A Genetic Algorithm to Configure Support Vector Machines for Predicting Fault-Prone Components

Sergio Di Martino[1], Filomena Ferrucci[2], Carmine Gravino[2], and Federica Sarro[2]

[1] University of Napoli "Federico II" Via Cinthia, 80126 Napoli, Italy
sergio.dimartino@unina.it
[2] University of Salerno, Via Ponte Don Melillo, 84084 Fisciano (SA), Italy
{fferrucci,gravino,fsarro}@unisa.it

Abstract. In some studies, Support Vector Machines (SVMs) have been turned out to be promising for predicting fault-prone software components. Nevertheless, the performance of the method depends on the setting of some parameters. To address this issue, we propose the use of a Genetic Algorithm (GA) to search for a suitable configuration of SVMs parameters that allows us to obtain optimal prediction performance. The approach has been assessed carrying out an empirical analysis based on jEdit data from the PROMISE repository. We analyzed both the inter- and the intra-release performance of the proposed method. As benchmarks we exploited SVMs with Grid-search and several other machine learning techniques. The results show that the proposed approach let us to obtain an improvement of the performance with an increasing of the Recall measure without worsening the Precision one. This behavior was especially remarkable for the inter-release use with respect to the other prediction techniques.

Keywords: Fault prediction, Support Vector Machines, Genetic Algorithm.

1 Introduction

Software testing is one of the most expensive phases of the software development life cycle and at the same time it is very critical for the quality of a product. Thus, it is valuable for the competitiveness of a software company to have tools able to better support this phase. A huge amount of research in Software Engineering has been devoted to improve the efficiency of testing. Among these, considerable efforts have been aimed towards the definition of techniques able to predict the components of a software system that more likely will contain faults (e.g., [8, 13, 18, 19, 23, 26]). The rationale is that, once identified these potentially defective components, project managers can better decide how allocate resources to test the system concentrating their testing efforts to fault-prone components. As a result, the reliability, the quality, and the cost/effectiveness of the software product can be improved.

Usually the approaches for predicting fault-prone components exploit statistical models able to relate some software structural metrics with the probability of the presence of faults. Typical employed metrics are the Lines of Code (LOCs), the Halstead measures [15], or the Chidamber and Kemerer (CK) metrics for Object-Oriented systems [7]. Among the machine learning techniques, Neural Networks, Bayesian Networks, Logistic Regression, and Decision Tree algorithm C4.5 are widely used [1, 8, 11, 17, 25, 29].

Many efforts have been devoted to understand what are the most explicative metrics for the phenomenon at the hand, and to tailor machine-learning techniques in order to provide more accurate identification of the defective components (e.g., [1]). In this context, some studies have reported a good performance of Support Vector Machines (SVMs) to predict fault-prone components (see e.g., [8, 12]).

SVMs are supervised learning methods that can be applied to classification tasks; i.e., given a set of data, each marked as belonging to one of two groups, a model is constructed to predict whether a new item falls into one group or the other. SVMs have turned to be a powerful tool in several contexts. Nevertheless, the application of the technique is not straightforward since some parameters must be carefully set to get more accurate classifications. Moreover, the suitable values of such parameters depend on the characteristics of the dataset, thus no rule of thumb can be defined, but a search technique has to be employed. The most of the studies presented in the literature adopts a "Grid-search" [6] approach, whose grain is very coarse.

To overcome these limitations, in this paper we propose a solution for the identification of error-prone components, based on the combination of a Genetic Algorithm (GA) and SVMs. In particular, a GA is exploited to search for a suitable SVM parameter setting exploiting a fitness function. Each criterion that is used to determine the performance of fault prediction methods can be used as fitness function.

The proposed approach has been assessed by an empirical analysis meant to verify the effectiveness of the approach to configure SVMs. As datasets, we have employed data on jEdit, a well-known text editor written in Java, whose information in terms of metrics and faults are available in the PROMISE database [24] for different releases. This has allowed us to analyze both the inter- and the intra-release performance of the method. In particular, we used a 10 fold cross-validation for the intra-release analyses, and a hold-out validation for an inter-release assessment, thus replicating with the latter validation a situation that typically arises in real software testing context, where data from the former releases are exploited to train the model that is used to predict faults for a new release. As benchmarks we employed SVMs in combination with the use of Grid-search for parameter selection [6] and six other widely used machine learning techniques, namely Logistic Regression, Decision Tree Algorithm C4.5, Naïve Bayes, Multi-Layer Perceptrons, K-Nearest Neighbor, and Random Forest. As measures to compare performance, we have employed Accuracy, Precision, Recall, and F-Measure. All these measures were also experimented as fitness function of GA.

The remainder of the paper is structured as follows: in Section 2 we describe the proposed approach reporting the main concepts of SVMs and GA and illustrating the key aspects of the defined GA. Section 3 is devoted to present the planning of the empirical analysis, in terms of the investigated research goals, the employed datasets, and the adopted validation method and evaluation criteria. In Section 4 we report and discuss the results, while in Section 5 we analyze the threats to the validity of the empirical study. A review of the related work is presented in Section 6, while some final remarks conclude the paper.

2 Support Vector Machines and Genetic Algorithms

In the following, we provide a brief description of Support Vector Machines and Genetic Algorithms, and how we tailored a Genetic Algorithm to configure Support Vector Machines.

2.1 Support Vector Machines

Support Vector Machines (SVMs) are a classification technique developed by Vapnik at the end of '60s [28, 29]. Since then the technique has been deeply improved, being applied in many different contexts. SVMs are known as maximum-margin classifiers, since they find the optimal hyperplane between two classes, defined by a number of support vectors [13]. The well-known generalization feature of the technique is mainly due to the introduction of a penalty factor, named *C*, that allows us to prevent the effects of outliers by permitting a certain amount of misclassification errors.

Although the original technique was able to provide only linear classification, thanks to the use of the kernel trick, SVMs can handle also non-linear problems. Indeed, in this case a kernel function is used to implicitly map the data points into a higher-dimensional feature space. Consequently, every dot product is replaced by the nonlinear kernel function, allowing the technique to find the maximum-margin hyperplane in a transformed, higher dimensional space. The rationale is that data is more likely to be linearly separable in the higher feature space [16]. A lot of kernel functions have been proposed in the literature. Among them, the ones based on Radial Basis Functions (RBF) are widely employed. In this study we decided to employ the RBF kernel since it was previously used in defect prediction context [13, 26, 27] and usually yields better performance than other kernels [5, 16].

When SVM is used with the RBF kernel, two parameters, C and γ, have to be set by the user. The selection of appropriate values for these parameters is crucial to obtain good classification performance. As described before, C is the penalty factor for misclassified points. If it is too large, a higher penalty for non-separable points is added, leading to store too many support vectors and thus over fit. On the other hand, if C is too small, an underfitting can occur. The γ parameter specifies the radius of the RBF, also having a strong impact on the accuracy.

A suitable combination of C and γ is often selected by a Grid-search with exponentially growing sequences of values, as done in LibSVM [6], one of the most employed, freely available library for SVMs. However, this approach has a twofold problem: 1) it has a very coarse grain, and thus it is likely to miss optimal values; 2) always the same couples of values for C and γ are explored, without taking into account the problem at the hand. In other contexts, the use of some search-based approaches [16] has been investigated to address the problem. For instance, in [5] Tabu Search (TS) has been employed to configure Support Vector Regression (SVR), i.e., the regression version of SVM, for software development effort estimation. TS is a meta-heuristic relying on adaptive memory and responsive exploration of the search space that has been used to address several optimization problems [10]. The results of the case study presented in [5] revealed that TS was able to suitably set the parameters of SVR. In this paper, to configure SVMs for fault prediction we investigate the use of another search-based approach, namely a Genetic Algorithm. In the next subsection we detail the approach.

2.2 A Genetic Algorithm to Configure SVMs

Genetic Algorithms [11] belong to the family of evolutionary algorithms that, inspired by the theory of natural evolution, simulate the evolution of species emphasising the law of survival of the strongest to solve, or approximately solve, optimisation problems. The idea of exploiting this method to configure SVMs for fault prediction is based on the observation that the setting of SVMs can be formulated as an optimisation problem. As a matter of fact, among the possible configurations (solutions), we have to identify the one which leads to the optimal SVMs performance.

A typical Genetic Algorithm (GA) creates consecutive populations of individuals (i.e., chromosomes), considered as feasible solutions for a given problem, to search for a solution which gives the best approximation of the optimum for the problem under investigation. To this end, a fitness function is used to evaluate the goodness (i.e., fitness) of the chromosomes and genetic operators based on selection and reproduction are employed to create new populations (i.e., generations). Despite of a number of variations, the elementary process of a GA is the follows: (i) first a random initial population, i.e., a set of chromosomes, is generated; (ii) then, new individuals (i.e., offspring) are created by applying genetic operators (i.e., crossover and mutation) and a selection based on individual's fitness value is applied to determine who will survive among the offspring and their parents; (iii) the second step is repeated until either the fitness of the best solution has converged to the optimal value or if the optimal is not knew until a certain number of generations have been made. The individual that gives the best solution in the final population is taken in order to define the best approximation to the optimum for the problem under investigation. The analysis of this process suggests that the following design choices have to be made for tailoring GA to a given optimisation problem [11]:

1. defining the chromosome for representing a solution (i.e., *solution encoding*) and the number of initial solutions (i.e., *population size*);
2. choosing the criterion (i.e., *fitness function*) to measure the goodness of a chromosome;
3. defining the combination of *genetic operators* to explore the search space;
4. defining the *stopping criteria*.

In the following we provide the details regarding the choices we made for points 1-4 to design a GA for configuring SVMs. Concerning the *solution encoding*, since a solution has to represent an SVM configuration, the corresponding chromosome is composed by two genes, i.e., one for each SVM parameter, and the values for the genes are selected in the ranges [0.01, 32000] and [1.0E-6, 8] for the genes representing C and γ, respectively. These ranges are the same adopted in the Grid-search included in LibSVM [6]. The initial population is composed by 100 chromosomes that are created assigning random values to each gene. To assign the fitness value, the goodness of chromosomes is evaluated by running SVMs with the configuration represented by each chromosome and employing as *fitness function* some widely used performance measures (i.e., Accuracy, F-measure, Precision, and Recall) which are described in section 3.3. As for the *genetic operators*, we employed a single point crossover which combines two individuals (i.e., parents) to form a new individual by randomly selecting a point of cut and swapping all genes beyond that point in either parent. Concerning the mutation operator, it selects a random gene of the chromosome and randomly changes the associated value. Crossover and mutation rates are fixed to 0.5 and 0.1, respectively. To determine the individuals that are included in the next generation (i.e., survivals) we employed the tournament selector, where only the best *n* solutions are copied straight into the next generation.

The evolutionary process is terminated according two *stopping criteria*, i.e. after 300 generations or if the fitness value of the best solution does not change after 30 generations.

3 Case Study Planning

In this section, we present the design of the empirical study we performed to get an insight on the use of the proposed GA to configure SVMs for fault prediction. In particular, to verify the effectiveness of the combination of GA and SVMs described in the previous section, we compared the obtained predictions with those achieved using SVMs configured with the Grid-search provided by LIBSVM [6]. Since the choice of the fitness function can play a crucial role in the application of a GA we have also experimented the impact of using different fitness functions. Furthermore, to understand the actual effectiveness of the proposed approach with respect to other methods we have also investigated some techniques available in the Weka tool [14], i.e., Logistic Regression (LR), Decision Tree Algorithm C4.5 (C4.5), Naïve Bayes (NB), Multi-Layer Perceptrons (MLP), K-Nearest Neighbor (KNN), and Random Forest (RF).

To perform this analysis we employed two datasets, namely *jEdit: 4-0-final_4-2-final* (in the following *jE1*) and *jEdit: 4-2-final_4-3-pre12* (in the following *jE2*)[1], obtained from different releases of the same software (*jEdit*) and included in the PROMISE repository [24]. In particular, we carried out two kinds of analysis: intra-release and inter-release similarly to [30].

Thus, the research questions of our study can be outlined as follows:

- RQ1: Can the choice of the fitness function affect the prediction performance of the combination of GA and SVMs?
- RQ2: Is the proposed GA able to effectively configure SVMs parameters for fault prediction?
- RQ3: Is the fault prediction performance of the combination of GA and SVMs superior to the ones obtained by other techniques?
- RQ4: Are there differences in the performance of the considered techniques for intra- and inter-release fault prediction?

[1] jE1 and jE2 are available at http://promisedata.org/?p=74 and http://promisedata.org/?p=73, respectively.

3.1 Dataset

To carry out the empirical evaluation of the proposed technique, we selected two datasets from the PROMISE repository, which contains data made publicly available in order to encourage repeatable, verifiable, refutable, and/or improvable predictive models of software engineering [24].

The datasets we employed are related to versions 4.0, 4.2 and 4.3 of the jEdit system, a well-known text editor written in Java. They contain data for the Chidamber and Kemerer (CK) metrics [7], a suite of six complexity metrics widely employed to measure product attributes of Object Oriented software systems and described in Table 1 together with Number of Public Methods and Number of Lines of Code.

Metrics data were computed by exploiting Understand IDE, a tool able to measure many characteristics of a software application. As for the fault data, they were obtained from SVN log files (where "true" means fault existence and "false" means fault nonexistence) [24]. In particular, for dataset *jE1* the metric data were computed based on jEdit release 4-0-final, while fault data were extracted between releases 4-0-final and 4-2-final. Similarly, for dataset *jE2* the metrics data were computed based on jEdit release 4-2-final, while fault data were extracted between releases 4-2-final and 4-3-pre12. The descriptive statistics of the metrics and fault data for the employed datasets are reported in Tables 2 and 3.

3.2 Validation Method

To assess the effectiveness of the fault predictions obtained using the techniques described herein, we have employed a k-fold cross validation for the intra-release analyses and a hold-out validation for the inter-release analysis.

Table 1. Metrics of the employed datasets

Name	Definition
Weighted Methods per Class (WMC)	The WMC metric represents the number of methods in the class (assuming unity weights for all methods).
Depth Inheritance Tree (DIT)	The DIT metric provides for each class a measure of the inheritance levels from the object hierarchy top.
Number Of Children (NOC)	The NOC metric measures the number of immediate descendants of the class.
Coupling Between Object Classes (CBO)	The CBO metric represents the number of classes coupled to a given class.
Response For Class (RFC)	The RCF metric measures the number of different methods that can be executed when an object of that class receives a message.
Lack of Cohesion Metric (LCOM)	The LCOM metric counts the sets of methods in a class that are not related through the sharing of some of the class fields.
Number of Public Methods (NPM)	The NPM metric counts all the methods in a class that are declared as public. The metric is known also as Class Interface Size (CIS).
Lines Of Code (LOC)	The LOC metric is the number of instructions in each method of the class.

Table 2. Descriptive statistics for the employed datasets

Dataset	Variable	Min	Max	St.dev	Mean
jE1	WMC	0	407	31.2	11.73
	DIT	0	7	1.98	2.50
	NOC	0	35	3.1	0.72
	CBO	0	105	14.13	12.64
	RFC	0	843	269.59	174.98
	LCOM	0	100	33.52	46.24
	NPM	0	193	17.12	7.78
	LOC	3	6191	529.66	206.21
jE2	WMC	0	351	26.46	11.66
	DIT	0	8	2.01	2.4
	NOC	0	38	3.04	0.71
	CBO	0	125	14.34	13.46
	RFC	0	862	268.58	169.98
	LCOM	0	100	33.66	48.86
	NPM	0	214	16.60	7.68
	LOC	3	5317	478.13	225.28

Table 3. Existence of faults

Dataset	# Elements with No Faults	# Elements with Faults
jE1	140	134
jE2	165	204

Cross validation is widely used in the literature to validate prediction models when dealing with medium/small datasets (see e.g., [4]). In particular, the k-fold cross validation is applied by partitioning each original dataset into k training sets, for model building, and test sets, for model evaluation. This is done in order to avoid optimistic predictions [20]. The errors from applying a given prediction technique are summarized using several performance measures (described in the following). In our study, we applied a 10-fold cross validation on the selected datasets, thus obtaining k=10 randomly test sets, and then for each of them, the remaining observations formed the training set to build the prediction model. As for the hold-out validation we have employed the first dataset, i.e., *jE1*, as training set and the second dataset, i.e., *jE2*, as test set.

3.3 Evaluation Criteria

Concerning the evaluation of the predictions obtained with the analyzed methods, we have used the following widely employed performance measures: Accuracy, Recall, Precision, and F-measure [1, 3]. In order to calculate them, we have exploited the concepts of True Positive (TP), False Negative (FN), False Positive (FP), and True Negative (TN). This classification is represented in the Confusion Matrix reported in Table 4.

Accuracy is defined as the ratio between the number of components correctly predicted (i.e., classified as TP and TN) and the total number of components (i.e., the sum of TP, TN, FP, and FN).

Table 4. The confusion Matrix

		Predicted	
		Defective	Non Defective
Actual	Defective	True Positive	False Negative
	Non Defective	False Positive	True Negative

Precision is defined as the ratio between the number of components classified as TP and the number of components classified as TP or FP.

Recall is defined as the ratio between the number of components classified as TP and the number of components classified as TP or FN.

This means that Precision concerns the correctness of the responses provided by the method, while the completeness of the responses is measured by employing Recall. It is easily understandable that in the context of fault prediction, the consequence of low Recall is far more important than low Precision [22]. Thus, it is fundamental to employ a technique able to maximize Recall [30]. Indeed, in this way, the technique is able to detect all the potential defective components, even at the cost of a lower Precision, in order to avoid that potential defective components are missed. In this scenario, a subsequent manual check is responsible of deleting false positives inserted in this step. On the other hand, having a too low Precision requires an excessive post-processing, making the technique useless. A measure that provides an indication of a balance between correctness and completeness is the harmonic mean of Precision and Recall (or F-measure), defined as:

$$F - measure = 2 * \frac{precision * recall}{precision + recall} \quad (1)$$

A key advantage of GA is that it allows project managers to select the preferred criterion to drive the search for the prediction model and try to optimize a given performance measure (see e.g., [9, 17]).

4 Results and Discussion

The following subsections present and discuss on the results achieved in the empirical study.

4.1 Fitness Function Impact and Effectiveness of GA to Configure SVMs

In this section, we consider the first and the second research questions. The results of the application of the combination of GA and SVMs (GA+SVM in the following) for the inter- and the intra-release analysis and related to each of the investigated fitness functions (i.e., Accuracy, Precision, Recall, and F-measure) are reported in Table 5. These results suggest that the performance of GA+SVM are affected by the specific fitness function employed. In particular, the best value for each criterion has been obtained in almost all the cases using such a criterion as fitness function. Moreover, there are some fitness functions that pay this with a worsening of the other criteria. In particular, the fitness function Recall was not able to provide high scores in terms of Accuracy and Precision. This is an expected result since it is widely recognized that any attempt to improve Recall is often paid back by a decreasing of Precision because it gets increasingly harder to be precise as the sample space increases [3]. Nevertheless, this phenomenon is not observed for all the fitness functions. Indeed, the best overall performances on both datasets *jE1* and *jE2* have been obtained by using F-measure as fitness function. Indeed, it has allowed us to obtain Recall and Precision values very close to the best ones. Furthermore, the fitness function F-measure has also provided the best Accuracy value in the case of the first dataset and a value close to the best one for the second dataset.

The results for the hold-out validation related to the inter-release assessment are also reported in Table 5. As expected these results are worse than those achieved by the intra-release analysis. Nevertheless, similarly to intra-release prediction models, we can observe that the best overall results were obtained using F-measure as fitness function. Thus, we can positively answer the research question RQ1 outlined in Section 3, i.e., the choice of the fitness function affected the accuracy of the prediction models built using GA+SVM.

In order to answer the research question RQ2 we compared the results achieved with GA+SVM with those obtained by applying SVMs configured with Grid-search of LIBSVM [6] (SVM-Grid in the following). The obtained results are shown in Table 6 together with the ones obtained with the other considered techniques and discussed in the next subsection. To facilitate the comparison in Table 6 we have also reported the results obtained with GA+SVM using F-measure as fitness function (since it provided the best performance).

The analysis of these results reveals that, for the first dataset, GA+SVM has allowed us to obtain better predictions than SVM-Grid in terms of Accuracy (c.a., 0.7%), Recall (c.a., 11.2%), and F-measure (c.a., 5%), while it has produced almost the same results in terms of Precision. With regard to the second dataset, GA+SVM

Table 5. Results using the combination of GA and SVMs with different fitness functions (the best results are in bold)

Dataset	Fitness function	Results Accuracy	Precision	Recall	F-measure
jE1	Accuracy	0.719	0.677	0.813	0.739
	Precision	0.723	**0.704**	0.746	0.725
	Recall	0.635	0.582	**0.903**	0.708
	F-measure	**0.741**	0.701	0.821	**0.756**
jE2	Accuracy	0.637	0.610	0.951	0.743
	Precision	**0.648**	**0.635**	0.853	0.728
	Recall	0.572	0.564	**1**	0.721
	F-measure	0.629	0.601	0.980	**0.745**
jE1&jE2 (inter-release)	Accuracy	0.580	0.616	0.637	0.627
	Precision	0.588	**0.626**	0.632	0.629
	Recall	0.591	0.604	0.755	0.671
	F-measure	**0.615**	0.609	**0.848**	**0.709**

Table 6. Results for the employed prediction techniques

Dataset	Technique	Results			
		Accuracy	Precision	Recall	F-measure
	GA+SVM F-measure	0.741	0.701	**0.821**	**0.756**
	SVM-Grid	0.712	0.704	0.709	0.706
jE1	LR	0.715	0.719	0.687	0.715
	C4.5	0.708	0.696	0.716	0.706
	NB	0.693	**0.784**	0.515	0.622
	MLP	0.701	0.676	0.746	0.709
	KNN	0.723	0.730	0.687	0.708
	RF	**0.748**	0.721	0.791	0.754
	GA+SVM F-measure	0.629	0.601	**0.980**	**0.745**
	SVM-Grid	0.648	0.657	0.754	0.702
jE2	LR	0.634	0.658	0.706	0.681
	C4.5	0.612	0.660	0.618	0.638
	NB	0.531	**0.763**	0.221	0.342
	MLP	0.607	0.632	0.691	0.660
	KNN	**0.661**	0.691	0.701	0.696
	RF	0.653	0.673	0.725	0.698
jE1&jE2	GA+SVM F-measure	**0.615**	0.609	**0.848**	**0.709**
(inter-release)	SVM-Grid	0.553	0.602	0.564	0.582
	LR	0.561	**0.615**	0.549	0.580
	C4.5	0.534	0.575	0.598	0.587
	NB	0.501	0.585	0.338	0.429
	MLP	0.542	0.579	0.627	0.602
	KNN	0.537	0.588	0.539	0.563
	RF	0.558	0.596	0.623	0.609

scored better than SVM-Grid for Recall (c.a., 22.6%) and F-measure (c.a., 4.3%), while it has produced worse results than SVM-Grid for Accuracy (c.a., 1.9) and Precision (c.a., 0.1%). As for the inter-release prediction, GA+SVM was able to outperform SVM-Grid for all the considered measures, with an impressing improvement in terms of Recall of more than 28%, having at the same time also a slight improvement in Precision (c.a., 0.7%). Consequently, the F-measure value growth of almost 13%.

The above results suggest that the use of GA to configure SVMs allowed us to obtain better results than SVM-Grid. This improvement is particularly evident and significant for Recall and is paid back by a little decrease in the values of Precision (only for *jE2*). Moreover, if the project manager is particularly interested in maximizing Recall also accepting a decrease of Precision, he/she can even exploit Recall as fitness function.

Thus, we can also positively answer the research question RQ2 outlined in Section 3, i.e., GA is able to effectively set SVMs configuration parameters for fault prediction.

4.2 Comparison of GA+SVM with other Estimation Techniques and between Intra- and Inter-release Prediction Performance

In this section, we report the results related to the research questions RQ3 and RQ4 and obtained by comparing the predictions provided by the combination of GA and

SVMs with the ones achieved by using other classification techniques available in the Weka tool [14], i.e., LR, C4.5, NB, MLP, KNN, and RF. Preliminarily, we compare the results obtained with these other techniques and reported in Table 6. We can observe that the best performance for the intra-release analysis was obtained with RF while NB provided the best Precision value but also the worst Recall and F-measure values. Regarding the results for the inter-release analysis, we can observe that the best Recall and F-measure values were obtained with RF and MLP, while the best Accuracy and Precision values were achieved with LR and RF. The results also suggest that NB provided the worst Precision, Recall, F-measure, and Accuracy values.

Concerning the comparison between the results we achieved with GA+ SVM and those obtained using the above techniques we can observe that GA+SVM, exploiting F-measure as fitness function, allowed us to get the best results in terms of Recall and F-measure for all the considered experimental settings.

Regarding the results obtained for the inter-release analysis (see Table 6), we can observe that among the prediction techniques we employed only the predictions obtained with GA+SVM with F-measure are very similar to the ones achieved in the intra-release assessment. Indeed, the predictions provided by the other techniques are worse than the ones obtained in the intra-release context. Moreover, also in this case GA+SVM with F-measure yielded to the best results in terms of Accuracy, Recall, and F-measure and comparable results in terms of Precision with respect to the other techniques we employed.

Thus, we can positively answer to research question RQ3, i.e. the predictions obtained by using the combination of GA and SVMs are superior to the ones obtained by the other techniques. As for RQ4 there are differences in the performance achieved with intra- and inter-release prediction for all the considered techniques except for GA+SVM using F-measure as fitness function.

5 Validity Evaluation

Several factors can bias the validity of empirical studies. Here we consider three types of validity threats: *Construct validity*, related to the agreement between a theoretical concept and a specific measuring device or procedure; *Conclusion validity*, related to the ability to draw statistically correct conclusions; *External validity*, related to the ability to generalise the achieved results. As highlighted by Kitchenham *et al.*[20], in order to satisfy construct validity a study has "to establish correct operational measures for the concepts being studied". Thus, the choice of the measures and how to collect them represents the crucial aspects. We tried to mitigate such a threat by evaluating the employed prediction methods on publicly available datasets from the PROMISE repository [24], that have been previously used in other empirical studies (see e.g., [19, 25, 30]). In relation to the conclusion validity, we carefully calculated the employed performance measures and the obtained data was cross-checked by two authors. Finally, observe that also the fact of having used open source software systems could affect the results of the presented study. This means that we cannot conclude that the results of this study promptly apply to other software development settings. This could represent an important external validity threat that can be mitigated only replicating the study on other datasets. Accordingly, we plan to conduct a further investigation on commercial software systems.

6 Related Work

Fault prediction is a very active research field within Software Engineering and many studies have addressed this key issue using a variety of different methods. Some interesting literature reviews are available in [1, 2]. For sake of space, we limit our description to the research that employed Support Vector Machines (SVMs) and Genetic Algorithms (GA) for fault prediction.

SVMs were used in many works together with other classification techniques, obtaining different results. In [8] SVMs were compared against eight modeling techniques in terms of several performance measures (i.e., Accuracy, Recall, Precision, and F-measure) using four datasets from the NASA Metrics Data Program Repository (MDPR) [21]. The results revealed that none of the employed techniques was significantly better than the others. On the other hand, Gondra [12] reported that, on the JM1 dataset in the NASA MDPR [21], SVMs significantly outperformed an Artificial Neural Network, achieving a percentage of correct classifications on the validation set of 87.4%, versus the 72.61% got by the Artificial Neural Network, thus suggesting that SVMs could be a promising technique for predicting fault-proneness software components.

Gray *et al.* [13] also carried out an empirical study employing SVMs on eleven NASA datasets but with a different purpose. Indeed, they aimed to analyze the performance of this technique when only static code metrics were used. Moreover, in this paper a data-preprocessing step was performed including selection of instances and variables, data normalization, and balancing of faulty and non faulty classes. The obtained results, evaluated only in terms of Accuracy, showed that SVMs yielded at an average Accuracy of 70% on the employed datasets. Similarly, Singh *et al.* [26] exploited SVMs on the KC1 NASA dataset [21] with the goal to analyze the relationship between the Objected-Oriented metrics given by Chidamber and Kemerer [7] and fault proneness by means of many indicators, such as Precision, Recall, and Receiver Operating Characteristic (ROC) analysis. The results revealed that some metrics (i.e., CBO, RFC, and SLOC) were significantly related to fault proneness. The authors replicated the study applying SVMs to jEdit [27], unfortunately, authors did not provide enough information to replicate the settings of the experiment, neither in terms of dataset (version, validation, etc.), nor in terms of the SVMs setting employed.

A variation of the regression version of SVMs (namely SVR) for fault prediction was presented in [18]. Indeed, the authors proposed the use of a Fuzzy SVR (FSVR) for predicting software fault number and analyzed whether the fuzzification of the input allowed SVRs to handle unbalanced software metrics datasets. The results obtained employing the MIS and the RSDIMU datasets revealed that FSVR yielded to a lower Mean Squared Error and higher Accuracy respect to the use of SVR.

As we can observe the majority of the above works employed datasets contained in the NASA Metrics Data Program Repository [21]. This is a useful database for fault prediction empirical analyses, but almost all the contained projects are highly unbalanced, since non-faulty components are many times more than the faulty ones. As a consequence, the most of the studies about the use of SVMs employed a random undersampling of the datasets, making impossible for us to replicate the experimentations or compare the results we achieved in this study. We do not know other studies that have addressed the problem of configuring SVMs parameters for fault prediction: usually the parameter Grid-search feature included in LibSVM has been employed (e.g., [13, 26]). We recall that this feature represented the baseline for our experimental study.

The use of a GA for the fault prediction problem is presented in [25], where the authors reported on the results of, the application of GA to dataset jEdit 4.0. Again, no details at all are provided, either on the employed GA, either on the assessment protocol. Moreover, only a cumulative Accuracy of 80.14% is reported, but it is not clear how it was computed.

Finally, we want to highlight that we applied an intra-release prediction analysis and an inter-release prediction analysis and employed the same two datasets of the study presented in [30]. In particular, Watanabe *et al.* [30] exploited the algorithm C4.5 of Weka and a 10-fold cross validation. The results we have achieved using C4.5 on dataset *jE2* are different from those presented in [30] and maybe this is due to the different folds employed for the 10-fold cross validation. As expected, the same results were instead achieved when performing the inter-release prediction analysis. We observe that also RF was applied in the past on dataset *jE1* [19], however the authors did not declare what implementation of RF they employed and what k they used for the k-fold cross validation. Nevertheless, the RF results we achieved are very close to the ones they reported.

7 Conclusions

To improve software quality, it is fundamental to be able to predict defective software components. To address this issue, in this paper we have presented an approach that exploits a Genetic Algorithm (GA) to configure Support Vector Machines (SVMs) for predicting fault-prone software components, on the basis of some structural metrics. In this way, it is possible to fully exploit the potentiality of SVMs, whose performances depend on the configuration of parameters.

The proposed approach has been assessed in an empirical analysis, based on the jEdit data from the PROMISE repository. In particular, we have performed an inter- and an intra-release assessment of the proposal. The latter setting is the most realistic one, since it replicates the typical software development scenario, where data from previous releases are used to predict fault components in a newer version of the software under development. As benchmarks we exploited SVMs with the Grid search provided by LibSVM, and six other techniques frequently used in the context of fault prediction, namely Logistic Regression, Decision Tree Algorithm C4.5, Naïve Bayes, Multi-Layer Perceptrons, K-Nearest Neighbor, and Random Forest.

With respect to the four research questions addressed in the empirical analysis, the results reported in the paper show that:

- the prediction performance of GA+SVM is affected by the choice of the fitness function. The approach represents a flexible tool to support the strategies of project managers that might prefer to maximize a specific performance criterion (for instance Recall rather than Precision);
- GA is able to effectively set SVMs parameters in order to improve fault predictions;

- the fault predictions performance of GA + SVM was superior to the ones obtained by the other techniques;
- the improvement of GA+SVM with respect to the other investigated prediction techniques was especially remarkable for the inter-release use.

As we have mentioned in Section 5, we cannot promptly apply these results to other software systems different from the ones we employed. To address this issue, in the future we intend to replicate the performed analysis using other datasets. This is the only way to get better confidence on the generalizability of the results.

References

1. Arisholm, E., Briand, L., Johannessen, B.: Data mining techniques, candidate measures and evaluation methods for building practically useful fault-proneness prediction models. Simula Research Laboratory Technical Report, 2008-06
2. Arisholm, E., Briand, L., Johannessen, B.: A systematic and comprehensive investigation of methods to build and evaluate fault prediction models. Journal of Systems and Software 83, 2–17 (2010)
3. Baeza-Yates, R., Ribeiro-Neto, B.: Modern Information Retrieval. Addison-Wesley, Reading (1999)
4. Briand, L., Langley, T., Wiekzorek, I.: A Replicated Assessment and Comparison of Common Software Cost Modeling Techniques. In: Procs of the International Conference on Software Engineering, pp. 377–386. IEEE press, Los Alamitos (2000)
5. Corazza, A., Di Martino, S., Ferrucci, F., Gravino, C., Sarro, F., Mendes, E.: How Effective is Tabu Search to Configure Support Vector Regression for Effort Estimation? In: Procs of the International Conference on Predictive Models in Software Engineering, p. 4 (2010)
6. Chang, C.C., Lin, C.-J.: LIBSVM: a library for support vector machines, (2001), Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm
7. Chidamber, S.R., Kemerer, C.F.: A metrics suite for object oriented design. IEEE Transactions on Software Engineering 20(6), 476–493 (1994)
8. Elish, K.O., Elish, M.O.: Predicting defect-prone software modules using support vector machines. Journal of Systems and Software 81(5), 649–660 (2008)
9. Ferrucci, F., Gravino, C., Oliveto, R., Sarro, F.: Genetic Programming for Effort Estimation: an Analysis of the Impact of Different Fitness Functions. In: Procs of the 2nd International Symposium on Search Based Software Engineering, pp. 89–98. IEEE Computer Society, Los Alamitos (2010)
10. Glover, F., Laguna, M.: Tabu Search. Kluwer Academic Publishers, Boston (1997)
11. Goldberg, E.: Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, Reading (1989)
12. Gondra, I.: Applying machine learning to software fault-proneness prediction. Journal of Systems and Software 81, 186–195 (2008)
13. Gray, D., Bowes, D., Davey, N., Sun, Y., Christianson, B.: Using the Support Vector Machine as a Classification Method for Software Defect Prediction with Static Code Metrics. In: Palmer-Brown, D., Draganova, C., Pimenidis, E., Mouratidis, H. (eds.) EANN 2009. Communications in Computer and Information Science, vol. 43, pp. 223–234. Springer, Heidelberg (2009)
14. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The WEKA Data Mining Software: An Update. SIGKDD Explorations 11(1) (2009)
15. Halstead, M.H.: Elements of Software Science. Elsevier North-Holland, New York (1977)

16. Harman, M., Jones, B.F.: Search based software engineering. Information and Software Technology 43(14), 833–839 (2001)
17. Harman, M., Clark, J.A.: Metrics Are Fitness Functions Too. IEEE Metrics, 58–69 (2004)
18. Yan, Z., Chen, X., Guo, P.: Software Defect Prediction Using Fuzzy Support Vector Regression. In: Procs of the International Symposium on Neural Networks, pp. 17–24 (2010)
19. Kaur, A., Malhotra, R.: Application of Random Forest in Predicting Fault-Prone Classes. In: Procs of the International Conference on Advanced Computer Theory and Engineering
20. Kitchenham, B., Pickard, L., Peeger, S.: Case studies for method and tool evaluation. IEEE Software 12(4), 52–62 (1995)
21. NASA – Metrics data program, `http://mdp.ivv.nasa.gov/`
22. Ostrand, T.J., Weyuker, E.J.: How to measure success of fault prediction models. In: Procs of the Fourth Workshop on Software Quality Assurance, pp. 25–30 (2007)
23. Ostrand, T.J., Weyuker, E.J., Bell, R.M.: Predicting the Location and Number of Faults in Large Software Systems. IEEE Trans. Software Eng. 31(4), 340–355 (2005)
24. PROMISE Repository of empirical software engineering data, `http://promisedata.org`
25. Sandhu, P.S., Dhiman, S.K., Goyal, A.: A Genetic Algorithm Based Classification Approach for Finding Fault Prone Classes. World Academy of Science, Engineering and Technology 60 (2009)
26. Singh, Y., Kaur, A., Malhorta, R.: Software Fault Proneness prediction Using Support Vector Machines. In: Procs of the World Congress on Engineering, vol. I, pp. 240–245 (2009)
27. Singh, Y., Kaur, A., Malhorta, R.: Application of Support Vector Machine to Predict Fault Prone Classes. ACM SIGSOFT Software Engineering Notes 34(1) (2009)
28. Vapnik, V., Chervonenkis, A.Y.: Theory of Pattern Recognition (1974) (in Russian)
29. Vapnik, V.: The nature of Statistical Learning Theory. Springer, Heidelberg (1995)
30. Watanabe, S., Kaiya, H., Kaijiri, K.: Adapting a Fault Prediction Model to Allow Inter Language Reuse. In: Procs of the International Conference on Predictive Models in Software Engineering, pp. 19–24 (2008)