A Further Analysis on the Use of Genetic Algorithm to Configure Support Vector Machines for Inter-Release Fault Prediction

F. Sarro
University of Salerno
Via Ponte Don Melillo,
84084 Fisciano (SA), Italy
{fsarro@unisa.it}

S. Di Martino
University of Naples "Federico II"
Via Cintia
80127 Naples, Italy
sergio.dimartino@unina.it

F. Ferrucci, C. Gravino
University of Salerno
Via Ponte Don Melillo,
84084 Fisciano (SA), Italy
{fferrucci | gravino@unisa.it}

ABSTRACT
Some studies have reported promising results on the use of Support Vector Machines (SVMs) for predicting fault-prone software components. Nevertheless, the performance of the method heavily depends on the setting of some parameters. To address this issue, we investigated the use of a Genetic Algorithm (GA) to search for a suitable configuration of SVMs to be used for inter-release fault prediction. In particular, we report on an assessment of the method on five software systems. As benchmarks we exploited SVMs with random and Grid-search configuration strategies and several other machine learning techniques. The results show that the combined use of GA and SVMs is effective for inter-release fault prediction.

Categories and Subject Descriptors
D.2.5 [**Software Engineering**]: Testing and Debugging

General Terms
Management, Experimentation.

Keywords
Fault prediction, Genetic Algorithm, Support Vector Machines

1. INTRODUCTION
In the last decades, considerable efforts have been aimed at developing methods able to predict the components of a software system that more likely will contain faults [1]. The research is motivated by the need to improve the efficiency of software testing, that is one of the most expensive phases of the software development effort. Indeed, knowing in advance the potentially defective components, project managers can better decide how allocate resources to test the system concentrating their efforts on fault-prone components. This can allow them to improve the dependability, the quality, and the cost/effectiveness of the software product. Two main research directions can be outlined in this context. One aims at understanding what are the most explicative metrics for the phenomenon at the hand. Typical employed metrics are the Lines of Code and the Chidamber and Kemerer metrics for Object-Oriented systems [6]. A second research direction aims at defining techniques able to provide a more accurate identification of the faulty components. To this end, several machine learning approaches have been investigated [8][10][20]. Usually they construct prediction models able to relate software metrics with the presence of faults in a component. In this context, recent studies have reported a good performance of Support Vector Machines (SVMs) (e.g., [8][10]). SVMs are supervised learning methods that can be exploited for binary 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. Nevertheless, the use of SVMs is not straightforward since some parameters must be carefully set to get more accurate classifications. Moreover, such setting depends on the characteristics of the dataset, and thus no rule of thumb can be defined but a search strategy has to be adopted. To address the problem, several approaches have been proposed in the different applications of SVMs. As for fault prediction a "Grid-search" [5] approach, with a very coarse grain, has been usually applied. A more sophisticated approach based on a meta-heuristic search technique was proposed in [7] where a preliminary assessment was also provided. In particular, a Genetic Algorithm (GA) was defined to search for suitable SVM parameter settings. Promising results - especially in terms of Recall and F-measure - were obtained highlighting that GA can be effective to configure SVMs. Nevertheless, the assessment was carried out employing data related to only one software project. Motivated by the observation that several studies are necessary to get better confidence on the generalizability of the results, in this paper we report on a further assessment of the performance of the method. To this end, we employed data related to 5 software projects from the PROMISE repository [18], containing more than one release. This let us analyze the inter-release performance of the proposed method, exploiting data from the former releases of a software project to build the prediction model to be used for predicting faults for the last release [17]. This kind of analysis reflects a real software development context, where a project manager exploits knowledge from previous releases of a system for a more conscious management of a subsequent version. As benchmark several techniques were taken into account: SVMs configured with random settings, that is a natural benchmark when using meta-heuristic search techniques; SVMs configured with Grid-search; and other classification methods widely used in previous work for fault prediction (e.g., [8]), namely Logistic Regression, Decision Tree Algorithm C4.5, Naïve Bayes, Multi-Layer Perceptrons, K-Nearest Neighbor, and Random Forest. As measures to compare performance, Accuracy, Precision, Recall, and F-Measure were employed.

ACM SAC'12, March 26-30, 2012, Riva del Garda, Italy.

The remainder of the paper is structured as follows: in Section 2 we present SVMs and the GA used to configure them. Section 3 describes the planning of the empirical study, while in Section 4 we report and discuss the achieved results. Section 5 analyzes the threats to the validity of the empirical study. A review of the related work is presented in Section 6. Final remarks and future work conclude the paper.

2. SUPPORT VECTOR MACHINES AND GENETIC ALGORITHM

We provide a brief description of Support Vector Machines and the proposed Genetic Algorithm to configure them.

2.1 Support Vector Machines

Support Vector Machines (SVMs) are a maximum-margin linear classification technique [19], since they find the optimal hyperplane between two classes defined by a number of support vectors [19]. This technique turns out to be very flexible, since it has good generalization ability, thanks to the introduction of a penalty factor, named *C*, which allows us to cope with the effects of outliers, permitting a certain amount of misclassification errors. Moreover, the use of a kernel function to map the data points into a higher-dimensional feature space allows SVMs to handle also non-linear problems [19].

This flexibility of SVMs needs to be suitably managed. Indeed, the selection of appropriate values for *C* is crucial to obtain good classification performance and avoid over- or under-fitting phenomena. As for the kernel function, we employed the most used and advised in the literature, namely the Radial Basis Functions (RBF) kernel, previously also used in the fault prediction context [7][11]. Besides *C*, we had to accurately set also the parameter γ that represents the radius of the RBF kernel and so can impact on the SVMs accuracy.

To address the problem, several approaches have been proposed in the different applications of SVMs. As an example, the use of the meta-heuristics Tabu Search has been investigated in [4] to configure Support Vector Regression, i.e., the regression version of SVMs, for software development effort estimation. However, in the context of fault prediction the most common approach used to set SVMs parameters is based on a Grid-search with a fixed exponentially growing sequence of values, as the one provided by LibSVM [5], 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 dataset at the hand. A more sophisticated approach based on the use of GA to configure SVMs for fault prediction was proposed in [7]. In the next subsection we summarize this approach since we empirically analyze it in the present paper.

2.2 A Genetic Algorithm to Configure SVMs

Genetic Algorithm (GA) [9] is a meta-heuristic search technique belonging to the class of evolutionary algorithms which, inspired by natural evolution, create consecutive populations of individuals, considered as feasible solutions for a given problem, to search for an optimal solution for the problem under investigation guided by a fitness function.

The idea of exploiting GA to configure SVMs for fault prediction is based on the observation that such problem can be formulated as an optimisation problem: among the possible configurations, we have to identify the one which leads to the optimal SVMs performance.

Despite of a number of variations, the elementary process of a GA is the follows: (i) a random initial population is generated; (ii) 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 some stopping criteria hold. 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 a GA to a given optimization problem: 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 made for points 1-4 to design a GA for configuring SVMs.

A solution to our problem is an SVM configuration consisting of two parameters (C and γ) that can be encoded as a chromosome composed by two genes whose values vary in the ranges [0.01, 32000] and [1.0E-6, 8] for the genes representing C and γ respectively. Let us note that these ranges are the same adopted in the Grid-search provided by LibSVM [5] to allow for comparison. An initial population of n=100 chromosomes is created by assigning random values to each gene. To compute the fitness of a chromosome representing an SVM configuration, we execute SVM with such a configuration thus obtaining the fault predictions. Such predictions are then evaluated using F-measure as performance criterion. F-measure is widely used in the fault prediction context and will be described in Section 3.3. To create the new offspring, genetic operators (i.e., crossover and mutation) are applied within a certain probability to a current population. In particular, a single point crossover is employed to combine two individuals (i.e., parents) forming new individuals by randomly selecting a point of cut and swapping all genes beyond that point in either parent. As for the mutation operator, it modifies with probability 1/2 each gene of a chromosome, randomly changing its value. Crossover and mutation rates are fixed to 0.5 and 0.1, respectively. To determine which individuals will take part of the next generation a tournament selector is employed, where only the best n solutions take part to the next generation. The evolutionary process is terminated according to two stopping criteria, i.e., after 300 generations or if the fitness value of the best solution does not change after 30 generations. The population size, stopping criteria, crossover and mutation rates were empirically determined as it is usual when no guidelines are available. In particular, we observed that increasing them did not allow us to improve the estimation accuracy while wasting computation time. Moreover, to cope with the non-deterministic nature of GA, 10 executions were done and it was retained as final solution the one providing fitness value closest to the mean of the fitness values obtained in the 10 executions.

3. CASE STUDY PLANNING

We investigated the combination of GA and SVM (in the following GA+SVM) for inter-release fault prediction by

exploiting five software projects, aiming at addressing the following research questions:

(RQ1): Is the proposed GA able to effectively configure SVMs parameters for inter-release fault prediction?

(RQ2): Is GA+SVM an effective technique for inter-release fault prediction?

To address RQ1 we first compare the performance of GA+SVM with respect to SVM configured with a simpler approach consisting of the generation of random configurations (SVM-Rand, in the following), which is a usual benchmark when using meta-heuristic search techniques. To be fair, the same number of solutions exploited by GA+SVM (i.e., 300*100 SVM configurations) were generated in a totally random fashion within the same ranges used for GA (see Section 2.2) and among them the best one was selected according to the same criteria employed for GA+SVM but without guiding the search in any way. Moreover, we also compared the performance of GA+SVM with the ones obtained using the Grid-search algorithm provided by LibSVM [5] (SVM-Grid in the following), which is the strategy usually employed in defect prediction context to configure SVMs.

To address RQ2 we compared the performance of GA+SVM with respect to the performances of six techniques widely used in the literature and available in the Weka tool [12], namely 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). These are representative of different classes of techniques for fault prediction, indeed LR and NB are two statistical classifiers, KNN is an instance-based learning algorithm, MLP is a Neural Network method, C4.5 is a decision tree approach and RF is an ensemble method. For sake of space we refer the reader to [3][15] for details on these techniques. In the following, we summarize the design of our empirical study by presenting the employed datasets, the validation method, and the evaluation criteria.

3.1 Datasets

We exploited data from the PROMISE repository [18], which contains several datasets for fault prediction and we chose among them the software projects with more than two releases, in order to perform the inter-release analysis. Thus, we retained 5 datasets for a total of 18 releases: Log4j (vv. 1.0, 1.1, 1.2), Lucene (vv. 2.0, 2.2, 2.4), POI (vv.1.5, 2.0, 2.5, 3.0), Xalan (vv. 2.4, 2.5, 2.6, 2.7), Xerces (vv. init, 1.2,1.3,1.4). Each release contained a set of components (i.e., Java classes) described in terms of Chidamber and Kemerer (CK) metrics [6], Number of Public Methods (NPM), and Lines of Code (LOC). Details about those software projects and how the data was collected can be found in [13].

3.2 Validation Method

To assess the effectiveness of the considered techniques for inter-release fault prediction, we employed a hold-out validation. In particular, given a software project having *n* releases, we used as training set the data collected in the first *n*-1 releases of the project and as test set the data collected for the last release. This kind of validation allowed us to simulate the situation that typically arises in real software development contexts, where a project manager can learn some phenomena and/or patterns from previous releases and exploit this knowledge for a more conscious management of the development of a subsequent version. Moreover, if he/she uses a fault prediction model, the typical setting is that data from the former releases are exploited to build the model to predict faults for a new release [17]. The fault data for the employed training and test sets is reported in Table 1 together with the percentage of faulty and non faulty components. We can observe that for Lucene and POI projects, the training sets and the corresponding test sets have quite similar percentages of faulty and no faulty components. On the contrary, the number of non faulty components contained in Log4j, Xalan, and Xerces training sets is much higher than the number of the faulty ones and viceversa the number of faulty components contained in the corresponding test sets is much higher than the number of the non faulty ones. In particular, for Log4j and Xalan test sets the percentage of faulty classes is more than 90%.

Table 1. Existence of faults on training and test sets

Dataset	Training Set		Test Set	
	Non faulty classes	Faulty classes	Non faulty classes	Faulty classes
Log4j	173 (71%)	71 (29%)	16 (8%)	189 (92%)
Lucene	207 (47%)	235 (53%)	137 (40%)	203 (60%)
POI	510 (54%)	426 (46%)	161 (36%)	281 (64%)
Xalan	1503 (62%)	908 (38%)	11 (1%)	898 (99%)
Xerces	838 (79%)	217 (21%)	151 (27%)	437 (74%)

3.3 Evaluation Criteria

To evaluate the predictions we employed four widely used performance measures (i.e., Accuracy, Precision, Recall, and F-measure [2]) which leverage on the concepts reported in the confusion matrix of Table 2 and are defined as follows.

Accuracy is 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, FN).

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

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

Table 2. The confusion matrix

		Predicted	
		Faulty	Non Faulty
Actual	Faulty	True Positive (TP)	False Negative (FN)
	Non Faulty	False Positive (FP)	True Negative (TN)

These definitions suggest that Precision concerns with the correctness of the responses provided by a method, while Recall allows us to measure the completeness of the responses. We can easily understand that in the context of fault prediction the consequences of low Recall are far more important than the ones of low Precision [17], since it is essential to identify all the potentially faulty modules. Thus, the use of a technique able to maximize Recall is fundamental in this context [17]. To avoid missing potential faulty components, the prediction method has to detect all the potential faulty components, even at the cost of a lower Precision. In this scenario, a subsequent manual check is responsible of deleting false positives. However, 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 follow:

$$F-measure = 2 * \frac{\mathrm{Pr}ecision * \mathrm{Re}call}{\mathrm{Pr}ecision + \mathrm{Re}call}.$$

Let us recall that we employed F-measure as fitness function for the proposed GA aiming at taking into accounts both correctness and completeness.

4. RESULTS AND DISCUSSION

In this section, we present the results of the empirical study.

4.1 Research Question RQ1

To assess whether the proposed GA is able to effectively configure SVMs parameters for fault prediction (i.e., RQ1) we compared the predictions obtained using GA+SVM with those obtained by using random SVM configurations (i.e., SVM-Rand) and by applying SVMs configured with Grid-search of LibSVM [5] (i.e., SVM-Grid). Table 3 reports on the results achieved by these techniques in terms of the performance measures Accuracy, Precision, Recall, and F-measure. We can observe that GA+SVM obtained better results than SVM-Rand on all the datasets. In particular, using GA+SVM we obtained high relative improvements[1] with respect to SMV-Rand in terms of the Accuracy, Recall, and F-measure values averaged on all the datasets (i.e., 35%, 80%, and 48%, respectively), paying back only 4 percentage points with respect to the Precision obtained by SVM-Rand.

Table 3. Results for RQ1

Dataset	Technique	Results			
		Accuracy	Precision	Recall	F-measure
Log4j	GA+SVM	**0.35**	**0.94**	**0.31**	**0.47**
	SVM-Rand	0.18	0.86	0.12	0.21
	SVM-Grid	0.31	**0.94**	0.26	0.41
Lucene	GA+SVM	0.61	0.62	**0.90**	**0.73**
	SVM-Rand	0.61	0.68	0.65	0.66
	SVM-Grid	**0.67**	**0.74**	0.68	0.71
POI	GA+SVM	0.67	0.69	**0.86**	**0.77**
	SVM-Rand	0.62	0.76	0.59	0.66
	SVM-Grid	**0.74**	**0.92**	0.63	0.75
Xalan	GA+SVM	**0.43**	**1.00**	**0.43**	**0.60**
	SVM-Rand	0.35	**1.00**	0.34	0.50
	SVM-Grid	-	-	-	-
Xerces	GA+SVM	**0.47**	**0.94**	**0.30**	**0.46**
	SVM-Rand	0.32	0.87	0.11	0.19
	SVM-Grid	-	-	-	-

Concerning the comparison with SVM-Grid, first of all we observe that for dataset Log4j, GA+SVM provided better predictions than SVM-Grid in terms of Accuracy, Recall, and F-measures (with an improvement of 13%, 19%, and 14%, respectively) and same Precision. As for Lucene and POI datasets, GA+SVM provided better performance than SVM-Grid in terms of Recall and F-measure, while SVM-Grid provided better Accuracy and Precision. Indeed, on these datasets the relative improvement of GA+SVM with respect to the use of SVM-Grid was high in terms of Recall (i.e., 33% and 36% for Lucene and POI, respectively) and good in terms of F-measure (i.e., 14% and 3%, for Lucene and POI, respectively). On the other hand, the Accuracy slightly decreased by 9 and 10 percentage points for Lucene and POI, respectively, while a higher decreasing was observed for Precision, i.e., 17% and 25%, for Lucene and POI, respectively. However, the obtained Recall improvement of GA+SVM with respect to SVM-Grid is higher than the relative improvement of Precision and Accuracy of SVM-Grid with respect to GA+SVM. Finally, we observe that on the Xalan and Xerces projects SVM-Grid classified all the observations in the test set as non faulty, thus failing to provide its predictions. Obviously this is due to the fact that the Grid-search was not able in these cases to identify suitable SVR parameters. On the contrary, GA+SVM did never exhibit a similar behavior, thus performing better than SVM-Grid on those datasets.

The above analysis suggests that the use of GA to configure SVMs allowed us to obtain better results with respect to the ones obtained using SVM-Rand and SVM-Grid, especially in terms of Recall and F-Measure. This is particularly interesting since the consequences of low Recall are far more important than the ones of low Precision in the context of fault prediction [17]. Thus, we can positively answer research question RQ1 and confirm the result achieved in [7], i.e., GA is able to effectively set SVM configuration parameters for fault prediction.

4.2 Research Question RQ2

Table 4 reports on the performance measures related to the considered techniques, namely LR, C4.5, NB, MLP, KNN, and RF, taken into account as benchmarks to address RQ2. For sake of readability we report the results for GA+SVM as well. Moreover, we provide also the coefficient of variation[2] (CV) for each performance measure to have an indication about the dispersion of the distribution of each measure over all the employed techniques. Let us observe that for all datasets, the highest values of CV are related to Recall and F-measure thus highlighting that the performance of the considered techniques is more different in terms of these measures. This difference is especially evident for some datasets (i.e., POI and Xerces). Let us observe also that the smallest CV values are in 4 out 5 cases related to Precision. Moreover, these values are very small thus showing that the performances of the considered techniques are generally quite similar in terms of Precision. In particular, in three cases (i.e., Log4j, Xalan, and Xerces datasets) several techniques provided optimal Precision values (1 or close to 1). This is due to the fact that the percentage of faulty components contained in Log4j, Xalan, and Xerces test sets (i.e., 92%, 99%, and 74%, respectively) is higher than the percentage of non faulty ones (i.e., 8%, 1%, and 26%, respectively), thus limiting the number of possible False Positives. As for Accuracy, quite small CV values were obtained for Log4j and Lucene datasets, while higher values were got for the other datasets indicating for them a superior dispersion. Taking into account these observations, in the following we carry out the comparison among GA+SVM and the other techniques for each dataset focusing our attention on the measures showing higher dispersion. Concerning Log4j dataset we observe that the performance of GA+SVM is comparable with the ones of the other techniques with respect to all measures. As for Lucene dataset, among the benchmark techniques the best performances were obtained by LR, KNN, and RF. With respect to these techniques GA+SVM got a high improvement in terms of Recall (50%, 38%, and 186% over LR, KNN, and RF, respectively) and similar values in terms of the other measures.

[1] To compute the relative improvement of a technique A with respect to a technique B for a measure M we compare the values achieved with A and B (i.e., M_A and M_B, respectively) as follows: $((M_A - M_B) / M_B)*100$.

[2] The coefficient of variation is defined as the ratio of the standard deviation to the mean of a distribution and provides a normalized measure of the dispersion of the distribution.

Also on POI dataset, GA+SVM got the best Recall with a relative improvement of 17% over the best benchmark technique (i.e., C4.5).

Table 4. Results for RQ2

Dataset	Technique	Results			
		Accuracy	Precision	Recall	F-measure
Log4j	GA+SVM	**0.35**	0.94	**0.31**	**0.47**
	LR	0.31	**0.98**	0.26	0.41
	C4.5	0.35	0.97	0.30	0.46
	NB	0.32	**0.98**	0.27	0.42
	MLP	0.31	**0.98**	0.25	0.40
	KNN	0.34	0.95	0.30	0.46
	RF	0.35	0.95	0.31	0.46
	CV	0.06	0.02	0.09	0.07
Lucene	GA+SVM	0.61	0.62	**0.90**	**0.73**
	LR	**0.63**	0.73	0.60	0.66
	C4.5	0.62	0.74	0.55	0.63
	NB	0.56	**0.87**	0.32	0.46
	MLP	0.57	0.80	0.38	0.52
	KNN	0.62	0.67	0.66	0.67
	RF	0.60	0.67	0.65	0.66
	CV	0.04	0.12	0.33	0.15
POI	GA+SVM	**0.67**	0.69	**0.86**	**0.77**
	LR	0.55	0.73	0.47	0.57
	C4.5	**0.67**	0.74	0.74	0.74
	NB	0.45	**0.85**	0.16	0.28
	MLP	0.39	0.59	0.12	0.20
	KNN	0.64	0.74	0.67	0.70
	RF	0.62	0.75	0.60	0.67
	CV	0.20	0.11	0.55	0.41
Xalan	GA+SVM	**0.43**	**1.00**	**0.43**	**0.60**
	LR	0.19	**1.00**	0.18	0.31
	C4.5	0.30	**1.00**	0.29	0.45
	NB	0.17	**1.00**	0.16	0.27
	MLP	0.28	**1.00**	0.27	0.42
	KNN	0.42	**1.00**	0.41	0.58
	RF	0.42	**1.00**	0.42	0.59
	CV	0.35	0.00	0.37	0.30
Xerces	GA+SVM	**0.47**	0.94	**0.30**	**0.46**
	LR	0.28	0.92	0.03	0.05
	C4.5	0.31	0.90	0.08	0.15
	NB	0.32	0.88	0.10	0.18
	MLP	0.27	**1.00**	0.02	0.04
	KNN	0.36	0.90	0.15	0.26
	RF	0.35	0.88	0.15	0.27
	CV	0.20	0.05	0.80	0.72

Concerning Xalan dataset, GA+SVM provided comparable results with respect to the best benchmark techniques (i.e., KNN and RF), while on Xerces dataset, GA+SVM obtained better performance with respect to the best benchmarks (i.e., KNN and RF) providing a high improvement in terms of Recall and F-measure (up to 104% and 80%, respectively). Finally, we can observe that, in average over all datasets, GA+SVM allowed us to obtain high relative improvements in terms of Accuracy, Recall, and F-measures (i.e., 31%, 180%, 113%, respectively) with respect to the mean of the results provided by the other techniques, paying back only 4 percentage points in terms of Precision. Nevertheless, some differences can be observed over the datasets. Indeed the performance of GA+SVM is better for Lucene and POI datasets and not so good for the others where we observe low Accuracy and Recall. The main reason for that could be related to the opposite distribution of percentage of faulty and non faulty components between the training and test sets for Log4j, Xalan, and Xerces datasets. Indeed, the high percentages of non faulty components in the training sets could lead the model to predict more components as non faulty. This, together with the fact that in the corresponding test sets the percentages of non faulty components are very low, increases the probability to have False Negatives, thus affecting Recall and Accuracy. On the other hand, it is worth noting that for Xerces dataset some techniques got a Recall near 0, making the performance of GA+SVM (i.e., 0.30) especially remarkable.

The above analysis suggests that GA+SVM behaves consistently well over the considered datasets performing similarly or better than the other techniques and often providing a great improvement in terms of Recall and F-measure. Thus, we can positively answer research question RQ2, i.e., GA+SVM is an effective technique for fault prediction.

5. VALIDITY EVALUATION

The validity of our empirical studies can be biased by different factors. First of all, threats to construct validity can be due to the way the defect and the CK metrics were collected. We tried to mitigate such a threat employing publicly available datasets from the PROMISE repository [18] that have also been used in previous case studies on fault prediction. Threats to internal validity can be due to the bias introduced by the intrinsic randomness of GA. We mitigate such a threat by executing GA ten times and using the average results as detailed in Section 2.2. As for the external validity, it can be affected by the fact that all the employed projects were open-source. Indeed, there could be some differences between open-source and industrial development (e.g., some industrial settings enforce standards of code quality). However, this threat is mitigated by the fact that the employed projects are related to the Apache community, thus, while being open-source, they have a strong industrial background. A second threat concerns the language. Since all considered systems are written in Java, the obtained results cannot be generalized to non-Java software projects. Moreover, each project is related to a different type of software, was developed by an independent development team, and differs from others for size, percentage of faulty and non faulty components, and magnitude of CK metrics. Thus, we are confident that our results can promptly apply to software systems having similar characteristics, however further studies are desiderable to corroborate the obtained results.

6. RELATED WORK

Many studies have addressed the fault prediction issue using a variety of different methods. Arisholm *et al.* [1] provides an interesting literature review. For sake of space, we limit our description to the research that employed Support Vector Machines (SVMs). Apart from [6] we do not know other studies

that have addressed the problem of configuring SVMs parameters for fault prediction; usually the Grid-search feature included in LibSVM has been employed (e.g., [11]). We recall that this feature represented the baseline for our experimental study. As for SVMs they have been used in many works together with other classification techniques, obtaining different results. In [8] SVMs were compared against eight modeling techniques in terms of the performance measures Accuracy, Recall, Precision, and F-measure, using four datasets from the NASA Metrics Data Program Repository (MDPR) [16]. The results revealed that none of the employed techniques was significantly better than the others. On the other hand, Gondra [10] reported that, on the JM1 dataset in the NASA MDPR [16], SVMs significantly outperformed an Artificial Neural Network, thus suggesting that SVMs could be a promising technique for predicting fault-proneness software components. Gray *et al.* [11] also carried out an empirical study employing SVMs on eleven NASA datasets [16] to analyze the performance of this technique when only static code metrics were used. The obtained results, evaluated only in terms of Accuracy, showed that SVMs yielded at an average Accuracy of 70%. It is worth noting that the above works employed data from the NASA MDPR [16]. This is a useful database for fault prediction empirical analyses, but all the contained projects refer to only one release, thus making impossible for us to employ them in our inter-release analysis.

7. CONCLUSIONS

We have investigated the use of a Genetic Algorithm (GA) to configure Support Vector Machines (SVMs) for inter-release fault prediction. A first assessment of the approach was performed in [7] by employing information from two releases of the software jEdit included in the PROMISE repository. The results were interesting, since using GA to configure SVMs improved the performances of SVMs with respect to the use of SVM-Grid and other six techniques, namely LR, C4.5, NB, MLP, KNN, and RF, especially in terms of Recall. In this paper, we have further investigated the combination of GA and SVMs by considering other 5 datasets from the PROMISE repository, namely Log4j, Lucene, POI, Xalan, and Xerces. The performed empirical analysis has confirmed that: (i) GA is able to effectively set SVMs parameters in order to improve fault predictions; (ii) GA+SVM is an effective technique for inter-release fault prediction. Indeed the approach behaved consistently well over the considered datasets performing similarly or better than the other techniques and often providing a great improvement in terms of Recall and F-measure. Nevertheless, it is worth noting that the time performance of GA+SVM is slightly lower than the other benchmarks (although similar to SMV-Grid). However, we realized a tool that automatically extracts the necessary data from software releases and allows project managers to exploit GA+SVM and evaluate the obtained predictions in terms of Accuracy, Precision, Recall, and F-measures. Thus, time performance costs are negligible with respect to the prediction improvements provided by GA+SVM. As future work we will assess the effectiveness of the proposed technique also for cross-project fault prediction, i.e., transferring prediction models from one software project to another [20].

8. REFERENCES

[1] 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, (2010), 2–17.

[2] Witten, I., Frank, E.: Data Mining: Practical Machine Learning Tools and Techniques. Morgan Kaufmann, 2nd Ed., San Francisco (2005).

[3] Breiman, L.: Random Forests. Machine Learning 45 (1), (2001), 5–32.

[4] 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 PROMISE Procs, ACM NY, (2010), 4.

[5] 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.

[6] Chidamber, S.R., Kemerer, C. F.: A metrics suite for object oriented design. IEEE TSE, 20(6), (1994), 476–493.

[7] Di Martino, S., Ferrucci, F., Gravino, C., Sarro: A Genetic Algorithm to Configure Support Vector Machines for Predicting Fault-Prone Components. In PROFES Procs, LNCS Springer vol. 6759, (2011), 186-201.

[8] Elish, K.O., Elish, M.O.: Predicting defect-prone software modules using support vector machines. Journal of Systems and Software 81(5), (2008), 649-660.

[9] Goldberg, E.: Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, (1989)

[10] Gondra, I.: Applying machine learning to software fault-proneness prediction. Journal of Systems and Software 81 (2008), 186–195.

[11] 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. Communications in Computer and Information Science, 43 (2009), 223-234.

[12] 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).

[13] Jureczko, M, Madeyski, L.: Towards identifying software project clusters with regard to defect prediction. In the PROMISE Procs., ACM NY, (2010), 9.

[14] Kitchenham, B., Pickard, L., Peeger, S.: Case studies for method and tool evaluation. IEEE Software 12(4), (1995), 52-62.

[15] Kotsiantis, S. B.: Supervised Machine Learning: A Review of Classification Techniques. Informatica 31 (2007) 249-268

[16] NASA – Metrics data program. http://mdp.ivv.nasa.gov/

[17] Ostrand, T. J., Weyuker, E. J.: How to measure success of fault prediction models. In the WSQA Procs, (2007), 25-30.

[18] PROMISE Repository of empirical software engineering data, http://promisedata.org.

[19] Vapnik, V.: The nature of Statistical Learning Theory. Springer-Verlag, (1995).

[20] Watanabe, S., Kaiya, H., Kaijiri, K.: Adapting a Fault Prediction Model to Allow Inter Language Reuse. In the PROMISE Procs, ACM NY, (2008), 19-24.