Movie Recommendation System

Angad Dhillon, Camille Velarde, Chandler McLaren, Chisom Ozoemena

Why recommendation systems are important ?

- → Enhanced User Experience (by personalizing content, products or services)
- → Data-Driven Insights (by analyzing user interactions, preferences and feedback)
- → Strategic Decision Making (by understanding user preference, behaviour patterns etc.)

Movie Recommendation

The purpose of this model is to provide personalized movie suggestions to users based on their viewing history, preferences, and ratings.

EDA / Dataset

Data was collected from Kaggle.

The data was created by 138,493 users between January 09, 1995 and March 31, 2015.

We used features like user ID, title, genres, rating to create our model

EDA / Data Cleaning

	movieId	title	poster_path	cleaned_genres	userId	rating	timestamp
0	949	Heat	/zMyfPUelumio3tiDKPffaUpsQTD.jpg	Action|Crime|Drama|Thriller	23	3.5	1148721092
1	949	Heat	/zMyfPUelumio3tiDKPffaUpsQTD.jpg	Action|Crime|Drama|Thriller	102	4.0	956598942
2	949	Heat	/zMyfPUelumio3tiDKPffaUpsQTD.jpg	Action|Crime|Drama|Thriller	232	2.0	955092697
3	949	Heat	/zMyfPUelumio3tiDKPffaUpsQTD.jpg	Action|Crime|Drama|Thriller	242	5.0	956688825
4	949	Heat	/zMyfPUelumio3tiDKPffaUpsQTD.jpg	Action|Crime|Drama|Thriller	263	3.0	1117846575
...	...	...	...	...	...	...	...
44989	64197	Travelling with Pets	/fZlvSGtAVfnXkJCY3Gnev05rUFk.jpg	Romance|Drama	73	4.0	1441513491
44990	64197	Travelling with Pets	/fZlvSGtAVfnXkJCY3Gnev05rUFk.jpg	Romance|Drama	544	5.0	1435789819
44991	64197	Travelling with Pets	/fZlvSGtAVfnXkJCY3Gnev05rUFk.jpg	Romance|Drama	648	3.5	1241951834
44992	98604	Cinderella	/cBFOyxe5HzIOljJhipKQuslZsuV.jpg	Comedy|Romance	352	4.0	1420521986
44993	49280	The One-Man Band	/ZLOgl7KjtWby1NEg2pjU2Id60W.jpg	Fantasy|Action|Thriller	187	5.0	1228072108

44994 rows × 7 columns

- Used python to clean columns such as cleaned_genre, userId, and rating
- Merged the rating_small.csv and movie_metadata.csv

Database Storage

MongoDB

Because of it's schema flexibility.

```
# assign the database to a variable name
db = mongo['movies_database']
```

```
# review the collections in our new database
print(db.list_collection_names())
```

```
['movies_list']
```

```
# assign each collection to a variable
movies_list = db['movies_list']
```

```
#Display no of documents in each collection
print('the number of documents in movies list are: ',movies_list.count_documents({}))
```

```
the number of documents in movies list are:  43000
```

Web Application

- Html, css, js
- Features:
    - text input bar, search button, load spinner
- Output visuals:
    - top 10 movie recommendations with movie poster, genre, and overview
- Demo

M.L. Models

- ➔ KNN (K-Nearest Neighbours)
- ➔ TensorFlow Collaborative Filtering

KNN

- ➔ It initializes an empty list neighbour_ids to store the IDs of similar movies.
- ➔ It retrieves the index of the movie in the matrix X using a mapping dictionary movie_mapper.
- ➔ It increments k by 1 (because when finding the nearest neighbors, it includes the movie itself).
- ➔ It sets up a k nearest neighbors model (kNN) using the brute-force algorithm and the specified distance metric.
- ➔ It fits the kNN model with the matrix X.
- ➔ It finds the k nearest neighbors of the movie using the kNN model.
- ➔ It iterates over the indices of the nearest neighbors.
- ➔ For each index, it retrieves the corresponding movie ID using a reverse mapping dictionary movie_inv_mapper.
- ➔ It appends the retrieved movie ID to the list neighbour_ids.

```python
"""
Find similar movies using KNN
"""
def find_similar_movies(movie_id, X, k, metric='cosine', show_distance=False):

    neighbour_ids = []

    movie_ind = movie_mapper[movie_id]
    movie_vec = X[movie_ind]
    k+=1
    kNN = NearestNeighbors(n_neighbors=k, algorithm="brute", metric=metric)
    kNN.fit(X)
    movie_vec = movie_vec.reshape(1,-1)
    neighbour = kNN.kneighbors(movie_vec, return_distance=show_distance)
    for i in range(0,k):
        n = neighbour.item(i)
        neighbour_ids.append(movie_inv_mapper[n])
    neighbour_ids.pop(0)
    return neighbour_ids


movie_titles = dict(zip(movies_df['movieId'], movies_df['title']))

movie_id = 3

similar_ids = find_similar_movies(movie_id, X, k=10)
movie_title = movie_titles[movie_id]

print(f"Since you watched {movie_title}")
for i in similar_ids:
    print(movie_titles[i])
```

Collaborative Filtering

- (_init_):
  - Initializes the CFModel with parameters n_users, m_items, and k_factors.
  - It creates two layers for user (self.P) and item (self.Q) embeddings using tf.keras.Sequential.
  - It defines each embedding layer followed by a Reshape layer.
- Call method:
  - This method defines the forward pass of the model.
  - Takes inputs, of user_id and item_id.
  - Retrieves the embeddings for the user and item using the P and Q layers.
- Rate method:
  - This method predicts the rating for a given user_id and item_id.
  - Calls the call method internally to get the prediction.
  - Returns the prediction as an array.

```python
import tensorflow as tf
class CFModel(tf.keras.Model):
    def __init__(self, n_users, m_items, k_factors):
        super(CFModel, self).__init__()

        self.P = tf.keras.Sequential([
            tf.keras.layers.Embedding(n_users, k_factors, input_length=1),
            tf.keras.layers.Reshape((k_factors,))
        ])

        self.Q = tf.keras.Sequential([
            tf.keras.layers.Embedding(m_items, k_factors, input_length=1),
            tf.keras.layers.Reshape((k_factors,))
        ])

    def call(self, inputs):
        user_id, item_id = inputs
        user_latent = self.P(user_id)
        item_latent = self.Q(item_id)
        return tf.reduce_sum(tf.multiply(user_latent, item_latent), axis=1)

    def rate(self, user_id, item_id):
        user_embedding = self.P(tf.constant([user_id]))
        item_embedding = self.Q(tf.constant([item_id]))
        prediction = tf.reduce_sum(tf.multiply(user_embedding, item_embedding), axis=1)[0]
        return prediction.numpy()
```

Challenges

- Difficult to recommend movies based on ratings because we do not know why that user liked that specific movie.
- Mood, emotions of user plays key role in what they might want to watch, hard to get data for that.
- Difficult to evaluate model because user may or may not watch movies from recommendations. (User survey might help in improving model)

Challenges

- KNN Model: The main challenge is its scalability with large datasets.
  - Cannot evaluate accuracy since it is an unsupervised model
  - Does not consider demographics of user from user_table.csv
- Custom TensorFlow Model: This model require significant computational resources for training making it resource intensive.
  - Training phase took too long - had to reduce epochs to 1 for some of us

Conclusion

- In conclusion, the development of movie recommendation system has shown the power of personalized content.
- By exploring two different models we navigated the challenges of unsupervised learning. The use of the KNN model allowed the user interaction to be faster and more simple. Therefore being user friendly.
- Improvements:
  - Using cloud to load data faster and train larger datasets
    - Model would be saved on a cloud instead of on a local database

Thank YOU

Q & A

Citations

https://www.geeksforgeeks.org/recommendation-system-in-python/

https://www.kaggle.com/datasets/grouplens/movielens-20m-dataset

https://github.com/khanhnamle1994/movielens/tree/master

https://cssloaders.github.io/