Designing a Situation-aware Movie Recommender System for Smart Devices

Mhd Irvan, Na Chang, Takao Terano
Dept. of Computational Intelligence and Systems Science
Tokyo Institute of Technology
Yokohama, Japan
irvan@trn.dis.titech.ac.jp, changna@trn.dis.titech.ac.jp, terano@dis.titech.ac.jp

Abstract—**With the growing number of people using SmartTVs and Smartphones, designing a recommender system for on-demand streaming media, such as movie streaming, has been an attractive, yet challenging work. There are many factors that influence people to enjoy a movie. Smart devices provide many kinds of data from its sensors that can help us deduce, for example, whether it is the time, the day, the location, or the combination of those that makes a great experience in watching a particular movie. However, designing the algorithm to consider all these factors can lead into a very complicated decision tree. To address this issue, we propose a simple evolutionary computational approach that can be used to search through those huge numbers of possible combinations of solutions, and find the relevant factors when recommending a movie to particular type of users.**

Keywords-Recommender System; Classifier System; Genetic Algorithm; SmartTV; Smartphone

I. INTRODUCTION

Smart Devices, such as SmartTVs and Smartphones, provide the integration of various web services into televisions and mobile phones. One of such services is on-demand streaming media. On-demand streaming allows users to choose shows or movies they would like to watch. However, there are countless choices available from the streaming providers. This may lead users to confusion, as they might not know what would be interesting to watch [2].

Recommender systems address this problem. Rather than waiting for users to choose a movie, the system recommends movies that it thinks they will like. The recommendations are usually generated through learning from users' past behavior [3], or from other similar users' interest [5].

Many current recommender algorithms rely on users' feedback, such as rating, or profile. For example, when a user liked a movie, the system will search for other users who liked the same movie, and then, recommend other movies that those users also liked. While this might work very well for shopping websites, it might not be suitable for media streaming.

When a user enjoys a movie, there are many factors that affect his/her enjoyment at that moment. For example, a user who usually enjoys action movies on weekend might prefer watching drama movies on other days at night to relax after getting tired from work. A user might really like science fictions, but s/he only enjoys watching them from a large screen TV at home, and never on smartphones due to the small screen.

In other words, even if a user liked a movie, s/he might not be going to enjoy the same movie, had s/he watched it under different circumstances. Locations, devices, days, times, and other factors contribute to whether she will enjoy a movie or not. SmartTVs and smartphones can provide all these details, and it is only natural to use the information as basis for recommender systems. However, designing a recommender system to consider all these factors using typical recommender algorithms will end with a very complicated decision-making process. This paper offers a simple algorithm to address this issue using Learning Classifier System (LCS) [7], implementing genetic algorithm (GA) [1] and reinforcement learning (RL) [4].

LCS maintains a population of classifiers that predicts the best action given its input. The input we use is information that is available from smart devices, such as sensory data, geographical and device information, as well as date and time. GA is used to search the possible solution space to figure out which part of the inputs, or what kind of input combination affects the viewing experience. Solutions proposed by the GA are evaluated by RL, giving feedback whether they are accurate or not. During training, LCS repeat this process over and over again until it has a good population set with high average accuracy.

This paper starts with the introduction to recommender systems in Section I and reviews some of the literatures related to this field in Section II. We define our proposed method using LCS for recommending items in Section III. Finally, we put our conclusion and the discussion about future work in Section IV.

II. LITERATURE REVIEW

Mukherjee [3] proposed a movie recommender system using voting method. Their system tracks users' preference, such as favorite actors, actress, genres, etc. Each attribute of the preferences is given a weight value, which reflects the relative importance of those attributes. The voting system calculates these weights according Bayesian learning scheme and returns a ranking of alternatives when the user asked for a recommendation.

Salter [2] combined two popular recommender algorithms, Collaborative Filtering (CF) and Content-Based Filtering (CBF), into one system. The CBF was used to address the cold-start problem with CF not being able to make recommendation for new items.

Symeonidis [5] developed a recommender system with explanations. Theirs system gives the ability to a user to check the reasoning behind a recommendation. This allows users to accurately predict their true opinion of an item.

Those systems managed to make good predictions about movies that users would like. However, those systems were designed before the smart devices and streaming media went mainstream. The way people watch movie has changed, some people like to watch at home, some prefer to watch on mobile devices while commuting. They did not consider these possible factors and other information that can be provided by smart devices. We tackle this issue with our proposed method.

III. PROPOSED METHOD

A. *Learning Classifier System*

LCS [1] is a machine learning paradigm, in which an intelligent agent is interacting with an environment. LCS keeps a collection of classifiers, referred as population set. Each classifier is essentially a rule of condition-action set. The classifiers have a parameter that predicts the reward that the agent will receive, should it choose the action proposed by the relevant classifiers. LCS agent learns to perform the best action based on the condition. Whenever the agent performs an action, it receives feedback from the environment to inform the quality of the action.

There are many models of LCS available today, such as ZCS [6] and XCS [7]. Different models have different criteria as what is the "best" action. ZCS model trains the system to chase high rewards. Over time, the population set evolves into a set of classifiers that predicts high reward only. The downside of this model is, although it gets huge reward when it does predict correctly, many of the classifiers often predict incorrectly. This led into inconsistent accuracy [6].

XCS model tackles the issues related to ZCS. Each classifier in XCS maintains an additional parameter, referred as accuracy parameter. This parameter job is to keep track of how often its classifier made inaccurate predictions. XCS agents prefer actions proposed by classifiers with high accuracy value, although they may predict low reward. Thus, XCS is more suitable to problems where consistent accuracy is important [7]. For this reason, we choose XCS model as the basis of our proposed method.

Recommendation using LCS means that the systems can, unlike CF and CBF method [2], consider more factors in deciding which item to recommend. CF concerns only about similar users, while CBF concerns about similar items. While they are good for users of web shopping sites, they might not be suitable for users of media streaming services, where users do not simply like an item, but mood factors affect in a sense for example they might have different preferences in morning and night time. LCS can be used to consider these factors when making recommendations. In addition to recommender systems, our proposed method has also been applied to simulate security patrol [8].

B. *Generating Initial Classifiers*

The condition part of the classifiers is a string of input reflecting the situation that the agent encounters. In our recommender system, the input string consists of information that can be provided by smart devices: Day, time, user's age, gender, type of device, location, movie's release date, movie genres, movie stars, movie ID (Figure 1).

From the training set, when a user "Like"d a movie, the system generates several classifiers that represent the situation. It takes into consideration the information mentioned above.

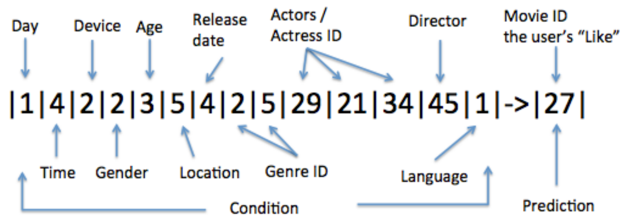


Figure 1. Classifier Representation

The numbers shown in Figure 1 are interpreted according to the following schema:

- Day: [0] = Unknown; [1] = Sunday; [2] = Monday; [3] = Tuesday; [4] = Wednesday; [5] = Thursday; [6] = Friday; [7] = Saturday.
- Time: [0] = Unknown; [1] = 6 AM ~ 8:59 AM; [2] = 9 AM ~ 11:59 AM; [3] = 12 AM ~ 2:59 PM; [4] = 3 PM ~ 6:59 PM; [5] = 7 PM ~ 8:59 PM; [6] = 9 PM ~ 11:59 PM; [7] = 12 PM ~ 2:59 AM; [8] = 3 AM ~ 5:59 AM.
- Type of device: [0] = Unknown; [1] = TV; [2] = Tablet; [3]=Phone.
- Gender: [0] = Unknown; [1] = Male; [2] Female.
- Age: [0] = Unknown; [1] = Below 18; [2] = 18 ~ 29; [3] = 30 ~ 39; [4] = 40 ~ 49; [5] = 50 ~ 59; [6] = 60~69; [7]=70~79; [8]=80 and above.
- Location ID (e.g., [5] = Tokyo).
- Release date: [0] = Unknown; [1] = Before 1970; [2] = 1970 ~ 1979; [3] = 1980 ~ 1989; [4] = 1990 = 1999; [5] = 2000 ~ 2010; [5] = 2010 and after.
- Genres ID (e.g., [2] = Drama; [5] = Romance).
- Actor/Actress ID (e.g., [29] = Leonardo DiCaprio; [21] = Kate Winslet, [34] = Billy Zane).
- Director ID (e.g., [45] James Cameron).
- Language ID (e.g., [1] = English).
- Movie ID (e.g., [27] = Titanic).

Suppose a user likes Titanic (a fact). The system considers the situation when it happened. On what day? What time? What kind of user? What kind of movie? Who are the actors? The classifier shown in Figure 1 can be read this way: "A female teenagers who live in Tokyo who likes 1990s movies AND likes drama and romantic movies AND likes movies starring Leonardo DiCaprio, Kate Winslet, and

Billy Zane AND likes movies directed by James Cameron AND likes English movie WILL enjoy watching Titanic ON a tablet AT night ON Sunday".

Since movies are usually related to multiple genres and have many actors/actresses involved, for each fact the system generates multiple classifiers picking (in our example) two random genres and three random actors/actresses related to the movie. This means that during learning the system will look for which combination of genres and actors/actresses are relevant to the user's profile.

Additionally, for each fact the system also generates classifiers that have "Wildcard" symbols along the input string. This means that during learning the system will look for which parts of the input that are not relevant (the noise) to the user's profile. Consider a more generalized classifier illustrated in Figure 2.

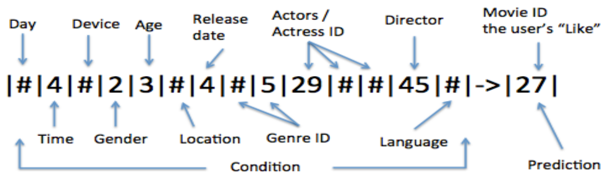


Figure 2. A Generalized Classifier

The classifier illustrated in Figure 2 shows some wildcard symbols ('#' symbols) along the input string. This classifier thinks that the "Day", "Device", "Location", one "Genre", two "Actors/Actresses", and "Language" parts are not relevant. This classifier can be read: "Regardless of the Day, Device and Location, a female teenager who likes 1990s romantic movies AND likes movies starring Leonardo DiCaprio AND likes movies produced by "James Cameron", WILL enjoy watching Titanic at Night".

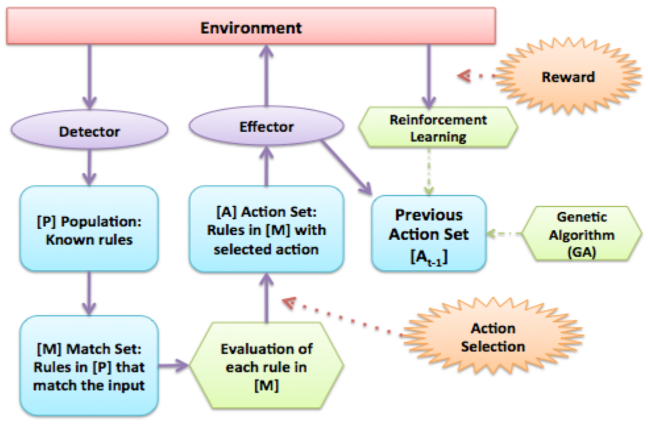
The more wildcard symbols a classifier has, the more generalized it is. After generating all the classifiers from the initial set of facts, the system will test those classifiers against users in the testing set. During each learning loop, the system will evolve classifiers that make accurate prediction and delete inaccurate classifiers through evolutionary process. After the learning process is finished, the population set should consist the generalized classifiers (but not too generalized) with high accuracy.

The input length for LCS is flexible. If, for example, more profile data are available from the smart devices, it is possible to add those details into the input. If necessary, more metadata about the movies (such as more genres, studio name, music composer, or other metadata) can be added too. The longer the classifier, the more details are taken into consideration.

Obviously, when more details are necessary to be considered, using general decision process, such as decision tree, the decision-making process will end up with a very complicated procedure. LCS simplifies this process by representing the details as strings of possible solution and let the evolutionary process search for the good ones.

C. *Learning*

After the initial population set is generated, the learning phase begins (Figure 3).



Environment
Reward
Detector
Effector
Reinforcement Learning
[P] Population: Known rules
[A] Action Set: Rules in [M] with selected action
Previous Action Set [A_{t-1}]
Genetic Algorithm (GA)
Action Selection
[M] Match Set: Rules in [P] that match the input
Evaluation of each rule in [M]

Figure 3. Learning Process

LCS starts by sensing the current environment situation. Suppose in the training set, LCS encounters a user with the situation shown in Figure 4, and LCS has a Population Set [P] of eight Classifiers C shown in Figure 5.

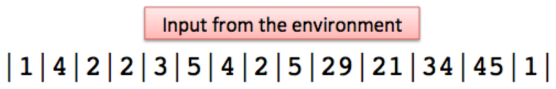


Figure 4. Input sensed from the environment situation

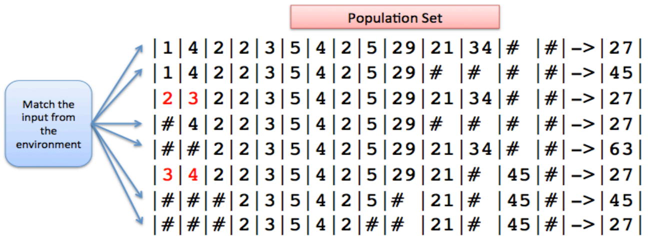


Figure 5. Population Set

From Figure 5, we can see that six classifiers match the input from environment. The first two digits of the other two classifiers do not match the input. LCS will select the six matched classifiers, and put them in a collection called Match Set [M] (Figure 6).

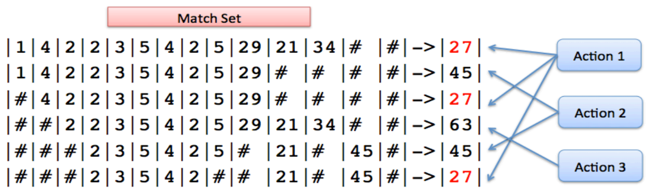


Figure 6. Match Set collected from the Population Set

Figure 6 shows that the six classifiers inside the Match Set predicted three different actions (three movies). For each action, LCS calculate the prediction array *P(A)* using the value of reward prediction (*C.p*) and fitness (*C.f*) of classifier C ∈ [P] (1).

$$P(A) = \frac{\sum_{C.a=a \wedge C \in [M]} C.p \times C.f}{\sum_{C.a=a \wedge C \in [M]} C.f} \tag{1}$$

Essentially, P(A) reflects the average of all reward prediction of classifiers in [M] that advocate action *a*. The algorithm chooses the action that maximizes P(A) (2).

$$A_{\max} = \arg\ \max_A P(A) \tag{2}$$

Suppose in our example above, Action 1 is calculated as the best action. Then, LCS puts the three classifiers in [M] that proposes Action 1 into an Action Set [A].

After LCS applies the action, it receives feedback from the environment whether if its prediction is correct or not. Through RL method [4], the classifiers in [A] are credited with reward *r* as a result of the action performed. Then, the prediction error of each classifier in [A] is updated (3).

$$\varepsilon \leftarrow \varepsilon + \beta\left(\left|r - p\right| - \varepsilon\right) \tag{3}$$

where β is the learning rate.

Next, reward prediction of each classifier in [A] is updated (4).

$$p \leftarrow p + \beta\left(r - p\right) \tag{4}$$

Unlike generic LCS, XCS classifiers maintain accuracy parameter that tracks the accuracy of the classifiers\ throughout the learning process. Each rule's relative accuracy is determined by dividing its accuracy the total of the accuracies in [A] (5).

$$\kappa' = \frac{\kappa}{\sum_{C \in [A]} C.\kappa} \tag{5}$$

Finally, fitness *f* is updated with respect to the relative accuracy (6).

$$f \leftarrow f + \beta\left(\kappa' - f\right) \tag{6}$$

The dataset is divided into training set and testing set. The training set is used to train the system to generate a population of classifiers that predict a recommendation for existing items. The testing set acts as new items encountered by the system. This is used to evaluate how well the system is able make predictions for unknown items.

D. Evolution

GA [1] is used to evolve the rules in [A]. GA is triggered when the average time period for classifiers within [A] since the last occurrence of GA is greater than GA's frequency parameter. The GA starts by selecting two "parent" classifiers from [A] with roulette wheel selection (7).

$$p_i = \frac{f}{\sum_{C \in [A]} C.f} \tag{7}$$

Once two parents have been selected, new offspring classifiers are generated by *crossover* and *mutation.* Crossover operation selects a point on the parent classifier strings. All digits beyond that point of one parent classifier are swapped with the digits of another classifier. Finally, the digits of the resulting strings from the crossover are mutated into different acceptable values. This means that two new offspring classifiers are generated maintaining some traits from their parents. The offspring classifiers are inserted into the population set, replacing classifiers with low fitness value. Then, the learning process is repeated again until the population set evolves into a collection of classifiers with relatively high accuracy values.

IV. Conclusions and Future Work

This short paper shows preliminary work on how to apply LCS to recommender system. It shows that LCS can be used to train for, not only predicting a movie a user will enjoy, but also finding the reason why s/he might enjoy it. This is useful to recommend a movie to another user that has similar profile.

LCS is flexible, in a sense that, if the system designer decides to change the input types, the learning algorithm stays the same. S/he may also add more details to the input string if s/he manages to gather more detailed data from the sensors of smart devices.

Ongoing work includes testing with real data, as well as cross-validating the result. Future explorations will include performance analysis, such as training time, as well as comparison to other solutions, such as Naïve Bayes Classifier and k-Nearest Neighbors.

References

[1] J. H. Holland, "Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence," Book ISBN 0262581116, A Bradford Book, 1992.

[2] J. Salter, "CinemaScreen recommender agent: combining collaborative and content-based filtering," Intelligent Systems, 2006, pp. 35-41.

[3] R. Mukherjee, G. Jonsdottir, and S. Sen, "MOVIES2GO: An Online Voting Based Movie Recommender System," Proceedings of the fifth international conference on Autonomous agents, 2001, pp. 114-115.

[4] R. S. Sutton, "Reinforcement Learning: An Introduction," Robotica, vol. 17, Issue 2, 1999, pp. 229-235.

[5] P. Symeonidis, and A. Nanopoulos, "MoviExplain: A Recommender System with Explanations," Proceedings of the third ACM conference on Recommender systems, 2009, pp. 317~320.

[6] S. W. Wilson, "ZCS: A Zeroth Level Classifier System," Evolutionary Computation, vol. 2, Issue 1, 1994, pp. 1-18.

[7] S. W. Wilson, "State of XCS classifier system research," Lecture Notes in Artificial Intelligence (LNAI-1813), 2000, pp. 63-81.

[8] M. Irvan, T. Yamada, T. Terano, "Multi-Agent Learning Approcah to Dynamic Security Patrol Routing," Proceedings of SICE Annual Conference, 2011, pp. 875-880.