Int. j. inf. tecnol.
https://doi.org/10.1007/s41870-017-0079-7

ORIGINAL RESEARCH

Improved decision making in multiagent system for diagnostic application using cooperative learning algorithms

Deepak A. Vidhate[1] · **Parag Kulkarni**[2]

Received: 17 May 2017 / Accepted: 22 December 2017

Abstract Cooperative nature in multiagent system inculcates more understanding and data by sharing the resources. So cooperation in a multiagent system gives higher efficiency and faster learning compared to that of single learning. However, there are some challenges in front of learning in a cooperative manner in the multiagent system that needs to pay attention. Making effective cooperative decisions that correctly and efficiently solve interacting problems requires agents to closely cooperate their actions during problem-solving. So various issues related with cooperative machine learning are implemented. Reinforcement learning is mainly implemented with game theory and robot applications. Paper gives the new approach for reinforcement learning methods applied to the diagnostic application. The novelty of the approach lies in the amalgamation two methods i.e. weighted strategy sharing with expertness parameter that enhances the learning performance. Weighted strategy method is implemented with Sarsa (λ), Q(λ) and Sarsa learning for cooperation between the agents that was not implemented previously. Cooperative learning model with individual and cooperative learning is given in this paper. Weighted Strategy Sharing algorithms calculate the weight of each Q table based upon expertness value. Variation of WSS method with Q-learning and Sarsa learning is implemented in this paper. The paper shows implementation results and performance comparison of Weighted Strategy Sharing with Q-learning, Q(λ), Sarsa learning and Sarsa(λ) algorithms.

Keywords Cooperative learning · Q learning · Reinforcement learning · Sarsa learning · Weighted strategy sharing

1 Introduction

In the application of retail market that has a number of shops all over state retailing many items or products to a large number of consumers. Each operation details i.e. customer ID, date items bought with the amount, the sum of money spent are noted down at the sale windows. It produces the huge bulk of data every day. The retailer needs to forecast who likely consumers for a particular item are. A simple algorithm is not sufficient for this prediction. There is need to analyze the accumulated data to convert into useful information that can be further useful for the item forecast. It is not known in advance which consumers are probable to buy this product, else another product [1]. It is understood that there is a process that gives details the data that is observed. But details of the underlying process are completely unknown. In the application of consumer behavior, it is not totally arbitrary [2].

Consumers are not purchasing the item arbitrary. When consumer purchases cold drinks, they may purchase chips; or consumer purchases ice cream in summer and hot tea in winter. There are fixed models in the data. It might not be possible to distinguish the entire procedure, but still, a *good and useful estimation* can be created. That estimate may not make clear everything, but may still be able to focus on

✉ Deepak A. Vidhate
dvidhate@yahoo.com

Parag Kulkarni
parag.india@gmail.com

[1] Department of Computer Engineering, College of Engineering, Shivajinagar, Pune, India

[2] iKnowlation Research Labs Pvt. Ltd, Shivajinagar, Pune, India

Published online: 29 December 2017

some part of the data [3]. Though identifying the total procedure may not be possible, but still, model or regularities can be identified. Such models may help to identify with the process and make a forecast. Forecasting is useful keeping in view that the near upcoming will not be much different from the history and future forecast can also be likely to be right [4].

Many real-world applications are involved over one entity for an increase an outcome. In a situation of retail stores in which store A trade clothes, store B trade jewelry, store C trade footwear. In order to develop a single system to intelligent (certain aspects of) the marketing procedure, an internal of all shops A, B, C, and D can be estimated. The only feasible answer is to permit a mixture of shops to build their individual strategies that precisely characterize their objectives and benefits [5]. They must then be combined into the system with the aid of some of the techniques. The objective of each shop is to increase the revenue by maximizing sale i.e. yield maximization. Diverse factors are to be thought in this: the dependency of items, special discount policy, dynamic nature of seasons, concession, market situation etc. Different shops need to coordinate with one another for an increase in profit at various conditions. Numerous autonomous tasks that can be handled by individual agents could benefit from cooperative nature of agents [5, 6].

The novelty of the approach is clearly understood by three major contributions made by the paper: first, reinforcement learning methods are applied to the diagnostic application. In the literature, it is found that reinforcement learning is mainly implemented with game theory and robot applications. Secondly, it combines two methods i.e. weighted strategy sharing with expertness parameter to enhance the performance. Third, weighted strategy method is implemented with Sarsa (λ), Q(λ) and Sarsa learning for cooperation between the agents that was not implemented previously.

The paper is ordered: Sect. 2 provides a concept about cooperative multiagent agent learning, Sect. 3 describes various Weighted Strategy Sharing methods using Q & Sarsa Learning. Section 4 gives experimental setup and Sect. 5 put up the result comparisons of all four algorithms i.e. Weighted Strategy Sharing using Q Learning, Weighted Strategy Sharing using Sarsa Learning, Weighted Strategy Sharing using Q(λ) Learning and Weighted Strategy Sharing using Sarsa(λ) Learning. Final concluding remark and future scope are mentioned in conclusion.

2 Cooperative learning algorithms

Several multiagent learning systems are designed to pace up learning and/or to increase precision. Learning in MASs can be seen from different points of view. In the simplest form, each agent just learns individually, without any attention to others and without any cooperation [7]. But, in a multiagent system, an agent's knowledge may have a positive effect on other agents' learning and knowledge acquisition. Therefore, cooperative learning may result in higher efficiency [8].

For different people or organizations with different goals and information, an interaction can be done using a multiagent system. These agents may cooperate to achieve the common goal by sharing their expertise/knowledge. Such teamwork would lead to improved performance. Cooperative nature of multiagent system inculcates more understanding and information by sharing the resources. So teamwork in a multiagent system gives higher efficiency and faster learning compared to that of single learning. Hence working in a cooperative team has significant advantages [8, 9]. Making effective cooperative decisions that correctly and efficiently solve interacting problems requires agents to closely cooperate their actions during problem-solving. So various issues related with cooperative machine learning are studied and implemented.

Agents in a multi-agent scheme are trained from all of the agents in strategy-sharing algorithms. Q learning algorithm of Reinforcement Learning is responsible for independent agent learning [10]. Q tables of other agents are compiled together by an agent to calculate their new policy as the average of Q tables. The agents do not contain the capability to explore expertise agents and information of all of the agents is uniformly utilized. The only average of the Q tables is not useful if they contain dissimilar ability with expertise. After every cooperation step, The Q-tables of the agents turns out to be equivalent. Agents' flexibility to changing situation is decreased due to this [11]. A strategy-sharing method depends on expertise detection is proposed to overcome these limitations. In this, the agents allocate some weight to the other agents' Q-tables is used [11, 12].

3 Weighted strategy sharing methods

In previous work on multiagent learning, coordination between agents is one way among different agents. All of the agents can discover somewhat from one another; even from the non-expertise agents also in the real world applications. Weighted strategy sharing for cooperative learning is implemented. Each agent allocates a weight to the information of another agent so as to use it depending on their expertise [13].

3.1 Weighted strategy sharing using Q learning

It is understood that elements of a collection of n uniform agents are training in some situation in WSS method. Agents actions do not modify the others learning situation. Two modes of learning are used i.e.: Independent Learning and Cooperative Learning Mode. Initially, all agents are in the Independent Learning form. Agent i trains t_i learning. Training experiment begins from an arbitrary state and stops when the agent arrives at destination state. All agents stop the Individual Learning mode at the time when a particular amount of independent experiments are performed (that is referred as coordination time) [14]. Then an agent switches to Cooperative Learning form. Each learner allocates some weight to another agent as per their expertise values in the Cooperative Learning form. Then, it calculates a weighted mean of the other agents' Q tables. Resultant Q table is found out that is used as its new Q table [15, 16].

3.1.1 Independent learning based on Q-learning

The agent obtains reinforcement after completion of every action. The Q-table (state-action), that calculates the long-term discounted reinforcement for every state/action couple to determine the trained strategy of the agent [14–16]. In Q-learning, action selection by an agent with the probability P given is by:

$$P(a_i|x) = \frac{e^{Q(x,a_i)/t}}{\sum e^{Q(x,a_k)/t}}, \tag{1}$$

where t regulate the arbitrariness of the choice. The agent carries out the action obtained an instantaneous reward r, shift to the subsequent state y and modify Q as:

$$Q_i^{new}(x_i,\ a_i) := (1-\beta_i)Q_i^{Old}(x_i,\ a_i) + \beta_i(r_i + \gamma_i V(y_i)) \tag{2}$$

where β is the learning rate, ($\gamma < 0\ \gamma < 1$) is a discount parameter. Q is enhanced steadily and the agent learns when it explores the state space.

3.1.2 Measuring the expertness values

Expertise is defined as the "personification of knowledge and skills within persons". In human societies, it is observed that a learner assesses the others' understanding with respect to their expertise. Each learner attempts to the best assessment technique to find out how much the others' understanding is trustworthy. In this WSS Method also, the weight of each agent's understanding is carefully calculated in order that the team training competence is increased. This principle assigns extra weight to such agents who have acquired more rewards and fewer punishments [17]. This is represented as a total of the reward signals

$$e_i^{Nrm} = \sum_{i=1}^{n} r_i(t) \tag{3}$$

where $r_i(t)$ is the amount of reinforcement signal that environment gives to agent *i* in step *t*. It is referred as Normal expertness as it is the just algebraic sum of rewards [18].

3.1.3 Weight-assigning mechanism

Q-tables of more expert agents is used by the learner to reduce the amount of coordination necessary to swap Q-tables. Hence, fractional weights of the inexpert agents are treated as zero. Learner, i assign the weight to the understanding of agent j as:

$$W_{ij} = \begin{cases} 1-\alpha_i & \text{if } i = j \\ \dfrac{e_j - e_i}{\sum e_k - e_i} & \text{if } e_j < e_i \\ 0 & \text{otherwise} \end{cases}, \tag{4}$$

where $0 < \alpha < 1$ is the impressibility parameter to prove agent i depend on another agents information. e_i and e_j are the expertise value of agents i and j, and n is the total number of the agents. Weights assigned by each agent to other agent's information are shown in Fig. 1. w12 is the weight assign by agent 1 to the information of agent 2 and w21 is the weight assign by agent 2 to the information of agent 1 [18, 19].

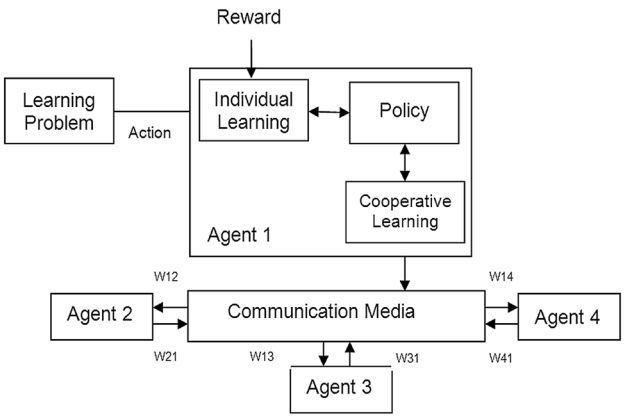

Fig. 1 Cooperative learning by weighted strategy sharing

Algorithm 1: WSS algorithm for agent a_i

1. *if InIndependentLearning Mode then*
2. $s_i := GetCurrentState()$
3. $a_i := ActionSelection()$
4. $Action(a_i)$
5. $r_i := Reward()$
6. $y_i := GoToNextState()$
7. $v(y_i) := Max_b \in Q(y_i,b)$
8. $Q_i^{new}(s_i,a_i) := (1-\beta_i)Q_i^{Old}(s_i,a_i)+\beta_i(r_i+\gamma_i V(y_i))$
9. $e_i := UpdateExpertness(r_i)$
10. *else {Cooperative Learning}*
11. *Loop* $j := 1$ *to* n
12. ***calculate normal expertise as*** $e_j^{Nrm} := \sum_{i=1}^{n} r_i(t)$
13. $Q_i^{new} := 0$
14. *Loop* $j := 1$ *to* n
15. $W_{ij} := ComputerWeights(i,j,e1.....en)$
16. $Q_j^{old} := GetQ(A_j)$
17. $Q_i^{new} := Q_i^{new} + W_{ij} * Q_j^{old}$

Variations in the Weighted Strategy Sharing method has been introduced by replacing Q learning under Independent Learning by Sarsa learning, Q(λ) learning and Sarsa(λ) learning algorithms [10, 11]. Results of each algorithm have been found and compared.

3.2 Weighted strategy sharing using Sarsa learning

Weighted strategy sharing (WSS) using Sara for cooperative learning is implemented. Every agent allocates a weight to their information and utilizes it depending on the amount of its teammate expertise in WSS method [20]

Algorithm 2 : WSS using Sarsa learning

1. *if InIndependentLearning Mode then*
2. $s_i := GetCurrentState()$
3. $a_i := ActionSelection()$
4. $Action(a_i)$
5. $r_i := Reward()$
6. $Q_i^{new}(s_i,a_i) := (1-\beta_i)Q_i^{Old}(s_i,a_i) + \beta_i(r_i+\gamma_i\, Q_i^{new}(s_i,a_i) - Q_i^{Old}(s_i,a_i))$
7. $s_i := FindNextState()$
8. $a_i := NewAction()$
9. $e_i := UpdateExpertness(r_i)$
10. *else {Cooperative Learning}*
11. *Loop* $j := 1$ *to* n
12. ***calculate normal expertise as*** $e_j^{Nrm} := \sum_{i=1}^{n} r_i(t)$
13. $Q_i^{new} := 0$
14. *Loop* $j := 1$ *to* n
15. $W_{ij} := ComputerWeights(i,j,e1.....en)$
16. $Q_j^{old} := GetQ(A_j)$
17. $Q_i^{new} := Q_i^{new} + W_{ij} * Q_j^{old}$

3.2.1 Independent learning based Sarsa learning

Sarsa learning is used for the Independent Learning Mode. Sarsa is an on policy version of Q-learning where policy is used to determine also the next action. The on policy Sarsa make use of the strategy resulting from Q values to decide subsequent action a and utilizes its Q value to determine the temporal difference. It is not waiting for all possible action to select the best. On policy, methods calculate the value of a policy while by means of it to get actions. They estimated Q value, the action values for current strategy, and then build up strategy slowly depend upon rough values for the present strategy. The plan enhancement is carried out in the easiest way using **ε**-greedy strategy with reference to current action value estimation. Sarsa learning algorithm is used for this purpose [19, 20].

3.3 Weighted strategy sharing using Q(λ) learning

Weighted strategy sharing (WSS) using Q(**λ**) for cooperative learning is implemented. Every agent allocates a weight to their information and utilizes it depending on the amount of its teammate expertise in WSS method [21]

Algorithm 3 : WSS using Q(λ) learning

1. *if InIndependentLearning Mode then*
2. $s_i := GetCurrentState()$
3. $a_i := ActionSelection()$
4. $Action(a_i)$
5. $r_i := Reward()$
6. $a^* \leftarrow argmax_b\, Q(s_i, a_i)$
7. $\delta \leftarrow r + \gamma Q(s', a^*) - Q(s_i, a_i)$
8. $e(s_i, a_i) \leftarrow e(s_i, a_i) + 1$
9. *for all* s_i, a_i
10. $Q(s_i, a) \leftarrow Q(s_i, a_i) + \alpha\delta e(s_i, a_i)$
11. *If* $a_i' = a^*$ *then* $e(s_i, a_i) \leftarrow \gamma\lambda e(s_i, a_i)$ *else* $e(s_i, a_i) \leftarrow 0$
12. $s_i := FindNextState(), a_i := NewAction()$
13. $e_i := UpdateExpertness(r_i)$
14. *else {Cooperative Learning}*
15. *Loop* $j := 1$ *to* n
16. ***calculate normal expertness as*** $e_j^{Nrm} := \sum_{i=1}^{n} r_i(t)$
17. $Q_i^{new} := 0$
18. *Loop* $j := 1$ *to* n
19. $W_{ij} := ComputerWeights(i,j,e1.....en)$
20. $Q_j^{old} := GetQ(A_j)$
21. $Q_i^{new} := Q_i^{new} + W_{ij} * Q_j^{old}$

3.3.1 Independent learning based Q(λ) learning

Q(λ) does not consider the end of the event in its support. It only considers next exploratory action. Q(λ) looks one action previous the first searching using its awareness of the action values. The trace revise is considered as happening in two stages [20, 21].

3.4 Weighted strategy sharing using Sarsa(λ) Learning

Weighted strategy sharing (WSS) using Sarsa (λ) for cooperative learning is implemented. Every agent allocates a weight to their information and utilizes it depending on the amount of its teammate expertise in WSS method.

Algorithm 4 : WSS using Sarsa(λ) learning

```
if InIndependentLearning Mode then
    s_i := GetCurrentState()
    a_i := ActionSelection()
    Action(a_i)
    r_i := Reward()
        δ ← r + γQ(s'_i, a*) – Q(s_i, a_i)
        e(s_i, a_i) ← e(s_i, a_i) + 1
    for all s_i, a_i
        Q(s_i, a) ← Q(s_i, a_i) + αδe(s_i, a_i)
        e(s_i, a_i) ← γλe(s_i, a_i)
    s_i := FindNextState()
    a_i := NewAction()
    e_i := UpdateExpertness(r_i)
else {Cooperative Learning}
    Loop j := 1 to n do
    calculate normal expertness as e_j^Nrm := Σ_{i=1}^{n} r_i(t)
    Q_i^new := 0
    Loop j := 1 to n do
        W_ij := ComputerWeights(i,j,e1.....en)
        Q_j^old := GetQ(A_j)
        Q_i^new := Q_i^new + W_ij * Q_j^old
```

3.4.1 Independent learning based Sarsa (λ) learning

The eligibility trace version of Sarsa is called as Sarsa(λ) [22]. The trace for state action pair of x, y is denoted by $e_t(x, y)$; substituting state action variables for state variables the equation becomes $Q_{t+1} = Q_t(x, y) + \alpha\delta_t e_t(x, y)$ for all x, ywhere

$$\delta_t = r_{t+1} + \gamma Q_t(x_{t+1}, y_{t+1}) - Qt(x_t, y_t)$$

and

$$e_t(x, y) = \gamma\lambda e_{t-1}(x, y) + 1 \quad \text{if } x = x_t \text{ and } y = y_t$$
$$= \gamma\lambda e_{t-1}(x, y) \quad \text{otherwise}$$

The Sarsa (λ) trace method strengths many actions of the sequence [23].

4 Experimental setup

Maximize the sale of products that depend on the price of the product, customer age and period of sale. These are the information available to each agent i.e. shop. So it becomes the state of the environment. The final result is to increase the revenue by increasing total retailing of products.

4.1 Input dataset

The action set is defined as the retailing of probable items. i.e. $A = \{p1, p2, p3. \ldots . p10\}$.

Hence action a ∈A. State of the system is a line of consumers in given period for given store agent [24]. State can be defined as

$$S(t) = \{ s_{1(t)}, s_{2(t)}, n \}$$

where $s_1 \rightarrow \{Y, M, O\}$ is the consumer queue with i.e. young age, middle age and old age consumer, $s_2 \rightarrow \{H, M, L\}$ is the Highest price, Medium price, Lowest price, $n \rightarrow \{1, 2, 3, 4\ldots\ldots\ldots..12\}$ is the month of item sale.

In a system minimum, 108 states and actions are possible. A number of state-action increases as the number of transactions increases. For simplicity, it is assumed that single state for each transaction else the state space becomes infinitely large [23, 24]. Shop agent observes the queue and decides product i.e. action for each customer/state. After every sale reward is given to the agent. The Table 1 shows the snapshot of the dataset generated for single shop agent.

In a particular season, the sale of one shop increases. With the help of cooperative learning, other shops learn about the increase in the sale and they can take necessary actions for their profit maximization.

4.2 State and action selection

Action selection mechanism in Q learning is accountable for choosing the actions such as the agent would carry out throughout the training procedure [23–25].Let $s = \{s_1, s_2 \ldots s_i\}$ be one of these vectors, then the probability s_i of selecting action i is given by

Table 1 Dataset

Transaction ID	Age	Price	Month	Action selected (product)
1	Y	L	1	P1, P2, P4
2	Y	M	1	P2, P3
3	Y	H	1	P3, P4
4	M	L	1	P1, P2
5	M	M	1	P1, P2, P3
6	M	H	1	P4, P2
7	O	L	1	P1, P3

$$s_i = (1-\varepsilon) + (\varepsilon/m) \quad \text{if Q of i is maximum}$$
$$= \varepsilon/m \quad \text{otherwise}$$

where m is the number of actions in the set.

One way to assign such probabilities is

$$P(x_i/y) = C^{Q\prime(x,yi)} \Big/ \sum_{j.} C^{Q\prime(x,yj)},$$

$P(x_i/y)$ is the action selection probability y_i, x is the current state, C is the constant > 0. The high value of C assigns high probabilities to action i.e. maximum reward and the small value of K assign higher probabilities to other action i.e. minimum reward [25].

5 Results

Weighted strategy sharing algorithms i.e. one step Q learning, Q(λ) learning, Sarsa learning and Sarsa (λ) learning are compared using two parameters reward vs episodes as shown in Fig. 2. Sarsa (λ) learning gives highest rewards compared to other three methods due to the addition of eligibility traces. However, it's graph is fluctuating in nature. Sarsa learning gives second highest reward values and smoothly decreases rewards as an increase in a number of episodes. Q(λ) learning receives low rewards compared to Sarsa & Sarsa (λ) learning. There is a huge difference between the rewards received by Q(λ) learning (2500–3000) and Sarsa (λ) learning (7000–8000). One step Q learning receives lowest rewards and numbers of rewards decreases as an increase in a number of episodes and after some episodes (50) it remains constants.

Weighted strategy sharing algorithms i.e. one step Q learning, Q(λ) learning, Sarsa learning and Sarsa (λ) learning are compared using two parameters reward vs learning rate as shown in Fig. 3. Sarsa (λ) learning gives highest rewards compared to other three methods due to the addition of eligibility traces. Increase in learning rate steadily increases the number of rewards received by the agent. Sarsa learning gives second highest reward values and nature is not fixed. At learning rate 0.3, a number of rewards received drop suddenly and after that number of rewards increases. Q(λ) learning receives comparable rewards compared to Sarsa & Sarsa (λ) learning At learning rate 0.3, rewards received by Q(λ) and Sarsa learning are same. The difference is not much more between the rewards received by Q(λ) learning, Sarsa learning, and Sarsa (λ) learning. One step Q learning receives lowest rewards within the range of 500. After learning rate 0.4 one step Q learning has some increase in rewards.

Weighted strategy sharing algorithms i.e. one step Q learning, Q(λ) learning, Sarsa learning and Sarsa (λ) learning are compared using two parameters reward vs

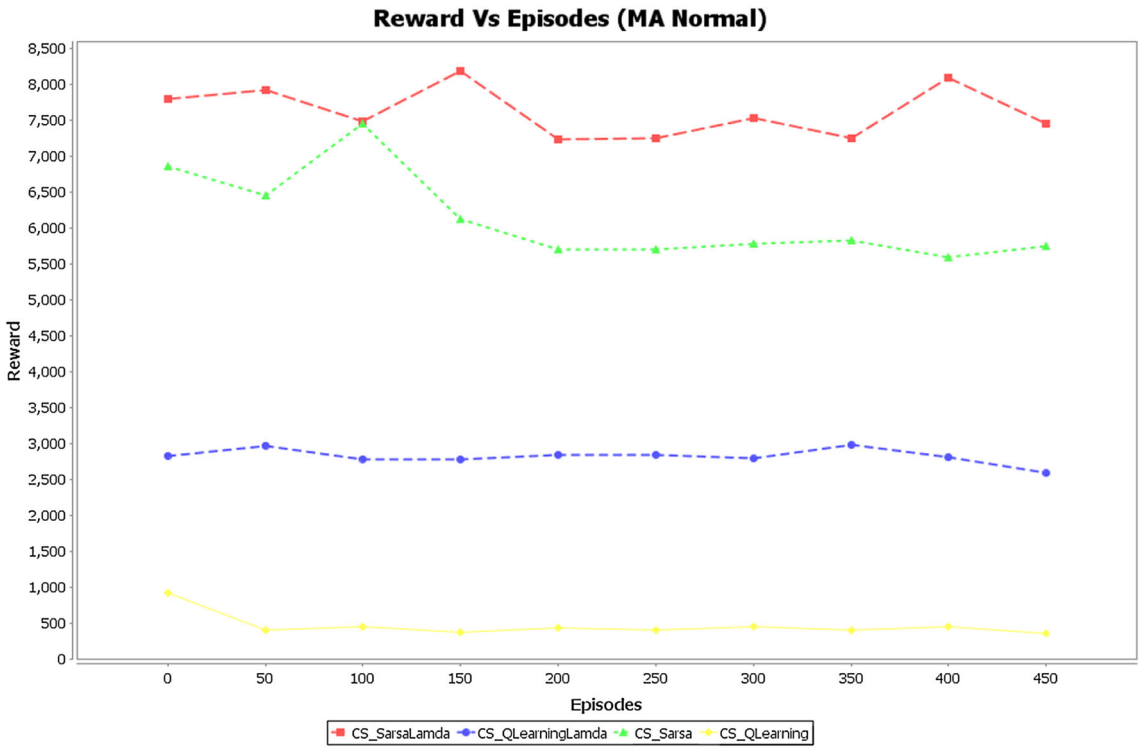

Fig. 2 Graph of reward Vs episode for four algorithms

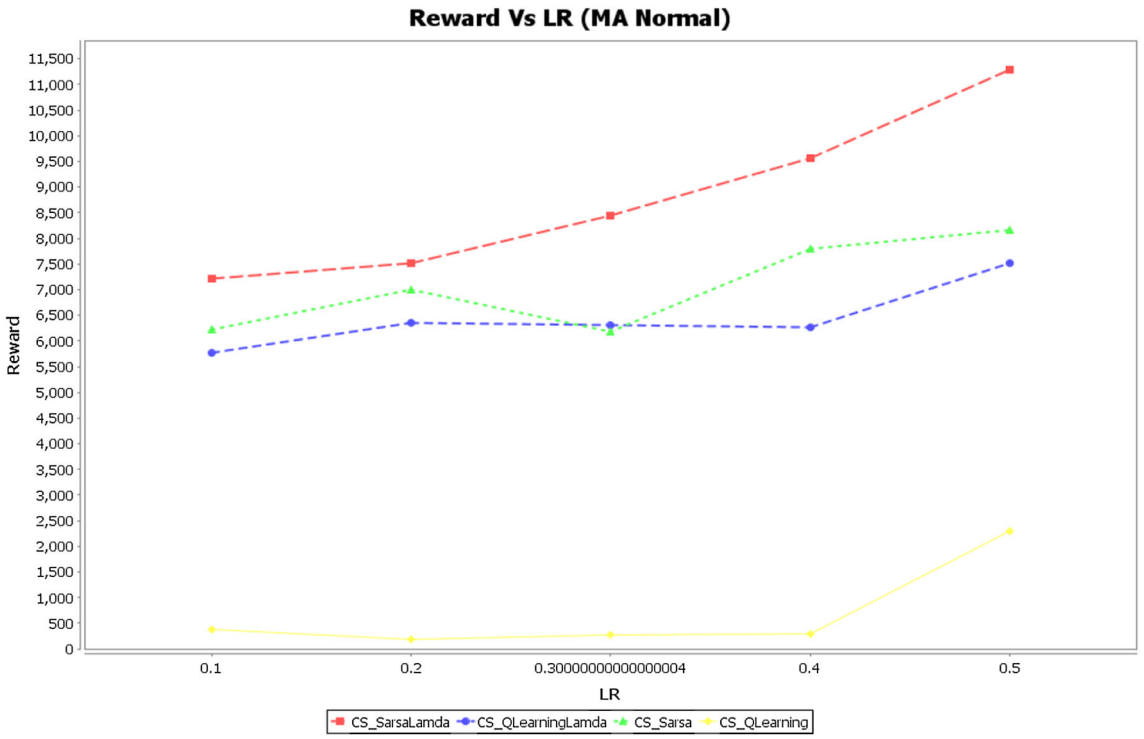

Fig. 3 Graph of reward Vs learning rate for four algorithms

discount rate as shown in Fig. 4. Sarsa (λ) learning gives highest rewards compared to other three methods due to the addition of eligibility traces. However, it's graph is fluctuating in nature. Sarsa learning gives second highest reward values and smoothly increases rewards as an increase in discount rate. Q(λ) learning receives moderate

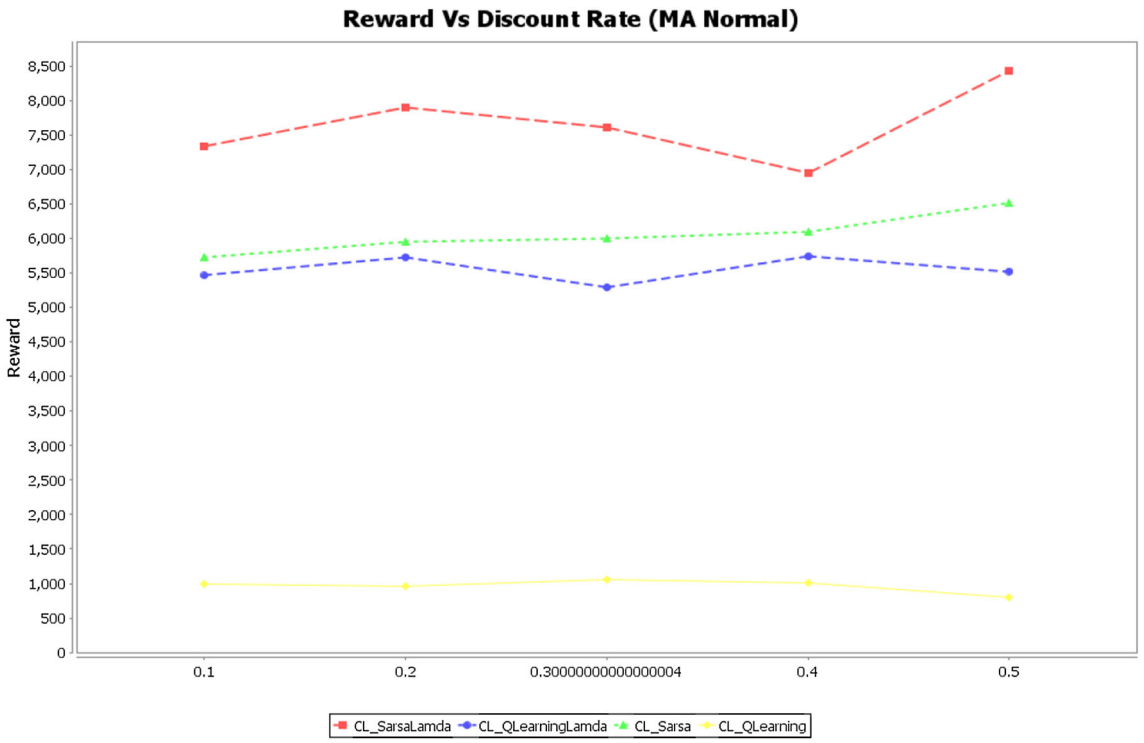

Fig. 4 Graph of reward Vs discount rate for four algorithms

rewards compared to Sarsa & Sarsa (λ) learning. There is a large difference between the rewards received by Q(λ) learning (5000–6000) and one step Q learning (1000–1500).

It has demonstrated that a shop agent can successfully make use of reinforcement learning in selecting items dynamically to increase its profit matrix. It is believed that this is a promising approach for profit maximization in retail market environments with limited information. In cooperative learning with Weighted Strategy Sharing algorithm, two agents use one another's knowledge and action set. After learning cooperatively from each other each one receives its Q table. Significant improvement is seen in the results compared to multiagent learning as agents receive more knowledge. Both agents are enhancing the sale of products to increase the revenue by learning cooperatively. Above graphs demonstrate the performance of Weighted Strategy Sharing algorithms with Normal Expertness for rewards with reference to three parameters discount rate, learning rate and a number of episodes. Weighted strategy sharing with normal expertise outperforms by implementing Sarsa (λ), Sarsa and Q(λ) learning as compared to one step Q learning. It receives maximum rewards for this three algorithm. Profit calculated by each shop agent directly depends on the rewards received by that agent. three shop agents can obtain the maximum profit by following Sarsa (λ), Sarsa and Q(λ) learning. In other words, cooperation based on normal expertness gives more benefit in terms of profit for three shop agents. The results obtained by the proposed cooperation methods show that such methods can put into a quick convergence of agents in the dynamic environment. It also shows that cooperative methods give a good presentation in dense, incompletely and composite circumstances.

6 Conclusion

Cooperative learning algorithms are more efficient and effective and produce best results. Learning algorithms are best suitable for decision making. In cooperative learning, sharing of more knowledge and information is possible, all agents' knowledge is used equally, jointly solves the problem. The performance of cooperative learning algorithms is improved as compared to multiagent learning approach. Reinforcement learning is mainly implemented with game theory and robot applications. Paper gives the approach for reinforcement learning methods applied to the diagnostic application. Combination two methods i.e. weighted strategy sharing with expertness parameter certainly enhances the performance of learning. Weighted strategy method is implemented with Sarsa (λ), Q(λ) and Sarsa learning for cooperation between the agents that was not implemented previously. However, these methods are still unable to find a more expert agent as it calculates expertise value only using the algebraic sum of the reinforcement signals. Hence, the future scope of this paper shall be emphasized on enhancing the cooperative learning algorithms for decision making using with different expertise measures.

References

1. Vidhate DA, Kulkarni P (2017) "A Framework for Improved Cooperative Learning Algorithms with Expertness (ICLAE)", International Conference on advanced computing and communication technologies advances in intelligent systems and computing, 562nd edn. Springer, Singapore, pp 149–160
2. Vidhate DA, Kulkarni P (2017) "Expertise Based Cooperative Reinforcement Learning Methods (ECRLM)", International Conference on Information & Communication Technology for Intelligent System, Springer book series Smart Innovation, Systems and Technologies (SIST, volume 84). Springer, Cham, pp 350–360
3. Park K-H, Kim Y-J (2015) Modular Q-learning based multi-agent cooperation for robot soccer. Robot Auton Syst 35:3026–3033
4. M Camara, O Bonham-Carter, J Jumadinova (2015) A multi-agent system with reinforcement learning agents for biomedical text mining. Proceedings of the 6th ACM Conference on Bioinformatics, Computational Biology and Health Informatics, BCB '15. NY, USA, ACM pp 634–643
5. Vidhate DA, Kulkarni P (2016) "Innovative Approach Towards Cooperation Models for Multi-agent Reinforcement Learning (CMMARL)" Springer Nature series of communications in computer and information. Science 628:468–478
6. H. Iima, Y Kuroe (2015) Swarm reinforcement learning methods improving certainty of learning for a multi-robot formation problem. CEC pp 3026–3033
7. DA Vidhate, P Kulkarni (2017) "Enhanced Cooperative Multi-agent Learning Algorithms (ECMLA) using Reinforcement Learning" International Conference on Computing, Analytics and Security Trends (CAST), IEEE Xplorer, pp 556–561
8. Al-Khatib AM (2011) Cooperative machine learning method. World Comput Sci Inform Technol J 1(9):380–383 **(ISSN: 2221–0741)**
9. Araabi Babak Nadjar, Mastoureshgh Sahar, Ahmadabadi Majid Nili (2010) A Study on Expertise of Agents and Its Effects on Cooperative Q-Learning. IEEE Transact Evol Comput 14:23–57
10. DA Vidhate, P Kulkarni (2016) "New Approach for Advanced Cooperative Learning Algorithms using RL Methods (ACLA)" Proceedings of the Third International Symposium on Computer Vision and the Internet, ACM, pp 12–20
11. Dr. Hamid R. Berenji David Vengerov (2000) "Learning, Cooperation, and Coordination in Multi-Agent Systems", In Proceedings of Ninth IEEE International Conference on Fuzzy Systems
12. EM de Cote, A Lazaric, M. Restelli (2006) Learning to cooperate in multi-agent social dilemmas. Autonomous Agents & Multi-Agent System, pp. 783–785
13. DA Vidhate, P Kulkarni (2016) "Performance enhancement of cooperative learning algorithms by improved decision making for context-based application", International Conference on Automatic Control and Dynamic Optimization Techniques (ICACDOT) IEEE Xplorer, pp 246-252,

14. Jun-Yuan Tao, De-Sheng Li "Cooperative Strategy Learning In Multi-Agent Environment With Continuous State Space", IEEE International Conference on Machine Learning and Cybernetics, 2006
15. Panait L, Luke S (2005) Cooperative multi-agent learning: the state of the art. J Auton Agents Multi-Agent Syst 11(3):387–434
16. Vidhate DA, Kulkarni P (2017) Multi-agent cooperation models by reinforcement learning (MCMRL). Int J Comput Appl 176(1):25–29
17. Nagendra Prasad MV, Lesser VR (1999) Learning situation-specific coordination in cooperative multi-agent systems. J Auton Agents Multi-Agent Syst 2(2):173–207
18. Vidhate DA, Kulkarni P (2016) Enhancement in decision making with improved performance by multiagent learning algorithms. IOSR J Comput Eng 1(18):18–25
19. Vidhate DA, Kulkarni P (2016) Single agent learning algorithms for decision making in diagnostic applications. SSRG Int J Comput Sci Eng 3(5):2348–8387
20. Z Abbasi, MA Abbasi (2002) "Reinforcement Distribution in a Team of Cooperative Q-learning Agent", Proceedings of the Ninth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing
21. Vidhate DA, Kulkarni P (2016) Implementation of multiagent learning algorithms for improved decision making. Int J Comput Trends Technol 35(2):60–66
22. Young-Cheol Choi, Student Member, Hyo-Sung Ahn (2010) "A Survey on Multi-Agent Reinforcement Learning: Coordination Problems", IEEE/ASME International Conference on Mechatronics and Embedded Systems and Applications, pp 81–86
23. Vidhate DA, Kulkarni P (2016) A step toward decision making in diagnostic applications using single agent learning algorithms. Int J Comput Sci Inform Technol 7(3):1337–1342
24. M Camara, O Bonham-Carter, J Jumadinova (2015) "A Multi-agent System with Reinforcement Learning Agents for Biomedical Text Mining", Proc. of the Sixth ACM Conf. on Bioinformatics, Computational Biology, and Health Informatics, BCB'15, USA, ACM, pp. 634–643
25. Vidhate DA, Kulkarni P (2014) Multilevel relationship algorithm for association rule mining used for cooperative learning. Int J Comput Appl 86(4):20–27