Pcard: Personalized Restaurants Recommendation from Card Payment Transaction Records

Min Du
University of Utah

Robert Christensen, Wei Zhang
Visa Research

Feifei Li
University of Utah

ABSTRACT

Personalized Point of Interest (POI) recommendation that incorporates users' personal preferences is an important subject of research. However, challenges exist such as dealing with sparse rating data and spatial location factors. As one of the biggest card payment organizations in the United States, our company holds abundant card payment transaction records with numerous features.

In this paper, using restaurant recommendation as a demonstrating example, we present a *personalized* POI recommendation system (Pcard) that learns user preferences based on user transaction history and restaurants' locations. With a novel embedding approach that captures user embeddings and restaurant embeddings, we model pairwise restaurant preferences with respect to each user based on their locations and dining histories. Finally, a ranking list of restaurants within a spatial region is presented to the user. The evaluation results show that the proposed approach is able to achieve high accuracy and present effective recommendations.

ACM Reference Format:
Min Du, Robert Christensen, Wei Zhang, and Feifei Li. 2019. Pcard: Personalized Restaurants Recommendation from Card Payment Transaction Records. In *Proceedings of the 2019 World Wide Web Conference (WWW '19), May 13–17, 2019, San Francisco, CA, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3308558.3313494

1 INTRODUCTION

Personalized recommendation has greatly facilitated our daily life. For example, YouTube recommends a personalized video watch list based on each user's watching history as well as the reviews and ratings from the crowd [4, 5]. It is appealing to have such a personalized recommendation system for restaurant recommendations.

To recommend users with items particularly interesting to them, traditional personalized recommendation systems typically use collaborative filtering [20]. To provide recommendations to a user u, collaborative filtering first finds a group S of users that have similar tastes with u (e.g., they have rated similar items with a similar score for each), and then recommends user u with items that are liked by S but haven't been explored by u. An essential requirement of such an approach is the availability of user rating scores, such as Yelp ratings and reviews. However, such information is sparse and often hard to obtain. Furthermore, if a user never gives a rating or review, which is quite common, it is almost impossible to learn this user's preference to provide personalized recommendation. To make the matter worse, reviews and ratings are not always reliable as there are increasingly more artificially generated fake reviews.

To tackle this problem, implicit ratings have been proposed to replace explicit scores. For example, the YouTube [4] recommendation system considers a feedback as positive if a user watches a video until the end rather than aborting early. Yet similar features are not available for restaurant recommendations: knowing a person has visited a restaurant is relatively easy, but it is hard if not impossible to know whether this restaurant is being liked. Also, unexplored restaurants may be due to omission rather than being disliked. As such, simply assigning a visited restaurant with positive rating while unvisited ones with negative as in [12, 28] could make the results highly biased. A naïve idea to model user preferences is to use transaction frequencies made for different merchants as implicit ratings. However, for restaurant recommendation, this approach may not be effective due to the challenge introduced by location factor. User activities are greatly restricted by distance, especially when it comes to *where* to dine.

Our contributions. Our company holds one of the world's largest payments processing networks and handles hundreds of millions transactions every day. Most people in the US have transaction history in our network. Such scales of transaction data are invaluable towards learning users' purchasing behaviors for various recommendation purposes. This work shares our experience in mining the abundant payment card transaction history records to learn user preferences, with the particular focus of personalized restaurant recommendations as an example. Note that all user identity related attributes are processed as double hashing values, thus it is not able to identify a user based on a single transaction record. Further investigation on protecting user privacy while dealing with transaction data with advanced privacy-preserving techniques are also in place, which are not the focus of this work.

To the best of our knowledge, this is the first work that explicitly learns user preferences *based only on* card transaction records, without any rating or review data nor detailed metadata about the restaurants. Our contributions are summarized as follows.

(1) We adopt a novel embedding scheme to represent each user and each restaurant, where similar users/restaurants are represented by vector embeddings close in distance.
(2) We propose a novel approach to model user preferences over restaurants, which considers location effect while generating data samples to learn, but removes the influence of location information while modeling, for better generalization.
(3) We utilize user preferences learned from historical payment card transaction data to provide an effective ranked list of restaurants given any region.
(4) We evaluate our method on large-scale real transaction data to demonstrate its effectiveness, and a web demo was created to showcase the proposed approach.

ACM ISBN 978-1-4503-6674-8/19/05.
https://doi.org/10.1145/3308558.3313494

2 PRELIMINARY

2.1 Dataset

For each card payment through our network, a *transaction record* is logged which contains numerous features: user id, merchant id, merchant location, transaction time, transaction amount, and etc..

2.2 Embeddings

To model the preference of any user to any restaurant, a major challenge is to find a meaningful and effective representation of each user and each restaurant. The number of users we have is comparable to the population in the United States, which makes it unrealistic to represent each user using a naïve method, say, one-hot encoding. Dimension reduction is necessary, while a question raised by which is how to find meaningful representations in low-dimensional space for hundreds of millions of users.

Specifically, for the time window between January 2016 and April 2016, we have around 30 billion transactions available in our database, among which there are around 220 million active cardholders (users) and 78 million active merchants (e.g., online shopping websites, local supermarkets, and restaurants). For each transaction, we extract a pair of user ids (i.e., cardholder account) and merchant ids, denoted as (uid_i, mid_i), which indicates that user uid_i has made a payment to merchant mid_i.

User embeddings. All (uid, mid) pairs are grouped by distinct merchants, i.e., by *mid*s. Thus, each merchant group contains a sequence of all users who have made payments to this merchant, represented by mid_i: $[uid_{i,1}, \dots, uid_{i,n_1}]$. Finally, each *uid* list is treated as a sentence and "sentences" from all merchants are treated as an article, and word2vec [16] is used to generate an embedding for each user *uid*.

Merchant embeddings. All (uid, mid) pairs are firstly grouped by distinct *uid*s. A list of all merchants that have been attended by a user is treated as a sentence and all such "sentences" are treated as an article. Word2vec is used again to generate an embedding for each merchant *mid*.

Each user embedding is a 200 dimension float vector while each merchant embedding has 400 dimensions. The number of dimensions for each vector space are chosen to be both efficient and effective for different tasks.

All user embeddings and merchant embeddings are stored in two tables: *global_user_embeddings* and *global_merchant_embeddings*. Standard similarity measures such as k-nearest neighbors with euclidean distance or cosine similarity demonstrate that this embedding approach, as a representation method, is able to model intrinsic features of different entities. These embeddings have been shown effective in our other applications, e.g. fraud detection.

The embeddings are created in a way such that *similar* users have closer distance in the embedding space, e.g., if they live/work in the same area, or have similar shopping behaviors. Likewise, *similar* merchants also have closer distance in the embedding space if they are in the same category, in the same area or they accommodate similar users. Note that this property is essential towards building effective recommendation systems. Recall that the core idea of collaborative filtering is to provide a user with recommendations that are liked by other users who share *similar* tastes. Using this embedding approach, we are able to provide a user with effective restaurant recommendations even if this user has never been to a restaurant within our modeled time period. If this user has ever made any card payment through our network, e.g., shopping at a local grocery store, we could recommend to this user restaurants that are liked by other users who have similar purchasing behaviors.

3 METHODOLOGY

3.1 Overview

In this section, we will describe how Pcard works using the embeddings generated for each user and each restaurant. The objective is to provide a ranked list of restaurants for a given user u in a selected geographical region *RG*.

Pcard proceeds in two stages: 1) in an *offline learning stage*, a machine learning model is trained to learn different users' preferences over different restaurants, which we refer to as a *preference model*; 2) in an *online recommendation stage*, given a user u and a region *RG*, we rank all restaurants within *RG* based on u's preference over each restaurant in *RG*, provided by our preference model.

An intuition that we leverage is that a user's preference over a restaurant is typically a *relative ranking* compared to another restaurant, instead of an *absolute ranking*. Based on this observation, we learn user preferences by comparing pairs of restaurants for each user. Specifically, we extract data samples of $[u, r_1, r_2]$ from all transaction records within our learning time period, where r_1 and r_2 stand for two restaurants u has attended. The embeddings of u, r_1, r_2 are concatenated as the input vector for our machine learning model, and a label of 1 or 0 is given to indicate whether r_1 or r_2 is more preferred (by comparing the frequencies that u visited r_1 and r_2 respectively). Finally, given a user u and a region *RG*, we compare all pairs of restaurants within *RG* for u and output a ranked list, similar to ranking in a chess tournament.

However, for preference model learning, challenges exist for extracting data samples $[u, r_i, r_j]$ to learn. The major challenges are brought by location factors. First of all, location could be the major (or only) concern while considering *where* to dine, e.g., for work lunch, but it is not fair to say a company cafeteria is preferred over a fine restaurant that is far away. Thus, location information should be considered when generating data samples to learn. Secondly, a recommendation system that is built upon data from one location, should be generalized to provide recommendations for other locations. Therefore, location information should be removed while constructing the learning model. We address the two challenges at once by comparing restaurants that are close, and use visit frequencies as the preference measurements. For one, imagine if we are surrounded by multiple restaurants which are similar in distance, choosing which restaurant reveals our preference. For another, a pair of restaurants that are close have similar "location information" in their embedding space, which are automatically omitted in preference modeling since they do not provide a difference.

Another challenge is for a pair of restaurants (r_i, r_j) nearby, it is not certain to claim that a user u prefers r_i over r_j if, say, r_i was visited for 5 times but r_j for 4 times. As a result, we only assume that a restaurant r_i is preferred over r_j if r_i is visited many times but r_j few. We do not consider any restaurant that u has not visited during the learning period, since that could be simply due to that a user is unaware of this restaurant rather than he/she doesn't like the restaurant. Our embedding technique guarantees similar users/restaurants would have similar embeddings, which enables the effective learning of user preferences on a subset of data.

To summarize, in the offline learning stage, from all transaction records within the learning period, we extract a data sample $[u,r_i,r_j]$ if: 1) r_i and r_j are within a distance threshold; 2) u attends r_i greater than or equal to f_u times while r_j less than or equal to f_ℓ times.

Finally, after the offline preference model is successfully trained, we could serve this model in the online recommendation stage for any user u and any region RG. The Pcard recommendation system, as shown in Figure 1, works as follows: 1) find all restaurants within region RG, denoted as *restSet*; 2) compare each pair of restaurants in *restSet* for u using our preference model; 3) compute a ranked list of all restaurants in RG based on the pairwise comparison results.

Next, we will explain each of the two stages in detail: 1) train a model that learns any user's preference on any pair of restaurants; 2) design a ranking method that is able to calculate an effective ranked list based on the pairwise comparison results.

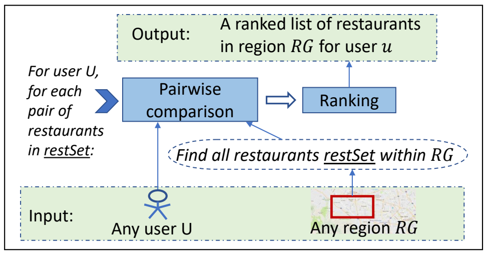

Output:
A ranked list of restaurants in region RG for user u
For user U, for each pair of restaurants in restSet:
Pairwise comparison
Ranking
Find all restaurants restSet within RG
Input:
Any user U
Any region RG

Figure 1: Recommendation system architecture

3.2 Learning a preference model

In this section, we train a preference model where given any user u and any pair of restaurants (r_i, r_j), it could output *how much u* prefers r_i compared to r_j. The resulting model serves as *Preference model* component in Figure 1.

Recall that each user is represented with an embedding such that users that are similar (e.g. have similar shopping preferences) have a smaller distance than users who are dissimilar. Such embeddings also exist for restaurants. Hence, although we're not able to cover all restaurants and all users for learning, the hope is if we have learned enough data, we are able to output meaningful results for similar users and restaurants that do not exist in training data.

To train a preference model, we take the following steps.

Step 1. Data extraction. In this step, we select all restaurant transaction records within the learning time period, and then group by user ids. For each user, we count the number of visits (i.e., visiting frequency f) to each distinct restaurant. The resulting data structure is: $\{\, u_1 : \{\, r_1{:}\ f_1,\ r_2{:}\ f_2, \ldots\},\ u_2 : \{\ldots\}, \ldots \}$, which represents the frequency of visits to each restaurant by each user.

Step 2. Preference pair generation. With the data from step 1, we generate user preference pairs as data samples. Specifically, for all the restaurants a user u has visited, if restaurant r_i and restaurant r_j are within distance threshold D_{max}, and r_i's visiting frequency is higher than or equal to an upper threshold f_u, while r_j's visiting frequency is less than or equal to a lower threshold f_ℓ, we say u prefers r_i over r_j, and generate a preference sample $[u,r_i,r_j]$.

Step 3. Preference model training. After step 2, we have generated many preference pairs $[u,r_i,r_j]$ to learn. Since the goal is to learn a model which could output a preference over two restaurants for any user (i.e., whether u prefers the first restaurant or the second one), it is straightforward to use a binary classifier to learn this property and output "u prefers r_i" (label 1) or "u prefers r_j" (label 0). To generate training data for such model, for each preference pair $[u,r_i,r_j]$, we create two training samples: $[u,r_i,r_j]$ with label 1 to say u prefers the first restaurant, and $[u,r_j,r_i]$ with label 0 to indicate u prefers the second restaurant. For this task, we leverage a deep neural network (DNN) classifier provided by TensorFlow. As in Figure 2, for each data sample $[u,r_i,r_j]$, we first find embeddings for u, r_i and r_j in the embedding datasets and concatenate them to construct the input vector: [u embedding]+[r_i embedding]+[r_j embedding]. Optionally, we could also concatenate relevant statistics to the input vector to learn together. The output is a probability distribution of all possible classes, indicating which restaurant is more preferred. For example, ground truth label "1" is denoted with probability distribution of [1,0], indicating the first restaurant is more preferred. The process of training a DNN classifier is thus to adjust the parameters, in order to minimize the error between the actual output and the ground truth output.

After training, the preference model could be served in online recommendation stage to find a preference, for any given user u and any two restaurants. To do that, a concatenation of their embeddings is fed into the DNN classifier, the output of which is a pair of preference probabilities $[p_{j,i}, p_{i,j}]$, indicating u prefers r_i by a probability of $p_{j,i}$, and prefers r_j by $p_{i,j}$, where $p_{j,i} + p_{i,j} = 1$.

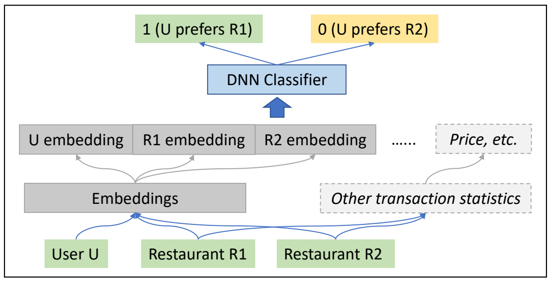


Figure 2: Preference model

3.3 Ranking from pairwise comparison

In online recommendation stage, for a given user u and a region RG, we find all pairs of restaurants within distance D_{max} and calculate their pairwise comparison results. With that, the next step is to output a meaningful ranking list for all distinct restaurants in RG. As discussed in Section 5, there are many previous works investigating the problem of ranking from pairwise comparisons.

3.3.1 *Our approach.*

Recall that in Section 3.2, for a user u and a pair of restaurants (r_i, r_j) to compare, our preference model outputs a preference probability pair $[p_{j,i}, p_{i,j}]$. For all restaurants within region RG, i.e., $\{r_1, r_2, \ldots, r_n\}$, there are a total of $\binom{n}{2} = \frac{n(n-1)}{2}$ pairs of restaurants. For now, suppose the diameter of region RG is less than D_{max}, i.e., all pairs are comparable. Next, we need to find a meaningful way to rank all n restaurants from a total of $\binom{n}{2}$ pairwise comparison results. We use $\Pi(j)$ to denote the final ranking score for restaurant r_j, i.e., r_i ranks higher than r_j if and only if $\Pi(i) > \Pi(j)$. Since all the probability pairs are normalized in pairwise comparison results, it is reasonable to sum up all probabilities of each distinct restaurant, as the final ranking score Π for that restaurant. For instance, if the pairwise comparison results for 3 restaurants $\{r_1, r_2, r_3\}$ are $[p_{2,1}, p_{1,2}]$ (for (r_1, r_2)), $[p_{3,2}, p_{2,3}]$ (for (r_2, r_3)), $[p_{3,1}, p_{1,3}]$ (for (r_1, r_3)) respectively, there are $\Pi(1)$=$p_{2,1}$+$p_{3,1}$, $\Pi(2)$=$p_{1,2}$+$p_{3,2}$ and $\Pi(3)$=$p_{2,3}$+$p_{1,3}$. Finally, we could rank (r_1, r_2, r_3) by comparing $\Pi(1)$, $\Pi(2)$ and $\Pi(3)$. The extension to a larger region *RG* (i.e., not all pairs of restaurants within *RG* are comparable) is discussed later in Section 3.3.3. In evaluation we'll show this seemingly naïve method performs very good in practice, e.g., it is able to reduce the amount of noisy pairwise comparison results. What's more, it also has nice theoretical guarantees.

3.3.2 Theoretical property. Balanced Rank Estimation (BRE) is a ranking method proposed by Wauthier [24] to calculate the ranking score of each item from pairwise comparison results. Specifically, for the j-th item of a total of n items, its ranking score is:

$$\Pi(j) = \frac{\sum_{i \neq j} s_{i,j}(2\bar{c}_{i,j} - 1)}{2m(n)} \tag{1}$$

where $m(n)/n$ is the probability for each pairwise comparison being measured, $s_{i,j} = 1$ if a pair (i, j) is measured, and $\bar{c}_{i,j}$ is the (possibly noisy) measurement that is made.

In our case, each pairwise comparison is measured with a probability of 1, which has: 1) $m(n)/n = 1$, and hence $m(n) = n$; 2) $s_{i,j} = 1$; and 3) $\bar{c}_{i,j} = p_{i,j}$. Therefore, equation 1 reduces to: $\Pi(j) = \frac{\sum_{i \neq j}(2p_{i,j}-1)}{2n} \propto \sum_{i \neq j} p_{i,j}$. This means that BRE method is proportional to our proposed approach to calculate ranking score, which is, for each distinct restaurant, the sum of corresponding probabilities in all pairwise comparison results. Therefore, the theoretical properties proved for BRE method all proportionally hold for our approach. These include upper bound on number of inverts (cases where $p_{i,j} > p_{j,i}$ while $\Pi(j) < \Pi(i)$), and sample complexity (number of pairwise results needed to compute the final ranking), details of which could be found in [24].

3.3.3 Extension. The approach in Section 3.3.1 is based on pairwise comparison results for all pairs of restaurants in region *RG*. However, our initial assumption is that, two restaurants are only comparable if they are close (i.e., within a distance threshold D_{max}). Therefore, if two restaurants in region *RG* are further than D_{max}, it's neither valid nor accurate to compare them using our preference model, since there might be location information inscribed in their embeddings that may obfuscate the results. Hence, if *RG* is too large, we could only get incomplete pairwise comparison results, i.e., a set of pairwise results for pairs of restaurants that are within distance D_{max}, among a total of $\binom{n}{2}$ possibilities. Ranking from incomplete pairwise results is an important research topic in theory community [24]. However, the equations are not ready to use because they assume each pairwise comparison is *independently measured with the same probability*, which is often not the case in reality. Combined with the results from [24] (Equation 1), and our ranking approach in 3.3.1, we adopt another simple yet meaningful approach for ranking from incomplete pairwise results. In this approach, the ranking score of a restaurant is the sum of corresponding probabilities in all pairwise comparison results, normalized by the count of pairwise comparisons being made on that restaurant. For example, For three restaurants (r_1, r_2, r_3), if pairs (r_1, r_2) and (r_2, r_3) are close while r_1 and r_3 are far away, we have pairwise comparison results of $(p_{2,1}, p_{1,2})$, $(p_{3,2}, p_{2,3})$, then the ranking score for each restaurant is: $\Pi(1) = p_{2,1}$, $\Pi(2) = \frac{p_{1,2}+p_{3,2}}{2}$, $\Pi(3) = p_{2,3}$, based on which we could rank all three restaurants. Note that, by normalization we make sure each restaurant's measurements have the same weights in final ranking score (i.e., each restaurant is normalized to be measured once). Suppose restaurant r_j is measured by t_j times, the probability it is measured is $t_j/\binom{n}{2}$, i.e., $m_j(n)/n = t_j/\binom{n}{2}$ in Equation 1, which gives $m_j(n) = 2t_j/(n-1)$. Finally, Equation 1 becomes:

$$\Pi(j) = \frac{\sum_{i \neq j} s_{i,j}(2\bar{c}_{i,j} - 1)}{4t_j/(n-1)} \propto \frac{\sum_{i \neq j} p_{i,j}}{t_j} \tag{2}$$

which is exactly what we propose, to normalize the sum of corresponding probabilities for each restaurant by the number of times that restaurant is measured.

4 EVALUATION

This section evaluates the performance of Pcard. For all experiments we use the time period of January to February 2016 for training and validation, and March to April 2016 for testing. Note that *none* of the previous work has explored personalized restaurant recommendation in our setting, that is, without any explicit/implicit rating, and for a humongous volume of data. As a result, for baseline comparison, we simply adopt crowdsourcing approaches which are the only practical alternatives that could be evaluated on our data.

4.1 Pairwise comparison evaluation

4.1.1 Dataset. The evaluation dataset is extracted from transaction records stored in our database. For pairwise comparison evaluation, we first train a preference model on data samples generated from transaction records of a selected spatial area (e.g., Bay Area). After that, the accuracy of the preference model is tested not only on data samples of the same spatial region, but also for a different region, to test *the influence of location factors.*

Given an area, we first select all restaurant transaction records within this area. The rest of dataset generation follows the procedure described in Section 3.2. Recall that for each user u and each pair of restaurants (r_i, r_j), we generate a data sample $[u, r_i, r_j]$ if r_i, r_j are within distance threshold D_{max} and u has visited r_i more than or equal to f_u times while r_j fewer than or equal to f_ℓ times. The parameters used for all experiments are shown in Table 1a.

To validate our approach, we extract transaction records from areas ranging from smaller ones such as Bay Area, New York City, to larger ones such as California (CA) and the contiguous United States as our geographical regions to learn user preferences. Note that, if a preference pair $[u, r_i, r_j]$ appears in both training and testing datasets, we would eliminate that from testing data, to make sure that our model in fact learns preferences from embeddings instead of just memorizing training results. The number of data samples $[u, r_i, r_j]$ for each dataset is listed in Table 1b.

distance threshold	D_{max}	500 meters
visiting frequencies threshold	f_u	5
	f_ℓ	1

(a) Parameters used for all experiments

(BA: Bay Area; NYC: New York City; CA: California)

region	BA	NYC	BA & NYC	CA	US
#samples	1,951,413	2,720,745	4,679,721	24,955,210	211,519,990

(b) Number of preference pairs generated

Table 1: Datasets

4.1.2 Baseline. Our baseline mimics the default setting in current recommendation services (e.g., Yelp), which simply recommends a restaurant preferred by most users. Specifically, for each data sample $[u, r_i, r_j]$ in testing dataset, this baseline outputs a preference for r_i

if r_i has a higher visiting frequency by *all card holders* in training period, otherwise r_j.

4.1.3 Evaluation results. As discussed in Section 3.2 and Figure 2, for each extracted data sample $[u,r_i,r_j]$, we create a concatenation of all their embeddings to train a DNN classifier using TensorFlow. The DNN classifier learns to output a pair of probabilities showing the preferences over two restaurants. For example, for an embedding concatenation $[u$ embedding$]+[r_i$ embedding$]+[r_j$ embedding$]$, the output of the classifier would be $[p_{j,i},p_{i,j}]$ where $p_{j,i} + p_{i,j} = 1$, showing the probabilities of liking r_i, r_j respectively. Note that, for a sample $[u,r_i,r_j]$, there are two possible input embedding concatenations, one is $[u$ embedding$]+[r_i$ embedding$]+[r_j$ embedding$]$, and the other is $[u$ embedding$]+[r_j$ embedding$]+[r_i$ embedding$]$, which outputs $[p'_{i,j}, p'_{j,i}]$. In an ideal case, $p_{j,i} = p'_{j,i}$ and $p_{i,j} = p'_{i,j}$ should be true. However, due to the high dimension of input embeddings and the un-interpretable DNN classifier with millions of weights, it is not even true that $[p_{j,i}, p_{i,j}]$ and $[p'_{i,j}, p'_{j,i}]$ always have the same order (i.e., $p_{j,i} > p_{i,j}$ and $p'_{j,i} > p'_{i,j}$ or $p_{i,j} > p_{j,i}$ and $p'_{i,j} > p'_{j,i}$). To correct this, we test a sample $[u,r_i,r_j]$ with the two different embedding concatenations as two separate inputs to the DNN classifier. Then, we output whether r_i or r_j is more preferred by comparing $p_{j,i} + p'_{j,i}$ and $p_{i,j} + p'_{i,j}$. The idea is that, although the order of $[p_{j,i}, p_{i,j}]$ and $[p'_{j,i}, p'_{i,j}]$ might not be the same, the case where the probability difference is higher means that the prediction label is more certain, where the other case might be because some part of the weights are not sufficiently trained. In our evaluation we found that this type of "two-way comparison" is more accurate than just using one type of embedding.

The prediction accuracy for each spatial area is shown in Table 2. Our method outperforms the baseline in all areas. We observe test accuracy decreases slightly when selected region size becomes larger, which could be caused by increased data varieties. Nevertheless, in the entire California state, the accuracy is still as high as 76.70%, and this accuracy does not drop further when the spatial area extends to the whole US region. Furthermore, to evaluate the influence of location information encoded in our embeddings, we use a classifier trained on New York City dataset to test preference accuracy in Bay Area, and the result is 80.77%, which is almost the same with the accuracy (81.33%) of using a classifier trained on Bay Area itself. It means that although restaurant embeddings may contain location information, by learning the preferences over pairs of restaurants within a region, DNN classifier is able to "ignore" that information which is similar between each pair of restaurants, and use other encoding information to learn a preference instead.

	BA	NYC	BA & NYC	CA	US
Pcard	81.33%	83.65%	80.64%	76.70%	76.66%
	Train on NYC and test on BA:			80.77%	
Baseline	64.27%	64.78%	64.53%	66.49%	68.38%

Table 2: Pairwise comparison evaluation results

4.2 Ranking from pairwise comparison results

In this section, we evaluate our ranking algorithm, which outputs a ranked list from pairwise preference probabilities provided by the preference model (see Figure 1 for detail).

To evaluate the correctness of ranking algorithm, previous works [19, 24] typically count the number of inverts. Given a set of n objects and binary pairwise comparisons between pairs of objects (e.g., object i > object j), an invert happens if the original pairwise order of object (i, j) is not preserved in the final ranked order. In previous work, the goal of ranking algorithm is to minimize the number of inverts.

However, in our work, there are two major differences. First, our pairwise comparison results are not the ground truth. To check final accuracy of ranking algorithm, we need to check with ground truth results from actual transaction history, instead of pairwise comparison probabilities output by the preference model. Second, it is very rare that a user u would have visited all restaurants within a region RG, which means that for ground truth data, we do not have all pairwise visiting frequency comparisons between any two restaurants within a region RG for a particular user u. Thus, we are not able to count the number of inverts in final ranking order compared to ground truth pairwise comparisons.

Instead, based on the data we have, we design a meaningful evaluation approach that could reveal the performance of our ranking algorithm. Specifically, in the testing dataset generated in Section 4.1, each sample $[u,r_i,r_j]$ has a ground truth label of 1 or 0, meaning which restaurant between r_i and r_j is preferred. Using this dataset, the following procedures are applied to each data sample $[u,r_i,r_j]$: 1) find the smallest region RG_{ij} that could enclose both restaurants; 2) find all restaurants within region RG_{ij}; 3) compute pairwise preference probabilities for each pair of restaurants in region RG_{ij}; 4) rank all restaurants within region RG_{ij} based on pairwise probability results, through summing up all corresponding probabilities of each restaurant as its score; 5) check the ranking order of restaurants r_i and r_j, and see if it's the same with the ground truth label of data sample $[u,r_i,r_j]$, i.e., whether the summed score of r_i in final ranking results is higher or lower than r_j, and whether the ground truth label of $[u,r_i,r_j]$ is 1 or 0.

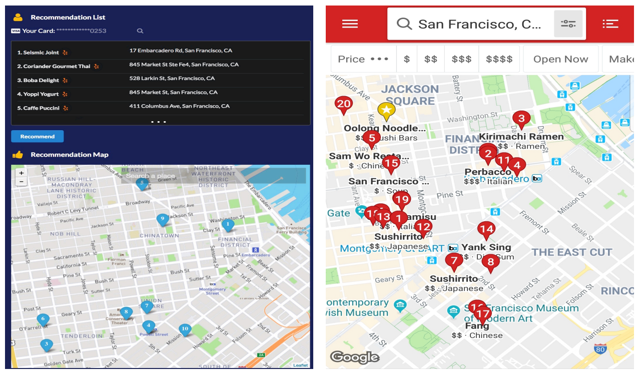
After above procedures, for all data samples in testing dataset, we get a *ranking order accuracy*, i.e., the percentage of the test samples that are correctly ordered in the *ranking results*. Then we compare this *ranking order accuracy* with the *pairwise comparison accuracy* in Section 4.1. If the *ranking order accuracy* is higher, it means that our ranking algorithm not only does not exacerbate previous step, but is even able to reduce noise in the pairwise comparison results.

The evaluation results show that the ranking algorithm has noticeably increased pairwise accuracy, from 79.95% (pairwise comparison accuracy) to 82.00% (ranking order accuracy) for Bay Area, which expresses the effectiveness of our ranking algorithm.

4.3 Overall recommendation performance

To test if our recommended restaurants are actually meaningful, for a given user, we use the whole end-to-end model shown in Figure 1 to output a ranked list of restaurants within a given region, and validate our results, by checking if this user actually attended the restaurants we recommended during the test period. We conducted this experiment for all users in Palo Alto area, and found that for over 80% of the cases, a restaurant the given user attended in test dataset is within top 30% of our ranked list. This means that if we recommend a list of 10 restaurants to a user, with large probability this user will go to one of our top 3 recommendations.

Baseline. For baseline, we adopt the crowdsourcing setting used by most services, where the recommendation list is sorted by visiting frequencies of all card holders. We found that, for over 80%


Recommendation List

Your Card: ************0253

1. Seismic Joint	17 Embarcadero Rd, San Francisco, CA
2. Coriander Gourmet Thai	845 Market St Ste Fe4, San Francisco, CA
3. Boba Delight	528 Larkin St, San Francisco, CA
4. Yoppi Yogurt	845 Market St, San Francisco, CA
5. Caffe Puccini	411 Columbus Ave, San Francisco, CA

Figure 3: Screenshots of Pcard **(left) and Yelp (right) showing restaurant recommendations in downtown San Francisco**

of the cases, an attended restaurant in test dataset is only at top 46.15% of the recommendation list; while by only 65.12% of the cases could the attended restaurant appear within top 30% of the list. Apparently, Pcard provides more effective recommendations.

4.4 Interactive Demo

An interactive Pcard demo has been deployed at our company front desk to collect user feedbacks which continuously help to improve the system. A user interacts with the demo by sliding a payment card and selecting a location (range) for recommendations. The front-end submits a request for the restaurants in the area to be ranked by this user's preference. On the back-end, the system will look up all relevant embeddings, score all restaurants as described in Section 3, and finally return top k (e.g., k=10) restaurants.

A screenshot of the demo is shown on Figure 3 left. All restaurants' locations are shown with a link to each restaurant's Yelp page for more information. Compared with the Yelp recommendation result shown in Figure 3 right, which lists an overwhelming number of recommendations to be filtered further, Pcard saves user efforts to input additional information such as cuisine preference, which is already encoded in embeddings. Providing useful and interesting recommendations without using manual data labels is an important advantage of Pcard. The feedback from our demo is positive and is used to continue to improve the performance of Pcard.

4.5 Remarks

Pcard in its current state has certain limitations. First, two restaurants must be sufficiently close in order to be compared. The distance threshold of 500 meters in Table 1a is selected to be conservative. An alternative is to use restaurant density together with distance, in order to decide whether two restaurants are comparable. Second, Pcard may emphasize less expensive restaurants because of higher transaction frequency. To accommodate for this limitation, we could leverage the price range of restaurants and aggregated visiting frequencies to help with the preference model learning.

Despite the above limitations which we could address further, Pcard outperforms current POI recommendation systems in many aspects. First of all, with Pcard and our demo app as shown in Figure 3, users never have to explicitly label what they prefer or dislike in order to get useful recommendations. What's more, by only learning information from transaction records, Pcard addresses the cold start problem since we do not need any review or rating data to begin with. Also, our deployed model only contains information about double-hashed user ids and restaurant visiting frequencies, which protects user privacy. Last but not the least, Pcard could be further improved with user review data. A user could provide feedback through our app, after he/she tries a recommended restaurant.

5 RELATED WORK

Recommendation systems have been an important research topic for decades. Collaborative filtering, used by most traditional recommender systems [5, 13], recommends a user u with unexplored items that are liked by other users having similar tastes [6]. To achieve this, it needs sufficient rating scores to find *similar* users, e.g., Amazon rating scores [13], Yelp reviews [15], and Netflix's movie ratings [2]. However, explicit ratings or reviews are often not available, and could be highly biased. Implicit ratings [18] are proposed for the aforementioned problem. For instance, YouTube recommendation system treats a video as being liked by a user if the user finishes that video [4]. Similarly, if a webpage is being scrolled to the end, then it is likely that webpage is being liked [11].

However, everything is more complicated when it comes to point-of-interest (POI) recommendation [26, 27], especially restaurant recommendation, partly due to the complexity introduced by geographical location factors [1, 9, 14, 21, 26]. For restaurant recommendation, GeoMF [12] models geographical preferences of users using check-in data on location-based social networks (LBSNs), but they treat locations that users have been to as what users are likely to prefer. Zhang etc. [28] integrate implicit check-in data and explicit review data for more accurate recommendation, while analyzing user preferences based on their demographics information and restaurants attributes. In contrast, we aim to remove the influence of location factor while modeling user preferences, without any need of review data.

Deep learning has been gradually replacing many traditional algorithms in various tasks including recommendation systems [4, 8, 22, 23, 29]. Despite of the successful results, these methods still need either explicit ratings or implicit ones to begin with.

In our approach, we propose a ranking method that is able to output a ranking list of restaurants based on pairwise comparison results. Ranking from pairwise comparison results is heavily used in tournaments like chess player games, e.g., ELO rating system [25]. Research efforts in this field include to rank more efficiently [10, 17, 19], rank from incomplete pairwise results [24], and to only select top-k candidates [3, 7, 19]. Our proposed method is easy to implement, and has the same theoretical bounds with [24].

6 CONCLUSION

In this paper, we address the problem of restaurant recommendation using payment card transaction records, without any review or rating data. The experiences learnt shed light on addressing the *cold start* problem in general recommendation systems, especially for POI recommendations. For future work, we plan to utilize extra information such as restaurant statistics mined from transaction records and data from other websites to improve the accuracy, and to learn time series visiting patterns of each user for better recommendation performance.

ACKNOWLEDGMENTS

Feifei Li is supported in part by NSF grants 1816149 and 1619287. Feifei Li is also supported in part by NSFC grant 61729202.

REFERENCES

[1] Jie Bao, Yu Zheng, David Wilkie, and Mohamed Mokbel. 2015. Recommendations in location-based social networks: a survey. *GeoInformatica* (2015).

[2] James Bennett, Stan Lanning, et al. 2007. The netflix prize. In *Proceedings of KDD cup and workshop*.

[3] Róbert Busa-Fekete, Balazs Szorenyi, Weiwei Cheng, Paul Weng, and Eyke Hüllermeier. 2013. Top-k selection based on adaptive sampling of noisy preferences. In *ICML*.

[4] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep neural networks for youtube recommendations. In *ACM RecSys*.

[5] James Davidson, Benjamin Liebald, Junning Liu, Palash Nandy, Taylor Van Vleet, Ullas Gargi, Sujoy Gupta, Yu He, Mike Lambert, Blake Livingston, et al. 2010. The YouTube video recommendation system. In *ACM RecSys*.

[6] Pierre Dillenbourg. 1999. *Collaborative learning: Cognitive and computational approaches. advances in learning and instruction series.* ERIC.

[7] Brian Eriksson. 2013. Learning to top-k search using pairwise comparisons. In *AISTATS*.

[8] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *WWW*.

[9] Tzvetan Horozov, Nitya Narasimhan, and Venu Vasudevan. 2006. Using location for personalized POI recommendations in mobile environments. In *IEEE SAINT*.

[10] Claire Kenyon-Mathieu and Warren Schudy. 2007. How to rank with few errors. In *ACM STOC*.

[11] Hyoung-rae Kim and Philip K Chan. 2005. *Implicit indicators for interesting web pages.* Technical Report.

[12] Defu Lian, Cong Zhao, Xing Xie, Guangzhong Sun, Enhong Chen, and Yong Rui. 2014. GeoMF: joint geographical modeling and matrix factorization for point-of-interest recommendation. In *ACM SIGKDD*.

[13] Greg Linden, Brent Smith, and Jeremy York. 2003. Amazon. com recommendations: Item-to-item collaborative filtering. *IEEE Internet computing* (2003).

[14] Bin Liu, Yanjie Fu, Zijun Yao, and Hui Xiong. 2013. Learning geographical preferences for point-of-interest recommendation. In *ACM SIGKDD*.

[15] Michael Luca. 2016. Reviews, reputation, and revenue: The case of Yelp. com. (2016).

[16] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).

[17] Sahand Negahban, Sewoong Oh, and Devavrat Shah. 2012. Iterative ranking from pair-wise comparisons. In *NIPS*.

[18] Douglas W Oard, Jinmook Kim, et al. 1998. Implicit feedback for recommender systems. In *AAAI workshop on recommender systems*.

[19] Nihar B Shah and Martin J Wainwright. 2015. Simple, robust and optimal ranking from pairwise comparisons. *arXiv preprint arXiv:1512.08949* (2015).

[20] Xiaoyuan Su and Taghi M Khoshgoftaar. 2009. A survey of collaborative filtering techniques. *Advances in artificial intelligence* (2009).

[21] Hao Wang, Manolis Terrovitis, and Nikos Mamoulis. 2013. Location recommendation in location-based social networks using user check-in data. In *ACM SIGSPATIAL*.

[22] Hao Wang, Naiyan Wang, and Dit-Yan Yeung. 2015. Collaborative deep learning for recommender systems. In *ACM SIGKDD*.

[23] Hao Wang, SHI Xingjian, and Dit-Yan Yeung. 2016. Collaborative recurrent autoencoder: Recommend while learning to fill in the blanks. In *NIPS*.

[24] Fabian Wauthier, Michael Jordan, and Nebojsa Jojic. 2013. Efficient ranking from pairwise comparisons. In *ICML*.

[25] Wikipedia. 2017. Elo rating system — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Elo_rating_system&oldid=792554580

[26] Mao Ye, Peifeng Yin, Wang-Chien Lee, and Dik-Lun Lee. 2011. Exploiting geographical influence for collaborative point-of-interest recommendation. In *ACM SIGIR*.

[27] Yonghong Yu and Xingguo Chen. 2015. A survey of point-of-interest recommendation in location-based social networks. In *AAAI Workshop*.

[28] Fuzheng Zhang, Nicholas Jing Yuan, Kai Zheng, Defu Lian, Xing Xie, and Yong Rui. 2016. Exploiting dining preference for restaurant recommendation. In *WWW*.

[29] Yin Zheng, Bangsheng Tang, Wenkui Ding, and Hanning Zhou. 2016. A neural autoregressive approach to collaborative filtering. *arXiv preprint arXiv:1605.09477* (2016).