Proceedings of the 2nd International Workshop on Social Influence Analysis (SocInf 2016)
July 9th, 2016 - New York, USA

Location Based Services Recommendation with Budget Constraints

Hao Wei[1⋆], Bei Shi[1⋆], and Junwen Chen[2⋆]

[1] The Chinese University of Hong Kong, Shatin, N.T., Hong Kong,
hwei, bshi@se.cuhk.edu.hk
[2] Internal Search Platform Department, Tencent Technology Co. Ltd,
rolanchen@tencent.com

Abstract. The paper reports the approach of our team "711" to recommend Brick-and-Mortar Store with Budget Constraints. By extensive studies, we find that the history data is very effective in this location based service recommendation and we exploit the history data in our recommendation from both the users' aspect and the merchants' aspect. Moreover, we extract predicative features from the train set and implement some existing methods, e.g., Xgboost. Our final ensemble method obtains a F_1 score of 0.452 and reaches the third place in the SocInf'16 competition.

1 Introduction

Nowadays with the development of mobile devices, people usually share their contents with locations. Such kind of sharing facilitates the accumulation of data of location based service (LBS). By investigating the data, we can analyze the relationship between the location and the actions of users. Moreover, the analysis of LBS data will profit the business of on-site merchants. For example, the recommendation based on location considering online behaviour and preference of users can bring enormous flow of people for the merchants. To promote the research on recommendation of LBS, the competition named as "Brick-and-Mortar Store Recommendation with Budget Constraints", which is held by Alibaba and IJCAI-16 workshop, focuses on nearby store recommendation when users enter new areas they rarely visited in the past. The contest mainly has two challenges as follows.

- **The correlation between online and on-site recommendation**. In the competition, both the data from Taobao and Koubei are provided to investigate the correlations between the visiting records of online and on-site. We need to identify the relationship between the click history of Taobao and the on-site visiting records of Koubei.
- **The budget constraints**. The recommendation of each merchant in the data set is bounded by the budget, specifically, the service capacity or number of coupons available at the stores. The solution in the competition should

⋆ All the authors contributed equally to this work.

consider the limit of budgets. Otherwise, the exceeding of budgets will decrease the final performance.

In the competition, given the user behavior records of online website and on-site merchants, we need to recommend the potential merchants that users will visit in the future. A test set of users on the specific location are also provided. For each merchant, the information about branches and budgets should also be considered practically. To prevent overfitting, this competition contains two stages. In the first stage, some merchants are removed from the evaluation. Specifically, the involved data are shown as follows.

- Online user behavior before Dec. 2015. (ijcai2016_taobao)
- Users shopping records at brick-and-mortar stores before Dec. 2015. (ijcai2016_Koubei_train)
- Merchant information. (ijcai2016_merchant_info)
- Prediction result. (ijcai2016_Koubei_test)

To tackle the task, we exploit historical data for each user and each merchant. Moreover, we extract predicative features from the train set and implement Xgboost and Factorization Machine in the task. Considering the budgets, we proposed a method to prune our outputs into the results which satisfy the constraints. By ensembling our result of different methods, we obtained a F_1 score of 0.452 and reached the third place.

2 Exploiting Historical Data

We analyze the given train data and split the Koubei train data into two parts. The first part is the train data with the date before November 23 and we treat it as the new train data, while the second part is the data with the date after November 23 and we treat it as the new test data in our analysis. By extensive studies on the generated dataset, we find that the history data is very effective in this location based service recommendation. That is, given the location, users tend to visit the same merchants they have visited before. From the angle of merchant, the popular merchants attracting many users in the history will remain to be visited by many customers in the near future. Based on the above observations, we exploit the history data from both the users' aspect and the merchants' aspect.

2.1 Users' history

We find that some queries (user, location) in test file have been occurred in the train data. Based on the observation that people tend to visit the same merchants visited before, a simple strategy is to answer these queries with the merchants the users have visited in those locations. However, the frequency of the merchant to be visited by the user affects the recommendation quality. The higher the frequency is, the higher accuracy of the recommendation we can get.

For example, if the user visit a merchant more than 6 times in the history, we can get over 90% success rate when recommending the merchant to the user in the future. On the other hand, every merchant has the budget constraint which may not be able to feed the need of all queries (user, location) that have visited the merchant before. To utilize the information of frequency, we represent the history data as a set of tuples (user, location, merchant, frequency) where frequency counts the number of tuples (user, location, merchant) occur in the history. Then we sort the tuples (user, location, merchant, frequency) in the order of decreasing frequency and recommend merchants to the queries by this order until the budget ran out. Since we find that every user visits only about 1.3 merchants in a place according to the train data, we limit the maximum number of merchants to be recommended as 4.

2.2 Merchants' history

After analyzing the train data, we find that different merchants have different popularity. For example, we find a merchant named '820' takes nearly a quarter of data records in the train data. In the following, we assume every merchant opens in only one location. For the case that the same merchant open in different locations, we treat them as the different merchants because even the same merchant may have different popularity in different locations. Supposed that 90% of users coming to location X will visit a merchant Y. In this case, we can achieve 90% precision for queries occurring in the location X by recommending merchant Y to users in location X no matter who they are. The percentage of pair (user, location) visiting a merchant is called the merchant's popularity. To calculate the popularity, we firstly count the number of users coming to a location X as N_X and then count the number of users visiting to the merchant of that location as N_Y. So the popularity of the merchant Y is calculated as $P_Y = \frac{N_Y}{N_X}$, $P_Y \leq 1$. We cannot get the popularity of every merchant in September directly but estimate it by every merchant's historical popularity. Note that the popularity of a merchant may vary with time. The popularity should be obtained based on the history data closed to September. After many testings, we choose to use the data with the date after November 23 as the source of computing popularity.

To improve the quality of recommendation, two key issues should be emphasized. The first one is how to make good use of merchant budget. It is obvious that the higher popularity the merchant has, the higher precision we can get. So we sort the pair (merchant, popularity) in the order of decreasing popularity and do the recommendation by the order. To get a higher F_1 score, after recommending merchant Y to a query, we deduct the expected value of precision P_Y, rather than 1, from the budget of merchant Y. Therefore, although a merchant Y has only B_Y budget, we may recommend $\frac{B_Y}{P_Y}$, whhich is larger than or equal to B_Y for $P_Y \leq 1$, users to it because we keep the same recommendation precision but get higher recall for this merchant because we get more, on average B_Y, correct answers in this case. The second issue is what popularity threshold we should set to obtain a higher F_1 score. We denote N_s as the number of submitted merchants, N_r as the number of correct answers in the submission, N_t as the

total number of correct answers of the whole test file. So the precision and recall are $P = \frac{N_r}{N_s}$, $R = \frac{N_r}{N_t}$. When answering a query with a merchant of popularity p, we investigate under what popularity p, the F_1 score can remain the same value. We formulate it as the mathematical problem below,

$$F_1 = \frac{2 * \frac{N_r}{N_s} * \frac{N_r}{N_t}}{\frac{N_r}{N_s} + \frac{N_r}{N_t}} = \frac{2 * \frac{N_r+p}{N_s+1} * \frac{N_r+p}{N_t}}{\frac{N_r+p}{N_s+1} + \frac{N_r+p}{N_t}} \tag{1}$$

$$p = \frac{1}{2} * F_1 \tag{2}$$

So we set the popularity threshold as $\frac{1}{2} * F_1$ at first and recommend merchants with popularity higher than the threshold. Since the estimated popularity may be different from the real values, we decide to raise the threshold a little bit after several testings and set it as 0.25.

2.3 Filtering by the connection between taobao data and Koubei data

Recommendation by merchants' popularity takes no consideration for every user's own feature. In fact, we can utilize users' taobao browsing record to do some filtering during the recommendation. For example, users browsing online sellers of electronic products should rarely visit on-site merchant of women's clothes. Therefore, we build up a connection table from online sellers to on-site merchant. If there exist records that users browsing online sellers A also visit the on-site merchant B, we put (A, B) into the connection table. Based on the connection table, for users having taobao records of browsing online seller A, we will only recommend on-site merchants that exist in connection table together with A. By this way, we filter the recommendations with low precision.

3 Modeling Techniques

In the competition, we have tried some methods for recommendation, including Xgboost, Logistic Regression and Matrix Factorization. Moreover, we generate all kinds of features for each user, location and item. In this section, we will introduce these methods and features in detail.

3.1 Data Set Generation

We formalize the original recommendation problem as binary classification. For example, in the original visiting history, user u has visited merchant m in the location l. For this training record, we can generate a 3-tuple, i.e., (u, l, m). For each 3-tuple in the original visiting history, we can generate a series of train records. First, we can produce positive instances by attaching a true label with the original 3-tuple. Second, we construct the negative instances by considering

all the other possible merchants in the location. For each merchant m' in l except m, we can generate the negative instances (u, l, m') which label is false. Therefore, we generate a training data set for the binary classifier. Similarly, we can also generate a test data set according to the given users and locations in the original test file. The recommended merchants are associated with an output label of true. To balance the counts of positive and negative records, we sample the negative records from the set of m' instead of all the m'.

To avoid overfitting and test our methods efficiently, we also construct an off-line data set denoted by $\mathcal{D}$. The train set $\mathcal{D}_s$ are generated from the orginal records of history before 20151123. The corresponding test set $\mathcal{D}_t$ is the data after 20151123. We test our algorithms on $\mathcal{D}$ to ensure the improvements of the on-line judge.

3.2 Feature Engineering

The generated features play an important role in modeling. The essential task in the recommendation problem is to design effective features. From the train set, we observe that some users usually visit merchants in their history records. Some users like to spend more time on popular merchants in that location. Some branches of popular merchants emerges after opening. Base on the above investigations, for each 3-tupe (u, l, m), we design a set of features to depicts these observations, which are shown as follows.

- `history_exists_flag`: the binary flag indicating there exists other history records of (u, l, m) before the timestamp t.
- `history_count`: the normalized count of history records (u, l, m) before the timestamp t.
- `user_id`: the one-hot encoding of user id.
- `merchant_id`: the one-hot encoding of merchant id.
- `merchant_location_count`: the normalized count of (l, m) in the training set.
- `merchant_count`: the normalized count of m in the training set.
- `open_interval`: the time interval between $t_{m,l}$ and 20150701. We select the first record of (l, m) by the order of the timestamp and $t_{m,l}$ denotes the timestamp of the first record which menas the opening date.
- `time_interval`: the time interval between t and $t_{m,l}$. t is the timestamp of the record. This feature captures the trend of m in l.
- `taobao_records_count`: the count of online records of taobao for user u.
- `taobao_category_id`: the one-hot encoding of categegory id for the taobao records that user u clicked or bought.
- `taobao_seller_id`: the one-hot encoding of seller id for the taobao records that user u clicked or bought.
- `taobao_buy_ratio`. the ratio between the count of user u bought and clicked.
- `taobao_buy_count`: the count of records that user u bought in the on-line Taobao.
- `taobal_click_count`: the count of records that user u clicked in the on-line Taobao.

- `user_latent_vector:` the latent vector for user u. We generat the latent vectors with the SVD factorization [4] of the history record provided by the scikit-learn package.
- `merchant_latent_vector:` the latent vector for merchant m. Similarly, we generate the latent vector of the merchant m using SVD factorization.

3.3 Xgboost

Xgboost [1] is a large-scale machine learning method which can be used to build scalable learning systems. Xgboost has been used by a series of kaggle winning solutions as well as KDD Cup winners, which prove the efficacy of this method. It is fast and optimized for out-of-core computations. XGBoost is short for Extreme Gradient Boosting, where the term Gradient Boosting is proposed in the paper [3]. The method of boosted trees has been around for really a while. It is trained with decision trees of a fixed sizes as base learner, which is robust to outliers. As a tree-based algorithm, the gradient boosting decision trees can also handle non-linear feature interactions.

In our competition, we use the implementation of Xgboost provided in the github[3]. We have tried two different optimization functions. One is the loss of Logistic classification and the other one is the rank loss. We mainly tune the parameters including the maximum depth of tree, the step size shrinkage used in update to prevents overfitting and minimum sum of instance weight(hessian) needed in a child. Finally, we set the maximum depth to 10, the step size to 0.3 and the minimum sum of instance weight to 2.0 in our competition.

3.4 Factorization Machine

The technique of factorization machine(FM) has been widely used in the recommendation problem [2, 5]. FM method can combine collaborative filtering with other large categorical features, especially sparse data. The time complexity of FM is $O(n)$, which makes FM applicable to handle large scale data size problems. For user u and the branch m in the location l, we can define the factorization machine as:

$$r_{ui} = \mu + b_u + b_m + q_m^T p_u \tag{3}$$

Where μ is the global bias. b_u and b_i is the bias of user and merchant respectively. p_u is the corresponding latent vector for user u. Similarly, q_m is the latent vector of the branch of m in the location l. For learning the model parameters, we can use the algorithms such as Stochastic Gradient Descent (SGD) and Markov Chain Monte Carlo (MCMC). In the competition, we exploited SGD to infer the parameters.

[3] https://github.com/dmlc/xgboost

Algorithm 1 Post-processing for instances of the merchant m

Initialize the threshold η.
Output the instances list L_m for the merchant m.
$L' = []$
for each instance i in L_m **do**
    if $p_i > \eta$ **then**
        add i into L'
    end if
end for
Rank L' according to p_i
Initialize the budget B_m, the output result listR_m
for each instance i in L' **do**
    if $B_m \geq 0$ **then**
        $B_m = B_m - p_i$
        add the 3-tuple u_i, l_i, m into R_m
    else
        break
    end if
end for
Output R_m

3.5 Post-processing

For each candidate 3-tuple (u, l, m) in the test set, our binary classification model will output the probability of the instance to be positive, i.e., the probability that user u visit the merchant m in the location l. Similarly with the threshold in Eq. 2, we set the threshold η as one half of F_1 measure.

Given the output probabilities of instances related to m, we should also consider the budget of mercant B_m. Assuming that we have N_m instances of m to recommend. The precision with respect to m is $\frac{\min\{B_m, N'_m\}}{N_m}$. N'_m is the count of the right instances for recommendation. Obviously, we should recommend $\eta \cdot B_m$ instances in total at least. Moreover, we should consider the instances with high probability to be positive at first. Therefore, considering merchant m, the algorithm of post-processing is shown as follows.

3.6 Ensemble

Our method based on history data has a higher precision compared with our learning method. Therefore, we ensemble our methods with the following steps.

1. Use the method based on history data in Section 2, we can get the result R_1 which has a higher precision and the slack of the budgets for each budget.
2. We use our learning methods in Section 3 to recommend the merchants with the constraints of the slack of the budgets. We get the result R_2.
3. After concatenating R_1 and R_2, we output the final result.

4 Experimental Results

Our experimental dataset consists of three parts,

- ijcai2016_Koubei_train: The train dataset have more than 1 million records from about 230 thousand users. Most of these data is collected in the period from October to November. On average, every user visits 1.05 locations and visits 1.25 merchants in a location, which shows that users tend to visit the same merchant in the future.
- ijcai2016_merchant_info: This file contains the budget constraint for every merchant. There are about 6,000 merchants and every merchant has a budget of at least 100. We find that the budget resource is very limited for popular merchants while the budget of the unpopular merchant is sufficient enough for recommendation. So the long tail effect will influence a lot for the final results.
- ijcai2016_taobao: This file stores over 40 million users' browsing record in Taobao. We mainly use it to predict the interest of users and perform filtering for the final recommendation.

The evaluation metric used in this competition is the F_1 score with budget constraint. Here we list some of our experiment results step by step.

Table 1. Experimental Results

Technique	F_1 score
user's history	≈ 0.25
+ merchant's history	≈ 0.42
+ Xgboost	≈ 0.45

5 Conclusion

In this paper, we study the location based services recommendation problem, which comes from the real applications. We design history based approach, modeling by Xgboost, factorization machines for this task, and get a final F_1 score of 0.45. We note that some issues still need to be further addressed. One is the long tail merchants which actually have a high weight under the setting that every merchant has a budget constraint. The other one is to build up a strong connection between user's online behavior in Taobao and his on-site behavior in Koubei. Although we design a filtering method based on it, we believe further work can be made to use the connection for recommendation directly.

Acknowledgments

We thank the organizer of this competition, Tianchi, for providing an interesting and challenging topic from real applications. We also thank other participants for their helpful discussions in the forum and we learn a lot from them.

References

1. Chen, T., Guestrin, C.: Xgboost: A scalable tree boosting system. arXiv preprint arXiv:1603.02754 (2016)
2. Chen, T., Zhang, W., Lu, Q., Chen, K., Zheng, Z., Yu, Y.: Svdfeature: a toolkit for feature-based collaborative filtering. The Journal of Machine Learning Research 13(1), 3619–3622 (2012)
3. Friedman, J.H.: Greedy function approximation: a gradient boosting machine. Annals of statistics pp. 1189–1232 (2001)
4. Furnas, G.W., Deerwester, S., Dumais, S.T., Landauer, T.K., Harshman, R.A., Streeter, L.A., Lochbaum, K.E.: Information retrieval using a singular value decomposition model of latent semantic structure. In: Proceedings of the 11th annual international ACM SIGIR conference on Research and development in information retrieval. pp. 465–480. ACM (1988)
5. Rendle, S.: Factorization machines with libfm. ACM Transactions on Intelligent Systems and Technology (TIST) 3(3), 57 (2012)