Dating Website Personalization

Pin-Hao Chen (phc2121), Siwen Tang (st3101), Leshen Sui(ls3452), Jingyi Wang (jw3562)

1. Abstract

This project focuses on data from the Czech dating site libiseti. The goal was to provide users with personalized dating candidates, according to the rating history of a user to the others. This project implemented collaborative filtering algorithms in two different processes: "building from scratch" and "using Surprise package". Each process would construct two models based on neighborhood-based or model-based algorithms. Through tuning the hyperparameter, evaluating the coverage, and calculating the accuracy, the models were continuously optimized.

2. Introduction

2.1. Background

This project focused on a dating website, where users can give ratings to other users who they have dated before. The website may recommend some new candidates for users according to their dating history.

For a popular dating websites, the higher the matching degree, the more the new joined users. The more the users, the better the reputation. The better the reputation, the more the profit. This turns out to be a positive circle for the dating website.

Thus, a better recommender system should help solve the problem about how to reach a high matching degree for each user. This refers to create a highly personalized dating candidates for each member.

2.2. Objective

The project aims to create models so as to offer each member personalized dating candidates. In this project two collaborative filtering algorithms, neighborhood-based and model-based algorithms is designed.

For the neighborhood-based algorithm, k nearest neighbors were considered while analyzing the sample matrix. For the model-based algorithm, matrix factorization was implemented.

The project evaluated two algorithms' models with row-mean baseline model which will be discussed in section 4. Results.

3. Methodology

3.1. Data

3.1.1. Data Source

The data was selected from http://www.occamslab.com/petricek/data/. This link contained two files: "rating.dat" and "gender.dat".

In the "rating.dat" file, there were about 17 million ratings made by 135,359 users on LibimseTi (a popular dating website). The data format *(id_A id_B, rating)* represented the rating score from user A to user B. The id of users was an integer number and the rating score was an integer number ranged from 1 to 10.

The "gender.dat" file stored the gender information of each user in a form of *(id, gender)*. The gender data was a character. It had 3 different types: M, F, and U for male, female, and unknown respectively.

3.1.2. Data Cleaning

In this project, heterosexuality was only considered in order to simplify the varying complexity. The rating score data was cleaned by removing user pairs with same or unknown genders. According to this process, the rating pairs were able to be represented by a bipartite graph. Moreover, the cleaned data could be stored in a user-item matrix. This project treated males as "user" and female as "item" or "target".

3.1.3. Data Sampling

In this project, three different user-item matics are sampled. First, the sampled matrix is selected from the top 10,000 males and 100 females with the most number of ratings. The reason for such sampling is:

1) It was easier to make the recommendation for loyal customers since there was more information on users with more ratings.
2) This user-item matrix has a higher density.

Due to reason 1), this sample could decrease the risk to recommend popular items; due to reason 2), there would be less missing data and more available data for training and testing.

Second, in order to understand how the model performed there were two extra sample dataset: fully random sampling (random 10,000 male and 100 female) and mixing sampling (top 5,000 male plus random 5,000 male and top 50 female plus random 50 female)

3.2. Model

The models were designed in two different methods: "build from scratch" and "utilize surprise package". Both methods included two collaborative filtering (CF) algorithms: neighborhood-based and model-based algorithms.

3.2.1 Build From Scratch

3.2.1.1. Neighborhood-based Model

The neighborhood-based model was designed under an item base concept, since computing the pairwise similarity on the item would result in a 100-by-100 matrix. However, the result of the pairwise similarity on users is a 10^4-by-10^4 matrix. Hence, by computing the pairwise similarity on the item, the algorithm could reduce a great number of running time in terms of selecting the nearest k neighbors when predicting the unknown ratings.

To measure the pairwise similarity of items, the algorithm computed the pairwise Pearson correlation coefficient, since this coefficient is mean centered and invariant to the rating scale. Due to the integer rating ranged from 1 to 10, data normalization through pairwise Pearson correlation coefficient is important for picking up the neighbors.

This model took two parameters:

1) A matrix with known entries (ranged from 1 to 10) and 0 for unknown entries
2) An integer number stands for the number of nearest neighbors.

Time complexity of this algorithm:
Let *n* be the number of users; let *m* be the number of items; let *k* be number of neighbors

1) Pre-compute the pairwise similarity: $O(m^2 n)$
2) Find k nearest neighbors for every item:
   $T(\text{sort and select top k values}) = O(mlogm * m) = O(m^2 logm)$
3) Predict unknown rating using k neighbor neighbors: $O(mnk)$

Real running time: about 5 minutes for n = 10000, m = 100, k = 10.
Space complexity: $O(mn)$

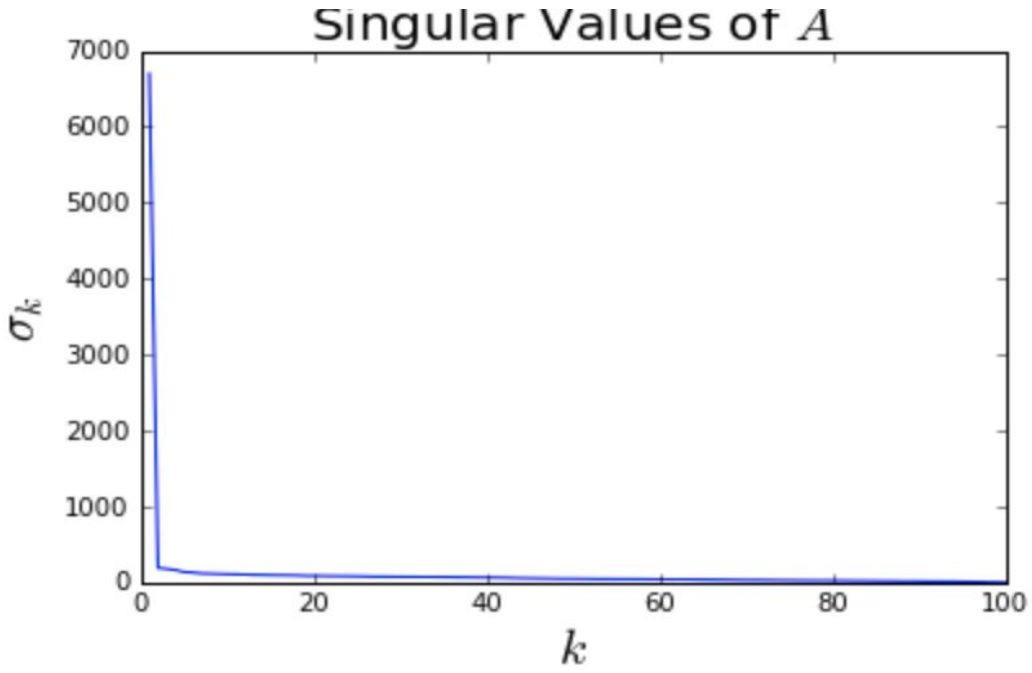
```
def memory_based_CF(df, n):
    similarity = pd.DataFrame(index=df.index,columns=df.index)
    for i in similarity.columns :
        for j in similarity.columns :
            similarity.loc[i,j] = pearsonr(df.loc[i,:],df.loc[j,:])[0]
    df_neighbours = pd.DataFrame(index=similarity.columns,columns=range(1,n+1))
    for i in similarity.columns:
        df_neighbours.loc[i,:n] = similarity.loc[0:,i].sort_values(ascending=False)[1:n+1].index
    predicted_rating = []
    for k in range(len(df_neighbours.index)):
        #print (k)
        name = df_neighbours.index[k]
        neighbor = df_neighbours.loc[name].values.tolist()
        neighbor = df.loc[neighbor]
        nm = df.loc[name].to_frame()
        mean = nm.loc[nm[name] != 0].mean().values[0]
        score = []
        for i in neighbor.columns:
            rate_difference = []
            similarity_score = []
            rated = neighbor[i].loc[neighbor[i] != 0]
            if len(rated.index)==0:
                score.append(mean)
            else:
                rated_mean = rated.mean()
                if df.loc[name,i] == 0:
                    for j in rated.index:
                            rate_difference.append(rated.loc[j]-rated_mean)
                            similarity_score.append(similarity.loc[neighbor.index].loc[j,name])
                    score.append(mean + dot(rate_difference,similarity_score)/sum([abs(x) for x in similarity_score]))
                else:
                    score.append(df.loc[name,i])
        predicted_rating.append(score)
    return pd.DataFrame(data=predicted_rating, index=df.index, columns=df.columns)
```

3.2.1.2. Model-based Model

The idea of the model-based algorithm was to factorize the sample dating matrix into two parts with low ranks. After some sort of initialization, the process utilized the alternate least square algorithm so as to minimize the MSE.

This process aimed to find the best “rank k” for the two factorized matrices. In order to calculate the best rank k, the algorithms included 5 steps as follow:

1) Filled the missing entries (0 values) with the row mean (i.e. average rating of a certain female).
2) Decomposed the rating matrix into 3 parts-- *(U, S, V)*-- by SVD.
3) Reconstructed the rating matrix by multiplying the first k columns in *U* and first k rows in *V*.
4) Plotted the corresponding error from $1 \leq k \leq 100$.
5) Chose the *k* by the elbow method from the below figure. Thus, k would be an integer number around 2 to 10.



This model included 5 parameters:

1) The rating matrix
2) The low-rank
3) The maximum number of iterations (default to be 100)
4) The learning rate (default to be 0.0002)
5) The regularization parameter (default to be 0.02)

Time complexity:

Let n be the number of users; let m be the number of items; let k the rank of factorized matrix

1) Filled all non-zero entries in the rating matrix: $O(mn)$
2) Compute the error using those non-zero entries: $O(n^2mk)$
3) Repeat x iterations until convergence: $O(n^2mkx)$

Real running time: about 5 minutes for $n = 10^4$, $m = 100$, $k = 10$, $x \leq 100$.

Space complexity: $O(mn)$

```
def matrix_factorization2(R, K, steps=100, alpha=0.0002, beta=0.02):
    R = R.as_matrix()
    u,s,v = np.linalg.svd(R)
    a = []
    for i in range(K):
        a.append(i)
    P = pd.DataFrame(u)[a].as_matrix()
    Q = v[0:K]
    N = R.shape[0]
    M = R.shape[1]
    nonzero = list(zip(np.nonzero(R)[0].tolist(), np.nonzero(R)[1].tolist()))
    for step in range(steps):
        print (step)
        for pair in nonzero:
            i = pair[0]
            j = pair[1]
            eij = R[i][j] - np.dot(P[i,:],Q[:,j])
            for k in range(K):
                P[i][k] = P[i][k] + alpha * (2 * eij * Q[k][j] - beta * P[i][k])
                Q[k][j] = Q[k][j] + alpha * (2 * eij * P[i][k] - beta * Q[k][j])
        eR = np.dot(P,Q)
        e = 0
        for pair in nonzero:
            i = pair[0]
            j = pair[1]
            e = e + pow(R[i][j] - eR[i][j], 2)
            for k in range(K):
                e = e + (beta/2) * ( pow(P[i][k],2) + pow(Q[k][j],2) )
        if e < 0.01:
            break
    return P, Q.T
```

3.2.2. Surprise Package

This project also built the two CF algorithms, neighborhood-based and model-based algorithms, with Surprise package. The Surprise-based algorithms aimed to analyze evaluation metrics while changing a range of hyperparameters and different input size.

3.2.2.1. Neighborhood-based Model with Hyper-Parameter Tuning

This algorithm based on KNNBasic method in Surprise package. It was designed to split the sample data into *s* parts and calculate the average MAE of *s* parts. To use the Surprise package, the input needs to meet the following conditions. First, the input data should be formatted in a 3 columns table: (userID, itemID, rating). Second, the hyperparameter "k" should be set when KNNBasic was initialized.

In this project, the s-splitted is set as 5 and hyperparameter k as a sequence of integers [1,2,5,8,10,15,20,30,50,100]. The output included a set of running time and mean MAE corresponding to the different hyperparameters.

```
def surprise_mem_base(df, s, hyparam):
    reader = sp.Reader(rating_scale=(1, 10))
    data = sp.Dataset.load_from_df(df[['userID', 'itemID', 'rating']], reader)
    data.split(s)

    result = dict()
    for nf in hyparam:
        print("%s :" % nf)
        sTime = time.time()
        model = sp.KNNBasic(k=nf)
        perf = sp.evaluate(model, data, measures=['RMSE', 'MAE'])
        #sp.print_perf(perf)
        #print("Running Time: %s sec" % (time.time()-sTime))
        result[str(nf)] = [perf, (time.time()-sTime)]
    return result
```

3.2.2.2. Model-based Model with HyperParameter Tuning

This algorithm based on SVD method in Surprise package. It had a similar design with the Neighborhood-based Model based on Surprise package. It also split the data into *s* parts, and the input data is in the form of (userID, itemID, rating). However, the hyperparameter for SVD was "n_factors".

This project designed the s-splitted as 5 and hyperparameter n_factors as a set of integers [1,2,5,8,10,15,20,30,50,100]. The output included a set of running time and a set of mean MAE referring to the different hyperparameters.

```
def surprise_model_base_0(df, s, hyparam):
    reader = sp.Reader(rating_scale=(1, 10))
    data = sp.Dataset.load_from_df(df[['userID', 'itemID', 'rating']], reader)
    data.split(s)

    result = dict()
    for nf in hyparam:
        print("%s :" % nf)
        sTime = time.time()
        model = sp.SVD(n_factors=nf)
        perf = sp.evaluate(model, data, measures=['RMSE', 'MAE'])
        #sp.print_perf(perf)
        #print("Running Time: %s sec" % (time.time()-sTime))
        result[str(nf)] = [perf, (time.time()-sTime)]
    return result
```

3.2.3. Baseline Model

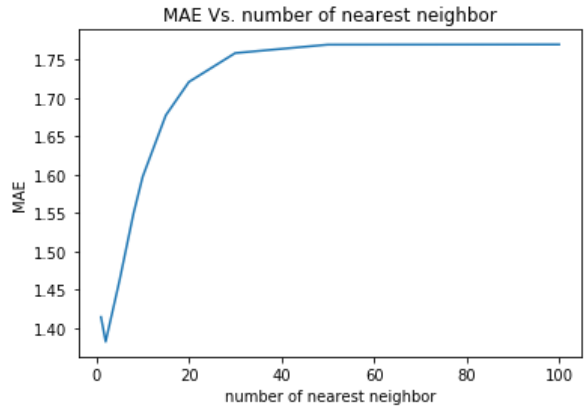
This project assumed that the users had the similar attitude and the external conditions (eg. weather, temperature, etc.) are also similar, the users should have similar rating scores while dating with others. Thus, based on this assumption, the baseline model was built by filling all the entries with the average rating of a certain female (row mean).

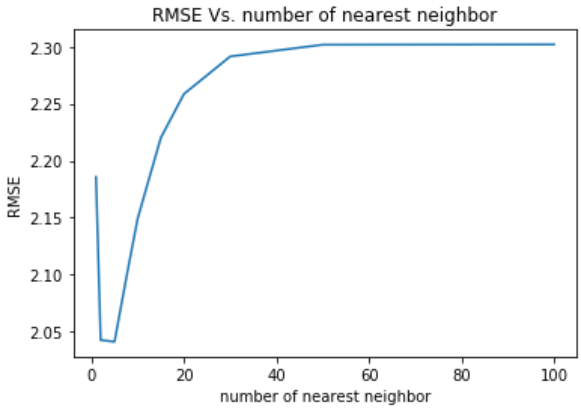
4. Result

4.1. Neighborhood-based Model (based on Surprise)

4.1.1. MAE and RMSE

Hyper Parameter K	MAE	RMSE
1	1.414221	2.1857
2	1.381972	2.04237
5	1.461870	2.04097
8	1.548849	2.10685
10	1.597334	2.14906
15	1.677612	2.22003
20	1.721239	2.25855
30	1.758892	2.29152
50	1.769887	2.30184
100	1.770113	2.30204



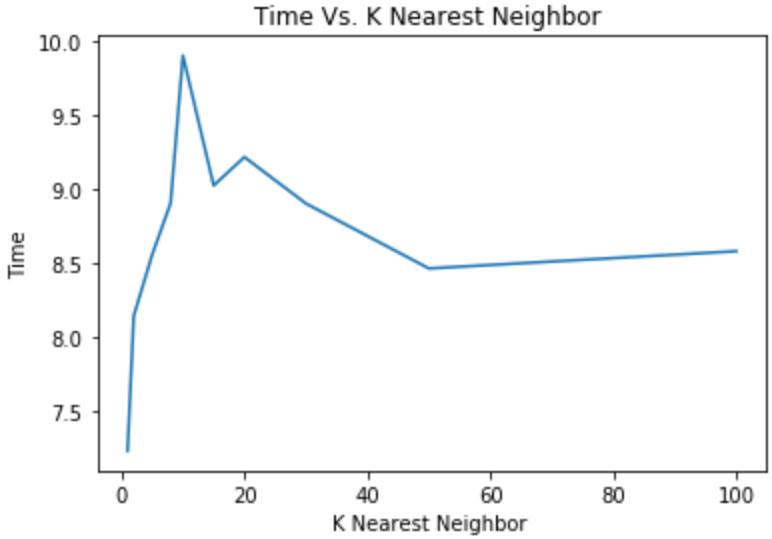


For both MAE and RMSE, as the number of k nearest neighbors included in KNN increase, the accuracy improved at first and then fastly dropped after the number of neighbors > 5.

Best hyper-parameter k nearest neighbors = 2 or 5, since the MAE was lowest when hyper-parameter k = 2 and the RMSE was lowest when hyper-parameter k = 5. For computation simplicity, nearest neighbors = 2 was chosen, for future testing.

4.1.2. Running Time

The following figures showed the relation between running time and hyper parameter k (nearest neighbor).



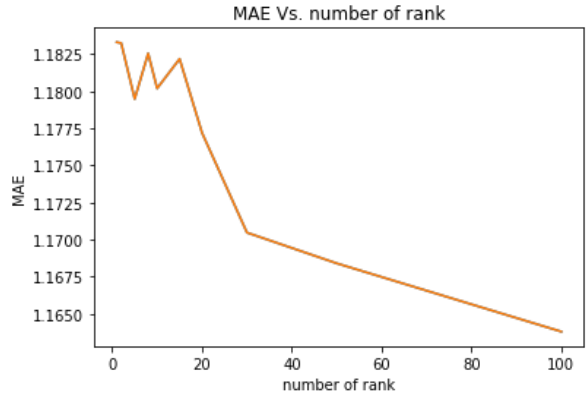
Time of computing Nearest Neighbor increased with a number of neighbors involved in calculation until k = 10, then the time needed for calculation decreased.

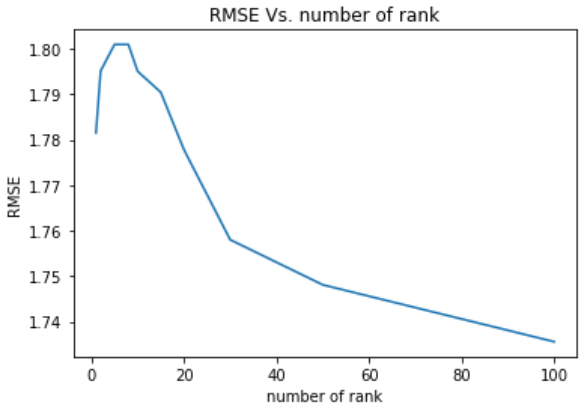
4.2. Model-based Model (based on Surprise)

4.2.1. MAE and RMSE

Hyper-Parameter n_factors	**MAE**	**RMSE**
1	1.183297	1.7815
2	1.183234	1.7951
5	1.179474	1.8009
8	1.182542	1.8009
10	1.180186	1.7950
15	1.182174	1.7904

20	1.177190	1.7778
30	1.170467	1.7580
50	1.168396	1.7481
100	1.163797	1.7356

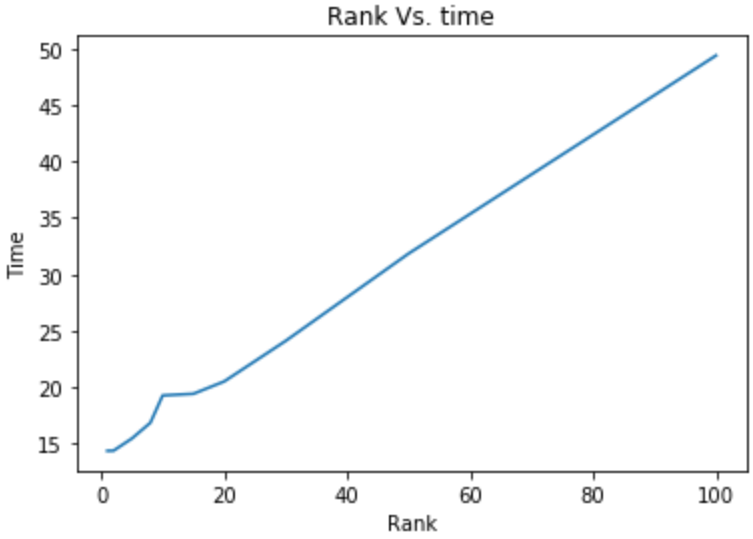




The more rank (n_factors) included in matrix reconstruction $R = UV$, the more accuracy attained for SVD, at the cost of computation time.

4.2.2. Running Time

The following figures showed the relation between running time and hyper parameter n_factors (rank).



Since the size of input data was relatively small, calculation of SVD with rank = 100 can be done in 50 sec. This project chose parameter n_factors (rank) = 100 to attain lowest MAE for further testing. However, it would no longer be applicable, when our input size was large.

4.3. Baseline RMSE and MAE

The baseline RMSE and MAE was computed by the baseline model (i.e. row mean for each entry) and the original sample data. The MAE was 1.1899 and RMSE was 1.8014 for baseline model.

5. Coverage Evaluation

The algorithm for coverage evaluation based on Surprise package. Thus, the input data should be in the form of (userID, itemID, rating). Besides, since the algorithms splitted data into 5 parts, there were 5 pairs of precision and recall values.

An item was defined to be “relevant” if the true rating was greater than 8. And the algorithm will recommend the item to the user if it satisfied two conditions -- its estimated rating was greater than 8 and it was within top 10 estimated ratings.

5.1. Neighborhood-based Model Coverage

1) Precision, Recall: 0.68, 0.03
2) Precision, Recall: 0.68, 0.04
3) Precision, Recall: 0.70, 0.02
4) Precision, Recall: 0.71, 0.05
5) Precision, Recall: 0.69, 0.04

The KNN model with Z-score had a precision of 69%. This referred that about 69% of top 10 recommendation is of users interests. Besides, this model had a recall about 4%. This meat that 4% of items is selected among all relevant.

5.2. Model-based Model Coverage

1) Precision,Recall: 0.82, 0.04
2) Precision,Recall: 0.81, 0.03
3) Precision,Recall: 0.80, 0.04
4) Precision,Recall: 0.82, 0.04
5) Precision,Recall: 0.81, 0.04

The SVD model had a precision of 81% recall of 4%. This meant that about 81% of top 10 recommendation is of users interests and about 4% of items were selected among all relevant.

In conclusion, the low recall rates in both models might be improved through changing to more complex algorithms. It will be the aspect needed to enhance the future project.

6. Further Discussion: Changing the size of input

Since MAE gave equal weights towards each error between the observed and the predicted without penalizing the outliers, the performance of this part of algorithms was analyzed by MAE.

6.1. Collaborative Filtering Algorithms with Different Input Size

The algorithms conducted 5-fold cross-validation on both neighborhood-based and model-based algorithms. The evaluations of each algorithm are as follows.

6.1.1. Neighborhood-based model with different input size

Input Size	Running time (sec)	MAE
25x50	0.0290	2.91
50x2500	1.12	1.91
100x5000	4.92	1.76
100x10000	10.2	1.38

6.1.2. Model-based algorithms with different input size

Input Size	Running time (sec)	MAE
25x50	0.0725	2.11
50x2500	6.52	1.19
100x5000	24.5	1.17
100x10000	55.1	1.16

6.2. Evaluation of C.F. Algorithms with Different Input Size

According to the above tables, the running time of both neighborhood-based and model-based algorithms increased with the size of the input. Moreover, higher accuracy (i.e. lower MAE) was achieved through increasing the size of the input matrix.

Additionally, there was an improvement on MAE by modifying KNN algorithm with conducting centralization (i.e. KNN with means). Assume a normal distribution of input data also enhanced the performance of KNN.

7. Conclusion

This project aims to optimize its running time and accuracy of our recommendation algorithms. The recommendation matrix produced by this project contains the existing ratings of male-female pairs and our predictive ratings of other possible pairs.

However, several issues still exist, while this project has sacrificed at this stage. First, due to the data insufficiency, specific information about users -- such as interest, location, and personality -- is unreachable. This leads to the difficulty of taking higher dimensions into the recommendation system. Especially for new users, the algorithms fail to have the basic assumptions. Therefore, if possible, high-dimensions analysis can lead to significant improvement in the accuracy of recommendation algorithms.

In addition, this project only concerns about the evaluations from males to females. Unfortunately, this ignores one important possibility that the female does not share the same interest in the male. Hence, it is pivotal to take the female evaluation into account.

Finally, the baseline model has had relatively low MAE and RMSE already. This is because the sample data had small variation and was centered around mean. A possible explanation for this issue was that the dating data was biased and not missing at random. The future projects should consider this problem. The future project should try other algorithms that can better capture the implicit information.

8. Reference

1. LibimseTi dating data: http://www.occamslab.com/petricek/data/
2. Surprise package: http://surpriselib.com/