DOI: https://doi.org/10.15276/aait.04.2021.5
UDC 004.922

Towards a software defect proneness model: feature selection

Vitaliy S. Yakovyna[1), 2)]
ORCID: https://orcid.org/ 0000-0003-0133-8591; yakovyna@matman.uwm.edu.pl. Scopus Author ID: 6602569305
Ivan I. Symets[2)]
ORCID: https://orcid.org/ 0000-0003-1873-3168; ivan.i.symets@lpnu.ua. Scopus Author ID: 57202453582
[1)] University of Warmia and Mazury in Olsztyn, 2, Oczapowskiego, St. Olsztyn, 10-719, Poland
[2)] Lviv Polytechnic National University, S. Bandery Str. 12, Lviv, 79013, Ukraine

ABSTRACT

This article is focused on improving static models of software reliability based on using machine learning methods to select the software code metrics that most strongly affect its reliability. The study used a merged dataset from the PROMISE Software Engineering repository, which contained data on testing software modules of five programs and twenty-one code metrics. For the prepared sampling, the most important features that affect the quality of software code have been selected using the following methods of feature selection: Boruta, Stepwise selection, Exhaustive Feature Selection, Random Forest Importance, LightGBM Importance, Genetic Algorithms, Principal Component Analysis, Xverse python. Basing on the voting on the results of the work of the methods of feature selection, a static (deterministic) model of software reliability has been built, which establishes the relationship between the probability of a defect in the software module and the metrics of its code. It has been shown that this model includes such code metrics as branch count of a program, McCabe's lines of code and cyclomatic complexity, Halstead's total number of operators and operands, intelligence, volume, and effort value. A comparison of the effectiveness of different methods of feature selection has been put into practice, in particular, a study of the effect of the method of feature selection on the accuracy of classification using the following classifiers: Random Forest, Support Vector Machine, k-Nearest Neighbors, Decision Tree classifier, AdaBoost classifier, Gradient Boosting for classification. It has been shown that the use of any method of feature selection increases the accuracy of classification by at least ten percent compared to the original dataset, which confirms the importance of this procedure for predicting software defects based on metric datasets that contain a significant number of highly correlated software code metrics. It has been found that the best accuracy of the forecast for most classifiers was reached using a set of features obtained from the proposed static model of software reliability. In addition, it has been shown that it is also possible to use separate methods, such as Autoencoder, Exhaustive Feature Selection and Principal Component Analysis with an insignificant loss of classification and prediction accuracy.

Keywords: Software reliability; machine learning algorithms; defect; feature selection; software defect prediction

***For citation*: Yakovyna V. S., Symets I. I. Towards a software defect proneness model: feature selection. *Applied Aspects of Information Technology.* 2021; Vol.4 No.4: 354–365. DOI: https://doi.org/10.15276/aait.04.2021.5**

INTRODUCTION

A software defect is a issue in system components or modules that adversely affects the appearance, performance, functionality, or productivity and may result in failure of certain functionalities or malfunctions of the system. Most software defects occur during software development in the source code of the program and are caused by a number of factors that occur at different stages of the product life cycle, namely: software code defects, communication problems, inaccuracies in software requirements, poor documentation and design, complexity of a system, deadline for completion a project, human factor, insufficient testing, etc.

In order to improve the quality and reliability of software, methods for predicting software defects, are used to identify potential defects. Traditional defect detection methods are based on the analysis of software product metrics and are used to classify potentially defective modules or to predict the approximate number of defects in a particular system module [1]. As a result, the software defects prediction method can help developers identify defects basing on available software metrics using data analysis techniques, and thus improve software quality, which, as a result, reduces software development costs during development and maintenance.

LITERATURE REVIEW

Relevance of the defect prediction process – Defect prediction is a popular area of research and is actively developing as far as it allows reducing the effort and resources required for software testing. This is evidenced by the large number of studies and works in this direction.

In the study [2] a comparative study of different classification algorithms for the classification of software modules into those that tend to defect and not tend to defects has been described. A number of well-known machine learning algorithms, such as Decision Trees, Artificial Neural Network (ANN), K nearest neighbor, SVM and Ensemble Learning have been considered, where the Stacking Ensemble technique proved to be the best with the best result for all data sets with an accuracy of defects prediction greater than 0.9. The study [3] proposes the Deep belief network prediction model (DBNPM), a system for determining whether a software module contains defects. The key idea of DBNPM is the Deep belief network (DBN) technology, which is an effective technique for deep learning in image and natural language processing, the characteristics of which are similar to the defects of the original program.

This study [4] describes the defects prediction based on the reduction of the scale of the two focuses (characteristics of defects and defects). It provides a deep analysis of various key issues including how to create a set of defects characteristics comparison and a set of defects comparison, the repulsion theory for defects and defects characteristics, as well as a methodology and a model for defects prediction. Experimental results demonstrate that this multi-agent predicting methodology is very effective for predicting the quality of space projects software.

The defects prediction models can be built basing on the project data from previous versions or releases, but in the case of a new project there is no such data for prediction available, and because of this, an approach known as cross-project defect prediction (CPDP) is used. The study [5] describes the application of the bandit algorithm (BA) to CPDP in order to select the most suitable training project from a set of projects. The experiment has been performed on two data sets (NASA and DAMB, a total of 12 projects), and it shows that the use of BA to predict defects in CPDP is promising and may surpass existing approaches. The study [6] proposes an algorithm based on TSboostDF transfer learning. TSboostDF integrates the BLS sampling method, which is based on sample weight, with the transfer training method in order to overcome the drawbacks of traditional algorithms used in CPDP.

Also, in addition to predicting module defects, it is important to determine the severity level of software defects, which indicates the affect of an error on the program performance and how quickly these errors should be corrected, as far as setting priorities for these defects manually based on experience can be an inaccurate severity prediction which delays correction of critical errors. In the study [7], methods for automating the assignment of the appropriate severity level based on the results of an error report using various methods of word embedding techniques are described.

Using feature selection in defect prediction – Using feature selection methods is a good solution to the problem of high dimension of data sampling and these methods are actively used in the context of software defect prediction. There is a large amount of research on the feature selection in the defect prediction, the main purpose of which is to choose the most accurate combination of the method of feature selection and the algorithm of machine learning for defects prediction.

In the work [8], 46 methods of defects prediction have been studied using the Decision Tree classifier for 25 sets of software defect data from 4 software repositories. The experimental results have shown that there is no single best FS method, as far as their respective indicators depend on the choice of classifiers, performance evaluation indicators and data set (however, recommendations for the use of methods have been provided).

The research work [9] focuses on two areas: the choice of function and the transfer of instances distance-weight. While reducing the difference between projects in terms of function engineering, a transfer learning technology to build an inter-project model for predicting defects WCM-WtrA and a model with many sources Multi-WCM-WTrA have been introduced. These results show that our method has an average improvement of 23 % compared to the TCA + algorithm for AEEEM datasets and an average improvement of 5 % for ReLink datasets.

In the work [10] 12 automated methods for selecting features for consistency, correlation, performance, computational value, and affect on the size of interpretation have been studied.

Also, along with existing and well-known methods of feature selection, new methods are being actively developed. The hybrid multi-filter wrapper method for feature selection is proposed to select features in the prediction of a software defect that combines the advantages of filtering and wrapping methods [11]. The study [12] proposes to select features using the Firefly algorithm (FA). Firefly algorithm is a new evolutionary computational technique inspired by the firefly flick process. It can quickly search in the function space for an optimal or near-optimal subset of features to minimize a particular suitability function. A new method of wrapping is proposed, which consists of two main stages of selection of features and classification [13]. The first stage uses Grey Wolf Optimization (GWO) to find the best characteristics in the defect identification dataset; the second stage evaluates the characteristics using a machine classifier of reference vectors.

Summing up, we can note that in recent years there has been an increased interest in building static models of software reliability and prediction software defects based on code metrics and project characteristics. As the analysis of the literature showed, the existing approaches do not have high enough accuracy of the prediction; there is no consensus on the impact of software code metrics on its quality indicators, and in particular, reliability; the question of transferability of results obtained on the basis of data on some projects to the characteristics of other software is open, and, accordingly, the question of determining the universal set of metrics of program code that most correlate with its quality indicators remains relevant. Therefore, the aim of this work is to improve the static model of software reliability based on the use of machine learning methods to select the indicators of software code that are most correlated with its reliability.

THE PURPOSE OF THE ARTICLE

This article is aimed at analyzing and improving static models of software reliability using machine learning methods to select the software code metrics that have the strongest affect on its reliability.

To achieve and study this goal, it is proposed to consider the following tasks:

1. Prepare a data set from the PROMISE Software Engineering Repository and combine the test results of 5 NASA Metrics Data Program projects.
2. Analyze the methods of feature selection, apply different methods to a balanced data set and identify features that are important for the model in each of the selected methods.
3. Compare different models of machine learning, such as methods: fandom forest (RF), support vector machine (SVM), k-nearest neighbors (kNN), decision tree classifier (DT), AdaBoost classifier (AB), Gradient Boosting (GB) for analysis of classification with selected code metrics.
4. Choose the best combination of program code metrics basing on the accuracy of each classifier that can be used to effectively predict system module defects.

DATA SET FOR RESEARCH

During the study the data from the publicly available PROMISE Software Engineering Repository have been used. To make the study more efficient, several data sets have been combined using code metrics. The following data have been collected in the NASA Metrics Data Program and they consist of information on the following projects:

Project KC1 is a software system written in C++, which implements resource management on the reception and processing of terrestrial data.

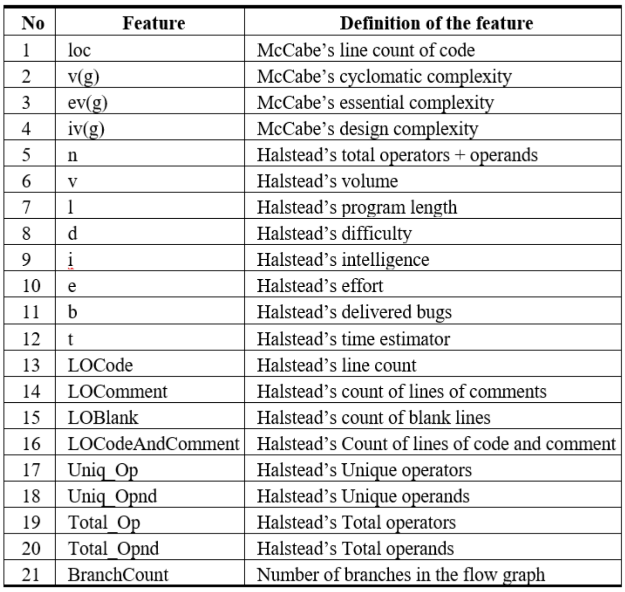
Project KC2 is a software system written in C++ that processes scientific data (the other part of KC1, which is written by another team, and it uses certain common libraries with KC1).

Project PC1 is a module of a software system written in C, for controlling the flight of a satellite in orbit.

Project CM1 is a module written in C to perform certain functions on a spacecraft.

Project JM1 is a system written in C used for predictions based on simulations for terrestrial systems.

The data set for the study consists of 15,123 records, which consist of the 21 code metrics (Fig.1); the target metric in this study will be the *defects* metric, which may contain the value true (module with the specified metrics with a defect) or false (module with the specified metrics without a defect).



No	Feature	Definition of the feature
1	loc	McCabe's line count of code
2	v(g)	McCabe's cyclomatic complexity
3	ev(g)	McCabe's essential complexity
4	iv(g)	McCabe's design complexity
5	n	Halstead's total operators + operands
6	v	Halstead's volume
7	l	Halstead's program length
8	d	Halstead's difficulty
9	i	Halstead's intelligence
10	e	Halstead's effort
11	b	Halstead's delivered bugs
12	t	Halstead's time estimator
13	LOCode	Halstead's line count
14	LOComment	Halstead's count of lines of comments
15	LOBlank	Halstead's count of blank lines
16	LOCodeAndComment	Halstead's Count of lines of code and comment
17	Uniq_Op	Halstead's Unique operators
18	Uniq_Opnd	Halstead's Unique operands
19	Total_Op	Halstead's Total operators
20	Total_Opnd	Halstead's Total operands
21	BranchCount	Number of branches in the flow graph

Fig. 1. **Description of metrics from the selected dataset**

Source: **compiled by the authors**

The data set is unbalanced and contains 2,665 records of defective modules and 12,458 records of modules without any defects.

Table 1 shows the distribution of data on projects.

Table 1. **Data distribution on projects**

Project	Defective modules	Modules without defects	Number of modules
KC1	326	1783	2109
KC2	107	415	522
PC1	77	1032	1109
CM1	49	449	498
JM1	2106	8779	10885
	2 665	**12 458**	**15 123**

***Source:* compiled by the authors**

As noted above, the data set is unbalanced with respect to the value of the defects metric (2,665/12,458). The effect of unbalanced data in machine learning is not evident, i.e. it does not cause an instantaneous error when creating and running a model, but the results can be false. If the degree of class misbalance for the majority class is extraordinary, then a machine-trained classifier can provide high general prediction accuracy, as far as the model is likely to predict the majority of samples belonging to the majority class.

Therefore, for further study, the data have been balanced as follows – a subset of 5,330 records has been selected in which the values of the defects metric have been distributed 50 % to 50 % and further studies have been performed on this data set.

Eighty percent of the data of 5,330 records have been used as a balanced training sample, and twenty percent of the unbalanced data have been used to test the obtained models.

FEATURE SELECTION PROCESS

Functions or features selection (also known as variables selection, attributes selection, or selection of a subset of variables) is the practice of selecting a subset of the corresponding features (predictors and variables) for being used in building a model. It is the automatic selection of attributes present in the data (for example, columns in the tabular data) that is the most relevant and appropriate to the problem of predictive modeling that is being studied.

The purpose of the feature selection in machine learning is to find the best set of functions that allows building useful models of the studied phenomena.

The use of feature selection methods gives us the following advantages at the output:

- a simpler model – simple models are easy to explain and understand – a too complex and incomprehensible model is not valuable;
- shorter learning time – a more precise subset of functions reduces the amount of time required to learn the model;
- decrease of variance – increase of the accuracy of estimates that can be obtained for this modeling;
- lower memory costs during performance.

As far as we have the defects target feature, during this study we are going to consider supervised function selection methods to perform the feature selection process, which in turn are divided into Filter methods, Wrapper methods, and Embedded methods.

Filter methods – These methods are based on probability theory and statistical approaches, and usually consider each feature independently. The main methods in this method class are the following: Chi-square, IG-indexing (information gain calculation), Variance Threshold, Fisher Score, Anova F-value. They assess the importance of features only on the basis of their inherent characteristics, not including any learning algorithm [14]. The usage of these methods is effective if the feature set is very large (one hundred or more), because filtering methods are fast, they may work well as the first step in selecting in order to exclude some variables.

As far as our sampling consists of 21 metrics, the usage of this class of methods will be inefficient for our study for the feature selection method.

Wrapping methods work by estimating a subset of functions by means of a machine learning algorithm that uses a search strategy to view the space of possible subsets of functions, estimating each subset based on the performance quality of this algorithm [15].

Embedded methods perform the process of functions selections within the construction of the machine learning algorithm. In other words, they perform a selection of functions when learning a model, so we call them embedded methods. This class of methods implements the advantageous aspects of the two previous classes of methods. Unlike wrapping methods, which iteratively consider unimportant features based on evaluation metrics, embedded methods perform function selection and algorithm learning in parallel [16].

All research in this work is performed using the Python programming language and mainly the sklearn library (Scikit-learn – one of the most widely used Python packages for Data Science and Machine Learning. It allows performing many operations and provides many algorithms for work).

Before starting using the feature selection methods, it is worth building a data correlation matrix, as it is a powerful tool for generalizing a large data set, as well as for determining and visualizing patterns in the data provided. It is possible to remove from the set of interrelated metrics all but one, without significant loss of information or affect on model quality. The correlation between metrics was determined for the study, there is a significant correlation between the metrics *e* and *t*, as well as between the metrics *b* and *v*, the coefficient of which is 1. The correlation between *n* (Operator and Operand total (Halstead)) and *total_Op* is 1, and between *n* (Operator and Operand total (Halstead)) and *total_Opnd* the correlation coefficient is 0.99. There is also a significant correlation, 0.93, between *loc* and *lOCode*.

Considering these results, it has been decided to discard the following metrics from our dataset: *t, b, total_Op. total_Opnd, lOCode.*

To select features, we have used the following feature selection methods:

Boruta (Using Random Forest Classifier)

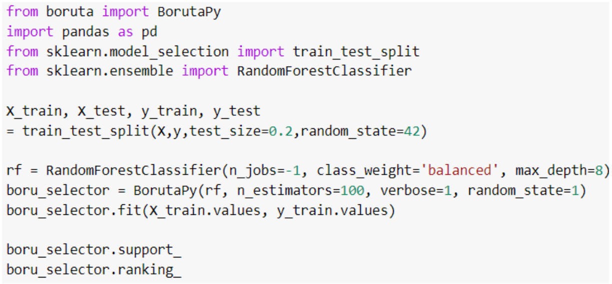
Boruta is a random forest method, so it works for tree models such as Random Forest or XGBoost, but also works with other classification models, such as logistic regression or SVM.

Boruta iteratively removes functions that are statistically less relevant than a random probe (artificial noise variables introduced by Boruta algorithm). The rejected variables in each iteration are removed from consideration in the next iteration. As a rule, this results in a good global optimization of the functions selection [17].

Applying this feature selection method for our sample, we have determined that the following metrics are important for further research (Fig. 2): *loc, v(g), iv(G), n, v, i, e, locCodeAndComment, uniq_Opnd, branchCount.*

Step-wise selection

Step-wise selection is a two-way method, based on a combination of Forward selection and Backward elimination. It is considered less resource-consuming as it reconsiders the possibility of adding predictors back to the model that has been removed (and vice versa). However, considerations are still made based on local optimization at any iteration.



```
from boruta import BorutaPy
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier

X_train, X_test, y_train, y_test
= train_test_split(X,y,test_size=0.2,random_state=42)

rf = RandomForestClassifier(n_jobs=-1, class_weight='balanced', max_depth=8)
boru_selector = BorutaPy(rf, n_estimators=100, verbose=1, random_state=1)
boru_selector.fit(X_train.values, y_train.values)

boru_selector.support_
boru_selector.ranking_
```

Fig. 2. **Implementation and results of the Boruta method**
***Source:* compiled by the authors**

As far as this method is based on sequential addition/subtraction of metrics from the set (it is necessary to specify their exact number for selection), the *plot_sequential_feature_selection* library has been used to visualize the estimate at different values of the *k_features* parameter in order to facilitate decision making (Fig. 3). From the given figures it is visible that the best value of accuracy metrics is at a choice of 13 metrics which will be used further*: loc , ev(g) , iv(G) , n , v , l , i , e, lOComment, locCodeAndComment , branchCount, uniq_Op, uniq_Opnd.*

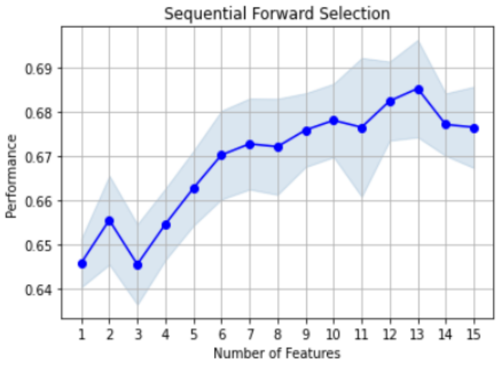
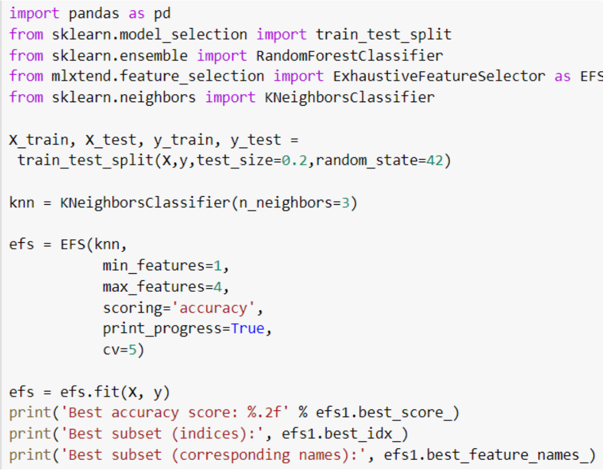


Fig. 3. **Visualization of the estimate at different values of the parameter k_features for the Step-wise method Exhaustive Feature Selection**
***Source:* compiled by the authors**

Exhaustive Feature Selection

In the case of Exhaustive Feature Selection, the efficiency of the machine learning algorithm is estimated on the basis of all possible combinations of functions in the data set. A subset of functions that provides the best performance is selected. Exhaustive search algorithm is the most resource-consuming algorithm of all wrapping methods, because it tests the whole combination of functions and chooses the best one [18].

Figure 4 shows the implementation of the method with the choice of the following features for analysis: *loc, v(g), N, V, I, branchCount.*



```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from mlxtend.feature_selection import ExhaustiveFeatureSelector as EFS
from sklearn.neighbors import KNeighborsClassifier

X_train, X_test, y_train, y_test =
 train_test_split(X,y,test_size=0.2,random_state=42)

knn = KNeighborsClassifier(n_neighbors=3)

efs = EFS(knn,
          min_features=1,
          max_features=4,
          scoring='accuracy',
          print_progress=True,
          cv=5)

efs = efs.fit(X, y)
print('Best accuracy score: %.2f' % efs1.best_score_)
print('Best subset (indices):', efs1.best_idx_)
print('Best subset (corresponding names):', efs1.best_feature_names_)
```

Fig. 4. **Implementation and results of the Exhaustive Feature Selection method**

***Source:* compiled by the authors**

Random Forest Importance

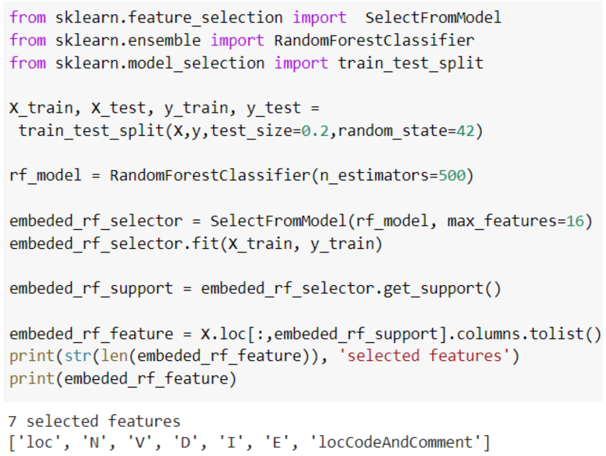
The functions selection by means of a random forest belongs to the category of embedded methods.

Random forests are a kind of data processing algorithm that combines a number of decision trees. Tree-based strategies used by random forests are naturally ranked according to how well they improve the cleanliness of a node, or in other words, reduce impurities (Gini Impurities) over all trees. The nodes with the lowest decrease in impurities occur at the beginning of the trees, while the nodes with the lowest decrease in impurities occur at the end of the trees. Thus, by pruning trees under a certain node, we can create a subset of the most important features [19].

We calculate the importance of the features using node impurities in each decision tree. In a random forest, the ultimate importance of the features is the average value of all the features of the decision tree.

Applying this method of feature selection for our sample, we have determined that the following metrics are important for further research (Fig. 5): *loc, N, V, D, I, E, locCodeAndComment.*

This approach is similar to the method described above (Random Forest Importance), but basically uses the Light GBM algorithm, because it also has the attribute feature_importance. Light GBM is a tree-based learning algorithm, Light GBM grows a tree vertically, while another algorithm grows it horizontally, which means that this algorithm grows on a tree by leaves, while another algorithm grows by level [19].



```python
from sklearn.feature_selection import  SelectFromModel
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test =
 train_test_split(X,y,test_size=0.2,random_state=42)

rf_model = RandomForestClassifier(n_estimators=500)

embeded_rf_selector = SelectFromModel(rf_model, max_features=16)
embeded_rf_selector.fit(X_train, y_train)

embeded_rf_support = embeded_rf_selector.get_support()

embeded_rf_feature = X.loc[:,embeded_rf_support].columns.tolist()
print(str(len(embeded_rf_feature)), 'selected features')
print(embeded_rf_feature)
```

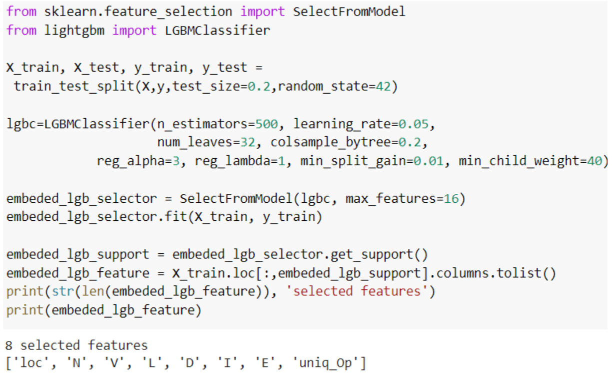
```
7 selected features
['loc', 'N', 'V', 'D', 'I', 'E', 'locCodeAndComment']
```

Fig. 5. **Implementation and results of the Random Forest Importance method**

***Source:* compiled by the authors**

Light Gradient Boosting Machine Importance

Applying this feature selection method for our sample, we have determined that the following metrics are important for further study (Fig. 6): *loc, N, V, L, D, I, E, uniq_Op.*



```python
from sklearn.feature_selection import SelectFromModel
from lightgbm import LGBMClassifier

X_train, X_test, y_train, y_test =
 train_test_split(X,y,test_size=0.2,random_state=42)

lgbc=LGBMClassifier(n_estimators=500, learning_rate=0.05,
                    num_leaves=32, colsample_bytree=0.2,
            reg_alpha=3, reg_lambda=1, min_split_gain=0.01, min_child_weight=40)

embeded_lgb_selector = SelectFromModel(lgbc, max_features=16)
embeded_lgb_selector.fit(X_train, y_train)

embeded_lgb_support = embeded_lgb_selector.get_support()
embeded_lgb_feature = X_train.loc[:,embeded_lgb_support].columns.tolist()
print(str(len(embeded_lgb_feature)), 'selected features')
print(embeded_lgb_feature)
```

```
8 selected features
['loc', 'N', 'V', 'L', 'D', 'I', 'E', 'uniq_Op']
```

Fig. 6. **Implementation and results of the Light GBM method**

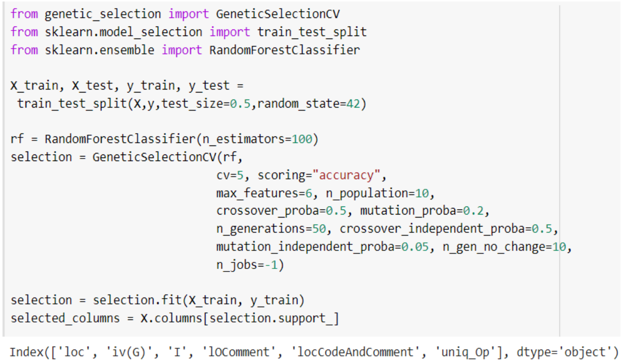
***Source:* compiled by the authors**

Genetic Algorithms

Genetic algorithms are global optimization methods for finding very large spaces. They were inspired by the biological mechanisms of natural selection and reproduction. They work in all populations of possible solutions (so-called generations), where each solution in the search space is represented as a string of finite length (chromosome) over a finite set of symbols, which then uses the target (or suitable) function to estimate the relevance of each solution [20].

In terms of feature selection, each chromosome will represent a subset of features, and it will be represented by binary coding: 1 means “select” a specific feature, and 0 means “do not select” a feature.

The implementation and results of the use of this method are shown in Fig. 7, and the following features are selected for further study: *loc*, *iv(G)*, *I*, *lOComment*, *locCodeAndComment*, *uniq_Op*.



```
from genetic_selection import GeneticSelectionCV
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier

X_train, X_test, y_train, y_test =
 train_test_split(X,y,test_size=0.5,random_state=42)

rf = RandomForestClassifier(n_estimators=100)
selection = GeneticSelectionCV(rf,
                              cv=5, scoring="accuracy",
                              max_features=6, n_population=10,
                              crossover_proba=0.5, mutation_proba=0.2,
                              n_generations=50, crossover_independent_proba=0.5,
                              mutation_independent_proba=0.05, n_gen_no_change=10,
                              n_jobs=-1)

selection = selection.fit(X_train, y_train)
selected_columns = X.columns[selection.support_]
```

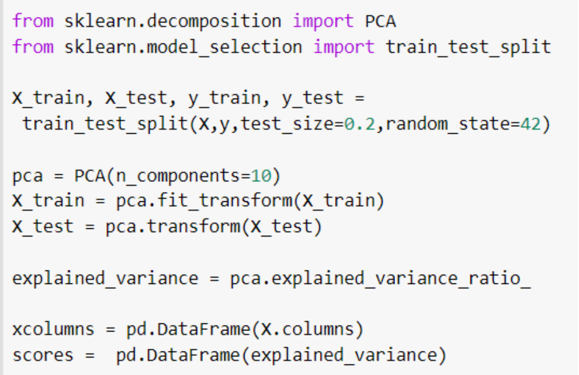
```
Index(['loc', 'iv(G)', 'I', 'lOComment', 'locCodeAndComment', 'uniq_Op'], dtype='object')
```

Fig. 7. **Implementation and results of the Genetic Algorithms method**
Source: **compiled by the authors**

Principal Component Analysis

Principal Component Analysis is a statistical method for converting large-dimensional data to low-dimensional data by selecting most important functions that cover maximum information about the data set. The characteristics are selected based on the variance they cause in the original data. The feature that causes the greatest variance is the first principal component. The function responsible for the second largest variance is considered to be the second principal component, and so on. It is important to note that the principal components do not have any correlation with each other [21].

Applying this feature selection method for our sample, we have determined that the following metrics are important for further study (Fig. 8): *loc*, *v(g)*, *ev(g)*, *iv(G)*, *N*, *V*, *L*, *D*, *I*, *E*.



```
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test =
 train_test_split(X,y,test_size=0.2,random_state=42)

pca = PCA(n_components=10)
X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)

explained_variance = pca.explained_variance_ratio_

xcolumns = pd.DataFrame(X.columns)
scores =  pd.DataFrame(explained_variance)
```

Fig. 8. **Implementation and results of the Principal Component Analysis method**
Source: **compiled by the authors**

Xverse python for feature selection

Xverse uses different methods for the feature selection. When the algorithm selects a function, it votes for that function. Finally, the Xverse calculates the total number of votes for each feature and then selects the best ones based on the votes. Thus, we select the best variables with minimal effort in the function selection process.

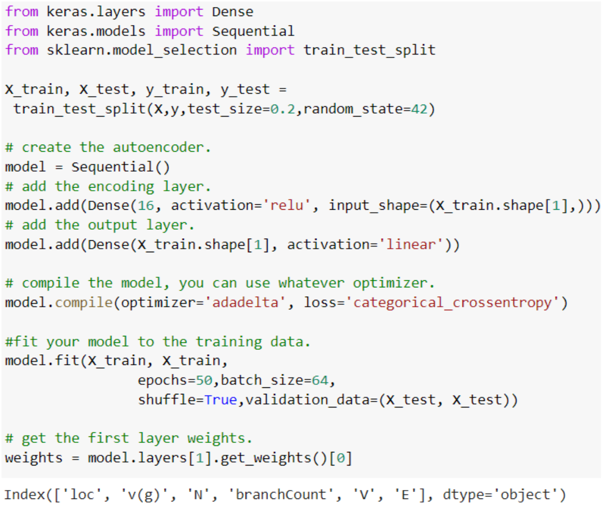
Xverse uses the following methods to select important features: Information Value using Weight of evidence; Variable Importance using Random Forest; Recursive Feature Elimination; Variable Importance using Extra trees classifier; Chi-Square best variables; L1 based feature selection.

According to the results of the performance of this method, we choose the metrics for which the majority of methods voted (4/6): *loc*, *locCodeAndComment*, *uniq_Opnd*, *branchCount*, *N*, *I*, *L*.

Autoencoders method (Deep learning)

Deep learning involves the use of neural networks to create highly effective models of machine learning. What is particularly interesting about neural networks is their ability to study the nonlinear relationships between features. Most of the traditional methods we have studied do not perform this: in general, the methods we have considered can only investigate linear relationships between objects. This method uses a special neural network architecture called autoencoders. AutoEncoder is used for feature selection in order to reveal existing nonlinear relationships between features.

By means of this method, the following important features have been identified (Fig. 9) :*loc*, *v(g)*, *N*, *branchCount*, *V*, *E*.



```
from keras.layers import Dense
from keras.models import Sequential
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test =
 train_test_split(X,y,test_size=0.2,random_state=42)

# create the autoencoder.
model = Sequential()
# add the encoding layer.
model.add(Dense(16, activation='relu', input_shape=(X_train.shape[1],)))
# add the output layer.
model.add(Dense(X_train.shape[1], activation='linear'))

# compile the model, you can use whatever optimizer.
model.compile(optimizer='adadelta', loss='categorical_crossentropy')

#fit your model to the training data.
model.fit(X_train, X_train,
                epochs=50,batch_size=64,
                shuffle=True,validation_data=(X_test, X_test))

# get the first layer weights.
weights = model.layers[1].get_weights()[0]
```

```
Index(['loc', 'v(g)', 'N', 'branchCount', 'V', 'E'], dtype='object')
```

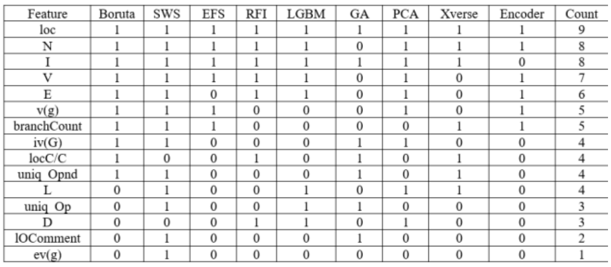
Fig. 9. **Implementation and results of the Autoencoders method**
Source: **compiled by the authors**

RESEARCH RESULTS

Nine feature selection methods have been used to select features, which differ in their types and approaches to feature selection:

- Boruta using Random Forest Classifier (RF)
- Stepwise selection (SWS)
- Exhaustive Feature Selection (EFS)
- Random Forest Importance (RFI)
- LightGBM Importance (LightGBM)
- Genetic Algorithms (GA)
- Principal Component Analysis (PCA)
- Xverse python for feature selection (Xverse)

The results of these methods are shown in Fig. 10, which describes the number of votes for a particular feature selected by the methods described above (1 – the feature has been selected by a feature selection method, 0 – not selected).



Feature	Boruta	SWS	EFS	RFI	LGBM	GA	PCA	Xverse	Encoder	Count
loc	1	1	1	1	1	1	1	1	1	9
N	1	1	1	1	1	0	1	1	1	8
I	1	1	1	1	1	1	1	1	0	8
V	1	1	1	1	1	0	1	0	1	7
E	1	1	0	1	1	0	1	0	1	6
v(g)	1	1	1	0	0	0	1	0	1	5
branchCount	1	1	1	0	0	0	0	1	1	5
iv(G)	1	1	0	0	0	1	1	0	0	4
locC/C	1	0	0	1	0	1	0	1	0	4
uniq Opnd	1	1	0	0	0	1	0	1	0	4
L	0	1	0	0	1	0	1	1	0	4
uniq Op	0	1	0	0	1	1	0	0	0	3
D	0	0	0	1	1	0	1	0	0	3
lOComment	0	1	0	0	0	1	0	0	0	2
ev(g)	0	1	0	0	0	0	0	0	0	1

Fig. 10. **Results of the feature selection methods**

***Source:* compiled by the authors**

The features (hereinafter this set of features will be called *Important features*) which received more than half of the votes (5/9) have been selected as important ones. They are *loc, N, I, V, E, v(g), branchCount.*

The next step is to compare different models of machine learning, such as fandom forest; support vector machine; k-nearest neighbors; decision tree classifier; AdaBoost classifier; Gradient Boosting for classification. We are going to compare the results of the classifiers with different features selected in the previous step in order to select the best combination of program code metrics based on the accuracy of each classifier that can be used to predict the defect of system modules.

The performance of these classifiers will be assessed by the accuracy metric. Accuracy is a metric of how often a learned model is correct, and classification results are often presented as an error matrix. The matrix consists of 4 different combinations of predicted and actual values. The predicted values are described as positive and negative, and actual – as true and false (TP – true positive; TN – true negative; FP – false positive; FN – false negative).

The accuracy metric shows the following ratio (the ratio of correct predictions to their total number):

$$Accuracy=(TP+TN)/(TP+TN+FP+FN).$$

Table 2 shows the results of the performance of classifiers with the accuracy of evaluation for the features selected by the feature selection methods. To train the models, 80 % of the data from the total data sample have been selected and balanced. 20 % of unbalanced data from the total data sample have been used for testing, so that the test sample reflected the real distribution of data in the projects.

Table 1. **Results of performance of classifiers with the accuracy of evaluation for the features**

Feature selection method	***Accuracy***					
	RF	*SVM*	*KNN*	*DT*	*AB*	*GB*
Boruta	0.811	0.827	0.782	0.778	0.793	0.807
SWS	0.782	0.797	0.765	0.754	0.786	0.776
EFS	0.840	0.852	0.804	0.795	0.821	0.841
RFI	0.803	0.820	0.817	0.752	0.816	0.800
LGBM	0.812	0.807	0.794	0.771	0.803	0.820
GA	0.795	0.801	0.780	0.7695	0.803	0.799
PCA	0.821	0.830	0.821	0.799	0.842	0.823
Xverse	0.805	0.810	0.789	0.778	0.808	0.799
Encoder	0.844	0.847	0.817	0.766	0.823	0.822
Important features	0.856	0.838	0.823	0.798	0.815	0.860
Whole features	0.702	0.725	0.618	0.617	0.715	0.691

***Source:* compiled by the authors**

The analysis of the table shows that the important features selected on the basis of voting from all methods show the highest predicting accuracy for 4 of the 6 used classification algorithms; in the case of SVM, although these measured features are not the most accurate, they are still among the three most accurate feature selection methods, and only for AdaBoost classifier classification method, the use of this set of features show slightly worse prediction result after PCA, Autoencoder, and EFS methods. In addition, the top three prediction accuracy methods include feature selection methods such as EFS – the accuracy of five of the six classification algorithms using this set of features is in the top three, and in the case of the SVM method, this accuracy is the highest of all; Autoencoder – 5 out of 6 classification algorithms are in the top three in terms of prediction accuracy, and PCA – 4 out of 6 algorithms are in the top three, but for two of them (decision tree and AdaBoost classifier) this accuracy is the highest. Thus, we can conclude that for this study and the classification algorithms used, the best accuracy of the prediction for most classifiers has been obtained using a set of features obtained by voting from all the studied algorithms. In addition, it has been shown that it is also possible to use certain methods, such as Autoencoder, EFS and PCA, with almost no loss of classification and prediction accuracy. In addition, a significant increase in the accuracy of software defect prediction by reducing the sample of features has been shown. The increase in prediction accuracy in this case (all features and selected features) ranged from 10 % to 21 %. In this case, the use of any method of feature selection increases the accuracy of classification by at least 10 % compared to the original dataset, which confirms the importance of this procedure for predicting software defects based on metric datasets that contain a significant number of software code metrics measured by different approaches which, however, have a strong correlation.

As we can see from the above, the features we have chosen increase the accuracy of software defect prediction, and the corresponding code metrics are related to its reliability and make it possible to build a static model of software reliability based on them. In this work, a set of code metrics obtained as the most important features in the previous stage and a statistical regression method were used to build a software reliability model. Because the target (dependent) variable is binary – the defects metric, which is true or false – logistic regression was chosen as the regression method. This type of regression is well suited to binary classification problems, which in our study means the classification of software modules into those that contain defects and no defects.

The resulting logistic regression equation is as follows:

$$
\begin{aligned}
defects = {} & \beta_1 \cdot loc + \beta_2 \cdot v(g) + \beta_3 \cdot N + \beta_4 \cdot I \\
& + \beta_5 \cdot branchCount + \beta_6 \cdot V \\
& + \beta_7 \cdot E + \beta_0.
\end{aligned}
$$

The values of the regression coefficients are shown in Table 2. The accuracy of the prediction of this regression equation for the test dataset was 0.8288, which is fully consistent with the data in Table 1.

As can be seen from Table 2, the greatest contribution to the probability that a software module contains one or more defects is made by McCabe's cyclomatic complexity, the number of branches in the program, and the number of Halsted operators and operands. This result is in good agreement with the conclusions made by one of the authors in previous works [23] that the reliability of software depends on its complexity, but this dependence is complex and includes several factors (complexity metrics).

Table 2. **The values of the coefficients of the software reliability model**

Coefficient	Value
β_0	-0.99489
β_1	8.8409e-03
β_2	-1.7126e-01
β_3	-1.5715e-02
β_4	-7.1093e-03
β_5	8.8159e-02
β_6	2.4233e-03
β_7	7.9552e-07

***Source:* compiled by the authors**

CONCLUSIONS

The research work is devoted to the improvement of static models of software reliability by means of machine learning methods for the selection of software code metrics that have the strongest impact on its reliability.

The dataset from the PROMISE Software Engineering repository, which has been merged from the datasets КС1, КС2, PC1, CM1, JM1, has been used for research. It contained data on testing software modules and 21 code metrics. Boruta, Step-wise selection, Exhaustive Feature Selection, Random Forest Importance, LightGBM Importance, Genetic Algorithms, Principal Component Analysis, Xverse python methods have been used to select the most important features that affect the quality of program code.

Based on the voting on the results of performance of the feature selection methods using logistic regression, a static (deterministic) model of software reliability has been built, which establishes the relationship between the probability of a defect in the software module and the metrics of its code. It has been shown that this model includes such code metrics as loc, n, i, v, e, v(g), branchCount, which affect the most on the reliability of the software module. The increase in the accuracy of predicting defective software modules in the case of using the constructed model (compared to the initial data set) ranged from 10 % to 21 %.

A comparison of the effectiveness of performance of different feature selection methods has been put into practice; in particular, a study of the effect of the feature selection method on the accuracy of classification has been completed.

Classification has been performed using the following methods: fandom forest; support vector machine; k-nearest neighbors; decision tree classifier; AdaBoost classifier; Gradient Boosting for classification. It has been shown that the use of a feature selection method increases the accuracy of classification by at least 10 % compared to the original dataset, which confirms the importance of this procedure for predicting software defects based on metric datasets that contain a significant number of highly correlated software code metrics.

It has been found that the best prediction accuracy for most classifiers was obtained using a set of features obtained from the proposed static model of software reliability. In addition, it has been shown that it is also possible to use separate methods, such as Autoencoder, EFS and PCA with an insignificant loss of classification and prediction accuracy.

The issue of cross-project defect prediction based on the results obtained in this article is a complex scientific task and requires further research. However, because different projects have been used for research, and software code metrics are widely accepted and widely used in software engineering, the authors believe that the code metrics they have most affected by software reliability are universal and can be extended to other software projects.

REFERENCES

1. Chen, X., Gu, Q., Liu, W. S., Liu, S. L. & Ni. C. “Survey of static software defect prediction”. *Journal of Software (in Chinese).* 2016; Vol. 27 No. 1: 1–25.

2. Sridhar, P. & Mehta, S. “Stacking based ensemble learning for improved software defect prediction”. *Proceeding of Fifth International Conference on Microelectronics, Computing and Communication Systems*. 2021. p. 167–178.

3. Wei, H., Shan, C., Hu, C., Zhang, Y. & Yu, X. “Software defect prediction via deep belief network”. *Chinese Journal of Electronics*. 2019; Vol. 28 No. 5: 925–932. DOI: https://doi.org/10.1049/cje.2019.06.012.

4. Hu, Changhong, Heqi Wang, Xiangzhi Li, Hailong Liu, Ming Sun & Wu Sun. “Decision-level defect prediction based on double focuses”. *Chinese Journal of Electronics.* 2017; Vol. 26 No. 2: 256–62. DOI: https://doi.org/10.1049/cje.2017.01.005.

5. Asano, T. & Tsunoda, M. “Using bandit algorithms for project selection in cross-project defect prediction”. *37th International Conference on Software Maintenance and Evolution (ICSME).* 2021. p. 626–643.

6. Tang, S., Huang, S., Zheng, C., Liu, E., Zong, C. & Ding, Y. “A novel cross-project software defect prediction algorithm based on transfer learning”. *Tsinghua Science and Technology.* 2020; Vol. 27 No.1: 41–57. DOI: https://doi.org/10.26599/tst.2020.9010040.

7. Kumar, L., Kumar, M. & Murthy, L. B. “An empirical study on application of word embedding techniques for prediction of software defect severity level”. *Proceedings of 16th Conference on Computer Science and Intelligence Systems (FedCSIS).* 2021. p. 477–484. DOI: https://doi.org/ 10.15439/2021F100/

8. Balogun, A. O., Basri, S., Mahamad, S., Abdulkadir, S. J. & Bajeh, A. A. “Impact of feature selection methods on the predictive performance of software defect prediction models: An extensive empirical study”. *Symmetry.* 2020; Vol. 12 No. 7: 1147–1156. DOI: https://doi.org/10.3390/sym12071147.

9. Lei T., Xue J., Wang Y., Niu Z., Shi Z. & Zhang Yu. “WCM-WTrA: A cross-project defect prediction method based on feature selection and distance-weight transfer learning”. *Chinese Journal of Electronics*. 2021. p. 133–140. DOI: https://doi.org/ 10.1049/cje.2021.00.119.

10. Jiarpakdee, J., Tantithamthavorn, C. & Treude, C. “The impact of automated feature selection techniques on the interpretation of defect models”. *Empirical Software Engineering.* 2020; Vol. 25 No. 5: 3590–3638. DOI: https://doi.org/10.1007/s10664-020-09848-1.

11. Belouch, M., Elhadaj, S. & Idhammad, M. “A hybrid filter-wrapper feature selection method for DDoS detection in cloud computing”. *Intelligent Data Analysis.* 2018: Vol. 22 No. 6: 1209–1226. DOI: https://doi.org/10.3233/ida-173624.

12. Anbu, M. & Anandha Mala, G. S. “Feature selection using firefly algorithm in software defect prediction”. *Cluster Computing.* 2017; Vol. 22 No. S5: 10925–34. DOI: https://doi.org/10.1007/s10586-017-1235-3.

13. Hans Rahul & Harjot Kaur. “Opposition-based enhanced grey wolf optimization algorithm for feature selection in breast density classification”. *International Journal of Machine Learning and Computing.* 2020: Vol. 10 No. 3: 458–64. DOI: https://doi.org/10.18178/ijmlc.2020.10.3.957.

14. Venkatesh, B. & Anuradha, J. “A review of feature selection and its methods”. *Cybernetics and Information Technologies.* 2019; Vol. 19 No. 1: 3–26. DOI: https://doi.org/10.2478/cait-2019-0001.

15. “Analytics Vishay. Feature selection using wrapper method – Python implementation”. 2020. – Available at: https://www.analyticsvidhya.com/blog/2020/10/a-comprehensive-guide-to-feature-selection-using-wrapper-methods-in-python. – [Accessed: 19 Nov. 2020].

16. Mwadulo, M. W. “A review on feature selection methods for classification tasks”. *International Journal of Computer Applications Technology and Research.* 2016; Vol. 5 No. 6: 395–402. DOI: https://doi.org/10.7753/ijcatr0506.1013.

17. Kursa, M. B., Jankowski, A. & Rudnicki, W. R. “Boruta – a system for feature selection”. *Fundamenta Informaticae*. 2010; Vol. 101 No. 4: 271–85. DOI: https://doi.org/10.3233/fi-2010-288.

18. Kohavi, R. & John, G. H. “Wrappers for feature subset selection”. *Artificial Intelligence.* 1997; Vol. 97 No.1-2: 273–324. DOI: https://doi.org/10.1016/s0004-3702(97)00043-x.

19. Chen, R. C., Dewi C., Huang S.W., & Caraka, R. E. “Selecting critical features for data classification based on machine learning methods”. *Journal of Big Data*. 2020; Vol. 7 No. 1. DOI: https://doi.org/10.1186/s40537-020-00327-4.

20. Chiesa, M., Maioli, G., Colombo, G. I. & Piacentini L. “GARS: Genetic algorithm for the identification of a robust subset of features in high-dimensional datasets”. *BMC Bioinformatics.* 2020; Vol. 54 No.1. DOI: https://doi.org/10.1186/s12859-020-3400-6.

21. Guo, Q., Wu, W., Massart, D.L., Boucon, C. & de Jong S. “Feature selection in principal component analysis of analytical data”. *Chemometrics and Intelligent Laboratory Systems.* 2002: Vol. 61 No. 1-2: 123–32. DOI: https://doi.org/10.1016/s0169-7439(01)00203-9.

22. Goodfellow, I., Bengio, Y. & Courville A. “Deep Learning”. Cambridge, Massachusetts: *The MIT Press*. 2016.

23. Yakovyna, V. “Method of software reliability analysis considering its complexity”. *Radioelectronic and Computer Systems*. 2015: No. 2 (72): 127–133.

Conflicts of Interest: the authors declare no conflict of interest

Received 14.01.2021
Received after revision 12.03.2021
Accepted 17.03.2021

DOI: https://doi.org/10.15276/aait.04.2021.5
УДК 004.922

Побудова моделі дефектності програм: вибір метрик

Віталій Степанович Яковина[1) 2)]
ORCID: https://orcid.org/ 0000-0003-0133-8591; yakovyna@matman.uwm.edu.pl. Scopus Author ID: 6602569305
Іван Ігорович Симець[2)]
ORCID: https://orcid.org/ 0000-0003-1873-3168; ivan.i.symets@lpnu.ua. Scopus Author ID: 57202453582
[1)] Вармінсько-Мазурський університет в Ольштині, вул. Очаповського, 2. Ольштин, 10-719, Польща
[2)] Національний університет «Львівська політехніка, вул. Степана Бандери, 12. Львів, 79000, Україна

АНОТАЦІЯ

Дана стаття націлена на удосконалення статичних моделей надійності ПЗ за рахунок використання методів машинного навчання для вибору метрик коду ПЗ, що найсильніше впливають на його надійність. У дослідженні було використано злитий датасет з репозиторію PROMISE Software Engineering, який містив дані про тестування програмних модулів п’яти програм (КС1, КС2, PC1, CM1, JM1) та двадцять одну метрику коду. Для підготовленої вибірки було здійснено вибір найважливіших ознак, які впливають на якість програмного коду за допомогою наступних методів вибору ознак: Boruta, Step-wise selection, Exhaustive Feature Selection, Random Forest Importance, LightGBM Importance, Genetic Algorithms, Principal Component Analysis, Xverse python. На основі голосування за результатами роботи методів вибору ознак побудовано статичну (детерміністичну) модель надійності програмного забезпечення, яка встановлює взаємозв’язок між ймовірністю появи дефекту в програмному модулі та метриками його коду. Показано, що в цю модель входять такі метрики коду як кількість гілок програми, кількість рядків коду та цикломатична складність за МакКейбом, загальна кількість операторів та операндів, інтелект, обсяг та кількість зусиль за Холстедом. Здійснено порівняння ефективності роботи різних методів вибору ознак, зокрема проведено дослідження впливу методу вибору ознак на точність класифікації із використанням наступних класифікаторів: Random Forest, Support Vector Machine, k-Nearest Neighbor, Decision Tree classifier, AdaBoost classifier, Gradient Boosting for classification. Показано, що використання будь-якого методу вибору ознак підвищує точність класифікації принаймні на десять процентів порівняно з початковим датасетом, що підтверджує важливість цієї процедури для прогнозування дефектів програмного забезпечення на основі метричних датасетів, які містять значну кількість сильно корелюючих метрик коду ПЗ. Встановлено, що найкращу для більшості класифікаторів точність прогнозу вдалось отримати з використанням набору ознак, отриманого із запропонованої статичної моделі надійності ПЗ. Крім того, показано, що можливим також є використання окремих методів, таких як Autoencoder, Exhaustive Feature Selection та Principal Component Analysis з незначною втратою точності класифікації та прогнозування.

Ключові слова: надійність програмного забезпечення; машинне навчання; дефект; вибір ознак; прогнозування дефектів програмного забезпечення

ABOUT THE AUTHORS

Vitaliy S. Yakovyna – D. Sc. (Eng), Professor of Artificial Intelligence Department of Lviv Polytechnic National University, 12, S. Bandera Str. Lviv, 79013, Ukraine
University of Warmia and Mazury in Olsztyn, 2, Oczapowskiego, St. Olsztyn, 10-719, Poland
ORCID: https://orcid.org/ 0000-0003-0133-8591; yakovyna@matman.uwm.edu.pl. Scopus Author ID: 6602569305
Research field: Software reliability; software safety; machine learning

Віталій Степанович Яковина – доктор технічних наук, професор, професор каф. Систем штучного інтелекту Національного університету «Львівська політехніка, вул. Степана Бандери, 12. Львів, 79000, Україна.
Д-р техніч. наук, професор Вармінсько-Мазурського університету в Ольштині, вул. Очаповська 2. Ольштин, 10-719, Польща

Ivan I. Symets – PhD Stutent, Assistant of Software Department of Lviv Polytechnic National University, 12, S. Bandera Str. Lviv, 79013, Ukraine
ORCID: https://orcid.org/ 0000-0003-1873-3168; ivan.i.symets@lpnu.ua. Scopus Author ID: 57202453582
Research field: Software reliability; defect prediction in software engineering; artificial intelligence

Іван Ігорович Симець – аспірант, асистент каф. Програмного забезпечення Національного університету «Львівська політехніка, вул. Степана Бандери, 12. Львів, 79000, Україна