ICIC Express Letters
Part B: Applications
Volume **13**, Number **1**, January **2022**

 ISSN 2185-2766
pp. 1–9

FEATURES ENGINEERING WITH GENERATING GENETIC ALGORITHM FOR SOFTWARE EFFORT ESTIMATION

Lindung Parningotan Manik[1,2]

[1]Research Center for Informatics
National Research and Innovation Agency
Jl. Sangkuriang/Cisitu No. 21/154D, Bandung 40135, Indonesia

[2]Faculty of Information Technology
University of Nusa Mandiri
Jl. Kramat Raya No. 18, Jakarta 10450, Indonesia
lind004@lipi.go.id; lindung.lpm@nusamandiri.ac.id

Received March 2021; accepted June 2021

Abstract. *In software engineering, effort estimation is the process of predicting the most realistic amount of effort, which could be expressed in terms of person-hours or person-months, required to develop software. The software effort estimation is a difficult task since different factors could have different effects on the efforts in the various environments. Thus, mining the project repositories based on historical data in an efficient manner is a critical issue to make an accurate estimation. Machine learning algorithms are used in this case as the solutions to predict the effort by considering the software development factors and environments as the machine learning features. Over the years, most of the research works in this field dealt with the attributes selection and features weighting to improve the estimations accuracy. This paper investigates generating genetic algorithm as the feature engineering approach for the software development effort estimation. This method may select some features, and it also may create new features from the original set of features. The experiment results show that the proposed method makes impressive performance improvement.*
Keywords: Software effort, Machine learning, Feature engineering, Genetic algorithm

1. **Introduction.** A software development project's success is influenced by many factors, including executive support, user involvement in the process, team experience, clear business objectives, and software infrastructure. Other factors are related to the project's timing and scope, including the minimal scope and reliable estimation [1]. The more accurate the estimation is, the more likely the software project is to be successful. The estimation at the early stage of software development is also very beneficial to make a project schedule, financial budgeting, and resource management plan.

Software development effort estimation (SDEE) predicts how many resources are needed to complete a project plan in terms of person-hours or person-months. To make an accurate estimation, all software development factors and its environments must be considered, such as software physical size estimation (as expressed in the size of lines of codes), and the functionalities or the requirements of the software (as conveyed in the function points or use-case points).

There are various techniques, estimation models, and tools used for software estimation. The most common approach and easy to implement is by using expert judgment [2]. In this approach, a project estimator tends to use their expertise based on historical data and similar projects to estimate the software effort. This method is very subjective and lacks standardization; thus, it cannot be reusable.

DOI: 10.24507/icicelb.13.01.1

Another approach to estimating the software effort is using algorithmic models such as the constructive cost model (COCOMO) or the system evaluation and estimation of resource-software estimation model [3]. The main driver of these models is the software size estimation, usually the source of lines of code (SLOC). To estimate the SLOC, function points [4] or use-case points [5] analysis is used. Besides the physical software size, the software effort is also computed by considering another set of cost drivers that include subjective assessment factors related to the project's environmental and technical complexity, such as hardware, product, project, and personnel attributes.

Recently, data mining usage with machine learning techniques has been an active research area to estimate software development effort [6] in conjunction or as an alternative to the algorithmic models. The analogy-based estimation (ABE) approach uses the underlying algorithmic model principle, which characterizes the project in features. It can be done by collecting the completed projects, then training the machine learning algorithms with the historical data to predict new projects' software effort. Various machine learning algorithms have been applied for software effort estimation, including gradient boosting machine and deep learning [7].

There are various obstacles encountered when using the machine learning approach to estimate the software development effort. One of them is to find the most relevant features that represent the resulting effort value. In addition to that, noisy features of the data set also affect the accuracy of the estimation. Combining the machine learning algorithms with features selection and weighting process which choose the best set of features and give relevant weights has been proven as a better solution to improve estimation accuracy [8]. It also reduces the complexity of a model and makes it easier to interpret.

Popular methods that are frequently used in searching for the best solution automatically using attributes selection and feature weighting are stepwise regression like forward or backward selection and bio-inspired metaheuristic approaches such as particle swarm optimization [9] or genetic algorithm [10]. These techniques are classified as a wrapper-based approach that involves training a learner during the searching process. This approach's primary disadvantage is that it is computationally slow, but it gives the best feature selection and weights in terms of accuracy.

Nevertheless, the attributes selection and feature weighting only are often not enough. Transforming original features to create new ones could provide superior performance compared to the model developed on the original features in predicting the outcomes [11]. This paper's major contribution is a novel feature engineering approach in the application of software effort estimation domain. The proposed method may select features and may also create new features from the original feature set. It is derived from the genetic algorithm since the algorithm can find a high-quality solution in the full search within a reasonable period of time [12]. As the main advantage of this study, the proposed approach is designed to improve the estimator's accuracy in existing studies. The rest of this paper is structured as follows. In Section 2, the research methods are presented. Meanwhile, the results and discussions are described in Section 3. Lastly, the conclusion is given in Section 4.

2. **Research Methods.**

Like the particle swarm optimization (PSO) algorithm, the genetic algorithm (GA) is also a heuristic technique that is used to construct valuable solutions to optimization and search problems. This heuristic starts with a group of randomly generated populations, evaluates, updates the population with a new generation, and performs a random search for the optimum. The GA imitates the natural evolution process, reflecting the process of natural selection where the fittest individuals from a population are chosen for reproduction to generate offspring of the next generation. A population is a set of chromosomes or, in this context, a set of individual solutions where each individual is composed of genes or feature vectors.

To generate an optimum solution within the possible solution set, the GA uses techniques inspired by natural evolution, such as selection, crossover, and mutation. In the proposed approach, transformations of the feature vectors are performed to boost the algorithm's performance. Unlike the simple GA, the proposed method, generating genetic algorithm (GGA), creates new features and thus may alter the individual's length. It is also known to be a blending process of selecting and generating attributes. Thus, a specialized mutation technique, namely generating mutation, is introduced. The GGA does not modify the original features until it achieves accuracy improvement by adding or deleting (or both) features.

The pseudo-code of the GGA is shown in Algorithm 1. At the first step, an initial population is created with a predetermined size, s. Then, the crossover operation is performed (lines 10-15). The operation recombines the genetic information (features) of two individuals. Then, the generating mutation operation (lines 16-23) performs one of the followings with different probabilities:

- Add a randomly selected original attribute to the feature vector.
- Add a newly generated attribute to the feature vector. The new features are created by applying feature generators such as basic arithmetic operations, and power functions, to the feature vector's random subset.
- Remove a selected attribute randomly from the feature vector.

The selection operation is performed to generate a new generation of the population (lines 24-41). It is composed of the best individual in the current population with the maximum fitness value and others that win the selection tournament. The searching stops when it meets one of two criteria, either the number of generations exceeds the agreed maximum number of generations, $\max_{gen}$, or the determined maximum fitness value, $\max_{fit}$, is less than or equals to the best individual's fitness value.

In the GA, the fitness function is used as the performance metric to evaluate how close the solution is to the problem's optimal solution. In this proposed method, the accuracy of the effort estimation is selected as the fitness criterion. The accuracy is calculated by computing the prediction error, which is the difference between actual and estimated effort. The smaller the error, the more accurate the prediction. In this research, the magnitude of relative error (MRE) is chosen as the performance metric. It is the absolute deviation of prediction from the actual value divided by the actual value.

Equation (1) computes the error where y_i is the predicted value and x_i is the actual value at the ith sample. Moreover, the mean magnitude of relative error ($MMRE$) is also considered as the sample average of the MRE's. It is computed by Equation (2), where n is the number of the samples, $y_1, y_2, \ldots, y_n$ are the predicted values, and $x_1, x_2, \ldots, x_n$ are the actual values. Fitness value in the GA is used for optimization purposes and must always be maximized. Since the error is better, the smaller the value is, the fitness is computed by negating the error, shown by Equation (3). As the best solution's error value is zero, then the maximum fitness value, $\max_{fit}$, should also be zero.

$$MRE = \frac{|y_i - x_i|}{x_i} \tag{1}$$

$$MMRE = \frac{1}{n}\sum_{i=1}^{n}\frac{|y_i - x_i|}{x_i} \tag{2}$$

$$fitness = -1 \times MMRE \tag{3}$$

The data sets are shown in Table 1. Three machine learning algorithms are selected in this experiment as the base regressors, that is, the regression tree (RT) using least square as the split criterion and a maximum depth of 10, the k-nearest neighbor (KNN) using Euclidean distance as similarity function and k of 5, and the generalized linear model (GLM) using Gaussian family. Shuffled five-fold cross-validation is used for learning and

Data: population size s, maximum fitness $\max_{fit}$, maximum number of generations $\max_{gen}$, feature vector f, number of iterations $i = 0$, feature generator vector g = {basic arithmetic operation, square roots, power functions, trigonometric function, exp value, log value, signum value, absolute value, floor ceil}, tournament fraction $tf = 0.25$

Result: the best individual from population p

$p \leftarrow \{p_1, p_2, \ldots, p_s\}$ where $p_i \in p$ has feature vector of a random subset of f;
$\max_{fp} \leftarrow -100$;
foreach individual $p_i \in p$ **do**
    $\max_{fp} \leftarrow \max(\text{fitness}(p_i), \max_{fp})$;
end
while $i < \max_{gen}$ *and* $\max_{fit} \leq \max_{fp}$ **do**
    $i \leftarrow i + 1$;
    $m \leftarrow 0$;
    $p' \leftarrow p$;
    while length$(p') < 1$ **do**
        $\{p_x, p_y\} \leftarrow p'$ random selection;
        remove $\{p_x, p_y\}$ from p';
        crossover(p_x, p_y);
        insert(p_x, p_y) into p;
    end
    foreach individual $p_i \in p$ **do**
        $p'_i \leftarrow p_i$;
        add an original feature randomly to p'_i;
        $g_i \leftarrow g$ random selection;
        generate features mutation by applying g_i to a random subset of features vector of p'_i;
        remove features randomly from p'_i;
        insert p'_i into p;
    end
    foreach $j \leftarrow 0$: length(p) **do**
        if fitness$(p_j) > \max_{fp}$ **then**
            $\max_{fp} \leftarrow \text{fitness}(p_j)$;
            $m \leftarrow j$;
        end
    end
    $new_{gen} \leftarrow \{p_m\}$;
    $ts \leftarrow \text{length}(p) \times tf$;
    while length$(new_{gen}) < s$ **do**
        $winner \leftarrow \emptyset$;
        foreach $k \leftarrow 0 : ts$ **do**
            $challenger \leftarrow p$ random selection;
            if fitness$(challenger) >$ fitness$(winner)$ **then**
                $winner \leftarrow challenger$;
                insert $winner$ into new_{gen};
            end
        end
    end
    $p \leftarrow new_{gen}$;
end
return p_1

Algorithm 1: Generating genetic algorithm

Table 1. Data sets

No	Dataset	Records	Features	Effort (unit of measures)
1	Albrecht [13]	24	8	Person-hours
2	China [14]	499	19	Person-hours
3	Cocomo81 [15]	63	17	Person-months
4	Deshernais [16]	81	12	Person-hours
5	Finnish [17]	38	9	Person-hours
6	Kemerer [18]	15	8	Person-months
7	Kitchenham [19]	145	10	Person-hours
8	Maxwell [20]	62	27	Person-hours
9	Miyazaki94 [21]	48	9	Person-months
10	NASA60 [22]	60	17	Person-months
11	NASA93 [23]	93	24	Person-months
12	SDR [24]	12	25	Person-months

testing data. Finally, a null hypothesis significance testing is used to compare the models with a predetermined significant level. It is a test designed to determine if two related treatments are statistically significantly different. A null hypothesis proposes that there is no significant difference between the models.

3. **Results and Discussions.** In this experiment, the PSO algorithm is compared with the proposed features engineering method. The comparison is performed with the proposed GGA without generating mutation (GGA1) and with generating mutation (GGA2). All features along with their weight that was greater than zero were utilized by the machine learning algorithms. In all experiments, the maximum number of generations, $\max_{gen}$, is limited to 30, and the value of the initial population size, s, is set to 5.

At first, the experiments on 12 data sets are conducted using the base regressors to estimate the effort label in the data sets. Not all attributes in the data sets are used as machine learning features since some do not exist in the early software development stage. For instance, attributes of "duration" and "productivity" can be only obtained after the development process. Thus, these attributes are not included in the base features. The number of initial features is shown in Table 2 in the 'Base' column. The table also reports the number of used features in the other columns as the results of experimenting with the implementation of the PSO, the GGA1, and the GGA2 using the base regressors. Meanwhile, the number in the brackets represents the number of generated features that are utilized by the GGA2. From the results, it can be highlighted that the GGA-based models utilized fewer features than the PSO-based models. It indicates that the models generated by the GGA were simpler than the PSO.

The relative errors of implementing the different approaches using the RT, the KNN, and the GLM as the base regressors are shown in Table 3 where the lowest error for each data set is boldly printed. On average, the base performance of the KNN was better than the RT and the GLM. This result is also confirmed by [25] which states the KNN is the most suitable single technique after the support vector regressor (SVR). It might be due to the KNN's simplicity that makes no assumption with the correlations (linear relationship) between the features and the label. Surprisingly, the GLM has the worst performance out of the RT and the KNN even though it is supported by [26] which reports that linear models like neural networks and simple linear regression perform much worse than other learners. If the hyperparameters of the GLM learning algorithm were tuned during the experiments, the model's performance could be better.

Moreover, on average, the attributes selection and features weighting using the PSO increased the base performance by 20% for the RT, 22% for the KNN, and only 2% for

TABLE 2. The number of used features in the RT, the KNN, and the GLM as base regressors

No	Data set	Base	RT + PSO	RT + GGA1	RT + GGA2	KNN + PSO	KNN + GGA1	KNN + GGA2	GLM + PSO	GLM + GGA1	GLM + GGA2
1	Albrecht	7	2	2	2 (1)	5	2	2 (1)	6	3	2 (6)
2	China	6	5	1	1 (1)	4	3	3 (3)	6	3	3 (4)
3	Cocomo81	16	12	7	5 (2)	5	9	7 (3)	9	13	9 (5)
4	Deshernais	8	5	2	2 (4)	6	4	2 (2)	3	2	1 (2)
5	Finnish	4	3	1	1 (2)	3	1	2 (4)	2	1	1 (5)
6	Kemerer	5	5	1	1 (2)	4	2	1 (1)	4	2	2 (5)
7	Kitchenham	3	2	1	1 (5)	3	2	2 (4)	1	1	1 (4)
8	Maxwell	24	13	3	6 (1)	17	1	2 (3)	19	2	2 (0)
9	Miyazaki94	7	3	2	1 (3)	3	1	1 (3)	7	1	1 (0)
10	NASA60	16	11	3	2 (1)	15	12	1 (2)	10	5	2 (2)
11	NASA93	22	14	3	1 (1)	16	6	1 (2)	18	8	1 (1)
12	SDR	23	20	8	1 (2)	12	9	1 (2)	13	4	1 (5)
	Average		67%	25%	36%	69%	40%	41%	69%	34%	42%

TABLE 3. The relative errors using the RT, the KNN, and the GLM as base regressors

	RT				KNN				GLM			
Data set	Base	PSO	GGA1	GGA2	Base	PSO	GGA1	GGA2	Base	PSO	GGA1	GGA2
Albrecht	1.025	**0.754**	**0.754**	**0.754**	0.859	**0.740**	0.816	0.845	1.411	1.406	1.784	**1.165**
China	2.075	1.913	1.683	**1.649**	1.780	**1.478**	1.581	1.555	**3.025**	**3.025**	3.390	3.154
Cocomo81	1.789	1.302	**1.072**	1.132	1.920	**1.234**	1.919	1.909	18.18	18.17	18.20	**18.04**
Deshernais	0.762	**0.667**	0.699	0.684	0.685	0.682	0.657	**0.639**	1.216	0.957	0.999	**0.909**
Finnish	1.005	0.937	1.114	**0.881**	1.114	**0.939**	1.114	1.065	2.640	2.639	2.639	**2.636**
Kemerer	1.692	1.692	1.692	**0.558**	1.052	**0.872**	0.896	0.881	1.496	1.495	1.568	**1.210**
Kitchenham	2.011	**2.001**	2.033	1.020	0.901	**0.805**	0.901	0.831	2.095	1.917	1.917	**1.599**
Maxwell	0.762	**0.511**	0.615	0.597	0.615	**0.509**	0.608	0.570	2.021	2.021	1.948	**1.837**
Miyazaki94	2.472	1.524	1.499	**1.066**	1.001	0.513	0.520	**0.463**	2.103	2.103	**1.196**	1.410
NASA60	0.518	0.455	**0.445**	0.467	0.442	**0.363**	0.456	0.430	5.036	4.954	5.001	**0.715**
NASA93	1.425	1.361	1.221	**0.826**	1.460	**0.952**	1.175	1.329	7.150	6.948	7.158	**2.243**
SDR	1.751	0.705	**0.670**	**0.670**	0.571	0.571	0.571	**0.537**	1.409	1.263	1.041	**0.730**
Average	1.441	1.152	1.125	**0.859**	1.033	**0.805**	0.935	0.921	3.982	3.908	3.904	**2.970**

the GLM. On the other hand, the GGA1 was no better than the PSO since it improved the base regressors' performance by 22% for the RT, only 9% for the KNN, and 2% for the GLM on average. Nevertheless, the features engineering with the proposed method, the GGA2, made a further improvement on the base performance by 40% for the RT, 11% for the KNN, and 25% for the GLM on average.

With fewer features, the performance of GGA1 is comparable to the PSO using the RT and the GLM as the base regressors. However, the performance of GGA2 is superior to the PSO for those regressors. On the other hand, the PSO makes impressive improvements using the KNN regressor. It might be due to proper weights assigned by the PSO to the KNN. It is supported by [27], which states that the PSO with the ABE method like the KNN leads to a high-performance model.

Since the distribution of the variance between the means of relative errors cannot be considered to be normally distributed, the paired two-tailed Wilcoxon signed-rank test is chosen as a non-parametric statistical hypothesis test [28] to compare the model's performance by computing the p_{value}. It is the likelihood of achieving test outcomes at least as extreme as the results currently obtained, assuming that the null hypothesis is correct [29]. A low p_{value} indicates that such an unusual observed result will be improbable under the null hypothesis. If the value is greater than the predetermined significance level of 0.05, then the null hypothesis cannot be rejected.

The results of p_{value} for each approach using the RT, the KNN, and the GLM as the base regressors are shown in Tables 4, 5, and 6, respectively. As can be inferred from the test results, the PSO implementation and the proposed method, GGA2, have significant differences with all base regressors. It means that the PSO and the GGA2 make an impressive performance improvement in estimating the software effort.

Furthermore, there are insignificant differences between the PSO and the GGA1 using the RT and the GLM as the base regressors. As can be seen from Tables 4 and 6, the p_{value} amounted 0.50926 and 0.54186 respectively. This result confirms [30] that a simple genetic algorithm does not have a significant distinction with the PSO when used as a feature selection method. Nevertheless, the PSO elevates the performance essentially when used as feature selection and weighting for the KNN regressor since it has notable differences with all approaches. As also can be seen from Table 5, there is no significant distinction between GGA1 and GGA2 for the KNN since the p_{value} is greater than 0.05. Moreover, from Table 6, it can be seen that GGA1 failed to improve the GLM base performance since the p_{value} is much higher than 0.05. However, the proposed method, GGA2, produces a further gain on the performance since it has notable differences, even with the PSO, using the RT and the GLM as the base regressors.

Table 4. Significant test results using the RT as the base regressor

	RT	RT + PSO	RT + GGA1	RT + GGA2
RT		0.00338 (sig.)	0.01278 (sig.)	0.00222 (sig.)
RT + PSO	0.00338 (sig.)		0.50926 (not sig.)	0.02642 (sig.)
RT + GGA1	0.01278 (sig.)	0.50926 (not sig.)		0.04660 (sig.)
RT + GGA2	0.00222 (sig.)	0.02642 (sig.)	0.04660 (sig.)	

Table 5. Significant test results using the KNN as the base regressor

	KNN	KNN + PSO	KNN + GGA1	KNN + GGA2
KNN		0.00338 (sig.)	0.02780 (sig.)	0.00222 (sig.)
KNN + PSO	0.00338 (sig.)		0.00758 (sig.)	0.03400 (sig.)
KNN + GGA1	0.02780 (sig.)	0.00758 (sig.)		0.09894 (not sig.)
KNN + GGA2	0.00222 (sig.)	0.03400 (sig.)	0.09894 (not sig.)	

Table 6. Significant test results using the GLM as the base regressor

	GLM	GLM + PSO	GLM + GGA1	GLM + GGA2
GLM		0.00512 (sig.)	0.47770 (not sig.)	0.00374 (sig.)
GLM + PSO	0.00512 (sig.)		0.54186 (not sig.)	0.00480 (sig.)
GLM + GGA1	0.47770 (not sig.)	0.54186 (not sig.)		0.00758 (sig.)
GLM + GGA2	0.00374 (sig.)	0.00480 (sig.)	0.00758 (sig.)	

4. **Conclusions.** A novel feature engineering approach for software effort estimation is proposed in this paper. The proposed method is applied to selecting features and generating new features from the original feature subset. The algorithm is employed to deal with the noise attributes problem and improve the estimation accuracy. A comparative study of the implementation of the proposed approach and the particle swarm optimization as the state-of-the-art using three machine learning algorithms applied to 12 data sets with the context of estimating software development effort was conducted.

The experimental results show that the proposed method made an impressive prediction improvement to the base performances. Furthermore, it achieved the lowest relative error

with fewer features using the regression tree and the generalized linear model as the base regressors. On the other hand, the particle swarm optimization approach obtained the highest accuracy using the k-nearest neighbor regressor.

From the comparison results, it also can be concluded that there is no significant difference between the particle swarm optimization and the generating genetic algorithm without generating mutation when utilized as feature selection using the regression tree and the generalized linear model as the base regressors. However, the last-mentioned method tends to utilize fewer features. Moreover, with generating mutation, the algorithm makes a further significant difference. Blending feature selection with generating new features works better than combining the feature selection and weighting in this case. More research will be conducted for investigating and benchmarking the metaheuristic methods to optimize the hyperparameters of the machine learning algorithms.

Acknowledgment. The computation in this work has been done using the facilities of HPC LIPI, the Indonesian Institute of Sciences (LIPI). The authors also gratefully acknowledge the helpful comments and suggestions of the reviewers, which have improved the presentation.

REFERENCES

[1] S. K. Sehra, Y. S. Brar, N. Kaur and S. S. Sehra, Research patterns and trends in software effort estimation, *Information and Software Technology*, vol.91, pp.1-21, 2017.

[2] Y. Mahmood, N. Kama and A. Azmi, A systematic review of studies on use case points and expert-based estimation of software development effort, *Journal of Software: Evolution and Process*, vol.32, no.7, 2020.

[3] K. E. Rao and G. A. Rao, Ensemble learning with recursive feature elimination integrated software effort estimation: A novel approach, *Evolutionary Intelligence*, vol.14, no.1, pp.151-162, 2020.

[4] J. F. Vijay, Enrichment of accurate software effort estimation using fuzzy-based function point analysis in business data analytics, *Neural Computing and Applications*, vol.31, no.5, pp.1633-1639, 2019.

[5] M. Azzeh, A. B. Nassif and S. Banitaan, Comparative analysis of soft computing techniques for predicting software effort based use case points, *IET Software*, vol.12, no.1, pp.19-29, 2018.

[6] P. Jodpimai, P. Sophatsathit and C. Lursinsap, Re-estimating software effort using prior phase efforts and data mining techniques, *Innovations in Systems and Software Engineering*, vol.14, no.3, pp.209-228, 2018.

[7] P. Phannachitta and K. Matsumoto, Model-based software effort estimation – A robust comparison of 14 algorithms widely used in the data science community, *International Journal of Innovative Computing, Information and Control*, vol.15, no.2, pp.569-589, 2019.

[8] A. K. Bardsiri and S. M. Hashemi, Machine learning methods with feature selection approach to estimate software services development effort, *International Journal of Services Sciences*, vol.6, no.1, pp.26-37, 2017.

[9] E. El-Kenawy and M. Eid, Hybrid gray wolf and particle swarm optimization for feature selection, *International Journal of Innovative Computing, Information and Control*, vol.16, no.3, pp.831-844, 2020.

[10] J. Murillo-Morera, C. Quesada-López, C. Castro-Herrera and M. Jenkins, A genetic algorithm based framework for software effort prediction, *Journal of Software Engineering Research and Development*, vol.5, no.1, p.4, 2017.

[11] S. E. Awan, M. Bennamoun, F. Sohel, F. M. Sanfilippo, B. J. Chow and G. Dwivedi, Feature selection and transformation by machine learning reduce variable numbers and improve prediction for heart failure readmission or death, *PLoS One*, vol.14, no.6, e0218760, 2019.

[12] M. Sharma and P. Kaur, A comprehensive analysis of nature-inspired meta-heuristic techniques for feature selection problem, *Archives of Computational Methods in Engineering*, 2020.

[13] A. J. Albrecht and J. E. Gaffney, Software function, source lines of code, and development effort prediction: A software science validation, *IEEE Transactions on Software Engineering*, vol.SE-9, no.6, pp.639-648, 1983.

[14] J. S. Shirabad and T. J. Menzies, *The PROMISE Repository of Software Engineering Databases*, School of Information Technology and Engineering, University of Ottawa, Canada, http://promise.site.uottawa.ca/SERepository, 2005.

[15] B. W. Boehm, Software engineering economics, *IEEE Transactions on Software Engineering*, vol.SE-10, no.1, pp.4-21, 1984.
[16] M. Shepperd and C. Schofield, Estimating software project effort using analogies, *IEEE Transactions on Software Engineering*, vol.23, no.11, pp.736-743, 1997.
[17] B. Kitchenham and K. Kansala, Inter-item correlations among function points, *Proc. of the 1st International Software Metrics Symposium*, pp.11-14, 1993.
[18] C. F. Kemerer, An empirical validation of software cost estimation models, *Commun. ACM*, vol.30, no.5, pp.416-429, 1987.
[19] B. Kitchenham, S. L. Pfleeger, B. McColl and S. Eagan, An empirical study of maintenance and development estimation accuracy, *Journal of Systems and Software*, vol.64, no.1, pp.57-77, 2002.
[20] K. D. Maxwell, *Applied Statistics for Software Managers*, Prentice Hall PTR, 2002.
[21] Y. Miyazaki, M. Terakado, K. Ozaki and H. Nozaki, Robust regression for developing software estimation models, *Journal of Systems and Software*, vol.27, no.1, pp.3-16, 1994.
[22] T. Menzies, D. Port, Z. Chen and J. Hihn, Validation methods for calibrating software effort models, *Proc. of the 27th International Conference on Software Engineering (ICSE2005)*, pp.587-595, 2005.
[23] E. Kocaguneli, T. Menzies and J. W. Keung, On the value of ensemble effort estimation, *IEEE Transactions on Software Engineering*, vol.38, no.6, pp.1403-1416, 2012.
[24] L. L. Minku and X. Yao, Ensembles and locality: Insight on improving software effort estimation, *Information and Software Technology*, vol.55, no.8, pp.1512-1528, 2013.
[25] M. Hosni, A. Idri, A. Abran and A. B. Nassif, On the value of parameter tuning in heterogeneous ensembles effort estimation, *Soft Computing*, vol.22, no.18, pp.5977-6010, 2018.
[26] J. Keung, E. Kocaguneli and T. Menzies, Finding conclusion stability for selecting the best effort predictor in software effort estimation, *Automated Software Engineering*, vol.20, no.4, pp.543-567, 2013.
[27] V. Bardsiri, D. A. Jawawi, S. M. Hashim and E. Khatibi, A PSO-based model to increase the accuracy of software development effort estimation, *Software Quality Journal*, vol.21, no.3, pp.501-526, 2013.
[28] F. Wilcoxon, Individual comparisons by ranking methods, *Biometrics Bulletin*, vol.1, no.6, pp.80-83, 1945.
[29] R. L. Wasserstein and N. A. Lazar, The ASA statement on p-values: Context, process, and purpose, *The American Statistician*, vol.70, no.2, pp.129-133, 2016.
[30] R. Wahono, N. Suryana and S. Ahmad, Metaheuristic optimization based feature selection for software defect prediction, *Journal of Software*, vol.9, no.5, pp.1324-1333, 2014.