Session-Based Path Prediction by Combining Local and Global Content Preferences

Kushal Chawla[1(✉)] and Niyati Chhaya[2]

[1] University of Southern California, Los Angeles, USA
kchawla@usc.edu
[2] Big Data Experience Lab, Adobe Research, Bengaluru, India
nchhaya@adobe.com

Abstract. Session-based future page prediction is important for online web experiences to understand user behavior, pre-fetching future content, and for creating future experiences for users. While webpages visited by the user in the current session capture the users' local preferences, in this work, we show how the global content preferences at the given instant can assist in this task. We present **DRS-LaG**, a Deep Reinforcement Learning System, based on Local and Global preferences. We capture these global content preferences by tracking a key analytics KPI, the number of views. The problem is formulated using an agent which predicts the next page to be visited by the user, based on the historic webpage content and analytics. In an offline setting, we show how the model can be used for predicting the next webpage that the user visits. The online evaluation shows how this framework can be deployed on a website for dynamic adaptation of web experiences, based on both local and global preferences.

1 Introduction

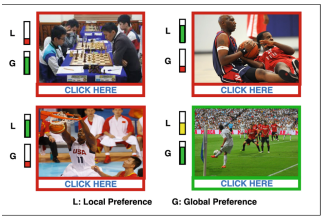
Users expect varied outcomes from their web experiences. Enterprises aim to create digital experience that not only cater to user intent but also help to improve their own business metrics. Given the variety of content, manual creation of customized and adaptive experiences is infeasible. Session-based future path prediction is necessary to understand user needs, pre-fetch future content, or even for adapting future experiences. Users' content creation and consumption patterns define their intent and needs. Their web tracks; i.e. the path that the user takes during their web journey is an essential ingredient for defining their interests and goals. In this work, we aim to create user intent models leveraging their consumption patterns combined with their website footprint to predict the potential user path and content needs.

Extensive studies have been conducted in the related space of recommender systems using traditional [2], deep-learning [6,7], and reinforcement learning [8]

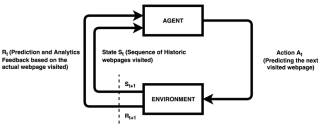
K. Chawla—Work done when author was a full-time researcher at Adobe Research.

J. M. Jose et al. (Eds.): ECIR 2020, LNCS 12036, pp. 126–132, 2020.
https://doi.org/10.1007/978-3-030-45442-5_16

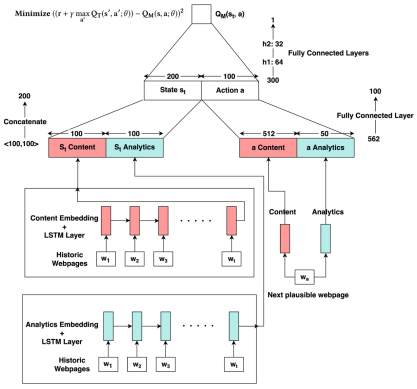
based techniques on both historic user-item interactions and session behavior. While the historic webpages visited in a session capture the users' local preferences, this work shows that the instantaneous global content preferences can further assist in understanding the future behavior of the users. We describe one such scenario in Fig. 1a. Specifically, we present a Deep Reinforcement Learning (RL) System, based on Local and Global preferences (**DRS-LaG**). Given the historic webpage content and analytics in a user session, our agent predicts the future preferences of the user. The model is trained on offline logs of a sports news website. Through offline evaluations, we show how the proposed model can be used to predict the next page user will go to. Our online evaluation shows how the predictions can be used to adapt future experiences of the users. RL allows our system to tackle the dynamic user preferences in news domain, while also incorporating expected future rewards when deployed in an online environment.



(a) Illustrative example, benefit of incorporating global content preferences into modelling user behavior.



(b) Model setup



(c) Architecture Diagram

Fig. 1. Internal workings of the proposed DRS-LaG framework.

2 DRS-LaG: Proposed Framework

Problem Formulation: We define an agent which models a user's session-level behavior to predict the next webpage user visits, based on the content and instantaneous analytics of the webpages visited in the current session. Since the predictions capture user preferences, they can then be recommended to the user or used to adapt future webpage experiences. At each timestep, the user (environment) provides feedback on the actions taken by the agent in the form of rewards. The agent is trained on offline session-level logs extracted from a sports news website. We illustrate this setup in Fig. 1b.

The task is modeled as a Markov Decision Process (MDP) with the tuple ($\mathcal{S}$, $\mathcal{A}$, $\mathcal{P}$, $\mathcal{R}$, γ): (1) State space $\mathcal{S}$: captures the current local and global content

Algorithm 1. Offline Training algorithm for our agent

Initialize replay memory M, Q-value model $Q_M(s,a)$ with random weights, target model $Q_T(s,a)$ with same wts as $Q_M(s,a)$.
Initialize webpage (action) pool P, webpage analytics A(action, kpi, time interval)
for e = 1,E **do** ▷ episodes or user web sessions in chronological order
  Reset environment state vector to a zero vector.
  for t= 1,T **do** ▷ timesteps in the current session
    Observe the current state s_t
    for n = 1,N **do** ▷ negative actions
      Sample a negative action a from pool P
      Observe the next state s_{t+1}, reward r
      Store transition (s_t, a, r, s_{t+1}, done=1) in M
      ▷ session ends after a negative sample
    end for
    Get the correct action a from the offline logs
    Observe the next state s_{t+1}, reward r
    Set done=1 if t==T, else 0
    Store transition (s_t, a, r, s_{t+1}, done=done) in M
    Update $s_t \leftarrow s_{t+1}$
    if M.length > batch-size **then**
      Sample a minibatch of transitions from M
      Set y =
$$\begin{cases} r, & \text{done=1} \\ r + \gamma \max_{a'} Q_T(s', a'; \theta), & \text{done=0} \end{cases}$$
      Minimize $(y - Q_M(s,a;\theta))^2$
    end if
  end for
  Update webpage pool P and webpage analytics A.
  Update target model after fixed number of iterations.
end for

preference, (2) Action space $\mathcal{A}$: set of all webpages, (3) Transition probabilities $\mathcal{P}$: probability $p(s'|s,a)$ to move to state s' by taking an action a in the state s, (4) Rewards $\mathcal{R}$: capturing the feedback received by the agent after taking a particular action and, (5) γ: the discount factor for future rewards in the current user session. The goal is to learn a policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ to maximize the cumulative reward of the system.

To deal with the dynamic action spaces, we use a Deep Q-Learning model-free approach. Figure 1c shows our architecture. Given a state-action pair, the network outputs the corresponding Q-value $Q(s,a)$. The optimal Q-value $Q^*(s,a)$ should follow the Bellman equation [1]: $Q^*(s,a) = E_{s'}[r + \gamma \max_{a'} Q^*(s',a')|s,a]$, where r is the corresponding reward for the given state-action pair.

Actions: Representing Webpages. The agent actions correspond to various webpages or URLs on the given website. Given the current state of a user, the agent returns a set of plausible webpages, using both the local and global content preferences. We hence represent webpages using both the content and the corresponding instantaneous analytics.

Webpage Content: The webpage text content is represented using Universal Sentence Encoder [3]. We leverage the pre-trained model using Tensorflow Hub[1] which returns a $d_C = 512$ dimensional representation for a given input.

Instantaneous Webpage Analytics: The incorporation of analytics allows the agent to better predict the future content preferences of the users, while

[1] https://tfhub.dev/google/universal-sentence-encoder/2.

also catering to business objectives. We divide the time scale into fixed-sized intervals. Let's consider a set of k analytics KPIs such as number of views and number of exits. While training and subsequent testing, we track the KPIs for all webpages seen until now. Analytics representation is obtained by combining the value for most recent d_A time intervals for each of the k KPIs hence resulting in a $d_A k$-dimensional vector. The final representation for an action is computed by concatenating both the content and analytics representations of the webpage, ending up with a $(d_C + d_A k)$ dimensional vector.

States: Historic Action Sequence. The current state must capture the session-level preference of the users. Hence, we aggregate the representations of all the historically visited webpages in the current session to define the state of the user. **DRS-LaG** uses two LSTM networks to combine the historic content and action analytics.

Defining Rewards. At each timestep, the agent receives a reward from the user, based on the action chosen in the given state. The complete reward for a given state-action pair $r(s, a)$ is a combination of prediction and instantaneous analytics: $r(s, a) = r_P(s, a) + (r_A^1(a) + r_A^2(a) + r_A^3(a)...r_A^k(a))$.

Where $r_P(s, a)$ refers to prediction reward, whether the corresponding webpage was visited by the user in the offline data logs, and r_A^i refers to the instantaneous analytics of the action a with respect to KPI i.

Learning Stage. The training algorithm is discussed here (see Algorithm 1). First, experience replay [4] and target network [5] are used to stabilize the training process. Second, at each timestep, apart from considering the actual action from the data, we also sample N negative actions from the webpage pool P. This is necessary as the offline logs only contain the positive samples for next-page prediction. Moreover, this allows the agent to explore the instantaneous analytics values of webpages, beyond those seen in the current session. Third, the model is trained using the Bellman Equation. Fourth, we skip the replay memory update for the first few webpages in every session, owing to the inadequacy of the initial webpages to capture the context. This detail is removed from Algorithm 1 for simplicity. Finally, since the model is trained on instantaneous analytics values, we update both the webpage pool and analytics values after each episode.

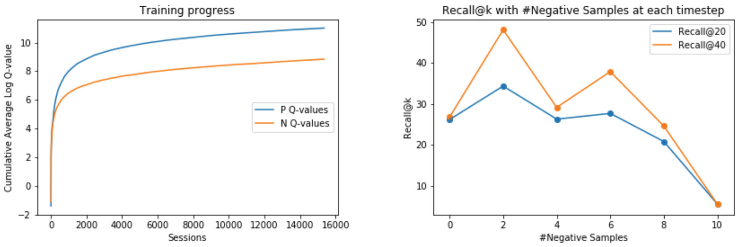
Test Stage - Offline: Given the state, the model is asked to predict the next webpage user will go to. Keeping $\gamma = 0$, the model is trained using Algorithm 1 to incorporate only the immediate reward, as appropriate for next-page prediction. The test data is parsed similar to the training procedure. At every timestep, the recall is observed based on the predictions from a trained model $Q_M(s, a)$ and the actual action from offline logs.

Online: We also evaluate our framework in an online simulated environment. Given the complexity of setting up an online evaluation, following prior work [8], we resort to a framework which effectively simulates the real-time environment with the capability to provide immediate feedback given state and action. We split our data into two and train this simulator on the first half, keeping the

second for training. The simulator architecture is same as in Fig. 1c and is trained to only predict the immediate feedback. The performance of our model in the offline setting attests to the performance of the simulated environment.

3 Experiments

Dataset: The experiments are based on a snapshot of a sports website[2]. The clickstream is gathered using an enterprise analytics tool deployed on the website. The data consists of 37,667 user sessions. We maintain a temporal order in the paths based on timestamps associated with each session. Minimum path length is kept at 3 and maximum as 50. The data contains 1,599 unique urls. The first 33,900 paths are kept for training, next 1,883 paths for validation, and last 1,884 paths for testing.



(a) Training progress of DRS-LaG. **P Q-values**: Q-values of the actual action from the data. **N Q-values**: Average Q-values for the negative actions, sampled uniformly from the action pool.

(b) Effect of varying the number of negative samples for DRS-LaG on Recall@20 and Recall@40 in the offline evaluation task. For this analysis, the models were trained on a 20% data.

Fig. 2. Training progress and the impact of the number of negative samples for DRS-LaG.

Hyperparameters: Content representations are 512 dimensions while the instantaneous analytics are 50-d. The batch size is 16, with learning rate for Adam optimizer as 0.01, number of negative samples as 2, interval size as 5 s and size of replay buffer as 5000 transitions. The weights are transferred to the target network after every 1000 replay iterations. The prediction reward is set to 3 for correct prediction and 0 otherwise, while the analytics reward is fixed to the total change in KPI value over the past 50 intervals. Number of views is considered as the KPI for all experiments. These parameters are tuned on the validation dataset. Once tuned, the models are trained on 'training+validation' data for evaluation on the test data.

Training Progress: Fig. 2a visualizes the training progress of **DRS-LaG**. We track two metrics: (1) **P Q-values**: Q-values of the actual action taken from the data and (2) **N Q-values**: Average Q-values for the negative webpages, sampled

[2] We cannot reveal the name of the website because of privacy constraints.

Table 1. Performance based on the offline logs for next-page prediction task.

Model	Offline	
	Recall@20	Recall@40
Random	2.00	3.18
Majority	27.40	38.75
W-Avg-c	27.16	41.19
LSTM-c	29.52	47.99
DRS-LaG: $r_A = 0$	35.55	50.46
DRS-LaG	**36.34**	**51.36**

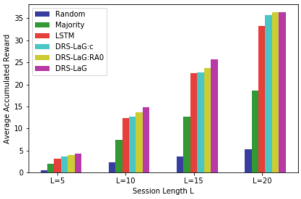


Fig. 3. Performance comparison on our online test based on average reward in a session.

uniformly from the action pool at each timestep. As expected, the two graphs for Cumulative Log values deviate as the training proceeds.

Offline Results: We use two metrics, Recall@20 and Recall@40: what percent of times the correct webpage visited by the user appears in top 20 and 40 webpages returned by the model respectively.

DRS-LaG is trained to predict only the immediate reward at every timestep by keeping $\gamma = 0$. Comparison against baseline models is provided. **Random** ignores the current state and returns a random set of webpages at every timestep. **Majority** returns a list of most-viewed webpages at every timestep. **Majority** can be a really strong baseline in hierarchical website environments. **W-Avg-c** combines only the content representations of the past webpages in the current session using a exponentially-decaying weighted average, to predict the future path. Given the dynamic nature of the websites, instead of predicting a softmax over all the webpages, given the historic webpages and a plausible next webpage, **W-Avg** is trained to predict a score that the plausible webpage will next be visited. At the time of testing, the model returns the webpages with the maximum scores. Similarly, **LSTM-c** uses a Long Short Term Memory recurrent network to capture the historic webpage content. **DRS-LaG:** $r_A = 0$ is trained with both local and global representations similar to **DRS-LaG** but without the analytics reward. Table 1 shows the results. **W-Avg-c** performs similar to the **Majority**, failing to capture the local context or preferences of the users. **LSTM-c** shows improvements, by using a recurrent network to combine historic content visited by the user. With the capability to incorporate both local and global content preferences, **DRS-LaG:** $r_A = 0$ outperforms the baseline methods. Using the analytics reward r_A, **DRS-LaG** shows further enhancement in the performance, attesting to the utility of our approach. We analyze the sensitivity of DRS-LaG in the offline evaluation task towards the number of negative examples sampled at each timestep in Fig. 2b. If the number is too low, the model may end up learning nothing, by learning to predict a high score for every webpage. If the number is too high, the model may consider some in-context webpages as negative,

again countering its own learning mechanism. We empirically identify the value 2 for our experiments (see Fig. 2b).

Online Results: These experiments evaluate the model, if deployed to recommend webpages or adapt future experiences of the users, in a simulated environment. We observe the average rewards in a session to evaluate the models. Session length considered are 5, 10, 15 and 20. To incorporate cumulative future rewards, **DRS-LaG** is trained is $\gamma = 0.95$. **Random**, **Majority** and **LSTM** are implemented in the same manner as before. **DRS-LaG-c** and **DRS-LaG RA0** are trained similar to **DRS-LaG**. However, the former only considers the content (local preference) and the latter keeps $r_A = 0$. The results for our online experiments are plotted in Fig. 3. **DRS-LaG-c** outperforms **LSTM** which is only trained to predict the immediate feedback, attesting to the utility of reinforcement learning. This observation is more evident in longer sessions. **DRS-LaG** $R_A = 0$ and **DRS-LaG** further improve the performance.

4 Conclusion

We presented **DRS-LaG** framework, with the objective of improving user web experiences while simultaneously catering to analytics KPIs. Using Deep RL, our model incorporates both local and global content preferences. We show the proposed method effectively predicts user behavior in a dynamic web environment using both offline and online setups.

References

1. Bellman, R.: Dynamic programming. Science **153**(3731), 34–37 (1966)
2. Burke, R.: Hybrid recommender systems: survey and experiments. User Model. User-Adap. Inter. **12**(4), 331–370 (2002)
3. Cer, D., et al.: Universal sentence encoder. arXiv preprint arXiv:1803.11175 (2018)
4. Lin, L.J.: Reinforcement learning for robots using neural networks. Technical report, Carnegie-Mellon Univ Pittsburgh PA School of Computer Science (1993)
5. Mnih, V., et al.: Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602 (2013)
6. Wu, S., Ren, W., Yu, C., Chen, G., Zhang, D., Zhu, J.: Personal recommendation using deep recurrent neural networks in netease. In: 2016 IEEE 32nd International Conference on Data Engineering (ICDE), pp. 1218–1229. IEEE (2016)
7. Zhang, S., Yao, L., Sun, A., Tay, Y.: Deep learning based recommender system: a survey and new perspectives. ACM Comput. Surv. (CSUR) **52**(1), 5 (2019)
8. Zhao, X., Zhang, L., Ding, Z., Xia, L., Tang, J., Yin, D.: Recommendations with negative feedback via pairwise deep reinforcement learning. In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1040–1048. ACM (2018)