num_cols = list(X_train.select_dtypes(include=[np.number]).columns)
cat_cols = [c for c in X_train.columns if c not in num_cols]

# --- Preprocesadores ---
preproc_std = ColumnTransformer([
    ("num", Pipeline([("imp", SimpleImputer(strategy="median")), ("sc", StandardScaler())]), num_cols),
    ("cat", Pipeline([("imp", SimpleImputer(strategy="most_frequent")),
                      ("oh", OneHotEncoder(handle_unknown="ignore"))]), cat_cols),
], remainder="drop")

preproc_nb = ColumnTransformer([
    ("num", Pipeline([("imp", SimpleImputer(strategy="median")), ("mm", MinMaxScaler())]), num_cols),
    ("cat", Pipeline([("imp", SimpleImputer(strategy="most_frequent")),
                      ("oh", OneHotEncoder(handle_unknown="ignore"))]), cat_cols),
], remainder="drop")

# --- Modelos a comparar (≥5) ---
models = [
    ("RandomForest",         Pipeline([("prep", preproc_std), ("clf", RandomForestClassifier(
                                n_estimators=400, max_features="sqrt", random_state=RANDOM_STATE, n_jobs=-1))])),
    ("HistGradientBoosting", Pipeline([("prep", preproc_std), ("clf", HistGradientBoostingClassifier(
                                random_state=RANDOM_STATE))])),
    ("LogisticRegression",   Pipeline([("prep", preproc_std), ("clf", LogisticRegression(
                                multi_class="multinomial", solver="lbfgs", max_iter=800, random_state=RANDOM_STATE))])),
    ("SVC",                  Pipeline([("prep", preproc_std), ("clf", SVC(
                                kernel="rbf", probability=True, random_state=RANDOM_STATE))])),
    ("NeuralNet_MLP",        Pipeline([("prep", preproc_std), ("clf", MLPClassifier(
                                hidden_layer_sizes=(128,64), activation="relu", max_iter=300,
                                random_state=RANDOM_STATE, early_stopping=True))])),
    ("XGBoost",              Pipeline([("prep", preproc_std), ("clf", XGBClassifier(
                                random_state=RANDOM_STATE, n_estimators=600, learning_rate=0.05, max_depth=6,
                                subsample=0.9, colsample_bytree=0.9, objective="multi:softprob",
                                tree_method="hist", eval_metric="mlogloss", n_jobs=-1))])),
    ("LightGBM",             Pipeline([("prep", preproc_std), ("clf", LGBMClassifier(
                                random_state=RANDOM_STATE, n_estimators=800, learning_rate=0.05,
                                subsample=0.9, colsample_bytree=0.9, n_jobs=-1))])),
]

# --- CV 5-fold ---
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=RANDOM_STATE)

cv_rows, test_rows = [], []
bars_acc, bars_f1, bars_ll = [], [], []

for name, pipe in models:
    print(f"\n=== {name} ===")
    # CV
    scoring = {"accuracy": "accuracy", "f1_macro": "f1_macro"}
    cv_res = cross_validate(pipe, X_train, y_train, cv=cv, scoring=scoring, n_jobs=-1)
    cv_rows.append({
        "Modelo": name,
        "CV5_Accuracy_mean": cv_res["test_accuracy"].mean(),
        "CV5_Accuracy_std": cv_res["test_accuracy"].std(ddof=1),
        "CV5_MacroF1_mean": cv_res["test_f1_macro"].mean(),
        "CV5_MacroF1_std": cv_res["test_f1_macro"].std(ddof=1),
    })
    print(f"CV5 Acc={cv_rows[-1]['CV5_Accuracy_mean']:.4f} ± {cv_rows[-1]['CV5_Accuracy_std']:.4f} | "
          f"MacroF1={cv_rows[-1]['CV5_MacroF1_mean']:.4f} ± {cv_rows[-1]['CV5_MacroF1_std']:.4f}")

    # Fit completo + Test
    pipe.fit(X_train, y_train)
    y_pred = pipe.predict(X_test)

    proba_supported = hasattr(pipe.named_steps["clf"], "predict_proba")
    y_proba = pipe.predict_proba(X_test) if proba_supported else None

    # Si Test tiene clases no vistas en Train, filtramos para métricas
    classes_ = pipe.named_steps["clf"].classes_
    mask_known = y_test.isin(classes_)
    y_test_eval = y_test[mask_known]
    y_pred_eval = y_pred[mask_known]
    y_proba_eval = y_proba[mask_known] if y_proba is not None else None

    acc = accuracy_score(y_test_eval, y_pred_eval)
    macro_f1 = f1_score(y_test_eval, y_pred_eval, average="macro")
    ll = (log_loss(y_test_eval, y_proba_eval, labels=classes_) if proba_supported else np.nan)
    hit3 = (top_k_accuracy_score(y_test_eval, y_proba_eval, k=min(3, len(classes_)))
            if proba_supported else np.nan)

    test_rows.append({"Modelo": name, "Test_Accuracy": acc, "Test_MacroF1": macro_f1,
                      "Test_LogLoss": ll, "Test_Hit@3": hit3})
    print(f"TEST Acc={acc:.4f} | MacroF1={macro_f1:.4f} | "
          f"LogLoss={ll if proba_supported else np.nan:.4f} | Hit@3={hit3 if proba_supported else np.nan:.4f}")

    # Artefactos por modelo
    mdir = ARTIF_DIR / name
    mdir.mkdir(parents=True, exist_ok=True)
    joblib.dump(pipe, mdir / f"model_{name}.pkl")

    pred_df = pd.DataFrame({"y_true": y_test.to_numpy(), "y_pred": y_pred})
    if ID_COL in df_test.columns:
        pred_df[ID_COL] = df_test[ID_COL].values
```

```python
        pred_df = pred_df[[ID_COL, "y_true", "y_pred"]]
    if y_proba is not None:
        proba_df = pd.DataFrame(y_proba, columns=[f"proba_{c}" for c in classes_])
        pred_df = pd.concat([pred_df.reset_index(drop=True), proba_df], axis=1)
    pred_df.to_csv(mdir / f"test_predictions_{name}.csv", index=False)

    # Gráficos por modelo
    plot_confusion_matrix(classes_, y_test_eval, y_pred_eval, mdir / "confusion_matrix.png")
    plot_scatter_real_vs_pred(classes_, y_test_eval, y_pred_eval, mdir / "scatter_real_vs_pred.png")

    # para comparativa global
    bars_acc.append((name, acc))
    bars_f1.append((name, macro_f1))
    if not np.isnan(ll): bars_ll.append((name, ll))

# Resúmenes y comparativas
cv_df   = pd.DataFrame(cv_rows).sort_values("CV5_MacroF1_mean", ascending=False)
test_df = pd.DataFrame(test_rows).sort_values("Test_MacroF1", ascending=False)
cv_df.to_csv(ARTIF_DIR / "cv5_summary.csv", index=False)
test_df.to_csv(ARTIF_DIR / "test_summary.csv", index=False)

display(cv_df.head(10))
display(test_df.head(10))

# Gráfico comparativo (similar a tu imagen)
def plot_bars(pairs, title, ylabel, out_name, highlight="RandomForest"):
    names = [p[0] for p in pairs]
    vals  = [p[1] for p in pairs]
    plt.figure(figsize=(9,4))
    bars = plt.bar(names, vals)
    if highlight in names:
        idx = names.index(highlight)
        bars[idx].set_linewidth(3.0)
        bars[idx].set_edgecolor("black")
    for i, v in enumerate(vals):
        plt.text(i, v + 0.002, f"{v:.3f}", ha='center', va='bottom')
    plt.title(title)
    plt.ylabel(ylabel)
    plt.xticks(rotation=25, ha='right')
    plt.tight_layout()
    plt.savefig(ARTIF_DIR / out_name, dpi=180)
    plt.show()
    plt.close()

plot_bars(bars_acc, "Comp Modelos Predictivos Cluster_6 - Test (Accuracy)", "Test Accuracy",
          "comparativa_test_accuracy.png")
plot_bars(bars_f1, "Comp Modelos Predictivos Cluster_6 - Test (Macro-F1)", "Test Macro-F1",
          "comparativa_test_macrof1.png")

if len(bars_ll) > 0:
    # LogLoss: más bajo es mejor
    bars_ll_sorted = sorted(bars_ll, key=lambda x: x[1])
    plot_bars(bars_ll_sorted, "Comp Modelos Predictivos Cluster_6 - Test (LogLoss ↓)", "Test LogLoss",
              "comparativa_test_logloss.png")

"""Celda 4.0.5 — “Resolver 1 vs 4” (cascada sobre tu RF)"""

# CARGA: pipeline RF ya guardado por la Celda 4
from pathlib import Path
import numpy as np, pandas as pd, matplotlib.pyplot as plt, joblib
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, f1_score, confusion_matrix, log_loss, top_k_accuracy_score
```

```python

ARTIF_DIR = Path("/content/cluster6_artifacts")
rf_path   = ARTIF_DIR / "RandomForest" / "model_RandomForest.pkl"
assert rf_path.exists(), "No encuentro el modelo base RandomForest. Ejecuta antes la Celda 4."

# Reutilizamos X_train, y_train, X_test, y_test definidos en la Celda 4
rf_pipe = joblib.load(rf_path)
classes_rf = list(rf_pipe.named_steps["clf"].classes_)

# ---------- 1) Entrenar clasificador binario 1 vs 4 ----------
pair = [1, 4]  # los clusters conflictivos
mask_pair_train = y_train.isin(pair)

# Preprocesador igual al del RF (estaba dentro del pipeline). Lo reconstruimos por seguridad:
num_cols = list(X_train.select_dtypes(include=[np.number]).columns)
cat_cols = [c for c in X_train.columns if c not in num_cols]
preproc_std = ColumnTransformer([
    ("num", Pipeline([("imp", SimpleImputer(strategy="median")), ("sc", StandardScaler())]), num_cols),
    ("cat", Pipeline([("imp", SimpleImputer(strategy="most_frequent")),
                      ("oh", OneHotEncoder(handle_unknown="ignore"))]), cat_cols),
])

# Modelo binario: simple y fuerte para separar dos clases
bin_pipe = Pipeline([
    ("prep", preproc_std),
    ("clf", LogisticRegression(max_iter=1000, random_state=42))
])

bin_pipe.fit(X_train[mask_pair_train], y_train[mask_pair_train])

# ---------- 2) Predicción base + lógica de ruteo ----------
# Predicción y probabilidades del RF
y_pred_rf = rf_pipe.predict(X_test)
proba_rf  = rf_pipe.predict_proba(X_test)  # columnas en el orden classes_rf

# Índices de probas de 1 y 4
i1 = classes_rf.index(1) if 1 in classes_rf else None
i4 = classes_rf.index(4) if 4 in classes_rf else None
assert i1 is not None and i4 is not None, "El RF no ha visto alguna de las clases {1,4} en train."

p1 = proba_rf[:, i1]
p4 = proba_rf[:, i4]

# Regla de activación del “resolver”:
# - si el RF predice 1 o 4
# - o si está indeciso entre 1 y 4 (|p1 - p4| < delta) y su masa conjunta es razonable (p1+p4 >= umbral)
delta   = 0.08  # tolerancia de indecisión entre 1 y 4 (ajustable)
umbral  = 0.50  # masa mínima en {1,4} para considerarlo candidato (ajustable)

cand_mask = np.isin(y_pred_rf, pair) | ((np.abs(p1 - p4) < delta) & ((p1 + p4) >= umbral))

# Predicción binaria SOLO en candidatos
y_pred_pair = y_pred_rf.copy()
if cand_mask.any():
    # Para el binario usamos EXACTAMENTE las mismas filas de test candidatas
    y_pred_pair[cand_mask] = bin_pipe.predict(X_test[cand_mask])

# ---------- 3) Evaluación global y foco 1↔4 ----------
def eval_and_plot(y_true, y_pred, title, out_dir):
    out_dir.mkdir(parents=True, exist_ok=True)
    # Métricas globales
    acc = accuracy_score(y_true, y_pred)
    macro = f1_score(y_true, y_pred, average="macro")
    print(f"{title} -> Acc={acc:.4f} | Macro-F1={macro:.4f}")

    # Matriz completa
    classes_all = sorted(pd.unique(pd.concat([y_true, pd.Series(classes_rf)])))
    cm = confusion_matrix(y_true, y_pred, labels=classes_all)
    plt.figure(figsize=(6,5))
    plt.imshow(cm, interpolation='nearest')
    plt.title(title)
    plt.colorbar()
    ticks = np.arange(len(classes_all))
    plt.xticks(ticks, classes_all, rotation=45, ha='right')
    plt.yticks(ticks, classes_all)
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            plt.text(j, i, format(cm[i, j], 'd'), ha="center", va="center")
    plt.ylabel("Eje Y: Reales")
    plt.xlabel("Eje X: Predicciones")
    plt.tight_layout()
    plt.savefig(out_dir / "confusion_matrix.png", dpi=180)
    plt.close()

    # Matriz SOLO 1 vs 4 (para ver la mejora puntual)
    cm14 = confusion_matrix(y_true, y_pred, labels=pair)
    plt.figure(figsize=(4,4))
    plt.imshow(cm14, interpolation='nearest')
    plt.title(title + " — foco 1 vs 4")
    plt.colorbar()
    ticks = np.arange(len(pair))
    plt.xticks(ticks, pair)
    plt.yticks(ticks, pair)
    for i in range(2):
        for j in range(2):
            plt.text(j, i, format(cm14[i, j], 'd'), ha="center", va="center")
    plt.ylabel("Reales")
    plt.xlabel("Predicciones")
    plt.tight_layout()
    plt.savefig(out_dir / "confusion_matrix_1v4.png", dpi=180)
    plt.close()
    return acc, macro, cm, cm14

# Evaluación RF base
acc_rf, f1_rf, cm_rf, cm14_rf = eval_and_plot(y_test, y_pred_rf, "RandomForest (base)", ARTIF_DIR / "RF_base_eval")

# Evaluación RF + resolver 1v4
acc_res, f1_res, cm_res, cm14_res = eval_and_plot(y_test, y_pred_pair, "RF + Resolver 1↔4", ARTIF_DIR / "RF_resolver_eval")

# Guardar CSV de predicciones y comparación
pd.DataFrame({
    "y_true": y_test.to_numpy(),
    "pred_rf": y_pred_rf,
    "pred_rf_resolver": y_pred_pair,
    "p1": p1, "p4": p4, "resolver_aplicado": cand_mask.astype(int)
}).to_csv(ARTIF_DIR / "rf_vs_resolver_predictions.csv", index=False)

print("\nGuardado todo en:", ARTIF_DIR)
print("Sugerencia: ajusta delta y umbral para mover la balanza (más/menos agresivo) en la resolución 1↔4.")

"""✅ Celda 4.1 — Matrices de confusión (Y=Reales, X=Predicciones)"""

# Genera matrices de confusión por modelo con ejes explícitos
# (lee /content/cluster6_artifacts/<Modelo>/test_predictions_*.csv)

from pathlib import Path
import pandas as pd, numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix

ARTIF_DIR = Path("/content/cluster6_artifacts")  # misma ruta usada en la Celda 4
model_dirs = [d for d in ARTIF_DIR.iterdir() if d.is_dir()]

for mdir in model_dirs:
    pred_csvs = list(mdir.glob("test_predictions_*.csv"))
    if not pred_csvs:
        print(f"Saltando {mdir.name}: no hay test_predictions_*.csv")
        continue

    dfp = pd.read_csv(pred_csvs[0])
    if not {"y_true", "y_pred"}.issubset(dfp.columns):
        print(f"Saltando {mdir.name}: faltan columnas y_true/y_pred")
        continue

    y_true = dfp["y_true"]
    y_pred = dfp["y_pred"]

    # Orden de clases: unión de etiquetas presentes en true y pred (orden alfabético)
    classes_ = sorted(pd.unique(pd.concat([y_true, y_pred], ignore_index=True)))

    cm = confusion_matrix(y_true, y_pred, labels=classes_)

    # Plot (Y = reales, X = predicciones)
    plt.figure(figsize=(6,5))
    plt.imshow(cm, interpolation='nearest')
    plt.title(f"Matriz de confusión — {mdir.name}")
    plt.colorbar()
    ticks = np.arange(len(classes_))
    plt.xticks(ticks, classes_, rotation=45, ha='right')
    plt.yticks(ticks, classes_)
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            plt.text(j, i, format(cm[i, j], 'd'), ha="center", va="center")
    plt.ylabel("Eje Y: Reales")
    plt.xlabel("Eje X: Predicciones")
    plt.tight_layout()

    out_png = mdir / "confusion_matrix_Yreal_Xpred.png"
    plt.savefig(out_png, dpi=180)
    plt.show()
    plt.close()

print("Listo: guardadas como 'confusion_matrix_Yreal_Xpred.png' en cada carpeta de modelo.")

"""✅ Celda 5 — Descargar todos los artefactos"""

# Crea un ZIP con todo y te lo bajas
import shutil
zip_path = shutil.make_archive("/content/cluster6_artifacts", "zip", "/content/cluster6_artifacts")
from google.colab import files
files.download(zip_path)
```