Baltic J. Modern Computing, Vol. 7 (2019), No. 1, pp. 31–46
https://doi.org/10.22364/bjmc.2019.7.1.03

Is Complexity-based Clustering of Process Metrics as Effective as in Static Code Metrics

Muhammed Maruf ÖZTÜRK

Suleyman Demirel University, Faculty of Enginnering, Department of Computer Engineering

`muhammedozturk@sdu.edu.tr`

Abstract. Defect prediction is not a new sub-field of software engineering. However, in this field, there are various research problems that are still attractive for many researchers. Cross-project defect prediction (CPDP), which is one of the popular issues of defect prediction, is still intriguing. To address this problem, generally instances or feature-focused experiments are performed but there is a lack of novel pre-processing methods. The main objective of this work is to propose a fuzzy clustering method that is based on complexity in CPDP. It helps selecting training data of CPDP. Hence, an opportunity that provides comparing static code and process metrics emerges. In this work, complexity-based fuzzy clustering that helps to select training instances of CPDP is proposed for process metrics. In the method, fuzzy membership levels are associated with a complexity value based on process metrics. In the experiment, 18 data sets including static code and process metrics together are employed. The findings of the experiment show that although static code metrics perform better than process metrics in terms of area under the curve (AUC), process metrics outperforms static code metrics in matthew's correlation coefficient (MCC) and F-measure parameters. Furthermore, in accordance with the used data sets, it has been detected that there is not any linear model among process metrics including number of revisions (NR), number of modified lines (NML), and number of distinct committers (NDC). This work asserts that the approach on the basis of training instance selection of CPDP yields remarkable success in process metrics. Moreover, in overall performance, process metrics are rather suitable for clustering-based instance selection.

Keywords: cross-project defect prediction, fuzzy clustering, process metrics

1 Introduction

Software defect prediction is conducted based on software metrics (Moser et al., 2008; Romano and Pinzger, 2011; Menzies et al., 2010). It is widely known that two types of metrics are commonly used: static code and process metrics (Shihab et al., 2010; Gray et al., 2009; Nagappan et al., 2005). Initially, static code metrics had gained a great popularity but methods developed for process metrics have shown a great growth over the last decade (Lee et al., 2016; Rahman and Devanbu, 2013).

CPDP is one of the defect prediction sub-working fields and maintains its validity. Software metrics are the origin of many difficulties while performing CPDP. For instance, metric heterogeneity is of great importance for the reliability of prediction results. Researchers show intensive effort to handle metric matching problem in heterogeneous software metrics. However, it may not be sufficient to match only software metrics in order to perform a consistent prediction process. In addition to this, practitioners should take pre-processing methods into account during this process. In doing so, not only proper training instances of CPDP can be selected, but also heterogeneous defect prediction can be facilitated. Sophisticated and exhaustive experiments are required to achieve this purpose.

CPDP is twofold: instance and feature-focused works. Further, hybrid methods are also employed in CPDP (Xia et al., 2016; Yu et al., 2016). However, instance-focused works mainly include novel prediction models and there are few works that especially bring novelty in terms of data pre-processing (Li et al., 2017). Data sets are exposed to various pre-processing methods in software defect prediction. The main purpose of those methods could be noise filtering (Kim et al., 2011), sampling (Li et al., 2012), clustering (Jureczko et al., 2010), and normalization (Wang et al., 2016).

Generally, a great number of pre-processing studies have been done in literature in which they are employed on a specific data set group. However, the effectiveness of pre-processing methods has not been considered depending on the types of metrics. Moreover, few works examined the efficacy of preprocessing methods in prediction performance. One of them is clustering. It is an unsupervised method that divides a given data sets into clusters. Despite a large amount of clustering alternatives, a decision should be taken by considering the characteristic of data sets. Some deficiencies originated from the software development process emerge during gathering the information of data sets. These deficiencies may cause misleading labeling of data set instances. Therefore, fuzzy clustering could provide a better insight for the data sets. In our preceding work (Ozturk, 2017), a complexity-based fuzzy clustering was developed for static code metrics. This work differs in determining the levels of fuzziness membership and the type of metrics.

Defect prediction data could have noisy instances. A noisy instance can be defined as an instance having wrong label that adversely affect prediction accuracy. They create adverse affects in testing process of a machine learning algorithm. To address this problem, whether checking the type of the data sets, they should be exposed to a reliable instance selection process. Fuzzy-rule based methods are preferable in such cases (Mendel et al. 2017; Ashfaq et al., 2017). They divide a data set into training and testing sets. However, some selected instances are labeled both for defective and non-defective due to noisiness. Such gray-labeled instances are used in twofold. In this way, practitioners can achieve a much more clear bias about the labels of the instances.

In this work, a novel fuzzy clustering method relying on the complexity of process metrics is proposed for selecting CPDP training data sets. This work also discusses how does fuzzy clustering yield favorable results in static code and process metrics depending on the complexity. complexFuzzy is the basis for the method presented in this paper. It was tried on static code metrics in the preceding work. CPDP has taken a great attention especially with regard to heterogeneity. Investigating static code and process metrics together in CPDP via clustering is also of crucial importance for the researchers who are focused on the heterogeneity of software metrics.

The contribution and the novelty of the paper can be summarized as follows:
1. It is investigated whether process metrics have linear models.
2. A fuzzy clustering method is presented for process metrics.
3. Complexity-based fuzzy clustering is evaluated in process and static code metrics through some performance parameters.
4. The success of CPDP is discussed in process metrics via training instances selected with complexity-based fuzzy clustering.

The remainder of the paper is organized as follows. Section 2 summarizes the literature. Section 3 details the method. Prediction and evaluation are presented in Section 4. The findings of the experiment are in Section 5. The effects of these findings are discussed in Section 6. Conclusion and future works are given in Section 7.

2 Literature review

The main aim of this section is to give information about the works which investigate the relationship between process metrics and software quality. In addition to this, summarizing literature explains specific reason why process metrics should regarded in the methods developed for defect prediction data sets.

It is widely known that static code metrics had been commonly used in initiative defect prediction works (Menzies et al., 2007). The primary reason is that some metric standards such as Halstead (Bailey and Dingee, 1981) and McCabe (Curtis et al., 1979) were designed for static code metrics. However, diversifying software development models (Clarke et al., 2016; Abrahamsson et al., 2017; Gousios et al., 2014) and considering other factors of software development revealed process metrics.

Developer behavior varies in open-source and industrial software projects. Process metrics were examined in detail once this case was noticed. It was observed that industrial project are far more preferable than open-source projects for process metrics (Madeyski and Jureczko, 2015). Moreover, the most effective process metrics should be detected via multi-correlation analysis rather than binary analysis. Micro-interaction metrics were developed for illustrating implications of developers in software defect prediction (Lee et al., 2016). They can be defined a sub-type of process metrics. Although these metrics greatly contribute classification, they could only be tested on single software project. It is expected that prediction models show better performance in increasing software versions. This success is directly related to the used software metrics. Rahman ve Devanbu (Rahman and Devanbu, 2013) stated that static code metrics are more stagnant than process metrics. They also detected that static code metrics do not evolve in varied number of defects. Thus, process metrics should be considered for each prediction model.

Code review process also makes contribution for process metrics. Poorly reviewed codes create an adverse effect on software quality (Wiese et al., 2014; Faucault et al., 2014). It is not negligible that social metrics are of great importance for prediction models as well as code review. While some of them produce favorable results, some of them do not. For instance, as the number of developers increases, defectiveness becomes seriously challenged.

Though ownership metrics, which are closely related to process metrics, cannot be defined as process metrics, their effects on software quality were investigated in preceding works (Faucault et al., 2014). According to the obtained results, ownership metrics have a poor efficacy on software quality. It is known among researchers that static code and process metrics yield similar results in terms of prediction performance (Stanic and Afzal, 2017). However, process metrics include some leading metrics that produce remarkable success (for instance: NDC). Churn metric (Layman et al., 2008), which is one of the process metrics, yielded favorable results in terms of prediction accuracy. This case is an indicator that each intervention to software development creates a favorable or unfavorable effect on software quality.

The works proposing new methods to gather process metrics are needed in this field. Extracting process metrics during software development is an effort-intensive operation if there is a lack of record. To solve this problem, Gyimesi (Peter, 2017) developed an automated tool that computes process metrics in each phase of software development. This tool is able to compute process metrics through a graph database. The method was tested with some classifiers and RandomForest was found to be best. The reason may be the structure of RandomForest that it converts data to trees.

Table 1: Metrics of experimental data sets.

Name	**Description.**	**Type**
wmc	Weighted Methods per Class	Static code
dit	Depth of inheritance	Static code
noc	Number of children	Static code
cbo	Coupling between objects	Static code
rfc	Response for a class	Static code
lcom	Lack of cohesion	Static code
ca	Afferent coupling	Static code
ce	Efferent couplings	Static code
npm	Number of Public Methods	Static code
lcom3	A variant of lcom	Static code
loc	Line of codes	Static code
dam	Data Access Metric	Static code
moa	Measure of. Aggregation	Static code
mfa	Measure of functionality abstraction	Static code
cam	Cohesion Among Methods of class	Static code
ic	Inheritance Coupling	Static code
cbm	Coupling between Methods	Static code
amc	Average Method Complexity	Static code
nr	Number of revisions	Process
ndc	Number of distinct committers	Process
nml	Number of modified lines	Process
ndpv	Number of defects fixed in previous version	Process
max_cc	Maximum Class Coupling	Static code
avg_cc	Average Class Coupling	Static code

In summary, the methods examining process metrics have become popular among researchers. Coding tendencies of development teams give tips to evaluate process metrics. Therefore, process metrics should be regarded along with theoric approaches and related experiments.

3 Method

In this section, essentials of complexity formula of process metrics, experimental data sets and developed fuzzy clustering algorithm are detailed. These operations require sophisticated statistical methods and rely on a powerful metric value observation as well.

3.1 Data sets

18 open-source data sets have been used in the experiment. These data sets also have different versions of same project. Further, static code and process metrics are together in the data sets (Madeyski and Jureczko, 2015). However, some metrics such as micro interaction are not involved in th experiment so that in its current design, the experiment cannot achieve the ultimate comprehensiveness in terms of the data sets. Micro interaction metrics are planned to be adopted in improved versions of the paper.

In order to reveal the relationship between those metrics and bugs should be examined in detail. To this end, connected scatterplot can be used. However, a data sheet could be generated to gather all the values of the data sets. In this sheet, statistical analysis can be conducted via regression and other methods. Hence, the fundamentals of complexity-based fuzzy clustering can be explained well.

The values of process metrics of the data sets are not complete for all projects. This case also creates a threat for the validity. Madeyski et al. (Madeyski and Jureczko, 2015) also used those data sets and it is clear that they were incomplete in their experiment. Note that sharing data corpus is so important in software engineering studies. Hence, the deficiencies of data sets can be removed or a window of opportunity is opened to work on complete data sets.

In Table 2, values of all the projects are presented. This table also includes different versions of a same project. The number of instances and process metric values are available in this table. For instance, in ”ant”, merely 1568 instances have NR records that it has 2442 instances. Table 1 gives used metrics with their definitions and types. There are 24 software metrics. 4 out of 24 metrics are of process metrics.

3.2 Statistical analysis of process metrics

The main aim of the analyzes presented in this section is to determine whether a threat is available for the formula generated with three process metrics. The corresponding formula is seen in Equation 2. A three-phase analysis process is designed. First of them is linear model analysis. This analysis helps to detect whether a linear model is available among process metrics. Second is scatterplot. It demonstrates that grid value endpoints are not compatible with metric values that prove nonlinearity. Third is kernel density analysis. It helps estimate the unknown probability distribution of a random variable that is taken from a sample of points. Thanks to this analysis, a nonparametric test can be applied on experimental metric set.

$$y_i = z_0 + z_1x_1 + z_2x_2 + ... + e_i \tag{1}$$

Table 2: Details of the projects used in the experiment.

Project	Number of instances	Number of NR	Number of NDC	Number of NML	Percentage of defectiveness
ant	2442	1568	2255	1487	14
camel	3576	1225	1228	1062	15
log4j	557	114	404	0	46
jedit	3695	1477	3190	1268	8
ivy	933	593	798	0	12
lucene	1205	587	918	506	36
pBeans	96	14	14	79	66
poi	1683	1141	1383	1	41
prop1	23058	18798	19366	19366	10
prop2	10686	8463	8822	8822	11
prop3	8867	7024	7158	7158	9
prop4	8572	5645	5645	5645	15
prop42	871	404	500	0	7
prop5	20407	14817	17417	17417	13
synapse	661	478	499	399	24
Velocity	731	443	507	0	50
xalan	4171	2597	3279	2249	43
xerces	1937	1481	1731	962	33

Analyzed mathematical model is seen in Equation 1. zs are the coefficients of dependent variables. xs are dependent variables and they predict independent variable. The error of the model is represented with e. Figure 1 has four sub-figures. These figures were drawn with lm function of R package. Here NR is the response variable and it is estimated with NDC and NML. R^2 value shows to what extent the model fits the data. Residuals vs. Fitted investigates whether residuals have a non-linear model. In the corresponding sub-figure, it can be concluded that there is not a non-linear model if the values are spread in a straight line. In the figure, we have not a straight line and there are discrete models. Due to this situation, there is a non-linear relationship among NR, NDC, and NML.

Normal Q-Q indicates whether residuals have a normal distribution. Scale-Location looks for equal variances. If there is a straight line, variance is equal. Cook's distance gives the magnitude of the effects of the cases on regression analysis. In the figure, the magnitude is significant. The details of linear model analysis are shown in Table 3. It can be clearly seen from Figure 1 that the linearity among process metrics is reasonably poor.

To better understand the underlying structure of the formula adapted to process metrics, regression and scatterplot analyzes are conducted. The corresponding metrics should be opposite to a linear relationship. The relationships are presented with scatterplot.

Another analysis that investigates whether there is a linear model between variables is scatterplot. If there is a linear model, fitted plane should be available. In Figure 2 and 3, an example scattterplot and the scatterplot of NR-NDC-NML are presented, respectively. It can be known from Figure 3 that fitted plane is not as desired. In the Figure 3, as grid does not comply with extreme values, it is difficult to say that there is a linear model between three process metrics. On the contrary, double examination of the metrics may disprove this assertion. Note that these analyzes supports the assertion that there is not any linear relation between three metrics.

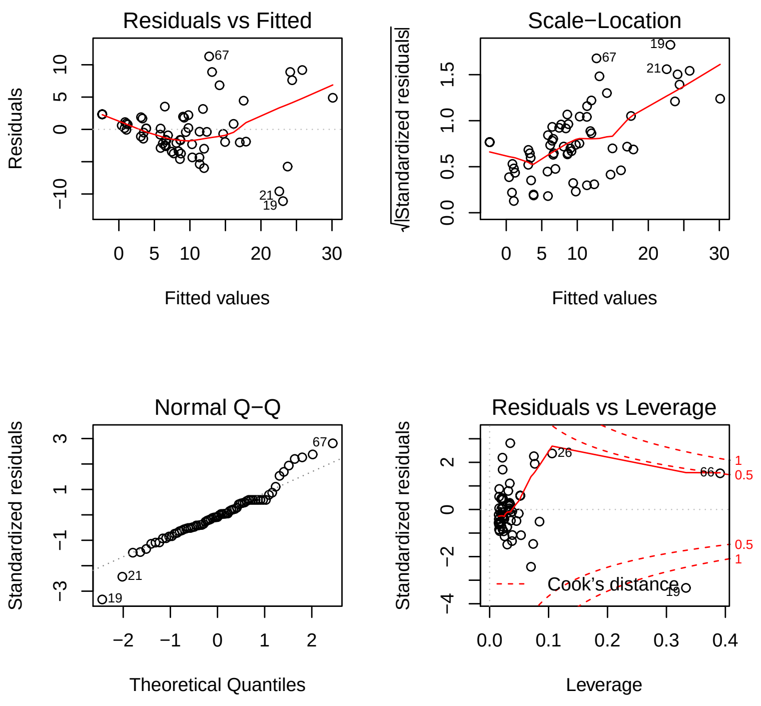

Fig. 1: Lineer model analysis of NR, NML, and NDC.

In order to apply non-parametric tests on data sets, it should be investigated whether data sets have a normal distribution. To this end, kernel density analysis is performed with three process metrics. Figure 4 shows that corresponding metrics have not a normal distribution.

Table 3: Details of the lineer model analysis (NR, NDC, and NML).

Residuals:				
Min	1Q	Median	3Q	Max
-11.1044	-2.1838	-0.3558	2.3380	11.2934
	Estimate	Std. Error	t value	Pr(> $\|t\|$)
(Intercept)	-2.337994	0.931316	-2.510	0.0145
nml	0.005360	0.001288	4.162	9.32e-05
ndc	2.685169	0.264997	10.133	4.43e-15
RSE: 4.087	66 degrees of freedom			
Multiple R-squared: 0.7821	Adjusted R-squared: 0.7755			
F-statistic: 118.5 on 2 and 66 DF	p-value: $<$ 2.2e-16			

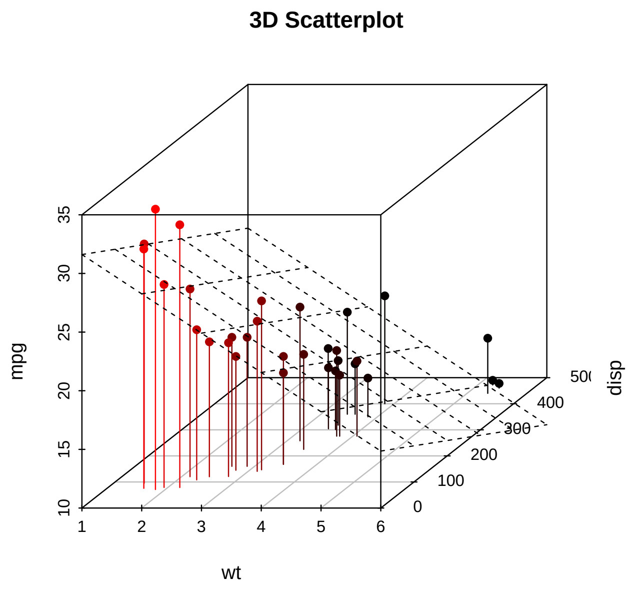

Fig. 2: An example scatterplot that is fitted to plane.

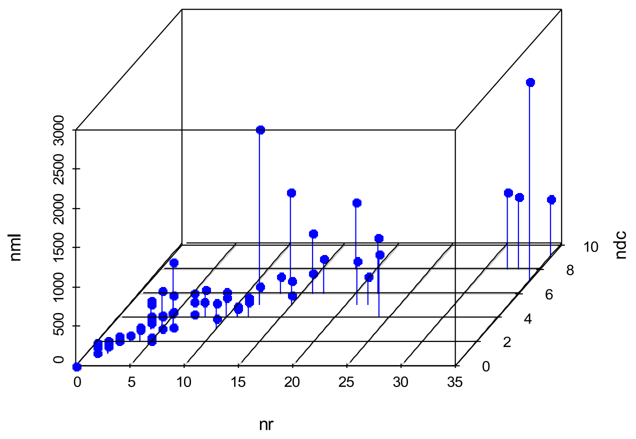

Fig. 3: Scatterplot of NR, NDC, and NML on average values of all the data sets.

Algorithm 1 complexFuzzy Algorithm

```
Step1: Input all instances as p(x,y) with clusters, fuzziness, and coEfficient.
Step2: Define membership matrix $U$ depending on the number of instances and the number of clusters.
Step3: Iterate through all instances to create initial $U$ matrix. (Compute $diff = \sqrt{(p_x - c_x)^2 + (p_y - c_y)^2}$ ), IF $coefficient > 10$ diff+=50 else diff-=50; $U[i,j]$=(diff==0) ? $10^{-5}$ : diff , S+=$U[i,j]$
Step4: Update $U[i,j]$ for each cluster by calculate $U[i,j] = 1.0/((U[i,j]/sum)^{2.0/(fuzziness-1.0)})$ sum2+=$U[i,j]$
Step5: $U[i,j] = U[i,j]/sum2$;
Step6: Recalculate cluster indexes by comparing $max$ and $U[i,j]$ values.
```

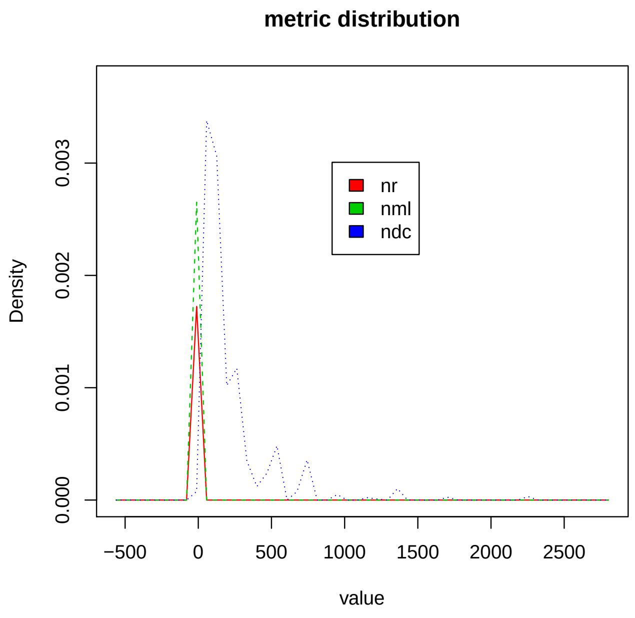

Fig. 4: Kernel density analysis of NR, NML, and NDC. The graph has been drawn with the mean values of all data sets.

3.3 Algorithm

First version of complexFuzzy had used static code metrics including WMC, LCOM, and RFC. In this study, $coEfficient$ of complexFuzzy is calculated using NR, NML, and NDC. The steps presented in Algorithm 1 and its preceding version is the same except for complexity-based fuzziness computation. While $p(x, y)$ denotes the instances, U is the membership matrix of clusters. They are represented with c and $diff$ refers to the distance from points to the cluster. In Step 3, $coEfficient$ affecting the level of membership is calculated. Cluster values of all instances are re-calculated by comparing with max. The difference emerges in the computation of $coEfficient$. In this computation, complexFuzzy had used static code metrics. Rather, a modified version of complexFuzzy has been designed. That formula is seen in Equation 2. In the formula, NR and NML are in numerator. The reason is that the number of revision and the number of modified code lines have similar correlation with complexity. However, these calculations are inversely proportional with NR. In other words, they are performed depending on the discrete commit count.

$$coEfficient = \frac{\sum_{i=0}^{n}(NR_i * NML_i)/NDC_i}{n} \tag{2}$$

4 Prediction and evaluation

Prediction experiment is established in accordance with CPDP configurations. For this purpose, combinations of CPDP are utilized. First, training data groups are selected by developed algorithm for each data set. Mean prediction performance is then recorded in which training data groups are tested on the other data sets. This process is repeated for SVM, Naive Bayes, Random Forest, and C4.5 algorithms. 10*10 cross-validation is conducted to obtain prediction results. In 10 fold cross-validation, a data set is divided into 10 sub-sets. For each iteration, one sub-set is used for testing and the others are utilized in training. During this process, testing sub-set is changed and average error is computed. Since total CPDP operations is $n = 18$, 306 is the number of predictions calculated with $combinationCount = n * (n - 1)$.

Performance parameters recorded during the experiment are F-measure, MCC, and AUC. The corresponding values are of the average of four predictors. AUC is area under the curve of receiver operating characteristic (ROC). The formulas of other parameters are presented in Table 4.

Table 4: Performance parameters used in the experiment.

name	formula
MCC	$\frac{(TP*TN-FP*FN)}{\sqrt{(TP+FP)*(TP+FN)*(TN+FP)*(TN+FN)}}$
F-measure	$\frac{(2*Recall*Precision)}{(Recall+Precision)}$

Experimental details of hardware and software of the study are seen in Table 5. As seen from this table, the experiment has been completed with two programming environments including C# and R package. These platforms have different advantages when they are employed in a software engineering case study. C# is easy to implement. On the other hand, R package has powerful options to conduct a statistical analysis with a machine learning model.

Table 5: Experimental environments.

Feature	**Configuration**
Processor	Intel i3-4005U
RAM	4 Gb
Machine word	64
CPU	1.70 Ghz
Operating System	Windows 8
Programming software	C#, R package

5 Results

Initially, it should be determined that what sort of tests to be used to evaluate the results. First of them is ROC. In this analysis, static code metrics yielded better results than process metrics in CPDP. Figure 5 and 6 illustrate the results. Data sets are composed of the same metrics that this case may have affected the results. Despite the fact that there is not any need for matching metrics to handle with heterogeneity, static code metrics are better than process metrics in terms of AUC if training instances are taken from a clustering method. However, the results need to be tested on some industrial data sets to generalize the bias.

MCC values of two algorithms obtained on static code and process metrics are presented in Table 6. These values are the mean of results recorded in CPDP combinations. During the experiment, all the training data groups have been generated using complexity-based fuzzy clustering. As known from the results, process metrics, which are not superior to static code metrics in AUC, outperformed static code metrics in MCC. The churn in MCC values may have originated from the scale of the data sets and project characteristics.

$$\delta_{ij} = +1 \rightarrow G_1 > G_2; -1 \rightarrow G_1 < G_2; 0 \rightarrow G_1 = G_2 \tag{3}$$

$$result = \frac{1}{nm} \sum_{i=1}^{m} \sum_{j=1}^{n} \delta_{ij} \tag{4}$$

Table 7 compares F-measure results of all the data sets both in static code and process metrics. Overall, static code metrics produced worse results than static code metrics except five data sets. In Section 3.2, it is demonstrated that process metrics of the experiments have not a normal distribution. Therefore, F-measure results are exposed to $Cliff'sDelta$ that is one of the non-parametric tests. To conduct this test, a dominance matrix δ is generated. Each value of this matrix emerges by using Equation 3

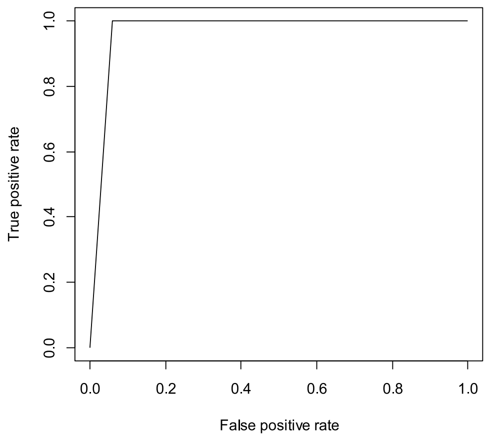

Fig. 5: Average AUC of process metrics of all the data sets. (AUC=0.75)

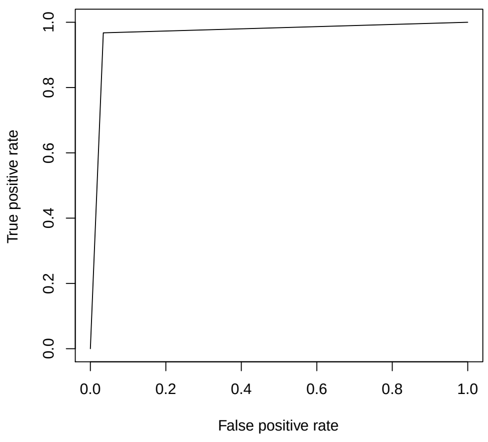

Fig. 6: Average AUC of static code metrics of all the data sets. (AUC=0.96)

of two observation groups. One of three values is added to the matrix by comparing two data groups. If this operation is completed for all values of data groups, this means that matrix is full. In the matrix, n denotes the number of column and m denotes the number of rows. With these parameters, Equation 4 is applied. As the value becomes close to $+1$, the magnitude of the effect increases. This test compares the magnitude of two vectorial result groups. The result of the test is taken with $Cliff'sDelta =$

Table 6: Details of MCC results. The results have been obtained using two distinct clustering algorithms based on fuzzy. First is for static code metrics and the second is designed using NR, NML, and NCDC. The values are the mean of 306 cross-project combinations.

	Static code	**Process metrics**
ant	0.6858126	0.926
camel	0.676	0.91
log4j	0.616	0.89
jedit	0.606	0.9234
ivy	0.641	0.9568
lucene	0.59707496	0.9102
pBeans	0.58111244	0.8236
poi	0.56514992	0.8301
prop1	0.5491874	0.774533333
prop2	0.53322488	0.734483333
prop3	0.51726236	0.824433333
prop4	0.50129984	0.825438333
prop42	0.48533732	0.826443333
prop5	0.4693748	0.827448333
synapse	0.45341228	0.828453333
Velocity	0.43744976	0.829458333
xalan	0.42148724	0.830463333
xerces	0.40552472	0.831468333

$0.4691358 (medium)$. Note that the magnitude of two groups is not high but significant.

6 Threats to validity

In this section, elements that threat the validity are presented by comparing the challenges encountered during the experiment. Firstly, the values of NR, NML, and NDC of data sets are incomplete. In some instances, values are not specified. The method proposed in this study works with numeric values. Thus, missing values are assigned to ”0” to disregard incompleteness. Because, values to be assigned should not affect the statistical analyzes of metric values.

Second, the experiment is focused on three different process metrics. The corresponding metrics do not include any other process metric. Although this case creates a threat for the validity, as the number of parameters that affects membership function of fuzzy clustering increases, the algorithm becomes more complex. To address this problem, if a great number of metrics are available, they can be minimized with conducting a feature selection operation.

Throughout the paper, process and static code metrics are compared over two versions of fuzzy clustering. Success rates of two different clustering are evaluated in CPDP. Concretely, the generality of the results can be validated by applying various clustering methods. All the data sets have same metric set. Thus, despite having ho-

Table 7: Mean F-measure values of four predictors involving Bayes, naiveBayes, Random Forest, and J48 on 29 data sets. Values have been generated employing 306 cross combinations. complexFuzzy are used to generate testing data. Boldfaced values are the best in related rows.

Data Set	Process metrics	Static code metrics
ant	**0.53**	0.50
camel	0.57	**0.57**
log4j	**0.59**	0.57
jedit	0.62	**0.64**
ivy	**0.71**	0.67
lucene	0.77	**0.79**
pBeans	0.78	**0.78**
poi	**0.79**	0.65
prop1	**0.79**	0.65
prop2	**0.81**	0.6
prop3	**0.83**	0.61
prop4	0.77	**0.79**
prop42	**0.78**	0.7
prop5	**0.76**	0.68
synapse	**0.74**	0.7
velocity	**0.78**	0.74
xalan	**0.81**	0.75
xerces	**0.83**	0.56

mogeneous metrics facilitates conducting the experiment, the fluctuations of the results should be observed in heterogeneous data sets.

7 Conclusion and future remarks

In this study, a fuzzy-clustering algorithm is proposed for selecting training data of CPDP. The algorithm updates the values of membership function based on the complexity of process metrics. The experiment including 18 data sets evaluates the performance of four different classifiers in three performance parameters.

In summary, while static code metrics produce better results in AUC, process metrics surpassed static code metrics in F-measure and MCC. In the experiment, 10*10 cross-validation is performed in other data sets after training data group of each data set is selected. Process metrics yielded high CPDP results especially in xerces and prop3 data sets. In general, process metrics outperformed static code metrics in training data groups that are selected with fuzzy clustering. Furthermore, any linear model was not found in the process metrics used in the experiment. It is expected that this work will close significant gap in CPDP with regard to the training instance selection.

In future works, the comprehensiveness of complexity formula utilized in fuzzy-clustering may be extended by adding new process metrics. Last, proposed formula is planned to adapt to micro-interaction and ownership metrics.

References

Abrahamsson, P., Salo, O., Ronkainen, J., Warsta, J. (2017). Agile software development methods: Review and analysis. arXiv preprint arXiv:1709.08439.

Ashfaq, R. A. R., He, Y. L., Chen, D. G. (2017). Toward an efficient fuzziness based instance selection methodology for intrusion detection system. International Journal of Machine Learning and Cybernetics, 8(6), 1767-1776.

Bailey, C. T., Dingee, W. L. (1981). A software study using Halstead metrics. ACM SIGMETRICS Performance Evaluation Review, vol. 10(1), pp. 189-197.

Clarke, P., OConnor, R. V., Leavy, B. (2016). A complexity theory viewpoint on the software development process and situational context. In Software and System Processes (ICSSP), 2016 IEEE/ACM International Conference, pp. 86-90.

Curtis, B., Sheppard, S. B., Milliman, P., Borst, M. A., Love, T. (1979). Measuring the psychological complexity of software maintenance tasks with the Halstead and McCabe metrics. IEEE Transactions on software engineering, vol. 2, pp. 96-104.

Foucault, M., Falleri, J. R., Blanc, X. (2014). Code ownership in open-source software. In Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering, p. 39.

Gousios, G., Pinzger, M., Deursen, A. V. (2014). An exploratory study of the pull-based software development model. In Proceedings of the 36th International Conference on Software Engineering, pp. 345-355.

Gray, D., Bowes, D., Davey, N., Sun, Y., Christianson, B. (2009). Using the support vector machine as a classification method for software defect prediction with static code metrics. In International Conference on Engineering Applications of Neural Networks, pp. 223-234.

Jureczko, M., Madeyski, L. (2010). Towards identifying software project clusters with regard to defect prediction. In Proceedings of the 6th International Conference on Predictive Models in Software Engineering, p. 9.

Kim, S., Zhang, H., Wu, R., Gong, L. (2011). Dealing with noise in defect prediction. In Software Engineering (ICSE), 2011 33rd International Conference, pp. 481-490.

Layman, L., Kudrjavets, G., Nagappan, N. (2008). Iterative identification of fault-prone binaries using in-process metrics. In Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement, pp. 206-212.

Lee, T., Nam, J., Han, D., Kim, S., In, H. P. (2016). Developer micro interaction metrics for software defect prediction. IEEE Transactions on Software Engineering, vol. 42(11), 1015-1035.

Li, Y., Huang, Z., Wang, Y., Fang, B. (2017). Evaluating Data Filter on Cross-Project Defect Prediction: Comparison and Improvements. IEEE Access, 5, pp. 25646-25656.

Li, M., Zhang, H., Wu, R., Zhou, Z. H. (2012). Sample-based software defect prediction with active and semi-supervised learning. Automated Software Engineering, vol. 19(2), pp. 201-230.

Madeyski, L., Jureczko, M. (2015). Which process metrics can significantly improve defect prediction models? An empirical study. Software Quality Journal, vol. 23(3), pp. 393-422.

McIntosh, S., Kamei, Y., Adams, B., Hassan, A. E. (2016). An empirical study of the impact of modern code review practices on software quality. Empirical Software Engineering, vol. 21(5), pp. 2146-2189.

Mendel, J. M. (2017). Uncertain rule-based fuzzy systems. Springer, Cham CrossRef MATH Google Scholar.

Menzies, T., Dekhtyar, A., Distefano, J., Greenwald, J. (2007). Problems with Precision: A Response to" comments on'data mining static code attributes to learn defect predictors'". IEEE Transactions on Software Engineering, vol. 33(9), pp. 637-640.

Menzies, T., Greenwald, J., Frank, A. (2007). Data mining static code attributes to learn defect predictors. IEEE transactions on software engineering, vol. 33(1), pp. 2-13.

Menzies, T., Milton, Z., Turhan, B., Cukic, B., Jiang, Y., Bener, A. (2010). Defect prediction from static code features: current results, limitations, new approaches. Automated Software Engineering, 17(4), 375-407.

Moser, R., Pedrycz, W., Succi, G. (2008). A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. In Proceedings of the 30th international conference on Software engineering, pp. 181-190.

Nagappan, N., Williams, L., Osborne, J., Vouk, M., Abrahamsson, P. (2005). Providing test quality feedback using static source code and automatic test suite metrics. In Software Reliability Engineering, 2005. ISSRE 2005. 16th IEEE International Symposium, pp. 10-94.

ÖZTÜRK, M. M., complexFuzzy: A novel clustering method for selecting training instances of cross-project defect prediction, unpublished.

Pter, G. (2017). Automatic calculation of process metrics and their bug prediction capabilities. Acta Cybernetica, vol. 23(2), pp. 537-559.

Rahman, F., Devanbu, P. (2013). How, and why, process metrics are better. In Software Engineering (ICSE), 2013 35th International Conference on pp. 432-441.

Romano, D., Pinzger, M. (2011). Using source code metrics to predict change-prone java interfaces. In Software Maintenance (ICSM), 27th IEEE International Conference, pp. 303-312.

Shihab, E., Jiang, Z. M., Ibrahim, W. M., Adams, B., Hassan, A. E. (2010). Understanding the impact of code and process metrics on post-release defects: a case study on the eclipse project. In Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, page 4.

Stanic, B., Afzal, W. (2017). Process Metrics are not Bad Predictors of Fault Proneness. In The 2017 IEEE International Workshop on Software Engineering and Knowledge Management SEKM'17, 25 Jul 2017, Prague, Sweden, pp. 493-499.

Xia, X., Lo, D., Pan, S. J., Nagappan, N., Wang, X. (2016). Hydra: Massively compositional model for cross-project defect prediction. IEEE Transactions on software Engineering, vol. 42(10), pp. 977-998.

Wang, S., Liu, T., Tan, L. (2016). Automatically learning semantic features for defect prediction. In Proceedings of the 38th International Conference on Software Engineering, pp. 297-308.

Wiese, I. S., Cgo, F. R., R, R., Steinmacher, I., Gerosa, M. A. (2014). Social metrics included in prediction models on software engineering: a mapping study. In Proceedings of the 10th International Conference on Predictive Models in Software Engineering, pp. 72-81.

Yu, Q., Jiang, S., Qian, J. (2016). Which Is More Important for Cross-Project Defect Prediction: Instance or Feature?. In Software Analysis, Testing and Evolution (SATE), International Conference, pp. 90-95.

Received June 9, 2018 , revised December 17, 2018, accepted January 27, 2019