4.3 Avance de proyecto 2: Sistema de Recomendación

Análisis de grandes volúmenes de datos

TC4034 grupo 10 Equipo 26

Luis Arturo Dan Fong | A01650672
Eduardo Rodríguez Ramírez| A01794892
Felipe Enrique Vázquez Ruiz | A01638116

26 de Mayo de 2024

Sistema de Recomendación de Música Personalizado Basado en Listas de Reproducción de Spotify

1.- Descripción del algoritmo de recomendación avanzado elegido:

El algoritmo de recomendación elegido es un modelo de recomendación colaborativa basado en embeddings. En este modelo, se utilizan embeddings para representar tanto a los usuarios como a los artistas en un espacio vectorial de dimensionalidad reducida. Luego, se entrena un modelo de aprendizaje automático para predecir las reproducciones de un usuario para un artista específico. El modelo se optimiza para minimizar el error cuadrático medio entre las reproducciones reales y las predicciones del modelo.

2.- Identificación y justificación de métricas de evaluación utilizadas:

1. Precision@K: Mide la proporción de elementos relevantes recuperados en las primeras K recomendaciones. Es útil para evaluar la precisión de las recomendaciones en una lista corta.

2. Mean Average Precision (MAP): Calcula el promedio de los valores de precisión para cada usuario sobre el conjunto de recomendaciones. Proporciona una medida agregada de la precisión de las recomendaciones.

3. Normalized Discounted Cumulative Gain (NDCG): Evalúa la calidad de las recomendaciones considerando la relevancia y el orden de las mismas en una lista de recomendación. Es una métrica útil para clasificar las recomendaciones en función de su relevancia y posición.

Estas métricas se seleccionaron porque proporcionan una evaluación integral de la calidad de las recomendaciones, considerando tanto la precisión de las mismas como su orden relativo.

3.- Experimentación con al menos 1 algoritmo de recomendación:

El modelo propuesto para este sistema de recomendación consta de una red neuronal basada en embeddings que utiliza información sobre las preferencias de los usuarios y las características de los artistas para hacer predicciones sobre las reproducciones de los usuarios, la arquitectura del modelo consta de lo siguiente:

Entradas: El modelo recibe dos tipos de entradas: el índice del usuario y el índice del artista. Estos índices se utilizan para buscar los vectores de embedding correspondientes en las capas de embedding del modelo.

Capas de embedding: El modelo utiliza capas de embedding para representar a los usuarios y a los artistas en un espacio vectorial de dimensionalidad reducida. Cada usuario y cada artista están asociados con un vector de embedding único que captura sus características y preferencias.

Capas densas: Después de obtener los vectores de embedding para el usuario y el artista, se concatenan y se pasan a través de capas densas de redes neuronales. Estas capas aprenden las relaciones complejas entre los usuarios y los artistas a partir de los vectores de embedding.

Capa de salida: La capa de salida del modelo produce una única salida, que es la predicción de la reproducción del usuario para el artista específico. Esta predicción se calcula como un valor continuo que representa la probabilidad de que el usuario reproduzca el artista.

Resultados de la evaluación del modelo:

Después de entrenar el modelo, se evaluó su rendimiento utilizando las métricas mencionadas anteriormente. Se obtuvieron los siguientes resultados de evaluación:

Precision@10: 0.2463

Mean Average Precision (MAP)@10: 0.0308

Normalized Discounted Cumulative Gain (NDCG)@10: 0.1447

Una precisión del 24.63% en el top 10 significa que aproximadamente una cuarta parte de las recomendaciones en las primeras 10 posiciones son relevantes para los usuarios lo cual indica una calidad razonablemente buena en términos de precisión de las recomendaciones que el sistema genera.

Un MAP de 3.08% indica que, en promedio, el modelo tiene un rendimiento bajo en términos de la precisión del ranking de las recomendaciones, quiere decir que aunque algunas recomendaciones en las posiciones superiores pueden ser acertadas pero no para la mayoría del ranking de cada usuario.

NDCG de 14.47% sugiere que el modelo tiene un rendimiento moderado en términos de la relevancia y el orden de las recomendaciones en las primeras 10 posiciones. Aunque no es excepcionalmente alto, indica que las recomendaciones relevantes tienden a estar más arriba en la lista.

Referencias:

Schedl, M., Zamani, H., Chen, C., Deldjoo, Y., & Elahi, M. (2018). Current challenges and future directions in music recommender systems research. International Journal of Multimedia Information Retrieval, 7(4), 277-300.

Bonnin, G., & Jannach, D. (2015). Automated playlist continuation at scale. In Proceedings of the 9th ACM Conference on Recommender Systems (pp. 115-122).

Implementación de al menos un algoritmo de recomendación avanzado

Instalando Dependencias

[ ] ↳ 12 celdas ocultas

Preprocesamiento

↳ 22 celdas ocultas

Cargando datos preprocesados

```
loaded_keys = load_dict_keys_from_pickle('keys.pkl')
dfs = load_dataframes_from_parquet(loaded_keys, working_path)
```

```
for k, df in dfs.items():
    print(k, df.head(10))
```

```
user_id [Row(user_id='00055176fea33f6e027cd3302289378b', user_id_index=1), Row(user_id='0007f3dd09c91198371454c608d47f22', user_id_index=2), Row(user_id='000b0f32b5739f052b9d40fcc5c41079', user_id_index=3), Row(user_id='000c11
artistname [Row(artistname=' Dolce', artistname_index=1), Row(artistname=' OneVoice', artistname_index=2), Row(artistname='!!!', artistname_index=3), Row(artistname='!!! (Chk Chk Chk)', artistname_index=4), Row(artistname='!!!
trackname [Row(trackname=' "Cachaito" López Y "Guajiro" Mirabal De Buena Vista Social Club Y Manuel "Galbán" Torralba"', trackname_index=1), Row(trackname=' 15 Years of Tummy Touch Records in Dub', trackname_index=2), Row(trac
playlistname [Row(playlistname=' ', playlistname_index=1), Row(playlistname='        waves', playlistname_index=2), Row(playlistname='  11', playlistname_index=3), Row(playlistname='  Frida', playlistname_index=4), Row(playlis
data [Row(user_id_index=15051, artistname_index=307, trackname_index=6, playlistname_index=100652), Row(user_id_index=5669, artistname_index=456, trackname_index=19, playlistname_index=869), Row(user_id_index=13756, artistname
data_top_artists [Row(artistname_index=258175, user_id_index=6734, trackname_index=45362, playlistname_index=53455), Row(artistname_index=258175, user_id_index=6734, trackname_index=57234, playlistname_index=53455), Row(artist
norm_data_top_artists [Row(user_id_index=6734, artistname_index=258175, reproductions=128, normalized_reproduction=0.0379671150971599 4), Row(user_id_index=15667, artistname_index=258175, reproductions=70, normalized_reproducti
```

```
#Dividiendo el dataset segun el numero de usuarios distintos

unique_usr_ids =  dfs['norm_data_top_artists'].select("user_id_index").distinct().orderBy("user_id_index")


train_usr_ids, test_usr_ids = unique_usr_ids.randomSplit([0.8, 0.2], seed=42)


train_data = dfs['norm_data_top_artists'].join(train_usr_ids, on="user_id_index", how="inner")
test_data = dfs['norm_data_top_artists'].join(test_usr_ids, on="user_id_index", how="inner")

print("Training DataFrame:")
train_data.show()

```

```
Training DataFrame:
+-------------+----------------+-------------+----------------------+
|user_id_index|artistname_index|reproductions|normalized_reproduction|
+-------------+----------------+-------------+----------------------+
|         6734|          258175|          128|    0.0379671150971599|
|        15667|          258175|           70|    0.0206278026905829 6|
|        14817|          258175|           99|    0.0292974588938714 5|
|         8561|          258175|           16|   0.00448430493273542 6|
|        14137|          258175|           48|   0.01405082212257100 1|
|         6498|          258175|          314|    0.0935724962630792 2|
|         5574|          258175|          113|    0.0334828101644245 2|
|        15611|          258175|          171|    0.0508221225710014 9|
|         5371|          258175|          106|    0.0313901345291479 8|
|         9708|          258175|           36|   0.01046337817638266|
|        15063|          258175|          106|    0.0313901345291479 8|
|          878|          258175|          160|    0.0475336322869955 2|
|         9000|          258175|          107|   0.03168908819133034 5|
|         3124|          258175|           89|   0.02630792227204783 3|
|         5559|          258175|          109|   0.03228699551569507|
|         9336|          258175|          165|    0.0490284005907907 3|
|          289|          258175|          106|    0.0313901345291479 8|
|        10098|          258175|           78|   0.02301943198804185 3|
|         4538|          258175|          170|    0.0505231689088191 3|
|        10063|          258175|          106|    0.0313901345291479 8|
+-------------+----------------+-------------+----------------------+
only showing top 20 rows
```

```
train_data.show()
```

```
+-------------+----------------+-------------+----------------------+
|user_id_index|artistname_index|reproductions|normalized_reproduction|
+-------------+----------------+-------------+----------------------+
|         5803|          285680|            4|   8.9686098654708 52E-4|
|         5803|          285075|            5|   0.0011958146487294 47|
|         5803|          278507|          221|    0.0657698056801195 9|
|         5803|          277472|           16|   0.00448430493273542 6|
|         5803|          272596|            2|   2.9895366218236 17...|
|         5803|          269254|            1|                   0.0|
|         5803|          260511|            2|   2.9895366218236 17...|
|         5803|          258829|           12|   0.0032884902840059 79|
|         5803|          257149|            5|   0.0011958146487294 47|
|         5803|          248402|            3|   5.9790732436472 35E-4|
|         5803|          246063|          106|    0.0313901345291479 8|
|         5803|          204527|           31|   0.0089686098654708 52|
|         5803|          203664|           12|   0.0032884902840059 79|
|         5803|          203240|            2|   2.9895366218236 17...|
|         5803|          169467|            9|   0.0023916292974588 94|
|         5803|          168699|            4|   8.9686098654708 52E-4|
|         5803|          164895|            1|                   0.0|
|         5803|          157985|            2|   2.9895366218236 17...|
|         5803|          138370|           20|   0.0056801195814648 73|
|         5803|          126554|           37|   0.0107623318385650 23|
+-------------+----------------+-------------+----------------------+
only showing top 20 rows
```

```
total_rows = train_data.count()

# Calculate unique counts
n_users = train_data.select("user_id_index").distinct().count()
n_artists = train_data.select("artistname_index").distinct().count()

# Ensure the maximum index is within range
max_user_id = train_data.select("user_id_index").rdd.max()[0]
max_artist_id = train_data.select("artistname_index").rdd.max()[0]

max_id_users = max(n_users, max_user_id + 1)
max_id_artists = max(n_artists, max_artist_id + 1)

print(f"Max ID users: {max_id_users}")
print(f"Max ID artists: {max_id_artists}")
```

```
Max ID users: 15915
Max ID artists: 285681
```

```
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Embedding, Flatten, Concatenate, Dense
from petastorm.spark import SparkDatasetConverter, make_spark_converter
from petastorm.tf_utils import make_petastorm_dataset
from tensorflow.keras.callbacks import EarlyStopping
```

```
# Calculate steps per epoch
batch_size = 2048
steps_per_epoch = total_rows // batch_size
epochs = 10
embedding_size = 100
```

```
# Initialize SparkDatasetConverter
converter = make_spark_converter(train_data)


user_input = Input(shape=(1,), name='user_input')
user_embedding = Embedding(max_id_users, embedding_size, input_length=1, name='user_embedding')(user_input)
user_vec = Flatten(name='user_flatten')(user_embedding)

artist_input = Input(shape=(1,), name='artist_input')
artist_embedding = Embedding(max_id_artists, embedding_size, input_length=1, name='artist_embedding')(artist_input)
artist_vec = Flatten(name='artist_flatten')(artist_embedding)

concat = Concatenate()([user_vec, artist_vec])
dense1 = Dense(128, activation='relu')(concat)
dense2 = Dense(64, activation='relu')(dense1)
output = Dense(1)(dense2)

model = Model([user_input, artist_input], output)
model.compile(optimizer='adam', loss='mean_squared_error')


# Training with Petastorm dataset
def transform_row(row):
    return {"user_input": row.user_id_index, "artist_input": row.artistname_index}, row.normalized_reproduction

# Create TensorFlow dataset
with converter.make_tf_dataset(batch_size=batch_size) as dataset:
    dataset = dataset.map(lambda x: transform_row(x))

    # Implement early stopping
    early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)

    model.fit(dataset, epochs=epochs, steps_per_epoch=steps_per_epoch, validation_data=dataset, validation_steps=steps_per_epoch, callbacks=[early_stopping])
```

```
/usr/local/lib/python3.10/dist-packages/petastorm/fs_utils.py:88: FutureWarning: pyarrow.localfs is deprecated as of 2.0.0, please use pyarrow.fs.LocalFileSystem instead.
  self._filesystem = pyarrow.localfs
WARNING:petastorm.spark.spark_dataset_converter:Converting floating-point columns to float32
WARNING:petastorm.spark.spark_dataset_converter:The median size 472859 B (< 50 MB) of the parquet files is too small. Total size: 906227 B. Increase the median file size by calling df.repartition(n) or df.coalesce(n), which mi
/usr/local/lib/python3.10/dist-packages/petastorm/etl/dataset_metadata.py:402: FutureWarning: Passing 'use_legacy_dataset=True' to get the legacy behaviour is deprecated as of pyarrow 11.0.0, and the legacy implementation will
  dataset = pq.ParquetDataset(path_or_paths, filesystem=fs, validate_schema=False, metadata_nthreads=10)
/usr/local/lib/python3.10/dist-packages/petastorm/etl/dataset_metadata.py:402: FutureWarning: Specifying the 'metadata_nthreads' argument is deprecated as of pyarrow 8.0.0, and the argument will be removed in a future version
  dataset = pq.ParquetDataset(path_or_paths, filesystem=fs, validate_schema=False, metadata_nthreads=10)
/usr/local/lib/python3.10/dist-packages/petastorm/etl/dataset_metadata.py:362: FutureWarning: 'ParquetDataset.common_metadata' attribute is deprecated as of pyarrow 5.0.0 and will be removed in a future version.
  if not dataset.common_metadata:
/usr/local/lib/python3.10/dist-packages/petastorm/reader.py:420: FutureWarning: Passing 'use_legacy_dataset=True' to get the legacy behaviour is deprecated as of pyarrow 11.0.0, and the legacy implementation will be removed in
  self.dataset = pq.ParquetDataset(dataset_path, filesystem=pyarrow_filesystem,
/usr/local/lib/python3.10/dist-packages/petastorm/reader.py:420: FutureWarning: Specifying the 'metadata_nthreads' argument is deprecated as of pyarrow 8.0.0, and the argument will be removed in a future version
  self.dataset = pq.ParquetDataset(dataset_path, filesystem=pyarrow_filesystem,
/usr/local/lib/python3.10/dist-packages/petastorm/unischema.py:317: FutureWarning: 'ParquetDataset.pieces' attribute is deprecated as of pyarrow 5.0.0 and will be removed in a future version. Specify 'use_legacy_dataset=False'
  meta = parquet_dataset.pieces[0].get_metadata()
/usr/local/lib/python3.10/dist-packages/petastorm/unischema.py:321: FutureWarning: 'ParquetDataset.partitions' attribute is deprecated as of pyarrow 5.0.0 and will be removed in a future version. Specify 'use_legacy_dataset=Fa
  for partition in (parquet_dataset.partitions or []):
/usr/local/lib/python3.10/dist-packages/petastorm/etl/dataset_metadata.py:253: FutureWarning: 'ParquetDataset.metadata' attribute is deprecated as of pyarrow 5.0.0 and will be removed in a future version.
  metadata = dataset.metadata
/usr/local/lib/python3.10/dist-packages/petastorm/etl/dataset_metadata.py:254: FutureWarning: 'ParquetDataset.common_metadata' attribute is deprecated as of pyarrow 5.0.0 and will be removed in a future version.
  common_metadata = dataset.common_metadata
/usr/local/lib/python3.10/dist-packages/petastorm/etl/dataset_metadata.py:350: FutureWarning: 'ParquetDataset.pieces' attribute is deprecated as of pyarrow 5.0.0 and will be removed in a future version. Specify 'use_legacy_dat
  futures_list = [thread_pool.submit(_split_piece, piece, dataset.fs.open) for piece in dataset.pieces]
/usr/local/lib/python3.10/dist-packages/petastorm/etl/dataset_metadata.py:350: FutureWarning: 'ParquetDataset.fs' attribute is deprecated as of pyarrow 5.0.0 and will be removed in a future version. Specify 'use_legacy_dataset
  futures_list = [thread_pool.submit(_split_piece, piece, dataset.fs.open) for piece in dataset.pieces]
/usr/local/lib/python3.10/dist-packages/petastorm/etl/dataset_metadata.py:334: FutureWarning: ParquetDatasetPiece is deprecated as of pyarrow 5.0.0 and will be removed in a future version.
  return [pq.ParquetDatasetPiece(piece.path, open_file_func=fs_open,
/usr/local/lib/python3.10/dist-packages/petastorm/arrow_reader_worker.py:132: FutureWarning: Passing 'use_legacy_dataset=True' to get the legacy behaviour is deprecated as of pyarrow 11.0.0, and the legacy implementation will
  self._dataset = pq.ParquetDataset(
/usr/local/lib/python3.10/dist-packages/petastorm/arrow_reader_worker.py:140: FutureWarning: 'ParquetDataset.fs' attribute is deprecated as of pyarrow 5.0.0 and will be removed in a future version. Specify 'use_legacy_dataset=
  parquet_file = ParquetFile(self._dataset.fs.open(piece.path))
/usr/local/lib/python3.10/dist-packages/petastorm/arrow_reader_worker.py:288: FutureWarning: 'ParquetDataset.partitions' attribute is deprecated as of pyarrow 5.0.0 and will be removed in a future version. Specify 'use_legacy_
  partition_names = self._dataset.partitions.partition_names if self._dataset.partitions else set()
/usr/local/lib/python3.10/dist-packages/petastorm/arrow_reader_worker.py:291: FutureWarning: 'ParquetDataset.partitions' attribute is deprecated as of pyarrow 5.0.0 and will be removed in a future version. Specify 'use_legacy_
  table = piece.read(columns=column_names - partition_names, partitions=self._dataset.partitions)
Epoch 1/10
101/101 [==============================] - 87s 847ms/step - loss: 1.8578e-04 - val_loss: 7.0677e-05
Epoch 2/10
101/101 [==============================] - 82s 815ms/step - loss: 6.2150e-05 - val_loss: 6.1484e-05
Epoch 3/10
101/101 [==============================] - 86s 852ms/step - loss: 5.1474e-05 - val_loss: 4.9467e-05
Epoch 4/10
101/101 [==============================] - 82s 817ms/step - loss: 4.5729e-05 - val_loss: 4.7544e-05
Epoch 5/10
101/101 [==============================] - 84s 836ms/step - loss: 5.1317e-05 - val_loss: 4.0930e-05
Epoch 6/10
101/101 [==============================] - 82s 814ms/step - loss: 3.9329e-05 - val_loss: 3.8764e-05
Epoch 7/10
101/101 [==============================] - 84s 832ms/step - loss: 3.6409e-05 - val_loss: 4.5511e-05
Epoch 8/10
101/101 [==============================] - 83s 819ms/step - loss: 2.5931e-05 - val_loss: 3.7847e-05
Epoch 9/10
101/101 [==============================] - 83s 827ms/step - loss: 4.0968e-05 - val_loss: 3.2006e-05
Epoch 10/10
101/101 [==============================] - 83s 819ms/step - loss: 3.1517e-05 - val_loss: 3.0189e-05
```

```
model_save_path = working_path + f"//model_{epochs}_{batch_size}"
```

```
model.save(model_save_path)
```

```
loaded_model = tf.keras.models.load_model(model_save_path)
```

Para evaluar este modelo se utillizaran 3 metricas:

- Presision@K
- Mean Average Precision (MAP)
- Normalized Discounted Cumulative Gain (NDCG)

```python
def recommend_top_k_artists(user_id_index, model, data, k=10):
    artist_ids = data["artistname_index"].unique()
    user_ids = [user_id_index] * len(artist_ids)
    user_ids_array = np.array(user_ids)
    artist_ids_array = np.array(artist_ids)
    predictions = model.predict([user_ids_array, artist_ids_array], verbose=0)
    predicted_scores = list(zip(artist_ids, predictions))
    top_k_artists = sorted(predicted_scores, key=lambda x: x[1], reverse=True)[:k]
    top_k_artist_ids = [artist_id for artist_id, score in top_k_artists]
    return top_k_artist_ids

#Precision@K

def precision_at_k(recommended_items, relevant_items, k):
    relevant_set = set(relevant_items)
    recommended_set = set(recommended_items)
    return len(recommended_set & relevant_set) / k

#MAP

def average_precision(actual, predicted):
    score = 0.0
    num_hits = 0.0

    for i, p in enumerate(predicted):
        if p in actual and p not in predicted[:i]:
            num_hits += 1.0
            score += num_hits / (i + 1.0)

    if not actual:
        return 0.0
    return score / min(len(actual), len(predicted))

def mean_average_precision(actual_items, predicted_items):
    return np.mean([average_precision(actual, predicted) for actual, predicted in zip(actual_items, predicted_items)])

# NDCG

def dcg_at_k(r, k):
    r = np.asfarray(r)[:k]
    if r.size:
        return np.sum(np.divide(np.power(2, r) - 1, np.log2(np.arange(2, r.size + 2))))
    return 0.0

def ndcg_at_k(r, k):
    idcg = dcg_at_k(sorted(r, reverse=True), k)
    if not idcg:
        return 0.0
    return dcg_at_k(r, k) / idcg

def normalized_discounted_cumulative_gain(actual, predicted, k):
    r = [1 if p in actual else 0 for p in predicted]
    return ndcg_at_k(r, k)
```

```python
def evaluate_model(test_data, model, k=10):
    user_ids = test_data['user_id_index'].unique()
    avg_map = 0.0
    avg_ndcg = 0.0
    precision_scores = []

    for user_id in user_ids:
        relevant_items = test_data[test_data['user_id_index'] == user_id]['artistname_index'].tolist()

        #Se ignoran ususarios que tengan pocos artistas (Arranque en frío)
        if len(relevant_items) < 25:
            continue

        recommended_items = recommend_top_k_artists(user_id, model, test_data, k)


        #Calcular Precision@K
        precision = precision_at_k(recommended_items, relevant_items, k)
        precision_scores.append(precision)

        # Calcular MAP para el usuario actual
        map_score = average_precision(relevant_items, recommended_items[:k])
        avg_map += map_score

        # Calcular NDCG para el usuario actual
        ndcg_score = normalized_discounted_cumulative_gain(relevant_items, recommended_items, k)
        avg_ndcg += ndcg_score

    # Calcular promedio de MAP y NDCG para todos los usuarios
    avg_precisionAtK = np.mean(precision_scores)
    avg_map /= len(user_ids)
    avg_ndcg /= len(user_ids)

    return avg_precisionAtK, avg_map, avg_ndcg

# Evaluar el modelo
avg_precisionAtK, avg_map, avg_ndcg = evaluate_model(test_data.toPandas(), model, k=10)
print(f"Precision@10: {avg_precisionAtK:.4f}")
print(f"Mean Average Precision (MAP)@10: {avg_map:.4f}")
print(f"Normalized Discounted Cumulative Gain (NDCG)@10: {avg_ndcg:.4f}")
```

```
Precision@10: 0.2463
Mean Average Precision (MAP)@10: 0.0308
Normalized Discounted Cumulative Gain (NDCG)@10: 0.1447
```

Dando Recomendaciones

```python
test_data.show()
```

```
+-------------+----------------+-------------+-----------------------+
|user_id_index|artistname_index|reproductions|normalized_reproduction|
+-------------+----------------+-------------+-----------------------+
|          355|          258175|           58|   0.017040358744394617|
|         8608|          258175|           42|   0.012257100149476832|
|        12719|          258175|          138|    0.04095665171898356|
|          550|          258175|          183|    0.05440956651718983|
|         4052|          258175|           89|   0.026307922272047833|
|        11906|          258175|           90|   0.026606875934230195|
|        12659|          258175|          126|    0.03736920777279522|
|         6366|          258175|           35|   0.010164424514200299|
|         3222|          258175|           17|   0.004783258594917788|
|          313|          258175|          108|    0.03198804185351271|
|        13393|          258175|           47|   0.013751868460388639|
|         2248|          258175|            5|   0.001195814648729447|
|         8162|          258175|           56|   0.016442451420029897|
|        13626|          258175|          113|    0.03348281016442452|
|         2830|          258175|           14|   0.003886397608370...|
|        14930|          258175|           12|   0.003288490284005979|
|         3102|          258175|            2|   2.989536621823617...|
|         6336|          258175|           47|   0.013751868460388639|
```

```
|          261|          258175|          12|   0.003288490284005979|
|        13235|          258175|          90|   0.026606875934230195|
+-------------+----------------+------------+-----------------------+
only showing top 20 rows
```

```
user_id_index=261
#13235

usr_reps = test_data.filter(fn.col("user_id_index") == user_id_index).distinct()\
    .join(dfs['artistname'],'artistname_index').orderBy('reproductions', ascending=False).select('artistname','reproductions')

n_artist_usr=usr_reps.count()
usr_reps.show(n=n_artist_usr,truncate=False)
```

```
+----------------------+-------------+
|artistname            |reproductions|
+----------------------+-------------+
|Grateful Dead         |42           |
|The Smashing Pumpkins |40           |
|Iron Maiden           |31           |
|Muse                  |16           |
|Blur                  |16           |
|Radiohead             |14           |
|Coldplay              |13           |
|U2                    |13           |
|Arctic Monkeys        |12           |
|Foo Fighters          |12           |
|The Smiths            |12           |
|The Rolling Stones    |12           |
|Kings Of Leon         |12           |
|David Bowie           |12           |
|The Killers           |11           |
|The Clash             |10           |
|The Who               |9            |
|Green Day             |9            |
|Nirvana               |8            |
|The Cure              |7            |
|The Strokes           |7            |
|R.E.M.                |6            |
|Weezer                |6            |
|Nine Inch Nails       |5            |
|Guns N' Roses         |5            |
|Beastie Boys          |5            |
|Florence + The Machine|4            |
|Daft Punk             |4            |
|Linkin Park           |4            |
|Pearl Jam             |3            |
|Depeche Mode          |3            |
|Arcade Fire           |3            |
|The Doors             |3            |
|Led Zeppelin          |3            |
|Queens Of The Stone Age|3           |
|Vampire Weekend       |2            |
|Mumford & Sons        |2            |
|Massive Attack        |2            |
|The White Stripes     |2            |
|Gorillaz              |2            |
|blink-182             |2            |
|Beck                  |2            |
|Fall Out Boy          |2            |
|Red Hot Chili Peppers |1            |
|Johnny Cash           |1            |
|Various Artists       |1            |
+----------------------+-------------+
```

```
#Generando 10 recomendaciones
from pyspark.sql.types import StructType, StructField, StringType, IntegerType


rec_items = recommend_top_k_artists(user_id_index, loaded_model, train_data.toPandas(), 10)

rec_items = [(int(item),) for item in rec_items]

rec_artist_names = spark.createDataFrame(rec_items, StructType([StructField("artistname_index", IntegerType(), True)])).join(dfs['artistname'],'artistname_index')

print(f"Recommended artists for user {user_id_index}")
rec_artist_names.show(n=rec_artist_names.count(),truncate=False)
```

```
Recommended artists for user 261
+---------------+----------------------+
|artistname_index|artistname            |
+---------------+----------------------+
|171812          |Miles Davis           |
|168699          |Metallica             |
|274942          |Vitamin String Quartet|
|127024          |John Williams         |
|258175          |The Smiths            |
|100812          |Grateful Dead         |
|269792          |U2                    |
|100085          |Gorillaz              |
|194927          |Pearl Jam             |
|136060          |Kendrick Lamar        |
+---------------+----------------------+
```

```
#Similutides:
usr_reps.join(rec_artist_names,'artistname').select('artistname').show(n=n_artist_usr)
```

```
+-------------+
|   artistname|
+-------------+
|     Gorillaz|
|Grateful Dead|
|    Pearl Jam|
|   The Smiths|
|           U2|
+-------------+
```

```
precision = usr_reps.join(rec_artist_names,'artistname').count()/10

print(f"Precisión de las recomendaciones para el usuario {user_id_index}: {precision}")
```

```
Precisión de las recomendaciones para el usuario 261: 0.5
```

Ejercicio complementario Arranque en frío

Instalación de paqueterías no incluidas en Google colab

```
pip install opendatasets
```

```
Requirement already satisfied: opendatasets in /usr/local/lib/python3.10/dist-packages (0.1.22)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from opendatasets) (4.66.4)
Requirement already satisfied: kaggle in /usr/local/lib/python3.10/dist-packages (from opendatasets) (1.6.14)
Requirement already satisfied: click in /usr/local/lib/python3.10/dist-packages (from opendatasets) (8.1.7)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.10/dist-packages (from kaggle->opendatasets) (1.16.0)
Requirement already satisfied: certifi>=2023.7.22 in /usr/local/lib/python3.10/dist-packages (from kaggle->opendatasets) (2024.2.2)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.10/dist-packages (from kaggle->opendatasets) (2.8.2)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from kaggle->opendatasets) (2.31.0)
Requirement already satisfied: python-slugify in /usr/local/lib/python3.10/dist-packages (from kaggle->opendatasets) (8.0.4)
Requirement already satisfied: urllib3 in /usr/local/lib/python3.10/dist-packages (from kaggle->opendatasets) (2.0.7)
Requirement already satisfied: bleach in /usr/local/lib/python3.10/dist-packages (from kaggle->opendatasets) (6.1.0)
Requirement already satisfied: webencodings in /usr/local/lib/python3.10/dist-packages (from bleach->kaggle->opendatasets) (0.5.1)
Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/python3.10/dist-packages (from python-slugify->kaggle->opendatasets) (1.3)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->kaggle->opendatasets) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->kaggle->opendatasets) (3.7)
```

```
pip install fuzzywuzzy
```

```
Requirement already satisfied: fuzzywuzzy in /usr/local/lib/python3.10/dist-packages (0.18.0)
```

Importar las librerías necesarias

```
import opendatasets as od
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics.pairwise import cosine_similarity
```

Descargar el dataset

```
od.download(
    "https://www.kaggle.com/datasets/andrewmvd/spotify-playlists")
```

```
Skipping, found downloaded files in "./spotify-playlists" (use force=True to force download)
```

Lectura del dataset y eliminación de observaciones no utiles

```
#leer el dataset
df = pd.read_csv("spotify-playlists//spotify_dataset.csv",on_bad_lines='skip')

#Renombrar las columnas
df.columns = ['user', 'artist', 'song', 'playlist']

#Eliminar todos aquellos valores NA
df.dropna(inplace=True)

#Eliminar todos los valores duplicados
df.drop_duplicates(inplace=True)

#Eliminar las observaciones que presenten los siguientes valores en la columna de playlist
#Estas observaciones se eliminan porque no son playlist publicas
df.drop(df[df['playlist']=='Starred'].index,inplace=True)
df.drop(df[df['playlist']=='Liked from Radio'].index,inplace=True)
df.drop(df[df['playlist']=='Favoritas de la radio'].index,inplace=True)
df.drop(df[df['playlist']=='New Playlist'].index,inplace=True)
df.drop(df[df['playlist']=='My Shazam Tracks'].index,inplace=True)
df.drop(df[df['playlist']=='All Live Files'].index,inplace=True)
df.drop(df[df['playlist']=='iPhone'].index,inplace=True)


#obtener un Id para cad usuario
user_id = {}
cnt = 1
for user in df['user']:
    if user not in user_id:
        user_id[user] = cnt
        cnt+=1

#Obtener el conteo de las playlist
playlists = {}
for playlist in df['playlist']:
    if playlist in playlists:
      playlists[playlist] += 1
    else:
      playlists[playlist] = 1
```

Generación de columnas útiles

```
#Anadir la dataset las columnas de ID y de reporducciones por playlist
df['User_id'] = df['user'].apply(lambda use: user_id[use])
df['Repros'] = df['playlist'].apply(lambda playlist: playlists[playlist])


```

```
df.head()
```

	user	artist	song	playlist	User_id	Repros
0	9cc0cfd4d7d7885102480dd99e7a90d6	Elvis Costello	(The Angels Wanna Wear My) Red Shoes	HARD ROCK 2010	1	67
1	9cc0cfd4d7d7885102480dd99e7a90d6	Elvis Costello & The Attractions	(What's So Funny 'Bout) Peace, Love And Unders...	HARD ROCK 2010	1	67
2	9cc0cfd4d7d7885102480dd99e7a90d6	Tiffany Page	7 Years Too Late	HARD ROCK 2010	1	67
3	9cc0cfd4d7d7885102480dd99e7a90d6	Elvis Costello & The Attractions	Accidents Will Happen	HARD ROCK 2010	1	67
4	9cc0cfd4d7d7885102480dd99e7a90d6	Elvis Costello	Alison	HARD ROCK 2010	1	67

Generar dataset con el número de reproducciones que tiene playlist

```
#Generar un dataset que solamente contenga el titulo de playlist y su numero de reproducciones
playlist_df = pd.DataFrame.from_dict(playlists,orient='index')
playlist_df.columns = ['Repros']
playlist_df.reset_index(inplace = True)
playlist_df.columns = ['Playlist','Repros']
playlist_df.sort_values(by='Repros',ascending=False,inplace=True)
playlist_df.reset_index(inplace = True,drop=True)
```

```
print(f'Existen {len(playlist_df)} playlists')
```

```
Existen 157313 playlists
```

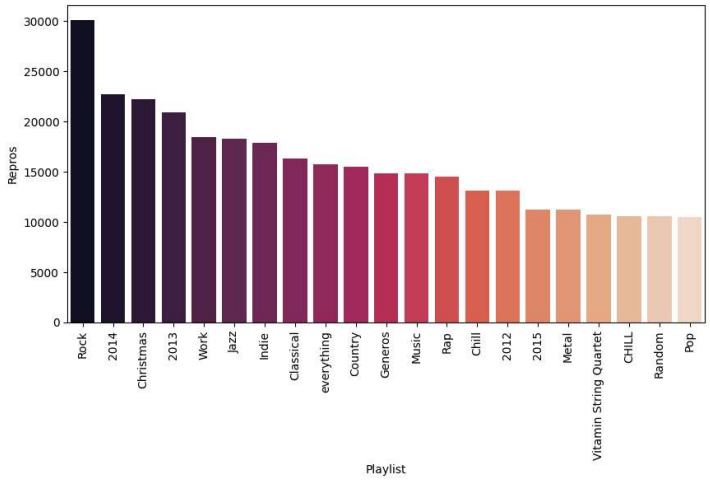
Obtención del top 20 playlist con más reproducciones

```
plt.figure(figsize=(10,5))
sns.barplot(x='Playlist', y='Repros', data=playlist_df.loc[0:20], palette='rocket')
plt.xticks(rotation=90)
plt.show()
```

```
<ipython-input-10-b488ca9cfa02>:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.barplot(x='Playlist', y='Repros', data=playlist_df.loc[0:20], palette='rocket')
```



Obtener solamente las 20 playlist con mas reproducciones en el dataset

```
toMatain = playlist_df.loc[:20]
del playlist_df
```

```
toMatain
```

	Playlist	Repros
0	Rock	30107
1	2014	22674
2	Christmas	22236
3	2013	20870
4	Work	18408
5	Jazz	18266
6	Indie	17858
7	Classical	16328
8	everything	15705
9	Country	15503
10	Generos	14854
11	Music	14843
12	Rap	14500
13	Chill	13113
14	2012	13089
15	2015	11185
16	Metal	11178
17	Vitamin String Quartet	10717
18	CHILL	10544
19	Random	10533
20	Pop	10454

Eliminar todas aquellas observaciones que no posean alguna playlist de las mostradas anteriormente.

```
df = df[df['playlist'].isin(toMatain['Playlist'])]
```

Realizar el mismo proceso que se lleve a cabo con las playlist, pero ahora con los artistas.

```
#Obtener las reproducciones por artista
ReprosArtistas = {}
for artist in df['artist']:
  if artist in ReprosArtistas:
    ReprosArtistas[artist]+=1
  else:
    ReprosArtistas[artist]=1
```

```
artists_df = pd.DataFrame.from_dict(ReprosArtistas,orient='index')
```

```
artists_df.reset_index(inplace=True)
artists_df.columns = ['artist','repros']
artists_df.sort_values(by='repros',ascending=False,inplace=True)
artists_df.reset_index(inplace=True)

toMatain = artists_df.loc[:20]
df = df[df['artist'].isin(toMatain['artist'])]
```

```
df.reset_index(inplace=True,drop=True)
df
```

	user	artist	song	playlist	User_id	Repros
0	50346e4190d1707ebc6b39a95f86927a	Miles Davis	Basin Street Blues	Jazz	14	18266
1	50346e4190d1707ebc6b39a95f86927a	Miles Davis	I Fall in Love Too Easily	Jazz	14	18266
2	50346e4190d1707ebc6b39a95f86927a	Miles Davis	My Funny Valentine	Jazz	14	18266
3	23a17802002bcd5ce26954efcbed2676	Johann Sebastian Bach	Cello Suite No. 1 in G Major, BWV 1007: Prelude	Classical	30	16328
4	23a17802002bcd5ce26954efcbed2676	Wolfgang Amadeus Mozart	Die Zauberflöte (The Magic Flute), K. 620, Act...	Classical	30	16328
...	...	...	...	...	...	...
28583	bf7fd0e14a76a1945d1bd0cce485fd26	Miles Davis	Water on the Pond	Jazz	15066	18266
28584	bf7fd0e14a76a1945d1bd0cce485fd26	Miles Davis	Weirdo	Jazz	15066	18266
28585	bf7fd0e14a76a1945d1bd0cce485fd26	Miles Davis	When Lights Are Low	Jazz	15066	18266
28586	bf7fd0e14a76a1945d1bd0cce485fd26	Miles Davis	Woody 'N You - Live Version	Jazz	15066	18266
28587	bf7fd0e14a76a1945d1bd0cce485fd26	Miles Davis	Yaphet	Jazz	15066	18266

28588 rows × 6 columns

Obtener las matrices dummy tanto de las playlist como de los artistas

```
#Matriz dummy de las playlist
dummyPlaylist = pd.get_dummies(df['playlist'],sparse=True)
dummyPlaylist
```

	2012	2013	2014	2015	CHILL	Chill	Christmas	Classical	Country	Generos	...	Jazz	Metal	Music	Pop	Random	Rap	Rock	Vitamin String Quartet	Work	everything
0	False	False	False	False	False	False	False	False	False	False	...	True	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False	...	True	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	...	True	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	True	False	False	...	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	True	False	False	...	False	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
28583	False	False	False	False	False	False	False	False	False	False	...	True	False	False	False	False	False	False	False	False	False
28584	False	False	False	False	False	False	False	False	False	False	...	True	False	False	False	False	False	False	False	False	False
28585	False	False	False	False	False	False	False	False	False	False	...	True	False	False	False	False	False	False	False	False	False
28586	False	False	False	False	False	False	False	False	False	False	...	True	False	False	False	False	False	False	False	False	False
28587	False	False	False	False	False	False	False	False	False	False	...	True	False	False	False	False	False	False	False	False	False

28588 rows × 21 columns

```
#Matriz dummy de los artistas
dummyArtirst = pd.get_dummies(df['artist'],sparse=True)
dummyArtirst
```

	Brad Paisley	Bruce Springsteen	Ella Fitzgerald	Eminem	Frank Sinatra	Glee Cast	Johann Sebastian Bach	Johnny Hallyday	Kylie Minogue	Led Zeppelin	...	Metallica	Michael Bublé	Miles Davis	Peggy Lee	Queen	Snoop Dogg	Sufjan Stevens	Vitamin String Quartet	Wolfgang Amadeus Mozart	Zac Brown Band
0	False	False	False	False	False	False	False	False	False	False	...	False	False	True	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False	...	False	False	True	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	...	False	False	True	False	False	False	False	False	False	False
3	False	False	False	False	False	False	True	False	False	False	...	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	True	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
28583	False	False	False	False	False	False	False	False	False	False	...	False	False	True	False	False	False	False	False	False	False
28584	False	False	False	False	False	False	False	False	False	False	...	False	False	True	False	False	False	False	False	False	False
28585	False	False	False	False	False	False	False	False	False	False	...	False	False	True	False	False	False	False	False	False	False
28586	False	False	False	False	False	False	False	False	False	False	...	False	False	True	False	False	False	False	False	False	False
28587	False	False	False	False	False	False	False	False	False	False	...	False	False	True	False	False	False	False	False	False	False

28588 rows × 21 columns

```
#Matriz dummy total
dummy = pd.concat([dummyPlaylist,dummyArtirst],axis=1)
del dummyArtirst
del dummyPlaylist

```

```
dummy
```

	2012	2013	2014	2015	CHILL	Chill	Christmas	Classical	Country	Generos	...	Metallica	Michael Bublé	Miles Davis	Peggy Lee	Queen	Snoop Dogg	Sufjan Stevens	Vitamin String Quartet	Wolfgang Amadeus Mozart	Zac Brown Band
0	False	False	False	False	False	False	False	False	False	False	...	False	False	True	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False	...	False	False	True	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	...	False	False	True	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	True	False	False	...	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	True	False	False	...	False	False	False	False	False	False	False	False	True	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
28583	False	False	False	False	False	False	False	False	False	False	...	False	False	True	False	False	False	False	False	False	False
28584	False	False	False	False	False	False	False	False	False	False	...	False	False	True	False	False	False	False	False	False	False
28585	False	False	False	False	False	False	False	False	False	False	...	False	False	True	False	False	False	False	False	False	False
28586	False	False	False	False	False	False	False	False	False	False	...	False	False	True	False	False	False	False	False	False	False
28587	False	False	False	False	False	False	False	False	False	False	...	False	False	True	False	False	False	False	False	False	False

28588 rows × 42 columns

```
#Encontrar la matriz de similaridad
cosine_sim = cosine_similarity(dummy, dummy)
print(f"Las dimensiones de similaridad coseno de las caracteristicas de nuestra matriz de similitud son: {cosine_sim.shape}")
```

```
Las dimensiones de similaridad coseno de las caracteristicas de nuestra matriz de similitud son: (28588, 28588)
```

```
from fuzzywuzzy import process

song_idx = dict(zip(df['song'], list(df.index)))
def encuentra_cancion(title):
    all_titles = df['song'].tolist()
    closest_match = process.extractOne(title,all_titles)
    return closest_match[0]

def obtener_recomendaciones_basadas_contenido(title_string, numero_recomendaciones=10):
    title = encuentra_cancion(title_string)
    idx = song_idx[title]
    sim_scores = list(enumerate(cosine_sim[idx]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    sim_scores = sim_scores[1:(numero_recomendaciones+1)]
    similar_movies = [i[0] for i in sim_scores]
    print(f"Las peliculas recomendadas con base en la pelicula {title} son:")
    print(df['song'].iloc[similar_movies])
```

```
/usr/local/lib/python3.10/dist-packages/fuzzywuzzy/fuzz.py:11: UserWarning: Using slow pure-python SequenceMatcher. Install python-Levenshtein to remove this warning
  warnings.warn('Using slow pure-python SequenceMatcher. Install python-Levenshtein to remove this warning')
```

```
obtener_recomendaciones_basadas_contenido('Basin Street Blues',10)
```

```
Las peliculas recomendadas con base en la pelicula Basin Street Blues son:
1                        I Fall in Love Too Easily
2                              My Funny Valentine
576                                    All Blues
582                                 Blue in Green
587                               Flamenco Sketches
588                              Freddie Freeloader
618                                      So What
814                                       Enigma
815                                         Feio
820    It Never Entered My Mind (Higher & Higher)
Name: song, dtype: object
```