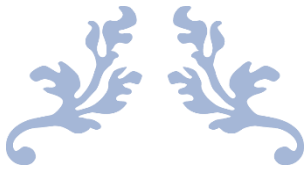


DATA ACQUISITION & PROCESSING

Project Report

Contents

Purpose of the Project

A Recommender System is a software for predicting user preferences and recommending items that are interesting for them. Recommendations make it easier for users to access content that users are interested in. There are three major ways: content-based, collaborative filtering, and hybrid.

In this project, my goal is implementing collaborating filtering algorithms and developing a movie recommender system based on the MovieLens 100K Dataset.

CF Algorithms

Collaborative filtering is commonly used in recommendation systems. Collaborative filtering algorithm makes recommendations based on historical users' preference for items. In our case its movie rating. Recommendations are created based on similarities among entities. There are two classes of Collaborative Filtering, I implemented both user-based CF and item-based CF.

I used Pearson Correlation for calculating user similarities to implement user-based CF. Pearson Correlation measures correlation between two variables with value between +1 and −1. A value of +1 is total positive correlation, 0 is no correlation, and −1 is total negative correlation. And for the prediction I used the following function in figures.

$$sim(a,b) = \frac{\sum_{p \in P}(r_{a,p} - \bar{r}_a)(r_{b,p} - \bar{r}_b)}{\sqrt{\sum_{p \in P}(r_{a,p} - \bar{r}_a)^2}\sqrt{\sum_{p \in P}(r_{b,p} - \bar{r}_b)^2}} \qquad pred(a,p) = \overline{r_a} + \frac{\sum_{b \in N} sim(a,b) * (r_{b,p} - \overline{r_b})}{\sum_{b \in N} sim(a,b)}$$

(a, b : users) (r(a,p) : rating of user a for item p) (P : set of items, rated both by a and b) (N: neighbors of user a)

I used Adjusted Cosine Similarity for calculating item similarities to implement item-based CF. Adjusted cosine similarity is a modified form of vector-based similarity. For the prediction I used the following function in figures.

$$sim(\vec{a},\vec{b}) = \frac{\sum_{u \in U}(r_{u,a} - \overline{r_u})(r_{u,b} - \overline{r_u})}{\sqrt{\sum_{u \in U}(r_{u,a} - \overline{r_u})^2}\sqrt{\sum_{u \in U}(r_{u,b} - \overline{r_u})^2}} \qquad pred(u,p) = \frac{\sum_{i \in ratedItem(u)} sim(i,p) * r_{u,i}}{\sum_{i \in ratedItem(u)} sim(i,p)}$$

(a, b : items) (U: set of users who have rated both items a and b)

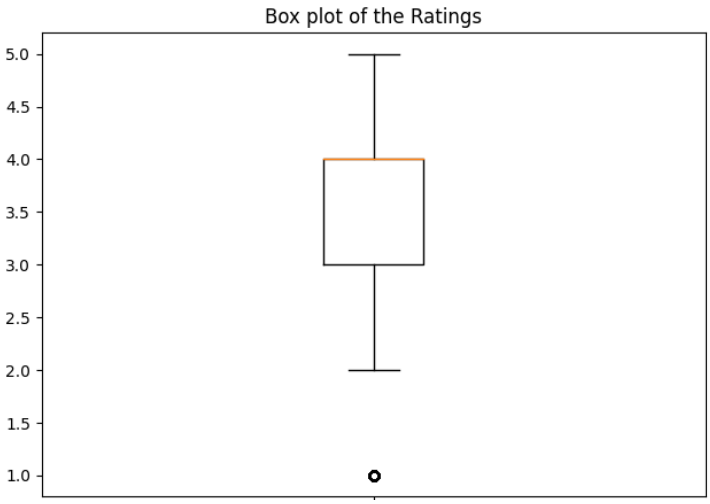
Dataset Overview

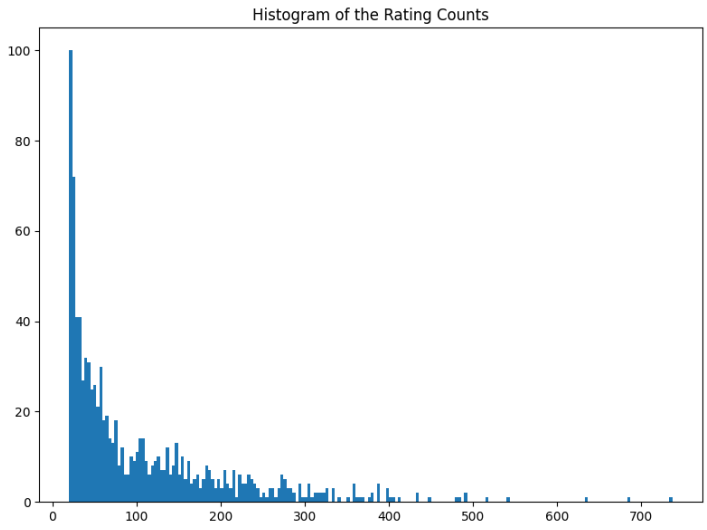
These figures are outputs of the visualization.py *application.*

Summary of Dataset	
Total number of Users	943
Total number of Movies	1682
Total number of Ratings	100000

Summary of Rating Counts	
Mean of the Rating Counts	106.045
Median of the Rating Counts	65
Mode of the Rating Counts	20
Minimum of the Rating Counts	20
Maximum of the Rating Counts	737

Summary of Ratings	
Mean of the Ratings	3.52986
Median of the Ratings	4
Mode of the Ratings	4
Minimum of the Ratings	1
Maximum of the Ratings	5





The dataset consists of 943 users, 1682 movies, and 100.000 ratings. Users have at least 20 ratings. Most users have 20 ratings and users have an average of 106 ratings.

Ratings are in range from 1-5. Average of the ratings is 3,5. And 4 is the most frequently used ratings by users.

Based on a 100,000 ratings and total number of 1,586,126 combination of users and ratings, 6,3% of the data set that can occur is full. I used dictionaries when reading data in algorithms.py application because I thought using matrix would create a sparse matrix.

Experimental Methodology

I used k-fold cross-validation algorithm. k-fold cross-validation divides the data into equal parts according to a specified number of k, allowing each part to be used for both training and testing. Also, deviation and errors caused by fragmentation are minimized. However, training and testing the model as much as k requires time. For 100.000 ratings I solve this problem with some optimizations. E.g., Creating dictionary for avoiding repetitive similarity calculation.

In experiments.py application I have created algorithms that can work for 5 and 10 folds but in the project, experiments and their results are calculated for the value of k 10.

I used k-nearest neighbor algorithm. The k-nn algorithm assumes that similar things are near to each other, there is no need to make additional assumptions. Also, the algorithm is versatile. I used Pearson correlation and Adjusted cosine similarity determining the distances between users / items.

One of the main objectives of the project is experiment with different k values and finding the optimum k number for k-nn. I coded experiments.py application for this purpose. The results obtained for k = 10, 20, ..., 80 from experiments.py are in section 5.

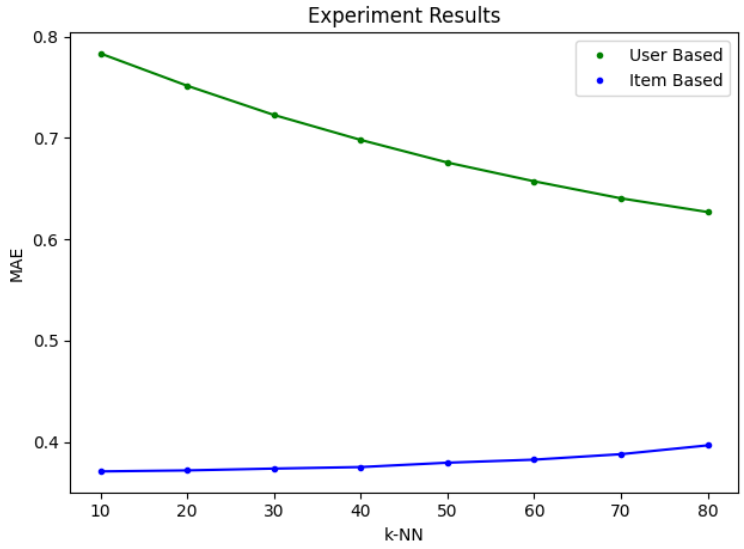
Experimental Results

I measured the success of the Recommender System using the mean absolute error method. MAE is a measure of errors between paired observations expressing the same phenomenon.

In experiments.py application after the completion of k-folds method a prediction dictionary is created. Error rates were generated by comparing the prediction dictionary and the rating dictionary. And for the calculation, MAE is used.

```
--------------------------
User based experiment
--------------------------
k-near = 80 | mae = 0.6269
k-near = 70 | mae = 0.6404
k-near = 60 | mae = 0.6572
k-near = 50 | mae = 0.6757
k-near = 40 | mae = 0.698
k-near = 30 | mae = 0.7226
k-near = 20 | mae = 0.7514
k-near = 10 | mae = 0.7832
--------------------------
```

```
--------------------------
Item based experiment
--------------------------
k-near = 80 | mae = 0.3968
k-near = 70 | mae = 0.3881
k-near = 60 | mae = 0.3827
k-near = 50 | mae = 0.3797
k-near = 40 | mae = 0.3754
k-near = 30 | mae = 0.3739
k-near = 20 | mae = 0.3721
k-near = 10 | mae = 0.3711
--------------------------
```



Conclusion

I read the course materials and articles. I completed the project by combining the knowledge I gained with my laboratory experience. In this project, I learned to implement k-fold and k-NN methodologies, user-based and item-based CF algorithms. Finally, I tested the recommendation system I developed.

Python Files

- **algorithms.py**

Includes implementation of reading dataset, similarity, prediction, evaluation functions

- **experiments.py**

Implementation of user-based and item-based experiments, k-fold and k-NN methodologies.

- **visulaization.py**

Includes Dataset and experiment result visualizations.

- **recommend.py**

Recommendation application. Gets a user input then recommends n movies by creating prediction for users' unrated movies. Sorts the predictions then returns n of them.