

# Predicción de popularidad de canciones

Daniel Montero, Juan Camilo Mejía, Simón Jaramillo

## 1. Preprocesamiento de datos

```
df_dataTraining = dataTraining.drop(columns=['track_id'])

# Se agrega una columna que cuenta el número de artistas para tratar de modelar las colaboraciones
df_dataTraining['num_artists'] = df_dataTraining['artists'].str.count(';') + 1

# Separación de datos en train y test
X = df_dataTraining.drop(columns='popularity')
y = df_dataTraining['popularity']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)

# Se utilizar CatBoostEncoder ya que no aumenta la dimensionalidad y respeta la relación variable objetivo
# Asigna un único valor numérico por categoría basado en la media del target,
# en lugar de crear columnas 0/1 como el one-hot.
# Ofrece una escala continua donde categorías de efecto similar quedan cercanas, ayudando a modelos lineales
# y de árbol a aprender más rápido y con mejor generalización.
from category_encoders import CatBoostEncoder

encoder_cb = CatBoostEncoder(cols=['track_genre', 'track_name', 'album_name', 'artists'])
encoder_cb.fit(X_train, y_train)

# Transformamos los datos
X_train = encoder_cb.transform(X_train)
X_test = encoder_cb.transform(X_test)

# Transformación 1
# Reducir el sesgo a la derecha y estabilizar la varianza.
for col in ['duration_ms', 'instrumentalness', 'num_artists']:
    X_train[col] = np.log1p(X_train[col])
    X_test[col] = np.log1p(X_test[col])

# Transformación 1
# Reducir el sesgo a la derecha y estabilizar la varianza.
for col in ['duration_ms', 'instrumentalness', 'num_artists']:
    X_train[col] = np.log1p(X_train[col])
    X_test[col] = np.log1p(X_test[col])

# Transformación 3
# Reducir la escala de rangos muy altos.
for col in ['tempo', 'album_name', 'track_name']:
    X_train[col] = np.square(X_train[col])
    X_test[col] = np.square(X_test[col])

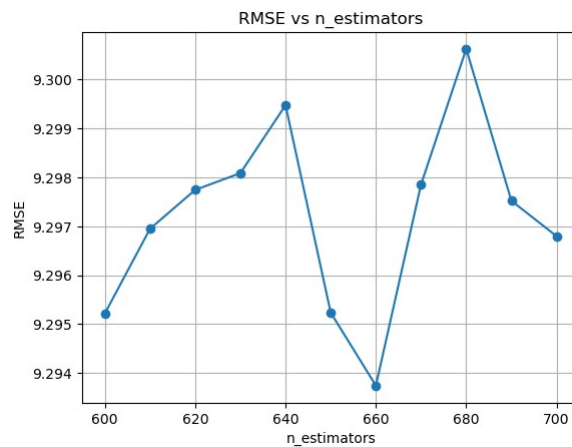
# Escalamiento de datos
# Normalizar los datos para que tengan media 0 y varianza 1.
scaler = StandardScaler()
X_train = pd.DataFrame(scaler.fit_transform(X_train), columns=X_train.columns, index=X_train.index)
X_test = pd.DataFrame(scaler.transform(X_test), columns=X_test.columns, index=X_test.index)
```

## 2. Calibración del modelo

```
# Calibración n_estimators
n_estimators_range = range(600, 701, 10)
rmse_scores = []

for n in n_estimators_range:
    model = XGBRegressor(n_estimators=n, random_state=42)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    rmse = mean_squared_error(y_test, y_pred) ** 0.5
    rmse_scores.append(rmse)
best_n_estimators = n_estimators_range[np.argmin(rmse_scores)]
plt.plot(n_estimators_range, rmse_scores, marker='o')
plt.xlabel("n_estimators")
plt.ylabel("RMSE")
plt.title("RMSE vs n_estimators")
plt.grid(True)
plt.show()
```

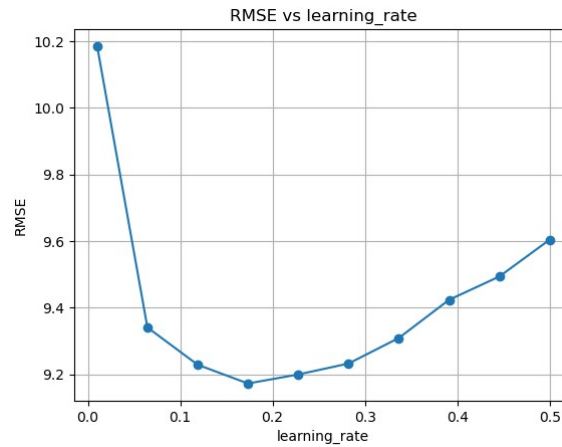
Python



```
# Calibración learning_rate
learning_rates = np.linspace(0.01, 0.5, 10)
rmse_scores = []

for lr in learning_rates:
    model = XGBRegressor(n_estimators=best_n_estimators, learning_rate=lr, random_state=42)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    rmse = mean_squared_error(y_test, y_pred) ** 0.5
    rmse_scores.append(rmse)
best_lr = learning_rates[np.argmin(rmse_scores)]
plt.plot(learning_rates, rmse_scores, marker='o')
plt.xlabel("learning_rate")
plt.ylabel("RMSE")
plt.title("RMSE vs learning_rate")
plt.grid(True)
plt.show()
```

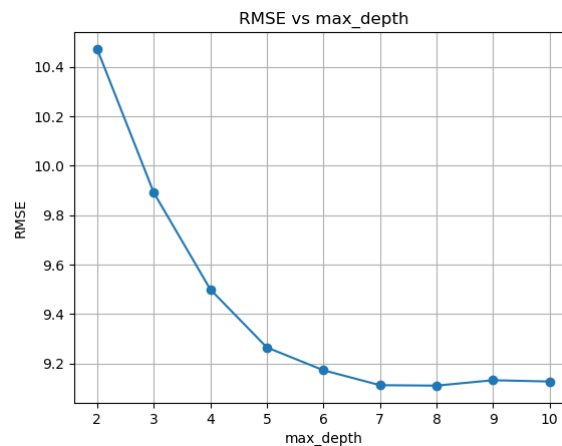
Python



```
# Calibración max_depth
depths = range(2, 11)
rmse_scores = []

for d in depths:
    model = XGBRegressor(n_estimators=best_n_estimators, learning_rate=best_lr, max_depth=d, random_state=42)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    rmse = mean_squared_error(y_test, y_pred) ** 0.5
    rmse_scores.append(rmse)
best_max_depth = depths[np.argmin(rmse_scores)]
plt.plot(depths, rmse_scores, marker='o')
plt.xlabel("max_depth")
plt.ylabel("RMSE")
plt.title("RMSE vs max_depth")
plt.grid(True)
plt.show()
```

Python



Optuna no evalúa exhaustivamente todas las combinaciones, sino que dirige las pruebas hacia las regiones con mejor desempeño. Es ideal cuando el numero de combinaciones posibles es muy grande.

```

# Optuna objective function
import optuna
def objective(trial):
    params = {
        'n_estimators': trial.suggest_int('n_estimators', 2900, 2900),
        'learning_rate': trial.suggest_float('learning_rate', 0.01, 0.2, log=True),
        'max_depth': trial.suggest_int('max_depth', 12, 12),
        'tree_method': 'gpu_hist', 'predictor': 'gpu_predictor',
        'random_state': 42
    }

    model = XGBRegressor(**params)
    model.fit(X_train, y_train)
    preds = model.predict(X_test)
    rmse = mean_squared_error(y_test, preds)**0.5
    return rmse

# Ejecutar la optimización
study = optuna.create_study(direction='minimize')
study.optimize(objective, n_trials=15)

# Mejor modelo
best_params = study.best_params
best_model = XGBRegressor(**best_params)
best_model.fit(X_train, y_train)
y_pred = best_model.predict(X_test)
rmse = mean_squared_error(y_test, y_pred)**0.5

print(f"Mejor RMSE con Optuna: {rmse:.4f}")
print("Mejores hiperparámetros:", best_params)

```

Python

Mejor RMSE con Optuna: 8.9765

Mejores hiperparámetros: {'n\_estimators': 2900, 'learning\_rate': 0.019133953427042664, 'max\_depth': 12}

Al optimizar los hiperparámetros por separado, se obtienen óptimos locales para cada uno de ellos. Sin embargo, algunos hiperparámetros tendrán mejor desempeño en conjunto con otros. Por ejemplo, *learning rate* bajo necesitará más estimadores para converger.

### Resumen de hiperparámetros del modelo seleccionado:

Parámetro	Valor óptimo	Cómo afecta al modelo
<b>n_estimators</b>	2900	Un gran número de árboles permite que el modelo agregue poco a poco correcciones, reduciendo el <b>sesgo</b> . Con un <i>learning_rate</i> pequeño ( $\approx 0.019$ ), hace falta más estimadores para converger, pero se logra un ajuste más fino y estable.
<b>learning_rate</b>	0.0191	El aprendizaje es lento, cada árbol aporta una pequeña corrección, lo que evita saltos gaudiosos en el aprendizaje. Esta tasa de aprendizaje requiere más <i>n_estimators</i> para que converja el modelo.
<b>max_depth</b>	12	Profundidad alta que permite capturar variaciones pequeñas y complejas entre variables, pero sin caer en profundidades extremas

	que generen sobreajuste descontrolado.
--	--

- El tipo de error para calibrar los modelos fue el RMSE. Esto debido a que la distribución de la variable a predecir no cuenta con outliers y sus muestras, según el análisis descriptivo, refleja que está centrada a valores medios bajos (dada su media y percentiles) por lo cual con el RMSE lo que se buscó fue tener una medición del error más equilibrada y en la escala de los datos.
- Bagging:
  - El número calibrado de `n_estimators` es 640, a partir de este valor, el error se estabiliza
  - El número calibrado de `max_samples` es 1 con el menor error
  - El número calibrado de `max_features` es 0.6
- Random Forest:
  - El número calibrado de `n_estimators` es 500
  - El número calibrado de `max_depth` es 25
  - El número calibrado de `max_features` es 1.0
- XGBoost:
  - El número calibrado de `n_estimators` es 2900
  - El número calibrado de `learning_rate` es 0.019133953427042664
  - El número calibrado de `max_depth` es 12

### 3. Entrenamiento del modelo

```
bg = BaggingRegressor(n_estimators=640, max_samples= 1.0, max_features= 0.6, random_state=42)
bg.fit(X_train, y_train)
y_pred_bg = bg.predict(X_test)
rmse_bg = mean_squared_error(y_test, y_pred_bg) ** 0.5
```

```
rf = RandomForestRegressor(n_estimators=500, max_depth=25, max_features=1.0, random_state=42)
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)
rmse_rf = mean_squared_error(y_test, y_pred_rf) ** 0.5
```

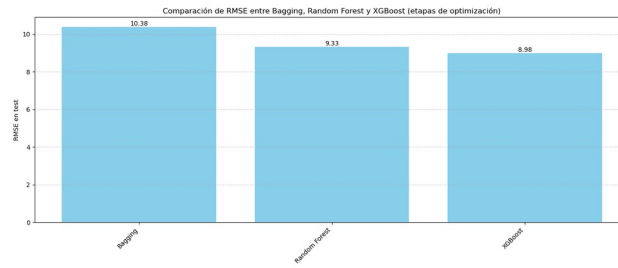
```
xb = XGBRegressor(n_estimators=2900, learning_rate=0.019133953427042664,
                  max_depth=12, tree_method='gpu_hist', predictor='gpu_predictor', random_state=42)
xb.fit(X_train, y_train)
y_pred_xb = xb.predict(X_test)
rmse_xb = mean_squared_error(y_test, y_pred_xb) ** 0.5
```

```
labels = ['Bagging', 'Random Forest', 'XGBoost']
rmse_values = [rmse_bg, rmse_rf, rmse_xb]

plt.figure(figsize=(14, 6))
bars = plt.bar(labels, rmse_values, color='skyblue')
plt.xticks(rotation=45, ha='right')
plt.ylabel('RMSE en test')
plt.title('Comparación de RMSE entre Bagging, Random Forest y XGBoost (etapas de optimización)')
plt.grid(axis='y', linestyle='--', alpha=0.6)

for bar in bars:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, yval, f'{yval:.2f}', ha='center', va='bottom')

plt.tight_layout()
plt.show()
```



El modelo seleccionado es el **XGBoost** ya que es el que mejor desempeño muestra en el RMSE evaluándolo en los datos  $X_{test}$

## 4. Disponibilización del modelo

Se disponibiliza el modelo usando AWS.

```
Running model on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5000
* Running on http://172.31.2.119:5000
Press CTRL+C to quit
```

Predicciones de 2 registros del *data set* test:

	artists	duration_ms	explicit	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	tempo	time_signature	track_genre
79644	Xanthochroid	201986	False	0.174	0.324	0	-13.455	1	0.0396	0.632000	0.847000	0.5580	0.0493	119.282	3	black-metal

→

No es seguro3.145.105.97:5000

🔍

🔗

Popularidad de la canción API1.0

[ Back ] [ / ]

Swagger JSON

Popularidad de la canción API

predictPopularity Prediction

GET /predict/

Parameters

Cancel

Name	Description
danceability * required	Danceability
number (query)	0.174
energy * required	Energy
number (query)	0.324
valence * required	Valence
number (query)	0.0493
duration_ms * required	Duration (ms)
integer (query)	201986
track_genre * required	Track Genre
string (query)	black-metal
artists * required	Artist Name
string (query)	Xanthochroid
explicit * required	Explicit (0 or 1)
integer (query)	0
key * required	Musical Key
integer (query)	0
loudness * required	Loudness
number (query)	-13.456
mode * required	Mode (0 or 1)
integer (query)	1
speechiness * required	Speechiness
number (query)	0.0396
acousticness * required	Acousticness
number (query)	0.632000
instrumentalness * required	Instrumentalness
number (query)	0.847000
liveness * required	Liveness
number (query)	0.5580
tempo * required	Tempo
number (query)	119.282
time_signature * required	Time Signature
integer (query)	3
X-Fields	An optional fields mask
string (header)	X-Fields

ExecuteClear

Responses

Response content typeapplication/json

Curl

curl -X 'GET' -s "http://3.145.105.97:5000/predict/?danceability=0.174&energy=0.324&valence=0.0493&duration\_ms=201986&track\_genre=black-metal&artists=Xanthochroid&explicit=0&key=0&loudness=-13.456&mode=1&speechiness=0.0396&acousticness=0.632000&instrumentalness=0.847000&liveness=0.5580&tempo=119.282&time\_signature=3" -H 'accept: application/json'

Request URL

http://3.145.105.97:5000/predict/?danceability=0.174&energy=0.324&valence=0.0493&duration\_ms=201986&track\_genre=black-metal&artists=Xanthochroid&explicit=0&key=0&loudness=-13.456&mode=1&speechiness=0.0396&acousticness=0.632000&instrumentalness=0.847000&liveness=0.5580&tempo=119.282&time\_signature=3

Server response

CodeDetails

200

response body

```
{  "result": 33}
```

Copy

	artists	duration_ms	explicit	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	tempo	time_signature	track_genre
54259	The Slabbacks	196747	False	0.591	0.908	7	-3.977	1	0.0804	0.243000	0.000046	0.0918	0.9200	119.581	4	rockabilly

Popularidad de la canción API

No es seguro 3.145.105.97:5000

Name	Description
danceability * required number (query)	Danceability 0.591
energy * required number (query)	Energy 0.908
valence * required number (query)	Valence 0.9200
duration_ms * required integer (query)	Duration (ms) 196747
track_genre * required string (query)	Track Genre rockabilly
artists * required string (query)	Artist Name The Slapbacks
explicit * required integer (query)	Explicit (0 or 1) 0
key * required integer (query)	Musical Key 7
loudness * required number (query)	Loudness -3.977
mode * required integer (query)	Mode (0 or 1) 1
speechiness * required number (query)	Speechiness 0.0804
acousticness * required number (query)	Acousticness 0.243000
instrumentalness * required number (query)	Instrumentalness 0.000046
liveness * required number (query)	Liveness 0.0918
tempo * required number (query)	Tempo 119.581
time_signature * required integer (query)	Time Signature 4
X-Fields string(\$mask) (header)	An optional fields mask X-Fields

Execute Clear

Responses Response content type application/json

Curl

```
curl -X 'GET' \
  'http://3.145.105.97:5000/predict/?danceability=0.591&energy=0.908&valence=0.9200&duration_ms=196747&track_genre=rockabilly&artists=The%20Slapbacks&explicit=0&key=7&loudness=-3.977&mode=1&speechiness=0.0804&acousticness=0.243000&instrumentalness=0.000046&liveness=0.0918&tempo=119.581&time_signature=4' \
  -H 'accept: application/json'
```

Request URL

```
http://3.145.105.97:5000/predict/?danceability=0.591&energy=0.908&valence=0.9200&duration_ms=196747&track_genre=rockabilly&artists=The%20Slapbacks&explicit=0&key=7&loudness=-3.977&mode=1&speechiness=0.0804&acousticness=0.243000&instrumentalness=0.000046&liveness=0.0918&tempo=119.581&time_signature=4
```

Server response

Code	Details
200	Response body {"result": 34}

Download

## 5. Conclusiones

- Variables numéricas presentan distribuciones muy sesgadas y colas largas (ej: duration\_ms, popularity).
- La matriz de correlaciones mostró relativamente baja multicolinealidad, lo que permite usar varias variables sin excesiva redundancia.



- track\_genre y artists tienen cardinalidad moderada; track\_name y album\_name son casi únicas.
- Para estas últimas, un one-hot sería inviable (dimensionalidad excesiva), por lo que se opta por codificación basada en target.
- Separación 67 %/33 % **antes** de cualquier transformación para evitar fugas de información.
- Uso de CatBoostEncoder en variables categóricas. La ventaja de este categorizador es que se obtiene una sola columna numérica por categoría y con regularización hacia la media global.
- **Log(1+x)** en duration\_ms, instrumentalness, num\_artists: comprime colas y estabiliza la varianza.
- **Exp(x)** en speechiness, liveness: amplifica diferencias sutiles en rangos [0-1].
- **Raíz cuadrada** en tempo y en las codificaciones del CatBooster, para moderar valores extremos.
- StandardScaler sobre todas las variables: media  $\approx 0$ , desviación  $\approx 1$ , mejor convergencia y comparabilidad de coeficientes.
- **BaggingRegressor**: Óptimo local de RMSE en  $\sim 660$  árboles, max\_samples=1.0, max\_features=0.6.
- **RandomForestRegressor**: Mejor desempeño a n\_estimators=500, max\_depth=25, max\_features=1.0.
- **XGBoostRegressor**: mínimo RMSE cerca de learning\_rate $\approx 0.15-0.2$  y max\_depth $\approx 6-7$ .
- **Optimización con Optuna** RMSE = 8.9765 n\_estimators = 2900 learning\_rate = 0.01913 max\_depth = 12. La mayor ventaja de Optuna es que captura interacciones entre parámetros y dirige la búsqueda hacia combinaciones globalmente óptimas sin hacer búsqueda exhaustiva.
- XGBoost gana con la menor métrica de error, compensando el mayor coste computacional con mejor precisión.