📊 Comparación de Modelos – Proyecto IMDb

Carlos Mejia, Miguel Rico, Juan Paez, Daniela Moreno

Objetivo

Comparar el desempeño de tres algoritmos (Random Forest, K-NN y K-Means) para estimar el ingreso bruto (Gross) de películas usando el dataset IMDb Top 1000.

Random Forest

Tipo: Aprendizaje supervisado (Regresión)

Funcionamiento:

Random Forest entrena múltiples árboles de decisión con subconjuntos aleatorios del dataset y promedia sus predicciones. Es robusto frente a sobreajuste y captura relaciones no lineales.

```
# ------------------------ 1. CARGA DE DATOS ------------------------
print("Descargando dataset desde Kaggle...")
path = kagglehub.dataset_download("harshitshankhdhar/imdb-dataset-of-top-1000-movies-and-tv-s

csv_files = [f for f in os.listdir(path) if f.endswith(".csv")]
df = pd.read_csv(os.path.join(path, csv_files[0]))

print(f"Dataset cargado: {csv_files[0]} con {df.shape[0]} filas y {df.shape[1]} columnas\n")

# Limpieza básica
df['Gross'] = df['Gross'].str.replace(',', '').astype(float)
df['Runtime'] = df['Runtime'].str.replace(' min', '').astype(float)
df['Released_Year'] = pd.to_numeric(df['Released_Year'], errors='coerce')

# Elimina filas con columnas clave nulas
df = df[['Runtime', 'Released_Year', 'IMDB_Rating', 'Meta_score', 'No_of_Votes', 'Gross']].dropna()

# ------------------------ 2. SEPARACIÓN DE DATOS ------------------------
X = df.drop(columns=['Gross'])
y = df['Gross']

X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.4, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)
```

```
# ----------------------- 3. ENTRENAMIENTO CON GRID MANUAL -----------------------
results = []
n_estimators_values = [50, 100]
max_depth_values = [5, 10]
min_samples_split_values = [2, 5]

for n_estimators in n_estimators_values:
    for max_depth in max_depth_values:
        for min_samples_split in min_samples_split_values:
            model = RandomForestRegressor(
                n_estimators=n_estimators,
                max_depth=max_depth,
                min_samples_split=min_samples_split,
                random_state=42
            )
            model.fit(X_train, y_train)
            pred_val = model.predict(X_val)
            mse = mean_squared_error(y_val, pred_val)
            r2 = r2_score(y_val, pred_val)

            results.append({
                'n_estimators': n_estimators,
                'max_depth': max_depth,
                'min_samples_split': min_samples_split,
                'MSE': mse,
                'R2': r2
            })

results_df = pd.DataFrame(results)
print(results_df)

# ----------------------- 4. MEJOR MODELO -----------------------
best_params = results_df.loc[results_df['R2'].idxmax()]
print("Mejores hiperparámetros:\n", best_params)

X_train_val = pd.concat([X_train, X_val])
y_train_val = pd.concat([y_train, y_val])

best_model = RandomForestRegressor(
    n_estimators=int(best_params['n_estimators']),
    max_depth=int(best_params['max_depth']) if not pd.isnull(best_params['max_depth']) else None,
    min_samples_split=int(best_params['min_samples_split']),
    random_state=42
)
```

```
best_model.fit(X_train_val, y_train_val)

# ---------------------- 5. EVALUACIÓN FINAL ----------------------
test_pred = best_model.predict(X_test)
print("Test MSE:", mean_squared_error(y_test, test_pred))
print("Test R2:", r2_score(y_test, test_pred))

# ---------------------- 6. PREDICCIÓN NUEVO DATO ----------------------
nuevo = pd.DataFrame([{
    'Runtime': 120,
    'Released_Year': 2020,
    'IMDB_Rating': 8.0,
    'Meta_score': 75,
    'No_of_Votes': 500000
}])

pred_gross = best_model.predict(nuevo)
print(f"Predicted Gross: {pred_gross[0]}")

# Guardar y cargar modelo
joblib.dump(best_model, 'random_forest_gross_model.pkl')
print("Modelo guardado correctamente.")
loaded_model = joblib.load('random_forest_gross_model.pkl')
print(f"Predicción con modelo cargado: {loaded_model.predict(nuevo)[0]}")
```

Mejores Hiperparámetros:

- n_estimators: 100
- max_depth: 5
- min_samples_split: 2

Resultados:

- **Mejor R^2:** 0.5456 (validación)
- **Test R^2:** 0.434
- **Predicción para nueva película:** $226,518,004.27

Conclusión:

Fue el modelo más preciso para predecir ingresos. Ideal para tareas de regresión donde hay relaciones complejas entre variables.

K-Nearest Neighbors (K-NN)

Tipo: Aprendizaje supervisado (Regresión)

Funcionamiento:

Predice el valor objetivo (Gross) promediando los valores de las k películas más cercanas (en cuanto a duración, rating, etc.).

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
import kagglehub
import os
import joblib

# ------------------------ 1. CARGA DE DATOS ------------------------
print("Descargando dataset desde Kaggle...")
path = kagglehub.dataset_download("harshitshankhdhar/imdb-dataset-of-top-1000-movies-and-tv-s

csv_files = [f for f in os.listdir(path) if f.endswith(".csv")]
df = pd.read_csv(os.path.join(path, csv_files[0]))

print(f"Dataset cargado: {csv_files[0]} con {df.shape[0]} filas y {df.shape[1]} columnas\n")

# Limpieza básica
df['Gross'] = df['Gross'].str.replace(',', '').astype(float)
df['Runtime'] = df['Runtime'].str.replace(' min', '').astype(float)
df['Released_Year'] = pd.to_numeric(df['Released_Year'], errors='coerce')

# Elimina filas con columnas clave nulas
df = df[['Runtime', 'Released_Year', 'IMDB_Rating', 'Meta_score', 'No_of_Votes', 'Gross']].dropna()

# ------------------------ 2. SEPARACIÓN DE DATOS ------------------------
X = df.drop(columns=['Gross'])
y = df['Gross']

X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.4, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)

# ------------------------ 3. ENTRENAMIENTO CON KNN ------------------------
results = []
```

```
k_values = [3, 5, 7, 9, 11]

for k in k_values:
    knn = KNeighborsRegressor(n_neighbors=k)
    knn.fit(X_train, y_train)
    pred_val = knn.predict(X_val)
    mse = mean_squared_error(y_val, pred_val)
    r2 = r2_score(y_val, pred_val)

    results.append({
        'k': k,
        'MSE': mse,
        'R2': r2
    })

results_df = pd.DataFrame(results)
print(results_df)

# ------------------------ 4. MEJOR MODELO ------------------------
best_params = results_df.loc[results_df['R2'].idxmax()]
print("Mejor valor de k:\n", best_params)

X_train_val = pd.concat([X_train, X_val])
y_train_val = pd.concat([y_train, y_val])

best_knn = KNeighborsRegressor(n_neighbors=int(best_params['k']))
best_knn.fit(X_train_val, y_train_val)

# ------------------------ 5. EVALUACIÓN FINAL ------------------------
test_pred = best_knn.predict(X_test)
print("Test MSE:", mean_squared_error(y_test, test_pred))
print("Test R2:", r2_score(y_test, test_pred))

# ------------------------ 6. PREDICCIÓN NUEVO DATO ------------------------
nuevo = pd.DataFrame([{
    'Runtime': 120,
    'Released_Year': 2020,
    'IMDB_Rating': 8.0,
    'Meta_score': 75,
    'No_of_Votes': 500000
}])

pred_gross = best_knn.predict(nuevo)
print(f"Predicted Gross: {pred_gross[0]}")
```

```
# Guardar y cargar modelo
joblib.dump(best_knn, 'knn_gross_model.pkl')
print("Modelo guardado correctamente.")
loaded_model = joblib.load('knn_gross_model.pkl')
print(f"Predicción con modelo cargado: {loaded_model.predict(nuevo)[0]}")
```

Mejor k: 11

Resultados:

- **Mejor R^2:** 0.304 (validación)
- **Test R^2:** 0.297
- **Predicción para nueva película:** $99,489,051.36

Conclusión:

Más sencillo y fácil de entender, pero menos preciso. Su rendimiento depende mucho de la distancia entre puntos y la escala de las variables.

K-Means

Tipo: Aprendizaje no supervisado (Clustering)

Funcionamiento:

Agrupa películas similares en clústeres. No predice directamente Gross, sino que asigna la película a un clúster y estima su ingreso con el promedio del clúster.

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
import kagglehub
import os
import numpy as np

# ----------------------- 1. CARGA DE DATOS -----------------------
print("Descargando dataset desde Kaggle...")
path = kagglehub.dataset_download("harshitshankhdhar/imdb-dataset-of-top-1000-movies-and-tv-s
csv_files = [f for f in os.listdir(path) if f.endswith(".csv")]
df = pd.read_csv(os.path.join(path, csv_files[0]))
print(f"Dataset cargado: {csv_files[0]} con {df.shape[0]} filas y {df.shape[1]} columnas\n")

# Limpieza
```

```
df['Gross'] = df['Gross'].str.replace(',', '').astype(float)
df['Runtime'] = df['Runtime'].str.replace(' min', '').astype(float)
df['Released_Year'] = pd.to_numeric(df['Released_Year'], errors='coerce')
df = df[['Runtime', 'Released_Year', 'IMDB_Rating', 'Meta_score', 'No_of_Votes', 'Gross']].dropna()

# ----------------------- 2. ESCALADO DE DATOS -----------------------
scaler = StandardScaler()
scaled_data = scaler.fit_transform(df.drop(columns=['Gross']))

# ----------------------- 3. K-MEANS CLUSTERING -----------------------
k = 5  # Número de clústeres
kmeans = KMeans(n_clusters=k, random_state=42)
df['Cluster'] = kmeans.fit_predict(scaled_data)

# ----------------------- 4. ANÁLISIS DE CLÚSTERES -----------------------
cluster_summary = df.groupby('Cluster')['Gross'].agg(['mean', 'count']).reset_index()
print("Ingreso bruto promedio por clúster:\n", cluster_summary)

# ----------------------- 5. VISUALIZACIÓN -----------------------
pca = PCA(n_components=2)
pca_data = pca.fit_transform(scaled_data)
df['PCA1'] = pca_data[:, 0]
df['PCA2'] = pca_data[:, 1]

plt.figure(figsize=(10, 6))
sns.scatterplot(data=df, x='PCA1', y='PCA2', hue='Cluster', palette='tab10')
plt.title("🎬 Películas agrupadas por similitud (PCA + KMeans)")
plt.xlabel("PCA1")
plt.ylabel("PCA2")
plt.tight_layout()
plt.show()

# ----------------------- 6. PREDICCIÓN PARA NUEVA PELÍCULA -----------------------
nueva_peli = pd.DataFrame([{
    'Runtime': 120,
    'Released_Year': 2020,
    'IMDB_Rating': 8.0,
    'Meta_score': 75,
    'No_of_Votes': 500000
}])
nueva_scaled = scaler.transform(nueva_peli)
clust_pred = kmeans.predict(nueva_scaled)[0]
gross_aprox = cluster_summary.loc[cluster_summary['Cluster'] == clust_pred, 'mean'].values[0]
```

```
print(f"\n🎯 La nueva película cae en el clúster {clust_pred} con un ingreso bruto promedio estimado
```

Resultados (5 clústeres):

- Película nueva asignada al **Clúster 3**
- **Ingreso promedio del clúster:** $74,120,313.46

Conclusión:

Útil para explorar patrones ocultos y segmentar datos. No es un modelo de predicción directa, pero sí una herramienta analítica complementaria.

⚖️ Comparación General

Algoritmo	Tipo	R^2 (Test)	**Predicción** Gross	**Observaciones**
Random Forest	Supervisado	0.434	$226M	Mejor modelo predictivo
K-NN	Supervisado	0.297	$99M	Menor precision que RF
K-Means	No Supervisado	N/A	$74M (estimado)	Útil para análisis exploratorio