For this question, I used two methods, and each one of them has its own advantages and disadvantages.

1. Multi-label classification.

By regarding the browsing history of each user as the user's 'labels', we can solve it as a multi-label classification problem. The user has the hotels he/she browsed as his/her labels. The algorithms analyze the connections between the 'labels' and the users' features, and are able to predict the labels for a given new user after training. I tried many algorithms in modules sklearn and skmultilearn for multi-label classification, including base classifier: KNN, SVM, random forest, naive bayes, adaboost; multi-label transformer: binary relevance, label power, classifier chain; multi-label ensemble: label space partitioning, RakelD, RakelO; partitioning clusterer: infomap, walktrap.

Since there are very limited possible combinations of the user/hotel features, this solution is actually for predicting the hotel/label distributions of different users based on their features, and then select a new recommendation from the distribution. I used 4 features home continent, gender, rating mean and rating variance of hotels the user browsed. Note I removed the user ID, because it induces unnecessary overfitting, and causes problem for the users' clustering (similar users' detection).

After many comparisons and grid search tunings, I picked ensemble (label space partitioning with infomap clusterer), label power and random forest combination for predicting multi-labels, and SVC, binary relevance combination for introducing noises to the labels by the probabilities it predicted (together as a kind of voting). By combining the results of the two algorithms on the full dataset together, we obtained a 'distribution' over the hotels for each user. Pick the highest one that they haven't browsed yet as the recommendation.

This idea in general is learning labeling distributions from the discrete signals, for the dataset's homogeneity. It can be generalized well for new user without training again. New user can get recommendation based on its 'properties', but for the browsing mean/var included, cold start is still a problem.

2. Recommender system.

This problem can intuitively be regarded as an unstandard recommender system problem, by taking the hotel website visited as preference (rating 1), and the hotel website not visited as 'temporary' no attraction (rating 0). Then with collaborative filtering (NN), or matrix factorization, we can infer each rating 'again' based the interactions among users. Briefly, for a certain hotel's rating from a certain user, we can do 'vote' within the user's neighbours, which are defined by the similarity measure. Although the unvisited websites are all initially zeros, the vote would give higher rating to the ones that are prefered by neighbors; the highest one is the recommendation. Since this is a pure recommender system base, the hotel/user's features are hard to be included through off-the-shelf functions in imported modules. I used the module Surprise for this problem, and tested on some generally high-efficiency algorithms, like SVD, CoClustering, NMF. After comparison through cross-validation and grid search, the SVD performs the best and the most stable. Without the user/hotel features, the information is pretty limited, so the result is highly homogeneous, with a certain hotel (#56) occupies over 75% in all recommendations. Unsurprisingly, the hotel website visited more is recommended more, and the new distribution is much more polarized than the original browsing history. I feel this kind of recommendation is actually safer, but may not capture users with strong personal preference.

The drawback is it focuses on the given users' recommendation, so for new user it needs additional more computation to give out a reasonable recommendation. The unincluded features are also big loss, but can be partially solved by customized recommendation system, using customized similarity function, or customized ratings.

3. Summary and achievement:

The two methods are the initial ones that come into my mind, which are based on different parts of traditional machine learning knowledge. They have different usage, so should be chosen depending on the business needs. For extension of multi-label classification, deep learning with multiple outputs is a powerful choice, but needs longer development period.

For multi-label classification after tuning parameters, the label space partitioning random forest gives 0.254 recall on hold-out set (25% browsing behavior can be covered by predictions); the SVM gives 0.0484 mean squared error on hold-out set.

For recommender system, the SVC outperforms CoClustering and NMF in stability and accuracy. After many attempts, the SVC reaches a rough optima with 0.052 cross-validated mean squared error on the whole dataset. With the tuned SVC I obtained the final predictions of the recommender system, which share 32.7% same predictions as the classification result. Therefore, the predictions of them are reasonable, except for the classification uses more information so the result is more heterogeneous.

If there's a more specific business context, it's easier to make choice of preprocessing methods, algorithms, and further techniques to improve accuracy and generalization.

Dependencies to run the codes:

1. Python modules: Surprise, Igraph, graph-tool, scikit-multilearn, scikit-learn, numpy, pandas, scipy
2. Put the homework folder and the codes in the same parent directory.

Note I recorded all my steps including testings in the codes, some parts don't contribute to the final result.