Expedia Hotel Recommendation System

Group 2 - Deep Data

Jozef Cibicek

Filip Ibl

Nino Khukhunaishvili

Eleftheria Kouppari

Eyad Al-Khayat

Charbel Merhej

Ana C. Rodríguez Rodríguez

ABSTRACT

The report explored the use of two different types of models to solve the problem of recommending suitable hotels for Expedia users. The first model dealt with the issue as a recommendation problem and devised an algorithm mainly based on the use of Matrix Factorization combining content-based and collaborative based approaches. The second approach handled it as a classification problem and ensemble four machine learning classification models using soft voting technique. The results of both approaches were compared and discussed.

KEYWORDS

recommender system, machine learning, hybrid models

1 PROBLEM DESCRIPTION

One of the main issues people encounter while preparing for a vacation is which hotel to book for their stay. With hundreds and possibly thousands of hotels to choose from at every destination, making a choice out of a few features such as price and pictures is not an easy task. This is why to provide their customers with a better booking experience, Expedia has uploaded a competition on Kaggle[1] with prizes for the top 3 results submitted. The competition's goal is to challenge researchers to contextualize customer data and predict the likelihood a user will stay at 100 different hotel groups. To evaluate the results, Expedia decided to use the Mean Average Precision @ 5 (MAP@5), defined by the following equation:

$$MAP@5 = \frac{1}{|U|} \sum_{u=1}^{|U|} \sum_{k=1}^{min(5,n)} P(k) \tag{1}$$

To tackle this problem, the group was divided into two subgroups where each worked on a different approach. The first subgroup implemented a recommender system, while the second handled the problem as a multiclass classification one. The goal of this was to explore and reflect on different techniques for solving this problem and efficiently use the effort of all the 7 members of the team. This not only enabled a smooth progression of the work, but also enabled everyone to acquire new knowledge.

[1]Expedia Hotel Recommendations - https://www.kaggle.com/c/expedia-hotel-recommendations

2 DATA OVERVIEW

The dataset of 'Expedia Hotel Recommendation' mixes categorical and continuous observations of customers behavior from Expedia.com. The given dataset contains 37M rows, making it challenging to apply algorithms that requires high computing resources. It is worth mentioning that the data in this competition is a random selection from Expedia full customers interactions and is not representative of the overall statistics.

Three CSV files were given in the competition: training, testing and destination datasets. The training and testing datasets were split based on time/date. The training included records from 2013 to 2014, while the testing data had 2015 records. Table 1 lists the 24 features that were identified in the training dataset. All data related to user and hotel information is anonymized by using ID's of entities.

The third file given was destination.csv, it included 149 anonymized further features about destinations where the hotel searches were performed and text description of each. This file data could be linked with the original dataset by using "srch_destination_id".

Due to the huge dataset initially given, we had to work with a small subset that we sampled from the original data to accommodate the limited computing resources we have. Samples of 1%, 5%, and 10% of the original dataset were created in a way that preserves the original distribution of the 100 hotel clusters. The first approach was able to work with the 10% while the second used the 1% sample. Figure 1 shows how the samples preserved the clusters distribution.

3 MODELS

In this section, the models that have been used for both approaches are described. First, the recommender system solution based on matrix factorization, and secondly, the multiclass classification problem that relies on an ensemble learning with soft voting.

3.1 Recommender System

The first approach to tackle the problem is based on a hybrid recommendation model proposed by [4]. This implementation uses both

Table 1: Dataset Features Description

Features	Description
date_time	Timestamp for record
site_name	ID of point of sale
posa_continent	ID of continent linked to site$_{name}$
user_location_country	User country ID
user_location_region	User region ID
user_location_city	User city ID
orig_destination_distance	Distance between customer and a hotel
user_id	User ID
is_mobile	1=>connected from mobile, 0=> other
is_package	if the click/booking is part of a package
channel	Marketing channel ID
srch_ci	Checkin date
srch_co	Checkout date
srch_adults_cnt	Number of adults specified
srch_children_cnt	Number of children
srch_rm_cnt	number of hotel rooms
srch_destination_id	ID of destination
srch_destination_type_id	Type of destination
hotel_continent	Hotel continent
hotel_country	Hotel country
hotel_market	Hotel market
is_booking	1 if booking, 0 if a click
cnt	Number of similar events in the same user session
hotel_cluster	ID of a hotel cluster

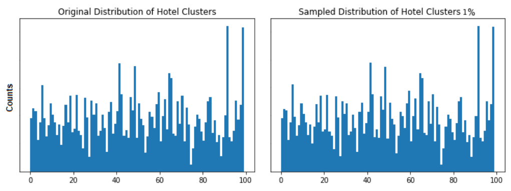

Figure 1: Original and Sampled Distribution of Hotel Clusters

Content-based Model and Collaborative Filtering to make recommendations. Nature of this approach did not require to immediately solve the problems with missing values in some features as in the case of second approach.

Content-based models make recommendations of items based on the information given by user ratings. We computed local popularity of hotel clusters based on user ratings.

The idea of Collaborative Filtering is that similar users, e.g. users who have the same features, are most probably going to like the same items. Therefore, Collaborative Filtering tries to find similar users and makes recommendations based on them.

Both filtering approaches have its limitations, therefore the Hybrid model implemented in [4] combines these two models by considering both hotel popularity in input destination and user preferences. Such models are already used in similar problems, with great success, and it seems to be a promising approach.

The structure of the hybrid model is demonstrated in Figure 2. The output matrix of content-based filtering is represented by blue field *Destination Top Hotel*. Output of Collaborative filtering is represented by *SVD Clustered Utility* field. Both fields are then combined to make a final recommendation.

The right side of the figure includes decision tree which solves the cold-start problem. More about this part of the system can be found in Cold-start problem section.

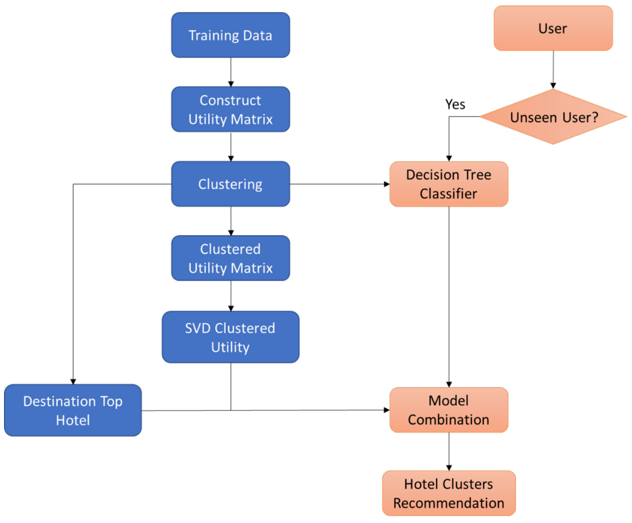

Figure 2: Recommender System Structure [4]

3.1.1 *Collaborative Filtering*

The first step is to create a matrix which would give us information about user preferences. We create a Utility Matrix M that will give us a user-hotel pair. The user ID is used to represent the users and hotel cluster number is used for the hotels. When a user i has viewed a hotel j, a rating of 1 is given to the ith row of the jth column, while if the user has booked the hotel, a rating of 5 is given. The resulting utility matrix is very sparse, as not a lot of users have booking history. Therefore we applied matrix factorisation to calculate the missing values by optimising the reconstruction loss with gradient descent.

In order to reduce the huge number of users, it was proposed that it would be better to cluster the user IDs. This was beneficial to have a much more compressed utility matrix that would allow to apply Matrix Factorization efficiently. Hierarchical Clustering was used to perform this step. The data were first normalised, because Hierarchical clustering uses cosine distance, which requires normalisation. The total number of clusters we specified was 112 that preserved maximum variance between clusters. Other clustering algorithms were tried without any significant improvement on both speed or accuracy.

Right after, we have an utility matrix with the rows being the user clusters and columns the hotel clusters. Unfortunately, this step presents the biggest overhead, and limits the training of the model only on a 10% of the training data, when using a virtual machine of 8 CPUs, 30GB RAM and a Nvidia Quadro P5000 GPU.

The clustered Utility Matrix remained sparse even after Clustering, as it is uncommon for all user clusters to rate all hotel clusters. Singular Value Decomposition and Stochastic Gradient Descent were used to fill the unrated entries and minimise the produced error.

We constructed a low-rank matrix X, so that $X = UV^T$. U is a left singular matrix with dimensionality $N{\times}C$ where N is a total number of customers. V is a right singular matrix with dimensionality $K \times C$ where K is a total number of hotel clusters. C denotes the rank of matrix M. Dimensionality of matrix X is $N \times K$.

We used gradient based approach to matrix factorization to complete the matrix.

Both matrices U and V were initialized with small random values and then iterated with following update rules to estimate their values (α is a learning parameter and λ is a regularisation parameter):

$$\begin{aligned} U_i &= U_i + \alpha((M_{ik} - V_k^T U_i)V_k - \lambda U_i) \\ V_k &= V_k + \alpha((M_{ik} - U_i^T V_k)U_i - \lambda V_k) \end{aligned} \tag{2}$$

User's i recommendation for hotel cluster k was then calculated by computing

$$U_i V_k^T \tag{3}$$

3.1.2 Content-based Model

To compute a local popularity of hotel cluster, we created a new matrix D, which represents the item properties.

$$D = \begin{vmatrix} d_{11} & d_{12} & .. & d_{1k} \\ d_{21} & d_{22} & .. & d_{2k} \\ .. & .. & .. & .. \\ d_{j1} & d_{j2} & .. & d_{jk} \end{vmatrix} \tag{4}$$

The matrix contains j rows denoting the hotel destinations and k columns denoting hotel clusters. Its values are the average of all ratings of a hotel cluster on a certain destination, where bookings are rated as 5 and clicks as 1. Matrix D represents the popularity of a hotel in a destination.

3.1.3 Combination

Finally, both matrices M and D were combined together. We combined the user preference with the item properties, by performing element multiplication on the vectors of the matrices M and D, resulting in a new matrix R of dimensions NxK, where N is the number of users with booking history and K the hotel cluster. Matrix R combines the user preferences that are in M and the popularity of the hotels in destination of D.

$$R_{ik} = M_k^i D_k^j \tag{5}$$

Index i denotes user ID and index j denotes destination ID, k is hotel cluster.

According to the original implementation [4], the combination of Collaborative Filtering and Content-based Model increased the performance by 4% in comparison to the purely content-based approach. Matrix R will give the top 5 hotels for each user, by sorting each row based on the ratings, and returning the 5 highest rated hotel clusters.

3.1.4 Cold-start problem

Recommender systems come with the well known problem of cold start. In order to overcome this, the model uses ontology decision. According to the theory of ontology decision, users with similar profile information, like age, gender, location, class etc., are likely to have the same preferences and thus, the same behaviour.

In our case the decision tree is trying to cluster the not yet seen users to one of the user clusters.

3.1.5 Results

We were able to run a 10% of the training data to train our model, on a virtual machine of 8 CPUs, 30GB RAM and a Nvidia Quadro P5000 GPU. The model requires a lot of computational power and memory to run, and unfortunately we did not have a more powerful machine to train the model. The testing was conducted on unlabeled test dataset and the results were submitted to Kaggle to receive the score. We were able to reach a MAP@5 13 or 14%.

3.2 Multiclass Classification Problem

The current challenge can also be considered as a multi-class classification problem with 100 different classes (i.e., hotel clusters). The aim is to recommend the top five hotel clusters that users are more likely to book or search for, making use of supervised methods. The proposed approach includes the combination of four different models: Random Forest, Naive Bayes, XGBoost and Multi-Layer Perceptron (MLP).

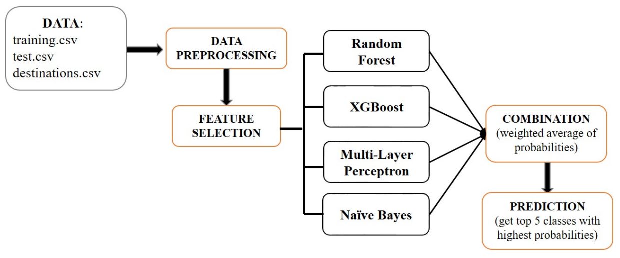

Figure 3: Approach Summary

The figure above shows the process that has been followed to tackle the problem. Considering that the test set does not include target values, and, therefore, no performance measure can be derived from it, it has not been used for this project. Furthermore, due to the large amount of training data (i.e., more than 30 million of instances) and the limitation on the resources, only 1% of this data has been processed (around 300,000 rows). The next sections will expose in detail each of the different steps: data preprocessing, feature selection, modelling and models' combination.

3.2.1 Data Preprocessing

The training set (1% of the original dataset) counts with 376,703 instances and 24 features related to users and destination information. One of the main issues with data preprocessing is handling missing values. There were 3 features related to dates: *date_time*, date of the booking or search, *srch_ci*, check-in date, and *srch_co*, check-out date. These features were transformed into datetime objects in order to extract their month and day, leading to new features: *booking_month*, *booking_day*, *ci_month*, *ci_day*, *co_month* and *co_day*. On the other hand, these date type features were used to engineer some new ones, as they were considered to add relevant knowledge for the models: *duration* of the stay, represents the difference between check-in and check-out date, *remaining_days*, number of days between the booking date and the check-in date, and, *is_weekend*, binary variable equal 1 if the booking dates include a weekend or 0 otherwise.

Another important matter in data preprocessing is handling missing values. There were three features that presented missing values: *srch_ci*, *srch_co* and *orig_destination_distance*. For the first two, considering the low proportion of instances that presented missing values (i.e., 0.11%), the proposed solution was to remove them. However, the latter feature was found missing in more than 35% of the instances. Since this feature represents the distance between users' and destinations' locations, filling the missing values would have significant influence on the overall accuracy of the hotel clusters' prediction. The proposed approach to impute the missing distances was to predict them making use of the following features:

- site_name (ID of the Expedia point of sale (i.e. Expedia.com, Expedia.co.uk, Expedia.co.jp, ...))
- posa_continent (ID of continent associated with site_name),
- user_location_country (The ID of the country the customer is located),
- user_location_region(The ID of the region the customer is located),
- user_location_city(The ID of the city the customer is located),
- srch_destination_id (ID of the destination where the hotel search was performed),
- hotel_continent,
- hotel_country

Ultimately, a different method was applied – a random forest regression to achieve more coverage. Random forest regression ('sklearn' library – 'IterativeImputer') was trained using all the features given above with tuned parameters that were obtained using GridSearch. The categorical features were not encoded since tree based models can handle this type of features on its own. Square root from MSE, calculated as square of the difference between actual distance and the corresponding distance that would have been predicted by the model if this distance were missing, was 47% of the average *orig_destination_distance* in the training dataset. Therefore, the mistake is on average 47% of the mean. This is quite a good result compared to imputing every missing value with a single measure such as median of *orig_destination_distance*.

There are also 150 latent features regarding each destination that represents comments and reviews by customers (in *destinations.csv*). Due to the high number of features the dimensionality reduction technique PCA was applied. Figure 4 illustrates the cumulative explained variance versus the number of principal components. A cumulative explained variance of 0.92 was considered as a good representation of the original features, and, therefore, 40 components were derived.

These 40 components for each of the 62,106 destinations were merged by destination id with the 1% of the training set resulting in a total of 67 features.

3.2.2 *Feature Selection*

When dealing with a considerable number of features, it is important to apply feature selection techniques so that features that do not contribute to model are discarded. In this case, feature importance of the 67 features resulted from the data preprocessing

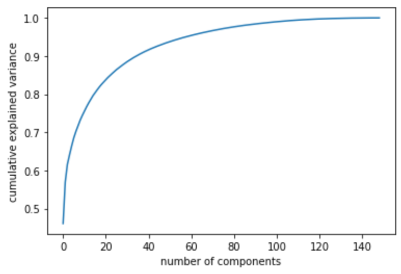

Figure 4: Cumulative Explained Variance VS Number of Components

step. Figure 5 shows the top 10 features with the highest feature importance:

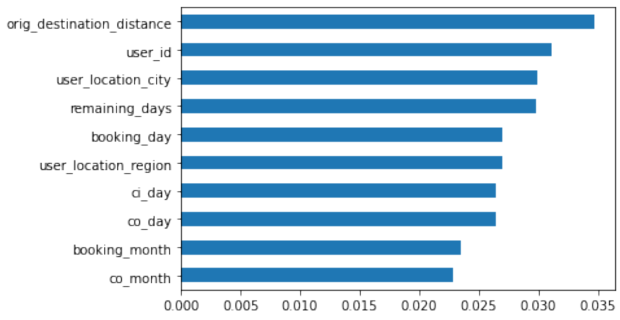

Figure 5: Top 10 features

As it can be observed, the feature *orig_destination_distance* presents the highest value, followed by *user_id*. In general, top features relate to users' location and dates regarding bookings (i.e., booking date and check-in/check-out dates). Furthermore, one of the engineered features, *remaining_days*, is also included as one of the most important features of the data. In order to decide which features were irrelevant, and therefore, could affect the accuracy of the model, a threshold for feature importance of 0.01 was set. This led to removing nine features: *cnt*, *srch_adults_cnt*, *srch_children_cnt*, *is_package*, *is_weekend*, *posa_continent*, *is_mobile*, *srch_rm_cnt*, *is_booking*.

3.2.3 *Models*

In order to build the different models and tune their parameters, a training/test split from the original training data has been applied. Note that the original testing set did not include the target values for hotel clusters. From the 1% of the data, 80% has been assigned as training data, and 20% as test data. In this line, 4 different models have been trained: Random Forest, Naive Bayes, XGBoost and MLP. These models will predict probabilities of each hotel cluster to be recommended to each user, so that, the top 5 are chosen. The evaluation metric is MAP@5, which computes the proportion of users that its target value (i.e., hotel cluster value in test set) is included in the top 5 recommended clusters by the models.

- **Random Forest**
  This model is in nature an ensemble of decision trees that

can identify both non-linear and linear patterns on data. One of its advantages is that it can handle both categorical and numerical data, with no need of specific transformations or encoding [1]. The main difference between Random Forest and a pure ensemble of decision trees is the extra randomness it applies as each decision tree is trained with a random subset of the training set. It can be Bagging or Pasting, depending on whether same subset can be used to train more than two classifiers or not, respectively. Random Forests count with several parameters that require to be tuned in order to get all their potential, such as the number of estimators and maximum depth of the trees. In this case, it has been done through Grid Search, making use of all 58 features. Additionally, training was done by chunks of 20,000 rows, due to limitations on the resources. This model provided a map@5 metric of 12.354.

- **Naïve Bayes**
  It is a probability based model that assumes independence between features and it is widely used for classification tasks for its fast training. There are different implementations of its principles that are suitable to use depending on the nature of the data [3]. In this case, since data counts with both categorical and numerical features, the proposed approach is to use two different Naive Bayes models for each of these types of variables. A multinomial Naive Bayes implementation has been used to train the categorical features and a Gaussian implementation for the numerical ones. The predicted probabilities for the different 100 classes of each implementation have been multiplied, which is a valid approach regarding the assumption of independence between features that was mentioned earlier. The combination of these two implementations led to a lower map@5 of 7.885 in comparison to Random Forest.
- **XGBoost (Extreme Gradient Boosting)**
  It is one of the most popular machine learning algorithms, famous for frequently outperforming other ML alternatives. XGBoost is a tree-based method which uses gradient boosting framework. Boosting refers to Ensemble methods that try to build a strong predictor from several weak predictors. Overall idea of boosting algorithms is to build models/trees sequentially in such a way that every new model improves the previous model by fitting a new model to the residuals from the prior step [2]. It is worth mentioning again that besides its outstanding performance, a tree-based algorithm is a good option for an additional reason. It does not require feature encoding and prevents the curse of dimensionality issue. Due to computational constraints, we were not able to run GridSearch and used hyper parameters which we thought would give good results. Some of the tuning parameters used for XGBoost are the following: (i) number of estimators/trees is equal to 300. In XGBoost this is an important parameter. Information about the previous tree is used for growing a current tree and increasing the number of models can lead to overfitting. (ii) 'Subsample'=0.7, which means that a random selection of 70% of the training data is used to train each tree. Higher subsample fraction leads to lower bias, but slows down the training speed. (iii) Maximum depth of a tree, number of nodes on the largest path from the root to the deepest node, is equal to 5. (iv) 'Colsample_bytree', fraction of columns/features to be considered for building each tree, equals 0.7. Depth and fraction of features are additional parameters used to avoid overfitting. (v) Learning_rate/shrinkage (0.01 in this case) decreases the impact of individual trees and leaves room for future trees to improve the model. Therefore, lower values of learning rate require higher number of trees but predictions usually generalize better [2]. These parameters led to XGBoost achieving a map@5 of 14.4 for this problem.
- **Multi-Layer Perceptron (MLP)**
  Feed-forward neural networks (NNs) or Multi-Layer Perceptrons are another very powerful machine learning algorithms. NNs are well known for their exceptional performance for different applications, especially for image and speech recognition systems. Neural networks are also used in recommendation systems (e.g., You Tube). MLP worked well for this classification problem. It has a map@5 equal to 19.9%, which is the highest one compared to the other three models. Before feeding into a network variables were preprocessed. This included standardizing them and using one-hot encoding for categorical features. Some of the parameters were dictated by the problem. Namely the model has a softmax output layer with 100 nodes (as there are 100 classes to be predicted). However, there are not many principles to follow while choosing the design of the network or the activation function. Therefore, the process of training involves trying different options and experimentation. For this problem the following specifications worked the best: the model has two hidden layers, the first layer with 200 nodes and the second one with 100 nodes. Rectified Linear Units (ReLU) are used as an activation function for hidden units. Dropout is equal to 0.5. This means that while training separate networks at each training step, 50% of nodes together with its input and output connections are randomly removed. Dropout is one of the most frequently used regularization techniques for avoiding over fitting of the resulting combination of networks.

3.2.4 Combining models

Often winning solutions in machine learning competitions involve ensemble of methods. In case of classification problem, a simple way to improve the model outcome is to aggregate the results by majority voting. In other words, by choosing the class which is predicted by most of the classifiers (hard voting). This kind of aggregation often achieves even better performance than the best classifier in the ensemble. If there are sufficient number of diverse classifiers, even weak learners (slightly better than random guessing), it can produce a strong ensemble classifier. Furthermore, if all of the models can produce class probabilities (which is the case in our problem), it is possible to predict the class with the highest probability. Class probability is calculated as average of all probabilities of individual classifiers. This is called soft voting. It often achieves higher accuracy than hard voting because it gives higher weight to confident votes [2]. For our problem we chose soft voting but instead of calculating a simple average, we used weighted average of class probabilities. In other words, we assigned different weights to different models in order to take the ‘quality’ of a model into account. We selected the weights from grid of values based on their performance on the test set. Combining the models this way, resulted in a slight improvement of overall accuracy to 20.12%. The next figure summarizes the accuracy of each of the models individually and their combination:

	RF	NB	XGB	MLP	COMBINATION
Map@5	12.354	7.885	14.4	19.9	20.12

4 DISCUSSION

4.1 Comparison with Benchmark

In order to have some sense how good our result is, we need a benchmark to compare it to. One of the options is to predict 5 most frequent hotel clusters in the same order for each case in the data set and consider corresponding MAP@5 as a benchmark. For 1% data set, this type of guessing gives MAP@5 equal to 5.4%. The result would be different if we picked different sample so it is a better option to consider expected value of MAP@5 instead. In order to calculate expected value of MAP@5, we need to calculate expected value of increase in nominator of MAP@5 for each case first. Denominator is the total number of cases in the data set and is fixed. If the first guess is correct MAP@5 nominator is increased by 1, if second guess is correct it is increased by 1/2 and so on by 1/5 if the fifth guess is correct. So expected gain considering each case separately would be

$$E = 1 \times P1 + 1/2 \times P2 + ... + 1/5 \times P5 = 5.3\% \quad (6)$$

where E- is the gain for an individual case, and P responds to the value of the Pth guess is correct taken from the full data set.

If we have one thousand cases, expected gain would be 0.053*1000 but the denominator of MAP@5 is also one thousand and they would eventually cancel, the expected value of the overall MAP@5 is 5.3%. This means that both approaches achieved satisfactory results. The techniques are considered good options for this type of problem and more work can be directed to refine them and increase their accuracy as discussed below.

4.2 Challenges

The most noticeable challenge was the large dataset of this competition. It was the main cause of not being able to get better performance in both approaches as the models were only trained on 10% for the first approach, and 1% for the other one. The trick that enabled the recommender system to be trained on more data was by grouping users into clusters. This saved up some memory space that allowed for the increased training data percentage.

4.3 Comparing Solutions

With the first approach, we were able to submit our model to Kaggle for scoring as the algorithm flow took into account the problem of unseen user ids in the testing set by predicting the user cluster, while the performance of the second approach was done by splitting the training dataset by time and test the model on it. This is why it is hard to know which approach was better than the other as they were tested differently. The winning solution on Kaggle achieved a MAP@5 of 60%, however it is worth noting that this result was obtained by using an underlying latent leakage in the dataset that enabled to uncover relations between records and the recommended clusters in the testing data. We did not use this data leakage in any of the approaches. Moreover, the winning solution had access to more computing resources and used the whole dataset for training the model.

4.4 Improvements

To achieve better results, we may first try to train the models on the whole training dataset that is available. This would certainly improve the achieved MAP@5 score for both approaches. The next thing we can do is to combine the two approaches together, using the ability of the first one to deal with unseen users and power of assembling several models in the second approach. Another thing we could explore is the ability to use iterative machine learning models instead of models that require the training data to work. This would reduce the dependence on using more computing resources to deal with the large dataset.

5 CONCLUSIONS

The "Expedia Hotel Recommendation" problem was tackled with two different approaches (i.e., Recommender System[2] and Multi-class Clasification[3]) which resulted in varying scores. In this implementation, the combination of classifiers from the Classification approach outperformed the Hybrid Model of the Recommender solution by a MAP@5 of 4%. However, it should be taken into account that these models were trained only on a small sample of the overall data (10% and 1%).

REFERENCES

[1] Archana Chaudhary, Savita Kolhe, and Raj Kamal. “An improved random forest classifier for multi-class classification”. In: *Information Processing in Agriculture* 3.4 (Dec. 2016), pp. 215–222. ISSN: 22143173. DOI: 10.1016/j.inpa.2016.08.002.

[2] Aurélien Géron. *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition.* 2019, p. 838. ISBN: 9781492032649. URL: http://shop.oreilly.com/product/0636920142874.do.

[3] Daeun Jung and Hyunggon Park. “An Iterative Algorithm of Key Feature Selection for Multi-class Classification”. In: *International Conference on Ubiquitous and Future Networks, ICUFN.* Vol. 2019-July. IEEE Computer Society, July 2019, pp. 523–525. ISBN: 9781728113395. DOI: 10.1109/ICUFN.2019.8806074.

[4] Jing Wang, Jiajun Sun, and Zhendong Lin. *Hotel Recommendation Based on Hybrid Model.* Tech. rep.

[2] Implementation of Recommender System - https://github.com/joc32/Expedia_Hotel_Recommendations

[3] Implementation of Multiclass Classification approach - https://github.com/anarguez/expedia-recommendations