Proyecto Final de Machine Learning

Sistema de Recomendación

Addison Amin Reyes Cedano, 2021-2026

Mi proyecto es el de un sistema de recomendación el cual dependiendo de las valoraciones del usuario a los anime que ya ha visto crea n recomendaciones al usuario de animes para ver, lo primero que hice fue buscar un dataset que cumpliera los requisitos del proyecto y que satisficiera mis necesidades. Este es el dataset que encontré:

https://www.kaggle.com/datasets/CooperUnion/anime-recommendations-database?select=rating.csv

Lo primero que realice fue crear el entorno virtual e importar todas las librerías que utilizare en el proyecto, aquí todas las librerías que utilice:

```
1  from sklearn.model_selection import train_test_split
2  from sklearn.preprocessing import LabelEncoder
3  from dataprep.eda import create_report
4  import matplotlib.pyplot as plt
5  import tensorflow as tf
6  import pandas as pd
7  import numpy as np
✓ 0.0s
```

Una vez importadas las librerías leo los dos archivos '.csv' que son "rating.csv" y " anime.csv"

```
1  ratings = pd.read_csv("rating.csv")
2  items = pd.read_csv("anime.csv")
✓ 1.4s
```

Ya con los dos archivos leídos inicia la preparación y preprocesamiento de la data, en donde junto los dos DataFrame en uno solo en donde tengo los usuarios, los id de los animes, la puntuación que le han dado al anime que han visto y el nombre del anime aparte del id, con este DataFrame completo hago ingeniería de características eliminando datos innecesarios, ajustando algunas columnas, revisando si hay filas vacías, etc..

```
1  ratings.head(6)
```

✓ 0.0s

	user_id	anime_id	rating
0	1	20	-1
1	1	24	-1
2	1	79	-1
3	1	226	-1
4	1	241	-1
5	1	355	-1

```
1  items.head(6)
```

✓ 0.0s Python Python

	anime_id	name	genre	type	episodes	rating	members
0	32281	Kimi no Na wa.	Drama, Romance, School, Supernatural	Movie	1	9.37	200630
1	5114	Fullmetal Alchemist: Brotherhood	Action, Adventure, Drama, Fantasy, Magic, Mili...	TV	64	9.26	793665
2	28977	Gintama°	Action, Comedy, Historical, Parody, Samurai, S...	TV	51	9.25	114262
3	9253	Steins;Gate	Sci-Fi, Thriller	TV	24	9.17	673572
4	9969	Gintama'	Action, Comedy, Historical, Parody, Samurai, S...	TV	51	9.16	151266
5	32935	Haikyuu!!: Karasuno Koukou VS Shiratorizawa Ga...	Comedy, Drama, School, Shounen, Sports	TV	10	9.15	93351

```
1  items_titles = items[["anime_id", "name"]]
2  items_titles.head(6)
```

✓ 0.0s

	anime_id	name
0	32281	Kimi no Na wa.
1	5114	Fullmetal Alchemist: Brotherhood
2	28977	Gintama°
3	9253	Steins;Gate
4	9969	Gintama'
5	32935	Haikyuu!!: Karasuno Koukou VS Shiratorizawa Ga...

```
1  items_ratings = ratings.merge(items_titles, on="anime_id")
2  items_ratings.head(6)
```

✓ 1.0s

	user_id	anime_id	rating	name
0	1	20	-1	Naruto
1	3	20	8	Naruto
2	5	20	6	Naruto
3	6	20	-1	Naruto
4	10	20	-1	Naruto
5	21	20	8	Naruto

```
1  items_ratings.rename(columns={'anime_id': 'item_id'}, inplace = True)
```

✓ 0.0s

```
1  items_titles[items_titles["name"] == "Naruto"]
```

✓ 0.0s

	anime_id	name
841	20	Naruto

```
1  items_ratings.info()
```

✓ 0.0s

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 7813727 entries, 0 to 7813726
Data columns (total 4 columns):
 #   Column   Dtype
---  ------   -----
 0   user_id  int64
 1   item_id  int64
 2   rating   int64
 3   name     object
dtypes: int64(3), object(1)
memory usage: 298.1+ MB
```

```
1  items_ratings.isna().any()
```
✓ 0.9s

```
user_id    False
item_id    False
rating     False
name       False
dtype: bool
```

```
1  items_ratings.loc[items_ratings['rating'] == -1, 'rating'] = 0
```
✓ 0.1s

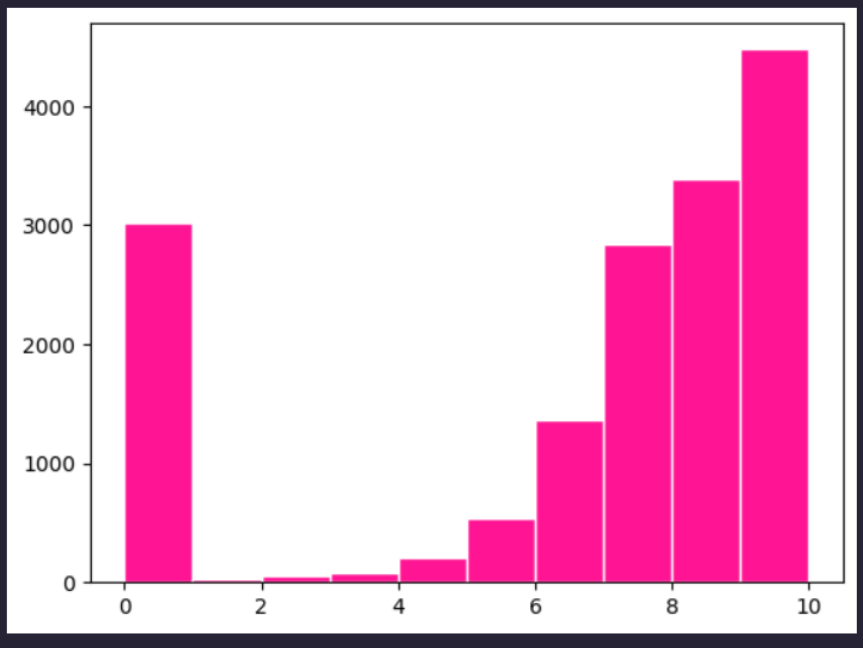
```
1  items_ratings = items_ratings.sample(16000)
```
✓ 0.5s

Hice el análisis descriptivo de los datos con dataprep, aquí el link y una captura del histograma a los 'ratings':

```
1  report = create_report(items_ratings)
2  report.save("Reporte")
```
✓ 2.3s Python

```
c:\Users\Addison\Documents\GitHub\Anime-System-Recomendations\venv\lib\site-packages\dask\core.py:119: RuntimeWarning: inva
  return func(*(_execute_task(a, cache) for a in args))
c:\Users\Addison\Documents\GitHub\Anime-System-Recomendations\venv\lib\site-packages\dataprep\eda\distribution\render.py:27
  df = df.append(pd.DataFrame({col: [nrows - npresent]}, index=["Others"]))
Report has been saved to Reporte.html!
```

```
1  plt.hist(items_ratings['rating'], color="deeppink", edgecolor="white")
2  plt.show()
```
✓ 0.1s



Después de analizar la data paso la columna de 'user_id' y 'item_id' por un LabelEncoder para luego hacer un train test Split con Scikit learn implemento un modelo de recomendación utilizando el método de factorización de matrices con redes neuronales para predecir la calificación que un usuario podrá asignar a un anime que aún no ha visto.

```
encoder = LabelEncoder()
```
✓ 0.0s

```
encoder.fit(items_ratings['user_id'])
items_ratings.loc[:, 'user_id'] = sorted(encoder.transform(items_ratings['user_id']))

encoder.fit(items_ratings['item_id'])
items_ratings.loc[:, 'item_id'] = encoder.transform(items_ratings['item_id'])
```
✓ 0.1s

```
df_train, df_val = train_test_split(items_ratings, test_size=0.2, random_state=42)
```

Primero se crean dos matrices de ceros, train_matrix y test_matrix, con un número de filas igual al número de usuarios y un número de columnas igual al número de películas únicas en el conjunto de datos. La matriz train_matrix se inicializa con las calificaciones de entrenamiento del conjunto de datos de entrenamiento, y la matriz test_matrix se inicializa con las calificaciones de prueba del conjunto de datos de prueba.

```
n_users = len(items_ratings.user_id.unique())
n_items = len(items_ratings.item_id.unique())

train_matrix = np.zeros((n_users, n_items))

for row in df_train.itertuples():
    train_matrix[row[1]-1, row[2]-1] = row[3]
```
✓ 0.1s

Luego, se definen dos capas de entrada para el modelo, input_users e input_items, que representan los identificadores de los usuarios y de las películas, respectivamente. Cada una de estas capas se conecta a una capa de embedding, embedding_users y embedding_items, que transforma los identificadores de los usuarios y las películas en vectores densos de baja dimensión. Estos vectores se aplanan en capas flatten_users y flatten_items, respectivamente.

```python
input_users = tf.keras.layers.Input(shape=[1])
embedding_users = tf.keras.layers.Embedding(n_users, 100)(input_users)
flatten_users = tf.keras.layers.Flatten()(embedding_users)

input_items = tf.keras.layers.Input(shape=[1])
embedding_items = tf.keras.layers.Embedding(n_items, 100)(input_items)
flatten_items = tf.keras.layers.Flatten()(embedding_items)
```

✓ 0.1s

Después, se concatenan los vectores de usuario y película en una única capa de entrada concat, que se alimenta a dos capas densas con activación relu y una tasa de dropout del 20% en cada capa, dense1 y dense2. Esto ayuda a evitar el sobreajuste y mejorar la generalización del modelo.

```python
concat = tf.keras.layers.Concatenate()([flatten_users, flatten_items])

dense1 = tf.keras.layers.Dense(128, activation='relu')(concat)
dropout1 = tf.keras.layers.Dropout(0.2)(dense1)

dense2 = tf.keras.layers.Dense(64, activation='relu')(dropout1)
dropout2 = tf.keras.layers.Dropout(0.2)(dense2)
```

✓ 0.0s

Finalmente, se utiliza una capa densa sin activación en la salida, output, que predice la calificación de la película para el usuario. El modelo se compila utilizando el error cuadrático medio (MSE) como función de pérdida y el optimizador Adam.

```python
output = tf.keras.layers.Dense(1)(dropout2)

model = tf.keras.Model(inputs=[input_users, input_items], outputs=output)
model.compile(loss='mse', optimizer='adam')
```

✓ 0.0s

Una vez tenido el modelo entrenado utilizo la función .fit del modelo para ajustar los pesos dl modelo para minimizar la pérdida de un anime a otro, por ultimo evaluó el modelo con los datos que me había dado el train_split_train.

```python
model.fit([df_train.user_id, df_train.item_id], df_train.rating, epochs=10, verbose=1, validation_split=0.2, batch_size=70)
```

✓ 28.6s

```
Epoch 1/10
147/147 [==============================] - 3s 20ms/step - loss: 0.9447 - val_loss: 16.2858
Epoch 2/10
147/147 [==============================] - 3s 20ms/step - loss: 0.8780 - val_loss: 16.0329
Epoch 3/10
147/147 [==============================] - 3s 19ms/step - loss: 0.8570 - val_loss: 16.2854
Epoch 4/10
147/147 [==============================] - 3s 19ms/step - loss: 0.8409 - val_loss: 16.2366
```

```
test_loss = model.evaluate([df_val.user_id, df_val.item_id], df_val.rating)
print('Test loss:', test_loss)
```

✓ 0.2s

```
100/100 [==============================] - 0s 1ms/step - loss: 15.7229
Test loss: 15.722870826721191
```

Por ultimo en el proyecto creo una función la cual es la que se encarga de devolver las recomendaciones al usuario eliminando los animes visto por el usuario y luego prediciendo de los animes no vistos cuales tienen más probabilidad de ser una mejor recomendación para luego devolver las n recomendaciones en un DataFrame.

```
def recommend_items(user_id, model, n=2):
    #Se obtiene items que ya ha visto el usuario
    seen_movies = items_ratings[items_ratings['user_id'] == user_id]['item_id'].tolist()

    #Se obtiene items que el usuario aún no ha visto
    item_ids = list(set(items_ratings['item_id']) - set(seen_movies))

    #Se predicen las puntuaciones para las items no vistas
    ratings_pred = model.predict([np.array([user_id]*len(item_ids)), np.array(item_ids)])
    ratings_pred = np.array([rating[0] for rating in ratings_pred])

    #Obtener los mejores n items recomendadas
    top_item_ids = (-ratings_pred).argsort()[:n]
    top_item_ids = [item_ids[i] for i in top_item_ids]
    top_item_ids += [0]*(n - len(top_item_ids))
    top_ratings_pred = ratings_pred[top_item_ids]

    #Se crea un DataFrame con los mejores n items
    items_pred = pd.DataFrame(columns=['anime_id', 'name', 'genre', 'type', 'episodes', 'rating', 'members'])

    i = 0
    for ids in top_item_ids:
        items_pred.loc[i] = items.iloc[ids]
        i += 1

    items_pred['predicted_rating'] = top_ratings_pred

    return items_pred
```

✓ 0.0s

Recomendaciones

```
user_id = 16
recommendations = recommend_items(user_id, model, n=3)
recommendations
```

✓ 0.4s

```
104/104 [==============================] - 0s 1ms/step
```

	anime_id	name	genre	type	episodes	rating	members	predicted_rating
0	6324	Omamori Himari	Action, Comedy, Demons, Ecchi, Harem, Romance,...	TV	12	7.19	119572	-0.018156
1	4186	Chrome Shelled Regios	Action, Adventure, Fantasy, School, Sci-Fi	TV	24	7.57	103615	6.951766
2	1434	Lupin III: Harimao no Zaihou wo Oe!!	Action, Adventure, Comedy, Shounen	Special	1	7.16	3057	0.009716

```
user_id = 69
recommendations = recommend_items(user_id, model, n=3)
recommendations
```

✓ 0.3s

```
104/104 [==============================] - 0s 1ms/step
```

	anime_id	name	genre	type	episodes	rating	members	predicted_rating
0	6024	Chi's Sweet Home: Atarashii Ouchi	Comedy, Kids, Slice of Life	TV	104	7.9	16670	9.494140
1	6324	Omamori Himari	Action, Comedy, Demons, Ecchi, Harem, Romance,...	TV	12	7.19	119572	6.650553
2	3483	Maison Ikkoku: Utsuriyuku Kisetsu no Naka de	Comedy, Drama, Romance	OVA	1	7.15	1458	3.755569