Mikayla Norton, Steven Strachan,
Lacey Hamilton, Saumya Shah
Spring 2023

Computational Optimization (CMSE 831)

Final Project

I. Introduction

For the goals of the final project in the spring 2023 Computational Optimization course, the team designed a project centered around movie recommendations systems. The team aimed to reformulate existing models with materials techniques used from the semester's teachings. The project utilized a large dataset from Kaggle, titled "MovieLens 20M Dataset". The data was collected from MovieLens, a film recommendation service, and contains around 20 million data entries. The final goal of the project aimed to generate various recommendation systems using different applications of course materials. The team also hoped to implement systems that used item-based, user-based, and hybrid recommendation practices.

II. Exploratory Analysis

The MovieLens 20M Dataset is the combination of six .csv files– "tags", "rating", "movie", "link", "genome scores", and "genome tags". Descriptions of each file are as follows:

i. tag.csv: Contains four columns of user ID, movie ID, timestamp of review, and tags applied to movies by users.
ii. rating.csv: Contains four columns of user ID, movie ID, rating and timestamp of said rating per entry.
iii. movie.csv: Contains three columns with detailed information for each movie. From each movie ID, the title and genre is also provided.
iv. link.csv: Contains three columns to connect movies to other sources and identifiers. Movies are linked to IMDb and TMDb with unique IDs for each.
v. genome_scores.csv: Contains relevant data to associate movie IDs and tag IDs. The data has three columns, movie ID, tag ID, and relevance (a correlative scoring metric).
vi. genome_tags.csv: Contains two columns to associate tag names with tag IDs.

For the majority of the team's modeling and algorithm construction, methods utilized a joined dataframe of ratings and movies on the movie ID. Throughout the full dataset, there at 671 unique user IDs of sequential order and 9066 unique movie IDs of non-sequential order. The data card also provides that each user has voted for at least 20 movies. The rating scale spans a zero to five interval.

III. Methods

i. *KNN Item-Based*

The KNN algorithm utilizes Euclidean distance, or an L2 norm to solve for a recommendation based off item-based tactics. To clean the data, this model replaced all NA values with zero values, and condensed the dataset down to movies with 1000 or more votes.

Through utilization of the "fuzzywuzzy" package in Python, a package which allows for handling of language data despite misspellings, movies were searchable in the data and would return similar movies. These similar movie recommendations were generated through the closest distance based on ratings from user IDs. This model was generated with six, eight, and ten neighbors, all with similar results.

ii. *Correlation Item-Based*

The Correlation algorithm utilized an item-based system to generate recommendations. The team used the movie titles and IDs, along with ratings and user IDs. The data was once again consolidated with the same methods as above.

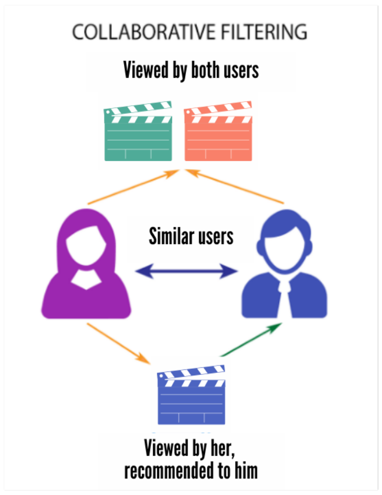
The algorithm generated random users by ID numbers and a list of movies rated highly by that user. Recommended movies are then suggested based on the correlation to the user's most highly rated movie.

iii. *Item Association Rule Mining*

Building upon the correlation method described above, the objective of this method is to uncover frequent item sets of features. From a completed user-feature matrix, we can selectively filter out the low ratings to uncover quality feature baskets to inform our user based recommendation methods.

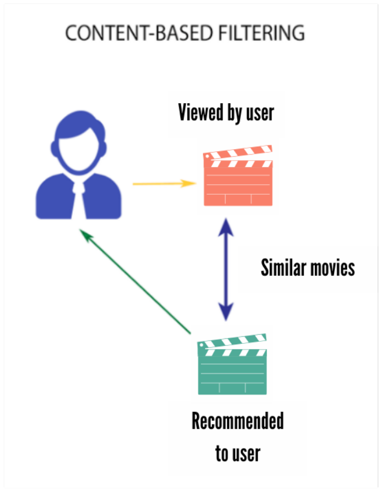
iv. *Collaborative Filtering User-Based*

Collaborative filtering systems consider two similar users within the data and make recommendations based off of shared viewing history. For example, movies seen by two users categorize them as similar, and movies seen by only one of user can then be recommended to the second due to the similar nature in viewing habits.



v. *Content-Based Recommender*

Content-based filtering examines similarities between movies and the respective content to create recommendations. When a user has viewed one film that has similar content to another film, that latter film is more likely to be recommended back to the user.



vi. *Mean Imputation and SVD*
Mean imputation revolved around replacement of missing values with the mean of observed values, and in this case, was used to replace missing user ratings. The dataset was modified to a user-item matrix and imputed with mean values where missing. The data was then set into sparse matrix format and factored into the SVD matrices. The SVD can then be used as a means of factoring the matrix into a user matrix, a factor matrix, and an item matrix, and from there was used to predict missing ratings and recommend items to users based on the predictions.

vii. *Singular Value Thresholding*
Using the same user-item matrix from above, SVT is utilized as a low-rank approximation recovery method. To solve $\|X - A\|_F$ subject to $\text{rank}(X) \leq k$, where X is the low rank approximation, A is the observed matrix, and k is the desired rank of the low-rank approximation, iterative thresholding was performed to convergence.

viii. *Principle Component Pursuit*
PCP aims to decompose the user-item matrix to low-rank and sparse components by solving the following optimization problem: $\arg\min \|L\|_\star + \lambda\|S\|_1$, subject to $L + S = A$. Where L is the low-rank component, S is the sparse component, λ is the regularization parameter, and A is the observed value matrix. By obtaining the components, low-rank can be used to make recommendations to users by finding the cosine similarity between low-rank component of the user-item matrix and each user's rating history. The top movies with highest similarity will be recommended to the user.

ix. *Soft-Impute*
Soft-Impute aims to solve $\arg\min \|X - A\|_F^2 + \lambda\|X\|_\star$ as an optimization problem in order to recover a low-rank approximation of the user-item matrix. This method is coupled with SVT and the matrix is truncated to a rank, k, during convergence. Once obtained, the approximation can be used to make recommendations to users based on cosine similarity.

x. *Low-Rank Matrix Recovery*
Using the user-item matrix, low-rank matrix recovery decomposed to a low-rank and a sparse component by solving the optimization problem $\arg\min \|L\|_\star$ subject to $A = L + S$. Solving via ADMM and calculating the cosine similarity between low-rank components and user rating history allows for the proposal of most-similar movies as recommendations to the user.
This method differs from PCP in that low-rank matrix recovery seeks to minimize the nuclear norm of the low-rank component, while PCP seeks to minimize the sum of the nuclear norm and $L1$ norm of the sparse component.

xi. *Non-Negative Matrix Factorization*
NMF factorizes the matrix into two non-negative matrices, a user-feature matrix and a feature-item matrix. The first represents the degree to which each user associates with each feature, and the second represents the degree to which each item associates with each feature.

xii. *Robust PCA*
This method first decomposes the original matrix into two low rank matrices with representing the underlying structure and the other representing noise. To better handle outliers or missingness, the sum of the nuclear norm of the structure matrix and the L_1 norm of the noise is minimized with the constraint that the matrix dimensions stay constant.

xiii. *Randomized Matrix Completion*
This method fits a Gaussian distribution to the recorded values for each feature and fills in missing values with a randomly generated value. Although not particularly useful on its own, can be used for partial imputation before other methods are applied.

xiv. *Deep Learning Method*
While there are various deep learning methods available, the implemented method uses PyTorch's SparceAutoencoder module. At a high level, the neural network is used to project the matrix into a lower dimension and then uses the lower dimension to construct a completed matrix. The SparceAutoencoder implementation uses a mean squared error loss function to make it sensitive to outliers.

xv. *Bayesian Probabilistic Matrix Factorization*
Using samples from an intractable prior distribution by Markov Chain Monte Carlo, Bayesian Probabilistic Matrix Factorization then generates approximate parameters. From these parameters, two lower dimensional matrices are created and dotted together to obtain value estimates.

xvi. *Gradient Descent*
Gradient descent can be used and iterated against an objective function that minimizes the difference between actual and predicted values. Common implementations include matrix-wide or single observation update methods.

IV. Algorithm Testing

After the implementation of each method within python, the testing mythology was designed. Initially, our sparse matrix had an additional five percent of non NaN values dropped out and recorded. We then pass our dropped out matrix into each method, and take a root mean squared error between the sparse matrix before dropout and after value completion at each of the dropped value locations. This process was repeated ten times and an overall average root mean squared error value was obtained as the final accuracy measurement.

V. Findings

Method	Error
Mean Imputation and SVD	0.948
Singular Value Thresholding	2.728
Principle Component Pursuit	3.676
Soft-Impute	0.947
Low-Rank Matrix Recovery	.330
Non-Negative Matrix Factorization	3.504
Robust PCA	2.440
Randomized Matrix Completion	2.878
Deep Learning Method	3.676
Gradient Descent	INT⋆
Bayesian Probabilistic Matrix Factorization	INT⋆

⋆ ”Intractable”

VI. **Further Steps**

With the combination of results from our correlation, association mining, matrix completion algorithms, and language processing, the next step is to make a composite model. One of many possible composite models is formulated as follows:

$\text{recommendation} = \max(\frac{1}{\text{Confidence Rank}} \cdot \text{MF value} \cdot \frac{1}{\text{NLP Rank}})$