Movie Recommendation Engine and Feedback Analysis

Alexander Karl,[1] Ankit Kumar,[1] Edwin Amponsah,[1] Ross Brancati[1] and Sathvik Thogaru[1]

[1]University of Massachusetts Amherst

Statistical Computing Final Presentation
December 2, 2021

Lab 4: Movie Recommendation

Lab Goal:

Use past movie ratings to predict how users will rate unwatched movies and generate recommendations for these users.

Recommendation Systems:

Movie recommendation systems feeds new user ratings into an engine to generate future recommendations, so recommendations are constantly updated.

Our Extensions:

1) Check if the initial preferences of the group of users change as more movies are seen
2) Check if group is recommended with a varying set of movies over time
3) Check is recommendations will converge to highly rated genre as more movies are seen

Methods: Overview

Part 1: Recommendation Engine Development

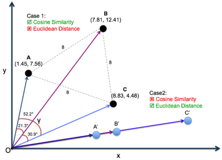
- Collaborative Filtering Techniques
- Machine Learning Algorithms Employed:
  - K-Nearest Neighbors
  - Matrix Factorization

Part 2: Simulation

- Simulate user ratings to answer the following questions:
  - Do movie recommendations change over time?
  - If so, how do they change over time?
  - Do global recommendations converge to one or more popular movies?

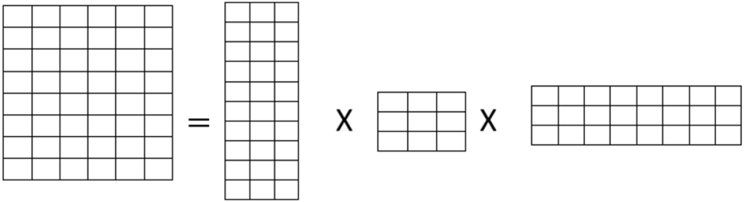
Model 1: K-Nearest Neighbors (KNN)

- KNN is a non-parametric machine learning algorithm that is used in classification and regression
- New data points are classified based on proximity to its number of nearest neighbors, where k represents the number of neighbors
- Using euclidean distance to measure distances suffers from the curse of dimensionality (use cosine similarity instead)



- This method has a few drawbacks, so we ended up generating a model using matrix factorization

Model 2: Matrix Factorization



$$M = U \times D \times I^T$$

where,
M := (n x n) utility matrix.

U := (n x r) represents the relationship between users and latent factors.

D := (r x r) diagonal matrix, which describes the strength of each latent factor.

I^T := (r x n) matrix, which indicates the similarity between items and latent factors.

Simulation Design

1. Begins each simulation with user movie ratings data
2. Runs base data through the recommendation engine to recommend a movie to 'watch'
   - 2.1 One movie generated for every user
   - 2.2 Intentional randomness. User randomly pics 1 from their 5 highest recommended movies
3. Updates the user movie ratings database
4. Runs modified database through the rec engine again, for 2nd generation movie rec
   - 4.1 Up through 20 generations of movies
   - 4.2 Data, and thus recommendations, change over time
5. Simulation: repeat steps 1-4 multiple times
   - 5.1 'Alternate Universes' of Netflix binges
   - 5.2 Intentional randomness becomes important here
6. Simulation allows us to see trends in how recommendations tend to change

Methods: Simulation Pseudo-code

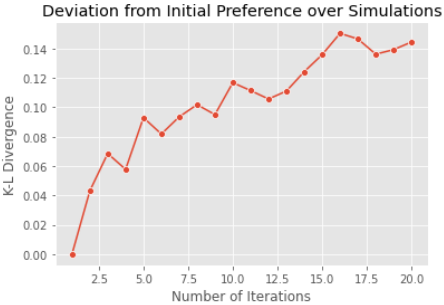
```
class Simulation():

  #INITIALIZE: use raw df of users's movie ratings
  def __init__(self, user_movie_df):
    self.user_movie_df = user_movie_df
    self.sim_results = []

  def simulate(self, n_sim, n_movie):

    for i in range(n_sim):
      #start each sim with the original data
      temp_user_movie_df = self.user_movie_df.copy(deep=True)

      for j in range(n_movie):

        #REC ENGINE: recommend and rate 1 movie for every user
        movie_recs = rec_engine(temp_user_movie_df)

        #UPDATE data for next run
        temp_user_movie_df.update(movie_recs)

      #STORE the result of this sim iteration
      self.sim_results.append(temp_user_movie_df)


# Execute ---------------------------------------------------
sim = Simulation(user_movie_df)
sim.simulate(n_sim=100)
sim.user_movie_df
```

Note the use of a simulation class

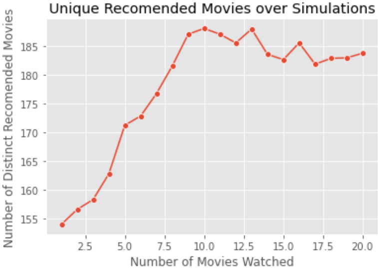
Results: Change in Group Preferences Over Time

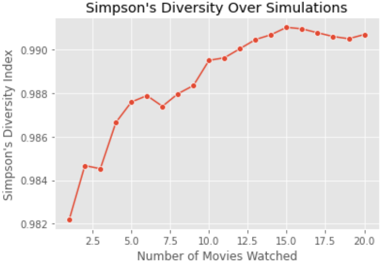
- Calculate the initial genre distribution and the genre distribution of the ith iteration
- Calculate the Kullback-Leibler divergence (KLD) between initial genre distribution and the genre distribution of the ith iteration

$$KL(P||Q) = \sum p_i(x) log(\frac{p_i(x)}{q_i(x)})$$



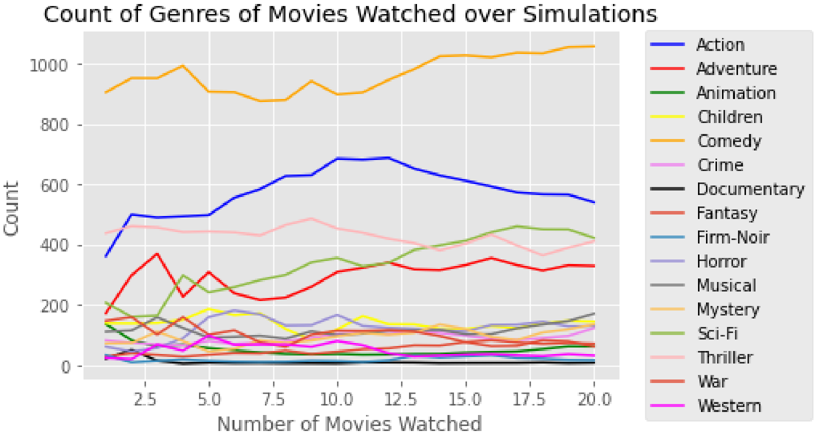
Homogenization of the User Experience





$$D = 1 - \left(\frac{\sum n(n-1)}{N(N-1)} \right)$$

Popularity Bias



Summary

- There was a higher deviation from the initial preference over time. This suggest that the taste of the users shift over time as more movies are watched by users
- More unique set of movies are recommended over time as more movies are watched by users
- More popular genres are more recommended over time as more movies are watched by users

References

1. Amatriain, X., Jaimes, A., Oliver, N., Pujol, J. M. (2011). Data mining methods for recommender systems. In Recommender systems handbook (pp. 39-71). Springer, Boston, MA.
2. Recommendation Systems, Chapter 9, Rajaraman, A., Ullman, J. D. (2011). Mining of massive datasets. Cambridge University Press.
3. Beheshti, B., Desmarais, M. C., Naceur, R. (2012). Methods to Find the Number of Latent Skills. International Educational Data Mining Society.
4. Singular Value Decomposition (SVD) Its Application In Recommender System, https://analyticsindiamag.com/singular-value-decomposition-svd-application-recommender-system/
5. Liao, K., movie_recommender (2018). GitHub repository, https://github.com/KevinLiao159/MyDataSciencePortfolio/