**Movie Recommender System – High-Level Information**

This is a movie recommender system that uses the movie lens latest dataset (link: <https://files.grouplens.org/datasets/movielens/ml-latest.zip> ) to generate movie recommendation for a user based on their already existing movie lens ratings.

*(You can rate movies on movie lens and download your ratings from there if you don’t already have Movie Lens Ratings).*

The program uses a KNN algorithm to generate content-based similarities and another KNN to generate user-based similarities. Both the resulting matrices are passed as two sets of visible dimensions to a modified RBM to generate recommendation for the user.

**Required Libraries:**

1. **Python 3.8** (should work with versions >= python 3.6)
2. **Surprise** (used to convert movie ratings into a training dataset)
3. **Tensorflow** (Used for the RBM)
4. **Numpy** (Used for both, the item similarity KNN and the user similarity KNN)
5. **Pickle** (used to save/load learned weights and matrices, in a highly compressed lzma format, to run the program without having to perform re-training)
6. ~~Scipy~~ (was used to create sparse matrices, however, the program now only uses dense matrices and scipy is now removed)

**Instructions:**

*Note: It is recommended that a machine with at least 16GB of RAM is used, however, the program should still work with 8GB RAM. Program tested only in Windows, however should work in a Mac as well*

1. First download anaconda 3 (preferably anaconda navigator as well to have an intuitive User Interface) and it is highly recommended to create an environment to run the program.
2. Download this git repository
3. Download the movie lens latest data set (from: <https://files.grouplens.org/datasets/movielens/ml-latest.zip> ) and paste it in the “ml-latest” folder downloaded from git. Make sure the files are in csv format.
4. Install the following libraries/packages: - python 3.8, Surprise, Tensorflow, Pickle, Numpy (and I also used Spyder as the IDE) – into the environment the program will be run in.
5. Download your ratings in csv format from Movie Lens (you can do this in settings > export ratings). Paste the downloaded ratings in the directory: “MyRatings”
6. If you would like to store the downloaded data csv files in different locations than default, please modify the “config.config” file in the “config” directory.
7. If you would like to filter certain words from movie titles being recommended – and effectively not recommend movies with those words in the title – please modify the “stoplist.txt” file in the “stoplist” directory.
8. Go to the directory: “src” and run the “main.py” file in the environment with all the installed libraries/packages.
9. Follow the instructions on the CLI. You can even enter specific general genres or combination of genres that you would like to only be recommended to you.
10. It is highly recommended not to retrain the model, as it takes several hours, however, you may choose to re-train if you plan to use more – or less - data from the data set, or simply test the training process. *(current, the program does a hard stop at 3,000,000 ratings, to be able to run on most 8GB systems)*
11. Simply choose to receive your recommendations (or re-train) and the program will take a few minutes to run.
12. The program will then display 25 of you topmost recommended movies, and will save a csv file with rankings of ALL the movie recommendations – consisting of all the movies in the dataset – in the directory “MyReccomendations”

**The Model:**

High Level Control Flow:

The program first chops the ratings data into only 3 million ratings (to fit into memory on personal computers) and saves a copy of the original file. The ratings are then fed into surprise library (not written by me) to convert the ratings into a dataset. This dataset is what is used as the initial training matrix in the model.

Data from other files are read and used to construct the genre vectors, genome vectors, average movie ratings, popularity ranks, movie years and saved matrices and model weights pre-compute by me so a user won’t have to (since it would take several hours to conduct training to get recommendations otherwise) that are used in the following algorithms. This and the following sections are written by hand.

The model uses a Content Based Similarity KNN, User Similarity KNN (collaborative filtering) and an RBM with two Visible Dimension inputs.

The model first uses KNN to compute content-based similarities for every movie and predict ratings for the unknown movie ratings for every user. The purpose of this is to remove the sparse matrix problem, as user ratings data is very sparse. The resulting predicted ratings from this algorithm are used to replace the ratings for the unrated movies for each user, resulting in a dense matrix.

This dense matrix, which is a merged user-rating-and- item-similarity-predicted-rating matrix, is then passed into another KNN which determines user similarities and predictions for all ratings from the previously constructed dense matrix – This is the collaborative filtering step.

The matrices computed from each of the KNNs – the first merged item similarity predicted movie ratings and the second user similarity predicted movie ratings – are both passed as visible dimension into a special custom RBM that takes two sets of visible dimensions, which then uses contrastive divergence to converge and predict ratings for every user.

The resulting ratings from the RBM are then further slightly modified by average user ratings based on time of rating, most popular genre for each use, overall popularity of movies weighted by number of ratings and finally the similarity of the movie’s release year to the median release year of the movies in each user’s dataset.

Movies with words in their titles matching words in the stoplist file (“stoplist.txt” in “stoplist” directory) are filtered out and not added in the recommendations.

These slightly adjusted ratings are then used to produce movie recommendations ordered by rank (predicted rating) for the user in question.

*Note: Due to the long computational times and high amount of RAM usage, the model is pretrained to determine movie similarities, and user ratings and RBM variable weights for all users in the data set – not including your personal ratings. After which only the personal movie similarity ratings, user similarity ratings and overall predicted movie ratings and recommendations are computed for the ratings added in the “MyRatings” directory.*

Algorithms Used:

Movie Similarity KNN

This algorithm uses a custom KNN algorithm written using numpy, to determine similarities between all movies by computing the cosine similarities between their genres and from the movie genome dataset downloaded from movie lens (genomes come from the movie lens dataset which contains the percentage of various tags that make up the movie.) Additionally, the difference between release years, overall popularity in the data set and overage ratings given by every user are also used in the similarity score.

These movie similarities are used to create top-k (k=40 set by me, you can set it to different values if you wish) neighborhoods of the most similar movies and determine the movie’s estimated rating for every user.

The main purpose of this KNN is to convert the sparse user ratings matrix into a dense matrix, by adding the Item Similarity ratings for each movie for each user in place of the unrated movies (rated 0 by default when unrated).

The reason this is done is because sparse data in an RBM leads to both overtraining and inaccurate results. So initially creating a dense matric can give much better predictions for the final recommendations.

User Similarity KNN

This algorithm uses a custom KNN written in numpy, that generates user similarities between users by computing the cosine similarities of the movies they rated (and the predicted ratings of the unrated movies from the Movie Similarity KNN algorithm from earlier).

The user similarities are used to generate the top-k similar users neighborhoods (k=40 set by me, but you can change this if you wish). The average rating for each movie and the total rating scores of each movie in each user’s neighborhood is used to determine the collaboratively filtered rating score for each movie for each user.

The main purpose of this KNN is to find ratings for movies for a user based on ratings given for movies by users similar to the user – collaborative filtering. Since the matrix is already made dense from the item similarity predicted ratings from earlier, there is no further processing needed for this resulting matrix.

RBM with two input visible dimension (Explainable RBM)

This algorithm takes the User Similarity Predicted Ratings Matrix and the Merged user-rating-and-item-similarity-predicted-rating Matrix as inputs (visible dimensions). These two matrices are multiplied by a set of “weights” for each to generate the hidden layer by adding a “hidden bias”. The hidden layer is then used to re-construct each of these matrices using a “bias” for each matrix and their respective “weights” used. The algorithm then converges using contrastive divergence until the input matrices can be re-constructed better, adjusting all the sets of “weights” and “biases” for the two input matrices in the visible layer and the hidden layer. Tensorflow is used to maintain the session and graph constructed for the RBM.

The computed “weights” and biases are used to predict the ratings for (each/) the user by passing the user’s ratings - merged with predicted item similarity ratings – and user’s user similarity predicted ratings as input and only the densified user ratings matrix as output.

**Sample Results After Training or Simply Running:**

Loss Graph from Training the RBM:

Chart, line chart

Description automatically generated

Computed User Recommendations File (Sample):

Graphical user interface, application, table, Excel

Description automatically generated