Recommender systems (RSs) have become an inseparable part of our everyday lives. They help us find our favorite items to purchase, our friends on social networks, and our favorite movies to watch.

The massive volume of information available on the web leads to the problem of *information overload*, which makes it difficult for a decision maker to make right decisions. The realization of this in our everyday lives is when we face a long list of items in an online shopping store; the more items in the list, the tougher it becomes to select among them. Recommender systems (RSs) are software tools and algorithms that have been developed with the idea of helping users find their items of interest, through predicting their preferences or ratings on items. In fact, the idea is to know the users to some extent, i.e., making a *user profile* based on their feedback on items, and to recommending those items that match their profile. Today, RSs are an essential part of most giant companies, like Google, Facebook, Amazon, and Netflix, and employed in a wide range of applications, including entertainment, e-commerce, news, e-learning, and healthcare.

*Recommendation system are widely used in e-commerce that is a part of ebusiness. It helps users locate information or products that they would like to make offers.*The growth of information on World Wide Web make users more difficult to search for relevant information as the amount of product in e-business increases rapidly. Customers suffer from searching for interested products. To avoid this problem, many websites use recommendation system to help customer finding the satisfy products. Recommendation systems are categorized into two major classes: content-based filtering and collaborative filtering [1]. In content-based filetering, the system tries to match the content of product with user profile, both content of product and user profile represented by keywords. Robin van Meteren and Maarten van Someren proposed PRES that use content-based filtering techniques to suggest document that relevance to user profile. The user profile was created by user feedback. In collaborative filtering, the system try to match user pattern with another users that

had the same taste then predict the most user’s interest to items. GroupLens [2] is a collaborative filtering of netnews, by rating articles after read, to suggest customers the articles that related with their interests.

Traditionally, the recommendation problem was considered to be a classification or prediction problem, but it is now widely agreed that formulating it as a sequential decision problem can better reflect the user-system interaction. Therefore, it can be formulated as a Markov decision process (MDP) and be solved by reinforcement learning (RL) algorithms. Unlike traditional recommendation methods, including collaborative filtering and content-based filtering, RL is able to handle the sequential, dynamic user-system interaction and to take into account the long-term user engagement. Although the idea of using RL for recommendation is not new and has been around for about two decades, it was not very practical, mainly because of scalability problems of traditional RL algorithms. However, a new trend has emerged in the field since the introduction of deep reinforcement learning (DRL), which made it possible to apply RL to the recommendation problem with large state and action spaces.

**Reinforcement Learning**

Reinforcement learning is one of powerful machine learning algorithm. Learning from reinforcement is a trial-anderror learning scheme.

Reinforcement learning (RL) is a machine learning field that studies problems and their solutions in which agents, through interaction with their environment, learn to maximize a numerical reward. According to Sutton and Barto, three characteristics distinguish an RL problem: (1) the problem is closed-loop, (2) the learner does not have a tutor to teach it what to do, but it should figure out what to do through trial-and-error, and (3) actions influence not only the short-term results, but also the long-term ones. The most common interface to model an RL problem is the *agent*

*environment* interface. The learner or decision maker is called *agent* and the *environment* is

everything outside the agent. Accordingly, at time step , the agent sees some representations/information about the environment, called *state*, and based on the current state it takes an *action*. On taking this action, it receives a numerical *reward* from the environment and finds itself in a new state.

More formally, the RL problem is typically formulated as a Markov decision process (MDP) in the form of a tuple (*S, A, R, P,*), where *S* is the set of all possible states, *A* is the set of available actions in all states, *R* is the reward

function, *P* is the transition probability, and is the discount factor.

The main elements of an RL system are:

**• Policy***:* policy is usually indicated by and gives the probability of taking action when the agent is in state . Regarding the policy, RL algorithms can be generally divided into *on-policy* and *off-policy* methods. In the former, RL methods aim at evaluating or improving the policy they are using to make decisions. In the latter, they improve or evaluate a policy that is different from the one used to generate the data.

**• Reward signal***:* upon selecting actions, the environment provides a numerical reward to inform the agent how good or bad are the actions selected.

**• Value function***:* the reward signal is merely able to tell what is good immediately, but the value function defines what is good in the long run.

**• Model***:* model provides the opportunity to make inferences about the behavior of the environment. For instance,

the model can predict next state and next reward in a given state and action.

• **State** *S***:** a state ∈ *S* is defined as the user preferences and their past history with the system.

• **Action** *A***:** an action ∈ *A* is to recommend an item to the user at time step .

• **Reward** *R***:** the RL agent receives reward r() ∈ *R* based on the user feedback on the recommendation

provided. Agent can learn to perform an appropiate action by recieving evaluation feedback. The objective is trying to maximize the expected sum of future values for each state. One major component of reinforcement algorithm is On-Policy TD control, called SARSA method. It consists of state-action pair and we can learn from the changing of value state Q(s,a) between stateaction pair to another state-action pair. The value state defined as:

Q Q +αQQ].......................1

where st is the state of agent at time t, at is the action of agent at time t, rt+1 is reward of state s that action a, α is learning rate (0≤α<1) and γ is discount rate (0≤γ<1).

**The Sarsa control algorithm is given as** :

Initialize Q(s, a)

Repeat

Initial s

Choose a from s using policy

Repeat

Take action a, observe r′, s′

Choose a′ from s′ using policy

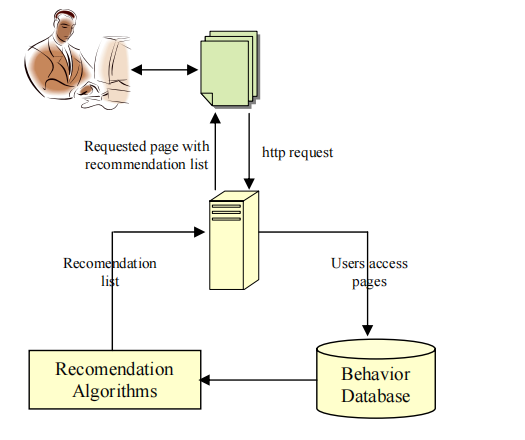
Update Q(s, a)

s′←s; a′←a;

until s is terminal

First, we initialize value of Q(s,a) and choose the initial state s. Second, we select action *a* at state *s* using policy. The policy can be greedy or ε-greedy policy. Next, repeat take action *a*, observe the reward *r* and the next state *s′*, choose *a′* from *s′* using policy for compute the value of future next state and then update the Q(s,a) of current state and change *s*←*s′*, *a*←*a′*.

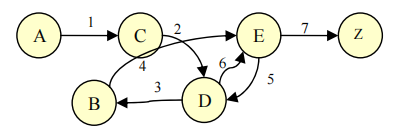
**System Architecture**



1. Behavior Database. The system keeps behavior of customers into two groups. First, the global behavior collected behavior of all customers, we can know another customers direction. Second, the local behavior collected behavior of each customer, we can know which product bought by them.

1. Recomendation Algorithms. It’s a part for learning by reinforcement learning and send recommendation list to users. When the customers login to system and open a page. The pages can be viewed as states *s* of the system and links in a page can be viewed as action *a* of state. The system puts the state diagram (page-to-page) as shown in the figure below and the changing value between states into Q-matrix.

**An example of a state diagram of user behavior.**

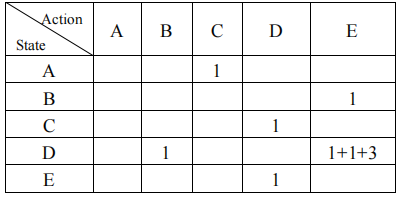


The system has two type of Q-matrix. First, global Q-matrix keeps the changeablility of whole system. The global Q-matrix can tell you the trend of customers or most popular products. Second, local Qmatrix keeps the changeablility of each customer. The local Q-matrix likes customer profile that record customer browsing behavior. To update of both Q-matrix system will get reward from click on the products and make offered, we call customer feedbacks. Customer feedbacks are important part of recommendation system. The Customer feedbacks may be explicit and implicit. Customers can send the explicit feedback by rating the products. The system will update Q-Matrix of that produce page. For example, the customer may give rate 3 out of 5. Although explicit rating is accurate, there are few customers who gave rating for

produces after they used. The system needs another feedback from customers. Implicit feedback is percieved by keeping customer’s behavior. It has two type of implicit feedback. First, when customer changes state that means click on product page. Second, when customer changes state to final state that means product added into shopping card and bought its.

In Table.1 shown you a local Q-matrix of customer who has the changing state like in Figure 3. The customer logins into website and click on product A. Then he/she has sequence of the changing state like A→C, C→D, D→B, B→E, E→D, D→E and bought product E. When customer click on each produce page, system will update both Q-matrix by plus 1 and when customer bought the product plus 3 to that product.

**Table 1. Q-matrix of user**



To predict next state or next product that customers may prefer to offer, the system will rank products. The ranking system separated into 2 parts. First part is the ranking of whole system, global ranking, system uses data from global Q-matrix to choose the next state by using ε-greedy policy.

If you maintain estimates of the action values, then at any time there is at least one action whose estimated value is greatest. We call this a greedy action. If you select a

greedy action, we say that you are exploiting your current knowledge of the values of the actions. If instead you select one of the nongreedy actions, then we say you are

exploring because this enables you to improve your estimate of the nongreedy action's value. The ε is a small probability to select an action at random. The advantage of ε-greedy policy over greedy policy are the ε-greedy policy continue to explore and improve their chances of recognizing the product that customer may make offered. The greedy policy will choose only the state that has maximum values. If the ε=0.2, the method explores more than, and usually finds the optimal action earlier, but never selects it more than 81% of the time. The ε=0.1 method improves more slowly. For example if the system have 5 ranks and ε=0.1. In each position of the system will have chance to choose the highest Q-value equal to 90% and choose another equal to 10%. To do like this the system gives a chance for new products or product that have few clicks on but may be match with your interest.

Second part is the ranking of each customer, local ranking, the system considered from local Q-matrix. To choose the next state, the system uses inverse ε- greedy policy. The states that customer ever visit, system will decrease important and gives a chance to explore other products.

After that system finds Qtotal by using eq 2.

= .............................................................................2

where w is the weight of Qglobal and *w*∈(0,1].

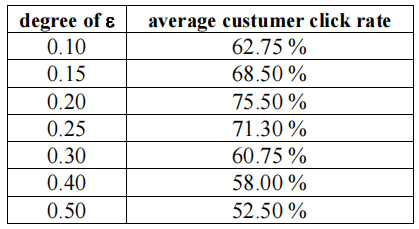
Final, system will rank produces by using Qtotal and recoment to customers.

**Experiments**

The objective of experiment was to find relationship between ε and user click rate, the click rate is a measure of how many of the presented products in recomendation list that customer clicks on. Tab. 2 shows the average customer clicks rate where

*w*=0.8.

**Table2. The average customer click rate.**



We found that ε = 0.2 can preserve balancing exploration and exploitation power. If ε less than 0.2 the system will exploit to the trend of system and gave a small chance to explore new product. So in the early time of user login the clicks rate was high and after that the system showed that the clicks rate dropped for the same previous product. Although *Qlocal* try to promote the new product, it is less effective than *Qglobal*. If ε more than 0.2 the system will too much explore the products. The system may include the product that does not match the customer.

**The current challenges or ethical issues**.

**RL Challenges.** There are some possible challenges when applying RL to any problem. A challenge well-known as *Deadly Triad* states that there is a *hazard* of instability and divergence when combining three elements in RL: function approximation, bootstrapping, and off-policy training.

Another challenge in RL is *sample inefficiency*, specifically in model-free RL algorithms. Current model-free RL algorithms need a considerable amount of agent-environment interaction in order to learn useful states. Moreover, since deep reinforcement learning is based on deep learning, it consequently inherits the famous feature of neural networks, i.e., being *black-box*. It is not obvious how weights and activations are changed, which makes them uninterpretable. The classical problem of exploration vs exploitation is still a challenge in RL and effective exploration is an open research problem

One of the challenges is the need to balance exploration and exploitation in the recommendation system. Finally, the problem of reward formulation in RL is a challenge and designing a good reward function is not very clear or straightforward.

**Conclusions and Future Work .**

In this paper a deep insight was derived for a general framework for recommendation system based on reinforcement learning. The system can learn direct from customer’s behavior. The learning process is using SARSA method and ε-greedy policy. The system consists of two parts, global model and local model. One important aspect of learning method is balance of exploration and exploitation. The ε-greedy policy can give a chance to explore new produces, but the same time its exploit to the trend of system. We showed this in a simple experiment about the effect of ε value and user click rate. The result of this experiment can explain that if you explore too much it has a chance to comment the uninterested product to users, if you exploit too much it has a chance to stick with other opinion and never see the other products. In the real world, it requires more space to keep global state and local state of all users. We plan to continue reducing the space by using another data structure. We also study in the effect of *w* to find the optimal *w* for *Qtotal*.

To further improve the current methods and techniques, it is essential to address the challenges of maintaining global and local state information for all users. This could involve optimizing the weight for the Qtotal calculation and exploring data structure improvements to reduce space requirements. Additionally, further research could focus on finding the optimal ε value for balancing exploration and exploitation in the recommendation system.

In conclusion, this paper presents a comprehensive overview of the recommendation system using reinforcement learning, its architecture, and the results of experiments, providing valuable insights for the development of advanced recommendation systems in e-commerce and e-business. The application of reinforcement learning in this context offers a promising approach to personalized recommendations and user satisfaction. However, addressing the challenges and continuing to improve the methods and techniques will be crucial for the future development of recommendation systems.