Temporal Behavioral Modeling for Recommendation - A Deep Learning Approach

Abstract: In this paper, we propose a novel method to identify and learn the temporal patterns of users' interests in interacting with items. Prediction of next basket, which is the set of items given previous basket of a user is a center of attention in recent recommendation systems. The problem is modeled as a sequence prediction model. The deep learning (DL) is a good candidate for the sequence prediction which is adopted for this task here. We define the combination of Recurrent Neural Networks (RNN) which has proved its usefulness for sequence learning, in combination with dimensionality reduction algorithms to overcome the sparsity problem including auto-encoder (AE), PCA and t-Distributed Stochastic Neighbor Embedding (t-SNE). Applying our method on open-source datasets reveals its effectiveness compared to an MLP-based next basket prediction as a benchmark.

Introduction

The collaborative Filtering (CF) is mostly rely on other users' information and fails to present and show a complete and a precise personalization based on users past interactions.

In addition, the traditional CF focuses only on the global change and everything is assumed to be fixed for different time span. Global model would be ineffective to explain the underlying structure of the system. Content-based model suffers from the problem of serendipity by recommending the very similar items to users. Hence, the temporal recommendation has been proposed by a large number of researchers to understand the user's drifting overtime behavior as well as looking at the local pattern changes in users' shopping behavior.

2 Related works

Several techniques have been proposed to create an optimum model for the temporal recommendation systems.

Some of them used discrete time-windows (probably making a very sparse data) and other used a fixed global continuous decay functions. For instance, in [3], the authors utilized the decay function f (x) = e□at to address the effect of users' past interaction. However, the control parameter has been considered fixed and global.

Liu et al. [7] defined a modified version of the previous weighting function and assumed that the old purchases are less correlated with recent users' interests. They have also defined additional similar decay function for rating prediction. The main contribution of the paper was introducing an incremental approach for updating similarity when a new dataset is incorporated into the model and this incremental approach is efficient in terms of computation time and memory usage. Note that applying low weight values for the past interaction leads to lose lots of useful information. Time-aware recommendation based on accuracy measured up to the computation time was proposed by [6]. In this paper, the authors have adjusted the number of the nearest neighbors for each users to optimize the precision.

However, the number of the neighbors was chosen from the limited number of potential values. In [9], Xiong et al. used a factorization model (SVD-like model) on a 3-D temporal utility matrix in which time is incorporated. To get the latent factors, they used Probabilistic Matrix Factorization

(PMF) algorithms (objective was maximum likelihood and the method was gradient descent). For scalability, since the problem is not scalable, they used Markov chain Monte

Carlo (MCMC) for sampling the data and run the model on the sampled data.

There are some papers working on similarities between the items and users by incorporating time into the model. They mostly focused on the drawbacks of the common similarity measures like cosine similarity and Pearson similarities and proposed a new similarity approach [4]. For instance, in [2], the authors extracted the semantics hidden in the interest sequences by the length of users longest common sub-IS (LCSIS) and the count of users total common sub-IS (ACSIS). They also observed regularities/ seasonality/spikes in temporal patterns of user interests which failed to be captured by static weighting functions proposed in previous works.

Sequential recommenders based on Markov chains which produce recommendations exploiting sequential information and recent purchases is introduced in [1]. Dynamic REcurrent bAsket Model (DREAM) in [2] exploits general customers' preferences and sequential information by using recurrent neural networks. A different hybrid approach is described in [4] where is developed a probability model by merging Markov chain and association patterns. Our basket representation for RNN part here is different from DREAM which in [2], average pooling or max pooling lose basket information.

In the following section, we propose a novel temporal modeling of user preferences that captures the context specific patterns through training based on historical data.

The model is able to predict the user specific temporal behavioral patterns.

Problem statement:

The problem that we are trying to solve is predicting the users' basket at time t where the true basket at time t is expressed as B_u^t from the last K baskets denoted by $B_u^{t-1}, \dots, B_u^{t-K}$. The predicted basket at time t is expressed as $\hat{B}_u^t$. By definition, basket is the set of items. This problem is basically sequence prediction therefore the recursive neural network (RNN) which is appropriate for sequence prediction is adopted here as the primary option. Constructing a fixed-length vector for baskets (will call it basket encoding) which uniquely depends on: 1- the items of the basket, 2-the number of items in the basket while have low dimension are important. To reduce the encoded basket dimension, the auto-encoder (AE) is deployed here. As the benchmark, we adopted multi-layer-perceptron (MLP) to compare the performance of AE +RNN with MLP. The embedded basket (encoded basket) is denoted by V_u^t for basket B_u^t. One-hot encoding for basket B_u^t is an example where V_u^t in this case a long vector whose elements corresponds to the items in is B_u^t is one and the rest is zero. Important aspects of the problem here is V_u^t is sparse because a user purchases only a few items from a large set of all items. AE is used here to overcome sparsity and provide lower dimension representation of basket B_u^t which also reduced the computation times for RNN.

Problem Formulation:

Next basket prediction can be viewed as sequence prediction. The ultimate goal here is how it is possible to have a model that it incorporates time into recommendation systems to this model adaptively learns time evolution and user preferences. This topic is somehow similar to next word prediction in a sentences by having prior words. The topic here is more complex that word prediction because of higher dimensionality and sparsity by its nature.

Assume user u at time $t-K$ has the basket B_u^{t-K} and let $|I|$ represents the total number of item. Each item is encoded by a $d \times 1$ vector (so-called embedding size). One-hot encoding which is a binary encoding would be an example. Distributed representation [] would be another example. The encoding goal here is to generate an appropriate vector representation of items in an item corpus. Appropriate means, for example, averaging or max pooling over s of vectorized items does not correspond to the vector representation of another item. With embedding size of d and total $|I|$items, each basket at time t is denoted as a $d|I| \times 1$ vector. Hence, the basket vector representation at time $t-i$ would be the input of the RNN and the basket vector representation at time $t-i+1$ is defined as the output of the RNN. The only drawback here is sparsity of the input/output vectors. The item mapping can be learned or be predefined in our model. To overcome the sparsity problem here, the dimensionality reduction including PCA, t-SNE and Auto-Encoder (AE) can be applied and investigate the appropriate one.

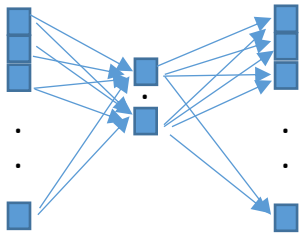

Fig. 1. Auto Encoder Neural Net Representation

For auto-encoder if $\boldsymbol{x} \in \mathbb{R}^{d|I| \times 1}$ ($\boldsymbol{x} = V_u^t$ here) denotes the input and $\hat{\boldsymbol{x}} \in \mathbb{R}^{d|I| \times 1}$ the output, the objective function is:

$$J(\boldsymbol{W_1}, \boldsymbol{W_2}, \boldsymbol{b_1}, \boldsymbol{b_2}) = \sum_{\boldsymbol{x} \in \boldsymbol{S}} \|\boldsymbol{x} - \hat{\boldsymbol{x}}\|^2$$

$$= \sum_{\boldsymbol{x} \in \boldsymbol{S}} \|\boldsymbol{x} - g(\boldsymbol{W_2} f(\boldsymbol{W_1 x} + \boldsymbol{b_1}) + \boldsymbol{b_2})\|^2$$

Where $\boldsymbol{W_1} \in \mathbb{R}^{k \times d|I|}$, $\boldsymbol{b_1} \in \mathbb{R}^{k \times 1}$ and $\boldsymbol{W_2} \in \mathbb{R}^{d|I| \times k}$, $\boldsymbol{b_2} \in \mathbb{R}^{d|I| \times 1}$ are the weights and biases for the first and second layer of auto-encoder and they are all the learning parameters. The set $\boldsymbol{S}$ denotes the training set. Then RNN is deployed to be trained based on $f(\boldsymbol{W_1 x} + \boldsymbol{b_1})$ and be tested based on $g(\boldsymbol{W_2 z} + \boldsymbol{b_2})$ where $\boldsymbol{z}$ is the estimated of next basket from RNN.

Hyper parameters, including k in AE, d for basket encoding and those for RNN and MLP, which will be introduced in this and following section, can be arbitrary chosen but it is shown in [3] that low k introduces high RMSE and high k introduces overfitting.

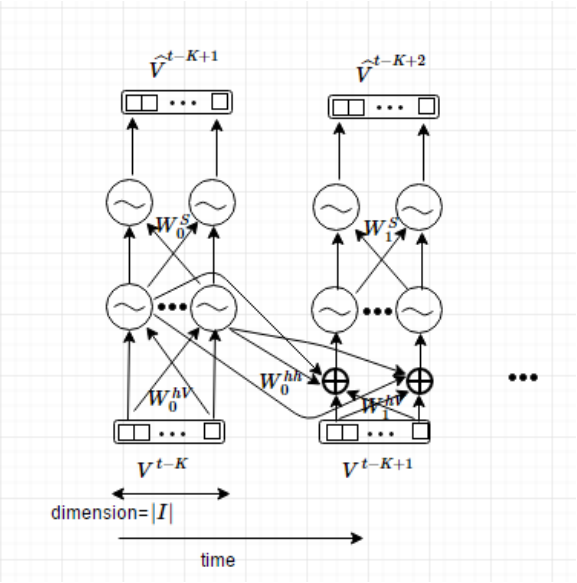

Fig 2. RNN configuration for Next Basket Prediction

The recursive neural network (RNN) for our problem based on the encoded input vectors $V^{t-K}, \dots, V^{t-1}$ is graphically represented in Fig 2. (The functions f, g here for RNN are not related to those introduced in AE). We dropped the subscript u for basket representation (i.e. V^{t-K} instead of V_u^{t-K}) to express that the RNN is trained for baskets only information but not user's information or meta-data for RNN. Mathematically the relationship between input vector, hidden layers ($h^{t-K} \dots h^{t-1}$) and output of RNN is expressed as:

$$h^{t-1} = f(W^{hh} h^{t-2} + W^{hV} V^{t-1})$$
$$\hat{V}^t = y_t = W^S g(h^{t-1})$$

h^{t-K-1} is an initialization vector for the hidden layer at time $t-K-1$. By assigning the *softmax* activation function after output layer in Fig 2, it is possible to train the RNN to provide the probability of having item i in basket B^t.

Benchmark:

Let's $U = \{u_1, u_2, \dots, u_{|U|}\}$ represents the user set and $I = \{i_1, i_2, \dots, i_{|I|}\}$ item set. For each user, the ordered buying behavior sequence is denoted by $S_u = (B_u^1, B_u^2, \dots, B_u^{t-1})$ where B_u^t is the set of items purchased by user u at time t. The goal here is to predict B_u^t from the sequence S_u. This problem can be summarized as sequence prediction problem. This mathematical problem can be represented by a neural network if the input layer embedded the user and its last k baskets information, the hidden layer transforms the feature vector to a space by a non-linear activation function and the output layer with $|I|$ neurons with the *softmax* activation function. After training with appropriate cost function which will be explained shortly, each output neuron indicates the probability of an item being in the basket B_u^t. Assume the neural network be multi-layer perceptron (MLP). The first layer as the embedding layer provides a vector corresponds to each user as $\boldsymbol{u} \in \mathbb{R}^{d_u}$, a vector for each item in basket B_u^{t-k} as $\boldsymbol{v} \in \mathbb{R}^{d_i}$ and do max-pooling or average-pooling, and finally concatenate them all. Therefore, the input neuron would be $d_u + k * d_i$. Defining the weights for the first layer as $\boldsymbol{W}_1 \in \mathbb{R}^{l \times (d_u + k * d_i)}$ and its bias $\boldsymbol{b}_1 \in \mathbb{R}^{l \times 1}$. If the activation function for the first layer is $tanh$, then the output for the first layer is $\boldsymbol{h}_2 = \tanh(\boldsymbol{W}_1 \boldsymbol{h}_1 + \boldsymbol{b}_1)$. The second layer with l input and $|I|$ output neuron, the output would be $\boldsymbol{s} = \boldsymbol{W}_2 \boldsymbol{h}_2 + \boldsymbol{b}_2$, where $\boldsymbol{W}_2 \in \mathbb{R}^{|I| \times l}$, $\boldsymbol{b}_2 \in \mathbb{R}^{|I| \times 1}$.

Defining the *softmax* activation function, each output neuron provides the following probability:

$$p(i \in B_t | u, B_{t-1}, \dots, B_{t-k}) = \frac{e^{s_j}}{\sum_{j=1}^{|I|} e^{s_j}}$$

For training the MLP, we propose the negative log likelihood as the cost function:

$$\min -\sum_u \sum_t \sum_i log\ (p(i \in B_t | u, B_{t-1}, \dots, B_{t-k})) + \lambda_1 ||\boldsymbol{W}_1||_2^2 + \lambda_2 ||\boldsymbol{W}_2||_2^2$$

The decision variables are: $\boldsymbol{W}_1, \boldsymbol{W}_2, \boldsymbol{b}_1$ and $\boldsymbol{b}_2$. For simplicity $l, d_u$and d_i can all set to the same value denoted as d.

The output layer activation function as softmax provides the probability of an item being in the next basket given the history of the user.

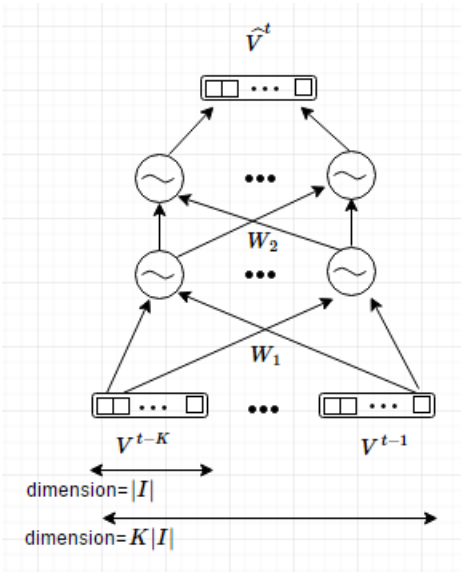

Fig 3. MLP configuration for next basket prediction

Experiment and Results:

We apply the next basket prediction by RNN, RNN+AE and MLP on Ta-feng dataset. Ta-feng dataset, containing 817741 transactions, 119578 baskets belonging to 32266 users and 23812 items. The three deep learning modeled are evaluated by recall, precision and F1-measure. By definition:

$$r_u^t = \frac{|B_u^t \cap \hat{B}_u^t|}{|B_u^t|}$$

$$p_u^t = \frac{|B_u^t \cap \hat{B}_u^t|}{|\hat{B}_u^t|}$$

$$F1_u^t = \frac{2p_u^t r_u^t}{p_u^t + r_u^t}$$

Hyper parameter of our RNN and AE:

All Python codes for next basket prediction are in this GitHub links: . It has two main parts. 1- Basket generation from the raw Ta-feng dataset 2- prepare the train and test data sets (with arbitrary ratio as we set 0.8 for training and 0.2 for test) and neural net configuration, compile and fit. Because of large size of V_u^t, the *fit_generator* method for Keras has been used to avoid in-memory issue.

basket_generation.py:

Input: Raw transaction data

Output: A file which each line is a dictionary {'uid': [[$i_j, \dots, i_l$] ... [$i_j, \dots, i_l$]]} ordered by time

```
def build_model(input_dim, input_length=None, hidden_units=[100, 100], dropout=0.1,
output_activation='sigmoid'):
    model = Sequential()
    model.add(LSTM(hidden_units[0], input_shape=(input_length, input_dim),
return_sequences=True))
    model.add(Dropout(dropout))
    for h in hidden_units[1:]:
        model.add(LSTM(h, return_sequences=True))
        model.add(Dropout(dropout))
        model.add(TimeDistributed(Dense(input_dim, activation=output_activation)))
    return model


model = build_model(input_dim=len(unique_item_id))
model.compile(loss='binary_crossentropy', optimizer='rmsprop', metrics=['accuracy'])
uid_train, uid_test = u_c_id_for_train_test(0.8)
model.fit_generator(x_y_train(uid_train), samples_per_epoch=len(uid_train),
nb_epoch=1)
```

References

[1] C. Chand, A. Thakkar, and A. Ganatra. Sequential pattern mining: Survey and current research challenges. IJSCE, 2(1):185–193, 2012.

[2] F. Yu, Q. Liu, S. Wu, L. Wang, and T. Tan. A dynamic recurrent model for next basket recommendation. In SIGIR, pages 729–732. ACM, 2016.

[3] S. Sedhain, A. K. Menon, S. Sanner, and L. Xie. 2015. AutoRec: Autoencoders Meet Collaborative Filtering. In *Proceedings of the 24th International Conference on World Wide Web* (WWW '15 Companion). ACM, New York, NY, USA, 111-112.

[4] P. Wang, J. Guo, and Y. Lan. Modeling retail transaction data for personalized shopping recommendation. In CIKM, pages 1979–1982.ACM, 2014.