Actividad de la Semana 05

Maestría en Inteligencia Artificial Aplicada
Curso: Proyecto Integrador
Institución: Instituto Tecnológico de Monterrey
Profesor titular: Dra. Grettel Barceló Alonso
Profesor titular: Dr. Luis Eduardo Falcón Morales
Profesora asistente Mtra. Verónica Sandra Guzmán de Valle

Avance #3

Actividad Baseline

Nombre del estudiante: María Figueroa Bejarano **Matrícula:** *A01114853*

Nombre del estudiante: David Hernández Castellanos
Matrícula: *A01795964*

Nombre del estudiante: Francisco Javier Ramírez Arias **Matrícula:** A01316379

Avance 3. Baseline (Modelo de referencia)

En este avance se construye un **modelo baseline** con el propósito de evaluar la viabilidad del problema de clasificación multiclase y establecer una referencia mínima contra la cual comparar modelos más avanzados.

Objetivos

- **3.1** Establecer medidas de calidad del modelo (métricas y criterios mínimos de desempeño).
- **3.2** Proporcionar un marco de referencia para evaluar y mejorar modelos posteriores.

Enfoque

Se emplean representaciones derivadas de imágenes construidas en el Avance 2 (embeddings + características construidas y preprocesadas). Se comparan:

1. Un baseline "al azar / mayoría" (DummyClassifier) para referencia mínima.
2. Un baseline simple e interpretable (Regresión Logística) para evaluar si existe señal predictiva.

Se analiza:

- Desempeño vs azar
- Sub/sobreajuste
- Métrica adecuada para el contexto multiclase
- Importancia/relevancia de características

Paqueterías

```
from pathlib import Path
import numpy as np
import pandas as pd
import json

from sklearn.dummy import DummyClassifier
from sklearn.metrics import classification_report, balanced_accuracy_score, f1_score
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression

import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import learning_curve
from sklearn.model_selection import cross_validate
```

```
BASE_PATH = Path("/content/drive/MyDrive/MNA/Proyecto Integrador")

X_train = np.load(BASE_PATH / "data" / "X_train.npy")
X_val   = np.load(BASE_PATH / "data" / "X_val.npy")
X_test  = np.load(BASE_PATH / "data" / "X_test.npy")

y_train = np.load(BASE_PATH / "data" / "y_train.npy")
y_val   = np.load(BASE_PATH / "data" / "y_val.npy")
y_test  = np.load(BASE_PATH / "data" / "y_test.npy")

# Clases (si ya lo guardaste)
classes_path = BASE_PATH / "metadata" / "classes.json"
if classes_path.exists():
    classes = json.loads(classes_path.read_text())
else:
    # fallback: si no existe
    classes = [str(i) for i in sorted(np.unique(y_train))]

X_train.shape, X_val.shape, X_test.shape, len(classes)
```

```
((8577, 1286), (2145, 1286), (2681, 1286), 8)
```

```
assert X_train.shape[0] == y_train.shape[0]
assert X_val.shape[0] == y_val.shape[0]
assert X_test.shape[0] == y_test.shape[0]

print("OK: shapes consistentes")
print("Clases únicas train:", len(np.unique(y_train)))
```

```
OK: shapes consistentes
Clases únicas train: 8
```

Métrica de evaluación

Dado que se trata de un problema **multiclase** y en el dataset original existe **desbalance** entre clases, se reportan métricas que no se ven dominadas por las clases más frecuentes:

- **F1-score macro**: promedia el F1 por clase (mismo peso a cada clase).
- **Balanced Accuracy**: promedio del recall por clase.

Estas métricas ofrecen una interpretación más adecuada del desempeño global cuando las clases no están perfectamente balanceadas y permiten identificar mejoras reales en clases minoritarias.

```
def eval_model(model, X_tr, y_tr, X_te, y_te, name="model"):
    model.fit(X_tr, y_tr)
    pred_tr = model.predict(X_tr)
    pred_te = model.predict(X_te)

    out = {
        "name": name,
        "train_macro_f1": f1_score(y_tr, pred_tr, average="macro"),
        "test_macro_f1":  f1_score(y_te, pred_te, average="macro"),
        "train_bal_acc":  balanced_accuracy_score(y_tr, pred_tr),
        "test_bal_acc":   balanced_accuracy_score(y_te, pred_te),
        "pred_te": pred_te,
        "pred_tr": pred_tr,
    }
    return out

dummy_mf = DummyClassifier(strategy="most_frequent", random_state=42)
res_dummy_mf = eval_model(dummy_mf, X_train, y_train, X_val, y_val, "Dummy-most_frequent")

dummy_strat = DummyClassifier(strategy="stratified", random_state=42)
res_dummy_strat = eval_model(dummy_strat, X_train, y_train, X_val, y_val, "Dummy-stratified")

res_dummy_mf, res_dummy_strat
```

```
({'name': 'Dummy-most_frequent',
  'train_macro_f1': 0.0324642386121538,
```

```
  'test_macro_f1': 0.032454361054766734,
  'train_bal_acc': np.float64(0.125),
  'test_bal_acc': np.float64(0.125),
  'pred_te': array([1, 1, 1, ..., 1, 1, 1]),
  'pred_tr': array([1, 1, 1, ..., 1, 1, 1])},
 {'name': 'Dummy-stratified',
  'train_macro_f1': 0.1171313387282749,
  'test_macro_f1': 0.13050795919183514,
  'train_bal_acc': np.float64(0.11718158842324872),
  'test_bal_acc': np.float64(0.1305316332428671),
  'pred_te': array([1, 5, 3, ..., 3, 0, 4]),
  'pred_tr': array([1, 5, 3, ..., 3, 3, 6])})
```

Interpretación esperada del baseline Dummy

- *most_frequent* representa una referencia mínima (predice siempre la clase más común).
- *stratified* simula una predicción "al azar" respetando la distribución de clases.

Si un modelo simple no supera de forma clara estos baselines, puede indicar que:

- el problema es difícil con la representación actual, o
- se requiere una ingeniería de características más rica/modelos más potentes.

Baseline 1: Regresión Logística

Selección del algoritmo baseline

Se utiliza **Regresión Logística multiclase** como baseline porque:

- Es un modelo **simple e interpretable**, adecuado para establecer una referencia mínima.
- Escala bien a **miles de observaciones** y **alta dimensionalidad** (embeddings).
- Su regularización ayuda a controlar sobreajuste y facilita comparar desempeño vs azar.
- Permite análisis de relevancia de características mediante coeficientes y permutation importance.

Además, se compara contra baselines Dummy (azar/mayoría) para validar que el desempeño no sea atribuible a la distribución de clases.

```
pipe_lr = Pipeline([
    ("scaler", StandardScaler()),
    ("clf", LogisticRegression(max_iter=4000, n_jobs=-1, multi_class="auto"))
])

res_lr = eval_model(pipe_lr, X_train, y_train, X_val, y_val, "LogReg+StandardScaler")
res_lr

pipe_lr_bal = Pipeline([
    ("scaler", StandardScaler()),
    ("clf", LogisticRegression(max_iter=4000, n_jobs=-1, class_weight="balanced"))
])

res_lr_bal = eval_model(pipe_lr_bal, X_train, y_train, X_val, y_val, "LogReg+Scaler (class_weight=balanced)")

pd.DataFrame([
    {"Modelo": res_lr["name"],      "Val Macro-F1": res_lr["test_macro_f1"],      "Val BalAcc": res_lr["test_bal_acc
    {"Modelo": res_lr_bal["name"],  "Val Macro-F1": res_lr_bal["test_macro_f1"],  "Val BalAcc": res_lr_bal["test_bal
])
```

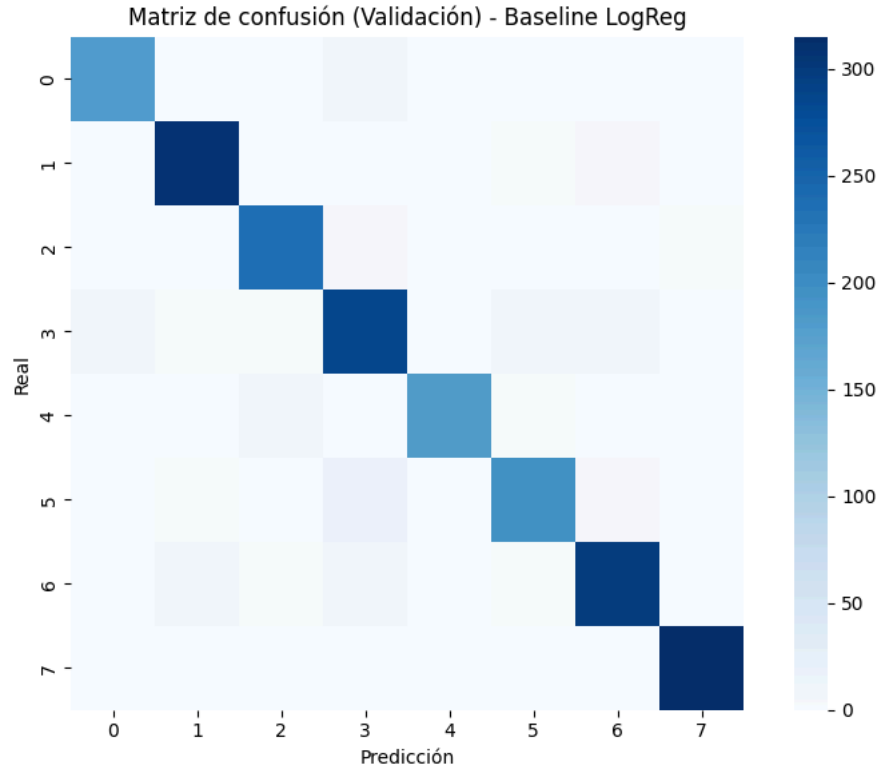
```
/usr/local/lib/python3.12/dist-packages/sklearn/linear_model/_logistic.py:1247: FutureWarning: 'multi_class' was depr
  warnings.warn(
```

	Modelo	Val Macro-F1	Val BalAcc
0	LogReg+StandardScaler	0.931160	0.930042
1	LogReg+Scaler (class_weight=balanced)	0.932757	0.932144

```
def plot_cm(y_true, y_pred, title):
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(8,6))
    sns.heatmap(cm, annot=False, cmap="Blues", square=True)
    plt.title(title)
    plt.xlabel("Predicción")
    plt.ylabel("Real")
    plt.tight_layout()
```

```
    plt.show()

print(classification_report(y_val, res_lr["pred_te"], target_names=classes, digits=4))
plot_cm(y_val, res_lr["pred_te"], "Matriz de confusión (Validación) - Baseline LogReg")
```

```
              precision    recall  f1-score   support

    basophil     0.9323    0.9179    0.9251       195
  eosinophil     0.9477    0.9625    0.9550       320
erythroblast     0.9325    0.9438    0.9381       249
          ig     0.8723    0.8969    0.8844       320
  lymphocyte     0.9581    0.9433    0.9506       194
    monocyte     0.8950    0.8634    0.8789       227
  neutrophil     0.9310    0.9281    0.9296       320
    platelet     0.9906    0.9844    0.9875       320

    accuracy                         0.9324      2145
   macro avg     0.9324    0.9300    0.9312      2145
weighted avg     0.9326    0.9324    0.9324      2145
```



Sub/sobreajuste

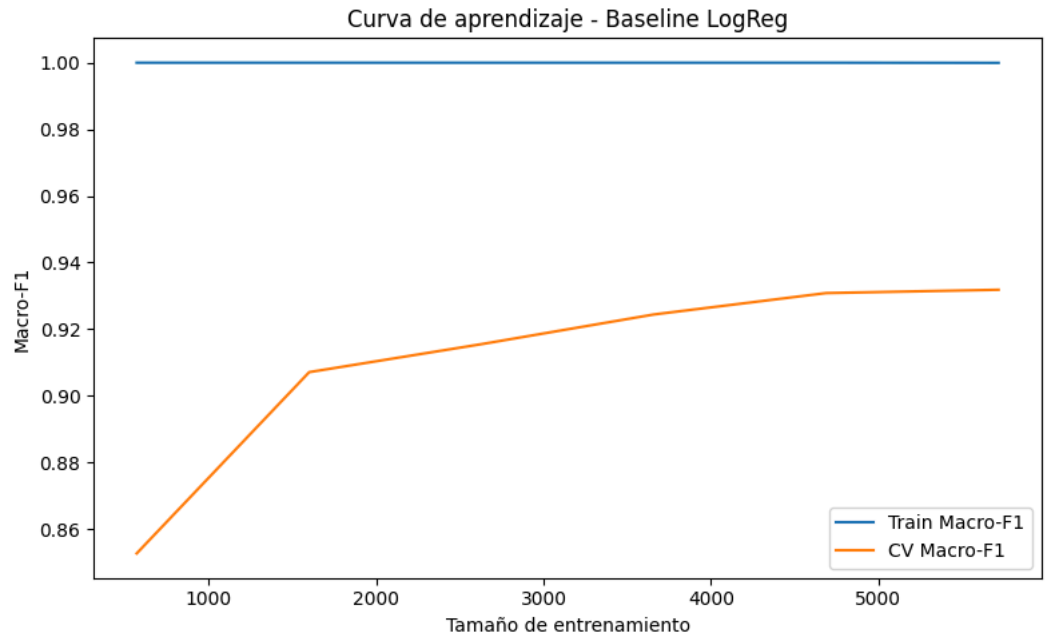
Se compara el desempeño en entrenamiento vs validación:

- Si el modelo rinde muy alto en train y mucho menor en validación → sobreajuste.
- Si rinde bajo en ambos → subajuste o representación insuficiente.

```
rows = []
for r in [res_dummy_mf, res_dummy_strat, res_lr, res_lr_bal]:
    rows.append({
        "Modelo": r["name"],
        "Train Macro-F1": r["train_macro_f1"],
        "Val Macro-F1": r["test_macro_f1"],
        "Train BalAcc": r["train_bal_acc"],
        "Val BalAcc": r["test_bal_acc"],
    })
pd.DataFrame(rows)
```

	Modelo	Train Macro-F1	Val Macro-F1	Train BalAcc	Val BalAcc
0	Dummy-most_frequent	0.032464	0.032454	0.125000	0.125000
1	Dummy-stratified	0.117131	0.130508	0.117182	0.130532
2	LogReg+StandardScaler	0.999902	0.931160	0.999902	0.930042
3	LogReg+Scaler (class_weight=balanced)	0.999902	0.932757	0.999902	0.932144

```
train_sizes, train_scores, val_scores = learning_curve(
    pipe_lr,
    X_train, y_train,
    cv=3,
    scoring="f1_macro",
    train_sizes=np.linspace(0.1, 1.0, 6),
    n_jobs=-1
)

train_mean = train_scores.mean(axis=1)
val_mean   = val_scores.mean(axis=1)

plt.figure(figsize=(8,5))
plt.plot(train_sizes, train_mean, label="Train Macro-F1")
plt.plot(train_sizes, val_mean, label="CV Macro-F1")
plt.title("Curva de aprendizaje - Baseline LogReg")
plt.xlabel("Tamaño de entrenamiento")
plt.ylabel("Macro-F1")
plt.legend()
plt.tight_layout()
plt.show()
```




```
cv = cross_validate(
    pipe_lr,
    X_train, y_train,
    cv=5,
    scoring={"macro_f1": "f1_macro", "bal_acc": "balanced_accuracy"},
    n_jobs=-1,
    return_train_score=True
)

print("CV Macro-F1  train:", cv["train_macro_f1"].mean().round(4), "+/-", cv["train_macro_f1"].std().round(4))
print("CV Macro-F1  valid:", cv["test_macro_f1"].mean().round(4), "+/-", cv["test_macro_f1"].std().round(4))
print("CV BalAcc    train:", cv["train_bal_acc"].mean().round(4), "+/-", cv["train_bal_acc"].std().round(4))
print("CV BalAcc    valid:", cv["test_bal_acc"].mean().round(4), "+/-", cv["test_bal_acc"].std().round(4))
```

```
CV Macro-F1  train: 0.9999 +/- 0.0001
CV Macro-F1  valid: 0.9322 +/- 0.0025
```

```
CV BalAcc    train: 0.9999 +/- 0.0001
CV BalAcc    valid: 0.9321 +/- 0.0026
```

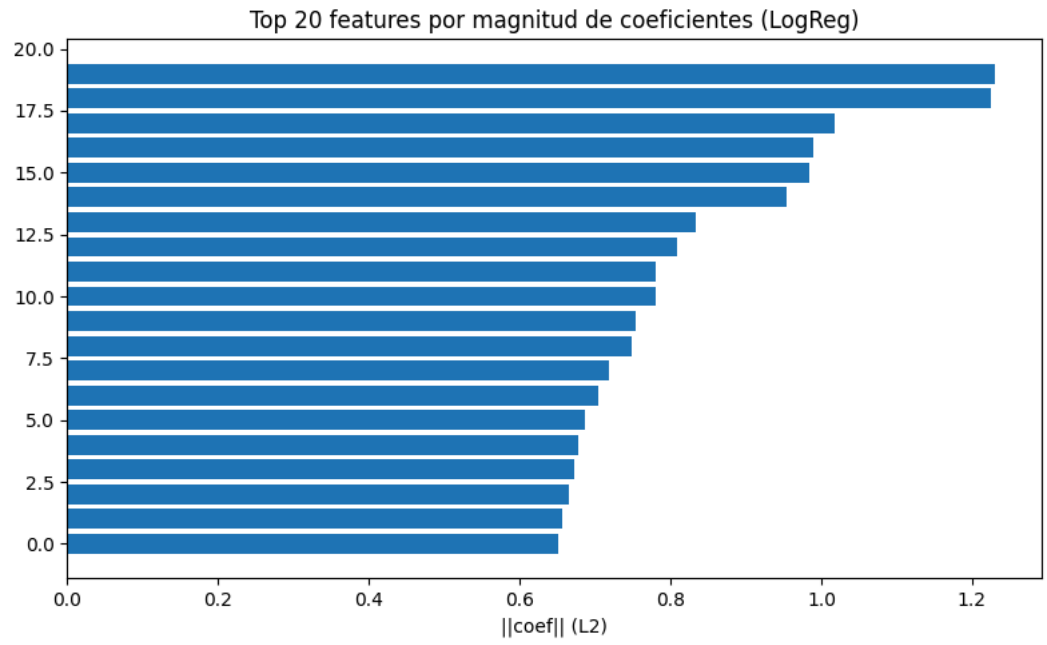
Diagnóstico sub/sobreajuste:

- Si el score de train es mucho mayor que validación (CV), hay sobreajuste.
- Si ambos son bajos, hay subajuste o la representación actual no captura suficiente señal.

Feature Importance

```
pipe_lr.fit(X_train, y_train)
clf = pipe_lr.named_steps["clf"]

# coef_ shape: (n_classes, n_features) → usamos norma L2 para ranking global
coef_norm = np.linalg.norm(clf.coef_, axis=0)

top_n = 20
top_idx = np.argsort(coef_norm)[-top_n:][::-1]

plt.figure(figsize=(8,5))
plt.barh(range(top_n), coef_norm[top_idx][::-1])
plt.title("Top 20 features por magnitud de coeficientes (LogReg)")
plt.xlabel("||coef|| (L2)")
plt.tight_layout()
plt.show()

top_idx[:5], coef_norm[top_idx[:5]]
```

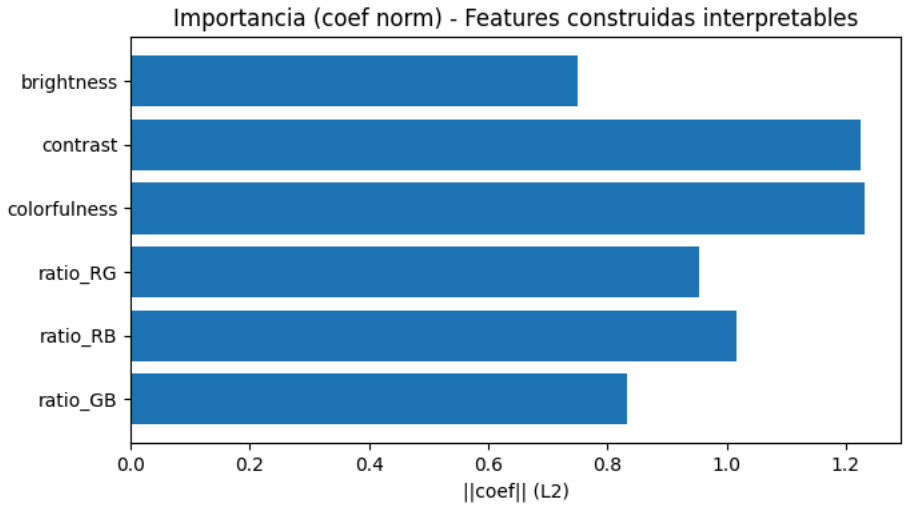
```
/usr/local/lib/python3.12/dist-packages/sklearn/linear_model/_logistic.py:1247: FutureWarning: 'multi_class' was depr
  warnings.warn(
```




```
(array([1282, 1281, 1284,  463,  899]),
 array([1.23076583, 1.22561368, 1.01759404, 0.98894434, 0.98434314]))
```

```
# Interpretabilidad: últimas 6 features (scalars) si X = [embeddings | scalars]
scalar_names = ["brightness", "contrast", "colorfulness", "ratio_RG", "ratio_RB", "ratio_GB"]

n_scalars = 6
scalar_coef_norm = coef_norm[-n_scalars:]

plt.figure(figsize=(7,4))
plt.barh(range(n_scalars), scalar_coef_norm[::-1])
plt.yticks(range(n_scalars), scalar_names[::-1])
plt.title("Importancia (coef norm) - Features construidas interpretables")
plt.xlabel("||coef|| (L2)")
plt.tight_layout()
```

```
plt.show()
```

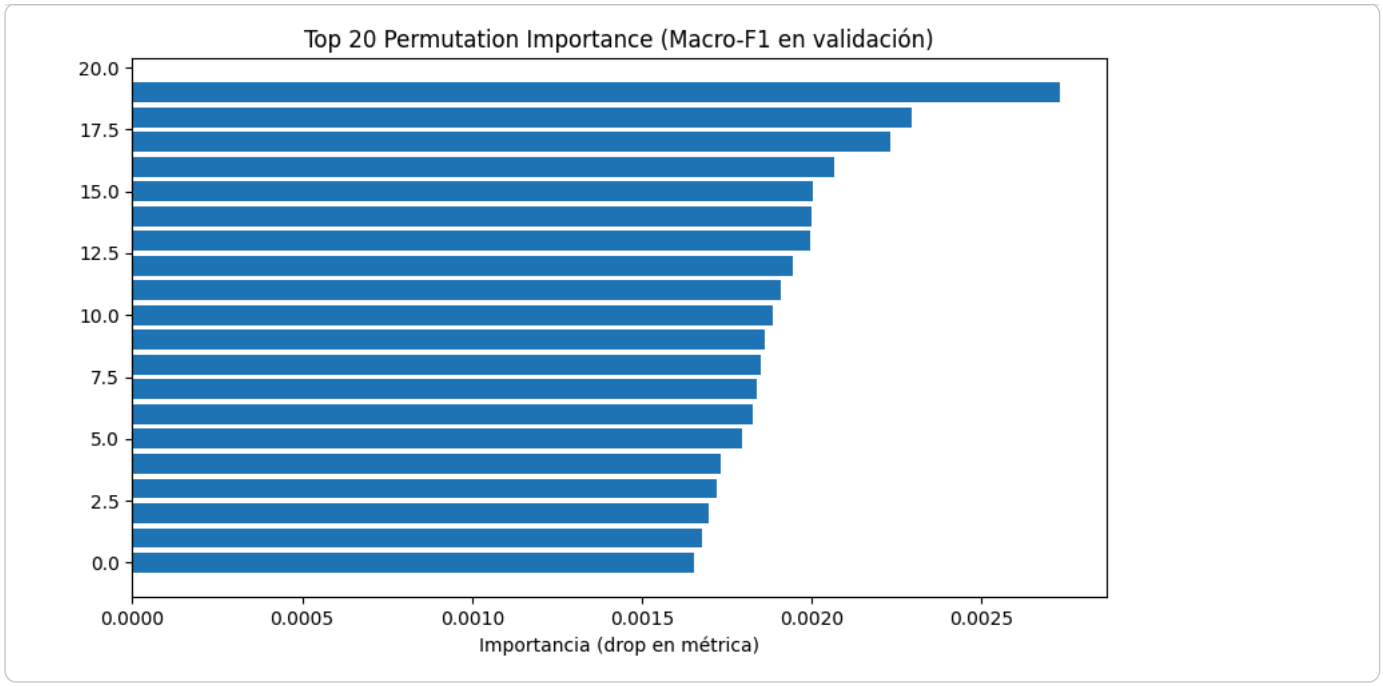


Interpretación

Aunque el embedding contiene cientos/miles de dimensiones no directamente interpretables, las **features construidas** (brightness/contrast/colorfulness/ratios) sí permiten interpretación de dominio. Su importancia relativa ayuda a detectar si el modelo está capturando variaciones de adquisición (iluminación/tinción) o patrones más estructurales representados en los embeddings.

Permutation importance

```
from sklearn.inspection import permutation_importance

perm = permutation_importance(
    pipe_lr,
    X_val, y_val,
    scoring="f1_macro",
    n_repeats=5,
    random_state=42,
    n_jobs=-1
)

imp = perm.importances_mean
top_idx = np.argsort(imp)[-20:][::-1]

plt.figure(figsize=(8,5))
plt.barh(range(20), imp[top_idx][::-1])
plt.title("Top 20 Permutation Importance (Macro-F1 en validación)")
plt.xlabel("Importancia (drop en métrica)")
plt.tight_layout()
plt.show()
```



Interpretación de relevancia de características

Se utilizaron dos enfoques complementarios:

- **Embedded**: magnitud de coeficientes en Regresión Logística (señal global).
- **Permutation importance**: impacto directo en Macro-F1 al permutar cada característica.

Esto permite identificar si el modelo depende de un subconjunto reducido de características y detectar posibles variables poco informativas.

Desempeño mínimo esperado

Un criterio práctico de desempeño mínimo es superar claramente los baselines Dummy (azar/mayoría) en métricas robustas como Macro-F1 y Balanced Accuracy.

- Si el baseline de Regresión Logística supera al dummy estratificado por un margen significativo, se concluye que **existe señal predictiva** en las representaciones utilizadas.
- Si el desempeño es cercano al azar, se consideraría enriquecer la representación (fine-tuning, data augmentation, modelos CNN end-to-end) o revisar calidad/etiquetas.

Desempeño mínimo

El desempeño mínimo debe superar claramente:

- **Dummy-stratified** (azar respetando distribución)
- **Dummy-most_frequent** (mayoría)

En este problema multiclase, se prioriza **Macro-F1** y **Balanced Accuracy** para evitar que el resultado esté dominado por clases más frecuentes.

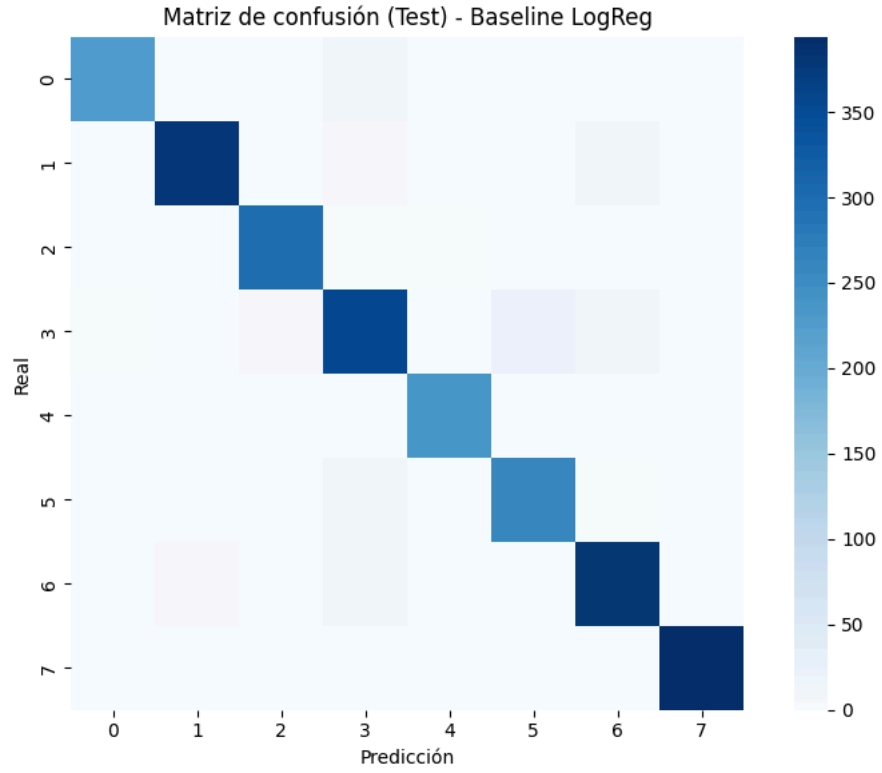
```
n_classes = len(np.unique(y_train))
chance_acc = 1 / n_classes
print("Número de clases:", n_classes)
print("Accuracy esperada al azar (aprox):", chance_acc)
```

```
Número de clases: 8
Accuracy esperada al azar (aprox): 0.125
```

```
# Entrenar en train+val y evaluar en test (baseline final)
X_trainval = np.vstack([X_train, X_val])
y_trainval = np.concatenate([y_train, y_val])
```

```
pipe_lr.fit(X_trainval, y_trainval)
pred_test = pipe_lr.predict(X_test)

print("Macro-F1 test:", f1_score(y_test, pred_test, average="macro"))
print("BalAcc test:", balanced_accuracy_score(y_test, pred_test))
print(classification_report(y_test, pred_test, target_names=classes, digits=4))
plot_cm(y_test, pred_test, "Matriz de confusión (Test) - Baseline LogReg")
```

```
/usr/local/lib/python3.12/dist-packages/sklearn/linear_model/_logistic.py:1247: FutureWarning: 'multi_class' was depr
  warnings.warn(
Macro-F1 test: 0.9428646152467914
BalAcc test: 0.9434865961450523
              precision    recall  f1-score   support

    basophil     0.9372    0.9180    0.9275       244
  eosinophil     0.9643    0.9450    0.9545       400
erythroblast     0.9645    0.9645    0.9645       310
          ig     0.8850    0.8850    0.8850       400
  lymphocyte     0.9480    0.9753    0.9615       243
    monocyte     0.8973    0.9225    0.9097       284
  neutrophil     0.9454    0.9525    0.9489       400
    platelet     0.9975    0.9850    0.9912       400

    accuracy                         0.9433      2681
   macro avg     0.9424    0.9435    0.9429      2681
weighted avg     0.9436    0.9433    0.9434      2681
```



PCA Baseline

```
pca_train_path = BASE_PATH / "data" / "X_train_pca.npy"
pca_val_path   = BASE_PATH / "data" / "X_val_pca.npy"

if pca_train_path.exists() and pca_val_path.exists():
    X_train_pca = np.load(pca_train_path)
    X_val_pca   = np.load(pca_val_path)

    res_lr_pca = eval_model(pipe_lr, X_train_pca, y_train, X_val_pca, y_val, "LogReg on PCA-features")
    print(res_lr_pca)

    # Comparación rápida
    comp = pd.DataFrame([
        {"Modelo":"LogReg (Full)", "Val Macro-F1": res_lr["test_macro_f1"], "Val BalAcc": res_lr["test_bal_acc"]},
        {"Modelo":"LogReg (PCA)",  "Val Macro-F1": res_lr_pca["test_macro_f1"], "Val BalAcc": res_lr_pca["test_bal_a
    ])
```

```
comp
```

```
/usr/local/lib/python3.12/dist-packages/sklearn/linear_model/_logistic.py:1247: FutureWarning: 'multi_class' was depr
  warnings.warn(
{'name': 'LogReg on PCA-features', 'train_macro_f1': 0.8791406086703494, 'test_macro_f1': 0.8684449738998355, 'train_
```

Conclusiones (Baseline en CRISP-ML(Q))

El baseline construido establece una referencia objetiva del desempeño alcanzable con un modelo simple e interpretable, utilizando las representaciones generadas en la fase de Preparación de Datos (Avance 2).