Movie Recommender Systems

Sohaib Jawad and Victor Samsonov

College of Computing
Illinois Institute of Technology

December 2, 2022

Presentation Structure

- Introduction and Motivation
- MovieLens Dataset EDA
- Content-based Filtering
- Collaborative Filtering
- DLRM - Deep Learning Recommendation Model

Motivation and Problem Description

Definition 1.1

Recommendation System:"is an AI algorithm, usually associated with machine learning, that uses Big Data to suggest or recommend additional products to consumers" (Nvidia).

We seek to implement a movie-oriented Recommendation System such that it can make adequate suggestions based on the interests of a user.

Recommendation systems are very present in our daily lives (Online shopping, restaurants, video apps, etc.).These ML algorithms use data based on past user behavior in order to make recommendations

Modern Recommendation Systems

Modern Recommendation Systems are very sophisticated and a lot of resources are being invested by large tech companies to improve these models.

Types of recommendation systems:

- Content-Based Filtering (item features to provide recommendations)
- Collaborative Filtering (similarities between users and items to provide recommendations)
- Hybird Filtering

MovieLens EDA

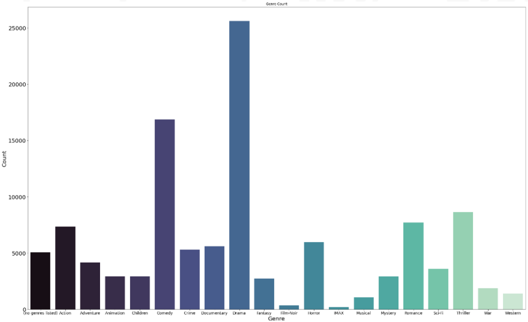
The MovieLens 1M dataset contains over 1 million movie ratings and tagging activities since the 1995.

Structure:

- tag.csv
- movie.csv
- rating.csv
- link.csv

MovieLens EDA

After preprocessing the genres and representing our genres into a vectorized form, we can visualize the count for each genre.




```python
movies.genres
out = []
for genre in movies.genres:
    out.append(genre.split('|'))
```

```python
from sklearn.preprocessing import MultiLabelBinarizer
binarizer = MultiLabelBinarizer()
binarized_genres = binarizer.fit_transform(np.array(out))
GENRES = binarizer.classes_
binarized_genres
```

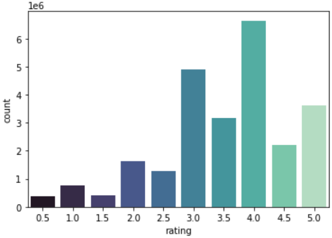
```
array([[0, 0, 1, ..., 0, 0, 0],
       [0, 0, 1, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
       [1, 0, 0, ..., 0, 0, 0],
       [0, 1, 1, ..., 0, 0, 0]])
```

```python
movies[GENRES] = binarized_genres
movies.drop(columns='genres', inplace=True)
movies
```

MovieLens

Ratings

```
The mean rating is: 3.533854451353085
The mode rating is: 4.0
```




```
sns.countplot(data=ratings, x='rating', palette='mako')
print(f"The mean rating is: {ratings.rating.mean()}")
print(f"The mode rating is: {ratings.rating.mode()[0]}")
```

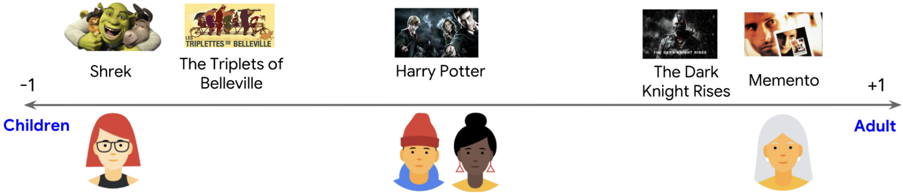
Naive Content Based Filtering Intuition

- Uses item features to recommend other items similar to what the user likes.
- Explicit and Implicit
- Choose a similarity metrics like Cosine similarity or dot product
- Set up a system to score each candidate

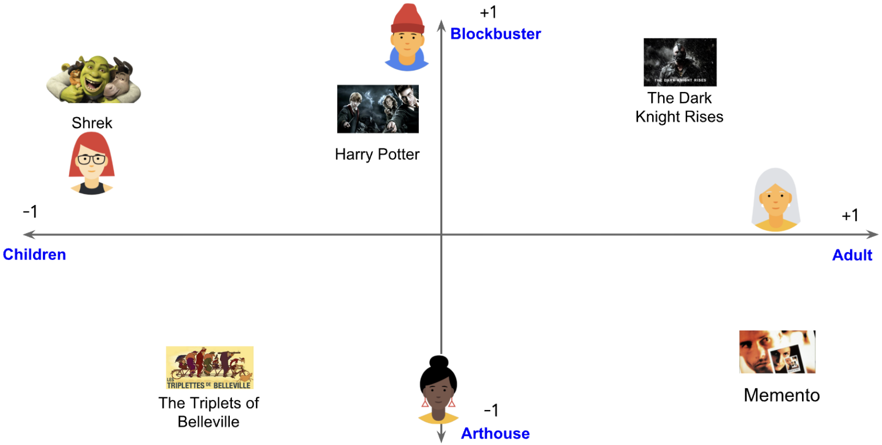
Collaborative Filtering Intuition

- Addresses limitations of content-based approach.
- Also checks similar user profiles
- 'Cold start' problem
- Don't need to set up a scoring system

Collaborative Filtering Intuition



Collaborative Filtering Intuition



Collaborative Filtering implementation

Naive implementation:

```python
def recommendation_score(movie_id, rating_thresh=3.5, percent_thresh=0.1, n_movies=5):
    '''
    description - Function that returns a recommendation score (higher score -> better recommendation)
    Inputs:
        movie_id: int
            Id of the movie we will use to find recommendations.
        rating_thresh:
            used to filter movies with a rating threshold greater than the value.
        percent_thresh:
            used to filter movies with a rating threshold greater than the value.
    '''
    # Subset of users that like the same movie
    user_subset = ratings[(ratings["movieId"] == movie_id) & (ratings["rating"] >= rating_thresh)]["userId"].unique()
    # Ratings of other movies that similar users
    user_subset_recommendations = ratings[(ratings["userId"].isin(user_subset)) & (ratings['rating'] > rating_thresh)]['movieId']

    # Percentage of users that liked each movie and filter based on percent_thresh
    user_subset_recommendations = user_subset_recommendations.value_counts() / len(user_subset)
    user_subset_recommendations = user_subset_recommendations[user_subset_recommendations > percent_thresh]

    # Find users that have rated the movies in our current subset
    all_users = ratings[(ratings["movieId"].isin(user_subset_recommendations.index)) & (ratings["rating"] > rating_thresh)]
    # Percentage of all users that liked the movie
    all_user_recommendations = all_users["movieId"].value_counts() / len(all_users["userId"].unique())

    # Calculate a recommendation score based on similar users and overall users (we aren't interested in obvious relationships)
    rec_percentages = pd.concat([user_subset_recommendations, all_user_recommendations], axis=1)
    rec_percentages.columns = ["subset_users", "avg_users"]
    rec_percentages['recommendation_score'] = rec_percentages["subset_users"] / rec_percentages["avg_users"]
    rec_percentages = rec_percentages.sort_values("recommendation_score", ascending=False)

    return rec_percentages.head(n_movies).merge(movies, left_index=True, right_on="movieId")
```

```python
recommendation_score(2, 4.0)
```

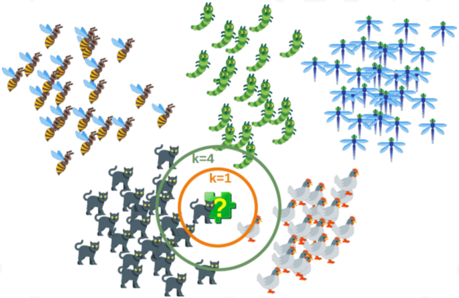
	subset_users	avg_users	recommendation_score	movieId	title	genres
1	0.322257	0.017706	18.200580	2	Jumanji (1995)	Adventure|Children|Fantasy
578	0.113461	0.020614	5.504137	586	Home Alone (1990)	Children|Comedy
495	0.168439	0.030844	5.460957	500	Mrs. Doubtfire (1993)	Comedy|Drama
362	0.117327	0.022579	5.196343	367	Mask, The (1994)	Action|Comedy|Crime|Fantasy
721	0.110561	0.022088	5.005570	736	Twister (1996)	Action|Adventure|Romance|Thriller

Code overview:

- Find users that like the same movies and use this information to find movies above a rating threshold.
- Obtain percentage of users that liked each movie and filter based on a relevant threshold value
- Calculate a recommendation score based on "similar" users and all the users (we are interested in higher scores).

K-Nearest Neighbors

KNN is a supervised learning algorithm that can be used for both regression and classification. Classes are predicted by calculating the distance between the test and training points and then select K number of points that are closest to our test



KNN Implementation

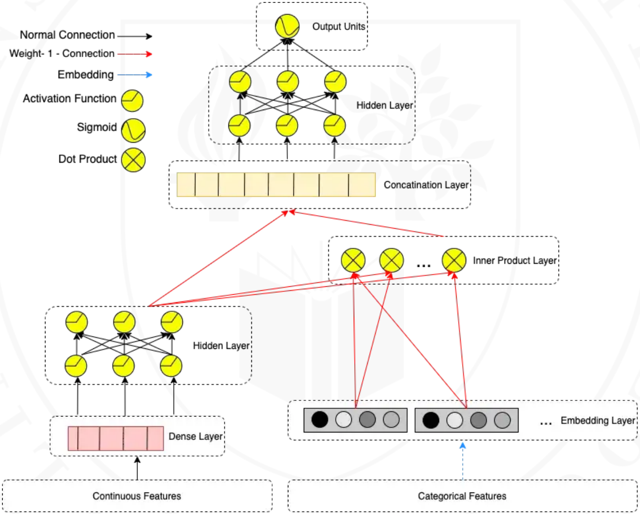
```
from sklearn.neighbors import NearestNeighbors
model = NearestNeighbors(metric='cosine', n_neighbors=20)
model.fit(taste_space_sparse)
```

```
def recommender(movie_name, data, n):
    # Allows us to match strings
    idx = process.extractOne(movie_name, movies['title'])[2]
    print(movies['title'][idx])
    distance, indices = model.kneighbors(data[idx], n_neighbors=n)
    print(movies['title'][indices[0]])
```

```
recommender('jurassic park', taste_space_sparse, 6)
```

```
Jurassic Park (1993)
418                    Jurassic Park (1993)
507    Terminator 2: Judgment Day (1991)
314                     Forrest Gump (1994)
97                        Braveheart (1995)
398                   Fugitive, The (1993)
334                           Speed (1994)
Name: title, dtype: object
```

DLRM - Structure



DLRM architecture

```
Model: "DL_Recommender"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
userId (InputLayer)             [(None, 1)]          0
__________________________________________________________________________________________________
movieId (InputLayer)            [(None, 1)]          0
__________________________________________________________________________________________________
userId_Embedding (Embedding)    (None, 1, 5)         3055        userId[0][0]
__________________________________________________________________________________________________
movieId_Embedding (Embedding)   (None, 1, 5)         968050      movieId[0][0]
__________________________________________________________________________________________________
concatenate_3 (Concatenate)     (None, 1, 10)        0           userId_Embeddi
ng[0][0]
                                                                 movieId_Embedd
ing[0][0]
__________________________________________________________________________________________________
relu1 (Dense)                   (None, 1, 32)        352         concatenate_3
[0][0]
__________________________________________________________________________________________________
relu2 (Dense)                   (None, 1, 16)        528         relu1[0][0]
__________________________________________________________________________________________________
output (Dense)                  (None, 1, 1)         17          relu2[0][0]
==================================================================================================
Total params: 972,002
Trainable params: 972,002
Non-trainable params: 0
__________________________________________________________________________________________________
```

Training

We split data into training and testing (20

Training parameters:

- Adam as optimizer
- lr of 0.0001
- MSE loss
- MAE metric

Training

Loss and MAE,

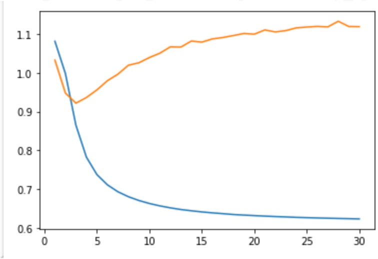


Figure 1: loss:.6207, val loss: 1.0882

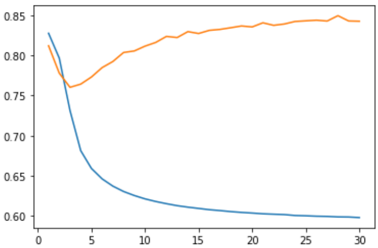


Figure 2: MAE:.5962, val MAE: 0.8289

Afterwards, we train the model on the entire training set for only 4 epochs and we obtain a **MAE of 0.681 on the test set**.

Thank you!