Expedia Hotel Recommendations

Yunwen Cai (yc3388), Zihan Ye (zy2293), Anna Zhou (yz3220)

1. Project Objective

In this project, we used the Expedia Hotel Recommendations datasets to give recommendations of potential hotels users like. The dataset was given as a Kaggle data challenge and Expedia is interested to know which hotels users would prefer. We considered two algorithms: neighborhood-based and model-based collaborative algorithms. In order to get some tastes of how these algorithms work with datasets, we randomly selected 10,000 users search records and 100 hotel clusters. This selection, compared to 300 million records in original dataset, caused decrease in generalization. From the dataset, we got information of whether users clicked and booked hotels labeled by clusters. Our goal is to predict whether users will book hotels given the information we already had, and optimize the hotel recommendations to users.

2. Data Preparation

2.1 Data Sampling

We randomly selected 10000 unique users from the original datasets and fetched all their search results, which are over 300k records. We only looked into three columns, 'user_id', 'is_booking', 'hotel_cluster'. 'Is_booking' is a binary column, with 1 represents the user made a search and a booking, and 0 represents the user made a search and DID NOT book. Hotel_cluster is hotel cluster labels which range from 0 to 99. We used this information to build the user-item matrix. If the user booked the hotel or only search, 1 would be filled in the corresponding user-hotel entry. If the user did not search the hotel cluster, the entry will be 0. Thus our user-item matrix is a binary matrix, with size 10,000 * 100. There was no missing value for 'is_booking' variable in our sample dataset, so we did not have to deal with missing value problems.

2.2 Train-test datasets split

We uniformly and randomly assigned probabilities between [0,1] to each entry of user-item matrix. Then entries with probabilities larger than certain probability were selected for train datasets and the left entries were used as test datasets. For instance, if chosen probability was 0.2, the train/test ratio was 0.8/0.2. Test datasets were remained untouchable before cross-validation.

3. Models Implementation

3.1 User-based collaborative filtering

We applied user-based collaborative filtering algorithm on this dataset. Due to the nature of Expedia Hotel dataset, items, which are hotel clusters in this case, were independent of each other. Additionally, further information of each hotel cluster was not given. Thus, we concluded that there is not much interest to gain if we compared item similarity in item-based collaborative filtering algorithm. On the other hand, we expected that users may share some similarities in their tastes in choosing hotels.

3.1.1 Similarities

We used cosine similarities to obtain pairwise similarities, between the user with other 9999 users. Because the user-item matrix was a binary matrix, there was no need to scale the matrix.

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|_2 \|\mathbf{B}\|_2} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}},$$

A, B are 1 * 100 binary vectors for different two users.

3.1.2 Prediction

Then we ranked similarities in a descending order and pick up N users, who were most similar to predict the targeted users' hotel tastes. We used the following equation for prediction:

$$P_{u,i} = \frac{\sum_{k\ most\ similar\ users}(S_{u,k} * R_{u,i})}{\sum_{k\ most\ similar\ users} |S_{u,k}|}$$

$P_{u,i}$: *Prediction on hotel cluster i for user u*
$S_{u,k}$: *similarity between user u and other k most similar users*
$R_{u,i}$: *rates, 1or 0, k most similar users give on hotel cluster i*

The prediction on hotel cluster i for user u was calculated by the sum of rating top k users gave on item i. Each rating was weighted by similarities between user i and j.

3.1.3 Parameters

We then used correct metrics to evaluate the model by matching the predicted entries with entries in the test matrix . We counted the number of pairs of matches and divided it by total pairs. We chose five-fold cross validation. Each time we randomly assigned probabilities to user-item matrix to obtain train and test datasets. The overall accuracy was the average of five-fold cross validation.

There were two hyper-parameters in this algorithm: train/test split ratio and neighbor size k for cosine similarity calculation. We tried different common train/test split ratio, 0.7/0.3, 0.8/0.2, 0.9/0.1. For each train/test split ratio, we implemented k values 20, 50, 80.

3.2 Model-based Collaborative Filtering

We implemented a model based collaborative filtering by using Singular Value Decomposition (SVD). SVD is a well known matrix factorization technique that reduces dimension by factorizing an $m \times n$ matrix X into three matrices as $X = USV^T$, where U is an $m \times r$ matrix, S is an $r \times r$ matrix and V^T is an $r \times n$ matrix. The matrix S is a diagonal matrix containing the singular values of the matrix X. There are exactly r singular values, where r is the rank of X.

In our case, X was the 10000×100 user-item matrix. SVD was helpful to us because we had a very sparse user-item matrix with a lot of dimensions, so we needed to restructure the matrix into low-rank structure, and then represent the matrix by the multiplication of two low-rank matrices U and V, where the rows contain the latent vector. Specifically, the U matrix represented the latent vectors corresponding to users and the V matrix represented the latent vectors corresponding to items. By multiplying these two low-rank matrices together, we got a prediction matrix $\widehat{X}$ with missing entries in the original user-item matrix filled up. Therefore we can use the prediction matrix to predict the unknown values in the original user-item matrix by simply looking up the entry for the appropriate user-item pair in the prediction matrix.

Our goal is to factorize X into U, S and V^T as shown below, and get the prediction matrix by taking dot product of U, S and V^T. Moreover, we wanted to calculate the accuracy of prediction by taking the average accuracy of a 5-fold cross validation, and use the accuracy to tune the value of k, which is the first k $(0 < k < r)$ singular values in S we would like to keep. Tuning the value of k will give us the best rank-k approximation to X, and thus effectively reduce the dimensionality of our original space.

$$
\begin{array}{ccccc}
X & = & U & S & V^T \\
\begin{bmatrix} X_{1,1} & \cdots & X_{1,100} \\ \vdots & \ddots & \vdots \\ X_{10000,1} & \cdots & X_{10000,100} \end{bmatrix} & = & \begin{bmatrix} u_{1,1} & \cdots & u_{1,r} \\ \vdots & \ddots & \vdots \\ u_{10000,1} & \cdots & u_{10000,r} \end{bmatrix} & \times \begin{bmatrix} s_{1,1} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & s_{r,r} \end{bmatrix} & \times \begin{bmatrix} v_{1,1} & \cdots & v_{1,100} \\ \vdots & \ddots & \vdots \\ v_{r,1} & \cdots & v_{r,100} \end{bmatrix} \\
10000 \times 100 & & 10000 \times r & r \times r & r \times 100
\end{array}
$$

3.3 Baseline

The original dataset suffered from sparsity. Most of the entries of the unbalanced matrix were 0 while others were 1. We set the baseline to predict all entries as 0, indicating that user was not interested in any hotel cluster. For each train/test split and top k hyperparameters choices, we calculated the baseline accuracy by using a model that gives zeros to all entries and comparing its differences from test matrix.

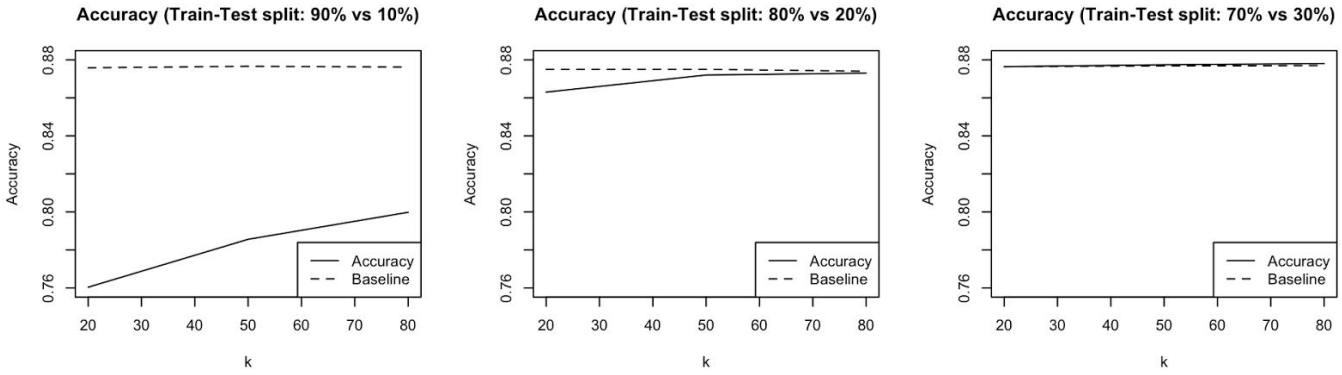
4. Evaluation

4.1 User based collaborative filtering

4.1.1 Correct Metrics

Correct metrics was used for accuracy calculation at first. We obtained the accuracy by using the percentage of correct predictions. It was the entry in test matrix sharing the same value in the predicted matrix.

For each train/test split, we ran user based collaborative filtering with k values of 20, 50, 80 and calculated the accuracy using 5-fold cross validation. The plots of accuracy versus k value are as follows:

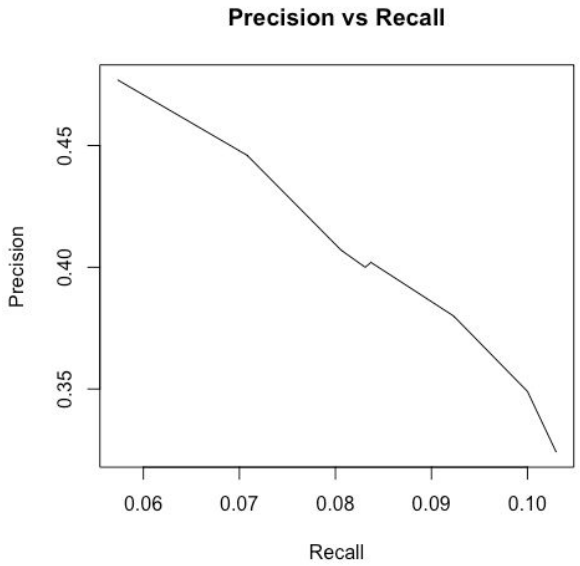


The highest accuracy with 90% vs 10% split is 0.8, at k =80, with baseline of 0.88. The highest accuracy with 80% vs 20% split is 0.873, at k =80 and the baseline accuracy is 0.874. The highest accuracy with 70% vs 30% split is 0.878, at k =80 and the baseline accuracy is 0.877.

Because the baseline was high, the range for model comparison was very small. Only when 70% vs 30% split, the model accuracy achieved the baseline. In all train-test split, the improvement in accuracy as k value increased from 50 to 80 was not that large. Because there was not much room for improvement and consideration for computing complexity, we did not continue to increase k value.

4.1.2 Precision and Recall

Additionally, we also used recall and precision to evaluate the model. At train-test split 80% vs 20%, recall vs precision is plotted as follows:



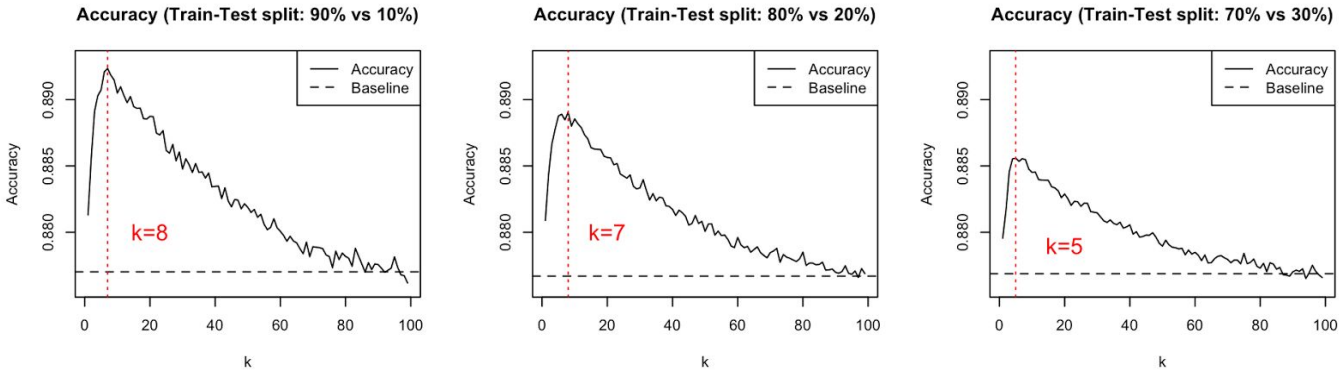
split	k	Recall	Precision
0.2	20	0.103	0.324
0.2	30	0.1	0.349
0.2	40	0.0923	0.38
0.2	50	0.0837	0.402
0.2	60	0.0831	0.4
0.2	70	0.0806	0.407
0.2	80	0.0708	0.446
0.2	110	0.0573	0.477

The plot shows the tradeoff between recall and precision. As k varies from 20 to 110, recall reaches it maximum 0.103 at k = 20. This recall values indicate, among hotel clusters users are truly interested in, the model can predict 10% correctly. This statistics adds additional explanations to results from correct metrics. The high accuracy of correct metrics is skewed due to the unbalanced user-item matrix, where most are zeros. Recall shows that the model does not perform well in correctly predicting hotel_clusters users are interested in. Thus, correct metrics accuracy is lower than the baseline for most cases.

Precision, on the other hand, reaches it maximum 0.477 at k = 110. Precision indicates that, among predicted matrix where entries are 1, the model predicted about 50% right.

4.2 Model based collaborative filtering

For each train/test split, we ran model-based collaborative filtering with k values in a range of 1 to 99, and calculated the accuracy using 5-fold cross validation. The accuracy vs k plot is as follows:



The highest accuracy with 90% vs 10% split is 0.892, at k=8. The highest accuracy with 80% vs 20% split is 0.889, at k=7. The highest accuracy with 70% vs 30% split is 0.886, at k=5. We managed to effectively reduce the dimensionality of our original space by using a small k value.

4.3 Coverage

In order to examine the coverage of our prediction and the original dataset, we looped over each user and marked down the 5 recommendations based on our prediction matrix. By taking the unique values of all recommendations, we were able to find out the number of hotel clusters covered. This evaluation procedure gave us a 100% coverage rate using the original matrix. User-based collaborative filtering also output 100 unique labels, which means that each hotel type will be at least recommended once to some

users according to our prediction matrix. For the prediction using model-based collaborative filtering, 88 out of 100 clusters were recommended to customers. For the purpose of this project, we hoped to build a recommender system that is able to give personalized hotel type recommendations to each user instead of just the popular items, so the resulted coverage with user-based and model-based collaborative filtering match with our expectations. Besides, we also compared 5 recommendations given by the prediction matrix and the original matrix for each user. It turned out that over 9200 out of 10000 users received the same recommendations if we use user-based collaborative filtering, and 9797 out of 10000 users if we use model-based collaborative filtering.

5. Conclusion

At this point, both of the collaborative filtering algorithms are able to produce a reasonably accurate hotel recommender system that gives customized and relevant hotel type options for users to choose. Comparing the above results, we would like to proceed with SVD model-based algorithm rather than the user-based algorithm in a real world setting. SVD model-based algorithm achieves high accuracy in correct metrics without losing too much of coverage. Also it scales better when dealing with datasets with millions of customers. Moreover, runtime is a huge concern in real world company. Model-based collaborative filtering takes much less time to produce outputs and tune key variables than user-based collaborative filtering. Thus it is favorable to companies. However, one thing we should watch out is that if we get a large k value as the number of ranks we intend to use for model-based collaborative filtering, we don't benefit a lot from using SVD to reduce dimensions.

6. Next Step

6.1 Other design choices

While building the user-item matrix, we also considered another way of coding user preferences. If user clicks the hotel cluster, but does not book, we encoded it as +1. If user clicks the hotel cluster and book, we encoded it as +2. If user does not click or book, we encoded it as -1. This design option can help to tell if user has interest in the hotel(+1) or not (-1). It can also tell how much a user has interest in the hotel, he/she might click or book several times.

6.2 Evaluation metrics

In current datasets, we weighted all 100 hotel clusters evenly. In larger dataset, the long tail can significantly affect accuracy calculation. Error will focus primarily on more

popular items, under-observed items are counted fewer times into accuracy. We will consider to use ranking accuracy when evaluating models by giving more weight on more popular items.

6.3 Accuracy change

Moving from small dataset to the large dataset, we expected the accuracy to increase. In the large size datasets, there is more freedom to choose neighborhood size. However, the sparsity will increase and cause the baseline to be very high.

6.4 Running time

User-based collaborative filtering can be very time-consuming dealing with large amount of users. It calculates pairwise similarities with all rest data, which cost $O(n^2)$ complexity and space, while n can be as large as 300 million in this datasets. The increase in complexity is almost exponential.
Model-based algorithm with SVD can scale better than user-based collaborative filtering. Its complexity for a m*n matrix is $O(\min\{m^2n, mn^2\})$. Since the n remains constant and smaller than m. Complexity would scale linearly with m, number of users in this case.

7. Reference

- Source of dataset: https://www.kaggle.com/c/expedia-hotel-recommendations/data
- https://cambridgespark.com/content/tutorials/implementing-your-own-recommender-systems-in-Python/index.html