Soft Computing (2019) 23:12673–12682
https://doi.org/10.1007/s00500-019-03819-5

METHODOLOGIES AND APPLICATION

Extracting and reusing blocks of knowledge in learning classifier systems for text classification: a lifelong machine learning approach

Muhammad Hassan Arif[1] · **Muhammad Iqbal**[2] · **Jianxin Li**[1]

Published online: 9 March 2019

Abstract
Human beings follow a continuous learning paradigm, i.e., they learn to solve smaller and relatively easy problems, retain the learnt knowledge and apply that knowledge to learn and solve more complex and large-scale problems of the domain. Currently, most machine learning and evolutionary computing systems lack this ability to reuse the previous learnt knowledge. This paper presents a lifelong machine learning model for text classification that extracts the useful knowledge from simple problems of a domain and reuses the learnt knowledge to learn complex problems of the domain. The proposed approach adopts a rule-based learning classifier system, and a rich encoding scheme is used to extract and reuse building units of knowledge. The experimental results show that the continuous learning approach outperformed the baseline classifier system.

Keywords Learning classifier systems · Lifelong learning · Code fragments · Transfer learning

1 Introduction

Lifelong machine learning (LML) is a learning paradigm that learns continuously like humans, and it retains the knowledge gained from solving previous tasks and uses it to solve future tasks (Chen and Liu 2016). In this process, the learning agent becomes more knowledgeable and can learn such problems which are not possible in isolation. Humans do not learn in isolation; instead, they learn continually and retain the knowledge previously learned from small and related problems which help them in future learning to solve complex problems. Currently, most machine learning (ML) algorithms work in isolation, train on the current problem, build a model and solve the current problem. They do not retain that knowledge, and whenever a new more complex problem is presented, they need to re-learn from the start. In modern day, artificial intelligence applications that interact with humans or systems in real time such as physical robots, chat bots, and intelligent assistants also need lifelong learning capabilities.

Humans build their language, understanding and knowledge throughout their life span, and they do not need to read 5000 negative product reviews and 5000 positive product reviews to classify a new review as either negative or positive. But if they do not build up this past knowledge, they cannot classify it manually even such number of training examples. Natural language processing (NLP) problems are the key target of LML because language learned from one problem can help to learn other problems (Chen et al. 2015). Sentences in different domains have the same structure and phrases in sentences and bear almost the same meanings in different domains. Therefore, the knowledge learned from solving one NLP problem can be used to learn other problems of the same or the related domains.

LML is currently a hot research area, and it is used in different fields of ML. The target of LML is those systems that learn multiple tasks sequentially from one or more domain. Silver et al. (2013) provided a review of the previous work on LML in supervised, unsupervised and reinforcement learning methods and advocated the use of LML techniques in all fields of artificial intelligence. Shu et al. (2017) applied LML

Communicated by V. Loia.

✉ Muhammad Hassan Arif
mhassanarif@act.buaa.edu.cn

Muhammad Iqbal
miqbal1@hct.ac.ae

Jianxin Li
lijx@act.buaa.edu.cn

[1] Advanced Innovation Center for Big Data and Brain Computing, School of Computer Science and Engineering, Beihang University (BUAA), Beijing 100191, China

[2] Faculty of Computer Information Science, Higher Colleges of Technology, Fujairah, United Arab Emirates

for the supervised aspect extraction. They updated the conditional random fields and used the lifelong learning pattern. It is found that LML model of reusing the previous knowledge of the aspect extractions from previous experiments improved the results.

Sutton et al. (2007) discussed that continuous learning should be incorporated in reinforcement learning model. A continuous learning agent is proposed that keeps learning during its operations throughout its life span, and it adapts multiple different environments during learning. Continuous learning achieved the better results than the static learning solution. Chen and Liu (2016) discussed LML paradigm in detail and compared it with other paradigms like transfer learning and multitask learning.

LML can be defined as: "It is a continuous learning process. The learner performs a sequence of N learning tasks, $\{T_1, T_2, \ldots, T_N\}$, on N data sets, i.e., $\{D_1, D_2, \ldots, D_N\}$. These tasks and data sets can be from the same or relevant domains. Learner extract useful knowledge learned from these N previous tasks and retain it in the knowledge base (KB). When the learner is presented with a new task, i.e., T_{N+1}, it uses the past knowledge stored in KB to learn $T_{(N+1)\text{th}}$ task. KB is updated after the completion of every task, and new useful knowledge is added for future use." Three important points can be depicted from this definition: (1) An LML model should follow the continuous learning paradigm. (2) It should devise a mechanism to extract, accumulate and maintain the past knowledge, and (3) It should be capable of reusing the past knowledge in some useful way and that past knowledge should help in learning the future tasks. Therefore, an LML agent should be capable of learning a series of problems of the same or the similar domain, extracts and stores useful knowledge and reuses that extracted knowledge to the future problems. This definition does not restrict the type of knowledge extracted from the previous problems and how it can be reused in the future problems.

Learning classifier systems (LCS) are rule-based systems (Urbanowicz and Moore 2009) that assimilate ML models and evolutionary computing techniques to evolve classifier rules (Iqbal et al. 2015). These rules have two main parts: the condition and the action, which are accompanied by a set of parameters. An LCS agent interacts with unknown environment and learns the problem by evolving the rule set on the basis of the reward given by the environment. Evolutionary component of LCS uses genetic algorithm's (GA) crossover and mutation operations to evolve the learning component and finds a better set of rules.

Wilson (1995, 2000) proposed XCS and XCSR, which are accuracy-based LCS models to learn binary and real-valued classification problems, respectively. XCSR uses interval-based conditions, and it tries to generate optimized rules. Hassan et al. applied XCSR, for the first time, in the text classification domain to solve social media text classification problems (Arif et al. 2017c, 2018). XCSR showed relative good performance in case of small data sets, but its performance decreased with the increase in number of records in a data set and the number of features in a record. XCSR learns interval for every feature that is very hard in case of sparse social media data sets. The standard XCSR system was enhanced by introducing the concept of "don't care" intervals. A "don't care" interval satisfies any value of the corresponding feature; therefore, the resulting system XCSR# does not need to learn interval range for all the features and it has potential to explicitly handle the sparseness issue. The introduction of explicit "don't care" intervals in XCSR# showed a great deal of improvement over XCSR. However, it was observed from the obtained results that when the problem size becomes very large, the benefit of "don't care" intervals decreases. So, a new condition representation was required to handle very large-scale high-dimensional social media text classification problems.

Previously, Iqbal et al. (2014) extended XCS to XCSCFC by introducing a genetic programming tree like rich encoding scheme (called code fragments) to represent the classifier conditions, to extract and reuse knowledge in order to solve large-scale Boolean problems. Recently, Arif et al. (2017b) extended XCSR to XCSRCFC using code fragment-based conditions and successfully solved high-dimensional real-valued text classification problems from various social media text analysis domains, i.e., sentiment analysis of tweets and reviews (IMDB, Amazon, Yelp), and spam detection in SMS messages and emails. This paper presents a further extension of XCSRCFC as a LML model for text classification (Arif et al. 2017a). Code fragments can be used as knowledge extractor and can store useful knowledge. In the proposed LML model, the learning procedure of XCSRCFC is updated according to the LML model. The structure of code fragment is updated, and in the new structure, either its leaf nodes can be terminal symbols, or these can be previously learned fitter code fragments.

In the proposed model, large problems are divided into a bottom-up hierarchical layers of problems where each higher level problem contains all the previous records and features. Suppose the original task (i.e., the full dataset) consists of N records and we want to decompose this task into five subtasks (i.e., sub-datasets). Then, the decomposition will be as follows. First of all, we choose five positive numbers, n_1, n_2, n_3, n_4 and n_5 such that $n_1 < n_2 < n_3 < n_4 < n_5 = N$. Level_1 task/dataset consists of n_1 records, which are randomly selected from the original dataset of N records. Level_2 task consists of n_2 records, which contains n_1 records from the level_1 task and $n_2 - n_1$ records from the original $N - n_1$ records (i.e., the original dataset excluding the level_1 records). Level_3 task consists of n_3 records, which contains n_2 records from the level_2 task and $n_3 - n_2$ records from the original $N - n_2$ records (i.e., the original dataset excluding the

$level_2$ records). Similarly, $Level_4$ task consists of n_4 records, which contains n_3 records from the $level_3$ task and $n_4 - n_3$ records from the original $N - n_3$ records (i.e., the original dataset excluding the $level_3$ records). Finally, the $level_5$ task consists of n_5 records that is actually the whole dataset.

Optimized rules with code fragment-based conditions trained on the previous simple problems contain relevant knowledge and can act as the past information stores (PIS) for future tasks. A knowledge miner (KM) system is devised to mine fitter rules from PIS, and these rules have fitness higher than average fitness of the population. All distinct non-don't care code fragments from the fitter rules are stored as the knowledge base (KB) for further learning. In the proposed LML system, a knowledge-based learner (KBL) is trained that uses fitter code fragments from KB and reuses them in learning future high-dimensional problems.

The key difference of LML from other transfer learning (TL) techniques is that TL-based methods are usually used in those situations when target domain has no or very little training data. In TL-based methods usually, a classifier is trained on one or more source domains and then the trained model is applied on the target domain that has very little or no training data. The knowledge extracted from the results of the previous learning is not used to learn new problems. TL-based methods are usually not applied in those cases when the target/future task has good training data. In cases when the target problem has good training data, TL-based methods usually give poor results than the traditional non-TL-based learning methods. In contrast, the target of LML is those cases when the target task has good training data, and its aim is to improve the learning process using both the knowledge gained during the previous learning and the training data of the target task. The target of LML is those problems that have a bottom-up hierarchical buildup, i.e., all the features from the previous level simple problems exist in next level complex problem.

The results show that training a code fragment-based XCSR agent to solve a small problem with the small feature set is comparatively easy. Code fragments can act as the building blocks of knowledge for further reuse because they store knowledge independent of any specific classifier.

The proposed LML paradigm satisfies the three key characteristics of LML:

1. It has a continuous learning process. It learns from small problems and then moves toward large problems.
2. The proposed LML model has explicit knowledge retention policy. It selects the experienced and accurate rules from the previously solved problems, and it extracts and stores distinct code fragments from these experienced rules.
3. These extracted fitter code fragments store the previously learned knowledge which can be reused. The proposed LML model introduced a new code fragment structure that reuses the previously learned code fragments and improves the future learning process.

The rest of this paper is organized as follows. Section 2 describes the implementation detail of the proposed approach. In Sect. 3, problem domains and the experimental setup are described. Results are presented in Sect. 4, and a comprehensive discussion is provided in Sect. 5. This study is concluded in Sect. 6.

2 The proposed method

It is easy for code fragment-based systems to generate an optimal population for small problems with a small feature set. A novel scheme is proposed in which LML model is used to extract useful information from the past problems to improve learning in the future problems. A new lifelong learning XCSRCFC system called "LL-XCSRCFC" is proposed that reuses the previously learnt knowledge by solving the small problems.

The proposed LML model tries to solve problems as humans do, i.e., it learns small and simple problems, retains the knowledge learnt from those problems, mines the useful information from the stored knowledge and reuses the extracted knowledge to learn future problems. Figure 1 shows the main components of the proposed LML model.

1. *Task Manager (TM)* In the proposed method, the main task of classifying a large text data set is divided into a bottom-up hierarchy of tasks. Large data sets are subdivided into a bottom-up hierarchy of data sets; for example, the data set at each level contains all records from the previous level plus some additional records. Therefore, initial levels contain a small number of records with a small number of features. At higher levels, the number of features increases with the increase in the number of records. The tasks are presented sequentially to the learning agent by the task manager.
2. *Past Information Store (PIS)* A population of optimal general rules is generated by XCSR-based systems as a solution to a given problem. The rule set generated by LL-XCSRCFC, generated for small problems, can act as PIS for the next level problems. PIS contains both experienced and non-experienced rules. Experienced rules have higher fitness value than the non-experienced rules. Whenever a new task is presented to the LL-XCSRCFC system by the task manager, then the system generates a new rule set as a solution to the presented task and PIS is updated to incorporate a new rule set. Therefore, PIS is updated after each learning cycle.

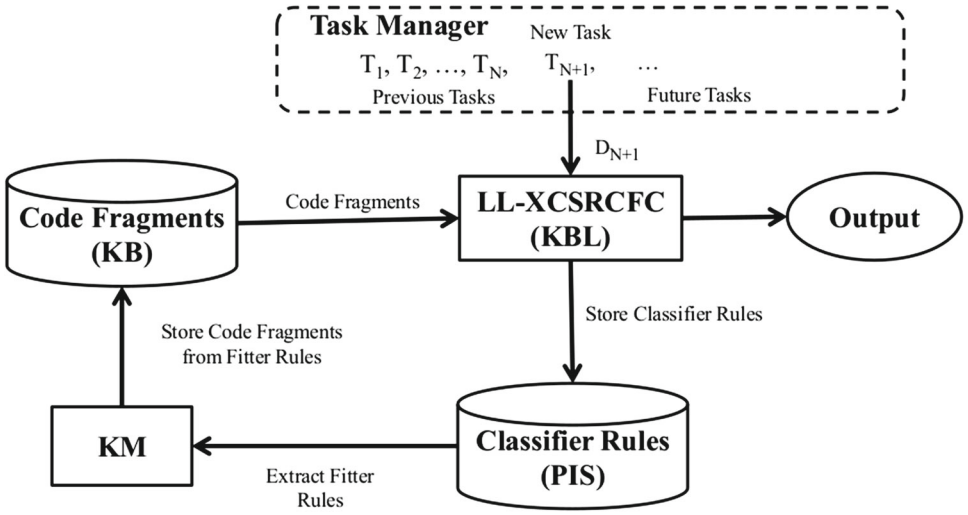


Fig. 1 Proposed lifelong machine learning model

3. *Knowledge Miner (KM)* PIS stores all the rules which are created during the learning of the previous task. PIS contains both accurate and non-accurate rules generated during the past learning. Only accurate and experienced rules best provides the solution and can be used for further learning. An important aspect of lifelong learning is to identify the useful information that can be extracted from past tasks and reused for future learning. A knowledge miner (KM) agent is used to extract useful knowledge from PIS. KM selects all experienced and accurate rules in the final population having fitness greater than average fitness. These rules are not directly used as part of the solution to next higher level problems. KM extracts distinct non-don't care code fragments from fitter rules for further reuse.
4. *Knowledge Base (KB)* Code fragments generated at any level store useful knowledge and can act as building blocks of knowledge for further reuse because they store knowledge independent of any specific classifier. KM extracts all distinct non-don't care code fragments from fitter rules of PIS and store them in knowledge base KB.
5. *Knowledge-Based Learner (KBL)* LL-XCSRCFC uses a knowledge-based learning agent, which reuses fitter code fragments from the previous KBs to solve high-dimensional problems at the higher level layers. The system can reuse any number of code fragments not just from one previous level KB, but from all the previous level KBs. These code fragments are used as terminal symbols in the next level code fragments. LL-XCSRCFC classifier is trained on the smaller problems, and the knowledge extracted from those problems is reused to train a classifier on the large problems. LL-XCSRCFC improves the learning process by using the target task data along with the knowledge extracted from the previous learning.

   A code fragment in a higher level problem uses code fragments from the previous KBs as leaf nodes with probability P_{kb}. In these experiments, P_{kb} is set to 0.5. In new structure of code fragment, either each leaf node refers to a feature in the input state with an attached random interval, or it refers to a code fragment from a previous KB. In a multiple level problem, a leaf node can be a code fragment from any previous level KB. It may be possible that a code fragment can have only one leaf node which can refer to a feature or it can be a code fragment from KB. Therefore, each terminal symbol of each non-don't care code fragments is set either according to the corresponding feature's value from the observed input state or if the leaf node is a previous code fragment, then it is set according to the output of that code fragment. This is further explained in the following subsection.

2.1 Structure and evaluation process of LML code fragment

In LL-XCSRCFC, code fragment structure is changed and it may have one or more previous level code fragments as its leaf node. Therefore, code fragment evaluation mechanism is changed in this method. When a code fragment is evaluated, its terminal values are loaded and if a terminal symbol refers to a code fragment from the previous level, it is evaluated first and its result is loaded as the terminal value. Therefore, in this approach code fragments are evaluated recursively. Figure 2 shows structure of a sample code fragment, using a previous level code fragment as its leaf node.

It is a depth 2 code fragment having four leaf nodes. One leaf node refers to a previous level code fragment, i.e., 13th code fragment from $KB2$, and other leaf nodes are terminal symbols that refer to the corresponding features of input, i.e., T_3, T_6 and T_7. A random interval of the form $[l_i, u_i]$ is attached with each terminal symbol. During the recursive

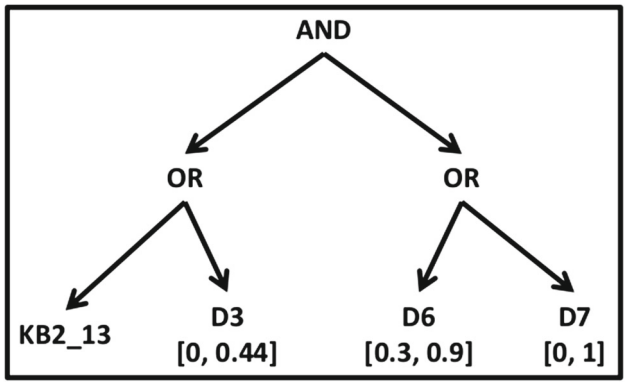


Fig. 2 A sample code fragment using a previous level code fragment as a leaf

evaluation of code fragments, firstly all the leaf nodes are checked and the terminal values are set either 0 or 1. If the leaf node refers to a terminal symbol with an attached interval, then the value of the corresponding input feature is matched against the interval range. If its value lies in the interval range, the terminal value is set to 1 else it is set to 0. If the leaf node refers to a previous level code fragment, it is evaluated first and its evaluation value is used as the terminal symbol. The code fragment in Fig. 2 has 4 leaves, but its value depends on only 2 nodes. One branch of tree always evaluates to 1 as the interval attached with $D7$ has maximum span. Therefore, this code fragment evaluates to 1, either if the previous level code fragment KB2_13 evaluates to 1 or the value of input feature 3 lies in the range [0, 0.44].

Algorithm 1 describes the code fragment evaluation procedure which is called recursively. Here, *num* is equal to the number of leaves in the code fragment. Its maximum value can be 4 as the largest code fragment with depth 2 can have maximum 4 leaves.

Algorithm 1: Evaluate_Code_Fragment(cf, T)

Data: a code fragment cf, and the current input state T
Result: the evaluated Boolean value of cf against the state T

```
num ← the number of leaves in cf
for i = 1 to num do
    if leaf_i is an extracted code fragment then
        terminalVal_i ← Evaluate_Code_Fragment(leaf_i, T)
    else
        if the feature value T_j falls in the leaf_i's interval D_j then
            terminalVal_i ← 1
        else
            terminalVal_i ← 0
        end
    end
end
evaluatedVal ← evaluate cf
return evaluatedVal
```

Table 1 A sample population of LML classifiers

Data set	Code fragment	
	Name	Expression
Level 1	KB1_0	(D1[0.1, 0.5] OR D2[0.1, 0.8]) AND (D3[0, 0.8] AND D6[0.2, 0.5])
	KB1_1	D5[0, 0.7]
	…	…
Level 2	KB2_0	(D0[0.13, 0.39] AND (D4[0, 0.55])) OR KB1_7
	KB2_1	(D4[0.33, 0.77] OR KB1_4) AND (KB1_34 OR D5[0.2, 0.84])
	…	…
Level 3	KB3_0	D4[0, 0.1] OR KB1_18
	KB3_1	KB2_31 NOR KB1_10
	…	…

LL-XCSRCFC is applied on the first level small data set, and the optimal general solution is provided by a set of rules. Then, the code fragments from fitter rules are extracted and reused at next level. Table 1 shows some sample code fragments from KBs of a three-level problem. The code fragments from level 1, 2 and 3 KB are named as $KB1_i$, $KB2_i$ and $KB3_i$, respectively. Each level can contain code fragments from all the previous levels. Small problems contain fewer features, and LL-XCSRCFC can learn these problems with a small number of rules, and each rule contains a small number of code fragments. To handle the large high-dimensional problems, the condition length of rule, i.e., number of code fragments used in a rule, can be increased.

The proposed LL-XCSRCFC approach improves code fragment-based XCSR system described in XCSRCFC reference in the following aspects: rule matching operation, the covering operation and mutation operation.

2.2 Condition matching operation

Each new example is matched against the current population $[P]$, and all matching classifiers are selected in the match set $[M]$. A classifiers condition is matched against the current input if all the code fragments in the classifier's condition output 1 against the current input. The condition matching procedure is updated in LL-XCSRCFC because the structure of code fragment is updated and a recursive evaluation procedure is introduced. A classifier condition in LL-XCSRCFC can contain three types of code fragments:

1. It can contains "don't care" code fragments with probability $P_{\text{don'tCare}}$. Since "don't care" code fragments always evaluate to 1, they are given a special ID. During condition matching process, if ID of code fragment

is matched with "don't care" code fragment's ID, then it is not further evaluated to improve speed.
2. It is a simple non-don't care code fragment, and all its leaves correspond to features in the current input state. Its terminal symbols are loaded according to the corresponding feature's value, i.e., if the feature's value lies in the corresponding interval range, it is set to 1 else set to 0.
3. It is a recursive non-don't care code fragment, and it contains one or more leaf nodes that correspond to a previous level code fragment. First, that code fragment is evaluated and its output is used as the terminal value in this code fragment. The previous level code fragment can be either a simple non-don't care code fragment or a recursive code fragment, and it is evaluated accordingly.

The condition matching process of LL-XCSRCFC is shown in Algorithm 2. Here, *cond* denotes the classifier condition and n is the length of the classifier condition.

Algorithm 2: Does_Match_Operation

Data: a classifier condition $condition$, and the current input state T
Result: returns true if the condition $condition$ satisfies the state T

```
n ← the length of condition
isMatched ← true
for i = 1 to n do
    cf ← the code fragment at condition[i]
    if cf ≠ "don't care" code fragment then
        evaluatedVal ← Evaluate_Code_Fragment(cf, T)
        if evaluatedVal ≠ 1 then
            isMatched ← false
            break
        end
    end
end
return isMatched
```

2.3 Covering operation

When any action is missing in $[M]$, the covering process is applied. Covering operation is also updated in LL-XCSRCFC. Newly created classifier condition can contain "don't care" code fragments with probability $P_{\text{don'tCare}}$. Each non-don't care code fragment can have the previous level code fragments as its leaf nodes with probability P_{kb}. A code fragment can have one or more previous level code fragments. These previous level code fragments may further contain the previous level code fragments as their leaf nodes. This is a recursive process; therefore, a classifier condition can contain code fragment from all the previous level KBs. Every non-don't care code fragment is evaluated recursively against the current input using Algorithm 1 and it must output 1. If any code fragment doesn't evaluate to 1, then it is discarded and a new random code fragment is created. Algorithm 3 describes the updated covering operation in LL-XCSRCFC. The action value of the new rule is equal to the missing action in $[M]$.

Algorithm 3: Covering_Operation

Data: state T, action act, don't care probability $P_{don'tCare}$
Result: a new classifier $clfr$ matching the current state T and having the action act.

```
initialize classifier clfr
n ← the length of classifier condition clfr.condition
for i = 1 to n do
    if random[0, 1) < P_don'tCare then
        clfr.condition[i] ← the don't care code fragment
    else
        initialize evaluatedVal ← 0
        while evaluatedVal ≠ 1 do
            cf ← a randomly created code fragment
            evaluatedVal ← Evaluate_Code_Fragment(cf, T)
        end
        clfr.condition[i] ← cf
    end
end
clfr.action ← act
return clfr
```

2.4 Mutation operation

The mutation process is one of the two important genetic operations (i.e., mutation and crossover) performed in any evolutionary technique including the proposed approach in this paper. The main purpose of the mutation operation is to maintain the genetic diversity in the population during the evolutionary learning process. Mutation operation is updated in LL-XCSRCFC to incorporate recursive code fragments. In new mutation operation, as shown in Algorithm 4, "don't care" code fragments are replaced with randomly created non-don't care code fragments and vice versa. These newly created non-don't code fragments may contain other non-don't care code fragments as their leaf nodes with probability P_{kb}. On the other hand, all non-don't code fragments of the selected classifier are replaced with "don't care" code fragments. Newly generated classifier must satisfy the current input state T, and therefore, each code fragment should output 1; else it is discarded and a new random code fragment is created. Then, the action of offsprings is also mutated with probability μ.

Algorithm 4: Mutation_Operation

Data: current input state T, a newly created classifier $clfr$, mutation probability μ
Result: the classifier $clfr$ after mutation

```
n ← the length of classifier condition clfr.condition
for i = 1 to n do
    if random[0, 1) < μ then
        if clfr.condition[i] = the don't care code fragment then
            initialize value evaluatedVal to 0
            while evaluatedVal ≠ 1 do
                cf ← a randomly created code fragment
                evaluatedVal ← Evaluate_Code_Fragment(cf, T)
            end
            clfr.condition[i] ← cf
        else
            clfr.condition[i] ← the don't care code fragment
        end
    end
end
if random[0, 1) < μ then
    act ← clfr.action
    repeat
        clfr.action ← a random action
    until clfr.action = act
end
return clfr
```

Table 2 IMDB movie reviews bottom-up hierarchical division

Data Set	Positive	Negative	Total	Number of Features
Level 1	500	500	1000	4895
Level 2	5000	5000	10,000	9573
Level 3	12,500	12,500	25,000	12,704

3 Experiment design

3.1 Problem domains

Social media text analysis data sets are used for these experiments because they can be subdivided into a bottom-up hierarchy of tasks. For these experiments, two large data sets are evaluated using LL-XCSRCFC, i.e., IMDB review (Maas et al. 2011) and Enron email data set [1]. These data sets are subdivided into a bottom-up hierarchical form. In this hierarchical subdivision, the feature set at each layer includes all the features from the previous level.

Table 2 shows division of IMDB movie review data set. It has total 25,000 records, and it is divided into 3 levels. Level 1 contains only 1000 records: 500 positive and 500 negative records, but out of these 1000 records 4895 distinct features are extracted. Then, in level 2 9000 next records are added to level 1, so it makes a total of 10,000 records with 5000 positive and 5000 negative reviews with 9573 features. The final level contains all 25,000 reviews containing 12,500 positive and 12,500 negative reviews with 12,704 features.

[1] https://www.cs.cmu.edu/~./enron/.

Table 3 Email bottom-up hierarchical division

Data Set	Ham	Spam	Total	Number of Features
Level 1	100	100	200	3684
Level 2	250	250	500	5345
Level 3	500	500	1000	6735
Level 4	1000	1000	2000	7981
Level 5	5000	5000	10,000	9946
Level 6	17,117	16,539	33,656	10,645

Table 3 shows division of Enron email data set. It has total 33,656 records, and it is divided into 6 levels. Level 1 contains only 200 records: 100 ham and 100 spam records, and 3684 distinct features are extracted. Then, in level 2, 300 next records are added to level 1, so it makes a total of 500 records with 250 ham and 250 spam emails with 5345 features. The next level contains 1000, 2000, 10,000 and 33,656 records with 3684, 5345, 6735, 7981, 9946 and 10,645 number of features, respectively. This data set is divided into a deep hierarchy for testing purpose.

3.2 Experimental setup

The commonly used parameter values in the literature, as suggested by Butz and Wilson (2002), are used: $\alpha = 0.1$ (fitness fall-off rate), $\nu = 5$ (fitness exponent), $\theta_{\text{GA}} = 25$ (threshold for GA application in the action set), $\beta = 0.2$ (learning rate), $\epsilon_0 = 10$ (prediction error threshold), $\mu = 0.04$ (mutation probability), $\chi = 0.8$ (probability of two-point crossover), $P_{\#} = 0.33$, $\theta_{\text{del}} = 20$ (threshold for classifier deletion), $\delta = 0.1$ (mean fitness for deletion), $\theta_{\text{sub}} = 20$ (threshold for subsumption), $F_{\text{I}} = 0.01$ (initial fitness), $\epsilon_{\text{I}} = 0.0$ (initial prediction error), fitnessReduction $= 0.1$ and the selection method is tournament with the size ratio 0.4. GA subsumption is activated, and the action set subsumption is not used. The value of $P_{\#}$ is set to 0.33 for don't care code fragments. For these experiments, P_{kb} is set to 0.5. The condition length and the population size (N) are gradually increased at each level in both experiments.

For the IMDB reviews data set, the value of N is set to 5000, 7000 and 10,000 for level 1, level 2 and level 3, respectively. The condition length, i.e., the number of code fragments in classifier's condition, is set to 200, 300 and 500 for level 1, 2 and 3, respectively. Similarly, the number of training examples is also increased at each level, i.e., 150, 000, 200, 000 and 300, 000 training examples are used for level 1, level 2 and level 3, respectively.

For email data set, the value of N is set to 3000 for level 1, 5000 for level 2, 6000 for level 3, 7000 for level 4, 8000 for level 5 and 10, 000 for level 6. The condition length is set to 200, 250, 300, 350, 400 and 500 for levels 1, 2, 3, 4, 5 and 6, respectively. For email data set, all the levels are tested with the same number of training examples, i.e., 125, 000, to understand the difference in learning.

The classification accuracy of the learning process is calculated for evaluation and compared with XCSRCFC. The results are calculated by applying XCSRCFC directly on each level and compared with LL-XCSRCFC which reused the knowledge retained from the previous tasks.

4 Results

The behavior of lifelong reinforcement learning model in social media text classification domain was studied, and the change in learning pattern was analyzed. The results of LL-XCSRCFC were compared with XCSRCFC at each level. The results showed the supremacy of lifelong reinforcement learning paradigm over standard reinforcement learning model.

The performance of XCSRCFC and LL-XCSRCFC on different hierarchical levels on email data set was compared as shown in Figures 3, 4, 5, 6 and 7. The email data set was divided into 6 levels, and LL-XCSRCFC showed its supremacy at all levels. At level 2, there were only 500 examples and the performance of both XCSRCFC and LL-XCSRCFC was very good, but LL-XCSRCFC learned very fast as shown in Fig. 3. At level 3, LL-XCSRCFC used code fragments stored in $KB2$ and $KB1$. Similarly, at each next level code fragments from all the previous levels stored in the respective knowledge base were used in learning a new task. The complete data set was analyzed in level 6, where XCSRCFC gave 86% accuracy, while LL-XCSRCFC reached up to 90% accuracy as shown in Fig. 7. It showed that lifelong learning model improved the learning process and reusing previously extracted knowledge gave better results.

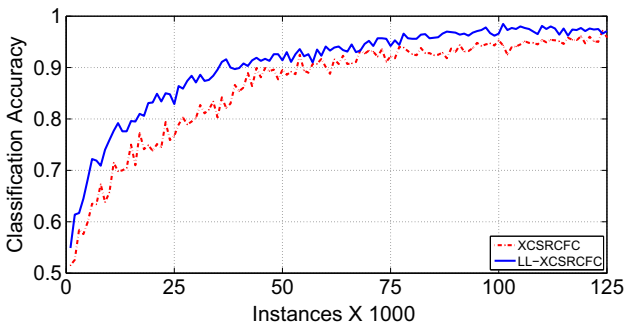


Fig. 3 Spam detection in email at level 2

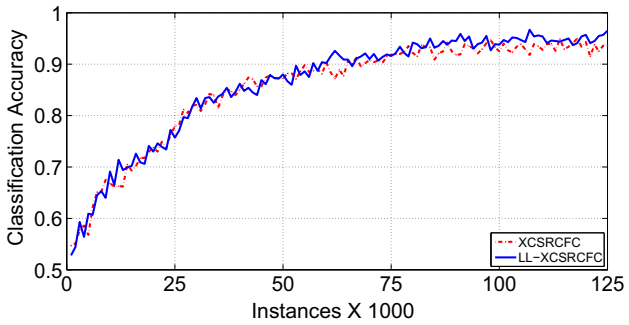


Fig. 4 Spam detection in email at level 3

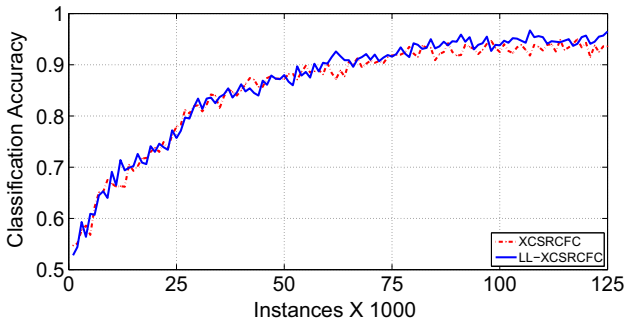


Fig. 5 Spam detection in email at level 4

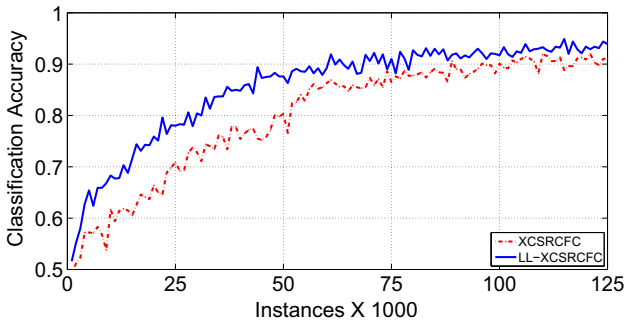


Fig. 6 Spam detection in email at level 5

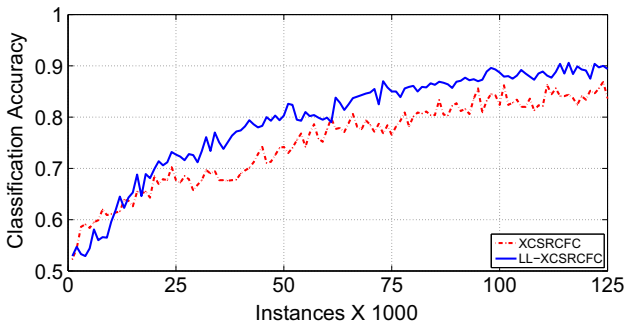
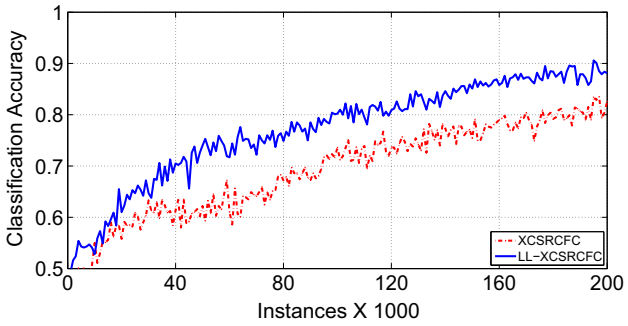
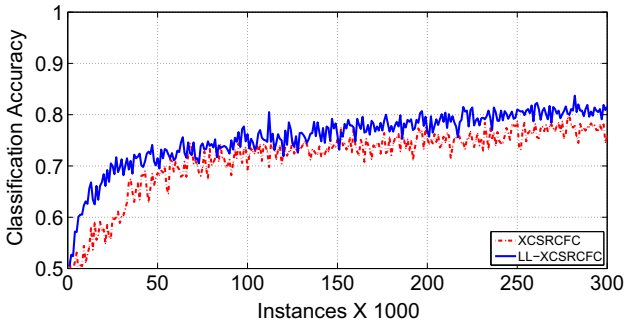


Fig. 7 Spam detection in email at level 6

The performance of XCSRCFC and LL-XCSRCFC on IMDB movie reviews data sets at level 2 and level 3 is shown in Fig. 8. The performance of LL-XCSRCFC showed



(a) IMDB (Level 2)



(b) IMDB (Level 3)

Fig. 8 Sentiment analysis of IMDB reviews at **a** level 2, and **b** level 3

a clear improvement on level 2 with 10,000 records and it reused the knowledge learned from level 1 problems as shown in Fig. 8a. The maximum performance of XCSRCFC reached 83%, while LL-XCSRCFC achieved 90% classification accuracy. While on level 3 the performance of both systems went down, but the performance of LL-XCSRCFC was still better than XCSRCFC. The maximum performance of XCSRCFC reached up to 78%, while the maximum performance of LL-XCSRCFC was reached to 82%. The results showed that lifelong reinforcement learning model improved the learning paradigm, and it showed statistically significant improvement over stand-alone learning.

5 Discussion

In XCSR, there is a direct linkage between a specific input feature and the corresponding interval in the classifier condition. Therefore, XCSR has to learn a specific interval for every input feature. In the case of social media text data sets which are very sparse and very high dimensional, this direct linkage results in the creation of very specific rules. However, in code fragment-based XCSRCFC system there is no such direct linkage between a specific code fragment in classifier condition and a specific feature in the input state. Therefore, due to this disassociation, condition matching process becomes more flexible and it produces optimal classifiers as they can be generated in multiple ways.

The classifier condition length is not fixed as in standard XCSR which has a specific interval for every input feature. In the case of XCSRCFC, condition length can be different from the number of input features. This gives additional flexibility to XCSRCFC system, and system can decide that how many non-don't care and "don't care" code fragments should be used which increases the chances to produce optimal rules. XCSRCFC takes longer training time than XCSR due to a general rule set. Training time can be managed by reducing the size of classifier condition.

Code fragments store useful knowledge, and they can act as building units of knowledge that can retain the learnt knowledge which can be reused. LML model is best suited to those tasks that can be subdivided into a bottom-up hierarchy of tasks. Experiments show that a social media text classification task can be divided into a sequence of tasks that contain a bottom-up hierarchical feature set.

In the proposed LL-XCSRCFC system, the population of the rules evolved by the system at any problem level stores important information that can be reused. A KM system is devised in which distinct non-don't care code fragments from fitter rules are extracted and reused by the KBL to solve higher level problems in the hierarchy. It shows that reinforcement learning systems learn small problems easily and code fragments retain the knowledge learned during the procedure. Experimental results show the supremacy of LL-XCSRCFC over simple XCSRCFC.

Although new rules and new code fragments are added to the system using mutation operation, in LL-XCSRCFC, GA operations of mutation and crossover are not performed at code fragment level. The performance and scalability of LL-XCSRCFC may improve if code fragments are evolved in the training process.

The main strength of the proposed approach is to extract knowledge, in the form of code fragments, in learning smaller tasks, which can be reused to learn the more complex tasks in the domain. However, this approach is best suited to those tasks that can be subdivided into a bottom-up hierarchy of sub-tasks. Further, the proposed system may hit a limit on the number of hierarchical problem levels because the depth of nested code fragments increases with every next level, which results in an increased search space and eventually demands more computational time.

6 Conclusion and future work

In this paper, a LML model for text classification is proposed. It reuses the stored knowledge extracted from the previous problems for the domain to solve high-dimensional text classification problems. Text classification problems can

be divided into a bottom-up hierarchy of tasks, and the rules learned at each level can act as PIS for the next levels. Code fragments extracted from the experienced rules can act as building blocks of knowledge and can be reused to learn higher level problems. These code fragments act as knowledge base KB for higher level problems. Each level can reuse the code fragments from all the previous levels. Results show the supremacy of lifelong reinforcement learning paradigm over standard reinforcement learning paradigm.

In the current work, code fragments are matched syntactically. but as a result, subsumption deletion process is almost impossible. In future, a mechanism needs to be designed to compare code fragments semantically which will enable the subsumption deletion process in its true sense and may result in reducing the size of the final population.

In the current implementation of LL-XCSRCFC, static code fragments are used as KB to create higher level code fragments. These code fragments are used as leaf nodes in higher level code fragments. This hierarchical approach may result in a limit on the number of hierarchical problem levels. Some other approaches may be needed to find better reuse of code fragments at the higher level.

The developed models are tested on sentiment analysis and spam detection data sets. A further investigation in other text classification problems with different feature vectors can be conducted in future. The current systems are developed and tested for two class problems, and it should be further investigated and extended for multi-class problems.

Acknowledgements This work is supported by NSFC program (Nos. 61472022, 61421003), SKLSDE-2016ZX-11 and partly by the Beijing Advanced Innovation Center for Big Data and Brain Computing.

Compliance with ethical standard

Conflict of interest The authors declare that they have no conflict of interest.

Ethical approval This article does not contain any studies with human participants or animals performed by any of the authors.

References

Arif MH, Jin X, Li J, Iqbal M (2017a) Text classification using lifelong machine learning. In: Proceedings of the 24th international conference on neural information processing (ICONIP), Springer International Publishing, pp 394–404

Arif MH, Li J, Iqbal M (2017b) Solving social media text classification problems using code fragment based XCSR. In: Proceedings of the international conference on tools with artificial intelligence (ICTAI), Boston, MA, USA, pp 485–492

Arif MH, Li J, Iqbal M, Peng H (2017c) Optimizing XCSR for text classification. In: Proceedings of the IEEE symposium on service-oriented system engineering (SOSE), San Francisco, CA, USA, pp 86–95

Arif MH, Li J, Iqbal M, Liu K (2018) Sentiment analysis and spam detection using learning classifier systems. Soft Comput 22:7281–7291. https://doi.org/10.1007/s00500-017-2729-x

Butz MV, Wilson SW (2002) An algorithmic description of XCS. Soft Comput 6(3–4):144–153

Chen Z, Liu B (2016) Lifelong machine learning. Morgan and Claypool, San Rafael

Chen Z, Ma N, Liu B (2015) Lifelong learning for sentiment classification. In: Proceedings of 53rd annual meeting of the association for computational linguistics and the 7th international joint conference on natural language processing of the Asian Federation of Natural Language Processing (ACL-IJCNLP 2015), vol 2. Association for Computational Linguistics (ACL), pp 750–756

Iqbal M, Browne WN, Zhang M (2014) Reusing building blocks of extracted knowledge to solve complex, large-scale Boolean problems. IEEE Trans Evolut Comput 18(4):465–480

Iqbal M, Browne WN, Zhang M (2015) Improving genetic search in XCS-based classifier systems through understanding the evolvability of classifier rules. Soft Comput 19(7):1863–1880

Maas AL, Daly RE, Pham PT, Huang D, Ng AY, Potts C (2011) Learning word vectors for sentiment analysis. In: Proceedings of the 49th annual meeting of the association for computational linguistics: human language technologies, Portland, Oregon, USA, pp 142–150

Shu L, Xu H, Liu B (2017) Lifelong learning CRF for supervised aspect extraction. In: Accepted at ACL 2017, Vancouver, Canada. Association for Computational Linguistics

Silver DL, Yang Q, Li L (2013) Lifelong machine learning systems: beyond learning algorithms. In: AAAI spring symposium series, California, USA, pp 49–55

Sutton RS, Koop A, Silver D (2007) On the role of tracking in stationary environments. In: In Proceedings of the 24th international conference on machine learning, (ICML '07), New York, NY, USA. ACM, pp 871–878

Urbanowicz RJ, Moore JH (2009) Learning classifier systems: a complete introduction, review, and roadmap. J Artif Evol Appl 2009:1–25

Wilson SW (1995) Classifier fitness based on accuracy. Evolut Comput 3:149–175

Wilson SW (2000) Get real! XCS with continuous-valued inputs. In: Lanzi PL, Stolzmann W, Wilson SW (eds) Learning classifier systems. Springer, Berlin, pp 209–219