Trivago RecSys Challenge 2019

Jaechan Park
Johannes Kepler University
figo721@hotmail.com

Teunis Isaak Timmermans
Johannes Kepler University
teuntimmermans@hotmail.com

ABSTRACT

In this article a recommender system for the Trivago RecSys challenge is presented. During this a look is taken into undiscovered values in the dataset. The recommendater algorithms *K-Nearest Neighbor* and *Recurrent Neural Network* are used to recommend new items and some unique post-processing techniques are described to increase the validity of the recommendations.

KEYWORDS

recommendation system, neural networks, trivago RecSys challenge 2019

1 INTRODUCTION

The goal of this work is to develop an optimal session-based recommendation system of accommodations for the Trivago RecSys Challenge of 2019. This will be done by attempting popular algorithms or defining own algorithms. It is challenging to provide the best recommendations for all users and to meet all needs from users/customers. In order to provide an optimal solution of recommendation items based on session-based and context-aware, choosing a suitable algorithm and building a model are the most important tasks. In this work, we made an approach to a simple and less time-consuming algorithm, *K-Nearest Neighbor*, to a more complex algorithm, *Recurrent Neural network*, for the sake of finding an optimal algorithm of recommendation system and building a model for the given task. Both of these are combined with other post-processing methods to deliver the best possible results.

2 DATA ANALYSIS

Raw Dataset

Trivago provides a set of training data and another set of test data, they both contain the users' activities and related accommodation information while the users searched for one or more their attractive accommodations. In the training set, the total number of samples is around 16,000,000 and it consists of 12 features/attributes which are *user_id*, *session_id*, *step*, *reference*, *action_type*, *platform*, *impressions*, *prices* and etc. Among the features in the training set, *impression* feature provides a list of recommended accommodations at the time of a user made an action of *clickout item* in the feature of *action_type*. The *prices* feature represents its corresponding prices of the list of recommended accommodations at the time of *clickout item* by a user. In addition, the *step* feature provides information about how many steps took to make an action of *clickout item* by a user. Trivago also provided a dataset which contained metadata for most of the accommodations.

Train and Test Split

Due to limited computing resources and time-effective, we split the given the raw train dataset into two sets. The first 12,000,005 rows are assigned to a new training data set, whereas the remaining rows are used for the new test data set.

3 ALGORITHMS

For this task, due to stability and quality of performance on algorithms, we implement two main algorithms, *K-Nearest Neighbor* and *Recurrent Neural Network*, from thh sars github repository[1]. In the *Recurrent Neural Network*, *Session-Based RNN* and *Personalized RNN* are implemented for this task.

4 EVALUATION

Mean Reciprocal Rank

In this work, the measurement basis for recommendation accuracy uses emphMean Reciprocal Rank (MRR). *Mean reciprocal rank* is for evaluating any process that returns a ranked list of possible response to a sample of queries. The reciprocal rank is the multiplicative inverse of the rank of the first correct item. A strength of this evaluation method is that every item does not have to be rated.

$$MRR = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{rank_i} \tag{1}$$

5 DATA PRE-PROCESSING

Data Sorting

The fundamental goal of this work is to build a session-based recommendation system to exposure attractable items to users in order to save time and effort to find potential interesting items for users' convenience. As well, companies could expect to maximize their profits by recommending the most attractable and customized items to motivate more purchasing items. Based on this fundamental goal, a strategy has been determined that the *clickout item* in the *action_type* feature can be treated as the most significant action by users.

[1]https://github.com/mquad/sars_tutorial

Along with the previous assumption, we sorted out the raw training dataset by selecting only *clickout item* in the feature of *action_type*. After sorting out *clickout item*, the next approach is to know how many steps have been made the users before they made an action of *clickout item.* As stated previous, the fundamental goal of building a recommendation system is to offer a series or list of potential attractable items to users/customers to motivate purchasing more products while they are online. Hence, a key aspect of building a recommendation system is that the system has to be able to provide a list of recommended items to users immediately on time. Based on this motivation, we determine that it is crucial to know how many steps and efforts a user made to find his/her interesting item. If an optimal recommendation system provides recommendation items for a target user immediately, he/she will make fewer steps before the action of *clickout item.*

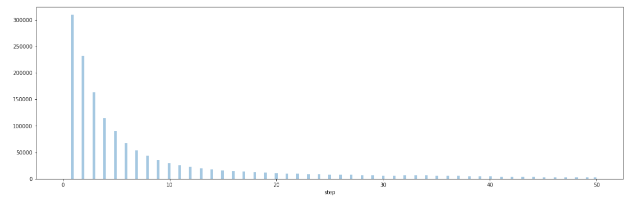

Figure 1: Frequency histogram of *step*

In figure 1, the x-axis represents the number of steps the users made before they made an action of *clickout item.* The y-axis is for the frequency of each step. It is obviously determined that the frequency histogram distribution is very right-sked and around the first 20 *step* dominate the major proportion of this distribution.

Log-Transformation and Pareto distribution

A log-transformation on the frequency number of each step has been applied in order to address the right-skew distribution and reduce a significant difference of the number for the frequency of each step. For this work, a natural log-transformation uses on the number of frequency for each step and then computes its cumulative percentage. As mentioned in the previous part, it is essential to find a meaningful and reasonable the step where it is the decision boundary to cutting-off the minor portion of the cumulative percentage in this frequency histogram. The Pareto distribution/principle uses to support our boundary decision of cutting-off. Based on the Pareto distribution (80-20 rule), it is required to find the *step* where the cumulative percentage has at least more than 80%. As shown in figure 2, the skyblue line represents the empirical result, while the dashed line is for the theoretical line based on its mean and standard deviation. Around after the *step* of 20 covers more than 80% of the cumulative percentage according to figure 2. In this work, we made an agreement to set the decision boundary step as to the 30 in order to less constraint of cutting-off.

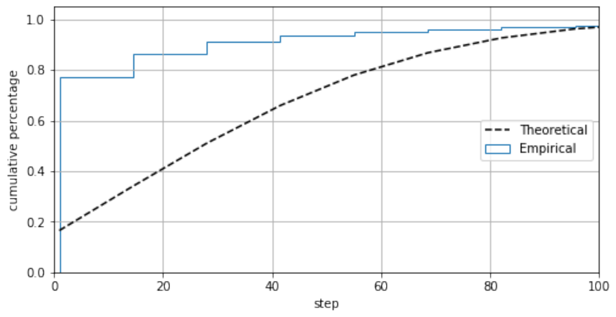

Figure 2: Frequency histogram of *step*

Feature Engineering by the Zipf's Law

Zipf's[2] law is a statistical distribution widely uses in the linguistic area, that the frequencies of certain words have its relationship of inversely proportional to their ranks. Zipf's law can be represented as the rth most frequent word/term has frequency f(r) that scales can be assigned to

$$f(r) = \frac{1}{r^a} \quad for \quad a \approx 1 \tag{2}$$

In this work, a new feature has been created by applying the Zipf's law on the log-transformed frequency number. The name of a created feature assigns to *searching satisfaction rating* and the rating scale is from 0-5 where 5 rating is the most satisfactory level and 0 is the lowest rating. Each step has been assigned based on Zipf's law and its log-transformation frequency value. The result can be seen in figure 3.

	Range
5 Rating	step 1 - step 3
4 Rating	step 4 - step 6
3 Rating	step 7 - step 12
2 Rating	step 13 - step 24
1 Rating	step 25 - step 30
0 Rating	> step 30

Figure 3: *searching satisfaction rating*

Bipartite graph for locations

There is another feature extracted from the test dataset. Each clickout session has a location linked to it and this location consists of a city and a country. Using this information the accommodations are put into a list which contains all of the cities that they were seen with. This list can be translated to a bipartite graph, with on the one side the cities and on the other side the accommodations. This graph will be used in the post processing with a simple bipartite graph algorithm.

6 *K*-NEAREST NEIGHBOURS

To build a recommendation system in this work, the first implementation algorithm is *K-NN* which stands for *K-Nearest Neighbor*. *K-NN* is one of popular algorithms for classification because of simple to use, learning incrementally and representing arbitrarily complex decision boundaries. Also, as *K-NN* is a non-parametric method for classification or regression, it is unnecessary to consider any assumptions to build a model. To design the optimal accuracy model with the *K-Nearest Neighbor* algorithm, choosing an optimal *k*, which is the number of the closest neighbor, is the most crucial part of the training phase. The reason is that too small number of choosing k may cause high bias during the training phase, whereas too large number of k may lead to an undesirable outcome of classification. Therefore, it is essential step to find the right *k* number of this algorithm. In this work, we use four different *k*s for the sake of finding the optimal accuracy by comparison. In addition, *Jaccard* distance[1] has been used for measuring the distance of the *k* nearest neighbors. Although the measurement of accuracy is based on the *Mean Reciprocal Rank*, the other two results are quite informative to understand and make an interpretation depends on algorithms. The *Precision* can be described as the proportion of positive predictions in the predicted class. The *Recall* is for the proportion of positive instances predicted correctly.

LOOK_AHEAD=25 STEP=1	GIVEN_K=5	GIVEN_K=10	GIVEN_K=25	GIVEN_K=50
Precision@25	0.0200	0.0200	0.00234	0.0388
Recall@25	0.4225	0.4228	0.4827	0.7275
MRR@25	0.4934	0.4935	0.5120	0.5721

Figure 4: Result with *K-NN*

Expectations and Results

As decreasing the value of *k* would lead to less stable for predictions of a model, our expectation for the accuracy with the value of k=5 will be the worst among the different expecting outcomes. On the other hand, predictions of a *KNN* model with a higher number of *k* will be more stable due to majority voting or averaging. Along with the theoretical, our expected outcome with the *k*=25 may provide the most accurate result based on the *MRR*. As seen in figure 4, our expected results have met with the empirical outcomes as a higher value of *k* returns more accuracy based on the *MRR*. To be more specified with the results, it could be neglected the difference between the value of *k*=5 and *k*=10. However, it is obviously distinguishable when we make a comparison with the value of *k*=5 and *k*=50. Based on the results in figure 4, the *MRR* accuracy with the value of K=50 outperforms any others in the *K-NN* algorithm under the same conditions.

7 RECURRENT NEURAL NETWORK

Apart from the *K-NN* algorithm, *Recurrent Neural Network (RNN)* uses to build a model of a session-based recommendation system. A characteristic of *Recurrent Neural Network (RNN)* is that the output at the current time step returns to the next step as the input. Therefore, this algorithm is able to consider not just the current items, but it has beneficial to remember the preceding elements. In order to optimize an algorithm In the neural network architecture, it is essential to set the right numbers of layers and units (neurons) in each hidden layer. Furthermore, the choice of activation function for both the hidden layer(s) and the output layer is a significant aspect to build a model. In this work, we use a Python Machine Learning library which is "theano" in order to build a Recurrent Neural Network. In this neural network, we use the *tanh* function for the units of the hidden layers and linear function for the output unit. The difference between the *Session-Based RNN* and *Personalized RNN* is the existence of "users-layer" in a neural network architecture. Due to the absent of "user-layer" in a *Session-Based RNN*, the recommendation system is more likely to represent short-term user preference, whereas a *Personalized RNN* is able to contain both long and short terms user preferences thanks to the existence of "user-layer". It is for updating a user *RNN* by the state of the session *RNN* at the end of each session.

Session-Based Recurrent Neural Network

LOOK_AHEAD=25 epochs=5	Session-layers=10	Session-layers=20	Session-layers=30
Precision@25	0.0292	0.0404	0.0437
Recall@25	0.5028	0.7121	0.7728
MRR@25	0.1676	0.2953	0.3523

Figure 5: Session-Based *RNN* with *different session-layers*

LOOK_AHEAD=25 session-layers=20	epochs=10	epochs=15	epochs=20
Precision@25	0.0418	0.0417	0.0418
Recall@25	0.7352	0.7334	0.7351
MRR@25	0.2782	0.2689	0.2638

Figure 6: Session-Based *RNN* with *different epochs*

The first approach with using *Session-Based RNN* is to compare the *MRR* accuracy with the different number of units per hidden layer at the session level. Intuitively, it can be expected a possible outcome is that the more units in *RNN* architecture will provide more accurate and powerful training to make predictions. Hence, in this work, the expected results of using *Session-Based RNN* are the more units for each hidden layers would return higher values of the *MRR*. As can be seen in figure 5, *Session-Based RNN* with the number of 30 units provides the highest accuracy of the *MRR* among the trials. Apart from the trials with the different layers, it is also an important task to find the right number of *epochs* for setting an optimal algorithm in a neural network model. The definition of an *epoch* can be determined one full forward and backward passes of all training examples. Again, an assumption has to be set before running this algorithm. As more iteration of training may lead to strength to make a better prediction in neural network architecture, the highest number of *epochs*, in this case, epochs=20, would represent the most accurate outcome based on the *MRR*. To make an explanation of the results in figure 6, the overall outcome with the different number of epochs does not match with the assumption of what we made. To be more specified, a phenomenon can be noticed that the accuracy of MRR has been decreased as increased in the number of epochs with same conditions. For instance, it is not hard to recognize that the accuracy of the MRR been reduced significantly when the number of epochs increased from 10 to 20. Among several potential causes, overfitting could be the most susceptible reason to make an interpretation for this phenomenon in this work, and it can be addressable to use early stop when the error rate is minimum.

Personalized Recurrent Neural Network

LOOK_AHEAD=25 epochs=5 & users-layers=20	Session-layers=10	Session-layers=20	Session-layers=30
Precision@25	0.0395	0.0467	0.0480
Recall@25	0.6663	0.7703	0.7898
MRR@25	0.3368	0.4445	0.4569

Figure 7: Personalized *RNN* with *different session-layers*

LOOK_AHEAD=25 session-layers=20 & users-layers=20	epochs=10	epochs=15	epochs=20
Precision@25	0.0464	0.0448	0.0451
Recall@25	0.7649	0.7383	0.7465
MRR@25	0.4192	0.3941	0.3974

Figure 8: Personalized *RNN* with *different epochs*

Another recurrent neural network model uses for this work which is *Personalized RNN*. This algorithm is able to provide both long-term and short-term user preferences thanks to its user-layers. A user-layer will be updated by the session state at the end of every session. For the sake of the consistency of this work, Personalized RNN also uses the same conditions with Session-Based RNN. As well, the number of user-layer is fixed to 20 layers for all trials. All assumptions in the previous RNN algorithm are also applied in this algorithm. As well, the expected overall accuracy with Personalized RNN could be better than the previous RNN algorithm because Personalized RNN contains both long-term and short-term user interests. As expected, Personalized RNN outperforms Session-Based RNN across all trials, according to the results in figure 7 and 8. With increasing the number of epochs, overfitting issue presents again in this algorithm.

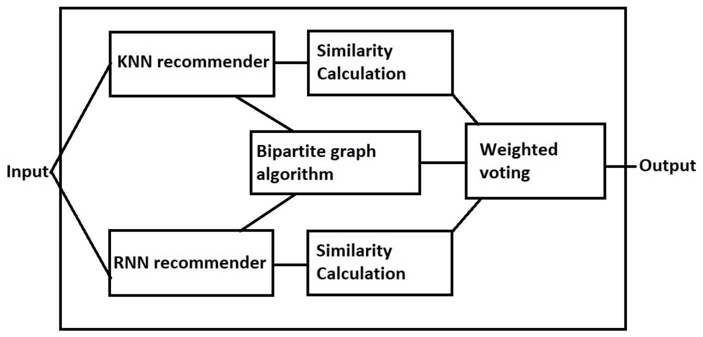

Figure 9: Visualisation of Hybridisation

8 POST-PROCESSING

The Post-Processing consists of two different phases. The recommendations, that are received form the *K-NN* and the *RNN* algorithms, go through a new algorithms that changes the order of the outcome. Simultaneously a new recommender algorithm is run using the results of both the recommenders and the bipartite graph made during the pre-processing. At the end a weighted voting system is used to determine what the best order of suggestions is. This means that the post-processing is a combination of both a Parallel and a Pipeline Design. The effects of all the post-processing techniques will be discussed in the end

Similarity calculation

This algorithm uses the metadata of the accommodations to calculate similarity between them. This is done using the Jaccard coefficient[4]:

$$SimliarityScore = \frac{Features \cap Features}{Features \cup Features}$$

The choice for the Jaccard coefficient came from the ease to implement and the fact that it gives an easy to interpret value for similarity, where 0 means completely dissimilar and 1 means completely similar. The similarity score is calculated between every suggested accommodation and accommodation in the input. For each suggested accommodation the average of all its similarity scores is calculated and saved for later use during the hybridisation process. The reason that this is done during the post-processing has to do with the size of the dataset. The dataset is too big to calculate these values for every possible set of accommodations, making it impossible to work this out for every combination. Another reason that the SimilarityScore was calculated during the post-processing is that the information that it provides is of low quality. There are some items in the test and training set that are not in the metadata set. This makes it that it can not be properly used in a recommender algorithm.

Bipartite graph algorithm

Using the suggestions of the recommender systems and the bipartite graph that was made during the pre-processing phase a final last algorithm is applied to increase the accuracy and order of the final predictions. For each of the recommended accommodations the amount of paths with length 2 to the input is calculated. This value is then divided by the total amount of paths from the accommodations in the recommended and in the input. This algorithm allows the final recommender to be more sensitive to the location of the accommodations. It also deals with the sparsity of the dataset by using new features. The algorithm used is a recreation of the Pinterest Related Pins algorithm, which uses a bipartite graph of pins and boards to recommend new content to its users.[3] Instead of using pins and boards, accomodations and cities are used.

Cascading Hybridization

During the cascading part of the hybridisation the order of suggestions of the RNN and the K-NN is shuffled around based on the similarity score that was calculated earlier. The similarity score is used in a logarithmic function which goes as follows:

$$Multiplier = log_2(1 + SimilarityScore)$$

The choice for a logarithmic function to transform the similarity score was made to decrease the effect that small dissimilarities between accommodations have. The reason behind this choice was that small differences between accommodations are no problem, but it becomes problematic when accommodations, that are nothing alike, are recommended.

Weighted Hybridization

At the end the two augmented rankings are combined together by a weighted voting system. Each suggestion receives a value based on their position in the recommendation (earlier position means higher value) and a weight is calculated for each recommender. The weights come from the MRR that the recommenders have on their own. The weights are calculated by multiplying the MRR by 10 and then rounding the outcome to an integer. This way the weights are based on how well the recommender systems are performing and good recommender systems will increase their performance. The weights are as follows:

	RNN	K-NN	Bipartite Graph
Weight	4	5	1

Table 1: Weights of the recommender algorithms

	Populair Submission
Before	0.2884
After	0.3027

Table 2: Effect of adding the bipartite graph algorithm on the MRR

The weight of the bipartite graph algorithm is set to 1, since it is dependent on the other recommenders to deliver the variables. The recommendations are then sorted from high to low and put in the final recommendation of this system.

Results

Here should be the results of the post-processing on the recommenders that we tested, but we ran into troubles. The post-processing is quite complex and thus takes a long time to calculate. This combined with issues with team members leaving the group created an impossible timeframe to completely test the post processing on our own recommenders. A lighter version only containing the bipartite graph was run on the populair submission algorithm that was provided as a tutorial by Trivago. These results are reported in the table below.

The results show that there is a small increase in the MRR after the post-processing. So the graph based algorithm has its effect.

9 CONCLUSIONS

The conclusions section of this report is unfortunately not complete because team members left before they finished their job. So there will be told about what can be reported and what could be reported if everything went according to plan. In the pre-processing of the data some interesting discoveries surrounding the steps preceding a clickout action were discovered. There it was found that the amount of steps follows a logarithmic distribution. These findings were then used to link Zipfs law and a new searching satisfaction rating was created using these findings. In this paper it was reported that K-NN works better for predicting the correct impression than any of the recurrent neural networks. It had the highers MRR, but it was found that the recall was low compared to the RNN. This tells that K-NN was better at placing the correct prediction at the right position, but worse at actually putting the correct prediction in the prediction list. This is something that could be solved in the post-processing. During the post-processing we found that there was an improvement in the MRR. We would have liked to present the effect of the similarity check as well, but there was not enough time to run this.

REFERENCES

[1] Kittipong Chomboon, Pasapitch Chujai, Pongsakorn Teerarassammee, Kittisak Kerdprasop, and Nittaya Kerdprasop. 2015. An Empirical Study of Distance Metrics for k-Nearest Neighbor Algorithm. 280–285. https://doi.org/10.12792/iciae2015.051

[2] ÅĄukasz DÄŹbowski. 2000. Zipf's Law: What and Why? (12 2000).

[3] David C. Liu, Stephanie Rogers, Raymond Shiau, Dmitry Kislyuk, Kevin C. Ma, Zhigang Zhong, Jenny Liu, and Yushi Jing. 2017. Related Pins at Pinterest: The Evolution of a Real-World Recommender System. In *Proceedings of the 26th International Conference on World Wide Web Companion (WWW '17 Companion)*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, Switzerland, 583–592. https://doi.org/10.1145/3041021.3054202

[4] Raimundo Real and Juan M. Vargas. 1996. The Probabilistic Basis of Jaccard's Index of Similarity. *Systematic Biology* 45, 3 (1996), 380–385. http://www.jstor.org/stable/2413572