

Proyecto final: el predictor de hits de PyRitmo

Durante el curso preparaste el problema, entrenaste regresiones y clasificadores, comparaste árboles con ensambles, limpiaste datos sucios, validaste con pliegues y ajustaste hiperparámetros. Ahora juntas todo en un solo modelo, desde el CSV sucio hasta una evaluación honesta. Se usan Python 3.12 o superior, scikit-learn 1.8 o superior, pandas y xgboost 2.1 o superior (`pip install xgboost`). El notebook `proyecto-predictor-de-hits` contiene una solución de referencia: consúltalo al terminar, no antes.

Contexto

PyRitmo, una app de streaming de música, recibe cada semana decenas de canciones nuevas de los sellos. El equipo editorial puede promover solo algunas. Quiere un modelo que, **antes del lanzamiento**, estime la probabilidad de que una canción entre al Top 50 semanal en su primer mes. Dejar pasar un hit cuesta más que promover una canción que no despegue, pero si el modelo marca casi todo como hit, el equipo deja de confiar en él.

Datos

Copia las funciones `generar_canciones` y `ensuciar` y las listas `NUMERICAS` y `CATEGORICAS` de la primera celda de cualquier notebook del curso. Trabaja **solo con la versión sucia**:

```
df, _ = generar_canciones(n=1500, semilla=42)
sucio = ensuciar(df, semilla=7)
print(sucio.shape)                # (1525, 15)
print(sucio.duplicated().sum())    # 25
print(sucio["genero"].nunique())    # 17
```

Si no obtienes esas tres cifras, revisa las semillas. El CSV trae filas duplicadas, géneros mal escritos, algunas duraciones en milisegundos y valores faltantes en `tempo_bpm` y `seguidores_miles`. El objetivo es `top50` (1 si la canción entró al Top 50). El segundo valor que devuelve `generar_canciones` son los estilos latentes: no los uses como entrada.

Requisitos mínimos

1. **Limpieza.** Escribe una función `limpiar(sucio)` que quite duplicados, unifique los géneros en 5 categorías y pase a segundos las duraciones en milisegundos. Comprueba el resultado: 1500 filas, 412 hits (27.5 %), duración máxima de 446

segundos, y aún 85 faltantes en `tempo_bpm` y 59 en `seguidores_miles`. Estos faltantes no se borran.

2. **X e y sin fuga.** Usa `X = limpio[NUMERICAS + CATEGORICAS]` e `y = limpio["top50"]`. Explica en una frase por qué `reproducciones_mes1` y `popularidad` no pueden ser columnas de entrada. Parte con `train_test_split(X, y, test_size=0.25, stratify=y, random_state=42)`: 1125 canciones en train y 375 en test. El test no se toca hasta el requisito 7.
3. **Pipeline con imputación.** Un `ColumnTransformer` con `SimpleImputer(strategy="median")` y `StandardScaler` para las numéricas, y `SimpleImputer(strategy="most_frequent")` y `OneHotEncoder(handle_unknown="ignore")` para el género, todo dentro de un `Pipeline` con el modelo.
4. **Línea base y comparación.** Un `DummyClassifier` y, como mínimo, un modelo lineal (regresión logística), uno por distancia o SVM (KNN o `svc`), un `Random Forest` y un `XGBClassifier`, todos con el mismo preprocesamiento y comparados con `cross_val_score, scoring="roc_auc"` y `StratifiedKFold(5, shuffle=True, random_state=42)` sobre train. Reporta media y desviación en una sola tabla.
5. **XGBoost: early stopping y búsqueda de hiperparámetros.** Aparta una partición de validación **dentro de train** (no uses el test) y entrena XGBoost con `early_stopping_rounds=50` y `eval_set` para conocer el orden de magnitud de `n_estimators`. Después ajusta XGBoost con `RandomizedSearchCV (scoring="roc_auc")`, buscando `n_estimators` y `learning_rate` juntos. Justifica los rangos y reporta el número de ajustes, `best_params_` y `best_score_`.
6. **Umbral de decisión o `scale_pos_weight`.** Con predicciones de validación cruzada sobre train, elige el umbral más alto que detecte al menos el 70 % de los hits y muestra `precision` y `recall` en al menos tres umbrales. Compara con la otra palanca, `scale_pos_weight = negativos / positivos` con umbral 0.5, y justifica cuál usas según el costo de dejar pasar un hit. No uses las dos a la vez sin medir.
7. **Evaluación final, una vez.** En test: AUC, matriz de confusión, `precision` y `recall` con tu umbral y con 0.5, y la comparación con el Dummy.
8. **Una canción nueva.** Predice la probabilidad de Top 50 de esta canción: energía 0.78, `bailabilidad` 0.84, `acústica` 0.10, 96 bpm, 185 segundos, 850 mil seguidores, con colaboración, no explícita, en playlist editorial, reguetón.

Pistas de implementación

En `xgboost 2.1` o superior, `early_stopping_rounds` y `eval_metric` van en el constructor. El `eval_set` debe pasar por el mismo preprocesamiento, así que ajusta el pre con la parte de ajuste y transforma ambas:

```
X_fit, X_val, y_fit, y_val = train_test_split(
    X_tr, y_tr, test_size=0.2, stratify=y_tr, random_state=42)
pre_es = crear_pre().fit(X_fit)
```

```
xgb_es = XGBClassifier(
    n_estimators=2000, learning_rate=0.05, max_depth=4,
    eval_metric="logloss", early_stopping_rounds=50,
    random_state=42)
xgb_es.fit(pre_es.transform(X_fit), y_fit,
           eval_set=[(pre_es.transform(X_val), y_val)],
           verbose=False)
print(xgb_es.best_iteration)
```

Dentro de RandomizedSearchCV no uses early stopping: con un eval_set fijo, los pliegues dejan de ser independientes. Usa loguniform para learning_rate y reg_lambda, randint para n_estimators y max_depth y uniform para subsample y colsample_bytree.

Para el umbral, cross_val_predict devuelve, para cada canción de train, la probabilidad que le dio un modelo que no la vio al entrenar:

```
mejor = busqueda.best_estimator_
proba = cross_val_predict(mejor, X_tr, y_tr, cv=cv,
                          method="predict_proba")[:, 1]
for u in [0.25, 0.30, 0.35, 0.50]:
    pred = (proba >= u).astype(int)
    p = precision_score(y_tr, pred)
    r = recall_score(y_tr, pred)
    print(u, round(p, 3), round(r, 3))
```

Al final, aplica el mismo umbral a busqueda.predict_proba(X_te)[:, 1]. No elijas el umbral mirando el test.

Lo que deberías obtener aproximadamente

Estas son las cifras de la solución de referencia. Las tuyas pueden variar si eliges otros modelos o rangos, pero deberían quedar cerca:

Paso	Referencia
AUC de validación cruzada (train)	Dummy 0.5, logística 0.828, SVC 0.825, Random Forest 0.851, gradient boosting 0.848, XGBoost sin ajustar 0.841
Early stopping (validación de 225 canciones, learning_rate 0.05)	best_iteration 97, log-loss de validación 0.4336

Búsqueda aleatoria de XGBoost (30 combinaciones, 150 ajustes)	learning_rate 0.0336, 332 árboles, max_depth 2, subsample 0.927, colsample_bytree 0.649, reg_lambda 0.715; best_score_ 0.8594
Umbral en validación cruzada	0.25: precision 0.528, recall 0.793; 0.30: 0.563 y 0.748; 0.35: 0.614 y 0.699; 0.50: 0.722 y 0.537. Umbral elegido: 0.30
scale_pos_weight 2.64 con umbral 0.5	precision 0.556, recall 0.751: un punto equivalente al umbral 0.30
Test con umbral 0.5	AUC 0.871, exactitud 0.827, precision 0.721, recall 0.602, matriz [[248, 24], [41, 62]]
Test con umbral 0.30	exactitud 0.776, precision 0.568, recall 0.767, matriz [[212, 60], [24, 79]]
Dummy en test	exactitud 0.725, sin detectar ningún hit
Canción nueva	probabilidad de Top 50: 0.951

Fíjate en el intercambio: con umbral 0.30 el modelo encuentra 79 de los 103 hits del test en lugar de 62, a cambio de 60 falsas alarmas en lugar de 24.

Extras opcionales

- **Regresión de popularidad.** Con los mismos datos limpios, predice popularidad con una regresión lineal usando $\log_{10}(\text{seguidores_miles})$ en lugar de la columna cruda (partición sin stratify). La referencia obtiene MAE 4.69 y R^2 0.716 en test, contra MAE 8.68 del DummyRegressor.
- **Estilo como característica.** Ajusta KMeans con k=4 sobre los rasgos de audio escalados, dentro del pipeline, y agrega el cluster como columna. Compara el AUC de validación cruzada con y sin ella.
- **Importancia por permutación.** Calcula permutation_importance sobre test con scoring="roc_auc" y comenta qué columnas pesan más.

Entregables

1. proyecto.ipynb, ejecutado de principio a fin sin errores, con la función limpiar y una celda de texto por requisito.
2. INFORME.md (una página): la tabla de comparación, el resultado del early stopping, los hiperparámetros elegidos, el umbral (o scale_pos_weight) y su justificación, las cifras de test, la predicción de la canción nueva y dos limitaciones del modelo. Los datos son sintéticos: describe lo que ves sin inventar causas.

Rúbrica (100 puntos, más 10 de extras)

- **Limpieza (15 puntos):** duplicados, géneros y unidades corregidos, con las cifras de control.
- **Datos y partición (10 puntos):** sin columnas de fuga, partición estratificada con la semilla indicada.
- **Pipeline con imputación (15 puntos):** imputación y codificación dentro del pipeline, sin fuga.
- **Comparación de modelos (15 puntos):** Dummy, modelo lineal, distancia o SVM, Random Forest y XGBoost con validación cruzada en una sola tabla.
- **XGBoost (20 puntos):** early stopping con validación separada de train (nunca el test), búsqueda aleatoria con rangos razonables, nombres `paso__parametro` y costo reportado.
- **Umbral y evaluación (15 puntos):** umbral o `scale_pos_weight` elegido sin mirar el test y justificado por el costo de perder un hit, test usado una sola vez, métricas correctas.
- **Informe (10 puntos):** conclusiones con cifras, la justificación del umbral y limitaciones honestas.
- **Extras (hasta 10 puntos):** cualquiera de los tres, con su comparación.