Google RECSIMs

Evaluating Recommender Systems

Hema Puchakayala (Group 12)

THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

Overview

- Project Description
- Outline
- Markov Decision Process (MDP) Formulation
- Code
- Results
- Outlook

Project Description

- Evaluate how effectively recommender systems respond to the evolving interests of users over time
- Recommend over 1100+ of news categories to users
- Synthetic users and documents are generated by sampling features

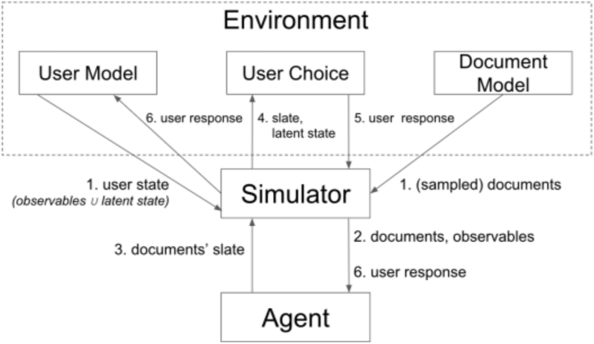


Figure 2: Control flow (single user) in the RecSim architecture.

- **DriftingDocument**
  Holds documents d_i with features: x topic, p popularity score, and q quality score:

$$d_i = (x_i,\ p_i,\ q_i)$$

- **DriftingUserState**
  Stores user's state $u_{j,t}$ at time t: latent preferences $\theta_{j,t}$, satisfaction $\sigma_{j,t}$, fatigue $\phi_{j,t}$, timestep $\tau_{j,t}$:

$$w_{j,t} = (\theta_{j,t},\ \sigma_{j,t},\ \phi_{j,t},\ \tau_{j,t})$$

- **States** (*S*):
  User state at time t: $s_t = (\theta_{j,t}, \sigma_{j,t}, \phi_{j,t})$
- **Actions** (*A*):
  Choice of document(s) to recommend at each step
- **Transition Model** (*P*):
  Defined by the user state update equations:

$$\theta_{j,t+1} = (1-\alpha)\theta_{j,t} + \alpha x_{i,t} + \epsilon_{j,t}$$

$$\sigma_{j,t+1} = (\alpha^2)\sigma_{j,t} + (1-\alpha)^2 p_i + 2\alpha(1-\alpha)q_i + \epsilon_\sigma$$
$$\phi_{j,t+1} = (\alpha^2)\phi_{j,t} + 2\alpha(1-\alpha)p_i + (1-\alpha)^2 q_i + \epsilon_\phi$$

- **Reward Function** (*R*):
  User click or engagement, as modeled by the response equation:

$$\Pr(\text{click}_{j,i,t} = 1) = \sigma\left(\theta_{j,t}^T x_i + \beta_1 q_i + \beta_2 p_i + \gamma_1 \sigma_{j,t} + \gamma_2 \phi_{j,t} + \xi_{j,i,t}\right)$$

- **DriftingResponseModel**
  Maps user/document features into click probabilities:

$$\Pr(\text{click}_{j,i,t} = 1) = \sigma\left(\theta_{j,t}^T x_i + \beta_1 q_i + \beta_2 p_i + \gamma_1 \sigma_{j,t} + \gamma_2 \phi_{j,t} + \xi_{j,i,t}\right)$$

- **DriftingUserModel**
  Advances user state by updating preferences with drift:

$$\theta_{j,t+1} = (1-\alpha)\theta_{j,t} + \alpha x_{i,t} + \epsilon_{j,t}$$

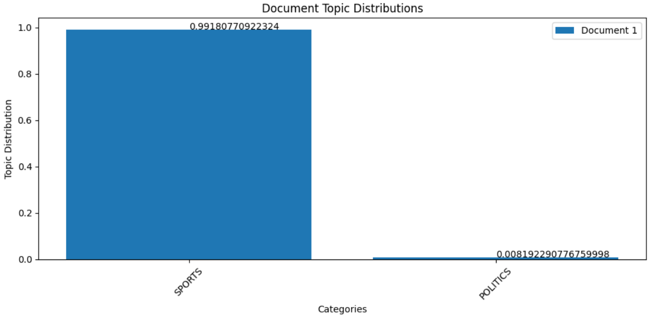
$$\sigma_{j,t+1} = (\alpha^2)\sigma_{j,t} + (1-\alpha)^2 p_i + 2\alpha(1-\alpha)\, q_i + \epsilon_\sigma$$
$$\phi_{j,t+1} = (\alpha^2)\phi_{j,t} + 2\alpha(1-\alpha)\, p_i + (1-\alpha)^2 q_i + \epsilon_\phi$$

```
class DriftingEnvironment:
    """Combines␣all␣components␣into␣a␣full␣environment."""
    def __init__(self, num_users, num_documents, alpha, a, b, beta_1, beta_2, gamma1, gamma2, categori
        self.num_users = num_users
        self.num_documents = num_documents
        self.categories = categories
        self.alpha = alpha
        self.a = a
        self.b = b
        self.beta_1 = beta_1
        self.beta_2 = beta_2
        self.rng = np.random.default_rng(seed)

        self.doc_sampler = DriftingDocumentSampler(categories, alpha=alpha, a=a, b=b, seed=seed)
        self.user_sampler = DriftingUserSampler(categories, alpha=alpha, a=a, b=b, seed=seed)
        self.response_model = DriftingResponseModel(beta1=beta_1, beta2=beta_2, gamma1 = gamma1,gamma2
        self.user_model = DriftingUserModel(alpha=0.01, seed=seed)

    def step(self, user, doc):
        response = self.response_model.simulate_response(user, doc)
        user = self.user_model.update_state(user, doc)
        return int(response), user

    def reset(self):
        users = self.user_sampler.sample_users(self.num_users)
        documents = self.doc_sampler.sample_documents(self.num_documents)
        return users, documents
```

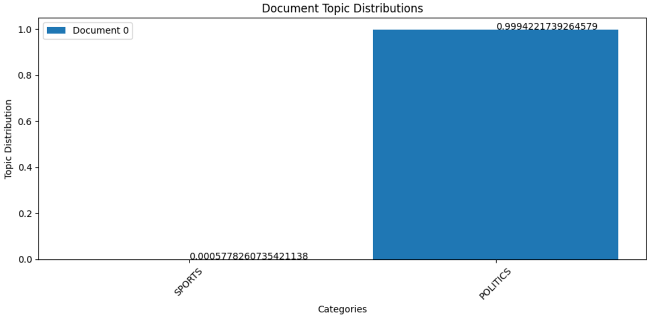
```
for step in range(NUM_ROUNDS):
    step_reward = 0.0
    for user_index, user in enumerate(users):

        click_probs = np.array([env.response_model.score(user, doc) for doc in documents])
        state_key = tuple(np.round(click_probs, 4))

        action = model.esoft(state_key, len(documents))
        selected_document = documents[action]

        reward, updated_user = env.step(user, selected_document)
        tdq_cum_rew += reward
        users[user_index] = updated_user

        scalar_reward = reward if np.isscalar(reward) else np.sum(reward)

        latent_preference_list_tdq.append(updated_user.theta.copy())
        sigma_list_tdq.append(updated_user.sigma)
        phi_list_tdq.append(updated_user.phi)

        click_probs_next = np.array([env.response_model.score(updated_user, doc) for doc in docume
        next_state_key = tuple(np.round(click_probs_next, 4))
        model.update(action, state_key, next_state_key, scalar_reward)
```

- Compared the performance of both a random model and a Q-learning model for recommending documents to users

Sports

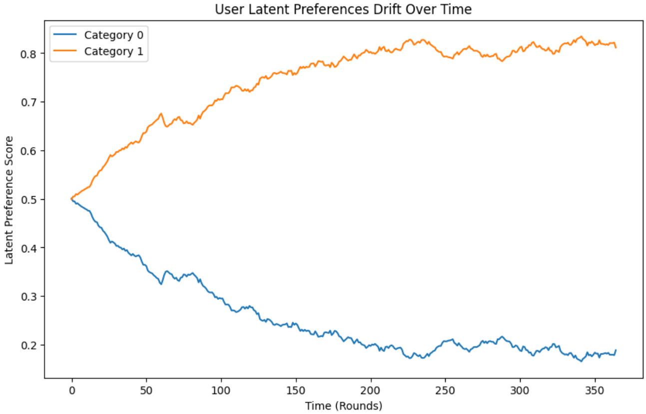

Document Topic Distributions
0.99180770922324
0.008192290776759998
Document 1
Topic Distribution
1.0
0.8
0.6
0.4
0.2
0.0
SPORTS
POLITICS
Categories

Politics

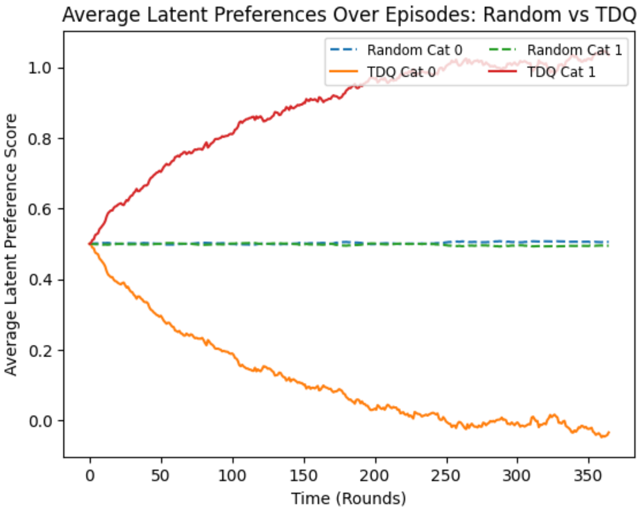


Document Topic Distributions
Document 0
0.9994221739264579
0.0005778260735421138
1.0
0.8
0.6
0.4
0.2
0.0
Topic Distribution
SPORTS
POLITICS
Categories

- Actual Latent Preference Drift



- Latent Preference Drift by the model



Outlook

- The current model is a simulation for studying adaptive recommender systems
- Compare the Random Model and Q-Learning Model
- Scaling up for larger user base and document categories