IMDB Movie Recommender System

Introduction & Overview

- To form the basis af this application, 9 basic genres were chosen. This was done by extracting and analysing data from websites.
- The **9 base genres** spanned over the *majority* of the movies.
- Additionally, a set of 500 organic users has also been generated. Certain statistical measures have been taken to ensure the *random* data generated is representative of real world users.

Datasets

The file `data_final.json` houses the 173 movie database that we'll be using for the recommender system. These were fetched using an open API

Loading the database

There is a data dump in the `data/` folder that can be used to directly populate your database with the required collections.

```
mongorestore --db <db_name> db_dump/test
```

Loading movies

To load the database, just run the `mongoimport` command to read form the JSON.

```
mongoimport --db <db_name> --collection movies --file data_final.json --jsonArray --drop
```

Loading the users

- The users have been precreated and stored in the `users.json` file.
- Additionally, you can use the `gen_users.py` to generate new users as well

```
./gen_users.py <no_users>
mongoimport --db <db_name> --collection users --file users.json --jsonArray
```

Application - Features & Overview

- The application is a `mongoDB` backed, Flask application that uses `jinja` for its frontend rendering.
- Users and movies have already been seeded and can be imported to have a ready-to-go system.

- Login based enforcement for movie information and user related pages.
- Internal likes are only counted when the user creates an account, and are registered using simple, intuitive sliders for each movie.
- Given the scarcity of the data, every user has a `ratings` key that contains a list of the all the movies he has rated with their respective ratings. This can be used (supplemented by additional ratings and movie information) to personalize recommendations.
- The application uses CSRF protection for added security against XSS attacks and stores password in a salted hash, ensuring security and integrity of the data.

Analysis & Data selection

Movie data selection

- $2^9 - 1(255)$ subsets were produced from the genre set by permuting. This covers all possible occurences of *key* genre sets.
- Each subset was looked up on imdb_top250 and the data for the movies obtained was dumped onto `data_final.json`. Various movies were selected from the categories on the basis of their statistical share hold across the genres.
- Thus 176 total movies were obtained which:
  - Represented the majority of the blockbuster movies.
  - Spanned over the entire spectrum of base genre permutations.
- The entire dataset was injected into the MongoDB with a special `onehot` vector which represented the frequency of each base genre occurence for the movie. This allows for a quantized representation of how **strong** the influence of the movie is in our dataset.

User data generation

Genre Population distribution

- Using the information obtained from, user distributions are calculated on the basis of gender(Male or Female), then age (12-24,25-35) with respect to the different genres.

12 - 24	12 - 14	14 - 24	24-34	Ratio	Category	Male	Female
15	22	8	15	1	Animation	75	65
33.5	35	32	30	1.1166	Comedy	90	91
8.5	5	12	11	0.7727	Crime	79	84
4	0.5	7.5	5.5	0.7272	Horror	57	47
40	40	40	40	1	Action	90	86

18	18	18	18	1	SciFi	76	62
23.5	20	27	27	0.8703	Drama	80	89
10.5	8	13	11	0.9545	Romance	55	77

- These probabilites were then used to distribute the various basic genres over the 500 users.
- Next, to further prepopulate the users, 12 **total** genres were picked up from the IMDB dataset and an n x n matrix was formed denoting the probabilistic distribution of the genres with one another. Entry $A[i][j]$ represents the occurences of the j given base tag i. Entry (i, i) represents the probability of this tag occuring alone. The genre distribution was augmented with these probabilities and user genre sets were updated to include these as well.

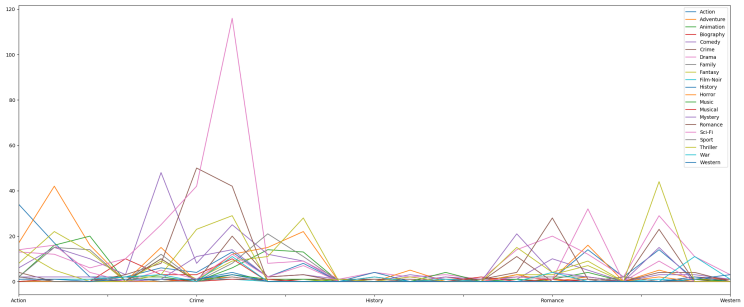

Figure 1: genre_dist

- The diagonal, thus, represents the occurences of the genre occuring alone.
- Due to the constraints and distributions used to generate the users, it is fair to say that they are a close-to-real representation of organic user data, something that is **essential** for consistent, accurate results.

Natural rating distribution

- Once the genres have been distributed, every user is then allocated a choice of $K \pm \sigma$ of favoured movies and upto 33% of like quantity-dislike titles are also chosen (*1 in 3 movies that a user watches might be rated badly*)
- After chosen per user genres, they are augmented by adding other non base genres and movies that contain said genres are isolated and sampled according to the distribution. (*liked movies from liked indices and disliked movies from disliked indices*)
- Finally, a normalized average rating is obtained given metacritic, IMDB and Rotten tomatoes. This normalized rating is then deviated according to whether the user has a positive liking or negative.

- This data is then dumped into a `json` to be imported into the database.

Note: In case the user database size was substantial ($> 10^7$), even a random genre sampling would have been okay. But since we're looking at a relatively small system, having this "non-random" seed *data will benefit the recommendation choice and give more organic results due to the natural nature of the "dummy" users.*

Recommender System

Deciding K

- K was defined as the number of input ratings by a single user.
- It was calculated to be 7 using the following idea:
  - By looking at data collected, it was evident that the movies were concentrated over the last 5 years.
  - On average, about 100 blockbuster movies are released every year, out of which a person on average watches 20 - 25 movies a year.
  - Thus a person watches around 22.5% of the major movies.
  - As our system contains 176 movies, extrapolation of this data yields

$$K = \frac{176 \times 22.5}{100 \times 5} \approxeq 7$$

- *Additionally, we could augment the results (at least for user user collaborative filtering) by collecting more information about the users viz. location, more diverse age options, region, etc.*

Recommendation Methods

User-User Collaborative Filtering

It is a form of collaborative filtering for recommender systems which identifies other users with similar tastes to a target user and combines their ratings to make recommendations for that user.

- Karl Pearson's correlation is used to see how similar two users are; this normalizes user optimism and ensures equal-scaled similarity matching.
- Given that our user genres have been modeled off real world statistics, this correlation metric is representative of a real-world setting.

Item-Item Collaborative Filtering

It is a form of collaborative filtering for recommender systems based on the similarity between items calculated using user's ratings of those items.

- *Item-Item* models use rating distributions per item, not per user. With more users than items, each item tends to have more ratings than each user, so an item's average rating usually doesn't change quickly. This leads to more stable rating distributions in the model, so the model doesn't have to be rebuilt as often. When users consume and then rate an item, that item's similar items are picked from the existing system model and added to the user's recommendations.
- To calculate similarity between two items, we look into the set of items the target user has rated and computes how similar they are to the target item i and then selects k most similar items. Similarity between two items is calculated by taking the ratings of the users who have rated both the items and thereafter using a similarity function.

Matrix Factorization

Matrix factorization is a class of recommender systems algorithms which work by decomposing the user-item interaction matrix into the product of two lower dimensionality rectangular matrices. (Matrix Decomposition)

- The strength of matrix factorization is the fact that it can incorporate implicit feedback, information that are not directly given but can be derived by analyzing user behavior. Using this strength we can estimate if a user is going to like a movie that (he/she) never saw. And if that estimated rating is high, we can recommend that movie to the user.
- Technique used: SVD++
  - SVD++ was designed to take into account implicit interactions.
  - Compared to older SVD, it also takes in account user and item bias.