Recommendation System Based on Hybrid Model

Using movie review dataset

Name	Workload
Dapeng Jin	Compute similarity in collaborative filtering model, and write related reports. Finished format design of the report.
Haoyang Zhang	Propose the hybrid model, split and preprocess datasets, implement content based model, and write related reports, including recomandation. Prepare for the presentation.
Kunpeng Song	Implement latent factorizing model, and wirte related reports.
Xueyu Wu	Programming and Training collaborative filtering model with initialized weights, implement hybrid model and recommandation list, and write related reports. Finetune the whole models.

ABSTRACT

We use a hybrid model based on three models, including Content Based model, Collaborative Filtering model and Latent Factor model, to implement Rating Prediction and Item Recommendation using MovieLens dataset.

KEYWORDS

Recommendation System, Content Based, Collaborative Filtering, Latent Factor, Rating Prediction, Item Recommendation

1 INTRODUCTION

1.1 Background

Recommender systems are utilized in a variety of areas, and are most commonly recognized as playlist generators for video and music services like Netflix, YouTube and Spotify, product recommenders for services such as Amazon, or content recommenders for social media platforms such as Facebook and Twitter.

These systems can operate using a single input, like music, or multiple inputs within and across platforms like news, books, and search queries. There are also popular recommender systems for specific topics like restaurants and online dating. Recommender systems have been developed to explore research articles and experts, collaborators, financial services, and life insurance. [1]

1.2 Problem Setting

In this project, we work on movie recommendation in online review systems using MovieLens dataset. For the movie recommendation dataset, we use the "small" dataset (1M). It includes the user_ID, item_ID, user-item ratings, time, user-movie tags, as well as the metadata of the movies, such as title and genre. It has 100,000 ratings and 3,600 tag applications applied to 9,000 movies by 600 users.

Our project mainly consists three steps: 1) Data Processing, create the training and testing dataset from the original dataset; 2) Conduct rating prediction and make evaluation based on MAE and RMSE; 3) Conduct Top-N Recommendation and make evaluation based on Precision, Recall, Fmeasure, and NDCG. [2]

1.3 Proposed Method

Firstly, we use three models, including Content Based model, Collaborative Filtering model and Latent Factor model, to predict rating and recommend movie, independently.

Secondly, we use regression model to combine the result of this three model to get the final result.

2 PROBLEM FORMALIZATION

The given data set contains 3 sheets: movies.csv, ratings.csv and tags.csv. The rating sheet can be considered as a giant user-by-movie sparse matrix, and our goal is to fill it. The other 2 CSV files are description of movies created by the system in terms of genres and users in term of tags.

Generally, we can assume that each user has its own baseline rating score, and each movie rating is a function of the interaction between the user and the movie. We can model the interaction in terms of the preference of users' "tastes" and the features of movies. Also, we assume the interaction is relatively constant in terms of these latent preference and features. Namely, a user will assign similar ratings to similar movies (, if any), and a movie will get similar ratings from similar users (, if any).

Furthermore, we assume the baseline ratings is smooth in terms of time. Namely, users' overall ratings may gradually increase or decrease because of some external factor, say the development of movie industry, but the baseline will not change dramatically between 2 days.

Therefore, we can think of this problem as filling blank matrix entries given the rank of this matrix is relatively limited together

with some prior knowledge, some heuristics of the correlation between each column.

3 THE PROPOSED MODEL

Given the mathematics model, we have 3 derived heuristics to fill the rating sheet:

1. Content-based model: use movie description to compute feature vectors of each movie, say profiles of each movie, use movie profiles to analysis users' "taste" vector based on their ratings, say user profiles, and compute the similarity between each user-movie pair.
2. Collaborative filtering model: Since the interaction is relatively constant, given a movie to be predicted, we can find some other similar movie already rated by this given user, and use these ratings to predict.
3. Latent factorizing model: Note that the rank of this matrix is relatively limited. We can directly factorize this matrix into 2 parts: users' "tastes" and movie features. The latent dimension is the feature space, and each factor matrix presents the weight of each user and movie on the feature space. Therefore, we can use matrix multiplication to restore blank ratings.

Our model combined above 3 models based on regression. (Fig. 1)

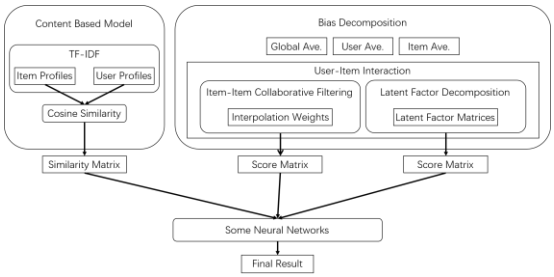

Figure 1: The recommend system model combined content-based model, collaborative filtering model and latent factorizing model.

Generally, our model can be divided into 3 parts: similarity part, which is based on content based model, rating prediction part, which is based on collaborative filtering and latent factorizing, and regression part, whose inputs is the computed similarity and predicted ratings, and the output is the final predicted ratings.

For simplicity, we further assumed the relation between ratings and time is linear, and therefore we preprocess all ratings with a linear regression based on the time it is assigned. For the rating prediction part, we further decomposed each rating into 4 parts: global baseline, user baseline, movie baseline and the interaction between the user and movie. Focus our model predicting the interaction part.

4 EXPERIMENTS

4.1 Splitting Dataset

Considering that our model contains some gradient decent parts, we divided the original data set into 3 parts: training set with about 80% given ratings, test set with about 10% and validation with about 10%. Furthermore, we require every user and every movie presents at least once in the train set, such that the latent factorizing model can somehow predict every user and movie pair.

Therefore, we sampled about $\frac{1}{2.2}$ rows and $\frac{1}{2.2}$ columns, and all ratings that are not located on the intersected position are immediately assigned to training set. At this time, we checked whether every user and every movie was in the training set, and if not, we moved them from the left set to training set. Then, assign the left ratings to test and validation set by flipping a coin. In this way, about $1-(\frac{1}{2.2})^2 = 79\%$ ratings are in training set, and the left ratings are divided into test and validation evenly.

Eventually, we have 81063 ratings in training set, 9932 ratings in test set and 9841 ratings in validation set.

4.2 Data Preprocessing

4.2.1 Linear Regression on Time. We define the object as minimizing the mean square error, and therefore we have the optimum regression function below:

$$\hat{r} = kt + b$$
$$k = \frac{\sum(t-\bar{t})(r-\bar{r})}{\sum(t-\bar{t})^2}$$
$$b = \bar{r} - k\bar{t}$$

Eventually, we have $k = -3.8527375006767954 \times 10^{-5}, b = 0.2567225478923114$. Therefore, we can focus on the time-independent rating $r'_{xi} = r_{xi} - kt - b$

4.2.2 Baseline Decomposition. Here we think of each rating as below:

$$r_{xi}{}' = \mu + b_x + b_i + r''_{xi}$$

We have $\mu = \overline{r'}$, $b_x = \overline{r_x'} - \mu$, $b_i = \overline{r_i'} - \mu$. In this case we can consider the unbiased rating $r''_{xi} = r'_{xi} - \mu - b_x - b_i = r'_{xi} + \mu - \overline{r_x} - \overline{r_i}$.

4.3 Model 1: Content Based Model

We use TF-IDF method to find features. Here is its definition:

$$TF_{ij} = \frac{f_{ij}}{\max_k f_{kj}}$$
$$IDF_i = \frac{N}{n_i}$$

Here f_{ij} indicates the frequency of tag i in movie j, n_i indicates the number of movies that contains tag i, and N is the total number of movies.

Using the TF-IDF score based on movie.csv and tags.csv, we can construct movie profile as a vector.

In order to compute user profile, we choose those high rated movies, and use their rates as weights. More specifically, we further assume

each user's rates are normally distributed. Therefore, we choose highest 54% (i.e. above $\mu - 0.1\sigma$) ratings to compute user profiles.

We define the similarity as cosine similarity. Namely, we have:

$$sim(p_x, p_i) = \frac{p_x^T p_i}{\|p_x\|\|p_i\|}$$

Intuitively, the more similar, the higher ratings it should be, since the similarity describes the degree where the movie corresponds to this user's "tasts".

4.4 Model 2: Collaborative Filtering Model

The basic idea of Collaborative Filtering is that estimate rating for item i based on similar items rated by the same user.

$$r_{xi} = \frac{\sum_{j \in N(i;x)} s_{ij} r_{xj}}{\sum_{j \in N(i;x)} s_{ij}}$$

where S_{ij} is the similarity of items i and j

r_{xj} is the rating of user u on item j

$N(i;x)$ is a set items rated by x similar to i.

4.4.1 Similarity Matrix Problems:

1) Similarity Matrix is too large to fit in the memory.
- Use key value pairs like ((position1, position2), similarity) to calculate the final result.

2) If I use key value pairs, how can I find $\boldsymbol{S_{ij}}$?
- Use the idea of combination. I will show it in detail in step 3).

Steps:

$$sim(i,j) = \frac{\sum_{s \in S_{ij}} (r_{is} - \overline{r_i})(r_{js} - \overline{r_j})}{\sqrt{\sum_{s \in S_{ij}} (r_{is} - \overline{r_i})^2} \sqrt{\sum_{s \in S_{ij}} (r_{js} - \overline{r_j})^2}}$$

1) Calculate the mean rate of every movie $\boldsymbol{\overline{r_i}}$ and $\boldsymbol{\overline{r_j}}$ using mean_ri.
   a. The form of original data is a tuple, (user ID, movie ID, rate, time).
   b. If I want to calculate means, we do not need user ID and time, so that we map it to (movie ID, (rate, 1)).
   c. Then, I reduce the data by key, and I get (movie ID, (total rate, times))
   d. Finally, I get (movie ID, mean)
2) In the formula, I only need to calculate $(r_{is} - \overline{r_i})(r_{js} - \overline{r_j})$, so I first calculate every $(r_i - \overline{r_i})$ using minus_mean.
   a. I can get minus_mean (user ID, movie ID, rate –mean rate).
3) Find S_{ij} and calculate $\sum_{s \in S_{ij}} (r_{is} - \overline{r_i})(r_{js} - \overline{r_j})$ and $\sqrt{\sum_{s \in S_{ij}} (r_{is} - \overline{r_i})^2} \sqrt{\sum_{s \in S_{ij}} (r_{js} - \overline{r_j})^2}$ without using matrix.
   a. Basically, $\boldsymbol{S_{ij}}$ means that a user, say x, both rates the movie i and movie j. Therefore, when we calculate $\sum_{\boldsymbol{s \in S_{ij}}}$, we need to find all rates that user both rates the movie i and movie j, but we do not need to know who is this user.
   b. In detail, I can map my minus_mean (user ID, movie ID, rate–mean rate) to (user ID, (movie ID, rate–mean rate)) and group it by key.
   c. I get (user ID, a list of [(movie ID, rate–mean rate), ……]).
   d. Sort the list by movie ID, cause movie ID must be in ascending order.
      i. Reason: for example, the combination of [1, 2, 3] are (1, 2), (1, 3), (2, 3).
      - I do not want that I have an element (i, j) that i > j.
      - It is very easy to find (i, j), if i < j.
   e. Discard user IDs. I get a list of [(movie ID, rate–mean rate), ……].
- Use combination to combine the list to get ((movie ID_1, movie ID_2), (rate_1, rate_2)) (movie ID_1 and movie ID_2 are rated by one user)

4) Then I can get the Similarity Matrix using the formula

$$sim(i,j) = \frac{\sum_{s \in S_{ij}} (r_{is} - \overline{r_i})(r_{js} - \overline{r_j})}{\sqrt{\sum_{s \in S_{ij}} (r_{is} - \overline{r_i})^2} \sqrt{\sum_{s \in S_{ij}} (r_{js} - \overline{r_j})^2}}$$

4.4.2 Interpolation Weights. However, there are some setbacks of similarity matrix.

1) Firstly, similarity measures are "arbitrary". Although it is easy to understand the meaning of cos similarity, we still do not know what the best measurement is.
2) Besides, Pairwise similarities neglect interdependencies among users. We want our model base on items as well as users such that the prediction may be more precise.
3) Thirdly, taking a weighted average can be restricting. We cannot arbitrarily assign weights according to our guessing.

So instead of sij , using wij that we estimate directly from data might be a good solution.

$$\widehat{r_{xi}} = b_{xi} + \sum_{j \in N(i;x)} w_{ij} (r_{xj} - b_{xj})$$

Where $N(i;x)$ is a set of movies rated by user x that are similar to movie i,

w_{ij} is the interpolation weight, which indicate the interaction between pairs of movies.

Now we have a prediction $\boldsymbol{\widehat{r_{xi}}}$ for a certain x and i. To measure the difference between this prediction and the true rating, we call SSE $\sum_{(i,x) \in R} (\hat{r}_{xi} - r_{xi})^2$ as our error. Then the key idea is to find $\boldsymbol{w_{ij}}$ that minimize SSE on training data by Gradient Descent.

1) Initial weight matrix
- Since we similarities between different movies represent some connection of items, we can initiate our weights based on our similarity matrix. Besides, training time might be tremendous because of a large training data. So

if we initiate our matrix properly, we can get the final weights sooner and better.

- $w_{ij} = s_{ij}$

2) Define training error

- As mentioned before, we set SSE as our training error.
- $J(w) = \sum_{x,i}\left(\left[b_{xi} + \sum_{j \in N(i;x)} w_{ij}\left(r_{xj} - b_{xj}\right)\right] - r_{xi}\right)^2$

3) Update parameters

- In order to minimize $J(w)$, we take the derivate and using gradient descent to update our matrix w.
- $\nabla_w J = \left[\frac{\partial J(w)}{\partial w_{ij}}\right] = 2\sum_{x,i}\left(\left[b_{xi} + \sum_{k \in N(i;x)} w_{ik}(r_{xk} - b_{xk})\right] - r_{xi}\right)\left(r_{xj} - b_{xj}\right)$
- $w = w - \lambda \nabla_w J$

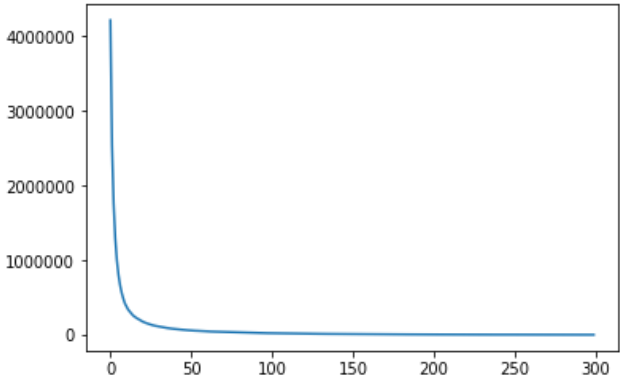

Figure 2: Training error of collaborative filtering

```
    print(user, '\t', movie, '\t', rate, '\t', pre)
print('Testing average error:', terror/tcount)
```

1	6	3.754643027734685	1.861722146778157
1	110	3.754643027734685	3.2710458354543577
1	157	4.7546430277346845	7.290779966279305
1	235	3.754643027734685	5.673960663651059
1	356	3.754643027734685	5.174150156220023
1	362	4.7546430277346845	7.405136441474831
1	367	3.754643027734685	4.150252269325997
1	500	2.754643027734685	2.483503881677709
1	590	3.754643027734685	3.8872286947401435
1	596	4.7546430277346845	6.756416901474926
1	780	2.754643027734685	3.134570951907701
1	1073	4.7546430277346845	7.48280680698367
1	1090	3.754643027734685	3.7189541272193014

Figure 3: Collaborative filtering prediction

4.4.3 Analysis

1) Compare to content based method, collaborative filtering works for any kind of item and there is no feature selection needed. We can make predictions directly from data.
2) Although both w and s matrices are sparse, {key, value} pairs data structure reduces the memory complexity without losing computation convenience.
3) In weights interpolation, according to our trace of training error, it is easy to see that the precision of our prediction is increasing, which means general parameter matrix w does has better performance than sample similarity.
4) However, this method is hard to solve cold start of both users and items, we need related information of a movie or a user to make predictions.
5) This is a popularity bias recommendation model. We cannot recommend items to someone with unique taste and tend to recommend popular items.

4.5 Model 3: Latent Factor model

The Latent Factor model comes from matrix singular decomposition:

$$A = U\Sigma V^T$$

, in which A represents the user-movie data matrix, Σ represents the hidden user-concepts relationship and V^T represents the hidden concepts-movie relationship.

SVD algorithm helps us find the hidden common elements and concepts, which can be used to predict missing values in matrix A. The dimensionality of Σ denotes the number of hidden concepts and is predefined by human.

When predicting missing values, we do the following:

(1) Use SVD to decompose user-movie matrix A to $A = U\Sigma V^T$
(2) We don't need the exact value of Σ, so rewrite this equation as $R = PQ^T$, in which $R = A$ is true for non-zero valus in A and $Q^T = \Sigma V^T$.
(3) Predict $\widehat{r_{xi}} = p_x \cdot q_i = \sum_f p_{xf} \cdot q_{if}$

But the problem is matrix A itself has a lot of missing values remain to be predicted. If we fill these missing values as 0, then SVD will predict all these missing values as 0, which is not intended and unreasonable.

So we can't adopt the traditional SVD approach. And the evolved algorithm is Latent Factor Model. The key point of this model is:

(1) Goal: find matrix P and Q such that:
$R = PQ^T$, is satisfied except for the missing values.
(2) Approach: Stochastic Gradient Decent on P and Q.
$Loss = Sum\ of\ Error\ (difference\ at\ the\ non-zero\ values)$

That is to find, by SDG, P and Q such that:

$$\min_{P,Q} \sum_{(x,i) \in R} (r_{xi} - p_x \cdot q_i)^2$$

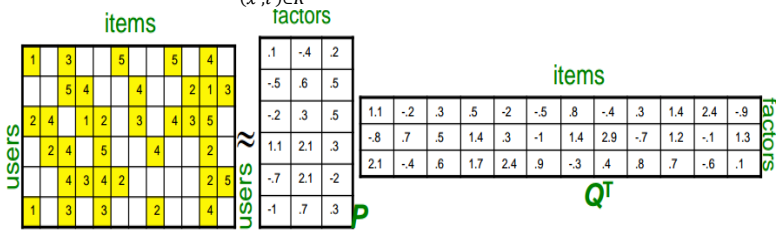

Figure 4: Latent Factor decomposition

In order to avoid overfitting, we add some regularization term to the loss function:

$$Loss = \sum_{(x,i)\in R} (r_{xi} - p_x \cdot q_i)^2 + \lambda_1 \sum_x ||p_x||^2 + \lambda_2 \sum_i ||q_i||^2$$

Notice: $(x, i) \in R$ means all missing values are ignored when computing Loss.

Solve this using Gradient Descent:

(1) Initialize P and Q (randomly with normal distribution)
(2) Compute Loss
(3) Calculate gradient for every elemtent in P and Q: $\partial Loss / \partial p_i$ and $\partial Loss / \partial q_i$
   (a) $\frac{\partial Loss}{\partial p_x} = \varepsilon_{xi} \cdot q_i - \lambda_1 \cdot p_x$
   (b) $\frac{\partial Loss}{\partial q_i} = \varepsilon_{xi} \cdot p_x - \lambda_2 \cdot q_i$
   , in which $\epsilon_{xi} = 2(r_{xi} - q_i \cdot p_x)$
(4) Do gradient descent:
   (a) ${p_x}^{n+1} = {p_x}^n - \eta \frac{\partial Loss}{\partial p_x}$
   (b) ${q_i}^{n+1} = {q_i}^n - \eta \frac{\partial Loss}{\partial q_i}$
   , in which η is the predefined learning rate.
(5) Until a fixed number of loops

About our code:

The python code for our implementation of Latent Factor Model is in LFM.ipynb

Here are some details:

(1) Use the same train data and validation data as used in Collaborative Filterring.
(2) We chose 100 hidden concepts, which is resulted from repeated experiments.
(3) Learning rate: 0.005
(4) Regularization parameter: 0.001

Experiments tell us:

(1) Training error continuously decreases
(2) Test error decreases at first and then increases

So we choose the number of iterations to be 24, giving us the minimum test error.

The final average error of this model (only LFM, no CF) when predicting user rating on test data is:

0.6654601311607836, when the full rating scale is 0 to 5.

4.6 Combination of Three Models Using Linear Regression

There are three different models mentioned previously and they all have their unique advantages as well as setbacks. So in order to improve the final performance and combine merits of these recommenders, a hybrid model is needed.

4.6.1 Linear model. By analyzing different methods, we can see that collaborative filtering and latent factor give us predicted ratings directly. On the other hand, since content-based model returns similarity of users and movies, we can assume that users like movies that they are most similar and tend to give a high rating to these movies, which might be a linear relationship. So in the end, we take a linear model to combine all collaborative filtering, latent factor model and content-based method.

Here we using a learning method of linear regression to get the corresponding weight for each recommender.

$$prediction = \sum_i w_i x_i$$

1) Set weight vector
- Initially we assign all weights as 1 for content similarity, tag similarity, collaborative filtering and latent factor predicter. Each weight will convert to their best respectively during training.

2) Define loss function
- Generally we use SSE as training error. However, in order to avoid over fitting, we add a penalty here to achieve a better performance.
- $J(w) = (wx - r)^2 + ||w||^2$

3) Update weights
- Still we using gradient descent to approach our parameters.
- $\nabla_w J = \left[\frac{\partial J(w)}{\partial w}\right] = (wx - r)x + w$
- $w = w - \lambda \nabla_w J$

```
No. 0 trained successfully! The main error is:  0.6119945616854807
No. 1 trained successfully! The main error is:  0.30239998623949244
No. 2 trained successfully! The main error is:  0.30239979240509846
No. 3 trained successfully! The main error is:  0.3023997924052048
No. 4 trained successfully! The main error is:  0.30239979240520487
No. 5 trained successfully! The main error is:  0.30239979240520487
No. 6 trained successfully! The main error is:  0.30239979240520487
No. 7 trained successfully! The main error is:  0.30239979240520487
No. 8 trained successfully! The main error is:  0.30239979240520487
No. 9 trained successfully! The main error is:  0.30239979240520487
```

Figure 5: Training error of regression model

```
    print(user, '\t', movie, '\t', rate, '\t', prediction)
print('Testing error:', error/count)
```

```
1       6       3.754643027734685       4.30145792562173
1       110     3.754643027734685       3.106666788912617
1       157     4.7546430277346845      4.305978024851584
1       235     3.754643027734685       4.285004318033492
1       356     3.754643027734685       4.847005302256845
1       362     4.7546430277346845      4.506140280559158
1       367     3.754643027734685       4.126233797791866
1       500     2.754643027734685       3.6154557093559854
1       590     3.754643027734685       4.162143030003576
1       596     4.7546430277346845      4.626948716229177
1       780     2.754643027734685       3.89192217695867
1       1073    4.7546430277346845      4.804498751036555
1       1090    3.754643027734685       4.2858443736268
1       1224    4.7546430277346845      4.655357093617042
1       1258    2.754643027734685       2.940986311487935
1       1298    4.7546430277346845      4.820759296598145
1       1445    2.754643027734685       3.2946589943496494
610     168252          4.788315953490599       4.896348263264218
Testing error: 0.4347319119089253
```

Figure 6: Prediction and testing MSE of the hybrid model

4.6.2 Analysis. After training, the final error is about 0.3, which is better than any single predicter of CF and LFM. So we can say that this hybrid model does improve the performance of our prediction and combine advantages from different methods. In the end we set it as our final predicter.

4.7 Computing Recommendation List

Since we have predicted all blank ratings in the whole user by movie matrix, we can pick highest N movies that occur in training set as a recommendation list R_x for each user. (It is possible that these movies occur in test and validation sets.) Therefore, we can compare these lists with sorted test and validation sets T_x to evaluate our result.

Notice that we might face the situation of tiebreaker. In this case, we will choose the movie with highest content-based similarity first because in general this similarity tells us to what extent this movie matches this user's taste, and the higher similarity, the more possible this user prefer this movie.

Here are 4 measures we choose:

1. Precision measures our false positive rate, which is defined as below:
$$Precision = \frac{1}{|X|}\sum_{x\in X}\frac{|R_x \cap D_x|}{|R_x|}$$
2. Recall measures our false negative rate, which is defined as below:
$$Recall = \frac{1}{|X|}\sum_{x\in X}\frac{|R_x \cap D_x|}{|D_x|}$$
3. F_1 combines Precision and Recall, which is defined as below:
$$F_1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$
4. NDCG further takes ranking order into consideration, which is defined as below:
$$NDCG = \frac{\sum_{i=1}^{|R_x|}\mathbb{I}\{R_{xi} \in D_x\}/\log_2(i+1)}{\sum_{i=1}^{|R_x \cap D\_x|}1/\log_2(i+1)}$$
,where $\mathbb{I}$ is the indicator function.

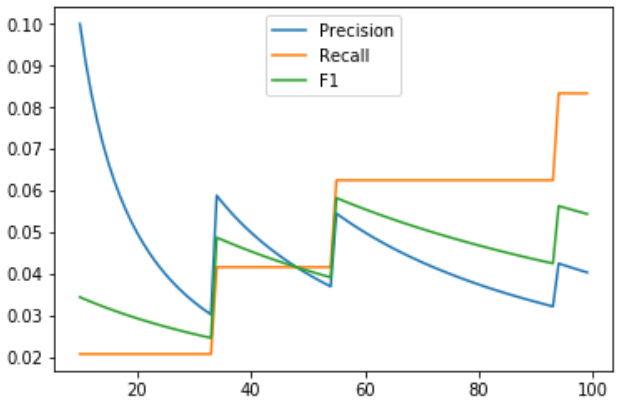

Figure 7: Relationship between measurement and list length N

```
# recommendation
n=10
user=1
rec_list=[]
for movie in r:
    if user in r[movie]:
        continue
    rec_list.append((movie, predict(movie, user)))
rec_list.sort(key=lambda a:a[1], reverse=True)
for i in range(n):
    print(rec_list[i])
```

```
(1237, 5.170540619308386)
(1276, 5.1506538029613065)
(1250, 5.13976236032679)
(142488, 5.13268847318119)
(1212, 5.130391470627705)
(5690, 5.123465486929454)
(1199, 5.116129455741395)
(1136, 5.101350017013809)
(928, 5.098734775000668)
(3451, 5.0915421750746)
```

Figure 8: Predicted recommendation lists for user whose ID is 1

```
Good movies recommended: 3
Recommendations: 55
Good movies 48

Precision: 0.05454545454545454
Recall: 0.0625
F1: 0.05825242718446602

DCG: 0.6826193324736132
IDCG: 2.1309297535714578
nDCG: 0.3203387307017216
```

Figure 9: Measurement of the best recommendation

As we can see, when the length of list is 55, we have our best performed recommandation.

5 CONCLUSIONS AND FUTUREWORK

5.1 Conclusions

On the given validation dataset, our rating prediction eventually get MAE and $RMSE$ (Fig. 10) and item recommendation evaluation. (Fig. 11)

```
    print(user, '\t', movie, '\t', rate, '\t', prediction)
print('Valid RMSE:', (se/count)**0.5)
print('Valid MAE:', ae/count)
```

610	113252	4.2883159534906	4.3173267447092
610	118702	2.7883159534906	2.9549863978361652
610	120635	0.7883159534906004	1.098427601324748
610	122886	4.287429823865445	4.428025128495039
610	122904	2.7883159534906	3.1087373855793765
610	122922	3.2883159534906	3.4056615075615033
610	129937	2.7883159534906	2.7996301144465487
610	130634	4.2883159534906	4.254981709721127
610	132422	1.7883159534906004	1.8306597886337603
610	134368	3.7883159534906	3.8559101378005622
610	140247	3.2883159534906	3.451797778021793
610	143385	3.7883159534906	3.4200545271466765
610	148424	2.7883159534906	2.8001319572734253
610	148626	3.7883159534906	4.067686396746554
610	152081	3.7883159534906	4.0246200797957155
610	158238	4.787429823865446	4.722451875649363
610	161582	3.7883159534906	3.838423440604074
610	162350	3.2883159534906	3.4364189949597113

```
Valid RMSE: 0.6588241608555914
Valid MAE: 0.4505755809210525
```

Figure 10: Overall valid data prediction, RMSE and MAE

```
Good movies recommended: 3
Recommendations: 55
Good movies 48

Precision: 0.05454545454545454
Recall: 0.0625
F1: 0.05825242718446602

DCG: 0.6826193324736132
IDCG: 2.1309297535714578
nDCG: 0.3203387307017216
```

Figure 11: Best recommended number and its performance

5.2 Future Work

In the dataset, links.csv provides external links to Internet Movie Database and The Movie Database, where we might grasp more information using web crawlers, such as the director and casts, which can be helpful to content based model.

REFERENCES

[1] Recommender system. (2019, April 25). Retrieved from https://en.wikipedia.org/wiki/Recommender_system

[2] Zhang, Yongfeng. CS 550: Massive Data Mining and Learning – Project Description. content.sakai.rutgers.edu/access/content/group/0c60a088-c616-4acf-8b58-9560f8d71efd/Lessons/CS550-Project-description.pdf.