A Hybrid Neural Network and Hidden Markov Model for Time-aware Recommender Systems

Hanxuan Chen and Zuoquan Lin*
Department of Information Science, School of Mathematical Sciences, Peking University, Beijing 100871, China

Keywords: Hidden Markov Model, Neural Network, Collaborative Filtering, Recommender Systems.

Abstract: In this paper, we propose a hybrid model that combines neural network and hidden Markov model for time-aware recommender systems. We use higher-order hidden Markov model to capture the temporal information of users and items in collaborative filtering systems. Because the computation of the transition matrix of higher-order hidden Markov model is hard, we compute the transition matrix by deep neural networks. We implement the algorithms of the hybrid model for offline batch-learning and online updating respectively. Experiments on real datasets demonstrate that the hybrid model has improvement performances over the existing recommender systems.

1 INTRODUCTION

Recommender systems help the users find interesting items from a large amount of products. Time-aware recommender systems (TARS) (Campos et al., 2014) exploit temporal information and track the evolution of users and items that are beneficial for giving satisfactory recommendations. Hidden Markov Models (HMMs) and Neural Network (NNs) are two major approaches to TARS.

As a probabilistic approach, HMMs use the hidden states to describe the dynamic of users and items (Sahoo et al., 2012). In the literature, recommender systems based on HMMs commonly use first-order HMMs, i.e., the hidden states only depend on the last one state. For real world recommendations, the users' interests have long-term dependencies. For example, on an online-shopping website, the customers of maternity dress are likely to look articles for babies several months later. If there is not long-term affect in the transitions of the hidden states, this interest propagations will be covered by more frequent purchases of daily uses. Higher-order HMMs (HOHMMs) are natural way to model the problem of long-term dependencies. However, it is impractical for recommender systems depended on HOHMMs, since the cost to compute the state transitions is exponential for the length of dependencies.

With the development of deep learning, NNs have got much attention at recommender systems in recent years (Zhang et al., 2017). Recurrent neural networks (RNNs) are suitable for sequential data with long-term dependencies and successful in natural language process (Hochreiter and Schmidhuber, 1997; Cho et al., 2014). There are a number of recommender systems based on RNNs (Hidasi et al., 2016a; Hidasi et al., 2016b; Jannach and Ludewig, 2017; Chatzis et al., 2017; Wu et al., 2017; Soh et al., 2017; Devooght and Bersini, 2016; Chen et al., 2018). They concentrate on the sessions or the behavior sequences of users in which RNNs are used to model the sequential data. Although the RNNs are usable for the sequence of users' behavior with long-term dependencies, they have several shortages compared with HMMs. Firstly, most RNNs use the order of the users' behaviors, but neglect the time span between the behaviors. Secondly, there is not an overall time axis in RNNs to indicate the actual time point of each behaviors. RNNs can not describe the temporal relations of the behaviors from multiple users. Thirdly, a single RNN to model the sequences from all the users makes the model lack of personalization. Finally, HMMs have the meaning of the hidden states for the analysis of users' types, while RNNs can not have such meaning.

In this paper, we propose a hybrid model NHM that combines NN and HMM for time-aware recommender systems. We use HOHMM to capture the

*Correspondent author

Chen, H. and Lin, Z.
A Hybrid Neural Network and Hidden Markov Model for Time-aware Recommender Systems.
DOI: 10.5220/0007380402040213
In *Proceedings of the 11th International Conference on Agents and Artificial Intelligence (ICAART 2019)*, pages 204-213
ISBN: 978-989-758-350-6

temporal information of both users and items in collaborative filtering. Because the computation of the transition matrix of HOHMM is hard, we replace the transition matrix by NN. The hybrid model takes advantages of NN and HOHMM and has improvement efficiency and better precision for recommendations. We implement the algorithms of NHM for offline batch-learning and online updating respectively. Experiments on real datasets show that the hybrid approach has better performances over the existing recommender systems.

The rest of this paper are organized as follows. In section 2, we present the hybrid model NHM. In section 3, we apply NHM in collaborative filtering and provide the algorithms for the routines of the recommendation. In section 4, we show the improvement performance of our algorithms. In section 5, we discuss related works. Finally, we make conclusions in the concluding section.

2 THE HYBRID MODEL

2.1 Model

What follows, we present the hybrid model NHM. We introduce the temporal state random variable X_t and the temporal evidence random variable E_t, whose possible values are in $\{1,\dots,N\}$ and $\{1,\dots,K\}$ respectively. Let these variables follow HOHMM assumptions:

$$\begin{aligned} &P(X_t|X_{t-1},X_{t-2},\dots,E_{t-1},E_{t-2},\dots) \\ =&P(X_t|X_{t-1},X_{t-2},\dots,X_{t-L}) \\ =&P(X_0|X_{-1},X_{-2},\dots,X_{-L}), \end{aligned} \tag{1}$$

$$\begin{aligned} &P(E_t|X_t,X_{t-1},X_{t-2},\dots,E_{t-1},E_{t-2},\dots) \\ =&P(E_t|X_t) \\ =&P(E_0|X_0), \end{aligned} \tag{2}$$

where L is the order. In HOHMM, a transition matrix with N^L rows is needed to describe $P(X_t|X_{t-1},X_{t-2},\dots,X_{t-L})$ in (1) for the N^L possible value combinations of $X_{t-1},X_{t-2},\dots,X_{t-L}$.

We use a neural network to replace the transition matrix for the consideration of computation. Given the marginal distribution of X at the previous L time points,

$$P(X_{t-i}) = \overrightarrow{x_{t-i}}, \quad i = 1,2,\dots,L. \tag{3}$$

Then, $P(X_t)$ is defined as

$$P(X_t) = \phi(\overrightarrow{x_{t-1}},\overrightarrow{x_{t-2}},\dots,\overrightarrow{x_{t-L}}), \tag{4}$$

where $\phi(\cdot)$ is a neural network, in which the input and output are vectors whose dimensions are LN and N respectively. The output satisfies $0 \le \phi_j \le 1$ and $\sum_{j=1}^{N} \phi_j = 1$.

Note that we do not specify the type or structure of the neural network $\phi(\cdot)$. It can be a multilayer perceptron, an RNN or any NN that can deal with the requirement of such input and output. For instance, we take GRU (Cho et al., 2014) to implement ϕ, that can be represented as follows:

$$\begin{aligned} h_{t-L} &= \mathrm{GRU}(\overrightarrow{x_{t-L}}, \overrightarrow{0}), \\ h_i &= \mathrm{GRU}(\overrightarrow{x_i}, h_{i-1}), \quad t-L < i \le t, \\ \phi &= \mathrm{softmax}(\overrightarrow{h_t}). \end{aligned} \tag{5}$$

The observation and initial states of the model NHM are simply defined with a matrix and a vector as usual in HMM:

$$P(E_t = k|X_t = j) = B_{j,k}, \tag{6}$$

$$P(X_t = i) = \pi_i, \quad -L \le t < 0. \tag{7}$$

2.2 Inference

The inference task is to find the conditional distribution $P(X_t|E_{0:T-1} = \overrightarrow{e})$, given an evidence sequence $\overrightarrow{e} = [e_0,e_1,\dots,e_{T-1}]$. To do this, we use the neural network approximated forward-backward algorithm. It imitates the procedure of the forward-backward algorithm of HMM (Rabiner, 1989), and calculates an approximate $\gamma(t) = P(X_t|E_{0:T-1} = \overrightarrow{e})$.

Firstly, we take the forward steps:

$$\alpha(t) = \begin{cases} \pi, & -L \le t < 0, \\ \phi(\alpha'(t-1),\dots,\alpha'(t-L)), & 0 \le t; \end{cases} \tag{8}$$

where

$$\alpha'(t) = normalize(\alpha(t) \odot B_{:,e_t}). \tag{9}$$

The symbol $\odot$ means element-wise product of two vectors. $B_{:,e_t}$ means the e_t column of the matrix B. If there is not evidence at t (for $t < 0$ or $t \ge T$), $B_{:,e_t}$ is $normalize(\overrightarrow{1})$. The function $normalize(\cdot)$ is defined as follows:

$$normalize(\overrightarrow{v}) = \frac{\overrightarrow{v}}{\sum_{j=1}^{N} |v_j|}. \tag{10}$$

Then, we take the backward steps:

$$\beta(t) = \begin{cases} normalize(\overrightarrow{1}), & T \le t < T+L, \\ \psi(\beta'(t+1),\dots,\beta'(t+L)), & t < T; \end{cases} \tag{11}$$

where

$$\beta'(t) = normalize(\beta(t) \odot B_{:,e_t}). \tag{12}$$

$\psi(\cdot)$ is another neural network whose input and output have the same lengths as $\phi(\cdot)$. We call it the reverse sequence neural network of ϕ. Intuitively, if ϕ is the function $(\overrightarrow{v_L}, \overrightarrow{v_{L-1}}, \ldots, \overrightarrow{v_1}) \to \overrightarrow{v_{L+1}}$, ψ is the function $(\overrightarrow{v_2}, \overrightarrow{v_3}, \ldots, \overrightarrow{v_{L+1}}) \to \overrightarrow{v_1}$.

Finally, we find γ in the following:

$$\gamma(t) = normalize(\alpha'(t) \odot \beta(t)). \tag{13}$$

Another inference task is to find the distribution at the next time point:

$$\gamma(T) = P(X_T | E_{0:T-1} = \overrightarrow{e}) = \phi(\gamma(T-1), \ldots, \gamma(T-L)). \tag{14}$$

By the above equation, we can find $\gamma(T+1)$ by $\gamma(T), \ldots, \gamma(T-L+1)$ and find any $\gamma(t)$ for $t > T$.

2.3 Learning

We provide the learning algorithm of the model NHM. Suppose that we have some evidence sequences $ES = \{\overrightarrow{e}^{(1)}, \overrightarrow{e}^{(2)}, \ldots, \overrightarrow{e}^{(R)}\}$, where $\overrightarrow{e}^{(r)} = [e_0^{(r)}, e_1^{(r)}, \ldots, e_{T^{(r)}-1}^{(r)}]$. We need to learn the parameters $\theta = \{\phi, \psi, \pi, B\}$.

The learning algorithm works by inference-updating iterations. At first, we use some initial parameters θ in the inference steps and calculate $z = \{\alpha, \alpha', \beta, \beta', \gamma\}$ for every evidence sequences. Then, we use z in updating steps to find a better θ^*. Finally, we use θ^* in inference steps and carry on until we find satisfactory parameters.

The updating steps of π and B follow the Baum-Welch algorithm of HMM (Rabiner, 1989) as follows:

$$\pi^* = \text{normalize}(\sum_{r=1}^{R} \sum_{t=-L}^{-1} \gamma^{(r)}(t)), \tag{15}$$

$$B^*_{j,k} = \frac{\sum_{r=1}^{R} \sum_{t=0}^{T^{(r)}-1} 1_{e_t^{(r)}=k} \gamma_j^{(r)}(t)}{\sum_{r=1}^{R} \sum_{t=0}^{T^{(r)}-1} \gamma_j^{(r)}(t)}; \tag{16}$$

where

$$1_{e_t^{(r)}=k} = \begin{cases} 1, & e_t^{(r)} = k, \\ 0, & \text{else}. \end{cases} \tag{17}$$

To update the neural networks ϕ and ψ, we need to build training sets for them. The sampling method is as follows: we firstly select a random $\overrightarrow{e}^{(r)} \in ES$, then select a random t such that $0 \le t < T^{(r)}$. According to the input and expected output of ϕ and ψ, we add the following two examples to the training sets, respectively:

$$(\alpha'^{(r)}(t-1), \alpha'^{(r)}(t-2), \ldots, \alpha'^{(r)}(t-L)) \to \gamma^{(r)}(t), \tag{18}$$

$$(\beta'^{(r)}(t+1), \beta'^{(r)}(t+2), \ldots, \beta'^{(r)}(t+L)) \to \gamma^{(r)}(t). \tag{19}$$

After building the training sets, we call standard training algorithm for these neural networks.

3 RECOMMENDER SYSTEM

3.1 Model

What follows, we use NHM to build a model for recommender system based on collaborative filtering. Let $RS =< User, Item, Time, Level, Rating >$ be a recommender system, where

- $User$ is the set of users, and $user$ (or u) $\in User$ is a user.
- $Item$ is the set of items, and $item$ (or i) $\in Item$ is an item.
- $Time = \mathbb{Z}$ is the set of time points, where $\mathbb{Z}$ is the integers.
- $Level = \{1, 2, \ldots, N\}$ is the set of rating levels, where N is a given integer.
- $Rating$ is the set of ratings, where $rating = (user, item, time, level)$ (or $r = (u, i, t, l)$) $\in Rating$ is a rating, which means $user$ gives $item$ rating $level$ at $time$.

We make recommendations by analyzing the similarity of both users and items. We introduce the user types and the item types to describe common properties of users and items respectively. The users with the same type have similar tastes. There are J user types and K item types, where J and K are given integers respectively. Any user (or item) has a type at a specific time. The users and items change, so do their types. We use temporal random variables $X_{user,t}$ and $Y_{item,t}$ to represent their types at time t respectively. For ratings, we define a random variable $R_{user,item,t}$ for each triplet $(user, item, t)$. By some known ratings, $R_{user,item,t}$ are (partially) observed, while $X_{user,t}$ and $Y_{item,t}$ are hidden states. Namely, we have the following definitions:

- $UserType = \{1, 2, \ldots, J\}$ is the set of user types.
- $ItemType = \{1, 2, \ldots, K\}$ is the set of item types.
- $X_{user,t} \in UserType$ is the random variable for the $user$'s type at t.
- $Y_{item,t} \in ItemType$ is the random variable for the $item$'s type at t.

- $R_{user,item,t} \in Level$ is the random variable for the rating that the *user* gives the *item* at t.

For example, consider a recommender system where $User = \{u_1, u_2, u_3\}$, $Item = \{i_1, i_2\}$, $N = 5$, $Rating = \{\ (u_1, i_1, 1, 5),\ (u_2, i_2, 3, 3),\ (u_1, i_2, 5, 1),\ (u_3, i_1, 6, 4)\ \}$ and $J = 2$, $K = 3$. There are three user random variables $X_{u_1,t}$, $X_{u_2,t}$ and $X_{u_3,t}$ whose possible values are in $UserType = \{1, 2\}$. The two item random variables $Y_{i_1,t}$, $Y_{i_2,t}$ have possible values in $ItemType = \{1, 2, 3\}$. There are four observed rating random variables $R_{u_1,i_1,1} = 5$, $R_{u_2,i_2,3} = 3$, $R_{u_1,i_2,5} = 1$, and $R_{u_3,i_1,6} = 4$.

We consider how the users and items generate ratings. The probability $p_{j,k}$ means that a user with the j-th type meets an item with the k-th type. We use the binomial distribution $B(N-1, p_{j,k})$ to convert $p_{j,k}$ into discrete ratings:

$$\begin{aligned} &P(R_{u,i,t} = n \mid X_{u,t} = j, Y_{i,t} = k) \\ =&\Pr(n-1; N-1, p_{j,k}) \\ =&\binom{N-1}{n-1}(p_{j,k})^{n-1}(1-p_{j,k})^{N-n}. \end{aligned} \tag{20}$$

The transitions of $X_{u,t}$ and $Y_{i,t}$ are described with two L-order NHMs whose parameters are $\theta = \{\phi, \psi, \pi\}$ and $\widetilde{\theta} = \{\widetilde{\phi}, \widetilde{\psi}, \widetilde{\pi}\}$ respectively, where L is a given integer, θ is shared by all the users and $\widetilde{\theta}$ is shared by all the items. There is not matrix B in θ because $p_{j,k}$ plays the role of generating evidences.

For users, we assume that the user u has ratings at $M(u)$ time points $t_1 < t_2 < \ldots < t_{M(u)}$. For $1 \le m \le M(u)$, if we know $P(X_{u,t_{m-l}}) = \overrightarrow{x}_{u,t_{m-l}}$, $l = 1, 2, \ldots, L$, then $P(X_{u,t_m})$ is as follows:

$$P(X_{u,t_m}) = \phi(t_m - t_{m-1}, \overrightarrow{x}_{u,t_{m-1}}, \ldots, t_m - t_{m-L}, \overrightarrow{x}_{u,t_{m-L}}). \tag{21}$$

Compared with (4), we make an adjustment here. We only consider the time points that the user has ratings, since there are a lot of time points that the user has not ratings in recommender system. To indicate the actual time length between t_m and t_{m-l}, we add L dimensions in the input of the ϕ. For t_l with index $l \le 0$, we set $t_l = t_1 - \tau$ and $\overrightarrow{x}_{u,t_l} = \pi$, where τ is a small given time span.

Similarly, for $Y_{i,t}$, if the item i is rated at $M(i)$ time points $t_1 < t_2 < \ldots < t_{M(i)}$ and we know $P(Y_{i,t_{m-l}}) = \overrightarrow{y}_{i,t_{m-l}}$, $l = 1, 2, \ldots, L$ for $1 \le m \le M(i)$, then $P(Y_{i,t_m})$ is

$$P(Y_{i,t_m}) = \widetilde{\phi}(t_m - t_{m-1}, \overrightarrow{y}_{i,t_{m-1}}, \ldots, t_m - t_{m-L}, \overrightarrow{y}_{i,t_{m-L}}). \tag{22}$$

3.2 Inference

We consider that a user u who has ratings at $M(u)$ time points $t_1 < t_2 < \ldots < t_{M(u)}$. At time points t_m, the user gives $S(u, t_m)$ ratings. These ratings are given to items $i_1, i_2, \ldots i_{S(u,t_m)}$ and the levels are $n_1, n_2, \ldots, n_{S(u,t_m)}$, respectively.

Firstly, we calculate the conditional probability that a type-j user gives these $S(u, t_m)$ ratings at t_m, which is denoted as $b_{u,t_m,j}$. By (20), we have

$$\begin{aligned} &b_{u,t_m,j} \\ =&P(R_{u,i_1,t_m} = n_1, \ldots, R_{u,i_{S(u,t_m)},t_m} = n_{S(u,t_m)} \mid X_{u,t_m} = j) \\ =&\prod_{s=1}^{S(u,t_m)} P(R_{u,i_s,t_m} = n_s \mid X_{u,t_m} = j) \\ =&\prod_{s=1}^{S(u,t_m)} \sum_{k=1}^{K} \Pr(n_s - 1; N-1, p_{j,k}).P(Y_{i_s,t_m} = k). \end{aligned} \tag{23}$$

The vector $b_{u,t_m,:} = (b_{u,t_m,1}, \ldots, b_{u,t_m,J})$ represents the probability that the user generates these ratings at t_m with each type. It plays the role of $B_{:,e_t}$ in 2.2.

Algorithm 1 is the forward-backward algorithm for users. When we refer to t_l with index $l < 1$, we set $t_l = t_1 - \tau$ and $\alpha'_u(t_l) = \pi$. For t_l with index $l > M(u)$, we set $t_l = t_{M(u)} + \tau$ and $\beta'_u(t_l) = normalize(\overrightarrow{1})$.

Algorithm 1: Forward-backward algorithm for users.

function INFERENCE_USER(u)
  for m=1 to $M(u)$ **do**
    $\alpha_u(t_m) \leftarrow \phi(t_m - t_{m-1}, \alpha'_u(t_{m-1}), \ldots, t_m - t_{m-L}, \alpha'_u(t_{m-L}))$
    $\alpha'_u(t_m) \leftarrow normalize(\alpha_u(t_m) \odot b_{u,t_m,:})$
  for $m = M(u)$ to 1 **do**
    $\beta_u(t_m) = \psi(t_{m+1} - t_m, \beta'_u(t_{m+1}), \ldots, t_{m+L} - t_m, \beta'_u(t_{m+L}))$
    $\beta'_u(t_m) \leftarrow normalize(\beta_u(t_m) \odot b_{u,t_m,:})$
  for m=1 to $M(u)$ **do**
    $\gamma_u(t_m) \leftarrow normalize(\alpha'_u(t_m) \odot \beta_u(t_m))$

The inference steps for items are similar to the one for users. Consider an item i that is rated at $M(i)$ time points and has $S(i, t_m)$ rating at t_m. These ratings come from $u_1, \ldots, u_{S(i,t_m)}$ and the levels are $n_1, \ldots n_{S(i,t_m)}$. Then, we have

$$b_{i,t_m,k} = \prod_{s=1}^{S(i,t_m)} \sum_{j=1}^{J} \Pr(n_s - 1; N-1, p_{j,k}) P(X_{u_s,t_m} = j). \tag{24}$$

The forward-backward algorithm Inference_Item(i) for items is similar to **Algorithm 1** except for the function name, function input and indexes.

Then, the inference steps are taken separately for each user and each item. When we do it for user u, we assume that the probability $P(Y_{i_s,t_m} = k)$ in (23) is known. Similarly, $P(X_{u_s,t_m} = j)$ in (24) is assumed to be known for the inference steps of item i. In practical, we use $P(X_{u,t} = j) = (\gamma_u(t))_j$ and $P(Y_{i,t} = k) = (\gamma_i(t))_k$. The symbol $(\cdot)_j$ means the j-th element of the vector. We first initialize γ_u and γ_i, then take the inference steps for u and i alternately to update them.

3.3 Learning

We update the parameters $\theta = \{\phi,\psi,\pi\}$, $\widetilde{\theta} = \{\widetilde{\phi},\widetilde{\psi},\widetilde{\pi}\}$ and $p_{j,k}$ with the α, β and γ that we calculate in the inference steps. The $p_{j,k}$ is updated according to the parameter estimation of binomial distribution:

$$p^*_{j,k} = \frac{\sum_{(u,i,t,l)\in Rating}(l-1)\gamma_u(t)\gamma_i(t)}{\sum_{(u,i,t,l)\in Rating}(N-1)\gamma_u(t)\gamma_i(t)}. \tag{25}$$

The prior π is updated as the sum of the distribution of all the users $t_{-L+1},\ldots,t_0$. To find $\gamma(t_l)$ with $l \le 0$, we take backward steps several times to find β and then γ.

$$\pi^* = \text{normalize}(\sum_{u\in User}\sum_{l=-L+1}^{0}\gamma_u(t_l)). \tag{26}$$

In practical, a fixed $\pi = normalize(\overrightarrow{1})$ often has good performance because ϕ can generate the first L value of $\alpha(t_l)$ for $l > 0$ from $normalize(\overrightarrow{1})$.

The ϕ and ψ are trained with standard training algorithm of these neural networks as usually in deep learning. We only need to build the training set for the ϕ and ψ. To sample examples, we first select a random user $u \in User$, then select a random t_m from the $M(u)$ time points when the user generates ratings. The following two examples will be added to the training set of ϕ and ψ, respectively:

$$(\alpha'_u(t_{m-1}),\ldots,\alpha'_u(t_{m-L})) \to \gamma_u(t_m). \tag{27}$$

$$(\beta'_u(t_{m+1}),\ldots,\beta'_u(t_{m+L})) \to \gamma_u(t_m). \tag{28}$$

The updating of $\widetilde{\pi}$ and the training example sampling steps of $\widetilde{\phi}$ and $\widetilde{\psi}$ are the same as those of π, ϕ and ψ except for indexes.

3.4 Algorithms

We provide the algorithms for the routines of the model NHM, including batch learning, online updating and prediction, see **Algorithm 2**.

The batch learning algorithm learns the model from a set of training ratings. It firstly initializes the θ, $\widetilde{\theta}$, $p_{j,k}$, $\gamma_u(t)$, $\gamma_i(t)$ and empty training sets for each neural network. Then it takes the inference-learning iterations for a given loop number. In the inference steps, we randomly select an $u \in User$ or an $i \in Item$, and call the Inference_User or Inference_Item functions to update γ. In the learning steps, we calculate $p_{j,k},\pi,\widetilde{\pi}$, build the training set for each neural network and train them with standard training algorithm of them.

The online updating algorithm updates the model when receiving a rating $r = (u,i,t,l)$. We take the inference steps only for both the user and item that are related to this rating. Then we update $p_{j,k},\pi,\widetilde{\pi}$ with (25) and (26). The first equation is a fraction of two sums in *Rating*, and the second equation is the sum in *User* (or *Item*). We only need to subtract the previous contribution of the updated rating, user or item in these sums, and add their new contributions. We don't need to calculate the sums again. For the neural networks, we sample some examples from u and i, and update the $\phi,\psi,\widetilde{\phi},\widetilde{\psi}$ with these examples, i.e., only run several steps of the training algorithm of the neural networks on them.

The prediction algorithm makes the prediction about the rating that a user u will give to an item i at time t. We firstly calculate $\gamma_u(t)$ and $\gamma_i(t)$ with the γ in the model:

$$\begin{aligned}\gamma_u(t) = \phi(&t-t_{M(u)},\gamma_u(t-t_{M(u)}),\ldots,\\ &t-t_{M(u)-L+1},\gamma_u(t-t_{M(u)-L+1})),\end{aligned} \tag{29}$$

$$\begin{aligned}\gamma_i(t) = \widetilde{\phi}(&t-t_{M(i)},\gamma_i(t-t_{M(i)}),\ldots,\\ &t-t_{M(i)-L+1},\gamma_i(t-t_{M(i)-L+1})).\end{aligned} \tag{30}$$

Then we use $\gamma_u(t)$ and $\gamma_i(t)$ to calculate $P(R_{u,i,t} = n)$, which is denoted as $q_{u,i,t,n}$. The algorithm returns the vector $q_{u,i,t,:} = (q_{u,i,t,1},\ldots,q_{u,i,t,N})$ as follows:

$$q_{u,i,t,n} = \sum_{j=1}^{J}\sum_{k=1}^{K}(\gamma_u(t))_j(\gamma_i(t))_k \Pr(n-1;N-1,p_{j,k}). \tag{31}$$

4 EXPERIMENTS

4.1 Setup

We make three kinds of experiments to test the performance in different environments.

Classical experiment (CL): The ratings in the datasets are randomly divided into training set (80%) and test set (20%). The algorithms are trained by the training set to provide prediction about the ratings in the test set.

Algorithm 2: Model routines.

function BATCHLEARNING(*Rating*)
  initialize $\theta = \{\phi, \psi, \pi\}$, $\widetilde{\theta} = \{\widetilde{\phi}, \widetilde{\psi}, \widetilde{\pi}\}$, $p_{j,k}$
  initialize $\gamma_u(t)$, $\gamma_i(t)$
  initialize empty training sets for $\phi, \psi, \widetilde{\phi}, \widetilde{\psi}$
  for $l_1 = 1, 2, \ldots, LoopNum1$ **do**
    for $l_2 = 1, 2, \ldots, LoopNum2$ **do**
      Select a random $u \in User$
      Inference_User(u)
      Select a random $i \in Item$
      Inference_Item(i)
    Calculate $p_{j,k}, \pi, \widetilde{\pi}$ with (25) and (26)
    Build each training sets with (27) and (28)
    Train $\phi, \psi, \widetilde{\phi}, \widetilde{\psi}$ with these training sets

function ONLINEUPDATING(u, i, t, l)
  Inference_User(u)
  Inference_Item(i)
  Update $p_{j,k}, \pi, \widetilde{\pi}$ with (25) and (26)
  Sample examples from u and i with (27) and (28)
  Update $\phi, \psi, \widetilde{\phi}, \widetilde{\psi}$ with these examples

function PREDICTION(u, i, t)
  calculate $\gamma_u(t)$ with (29)
  calculate $\gamma_i(t)$ with (30)
  for $n = 1, 2, \ldots, N$ **do**
    calculate $q_{u,i,t,n}$ with (31)
  return $q_{u,i,t,:}$

Time-order Experiment (TO): The ratings in the datasets are reordered according to the time they are generated. We take the former 80% as the training set and the latter 20% as the test set. This is a reasonable experiment setup for time-related recommender systems for it ensures that the algorithms predict the future by the past.

Time-order Online Experiment (TOO): The ratings in the datasets are reordered according to the time they are generated and imported one by one to the algorithms. For every rating, the algorithms will first be required to give their predicted rating, and then updated their parameters every time they receive new coming rating. This method simulates the situation of real-world online recommendation applications and is suitable to evaluate time-aware and online algorithms.

The algorithms are judged by RMSE (root mean square error) and MAE (mean absolute error) scores of the ratings in the test sets.

The test ratings related to the users or items that have no ratings in the training set will not be counted in the evaluation scores. In this case, the parameters of some algorithms for the related users or items are not defined, or just initialized by small random values. So the algorithms can not provide reasonable ratings. For the same reason, in time-order online experiments, the very first ratings for every user and item (that means the user or item has not appeared before) will not be counted in the scores.

4.2 Datasets

The experiments use the MovieLens100k dataset and the Epinions dataset as illustrated in Table 1.

MovieLens100k (ML) (Harper and Konstan, 2015) is a famous movie dataset collected through the web site MovieLens from 1997 to 1998. It consists of 100,000 ratings from 943 users on 1,682 movies by 1-5 level ratings. The dataset is cleaned up by its provider to ensure each user has rated at least 20 movies.

Epinions (EP) (Tang et al., 2012a; Tang et al., 2012b) is an e-commerce dataset collected through the web site Epinions from 1999 to 2011. It consists of users' opinions on items in 27 categories by 1-5 level ratings. We made a clean-up preprocess to produce a 20-core dataset (each user rated at least 20 items, and each item was rated by at least 20 users), which has 2,874 users, 2,624 items and 122,361 ratings. The additional information in this dataset, such as item categories or users' trust relationships, are not used in our experiments.

Table 1: Datasets.

Datasets	Users	Items	Ratings
MovieLens100k	944	1,683	100,000
Epinions	2,874	2,624	122,361

4.3 Results

We compare our algorithms of the model NHM with several representative algorithms including matrix factorization, neighborhood, hidden Markov model and neural network.

Probabilistic Matrix Factorization (PMF) (Salakhutdinov and Mnih, 2007): classical matrix factorization model for recommender systems. The model is batch-learning and time-independent. To apply the batch-learning model in online experiment settings, we retrain the whole model every 1% of the experiment process.

Time Weight Collaborative Filtering (TWCF) (Ding and Li, 2005): time-related item-based neighborhood method. It firstly uses Pearson correlation coefficient to calculate the similarity of items, and then uses an exponential time weight function and the item similarity to make predictions. Similar to other neighborhood methods, the algorithm is naturally able to run in online experiments.

Collaborative Kalman Filter (CKF) (Gultekin and Paisley, 2014): hidden Markov model that uses Kalman Filter to make recommendations. The model has continuous time axis and continuous random variables for users, items and ratings. It has an online updating algorithm, and each update step only uses the most recent one rating.

Recurrent Recommender Networks (RRN) (Wu et al., 2017): recurrent neural network model. It uses two separate RNN to model the temporal changes of users and items, and uses inner product as the interaction of them. An additional matrix factorization model is included in the model as the stationary components. Like PMF, we retrain the batch-learning model every 1% of the online experiment process.

The hyperparameters of the tested algorithms are decided by grid search. Each hyperparameter is selected from a set of candidates to produce the best performance. In detail, the learning rate and regularization coefficient of PMF, RRN, NHM and the decay rate λ of TWCF are selected from $\{1, 0.1, 0.01, 0.001, 0.0001\}$. Because the authors of CKF used a hyperparameter $\sigma = 1.76$ in their experiments, we select σ from $\{0.5, 0.6, 0.7, \ldots, 2.5\}$ for CKF. We use LSTM to implement the neural networks in NHM and RRN. Another hyperparameter is latent vector length (for MF, CKF, RRN and NHM). Because it is directly related to the time and space cost, we set it 10 for every algorithm for the sake of fairness.

Tables 2 and 3 show the results of the experiments. The experiment setting CL, TO and TOO stand for classical experiments, time-order experiments and time-order online experiments, respectively. From the experiment results, NHM has the best performance on most of experiments. In the two experiments with classical settings, our model has slightly better performance than the classical algorithm PMF, while many other time-related models fail to exceed PMF in this setting. This shows our algorithm has a stable performance even in a setting that is difficult for time-aware algorithm. In time-order experiments and time-order online experiments, our algorithm has the best performance. On the last three experiments, our algorithm has obvious improvement over other algorithms with about 0.1 RMSE difference. Compared with the overall performance of the hidden Markov model CKF and the neural network model RRN, our model shows its advantage of integrating NNs into HOHMM to exploit the time-aware recommendations.

5 RELATED WORK

First-order HMM was introduced in (Sahoo et al., 2012) for recommender systems. It has a lot of applications such as people-to-people recommendation (Alanazi and Bain, 2013; Alanazi and Bain, 2016), sport videos (Sannchez et al., 2012) and sequence pattern mining (Gu et al., 2014; Le et al., 2016). In (Zhang et al., 2016b; Zhang et al., 2016a), the authors proposed a hidden Semi-Markov model for recommender systems. Their model can capture the duration that a user stays in a state. This extends the first-order HMM's dependency length from one time point to one staying state. But it can not describe long-term affects in which the user turns to other interest halfway and comes back at last. Another kind of HMM that has been applied in recommender systems is the Kalman Filter (Lu et al., 2009; Paisley et al., 2010; Chang et al., 2017; Gultekin and Paisley, 2014; Sun et al., 2012; Sun et al., 2014). This approach has continuous state space and continuous time axis. The dependency length in this model is extended to the last time point that the user has ratings. But it has the same problem about the long-term interest that is covered by other purchases.

There are increasing interest in application of deep learning for recommender systems, including AE (Wang et al., 2015; Liang and Baldwin, 2015) , RBM (Salakhutdinov et al., 2007), CNN (Ding et al., 2017) and MLP (He et al., 2017; Xue et al., 2017). We pay attention to RNN for it models time-aware recommender systems. The sequential models (Soh et al., 2017; Devooght and Bersini, 2016; Chen et al., 2018) regard the user behavior history as a sequence and apply RNN on it. These approaches use the sequential order of the behavior generated by users but neglect the time span between the records. The session-based models (Hidasi et al., 2016a; Hidasi et al., 2016b; Jannach and Ludewig, 2017; Chatzis et al., 2017) make recommendations on the session data generated by users. Similar to sequential models, they use the order of user behavior sequence to make recommendations. Because there are not overall time axes in these two kinds of approaches, they can not find temporal relationships between different users' records and can not describe the changes of multiple users at the same time. In (Wu et al., 2017), time-aware model with two separate RNNs for users and items was introduced , which is similar to our approach. Because RNNs do not provide the meaning of hidden states, it's difficult to choose the function for interaction between users and items. As a result, the RNN time-aware model have to add some stationary components, i.e., an additional matrix factorization model, to undertake the recommendation task.

Table 2: RMSE values of experiments.

Setting Dataset	CL ML	CL EP	TO ML	TO EP	TOO ML	TOO EP
PMF	0.925	**1.043**	1.023	1.149	1.062	1.206
TWCF	0.959	1.219	1.141	1.324	1.017	1.495
CKF	1.101	1.133	1.081	1.151	1.041	1.276
RRN	0.924	1.112	1.021	1.118	1.065	1.199
NHM	**0.923**	**1.043**	**1.020**	**1.066**	**0.952**	**1.065**

Table 3: MAE values of experiments.

Setting Dataset	CL ML	CL EP	TO ML	TO EP	TOO ML	TOO EP
PMF	0.733	0.808	0.816	0.885	0.830	0.923
TWCF	0.756	0.958	0.891	1.004	0.794	1.108
CKF	0.865	0.911	0.833	0.895	0.816	0.995
RRN	0.936	0.862	0.829	0.848	0.837	0.929
NHM	**0.728**	**0.803**	**0.808**	**0.812**	**0.756**	**0.823**

There are works that integrate NNs and HMMs in other research fields. In speech recognition(Bourlard and Wellekens, 1990), the authors proposed a widely-used hybrid NN-HMM model which uses NN to improve the discriminating power of HMM. In their work, a neural network was considered as a general form of Markov model and used to capture contextual input information. However, their object and model structure are quite different with ours. In molecular biology (Baldi and Chauvin, 1995), the authors applied NN to reduce the parameter number of HMM. They handled the problem where there are a huge number of hidden states in HMM and used an NN to reduce the parameter size. They discussed the long-range dependencies problem, but chose multiple first-order HMMs rather than higher-order HMM.

6 CONCLUSIONS

We proposed a hybrid NN and HMM model NHM that takes advantages of NN and HOHMM for time-aware recommender systems. It can describe long-term dependencies of both users and items and is explainable to the interactions between users and items in collaborative filtering. We provided the algorithms of NHM for offline batch-learning and online updating that have better performance than the existing recommender systems.

We did not specify the type or structure of NNs in our models for the consideration of generality. If we use RNN implementation in our model to deal with unfixed length input sequences, the model can be extended to unfixed order hidden Markov models. That is, we do not need to assign the order of the hidden Markov models. The other hyperparameters such as the numbers of the user types and item types can have a way to be adjusted automatically. We consider the approach of NHM by combining NNs and HOHMMs are general enough to various recommender systems.

ACKNOWLEDGMENTS

The work is supported by Natural Science Fund of China under numbers 61672049/61732001.

REFERENCES

Alanazi, A. and Bain, M. (2013). A people-to-people content-based reciprocal recommender using hidden markov models. In *Proceedings of the 7th ACM conference on Recommender systems*, pages 303–306.

Alanazi, A. and Bain, M. (2016). A Scalable People-to-People Hybrid Reciprocal Recommender Using Hidden Markov Models. *The 2nd International Workshop on Machine Learning Methods for Recommender Systems*.

Baldi, P. and Chauvin, Y. (1995). Protein modeling with hybrid hidden markov model/neural network architectures. In *International Conference on Intelligent Systems for Molecular Biology*.

Bourlard, H. and Wellekens, C. J. (1990). Links between markov models and multilayer perceptrons. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(12):1167–1178.

Campos, P. G., Díez, F., and Cantador, I. (2014). Time-aware recommender systems: A comprehensive survey and analysis of existing evaluation protocols. *User Modeling and User-Adapted Interaction*, 24(1–2):67–119.

Chang, S., Yin, D., Chang, Y., Hasegawa-johnson, M., and Huang, T. S. (2017). Streaming Recommender Systems. In *International Conference on World Wide Web*, pages 381–389.

Chatzis, S., Christodoulou, P., and Andreou, A. S. (2017). Recurrent latent variable networks for session-based recommendation. In *the 2nd Workshop on Deep Learning for Recommender Systems*.

Chen, X., Xu, H., Zhang, Y., Tang, J., Cao, Y., Qin, Z., and Zha, H. (2018). Sequential recommendation with user memory networks. In *ACM International Conference on Web Search and Data Mining*, pages 108–116.

Cho, K., Van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. *Computer Science*.

Devooght, R. and Bersini, H. (2016). Collaborative filtering with recurrent neural networks.

Ding, D., Zhang, M., Li, S. Y., Tang, J., Chen, X., and Zhou, Z. H. (2017). Baydnn: Friend recommendation with bayesian personalized ranking deep neural network. In *Conference on Information and Knowledge Management*, pages 1479–1488.

Ding, Y. and Li, X. (2005). Time Weight Collaborative Filtering. In *Proceedings of the ACM International Conference on Information and Knowledge Management*, pages 485–492.

Gu, W., Dong, S., and Zeng, Z. (2014). Increasing recommended effectiveness with markov chains and purchase intervals. *Neural Computing & Applications*, pages 1153–1162.

Gultekin, S. and Paisley, J. (2014). A Collaborative Kalman Filter for Time-Evolving Dyadic Processes. In *IEEE International Conference on Data Mining*.

Harper, F. M. and Konstan, J. A. (2015). The MovieLens Datasets: History and Context. *ACM Transactions on Interactive Intelligent Systems*.

He, X., Liao, L., Zhang, H., Nie, L., Hu, X., and Chua, T.-S. (2017). Neural collaborative filtering. In *Proceedings of the 26th International Conference on World Wide Web*.

Hidasi, B., Karatzoglou, A., Baltrunas, L., and Tikk, D. (2016a). Session-based recommendations with recurrent neural networks. In *the International Conference on Learning Representations*.

Hidasi, B., Quadrana, M., Karatzoglou, A., and Tikk, D. (2016b). Parallel recurrent neural network architectures for feature-rich session-based recommendations. In *ACM Conference on Recommender Systems*, pages 241–248.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Jannach, D. and Ludewig, M. (2017). When recurrent neural networks meet the neighborhood for session-based recommendation. In *ACM Conference on Recommender Systems*, pages 306–310.

Le, D. T., Fang, Y., and Lauw, H. W. (2016). Modeling sequential preferences with dynamic user and context factors. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*.

Liang, H. and Baldwin, T. (2015). A probabilistic rating auto-encoder for personalized recommender systems. In *ACM International on Conference on Information and Knowledge Management*, pages 1863–1866.

Lu, Z., Agarwal, D., and Dhillon, I. S. (2009). A Spatio-temporal Approach to Collaborative Filtering. In *Proceedings of the Third ACM Conference on Recommender Systems*, pages 13–20.

Paisley, J., Gerrish, S., and Blei, D. (2010). Dynamic modeling with the collaborative kalman filter. In *NYAS 5th Annual Machine Learning Symposium*.

Rabiner, L. R. (1989). A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Readings in Speech Recognition*, pages 267–296.

Sahoo, N., Singh, P. V., and Mukhopadhyay, T. (2012). A Hidden Markov Model for Collaborative Filtering. *Mis Quarterly*, 36(4):1329–1356.

Salakhutdinov, R. and Mnih, A. (2007). Probabilistic Matrix Factorization. In *International Conference on Neural Information Processing Systems*, pages 1257–1264.

Salakhutdinov, R., Mnih, A., and Hinton, G. (2007). Restricted boltzmann machines for collaborative filtering. In *International Conference on Machine Learning*, pages 791–798.

Sannchez, F., Alduan, M., Alvarez, F., Menendez, J. M., and Baez, O. (2012). Recommender system for sport videos based on user audiovisual consumption. *IEEE Transactions on Multimedia*, 14(6):1546–1557.

Soh, H., Sanner, S., White, M., and Jamieson, G. (2017). Deep sequential recommendation for personalized adaptive user interfaces. In *International Conference on Intelligent User Interfaces*.

Sun, J. Z., Parthasarathy, D., and Varshney, K. R. (2014). Collaborative kalman filtering for dynamic matrix factorization. *IEEE Transactions on Signal Processing*.

Sun, J. Z., Varshney, K. R., and Subbian, K. (2012). Dynamic matrix factorization: A state space approach. In *IEEE International Conference on Acoustics, Speech and Signal Processing*.

Tang, J., Gao, H., and Liu, H. (2012a). mtrust: discerning multi-faceted trust in a connected world. In *Proceedings of the fifth ACM international conference on Web search and data mining*, pages 93–102. ACM.

Tang, J., Liu, H., Gao, H., and Das Sarmas, A. (2012b). etrust: Understanding trust evolution in an online world. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 253–261. ACM.

Wang, H., Wang, N., and Yeung, D. Y. (2015). Collaborative deep learning for recommender systems. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1235–1244.

Wu, C. Y., Ahmed, A., Beutel, A., Smola, A. J., and Jing, H. (2017). Recurrent recommender networks. pages 495–503.

Xue, H. J., Dai, X. Y., Zhang, J., Huang, S., and Chen, J. (2017). Deep matrix factorization models for recommender systems. In *International Joint Conference on Artificial Intelligence*, pages 3203–3209.

Zhang, H., Ni, W., Li, X., and Yang, Y. (2016a). A Hidden Semi-Markov Approach for Time-Dependent Recommendation. In *Pacific Asia Conference on Information Systems*.

Zhang, H., Ni, W., Li, X., and Yang, Y. (2016b). Modeling the Heterogeneous Duration of User Interest in Time-Dependent Recommendation: A Hidden Semi-Markov Approach. *IEEE Transactions on Systems, Man and Cybernetics Systems*, pages 2168–2216.

Zhang, S., Yao, L., and Sun, A. (2017). Deep learning based recommender system: A survey and new perspectives. *ArXiv e-prints*.