A Novel RVFL-Based Algorithm Selection Approach for Software Model Checking

Weipeng Cao[1,2], Yuhao Wu[2], Qiang Wang[3](✉), Jiyong Zhang[4], Xingjian Zhang[1], and Meikang Qiu[5]

[1] CAAC Key Laboratory of Civil Aviation Wide Surveillance and Safety Operation Management and Control Technology, Civil Aviation University of China, Tianjin, China

[2] College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, China
caoweipeng@szu.edu.cn

[3] Institute of System Engineering, Chinese Academy of Military Science, Beijing, China
18513688908@163.com

[4] School of Automation, Hangzhou Dianzi University, Hangzhou, China
jzhang@hdu.edu.cn

[5] Department of Computer Science, Texas A&M University-Commerce, Commerce, TX, USA

Abstract. Software model checking is the technique that automatically verifies whether software meets the given correctness properties. In the past decades, a large number of model checking techniques and tools have been developed, reaching a point where modern model checkers are sophisticated enough to handle large-scale software systems. However, due to the fact that the software model checkering techniques are diverse and each of them is designed and optimized for a specific type of software system, it remains a hard problem for engineers to efficiently combine them to verify the complex software systems in practice. To alleviate this problem, we propose a novel algorithm selection approach based on Random Vector Functional Link net (RVFL) for software model checking, namely Kaleidoscopic RVFL (K-RVFL). The novel design of feature hybridization and fusion enables K-RVFL to extract more diverse and multi-level features. We have also carried out a thorough experimental evaluation on a publicly available data set and compared K-RVFL with a number of neural networks, including RVFL, Extreme Learning Machine (ELM), Stochastic Configuration Network (SCN), Back Propagation algorithm (BP), and Supporting Vector Machine (SVM). The experimental results demonstrate the usefulness and effectiveness of K-RVFL.

Keywords: Software verification · Software model checking · Algorithm selection · Random vector functional link · Neural networks

G. Memmi et al. (Eds.): KSEM 2022, LNAI 13370, pp. 414–425, 2022.
https://doi.org/10.1007/978-3-031-10989-8_33

1 Introduction

Software model checking is a prevailing technique that aims at automatically verifying whether a software satisfies the given correctness properties [6]. In the past decades, a large amount of model checking techniques have been developed and successfully applied to many fields. However, given a number of software model checking tools, the engineers are still facing the problem of 'which is the most suitable one for verifying my software?'. This question arises since the underlying model checking techniques are diverse and there is no single solution that works well for all kinds of software systems. Each model checking technique has its individual characteristics and thus strengthens on a specific type of software. Moreover, there has been a growing awareness of the need to understand how each model checking tool performs and which one is most suitable for a specific type of software, in order to promote software model checking to a larger industrial scope [2,7,13].

Previously the machine learning based algorithm prediction problem has been investigated in some related work [9,12,13]. Instead of selecting the most suitable software model checking tool, their work aims to predict the ranking in terms of the verification performance. All the prediction models are constructed by using Supporting Vector Machine (SVM). In our previous work [14], we view the above problem as an instance of the algorithm selection problem and propose to build the selection model by using neural network techniques [4]. To boost the capability of neural networks, we also define a set of software features that are precise enough to represent the key structural characteristics of the software on the source code level, such as the variable role usage, control flow metrics and loop patterns. Compared with SVM, neural network techniques have several advantages. For example, neural networks are more efficient in handling large data sets than SVM, due to the fact that SVM suffers from the difficulty of parallelizing the learning process.

In this work, we go one step further and propose a novel Kaleidoscopic RVFL algorithm (K-RVFL) with a feature hybridization and fusion mechanism. The network structure of K-RVFL still maintains the direct links from the input layer to the output layer, such that the model's feature extraction ability is enhanced and a regularization for the randomization is provided. Different from RVFL, K-RVFL uses multiple types of bounded non-linear functions such as *Sigmoid* and *Sin* function as its activation function. The input data first undergoes the non-linear mapping through each activation function, and then the output matrix corresponding to each activation function is used as the input of other activation functions for secondary mapping. This process of feature hybridization provides the capability of extracting much more diverse features from the training data. Then, in the feature fusion step, we linearly fuse multi-level features (including the original ones) to get the final feature matrix. The output weights are computed by using the least square method. Last but not the least, K-RVFL maintains the non-iterative training mechanism of RVFL, which enables the ability of fast learning.

To this end, the following contributions have been made in this work.

(1) We present a general formalization of the algorithm selection problem for software model checking. Our work shows that the neural network techniques, in particular, the randomized learning algorithms can solve this problem efficiently and achieve state-of-the-art results. The proposed approach can be extraordinarily helpful for applying software model checking techniques to complex industrial software systems.
(2) We have proposed a novel RVFL algorithm named K-RVFL. Multiple types of activation functions, two-time non-linear mappings, and the fusion of multi-level features can bring rich features to K-RVFL, which enables the model to produce more accurate decisions. It is worth mentioning that the idea of K-RVFL can also be applied to other randomized algorithms such as ELM and SCN.
(3) We have carried out a thorough experimental evaluation, which shows that our proposed K-RVFL algorithm can achieve higher prediction accuracy than ELM, RVFL, SCN, BP, and SVM algorithms. Moreover, compared with BP and SVM algorithms, the training speed of K-RVFL is increased by more than 100 times and 300 times, respectively.

2 Related Works

The work in [13] proposes a SVM based technique to construct a strategy selector for the verification of different programs. The selector takes as input a set of program features and outputs a strategy for verification. A strategy defines the algorithm or parameter that can be useful for verifying the given program. The primary aim of the work in [8,9] is to empirically evaluate and explain the performance differences of the various model checkers in the annual software verification competition (SV-COMP). A portfolio solver based on support vector machine has also been proposed, which essentially chooses the most performant tool for a given model checking task based on the evaluations. In their work, they show that the overall performance of the portfolio solver outperforms all the other tools, which in our opinion demonstrates a strong viability and usefulness of algorithm selection for software verification The work in [7] studies the ranking prediction problem for software model checking. A ranking of the candidate model checkers can indicate which is the most suitable one for the give software at hand. However, their prediction model is also based on SVM.

The authors in [2] present a specific strategy selector for the model checker CPAChecker[1]. However, the selector only works for CAPChecker, because the strategies are merely different parameter specifications or model checking algorithm configurations integrated in CPAChecker. Moreover, the selection model is explicitly defined and implemented in CPAChecker. No machine learning techniques are applied. The experimental evaluation shows that the performance of their strategy selector is much better than any single algorithm configuration of CPAChecker. Similarly in [12], a technique called PeSCo has been proposed

[1] https://cpachecker.sosy-lab.org/.

to predict the likely best sequential combination of algorithm configurations in CPAChecker. The approach is also based on support vector machine.

In our previous work [14], we have proposed for the first time to employ the randomized learning algorithm to solve the algorithm selection problem for software verification. Later in [15] a novel algorithm based on the long-short term memory network (LSTM) has been proposed, which uses word2vec to obtain a representation of the software. For more preliminaries, please refer to [14,15].

3 Algorithm Selection Based on Neural Network with Random Weights

3.1 Extracting Features for Software Model Checking Tasks

In order to solve the algorithm selection problem using the framework of machine learning, we need to represent the software model checking tasks. For this purpose, we define a set of features and the corresponding metrics that can precisely characterize the software on the source code level, leveraging on the related works [8–10].

The first set of features are based on variable roles. Basically, a variable role indicates the usage pattern of the variable in the source code. The choice of roles is inspired by the standard concepts in programming, such as counter, bit-vector and file descriptor etc. In total, we have defined 27 variable roles. For a given software source code f and the set of variable roles $Roles$, we compute a mapping $Res : Roles \rightarrow Vars$ from variable roles to the program variables. The variable role metric m_R that represents the relative occurrence of each variable role $R \in Roles$ is defined as $m_R = |Res(R)|/|Vars|$, where $|Vars|$ represents the total number of variables.

The second set of features are based on loop patterns. In total, we define four different loop patterns including syntactically bounded loops, syntactically terminating loops, simple loops, and hard loops. The corresponding metric is called loop pattern based metric. For a given software source code f, we compute the set of syntactically bounded loops L^{SB}, the set of syntactically terminating loops L^{ST}, the set of simple loops L^{simple}, the set of hard loops L^{hard}. The loop pattern based metrics m_{lp} represents the relative occurrence of each loop pattern $lp \in \{ST, SB, simple, hard\}$ and it is computed as $m_{lp} = |L^{lp}|/|Loops|$, where $|Loops|$ represents the set of all loops.

Both variable role based metrics and loop pattern based metrics can be efficiently computed from the software source code by using static analysis techniques [11]. We omit the elaboration since it is out of the scope of this paper.

We denote a software model checking task by $v = (f, p, type)$, where f, p, and $type$ are the software source file, property, and the property type, respectively. We use $Tasks$ to represent the set of software model checking tasks. Each task serves as a sample of the data set for model training. The corresponding feature vector for a task v is defined by $\mathbf{x}(v) = (\mathbf{m}_R, \mathbf{m}_{lp}, type)$, where $type \in \{0, 1, 2, 3\}$ encodes the verification property type, i.e., reachability, memory safety, overflow and termination.

3.2 Formalizing the Algorithm Selection Problem

In this subsection we present how to encode the algorithm selection problem as a multi-classification problem. For each task $v = (f, p, type) \in Tasks$, the function $ExpAns : Tasks \rightarrow \{true, false\}$ defines the expected answer of whether software f satisfies property p. In other words, this function defines the ground truth for each task, which is regardless of the model checking tool being used.

Given a model checking tool $t \in Tools$ and a task $v = (f, p, type) \in Tasks$, applying tool t to verify task v in limited time and resources would produce an answer $PracAns(t, v) = (ans_{t,v}, time_{t,v})$, where $ans_{t,v} \in \{true, false, unknown\}$, and $time_{t,v}$ is the amount of computing time. We remark that the answer $ans_{t,v}$ could be *unknown*, meaning that the tool t is unable to check whether the property holds or not. due to the fact that the software model checking problem generally is undecidable [6].

In neural networks based machine learning techniques, we need labeled data for model training. In our case, the label of a task is the (likely) best tool that is able to verify this task correctly. We denote by $L : Tasks \rightarrow Tools$ the labeling function. Given a task $v \in Tasks$ and a tool $t \in Tools$, we set $L(v) = t$ if the following two conditions are satisfied:

1 the tool t provides the correct answer on v, i.e., $ans_{t,v} = ExpAns(v) \wedge ans_{t,v} \neq unknown$;
2 the tool t costs the least time among $Tools$ that can provide the correct answer, i.e., $\forall t' \in \{t' \mid t' \neq t \wedge (ans_{t',v} = ExpAns(v)) \wedge (ans_{t',v} \neq unknown)\}$, $time_{t',v} > time_{t,v}$.

Finally, the algorithm selection problem studied in this work can be formalized as follows. Given a set of software model checking tasks $Tasks$ and a set of model checking tools $Tools$, the algorithm selection problem for software model checking is to find a selection model $M : Tasks \rightarrow Tools$, such that $M(v)$ gives the best possible tool for solving v, i.e., $M(v) = L(v)$.

3.3 Our Proposed Kaleidoscopic RVFL Algorithm

In classical RVFL, only one type of activation function is used to perform the non-linear feature mapping, however, in K-RVFL three different non-linear functions are defined and used as the activation functions. In this work, we have chosen the Sigmoid function, Sin function, and Triangular basis transfer function as the three activation functions. Note that any arbitrary bounded non-linear function can be chosen as the activation function in K-RVFL.

The network structure of the K-RVFL algorithm is shown in Fig. 1, where X, O, d, m, ω, b, and β refer to the input of the model, the output of the model, the node number of the input layer, the node number of the output layer, input weights, hidden biases, and output weights, respectively. K-RVFL also has only one hidden layer and its input layer and output layer are directly connected. The difference between the RVFL and K-RVFL is that the single hidden layer of the RVFL only makes one-time non-linear feature mapping based on a single type of

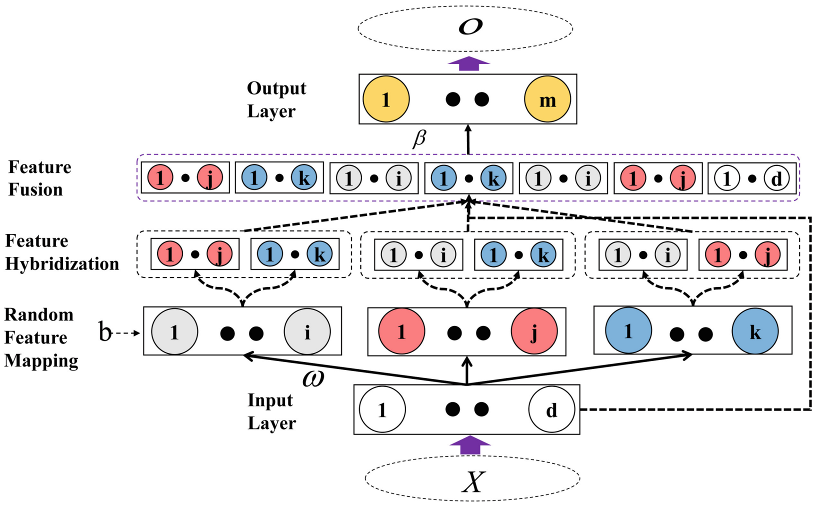

Fig. 1. The K-RVFL network structure.

activation function, while the hidden layer of the K-RVFL consists of multiple types of activation functions and makes two-time non-linear feature mapping, so that the K-RVFL can extract more diverse and multi-level features from the input data.

Specifically, In K-RVFL the input data first undergoes the non-linear mapping of each specific activation function (i.e., the random feature mapping layer). Then the corresponding output matrix will be used as the input of the other two activation functions for feature mapping for the second time. For example, if the input matrix first passes through the non-linear mapping of the Sigmoid function in the hidden layer, it will then be mapped by the Sin function and the Trigonometric basis function in the feature hybrid stage. This step is called the feature hybridization. After this two-stage nonlinear mapping, it is the feature fusion step, where we fuse the extracted features with the original features to get the final feature matrix of the hidden layer. Finally, the output weights are obtained by solving a system of linear matrix equations.

K-RVFL algorithm is summarized and depicted in Algorithm 1. For machine learning algorithms, the diversity of data features is very useful for the correct decision of the model. For example, one of the most important reasons for the success of deep learning is that it can achieve the features hybridization through the layer by layer processing of multiple hidden layers and then obtain the diverse and multi-level features. The basic idea behind the K-RVFL is to improve the diversity of features through multiple types of activation functions and multiple non-linear mappings without adding too many hidden layers. The advantage of this method is that the shallow network structure can make the K-RVFL maintain fast training speed and the feature hybridization and fusion strategy can make the model obtain better feature information. Fast and high accuracy

```
Algorithm 1: The proposed K-RVFL algorithm
Input: A training data set $X$, three types of activation functions (denoted as
    $G1(\cdot)$, $G2(\cdot)$, and $G3(\cdot)$), and the number of the hidden layer nodes using each
    type of activation function (denoted as $i$, $j$, and $k$).
Output: Output weights $\beta$.
1: Initialization: set the number of the input layer nodes and the output layer
   nodes equal to the number of the features and classes of the input data,
   respectively;
2: Random feature mapping stage:
     – Randomly generate the input weights $\omega$ from the range (-1, 1) under the
       uniform distribution and the hidden biases $b$ from the range (0, 1) under
       the uniform distribution.
     – Use the randomly generated $\omega$ and $b$ to linearly map the input data:
       $H0 = \omega X + b$
     – Non-linear mapping of $H0$ with three activation functions: (1)
       H1_G1_FirstTime = G1(H0), (2) H1_G2_FirstTime = G2(H0), and (3)
       H1_G3_FirstTime = G3(H0)
3: Feature hybridization stage: The nonlinear mapping matrix corresponding to
   each activation function is used as the input of the other two activation
   functions for the second-time nonlinear feature mapping.
     – For $G1$'s output matrix:
         $H2\_G1\_SecondTime = [G2(H1\_G1\_FirstTime), G3(H1\_G1\_FirstTime)].$
     – For $G2$'s output matrix:
         $H2\_G2\_SecondTime = [G1(H1\_G2\_FirstTime), G3(H1\_G2\_FirstTime)].$
     – For $G3$'s output matrix:
         $H2\_G3\_SecondTime = [G1(H1\_G3\_FirstTime), G2(H1\_G3\_FirstTime)].$
4: Feature fusion stage: Linearly fuse the non-linear mapping feature matrix with
   the original feature matrix.
       $\mathbf{H3} = [H2\_G1\_SecondTime, H2\_G2\_SecondTime, H2\_G3\_SecondTime, X].$
5: Solve the output weights: $\beta = \mathbf{H3}^{+}T$, where $T$ and $\mathbf{H3}^{+}$ refer to the samples'
   real labels and $\mathbf{H3}$'Moore-Penrose generalized inverse, respectively.
```

are both very important for the application of the software model checking tools recommendation system in practical engineering.

4 Experimental Evaluation

4.1 Preparation of the Data-Set

We collect the raw data for the verification tasks in the annual competition on software verification [1], in order to compare with the previous results in the related work [9]. We remark that each year the competition tasks are mostly the same, with only minor additions and changes in the category structure.

We extract the features of the tasks and compute the vector representations. Each dimension of the vector corresponds to a specific feature introduced in Sect. 3. In total, the data set has 31371 samples. For each sample, there are 46 attributes. Then for each task, we add a label to its vector representation to indicate the most suitable tool according to the competition results. We artificially create 3 classes for the classification.

In our experiments, the training set and testing set are divided according to 8:2. Similarly, we divide the original training set into pure training set and corresponding validation set according to 8:2 to select the best model for each method.

4.2 Parameters Settings

In our experiment, we choose the most commonly used randomization strategy for ELM and RVFL. That is, for their input weights, we generate them from (-1, 1) randomly and keep them unchanged throughout the subsequent training process. For the input weights of SCN, we generate them according to a supervisory strategy [5]. We set the number of hidden layer nodes in all comparison algorithms to the same and choose *Sigmoid* function as their activation function.

4.3 Experimental Results

In the experimental evaluations, we compare the accuracy and learning time of all the six relevant algorithms on the above dataset, including ELM, RVFL, SCN, BP, SVM, and K-RVFL. The number of hidden layer nodes in these algorithms is selected from {50, 100, 150, 200, 250, 300, 350, 400, 450, 500} one by one. The experiments are conducted with MATLAB R2016b software. Each experiment results are the average of 50 independently experiments. Figures 2–3 and Table 1 show our experimental results.

In Fig. 2, we compare the testing performance of five related algorithms with our proposed K-RVFL algorithm. According to these experimental results, one can infer that ELM, RVFL, SCN, and K-RVFL models can get better performance with the number increase of the hidden nodes. However, the performance of the BP model fluctuates greatly due to the difficulty of setting best values for its hyper-parameters and the instability of the gradient descent method. Therefore, compared with the traditional neural network BP algorithm, NNRW based algorithms always enjoy a higher accuracy and better stability. Compared with the SVM algorithm, all NNRW based algorithms except ELM can achieve better

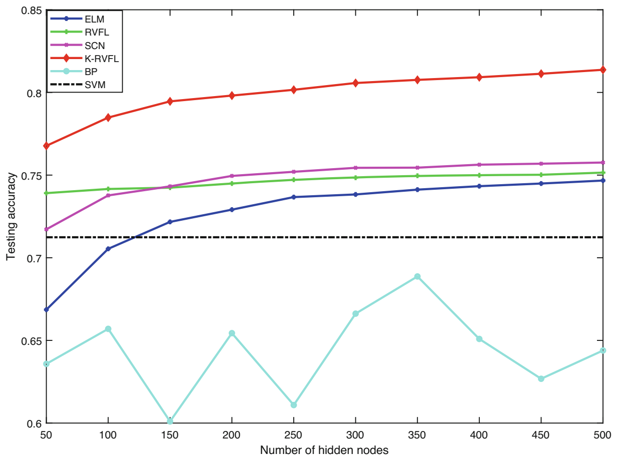

Fig. 2. Comparison of testing accuracy between the relevant algorithms.

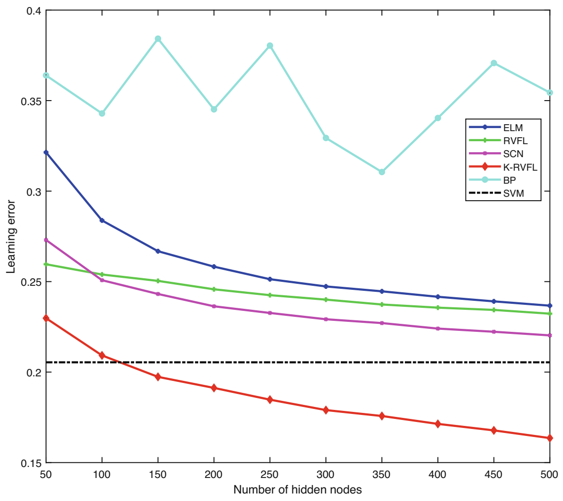

Fig. 3. Comparison of learning error between the relevant algorithms.

generalization performance, and they all have higher prediction accuracy when the hidden nodes are over 150. For the original NNRW algorithms (i.e., RVFL, ELM, and SCN), we can see that RVFL and SCN have better prediction performance than ELM regardless of the number of hidden layers. This is due to the special architecture design of RVFL and the supervised random method of SCN, which play a positive role in the model training. However, when the number of hidden layers increases, the differences between their performances decrease to a comparable level. Compared with the above algorithms, our proposed K-RVFL achieves the highest prediction accuracy in all cases. The margin shows that the improvement is substantial. It is worth mentioning that this is also the current SOTA result in the model checking algorithm selection task. Previous SOTA results are based on the SVM algorithm [9].

Figure 3 shows the learning error changing curves of ELM, RVFL, SCN, BP, SVM, and our proposed algorithm K-RVFL. In general, one can observe that the learning errors of the NNRW based models will gradually decrease with the increase of the hidden nodes, and they all have lower learning errors than the BP algorithm. Moreover, we can find that our proposed K-RVFL model Moreover, e can find that our proposed K-RVFL algorithm can achieve faster error reduction than other NNRW based algorithms (i.e., ELM, RVFL, and SCN) in the training process. This phenomenon implies that K-RVFL can approach the lower bound of error at the fastest speed. This also implies that compared with other algorithms, the proposed K-RVFL has faster convergence speed. And under the same network complexity, K-RVFL is expected to have better prediction performance than other models.

Table 1. Comparison of training time between ELM, RVFL, SCN, BP, SVM, and K-RVFL. The best results are in bold.

Hidden nodes	ELM	RVFL	SCN	KRVFL	BP	SVM
50	**0.1962**	0.4056	29.1138	0.8218	74.7245	5854.6707
100	**0.5014**	0.6767	73.1227	1.7042	114.2707	*
150	**0.7441**	1.0349	180.7078	2.9453	155.1274	*
200	**1.0371**	1.5728	278.0518	4.3165	185.7504	*
250	**1.5382**	1.8062	389.2362	5.6323	229.2435	*
300	**1.7697**	2.2901	432.3528	8.0344	267.0893	*
350	**2.2885**	2.8411	551.4656	10.4330	314.0300	*
400	**2.7827**	3.3347	719.3612	12.2311	334.8561	*
450	**3.2261**	4.1006	970.5653	14.3265	357.9287	*
500	**3.8875**	4.6264	1170.5231	17.1579	383.6377	*

In Table 1, we compare the training time of the proposed K-RVFL algorithm with other algorithms. From the experimental results, we can conclude that compared with the traditional neural network BP and SVM, the NNRW based algorithms have absolute advantages in the training speed of the model. Taking the proposed K-RVFL algorithm as an example, its model training speed is more than 100 times faster than BP and more than 300 times faster than SVM.

One can also find that the ELM algorithm can achieve faster learning time than other algorithms. The main reason is that compared with ELM, RVFL and K-RVFL have a relatively complex network structure, which may result in higher computational complexity. For SCN, it uses a supervised mechanism to initialize its input weights, so its training time is also longer than ELM.

We believe that the design of feature hybridization and fusion is a useful means, which can enable K-RVFL to extract more diverse and multi-level features and thus provide much better performance. This advantage also makes it great potential in time-critical applications [3].

5 Conclusion and Future Work

In this work, we propose to use neural network techniques to solve the algorithm selection problem for software model checking. We also propose an improved RVFL algorithm named K-RVFL to train the selection model. K-RVFL uses multiple types of activation functions to improve the diversity of features and uses feature hybridization to extract multi-level features, which have a positive impact on the correct decision-making of the model. K-RVFL inherits the non-iterative training mechanism of the RVFL and maintains the advantage of extremely fast training speed. Moreover, we conduct extensive experiments which demonstrate the effectiveness of neural network techniques for this problem. Our results show that K-RVFL has obvious advantages over the existing state-of-the-art algorithm (i.e., SVM) in both the prediction accuracy (81.37% vs 71.24%) and the training speed (17.16 s vs 5854.67 s). K-RVFL also outperforms the other randomized learning algorithms, including ELM, RVFL, and SCN and traditional neural network BP.

Acknowledgements. This work was supported by National Natural Science Foundation of China (Grant No. 62106150), CAAC Key Laboratory of Civil Aviation Wide Surveillance and Safety Operation Management and Control Technology (Grant No. 202102), and CCF-NSFOCUS (Grant No. 2021001).

References

1. Beyer, D.: Reliable and reproducible competition results with BenchExec and witnesses (report on SV-COMP 2016). In: Chechik, M., Raskin, J.-F. (eds.) TACAS 2016. LNCS, vol. 9636, pp. 887–904. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49674-9_55
2. Beyer, D., Dangl, M.: Strategy selection for software verification based on Boolean features. In: Margaria, T., Steffen, B. (eds.) ISoLA 2018. LNCS, vol. 11245, pp. 144–159. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03421-4_11
3. Cao, W., Gao, J., Ming, Z., Cai, S., Shan, Z.: Fuzziness-based online sequential extreme learning machine for classification problems. Soft Comput. **22**(11), 3487–3494 (2018). https://doi.org/10.1007/s00500-018-3021-4
4. Cao, W., Wang, X.-Z., Ming, Z., Gao, J.: A review on neural networks with random weights. Neurocomputing **275**, 09 (2017)
5. Cao, W., Xie, Z., Li, J., Xu, Z., Ming, Z., Wang, X.: Bidirectional stochastic configuration network for regression problems. Neural Netw. **140**, 237–246 (2021)
6. Clarke, E.M., Henzinger, T.A., Veith, H., Bloem, R.: Handbook of Model Checking, vol. 10. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-10575-8

7. Czech, M., Hüllermeier, E., Jakobs, M.-C., Wehrheim, H.: Predicting rankings of software verification tools. In: Proceedings of the 3rd ACM SIGSOFT International Workshop on Software Analytics, SWAN 2017, pp. 23–26 (2017)
8. Demyanova, Y., Pani, T., Veith, H., Zuleger, F.: Empirical software metrics for benchmarking of verification tools. In: Kroening, D., Păsăreanu, C.S. (eds.) CAV 2015. LNCS, vol. 9206, pp. 561–579. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21690-4_39
9. Demyanova, Y., Pani, T., Veith, H., Zuleger, F.: Empirical software metrics for benchmarking of verification tools. Formal Methods Syst. Des. **11**, 289–316 (2017). https://doi.org/10.1007/s10703-016-0264-5
10. Demyanova, Y., Veith, H., Zuleger, F.: On the concept of variable roles and its use in software analysis. In: FMCAD, pp. 226–229 (2013)
11. Nielson, F., Nielson, H.R., Hankin, C.: Principles of Program Analysis. Springer, Cham (2015). https://doi.org/10.1007/978-3-662-03811-
12. Richter, C., Wehrheim, H.: PeSCo: predicting sequential combinations of verifiers. In: Beyer, D., Huisman, M., Kordon, F., Steffen, B. (eds.) TACAS 2019. LNCS, vol. 11429, pp. 229–233. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17502-3_19
13. Tulsian, V., Kanade, A., Kumar, R., Lal, A., Nori, A.V.: MUX: algorithm selection for software model checkers. In Proceedings of the 11th Working Conference on Mining Software Repositories, MSR 2014, pp. 132–141 (2014)
14. Wang, Q., Cao, W., Jiang, J., Zhao, Y., Ming, Z.: NNRW-based algorithm selection for software model checking. In: International Conference on Extreme Learning Machine (ELM) (2019)
15. Wang, Q., Jiang, J., Zhao, Y., Cao, W., Wang, C., Li, S.: Algorithm selection for software verification based on adversarial LSTM. In: 2021 7th IEEE International Conference on Big Data Security on Cloud (BigDataSecurity), High Performance and Smart Computing, (HPSC) and Intelligent Data and Security (IDS), pp. 87–92 (2021)