Research Article

Software Metrics and tree-based machine learning algorithms for distinguishing and detecting similar structure design patterns

Mohammad Y. Mhawish[1] · Manjari Gupta[1]

Received: 24 September 2019 / Accepted: 28 November 2019 / Published online: 3 December 2019

Abstract
Design patterns are general reusable solutions for recurrent occurring problems. When software systems become more complicated due to the lack of documentation of design patterns in software and the maintenance and evolution costs become a challenge. Design pattern detection is used to reduce the complexity and to increase the understandability of the design in the software. In this paper, we propose a design pattern detection approach based on tree-based machine learning algorithms and software metrics to study the effectiveness of software metrics in distinguishing between similar structural design patterns. We build our datasets using P-MARt repository by extracting the roles of design patterns and calculating the metrics for each role. We used parameter optimization techniques based on the Grid search algorithm to define the optimal parameter of each algorithm. We used two feature selection methods based on a genetic algorithm to find features that influence the most in the distinguishing process. Through our experimental study, we showed the effectiveness of machine learning and software metrics when distinguishing similar structure design patterns. Moreover, we extracted the essential metrics in each dataset that supported the machine learning model to take its decision. We presented the detection conditions for each role in the design pattern by extracting them from the decision tree model.

Keywords Design pattern detection · Classification · Feature selection · Explain predictions · Parameter optimization

1 Introduction

Design patterns are defined as the general reusable solutions for the recurrent occurring problems in the software design. Design patterns are useful for providing more understandability for the software by increasing abstraction, and that leads to improving the evolution and maintainability of the software [1]. Due to some reasons, including the lack of software documentation, the design pattern causes complexity in the software, and that leads to increased evolution and maintenance efforts [2]. From this perspective, the researchers started to use design patterns in reverse engineering, and several studies have been published in terms of detecting design patterns from software systems [3].

From the proposed studies published in the field of design pattern detection, we found that one of the most impact challenges faced the detection process is the similarity of the structure between design patterns [4]. Whereas the detection results contain some false positive instances that detected because that only shared the structure of another design pattern, but it is different in the intents. The similarity of the design pattern structures occurs between design pattern components (design pattern roles) and the relationships between these components.

In the literature, there are several approaches proposed based on machine learning. Ferenc et al. [5] proposed an approach to detect design patterns. They used machine learning techniques in the second phase of

✉ Mohammad Y. Mhawish, bniyaseen@gmail.com; Manjari Gupta, manjari_gupta@rediffmail.com | [1]DST-CIMS, Banaras Hindu University, Varanasi, India.

their approach for filtering the candidates that generated from the first phase-which depends on the structural matching of design patterns. They built the learning dataset from the information structural information collected for each design pattern. They have employed the neural networks with backpropagation and the C4.5 decision tree classifiers to classify the candidate as true or false. Guéhéneuc et al. [6] they proposed an approach based on constraints and machine learning algorithms. The datasets built based on P-MART repository by extracting 13 software metrics for each class in the design pattern components. They applied machine learning algorithms in the first phase to reduce the search space in order to reduce the number of false-positive instances. They employed the JRIP algorithm and rule-learner in their approach. Chihada et al. [7] proposed an approach based on machine learning. The training datasets are built by calculating 45 software metrics for each role in the design pattern component. They supposed that each design pattern is a combination of four roles and the feature vector for each pattern is $4 * 45 = 180$ features that represent the labeled dataset. They used Simple Logistic, C4.5, KNN, SVM to classify patterns from source code. Tirkey et al. [8] proposed detection approach based on machine learning. In their study, they used software metrics and classification techniques to detect software design patterns. They build datasets by extracted software metrics in order to learn classifiers. They evaluated their work using three open source software systems JHotDraw, QuickUML, and JUnit.

In this paper, we proposed an approach to study the effectiveness of using machine learning techniques and software metrics in order to distinguish between similar structural design patterns. We conducted our experiment using Adapter Object and Command design patterns. Moreover, we have explored the decision of machine learning models by using the explain predictions algorithm and decision tree detection rules. The explanation for the predictions aimed to investigate the behavior of machine learning models in the distinction between similar roles in design patterns and defined the software metrics that supported the models to take its decisions in order to create a definition of the difference between the similar structure design pattern roles, based on the software metrics values.

We used three tree-based machine learning algorithms: decision tree algorithm, gradient boosting tree algorithm, and random forest algorithm. We used two feature selection methods based on genetic algorithm. We found the most impact features in each dataset to increase the performance and to increase the knowledge of the software metrics that play a significant role in distinguishing between similar roles in design patterns.

We tuned the hyper-parameters for each machine learning algorithm by using the parameter optimization technique based on the Grid search algorithm. We also presented the detection conditions for each similar design pattern roles by finding the threshold values for each software metric on which the decision tree algorithm relied to distinguish between those similar roles.

The paper is organized as follows: in Sect. 2, we introduce related work. In Sect. 3, we proposed the research methodology and the proposed design pattern detection method. In Sect. 4, we reported the experimental results and discussions. In Sect. 5, we present a conclusion.

2 Related work

In the following, we introduce some of the best design pattern detection approaches based on static and dynamic analysis proposed by different researchers working in this field.

Kramer et al. [9] proposed an approach based on Prolog rules detection, and they used the Prolog rules to recover design patterns from the design information extracted from C++ code. This approach represents a system as a set of Prolog predicates based on [1] definition. Thus, Prolog queries are applied to detect an instance of design patterns. They developed an approach as a tool for detecting design patterns-PAT. Keller et al. [10] proposed an approach for design pattern detection that splits the detection process into two phases. In the first phase, the UML diagram presented to the user by identifying the design pattern structure. In the second phase, they used user interaction to refine the result. Shi and Olsson [3] proposed an approach for design pattern detection based on a structural and behavioral aspect of design patterns. Behavioral aspect detection conducted by analyzing the abstract syntax tree (AST) to produce the Control Flow Graph (CFG) for all elements in the source code. The structural aspect of design patterns extracted for the source code system to generate inter-class relationships. Finally, they used the analyzed information to detect design patterns. Dong et al. [11] proposed an approach for detecting design patterns. They represented a system structure in matrix and weight. They used three analysis phases: structural, behavioral, and semantic. The structural aspect of the source code is extracted to build a matrix. The matrix represented all classes in the source code in its rows and columns; the relationships between classes represented as values in the matrix. Design patterns were also represented as matrices and weight. The weight value for each class is calculated as a number of attributes, a number of methods, and association, generalization, dependency, and aggregation relationships in a particular class. The pattern candidates are detected by matching the system matrix and the design pattern matrices. Depending on this approach they developed a design pattern detection tool-DP-MINER. Kaczor et al. [12] proposed an approach that formulates the design pattern detection as a combinatorial problem by proposing a bit-vector algorithm based solution. They expressed the pattern detection with operation on finite sets of bit-vectors. Design patterns and source code candidates are represented as string expressing classes and their relationships (aggregation, composition, inheritance, association, and instantiation). The detection process in this approach is applied by matching the string representation of design patterns and the analyzed source code. Frence et al. [5] proposed an approach based on machine learning to improve the results of structural matching methods by reducing the false positive. They used machine learning to refine the pattern detection results that returned by the matching algorithm. The detection process consists of the first transformation of the code into an Abstract Semantic Graph (ASG), then matching with the definition of design patterns. Next, manual inspection is used to examine the source code to decide the true or false hits of design pattern instance candidates. Finally, they performed the training of a machine learning system. In this approach, two popular approaches in the field of machine learning presented (a decision tree and a neural network). Tsantalis et al. [13] they have proposed a graph-based approach for detecting design pattern. This approach computes the similarity scoring between vertices of the graph. They used a class diagram for building a direct graph that is mapped into a square matrix representation. A similarity scoring algorithm is used to match system matrix and design patterns matrices. The result after matched is the amount of matching between the system matrix and design pattern matrices. Lucia et al. [14] proposed an approach to detect behavioral patterns which combine both static and dynamic analysis. In this approach, the structural aspects are captured using static analysis, and then a dynamic analysis is conducted on these candidates by tracing the cells of the methods at run-time. Heuzeroth et al. [15] proposed an approach where first analysis of the source code is done by extracting an Abstract Syntax Tree (AST) from source code. Then static information of source code is used to produce a set of candidates that matched with the design pattern static rules. These candidates were used as an input of the dynamic analysis phase. The detection process in the dynamic analysis is performed by matching the candidate rules and design pattern rules. Depending on matching the candidates are accepted or rejected. Hayashi et al. [16] they have described an integrated approach based on static and dynamic analysis. The design facts (classes, methods, etc.) extracted using static analysis and represented as facts in Prolog. The detection process in this approach is performed by matching the facts with a set of conditions that satisfy design patterns. These conditions are defined as Prolog rules. Finally, they extract the execution record from executing the source code.

3 Research methodology and the proposed approach

In this approach, we build a framework to distinguishing between similar structural design patterns and extract the detection conditions based on software metrics and based tree machine learning algorithms. The list of steps we followed is shown in Fig. 1. The following steps are performed to build the solution framework:

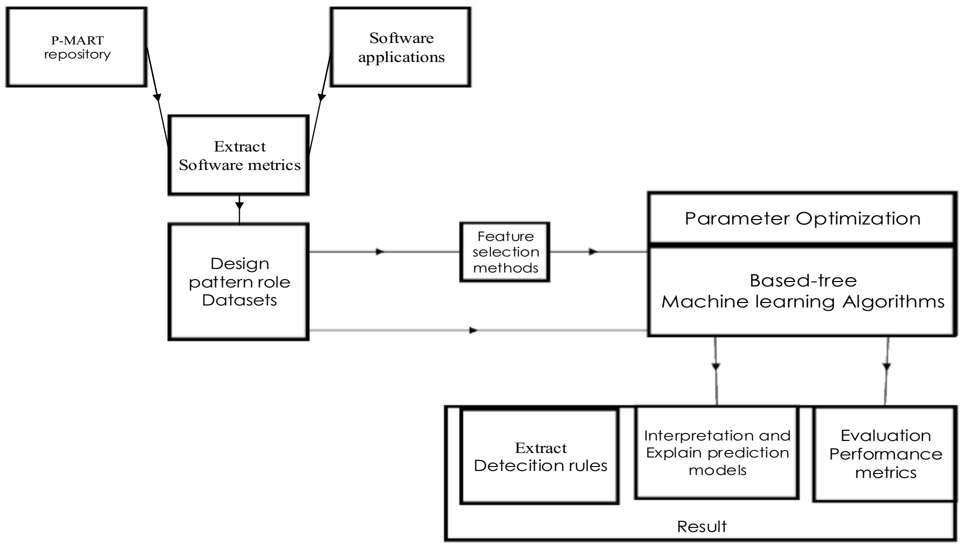
P-MART repository
Software applications
Extract Software metrics
Design pattern role Datasets
Feature selection methods
Parameter Optimization
Based-tree Machine learning Algorithms
Extract Detection rules
Interpretation and Explain prediction models
Evaluation Performance metrics
Result

Fig. 1 Proposed approach

- Parsing P-MARt repository to extract design pattern roles.
- Extracting software metrics for each role to build the datasets.
- Applying feature selection methods to select the relevant metrics in each dataset.
- Applying the parameter optimization technique to tune the hyperparameter for each machine learning algorithm.
- Training machine learning model.
- Extract the result.

3.1 Dataset description and representation

In this work, we created our datasets using PATTERN-LIKE MICROARCHITECTURE REPOSITORY (P-MARt) [17]. The repository contains the list of design patterns that detected from nine open-source software and manually reviewed by experts. We extracted the design patterns by developing XML parser tools.

We selected two design patterns that have a similar structure (Adapter and Command design patterns). Software metrics are calculated for each role in design patterns using JBuilder [18]. We have calculated 36 software metrics that cover size, cohesion, complexity, coupling, and inheritance for object-oriented class. The final datasets are created by merging similar roles in the different design patterns. In the case of Adapter and Command design patterns, there are two mains roles extracted for each pattern, a Receiver and the concreteCommand in case of Command design pattern, and Adapter and Adaptee roles in case of Adapter design pattern. Later we merged the similar structures roles in the same dataset in order to distinguish between them. We produced two datasets: Adapter/ConcreteCommand and Adaptee/Receiver datasets. Each example in the dataset are labelled with the role name, and each role consists of the feature vector of 36 software metrics. Tables 1 and 2 presents the statistics of datasets and the software metrics we calculated for each role, respectively.

3.2 Feature selection

Feature selection is a technique that aims to find the most impact features in the dataset by removing the redundant features in order to increase the performance and to increase the knowledge of the software metrics that play a significant role in distinguishing between similar roles in design patterns [19].

In this paper, we used the Genetic algorithm [20] to build two feature selection methods GA-Naïve Bayes and

Table 1 Dataset statistics

Dataset	Design pattern role	Number of instances	Fraction
Adapter/concreteCommand	Adapter	326	0.652
	ConcreteCommand	174	0.348
Adaptee/receiver	Adaptee	452	0.904
	Receiver	48	0.096

Table 2 Software metrics

Acronyms	Full Name	Acronyms	Full Name
AC	Attribute complexity	NOA	Number of attributes
AOFD	Access of foreign data	NOC	Number of classes
AUF	Average use of interface	NOCC	Number of child classes
CBO	Coupling between objects	NOCON	Number of constructors
CC	Cyclomatic complexity	NOED	Number of external dependencies
CL	Class locality	NOM	Number of members
CM	Changing methods	NOO	Number of operations
COC	Clients of class	NOOM	Number of overridden methods
ChC	Changing classes	NOPA	Number of public attributes
DAC	Data abstraction coupling	NORM	Number of remote methods
DD	Dependency dispersion	RFC	Response for class (RFC)
DOIH	Depth of inheritance hierarchy	TRAp	Total reuse of ancestor percentage
EC	Essential complexity	TRDp	Total reuse in descendants percentage
FO	FanOut	TRAu	Total reuse of ancestor unitary
MDC	Module design complexity	TRDu	Total reuse in descendants unitary
MIC	Method invocation coupling	WCM	Weighted changing methods
MNOL	Maximum number of levels	WMPC	Weighted methods per class
NAM	Number of accessor methods	WOC	Weight of a class

GA-CFS. The best features in those methods are selected based on the fitness functions we used: the degree of correlated features to the target class (CFS) [21] and the accuracy of training Naïve Bayes algorithm.

Figure 2 displays the genetic algorithm used to build feature selection methods. Initially, the algorithm generates an initial population with randomly selected features. These instances will be examined by the fitness function then tested the stopping criteria. If this solution not satisfied, a genetic algorithm operator will be applied and examine again by a fitness function to generate the optimal metrics for the dataset.

In the first method, Genetic algorithm based on Naïve Bayes (GA-Naïve Bayes), we have set the population size to 20 individuals with the maximum number of generations 40. We used non-dominated sorting to apply the chromosomes selection. We also used the one-point crossover to apply the cross-over operation. The cross over probability is set to 0.95, and the mutation probability is set to 0.1. The best generation of chromosomes is that they have the highest performance accuracy generated by training Naïve Bayes using tenfold cross-validation.

In the second method that is Genetic algorithm based CFS (GA-CFS), we have set the population size to 20 individuals with the maximum number of generations 100. The tournament selection method is used to apply the chromosomes selection. We used One-point crossover with probability 0.95 and mutation probability is set to 0.1. The best generation of chromosomes is that they have the highest degree of correlated features in the class. The high degree of correlation feature returns low CFS performance value, and a low degree of correlation returns high CFS performance value.

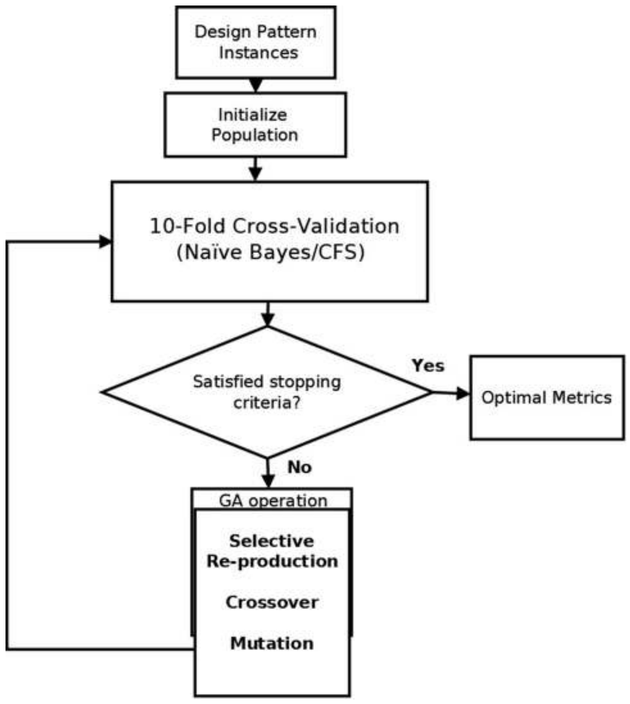

Fig. 2 Proposed feature selection method

3.3 Parameter optimization

In the machine learning algorithms, there are hyper-parameters that have to tune to ensure the improvement of the performance accuracy of the algorithms. In this paper, we used a grid-search algorithm based on parameter optimization technique. The grid search is an exhaustive search algorithm based on a defined set of values of parameters by defined upper and lower bound for each parameter and based on the assigned number of steps the parameter values for each parameter are assigned. Then the Grid search algorithm is testing every combination of parameters value to calculate the best value of parameters for each algorithm [22. 23]. In the tree-based algorithm, several parameters impact the performance accuracy. Table 3 shows the parameters that are selected to tune their values and value range and number of steps for each parameter.

3.4 Classification Models

In this paper, we used tree-based algorithms to build a design pattern roles detection approach.

The Tree-based algorithm is a supervised learning algorithm that is considered as one of the most accurate

Table 3 Tuning algorithms parameters and the specified steps assigned for each parameter

Model	Parameter	From	To	Steps
Random forest	Number of trees	1	100	10
	maximal depth	1	100	10
Gradient boosted trees	Number of trees	1	200	10
	Maximal depth	1	100	10
Decision tree	maximal depth	1	100	10
	Criterion	Gain ratio, information gain, Gini index, accuracy		

algorithms in the classification process. Tree-based algorithms map the nonlinear relationships among the instance features and the target classes in the dataset. Moreover, the algorithm provides classification rules that can be ease to interpret the classification models [24, 25].

In our approach, we use three tree-based algorithms: Decision Tree algorithm, Gradient Boosted Trees algorithm (GBT), and Random Forest algorithm.

The decision tree algorithm is a supervised learning algorithm that consists of nodes split based on splitting rules for each specific feature [26]. In the decision tree, the data is passed from root to leaves, and the feature value is separated based on splitting rules in each node according to predictor class until it reaches the stopping criteria. The hyperparameters of the decision tree algorithm, maximal depth and split criterion, are optimized using parameter optimization techniques. The Random Forest algorithm is an ensemble technique that consists of many decision trees. The ensemble technique used in the random forest algorithm is bagging. Bagging is used to reduce the variance of the classifier model by combining the results of multiple models that are trained in different sub-set of the training dataset to produce strong classifier model. The Gradient boosted trees (GBT) is a decision tree based algorithm that combines the boosting ensemble technique along with gradient-based optimization [27]. Boosting is an ensemble technique consisting of many decision trees that are trained sequentially by changing the weight of the dataset examples based on the classification error that is generated by the previously trained model until most of the examples in the dataset classified correctly.

The hyperparameters of the Gradient boosted trees and random forest algorithms, the number of trees and maximal depth of trees, are optimized using parameter optimization techniques.

3.5 Validation Methodology

3.5.1 Performance measures and evaluation parameters

In this study, a set of experiments we have done. In each experiment, five performance measuring parameters such as precision, recall, f-measure, AUC, and accuracy are used for measuring the performance of design pattern detection models. These parameters are computed using a confusion matrix that contains actual and predicted classification information identified by design pattern detection classifiers [28].

We give brief definitions of the performance parameters used for measuring the performance of the design pattern detection model as follows.

Accuracy is one of the performance measures for classification. It is the percentage of correctly classified instances in the positive and negative class calculated as follows:

$$\text{Accuracy (AC)} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

The accuracy of the design pattern detection approach is usually determined by the relationship between precision and Recall. Ideally, a reasonable approach should have good precision and recall rate, i.e., while Recall values increase the Precision values should remain high. Thus, we infer that a useful approach should have a high rate of true positives with a low rate of false positives and false negatives.

F-measure is defined as the harmonic mean of precision and recall, while the Precision is the positive-classified instances that are positive. The recall is the real-positive instances classified as positive. F-measure is a way of having a single number combining the two measures calculated using the following formulas.

$$\text{F-measure } = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \%$$

Area under the ROC Curve (AUC) AUC is one of the common measures of accuracy for classification models by computing the area under the Receiver Operating Characteristic (ROC) curves. ROC curves are a way to visualize the tradeoffs between true positive rate and false-positive rate in a classifier to analyze and compare the performance of the classifier models through visual analytics [29].

3.5.2 Models explanation

In supervised learning, performance accuracy could not be enough to evaluate the algorithm, especially when dealing with software engineering problems. The performance measures not give much comprehensive of how the model takes its decision and what the features that influenced in this decision [30].

In this paper, we used two types of explaining prediction models, the decision tree rules and LIME Algorithm [31], to provide more understanding for the interpretation of the model results and to find how each software metric affects the model predictions. In the decision tree model, the interpretation is conducted by starting from root among the nodes to reach the leaf nodes through edges, and each edge leads to which subsets of prediction by following decision rules that considered the threshold of metrics value that decide the decision path. All edges that passed through to reach to the leaf are connected with the AND operator as the following:

If metric X is [smaller/bigger] than c AND metric Y is [smaller/bigger] than d, then the model predicted W prediction. The Local interpretable model-agnostic explanations (LIME) [31] is used to explain the predictions of black-box models by explaining why each prediction was made, and what the most contributory metrics that effect to the decision of the prediction model. The LIME algorithm generates a set of data points around each software metric in the dataset, and train the model in these datasets and observing the change of the result to approximate the weight of software metric and its importance in changing the result. The weight of each metric is generated by calculating the correlation between each data point and the changing of the result.

4 Experimental results and discussions

In this experiment, we evaluate tree-based machine learning and feature selection methods using two datasets of similar design patterns; Adapter/concreteCommand and Adaptee/Receiver datasets. A tenfold Cross-validation is a technique used to evaluate machine learning models with ten times repetitions.

4.1 Machine learning algorithms results

In Table 4, we presented the set of software metrics selected by feature selection methods. In the Adapter/concreteCommand dataset, the GA-CFS and GA-Naïve Bayes methods selected 7 and 8 metrics respectively out of 36 metrics. In the Adaptee/Receiver dataset, the feature selection methods selected 7 and 6 metrics out of 36 metrics by GA-CFS and GA-Naïve Bayes methods, respectively. The main aim of applying feature selection methods is not only for improving the performance accuracy of classifiers but also to know the relevant software metrics that contribute to distinguish between particulars roles in similar structural design patterns.

Table 5 shows the evaluation results of machine learning algorithms in design-pattern-roles classification models. We evaluated the models using Adapter/concreteCommand and Adaptee/Receiver datasets with the selected features by feature selection methods and with all features.

As observed in Table 5 and Fig. 3, the feature selection methods impact the accuracy of the models by improving the accuracy of the models and by choosing the most influenced features that play a key role in the classification process.

In the Adapter/concreteCommand dataset, the Random Forest algorithm scored the best accuracy value by 96.86% when evaluated by the feature selected dataset generated

Table 4 Metrics selected using feature selection methods

Dataset	GA-Naïve Bayes	GA-CFS	Selected metrics by GA-Naïve Bayes	Selected metrics by GA-CFS
Adapter/concreteCommand	8	7	DOIH MNOL NOA NOOM TRAp TRAu WOC	CL EC NOOM NORM TRAp TRAu WOC
Adaptee/receiver	6	7	AUF AUF CC ChC MIC MNOL WCM	AC CM ChC DOIH NOPA TRAp WOC

Table 5 Evaluation results of machine learning algorithms

Model	Measures	Adapter/concreteCommand			Adaptee/receiver		
		All features (%)	GA-Naïve Bayes (%)	GA-CFS (%)	All features (%)	GA-Naïve Bayes (%)	GA-CFS
Decision tree	Accuracy	92.76	92.76	**94.49**	90.95	92.86	**94.05**
	F-measure	94.53	94.53	95.72	92.20	93.87	94.10
	AUC	87.40	87.40	85.80	87.10	90.10	92.20
Random forest	Accuracy	96.03	94.42	**96.86**	94.29	94.29	94.29
	F-measure	96.96	95.65	97.49	93.55	94.56	95.12
	AUC	99.10	98.30	98.80	92.10	92.10	93.50
GBT	Accuracy	**96.79**	94.42	95.32	94.29	**95.71**	**95.71**
	F-measure	97.46	95.78	96.30	93.58	94.45	95.72
	AUC	99.20	96.50	98.30	96.30	95.50	97.10

Bold indicates the best accuracy among all datasets that achieved in each model

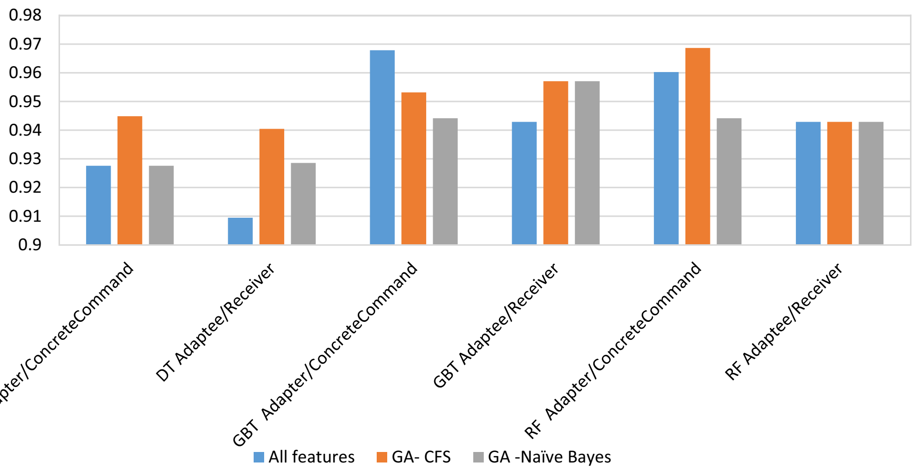

Fig. 3 Comparison of accuracy of all classifiers with using feature selection method and without

by GA-CFS. This accuracy was achieved by optimizing the parameter values for the Random Forest algorithm using the grid search algorithm. The hyper-parameter values for the Random forest algorithm have set as Number of trees = 21 and Maximal depth = 21.

The GBT algorithm also scored good accuracy in all feature dataset by 96.79% with the best F-measure by 97.46 and best AUC by 99.20.

We applied the explain Predictions technique using the Lime algorithm in the random forest models that are trained on Adapter/concreteCommand all-features dataset. We found that the most important metrics that support the decision of the model to classify between Adapter and concreteCommand roles are WOC, NOMM, and RFC software metrics that support the model to predict the Adapter roles. And CL and TRAp metrics that support the model to predict concreteCommand roles. We also observed that most of the metrics considered by the Lime algorithm as the supportive metrics to the model to classify the roles were also selected by feature selection method (GA-CFS) as the most relevant features, as shown in Table 4.

In the Adaptee/Receiver dataset, the Gradient Boosted Trees algorithm scored the best accuracy value by 95.71% when evaluated by the feature selected datasets that are generated by GA-CFS and GA-Naïve Bayes. This accuracy was achieved by optimizing the parameter values for the Gradient Boosted Trees algorithm using the Grid search algorithm. The parameter values have set for the number of trees = 10 and the maximal depth = 2.

By applying the Lime algorithm, we extracted the software metrics that support the GBT model in distinguishing between Adaptee and Receiver design pattern roles. We found that the most critical features to detect the Adaptee roles is WOC, AC, ChC, and NOA. On the other hand, the most critical feature that supports the model to detect Receiver roles is CM and DOIH. By comparing the software metrics that were selected by feature selection methods and the metrics that have been considered by the Lime algorithm as supportive metrics of the classification model between design pattern roles, we found that the metrics resulted from both methods are somewhat similar.

4.2 Discussion about software metrics and their effectiveness in distinguishing between similar structure design pattern

The decision tree model provides human-readable detection rules. The detection rules consist of a set of conditions. Each condition is a Boolean logical proposition that decides the threshold of the metric values to detect the design pattern roles. We have extracted the detection rules for the decision tree model that are trained in full feature datasets.

4.2.1 Adapter and concreteCommand Detection Conditions

Based on the detection rules, the Adapter roles in the Adapter design pattern were detected when achieved the following conditions:

```
TRAp ≤ 0.500 && WOC > 82
TRAp > 0.500 && NOOM > 3.500
```

The first combination of conditions, is that If the Total Reuse of Ancestor percentage (measure the degree of reuse of ancestor class variable) is less or equal than 0.50, and the Weight of a Class (measure the number of non-accessor methods divided by the total number of interface method) is higher than 82 then it detects the adapter roles in the dataset.

The second combination of the condition is: if the Total Reuse of Ancestor percentage is higher than 0.50 when the

Number of Overridden Methods is greater than 3 then it detects the remaining adapter roles in the dataset.

The concreteCommand roles in the Command design pattern were detected when achieved the following conditions:

```
TRAp > 0.500 && NOOM ≤ 3.500 && WOC ≤
90
TRAp > 0.500 && NOOM ≤ 2.500 && WOC >
90 && EC > 2.500 && NORM > 3.500
```

The first combination of conditions detects concreteCommand roles, and it shows that concreteCommand roles are distinguished from Adapter roles if the Total Reuse of Ancestor percentage is greater than 0.50 and the Number of Overridden Methods is equal or less than 3 with 90 or less Weight of Class values.

In the second combination of conditions, the concreteCommand roles are detected if Total Reuse of Ancestor percentage is greater than 0.50 and the Number of Overridden Methods is equal or less than 2 with greater than 2.5 of the Essential Complexity and the class have at least 3 Remote Methods.

4.2.2 Adaptee and receiver detection conditions

On the detection rules, that we have extracted from the decision tree model that trained in Adaptee/Receiver all-feature dataset, we determined the set of combination of conditions to detect Adaptee and Receiver roles. Adaptee roles in the Adapter design pattern were detected when achieved the following conditions:

```
WOC > 74.50 && ChC ≤ 23
WOC > 74.50 && ChC > 23 && AC > 0.50
WOC ≤ 74.50 && DOIH > 1.500
```

In the first combination of conditions, most of the Adaptee roles are detected, if the Weight of a Class is greater than 74 and if the number of client-classes where the changes must be operated in result a change in the server-class (ChC) is equal or less than 23. The second combination of conditions detected the Adaptee roles if the Weight of a Class is greater than 74 and the ChC is greater than 23 if the Attribute Complexity of the class is greater than 0.5. The remaining Adaptee roles are detected if the Weight of a Class is equal or less than 74 when the Depth of Inheritance Hierarchy is greater than 1.5.

Receiver roles in the Adapter design pattern were detected when achieved the following conditions:

```
WOC ≤ 74.500 && DOIH ≤ 1.500
WOC > 74.50 && ChC > 23 && AC ≤ 0.50
```

In the first combination of conditions, most of the Receiver roles are detected if the Weight of a Class is equal or less than 74 when the Depth of Inheritance Hierarchy is equal or less than 1.5. The remaining Receiver roles are detected when the Weight of a Class is greater than 74, the ChC is greater than 23 and if the Attribute Complexity of the class is equal or less than 0.5.

5 Conclusion

This approach aims to increase the understanding of the relationship between software metrics and to distinguish between similar structure design pattern. We build our datasets using P-MART design patterns repository [17] by extracting the similar roles in similarly structure design patterns (Adapter and Command) and merge the similar roles of the different design patterns in one dataset, and then we calculate the software metrics for each role.

We used two feature selection methods based on Genetic algorithm methods GA-Naïve Bayes and GA-CFS that aims to remove the redundant features and to increase the knowledge of the software metrics that play a significant role in distinguishing between similar roles in design patterns. Feature selection methods in our experiment reduced the number of features of datasets from 36 to 7 and 8 in the Adapter/concreteCommand dataset using GA-CFS and GA-Naïve Bayes, respectively. Also, reduced from 36 to 7 and 6 features in the Adaptee/Receiver dataset using GA-CFS and GA-Naïve Bayes, respectively. We evaluated and compared the classifier models with using feature selection method and without feature selection methods in the Adapter/concreteCommand and Adaptee/Receiver datasets. We observed that the performance accuracy values are increased when using feature selection methods in most of the algorithms.

In the decision tree models, the highest accuracy is achieved when using GA-CFS feature selection method by increasing from 92 to 94% and from 90 to 94% in the Adapter/concreteCommand and Adaptee/Receiver datasets, respectively. In the random forest algorithm, the highest accuracy is achieved when using GA-CFS feature selection method by 96% and 94% in the Adapter/concreteCommand and Adaptee/Receiver datasets respectively with slightly increased value when comparing the accuracy of the classifier model when used full features datasets. In the GBT algorithm, the highest scored is 96% achieved in the full feature dataset of Adapter/concreteCommand dataset. Also, 95% when using both feature selection methods in the Adaptee/Receiver dataset.

We applied two explain predictions methods: the decision tree roles and LIME Algorithm [31]. Moreover, we presented the most software metrics that influenced the model results for each dataset. We listed the detection rules that are produced from decision tree models and presented the combination of conditions that the decision tree model followed to distinguish between the similar roles in the similar structural design patterns. By using the LIME algorithm, we found the sets of software metrics that support the classifier models to take its decision in the classification process.

In future work, we will study the relationships between some quality problems in software systems and the presence of the design patterns by calculating values for each quality metric and investigate how the presence of design patterns impact of this metrics.

Compliance with ethical standards

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

References

1. Gamma E, Helm R, Johnson R, Vlissides J (1998) Design patterns CD: elements of reusable object-oriented software 47
2. Feitosa D, Ampatzoglou A, Avgeriou P, Chatzigeorgiou A, Nakagawa EY (2019) What can violations of good practices tell about the relationship between GoF patterns and run-time quality attributes? Inf Softw Technol 105:1–16
3. Shi N, Olsson RA (2006) Reverse engineering of design patterns from Java Source code. In: ASE'06 21st IEEE/ACM international conference on automated software engineering, pp 123–134
4. Izurieta C, Griffith I, Reimanis D, Schanz T, Burlington S (XXXX) Structural and behavioral taxonomies of design pattern grime, pp 1–11
5. Ferenc R, Beszedes A (2005) Design pattern mining enhanced by machine learning, 2005 (Icsm'05)
6. Guéhéneuc Y-G, Guyomarc'h Y-G, Sahraoui H (2010) Improving design-pattern identification: a new approach and an exploratory study. Softw Qual J 18(1):145–174
7. Chihada A, Jalili S, Hasheminejad SMH, Zangooei MH (2015) Source code and design conformance, design pattern detection from source code by classification approach. Appl Soft Comput 26:357–367
8. Tirkey A, Rath SK (2018) Software design pattern mining using classification-based techniques, pp. 1–15, 2018
9. Krämer C, Prechelt L (1996) Design recovery by automated search for structural design patterns in object-oriented software. In: Proceedings of the third working conference on reverse engineering, pp 208–215.
10. Keller RK, Schauer R, Robitaille S, Pagé P (1999) Pattern-based reverse-engineering of design components. In: Proceedings of the 21st international conference on software engineering, pp 226–235
11. Dong J, Lad DS, Zhao Y (2007) DP-miner: design pattern discovery using matrix. In: 14th annual IEEE international conference and workshop on the engineering of computer-based systems, pp 371–380
12. Kaczor O, Guéhéneuc YG, Hamel S (2006) Efficient identification of design patterns with bit-vector algorithm. In: Proceedings of the euromicro conference on software maintenance and reengineering, CSMR, pp 175–184
13. Papers R, Variation P (2010) Design pattern detection using similarity scoring.
14. De Lucia A, Deufemia V, Gravino C, Risi M (2009) Behavioral pattern identification through visual language parsing and code instrumentation. In: European conference on software maintenance and reengineering, CSMR, pp 99–108
15. Heuzeroth D, Holl T, Hogstrom G, Lowe W (2005) Automatic design pattern detection. In: 11th IEEE international workshop on program comprehension, pp 94–103, 2003
16. Hayashi S, Katada J, Sakamoto R, Kobayashi T, Saeki M (2008) Design pattern detection by using meta patterns. IEICE Trans Inf Syst E91-D(4):933–944
17. Guéhéneuc Y-G (2007) P-mart: pattern-like micro architecture repository. In: 1st EuroPLoP focus group on pattern repositories, pp 1–3
18. Lanza M, Marinescu R (2007) Object-oriented metrics in practice: using software metrics to characterize, evaluate, and improve the design of object-oriented systems. Springer, Berlin
19. Romero E, Sopena JM (2008) Performing feature selection with multilayer perceptrons. IEEE Trans Neural Netw 19(3):431–441
20. Subbulakshmi T, Ramamoorthi A, Shalinie SM (2010) Feature selection and classification of intrusions using genetic algorithm and neural network, pp 223–234
21. Hall MA (1998) Correlation-based feature subset selection for machine learning. Thesis Submitt. Partial fulfillment Requir. degree Dr. Philos. Univ. Waikato
22. Vapnik V (1998) Statistical learning theory. Wiley, New York
23. Huang CL, Wang CJ (2006) A GA-based feature selection and parameters optimizationfor support vector machines. Expert Syst Appl 31(2):231–240
24. Aha DW, Kibler D, Albert MK (1991) Instance-based learning algorithms. Mach Learn 6(1):37–66
25. Agrawal A, Menzies T (2020) Is AI different for SE?
26. Rokach L, Maimon OZ (2008) Data mining with decision trees: theory and applications, vol 69. World Scientific, Singapore
27. Click C, Malohlava M, Candel A, Roark H, Parmar V (2017) Gradient boosting machine with H_2O. https://www.H2O.Ai/Resources/, no. 6, p. 30
28. Catal C (2012) Performance evaluation metrics for software fault prediction studies. Acta Polytech Hung 9(4):193–206
29. Hand DJ, Till RJ (2001) A simple generalisation of the area under the ROC curve for multiple class classification problems. Mach Learn 45(2):171–186
30. Chakraborty S et al (2018) Interpretability of deep learning models: A survey of results. In: 2017 IEEE SmartWorld, ubiquitous intelligence & computing, advanced & trusted computed, scalable computing & communications, cloud & big data computing, internet of people and smart city innovation. SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI 2017, pp 1–6
31. Ribeiro MT, Singh S, Guestrin C (2016) Why should i trust you? Explaining the predictions of any classifier