Murillo-Morera *et al. Journal of Software Engineering Research and Development* (2017) 5:4
DOI 10.1186/s40411-017-0037-x

Journal of Software Engineering Research and Development

RESEARCH Open Access

A genetic algorithm based framework for software effort prediction

Juan Murillo-Morera[1*], Christian Quesada-López[1], Carlos Castro-Herrera[2] and Marcelo Jenkins[1]

*Correspondence:
juan.murillomorera@ucr.ac.cr
[1]Center for ICT Research, University of Costa Rica, San Pedro de Montes de Oca, San José, Costa Rica
Full list of author information is available at the end of the article

Abstract

Background: Several prediction models have been proposed in the literature using different techniques obtaining different results in different contexts. The need for accurate effort predictions for projects is one of the most critical and complex issues in the software industry. The automated selection and the combination of techniques in alternative ways could improve the overall accuracy of the prediction models.

Objectives: In this study, we validate an automated genetic framework, and then conduct a sensitivity analysis across different genetic configurations. Following is the comparison of the framework with a baseline random guessing and an exhaustive framework. Lastly, we investigate the performance results of the best learning schemes.

Methods: In total, six hundred learning schemes that include the combination of eight data preprocessors, five attribute selectors and fifteen modeling techniques represent our search space. The genetic framework, through the elitism technique, selects the best learning schemes automatically. The best learning scheme in this context means the combination of data preprocessing + attribute selection + learning algorithm with the highest coefficient correlation possible. The selected learning schemes are applied to eight datasets extracted from the ISBSG R12 Dataset.

Results: The genetic framework performs as good as an exhaustive framework. The analysis of the standardized accuracy (SA) measure revealed that all best learning schemes selected by the genetic framework outperforms the baseline random guessing by 45–80%. The sensitivity analysis confirms the stability between different genetic configurations.

Conclusions: The genetic framework is stable, performs better than a random guessing approach, and is as good as an exhaustive framework. Our results confirm previous ones in the field, simple regression techniques with transformations could perform as well as nonlinear techniques, and ensembles of learning machines techniques such as SMO, M5P or M5R could optimize effort predictions.

Keywords: Software effort estimation, Machine learning, Effort prediction model, Genetic approach, Learning schemes, Function points, ISBSG dataset, Empirical study

1 Background

Providing accurate software effort prediction models is complex but necessary for the software industry (Moløkken and Jørgensen 2003). Software effort prediction models have been studied for many years, but empirical evaluation has not led to simple nor consistent ways to interpret their results (Shepperd and MacDonell 2012). Many software companies are still using expert judgment as their preferred estimation method, thus producing inaccurate estimations and severe schedule overruns in many of their projects (Boehm 1981). Software project managers need to be able to estimate the effort and cost of development early in the life cycle, as it affects the success of software project management (Huang et al. 2015).

Several prediction models have been evaluated in the literature and inconsistent findings have been reported regarding which technique is the best (Jorgensen and Shepperd 2007; Shepperd 2007; Dejaeger et al. 2012; Shepperd and MacDonell 2012). The results of these studies are not univocal and are often highly technique-and dataset-dependent. Since there are many models that can fit to certain datasets, the selection of the most efficient prediction model is crucial (Shepperd 2007; Mittas and Angelis 2008). Additionally, the results of different studies are difficult to compare due to different empirical setups and data preprocessing, possibly leading to contradictory results. The issue of which modeling technique to use for software effort estimation remains an open research question (Dejaeger et al. 2012).

The automated selection and combination of techniques in alternative ways could improve the overall accuracy of the prediction models for specific datasets (Dejaeger et al. 2012; Malhotra 2014). The motivation behind the use of these methods is to make minimal assumptions about the data used for training. In this sense, genetic algorithms are search-based algorithms that are much faster than exhaustive search procedures (Malhotra 2014). According to Harman (2007), search based (SB) optimization techniques have been applied to a number of software engineering (SE) activities, of all optimization algorithms, genetic algorithms have been the most widely applied search technique in SBSE.

In our genetic approach, we address how to select the data preprocessing, attribute selection techniques and the learning algorithms automatically according to the characteristics of a specific data set. The main goal is to increase prediction performance optimizing processing time. In this case, the automatic selection of the learning scheme (preprocessing + attributes selection + learning algorithms) is determined by using a genetic approach. For example, the decision of how to select the different techniques considering the characteristics of a specific dataset through of genetic algorithms could be considered a search-based problem for software engineering.

This paper reports an empirical validation of an automated genetic framework. In total, 600 learning schemes that include the combination of 8 data preprocessors, 5 attribute selectors, and 15 modeling techniques represented our search space. The genetic framework through the elitism technique selects the best learning schemes automatically based on the highest correlation coefficient.

In this study, we conducted a sensitivity analysis across different genetic configurations to evaluate the stability of an automated genetic framework and to discover which genetic configurations report best performance. Further, we compared the automated genetic framework performance with a baseline random guessing (Langdon et al. 2016) and an exhaustive framework (Quesada-Lopez et al. 2016). Then, we analyzed the performance of the best learning schemes. We aim to find the best framework configuration (generation and population, mutation levels, crossover levels) according to given data set context. After that, we want to compare the genetic framework results with the exhaustive framework results in order to determine if our genetic approach presents similar solutions to the best solutions found by the exhaustive approach. Besides, we would like to evaluate if evaluation and prediction phases reported similar results as it would mean that our approach is reliable between phases. Finally, we would like to know the most frequently learning schemes selected and the learning schemes that reports the best performance in the effort prediction domain. The main contribution of this empirical study is that it presents a genetic framework, which can be used to automatically determine the best learning schemes to use according to the characteristics of a specific data set. Our approach is different respect to others because we use a full model (data preprocessing + attribute selection + learning algorithm) in our fitness function thus maximizing the three components.

This study generates through genetic algorithms software effort prediction models based on function point measure using the International Software Benchmarking Standards Group (ISBSG R12) Dataset (Hill 2010; ISBSG 2015) and evaluates their effectiveness. We analyze the performance of effort prediction models based on the base functional components (BFC) and unadjusted function point size (UFP) (Albrecht 1979; Jeng et al. 2011). We have established the following research questions as the focus of our analysis. We address each of these questions in the section referenced in parentheses.

- **RQ1**. Which genetic framework configuration (generation and population, mutation levels, crossover levels) did report the best performance when compared to the baseline exhaustive framework? (Section 5.1).
- **RQ2**. Is the performance of the genetic framework similar between evaluation and prediction phases? (Section 5.2).
- **RQ3**. Which are the learning schemes (data preprocessors, attribute selectors, learning algorithms) more frequently selected by the genetic framework? (Section 5.3).
- **RQ4**. Which learning schemes did report the best performance according evaluation criteria metrics? (Section 5.4).

The remainder of the paper is structured as follows: Section 2 briefly provides information on previous studies in effort prediction. Section 3 describes the genetic framework to select and evaluate automatically effort prediction models. Section 4 details the experimental design, and Section 5 presents the analysis and interpretation of results. Finally, Section 6 outlines conclusions and future work.

2 Related work

Several formal models have been employed in software effort prediction using a number of data mining techniques (Jorgensen and Shepperd 2007; Wen et al. 2012). These include several regression analysis techniques, neural networks, instance-based learners, tree/rule-based models, case-based reasoners, lazy learning, bayesian classifiers, support vector machines, and ensembles of learners (Jorgensen and Shepperd 2007; Shepperd and MacDonell 2012). Most studies evaluate only a limited number of modeling techniques on a dataset, which limits the generalization of results. In addition, the results of different studies are difficult to compare due to their different empirical setups, data preprocessing, and dataset characteristics (Shepperd and MacDonell 2012; Langdon et al. 2016). Therefore, the issue of which modeling technique to use for software effort estimation remains an open research question (Dejaeger et al. 2012).

2.1 Frameworks for benchmarking prediction models

Several prediction models have been evaluated in the literature and inconsistent findings have been reported regarding which technique is the best (Shepperd and MacDonell 2012; Dejaeger et al. 2012). For example, in (Jørgensen 2004) differences between model-based and expert-based estimation results were found. In (Mair and Shepperd 2005) regression and analogy methods for effort estimation where compared and conflicting evidence were found. In (Dejaeger et al. 2012) the authors conducted a large scale benchmarking study using different types of techniques and analyzed aspects related with the selection of features. The results indicated that ordinary least squares regression in combination with a logarithmic transformation performs best; however, the combination of other techniques could obtain similar results. Similar results were found in previous work conducted for the authors of this paper (Quesada-Lopez et al. 2016; Murillo-Morera et al. 2016a; Quesada-Løpez et al. 2016). In (Keung et al. 2013), ninety predictors are evaluated with 20 datasets, they used 7 performance measures to determine stable rankings of different predictors. They concluded that regression trees or analogy-based methods are the best performers and offered means to address the conclusion instability issue. In (Huang et al. 2015) several data preprocessing techniques were empirically assessed on the effectiveness of machine learning methods for effort estimation. The results indicate that data preprocessing techniques may significantly influence the predictions, but sometimes it might have negative impacts on prediction performance. They concluded that a careful selection is necessary according to the characteristics of machine learning methods, as well as the datasets.

In consequence, a number of frameworks for benchmarking prediction models in software effort estimation have been proposed. The main motivation of these studies is to achieve an unbiased criterion when comparing different software estimation models and evaluate the effectiveness of data preprocessing techniques, data attribute selectors, and machine learning algorithms in the context of software effort estimation. For example, in (Shepperd and MacDonell 2012) the authors proposed a framework for evaluating prediction systems to reduce the inconsistency amongst validation study results and provide a more formal foundation to interpret results on continuous prediction systems. The use of an unbiased statistic will assist in performing future meta-analyses and in providing more robust and usable recommendations to practitioners. In (Menzies and Shepperd 2012; Keung et al. 2013), conclusion instability in prediction systems is discussed with the intention of providing a framework for studies in the area. The paper analyzed known sources of instability such as the bias measures, variance from sampling, pre-processing and others; after that, it provided recommendations in order to reduce the instability problems. The authors state that an interesting research possibility is to tune the data mining and machine learning techniques using feedback from the domain. This approach generates the learner for a particular dataset. Finally, they concluded that learning learners is an active research area and much further work is required before we can understand the costs and benefits of this approach. In (Song et al. 2013) the authors proposed a framework to investigate to what extent parameter settings affect the performance of learning machines in software effort estimation, and what learning machines are more sensitive to their parameters. They concluded that different learning machines have different sensitivity to their parameter settings. Finally, (Dolado et al. 2016; Langdon et al. 2016) proposed a measure based on a random guessing framework to compare methods for software estimation.

2.2 Effort prediction approaches using genetic algorithms

Harman and Jones (2001) stated that software engineering is ideal for the application of metaheuristic search techniques, such as genetic algorithms, simulated annealing and tabu search. They argued that such search-based techniques could provide solutions to the difficult problems of balancing competing (and sometimes inconsistent) constraints and may suggest ways of finding acceptable solutions in situations where perfect solutions are either theoretically impossible or practically infeasible. In their work they briefly set out key ingredients for successful reformulation and evaluation criteria for search-based software engineering.

In (Harman 2007), Harman described a study on the application of optimization techniques in software engineering. The optimization techniques in Harman's work came from the operations research and metaheuristic computation research communities. His research reviewed the used optimization techniques and the key ingredients required for their successful application to software engineering, providing an overview of existing results in eight software engineering application domains. Harman's paper also described the benefits that are likely to accrue from the growing body of work in this area and provided a set of open problems, challenges and areas for future work. He stated that for new areas of software engineering that have yet to be attacked using search based approaches it remains acceptable to experiment with a variety of search algorithms in order to obtain baseline data and to validate the application of search. But, in order to develop the field of search-based software engineering, a reformulation of classic software engineering problems as search problems is required. In (Harman et al. 2012), the authors argued that search based software engineering has proved to be a very effective way of optimizing software engineering problems. Nevertheless, its full potential as a means of dynamic adaptivity remains under explored.

In (Lefley and Shepperd 2003), the authors investigated the use of various techniques including genetic programming with public data sets. They attempted to model and estimate software project effort. They analyzed when a genetic program can offer better solution search using public domain metrics rather than company specific ones. The study also offered insights into genetic programming performance. They determined that genetic programming performed consistently well, but was harder to configure. In (Arcuri and Fraser 2011; Sayyad et al. 2013), the authors conducted a comprehensive study analyzing the impact of parameter settings in machine learning and software effort estimation. They performed a large study of parameter settings using genetic algorithms. Their results showed that parameter tuning can have critical impact on algorithmic performance, and that overfitting of parameter tuning is a serious limitation of empirical studies in search-based software engineering. In (Aljahdali and Sheta 2013), the authors argued that recently, computational intelligence paradigms were explored to handle the software effort estimation problem with promising results. In this paper they evolve two new models for software effort estimation using Multigene Symbolic Regression Genetic Programming. One model utilizes the source line of code as input variable to estimate the effort; while the second model utilizes the inputs, outputs, files, and user inquiries to estimate the function points. Finally, in (Chen et al. 2017), the authors proposed that instead of mutating a small population, building a large initial population which is then culled using a recursive bi-clustering binary chop approach. They evaluated this approach on multiple software engineering models, unconstrained as well as constrained, and compared its performance with standard evolutionary algorithms. Using just a few evaluations (under 100), they can obtain the comparable results to standard evolutionary algorithms.

In (Singh and Misra 2012), the authors argued that COCOMO is used as algorithmic model and an attempt is being made to validate the soundness of genetic algorithm technique using NASA project data. The main objective of this research is to investigate the effect of crisp inputs and genetic algorithm techniques on the accuracy of system's output when a modified version of the famous COCOMO model is applied to the NASA dataset. In (Ghatasheh et al. 2015), a firefly algorithm is proposed as a metaheuristic optimization method for optimizing the parameters of three COCOMO-based models. These models include the basic COCOMO model and other two models proposed in the literature as extensions of the basic model. The developed estimation models are evaluated using different evaluation metrics. Experimental results show high accuracy and significant error minimization of firefly algorithm over other metaheuristic optimization algorithms including genetic algorithms and particle swarm optimization.

2.3 Techniques and algorithms applied to software effort estimation models

Several techniques have been applied to the field of software effort prediction. In our study, we applied and evaluated different data preprocessing approaches, attribute selector techniques, and machine learning algorithms, some of them representing groups of learning algorithms. In Table 1, we present a summary of techniques and algorithms based on previous literature in the domain of software effort prediction and other prediction contexts (Witten and Frank 2005; Song et al. 2011; Dejaeger et al. 2012). The data preprocessing approaches are represented by the tag (PP), the attribute selector techniques are represented by the tag (AS), and the learning algorithms are represented by the tag (LA). In the following sections, we briefly discuss each of the included techniques.

Table 1 Data mining techniques for learning schemes

DP	Id	Name	AS	Id	Name	LA	Id	Name
1	None	None	1	GS	GeneticSearch	1	GP	GaussianProcesses
2	Log	Logartihmic	2	BF	BestFirst	2	LMS	LeastMedSq
3	BC(-2)	Box-Cox(λ = -2)	3	LFS	LinearForwardSelection	3	LR	LinearRegression
4	BC(-1)	Box-Cox(λ = -1)	4	BE	Backward Elimination	4	MP	MultilayerPerceptron
5	BC(-.5)	Box-Cox(λ = -0.5)	5	FS	Forward Selection	5	RBFN	RBFNetwork
6	BC(.5)	Box-Cox(λ = 0.5)				6	SMO	SMOreg
7	BC(1)	Box-Cox(λ = 1)				7	AR	AdditiveRegression
8	BC(2)	Box-Cox(λ = 2)				8	BGN	Bagging
						9	CR	ConjunctiveRule
						10	DT	DecisionTable
						11	M5R	M5Rules
						12	ZR	ZeroR
						13	DS	DecisionStump
						14	M5P	M5P
						15	RT	REPTree

2.3.1 Data preprocessing (PP)

According to (Witten and Frank 2005), Data preprocessing is a group of techniques for clean the data set with different noise levels, delete missing values, and processing outliers among others funcionalities. Table 2 presents a summary of the data preprocessing techniques. For more details see (Witten and Frank 2005), p.432.

2.3.2 Attribute Selector (AS)

According to (Witten and Frank 2005), most machine learning algorithms are designed to learn which are the most appropriate attributes to use for making their decisions. For example, decision tree methods choose the most promising attribute to split on at each point and should in theory never select irrelevant or unhelpful attributes. Table 3 presents a summary of the attribute selector techniques. For more details, see (Witten and Frank 2005)p.487.

2.3.3 Learning Algorithm (LA)

According to (Witten and Frank 2005), learning algorithms are models that help to build classifiers. The main categories of classifiers are: Beyesian, Trees, Rules, Functions, Lazy, multi-instance and miscellaneous. Table 4 presents a summary of the learning algorithms techniques.

3 Genetic effort prediction framework

This section presents the main characteristics and steps of the automated genetic effort prediction framework. Our framework consists of two components: 1) Learning Scheme Generator and Evaluator and 2) Effort Prediction. The component of the learning schemes, generator and evaluator, is characterized by the generation and evaluation of learners. The best learning scheme is selected according to the maximum Correlation Coefficient (CC) computed in the fitness function. Moreover, the effort prediction component is characterized by the generation of the predictor (see Fig. 1).

The effort prediction models are built before and then evaluated. After that, the predictive performance of the learning schemes could be used as a reference, particularly for future data. The framework was constructed to support a continuous value as an output variable. It divides the datasets randomly into historical data represented by the 90% and the new data represented by the 10% before the N-PASS process (number N of PASS (executions) of our approach. The objective to avoid performance variability calculating the average of the executions (Song et al. 2011) generating the same subsets of instances per each dataset before each evaluation. Finally, the framework evaluates the 10% of the whole data, avoiding the overfitting. The overfitting refers to the condition where a predictive

Table 2 Data preprocessing

PP	Description
None(None)	This data preprocessing presents the data unchanged.
Logarithmic(Log)	In this data preprocessing, all the numeric values are replaced by their logarithmic values.
Box-Cox(BC)	Is a parametric power transformation technique in order to reduce anomalies such as non-normality and heteroscedasticity. This data preprocessing was introduced by Tukey (1957) as a family of power transformations such that the transformed values are a monotonic function of the observations over admissible range. (Sakia 1992)

Table 3 Attribute selector

AS	Description
Genetic Search (GS)	Uses a simple genetic algorithm. Parameters include population size, number of generations, and probabilities of crossover and mutation.
BestFirst (BF)	This attribute selector performs greedy hill climbing with backtracking. It is possible to specify how many consecutive non-improving nodes must be encountered before the system backtracks.
LinearForwardSelection (LFS)	This attribute selector is an extension of BestFirst that considers a restricted number of the remaining attributes when expanding the current point in the search.
Backward Elimination (BE)	This attribute selector starts with the whole set of attributes and eliminates one attribute in each iteration until no single attribute elimination improves the evaluation of the subset.
Forward Selection (FS)	This attribute selector starts from an empty set and evaluates each attribute individually to find the best single attribute. It then tries each of the remaining attributes in conjunction with the best pair to find the best group of the three attributes.

model (e.g., for predictive data mining) is so "specific" that it reproduces various idiosyncrasies (random "noise" variation) of the particular data from which the parameters of the model were estimated; as a result, such models often may not yield accurate predictions for new observations (e.g., during deployment of a predictive data mining project). The effort prediction part of the automated framework consists of predictor construction and effort prediction.

In this study, we improved the implementation of the framework proposed in (Murillo-Morera et al. 2016a) (Fig. 1, **Learning Scheme Evaluator, Step-3 to Step-**7). The main difference between this framework and others consisted of the use of a genetic algorithm to select the parts of the learning scheme, instead of evaluating only a group of pre-established combinations. The main extensions of this paper with respect to (Murillo-Morera et al. 2016a) are: (a) the extension of the search space to 600 learning schemes, (b) the sensibility analysis of the genetic operators (generation, population, crossover and mutation) to find learning algorithms with better performance and (c) the comparison of our approach against an exhaustive and a random guessing approach (performance and runtime).

3.1 Learning schemes generator and evaluator

The learning scheme generator is characterized by the selection of the best learning scheme according to the maximum Correlation Coefficient (CC) value computed and the configuration selected by the genetic approach (Fig. 1, Learning Scheme generator). The evaluator component (Fig. 1, Learning Scheme Evaluator) is represented by the fitness function based on Song methodology (Song et al. 2011). The main steps of this component are detailed below (Fig. 1):

1. Historical data was randomized and divided into a training set and a test set. This was done using a $M \times N$-fold cross-validation (**Evaluation, Step-3**).
2. The selected data preprocessing technique was applied to both the training and the test set (**Evaluation, Step-4**), resulting in modified training and test data.
3. The chosen attribute selection technique was applied only to the training set (**Evaluation, Step-5**) and the best subset of attributes was selected.

Table 4 Learning algorithm

LA	Description
GaussianProcesses (GP)	This algorithm implements the Bayesian Gaussian process technique for non-linear regression. This method is equivalent to kernel ridge regression.
LeastMedSq (LMS)	Is a robust linear regression algorithm that minimizes the median (rather than mean) of the squares of divergences from the regression line. It repeatedly applies standard linear regression to subsamples of the data and outputs the solution that has the smallest median-square error.
LinearRegression (LR)	This algorithm performs standard least-squares multiple linear regression and can optionally perform attribute selection, either greedily using backward elimination or by building a full model from all attributes and dropping the terms one by one, in decreasing order of their standardized coefficients.
MultilayerPerceptron (MP)	Is a neural network that trains using back-propagation. This network can be built by hand, created by an algorithm or both. The network can also be monitored and modified during training time. The nodes in this network are all sigmoid (except for when the class is numeric in which case the output nodes become unthresholded linear units).
RBFNetwork (RBFN)	Is an algorithm that implements a Gaussian radial basis function network, deriving the centers and widths of hidden units using k-means and combining the outputs obtained from the hidden layer using logistic regression, if the class is nominal and, linear regression if it is numeric.
SMOreg (SMO)	This method implements the sequential minimal-optimization algorithm for learning a support vector regression model. The parameters can be learned using various algorithms. The algorithm is selected by setting the RegOptimizer.
AdditiveRegression (AR)	It is an algorithm that enhances the performance of a regression base classifier. Each iteration fits a model to the residuals left by the classifier on the previous iteration. Prediction is accomplished by adding the predictions of each classifier.
Bagging (BAG)	Is an algorithm that reduces variance. This method can work for both classification and regression, depending on the base learner. In the case of classification, predictions are generated by averaging probability estimates, no by voting.
ConjunctiveRule (CR)	The algorithm learns a single rule that predicts either a numeric or a nominal class value. Uncovered test instances are assigned the default class value (or distribution) of the uncovered training instances.
DecisionTable (DT)	This algorithm consists of a hierarchical table in which each entry in a higher level table gets broken down by the values of a pair of additional attributes to form another table. The structure is similar to dimensional stacking (Becker 1998).
M5Rules (M5R)	This algorithm obtains regression rules from model trees built using M5 Ridor (Ripple Down Rule learner). This means that method learns rules with exceptions by generating the default rule, using incremental reduced-error pruning to find exceptions with the smallest error rate, finding the best exceptions for each exception, and iterating. (More details (Quinlan and et al. 1992; Holmes et al. 1999; Wang and Witten 1997))
ZeroR (ZR)	This algorithm predicts the test data's majority class (if nominal) or average value (if numeric). It is the simplest classification method, which relies on the target and ignores all predictors. ZeroR algorithm simply predicts the majority category (class).
DecisionStump (DS)	It is an algorithm designed for use with the boosting method. It builds one-level binary decision trees for datasets with a categorical and numeric class, dealing with missing values by treating them as separate values and extending a third branch from the stump.
M5P (M5P)	This algorithm combines a conventional decision tree with the possibility of linear regression functions at the nodes. First, a decision-tree induction algorithm is used to build a tree, but instead of maximizing the information gain at each inner node, a splitting criterion is used that minimizes the intra-subset variation in the class values down each branch.
REPTree (RT)	This algorithm builds a decision or regression tree using information gain/variance reduction and prunes it using reduced-error pruning. Optimized for speed, it only sorts values for numeric attributes once. It deals with missing values by splitting instances into pieces.

4. The attributes selected were then extracted for both the training and the test set (**Evaluation, Step-6**).
5. The Learning algorithm was built using the training set and evaluated with the test set. (**Evaluation, Step-7**).

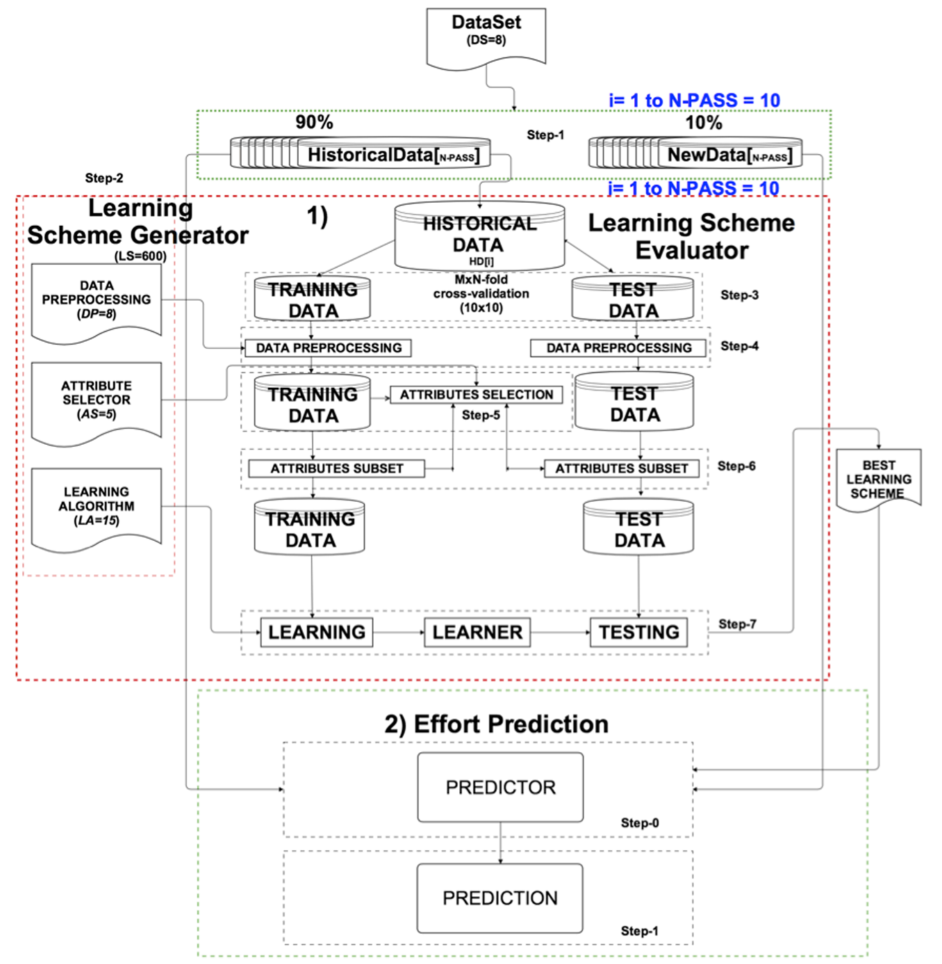

Fig. 1 Learning Schemes Generator and Evaluator: This figure represents the learning scheme generator and evaluator component. The generator is characterized by the selection of the best learning scheme according to the maximum Correlation Coefficient (CC) value computed and the configuration selected by the genetic approach. On the other side, the evaluator is represented by the fitness function based on Song methodology

3.2 Effort prediction

The main objective of this component is to build a predictor for new data, while the main objective of the generator of the learning scheme (first component) is to build a learner and select the best one with regard to its Correlation Coefficient (CC). A predictor is built using the best learning scheme identified in the previous stage. Once it is ready, this predictor can be used to compute the final performance score using new data (i.e. predicting on new data). This is simulated by running the predictor on the newData dataset that was reserved when doing the (90-10%) splits. We applied a N-PASS, where $N = 10$. The mean is computed after N iterations. This final CC represents our CC of prediction. This component is represented by (**Effort Prediction, Step-0, Step-1**).

3.3 Genetic approach

This section describes the genetic configuration used in this paper: Chromosome, Operators and Fitness Function. The genetic configuration and the proposed framework of this research is an adapted version of the framework presented in (Murillo-Morera et al. 2016a). A Genetic Algorithm (GA) is a search approach that mimics the biological process of natural selection in order to find a suitable solution in a multidimensional space. Genetic algorithms are, in general, substantially faster than exhaustive search procedures. In previous studies (Murillo-Morera et al. 2016c; 2016b), we have corroborated this experimentally in the area of software prediction.

Chromosome. The Chromosome of a genetic algorithm represents the set of possible combinations within the search space. It is commonly represented as a binary chain of 0s and 1s. In this paper, the Chromosome consists of three parts: Data preprocessing (DP), Attribute Selector (AS), and Learning Algorithms (LA); effectively constructing a triplet of $< DP, AS, LA >$. For DP, we considered two possibilities, represented by a binary chain of 1 bit ($2^1 = 2$). For AS, we considered five possible techniques, also represented by a binary chain of 3 bits ($2^3 = 8$). For LA, we considered fifteen different possibilities, which required 4 bits to represent ($2^4 = 16$). With this Chromosome representation, the goal of the GA is to find the Chromosome that maximizes a fitness function. We only worked with combinations within the proposed range. For example, the range of the attribute selection is between $1 - 5$. This means that combinations with the attribute selection set $6 - 8$ are not valid.

Fitness Function. The final value of the Correlation Coefficient (CC) is computed in the prediction phase. The N-PASS is a strategy to avoid variability of the final result of the Correlation Coefficient (CC) (Song et al. 2011), where the fitness score of a Chromosome is computed as the average of 10 runs. The whole dataset is split randomly in two data sets: histData and newData. The histData set represents 90% of the whole dataset used for training and testing. The newData set represents 10% of the whole dataset, representing a new project; we repeat this for each PASS in the evaluation phase. This algorithm performs a $M \times N$-fold cross-validation, where multiple rounds are performed with different partitions to reduce variability. Validation of results is averaged over rounds. Finally, in the prediction phase we generated the final value of the CC using the unseen new data and the best learning scheme (LS) computed in the evaluation phase. This solution is used in the prediction phase.

Operators. The operators of selection, reproduction, crossover and mutation used were configured using the default values provided by the WEKA genetic search (Witten and Frank 2005). The Learning Scheme Generator, shown in Fig. 1, is responsible for evaluating and selecting the different learning schemes. Selection is done through the elitism technique of the genetic algorithm.

Finally, the best learning scheme, in our case the Chromosome (with its data preprocessing, attribute selector and learning algorithm), was selected by the genetic algorithm.

3.4 Genetic setup

The genetic approach was implemented using JGAP-API (Meffert and Rotstan 2005). We generated the populations and generations randomly, using the standard configuration of WEKA's geneticSearch. However, our selection of population, generation, mutation and crossover levels is based on (Bala and Sharma 2015) (Table 5). We used tournament as operator of selection, with $tournamentk = 2$ and $tournamentp = 0.5$ and we set elitism to true. In the generation-evaluation phase, we applied a strategy for the selection of attributes called Wrapper (Witten and Frank 2005). It was used with the objective of selecting the attributes for each subset using an internal cross-validation. Wrappers

Table 5 Framework configuration for the sensibility analysis

Id	Population x Generation	Mutation	Crossover
1	10 x 10	0.01	0.6
2	20 x 20	0.01	0.6
3	40 x 40	0.01	0.6
4	20 x 20	0.01	0.6
5	20 x 20	0.033	0.6
6	20 x 20	0.1	0.6
7	20 x 20	0.01	0.6
8	20 x 20	0.01	0.7
9	20 x 20	0.01	0.9

generally provide better results than filters, but they are computationally more intensive (Song et al. 2011).

4 Methods

This section presents the design of the empirical study we carried out to validate the automated genetic framework. The controlled experiment follows the recommendations detailed in (Dejaeger et al. 2012). First, we present the experimental procedure. Second, we describe the datasets and learning schemes used. Finally, we detail the evaluation criteria and statistical tests used to assess the results. We provide an overview of the evaluation process in the Fig. 2.

4.1 Experimental procedure

The empirical validation assesses different learning schemes according to specific data sets. We analyzed the performance of effort prediction models based on different evaluation criteria detailed in Section 4.4. Then, we compared our genetic framework results with a baseline exhaustive framework (Quesada-Lopez et al. 2016) and a baseline random guessing framework based on (Langdon et al. 2016) to evaluate its performance. Finally, we applied a sensitivity analysis with the objective to find which genetic configurations reported best performance. We executed the following specific steps for the experimental procedure to collect the results (Fig. 2):

- **Step-1.** We selected $n = 8$ sub datasets from the ISBSG R12 database according the procedure and characteristics detailed in Section 4.2.
- **Step-2.** We executed the three frameworks: (a) genetic, (b) exhaustive and (c) random guessing using these 8 datasets and collected the performance measures detailed in Section 4.4.
- **Step-3.** The (a) genetic and (b) exhaustive frameworks used 600 learning schemes that included the combination of 8 Data Preprocessing, 5 Attribute Selectors and 15 Learning Algorithms detailed in Section 4.3.
- **Step-4.** We executed the (a) genetic framework with nine different configurations that included 3 levels of Generation, 3 levels of Population, 3 levels of Mutation and 3 levels of Crossover as shown in Table 5.
- **Step-5.** After the results were collected, each analysis was conducted according to Section 4.5.

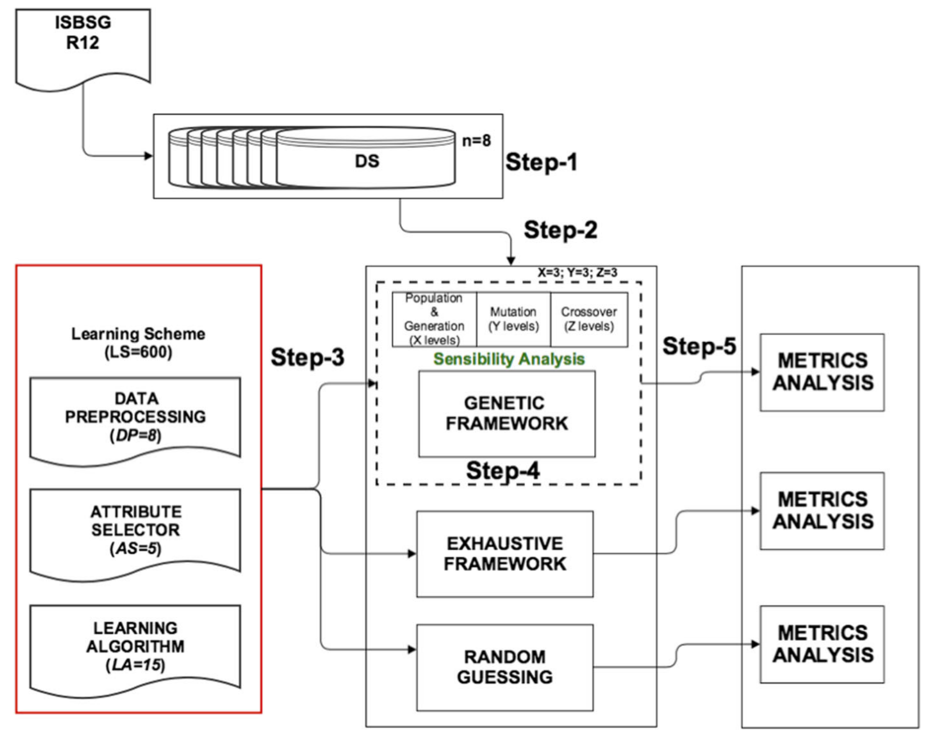

Fig. 2 Experimental Design Steps: This figure represents the experimental design steps. In this experiment, the set $N - PASS = 10$ and computed the Correlation Coefficient (CC) average is set after $N - PASS$ runs. For each $PASS$, 90% of the data as historical at random is selected. An $N = 10$ x $M = 10$-fold cross-validation is used to evaluate each learning scheme. The evaluation metrics were computed after N x M-fold cross-validation. The fitness function of each genetic individual was executed in 1000 holdout experiments, ($N - PASS = 10$) and $N = 10$ x $M = 10$-fold cross-validation. The mean of the 1000 CC measures was reported as the evaluation performance per genetic individual. The historicalData (90%) was preprocessed considering preprocessing techniques. Then, the predictor was used to predict with the newData (10%), which was preprocessed the same way as the historical data

In our experiment, we set $N - PASS = 10$ and computed the Correlation Coefficient (CC) average after $N - PASS$ runs. For each $PASS$, we selected 90% of the data as historical at random. An $N = 10$ x $M = 10$-fold cross-validation was used to evaluate each learning scheme. The evaluation metrics were computed after N x M-fold cross-validation. The fitness function of each genetic individual was executed in 1000 holdout experiments, ($N - PASS = 10$) and $N = 10$ x $M = 10$-fold cross-validation. The mean of the 1000 CC measures was reported as the evaluation performance per genetic individual. The historicalData (90%) was preprocessed considering preprocessing techniques. Then, the predictor was used to predict with the newData (10%), which was preprocessed the same way as the historical data (Section 3).

4.2 Dataset selection

This study evaluates the effectiveness of software effort prediction models based on function point measures using the ISBSG R12 Dataset (Hill 2010; ISBSG 2015). ISBSG has collected and refined its database over many years based on the metrics that have proven to be most useful in management of software development and processes. ISBSG R12 contains projects from 24 countries and several organizations. Due to the heterogeneousness of the data, ISBSG recommends extracting a suitable subset of projects. We analyzed the performance of effort prediction models based on base functional components and unadjusted function point size (Albrecht 1979; Jeng et al. 2011). The subset of data projects for our study was selected according to the criteria shown in Table 6 based on recommendations presented in (Dejaeger et al. 2012; Murillo-Morera et al. 2016a; Mendes et al. 2005; Mendes and Lokan 2008; Seo et al. 2013; Quesada-López and Jenkins 2014; 2015; 2016). Projects for which all functional components (UFP and BFC) of function points were missing were discarded.

For our study, we selected the variables related to FPA functional size components (BFC), effort of software development, and context attributes according to recommendation in (González-Ladrón-de-Guevara and Fernández-Diego 2014). The list of selected variables is: Input count (EI), Output count (EO), Interface count (EIF), File count (ILF), Enquiry count (EQ), Functional size in Unadjusted Function Points (UFP), Work Effort in man-hours, Development Type, Relative Size, Team Size Group, Development Platform, Architecture, Language Type, Program Language, and others. The description of context variables can be found in (Hill 2010; ISBSG 2015). The variables selected in this study are reported in Appendix A.

Two groups of datasets were used. The first dataset group is represented by 72 projects ($\geq$ 2008, Rating A). Twenty-nine of them are from 2008, twenty-five from 2009, thirteen from 2010, and five from 2011. The smallest project size is 24 UFPs, the average is 240 UFPs, and the largest project is 1,337 UFPs. The second dataset group is represented by 202 projects ($\geq$ 2005, Ratings A or B). Forty-three of them are from 2005, 34 from 2006, 46 from 2007, 35 from 2008, 26 from 2009, 13 from 2006, and 5 from 2011. The smallest project size is 6 UFP, the average is 247 with a median of 184 UFP, and the largest project is 1,337 UFP. The dataset is positively skewed for all variables indicating that the quantity of small and medium projects is higher than the number of large projects. Most of the projects were small (between 32-99 UFP) and medium size (between 102-993 UFP). Extra small projects (between 6-28 UFP) and large projects (between 1,018-1,337 UFP) show lower productivity than small and medium size projects. Data indicates that the use small teams in small and medium size projects is better in terms of productivity. Descriptive statistics for both datasets are presented in Appendix B. A completed description of the ISBSG database could be found in (Hill 2010; ISBSG 2015).

Before applying the framework, the two used datasets were split in four subsets of data (Table 7). In the first subset ([1]72-UFP and [5]202-UFP), unadjusted function points size (UFP) was selected as a predictor and effort of software development as the dependent variable. In the second subset selected ([2]72-BFC and [6]202-BFC), basic functional components size (BFC) were selected as a predictor and effort of software development as the dependent variable. In the third subset ([3]72-UFP-CTX and [7]202-UFP-CTX), unadjusted function points size (UFP) and nominal context attributes (CTX)

Table 6 Project selection criteria

Criteria	Values	Motivation
Count Approach	IFPUG 4+	Latest FPA standard and counting rules.
Data Quality	A	Only data with a high level of quality and integrity.
UFP Rating	A or B, A	Counting data with a high level of quality and integrity.
Year of project	$\geq$ 2005, $\geq$ 2008	New projects using new technologies.
Application group	BA	Business application domain.
Resource Level	1	Only development team effort included.

Table 7 DataSet description

DS	Name	S/M	Domain	Year	FP	Quality	Metric	N
1-4	ISBSG R12	Multi	MIS	2009-2011	IFPUG	A	FPA	72
5-8	ISBSG R12	Multi	MIS	2006-2011	IFPUG	A,B	FPA	202

were selected as predictors and effort of software development as the dependent variable. In the fourth subset ([4]72-BFC-CTX and [8]202-BFC-CTX), basic functional components size (BFC) and nominal context attributes (CTX) were selected as predictors and effort of software development as dependent variable, as shown in Table 8. Nominal variables were transformed by dummy coding, where each variable was coded as a 0 or 1 by applying a nominal to binary filter.

4.3 Learning schemes

The learning schemes are composed of three parts: Data Preprocessing (DP), Attribute Selector (AS) and Learning Algorithm (LA). In total, we tested in a search space of 600 different learning schemes (8 Data Preprocessing * 5 Attribute selectors * 15 Learning algorithms). The details of the different techniques used for each learning scheme are presented in the Section 2.3.

4.4 Evaluation criteria

The prediction accuracy of the models was tested following the criteria applied in (Shepperd and MacDonell 2012; Dejaeger et al. 2012; Langdon et al. 2016; Seo et al. 2013; Lavazza et al. 2013; Lokan and Mendes 2006). The differences between the actual effort (e_i) and the predicted effort ($\hat{e}_i$) should be as small as possible because large deviations between ei and $\hat{e}i$ will have a significant impact on the software development costs (Dejaeger et al. 2012). The criterion applied in our evaluation were Spearman's rank correlation (SC), mean of the magnitude of relative error (MRE), median of the magnitude of relative error (MdMRE), mean of the absolute residuals (MAR), standardized accuracy (SA), and number of predictions within % of the actual ones (Pred25) as shown in Appendix C.

MRE is the difference between the actual effort and the predicted effort. The MRE value of individual predictions can be averaged, resulting in the Mean MRE (MMRE). The MMRE is computed for each observation and is defined in Eq. 1. MMRE can be highly affected by outliers and is considered a biased and deprecated accuracy statistics (Shepperd and MacDonell 2012), but it has been extensively used by previous studies. To address this shortcoming, some authors used the MdMRE metric, which is the median of

Table 8 Attributes for dataSet

DS	Total	Size	Context	Variables
1	1	1	0	UFP
2	5	5	0	EI,EO,EQ,ILF,EIF
3	24	1	23	UFP
4	28	5	23	EI,EO,EQ,ILF,EIF
5	1	1	0	UFP
6	5	5	0	EI,EO,EQ,ILF,EIF
7	24	1	23	UFP
8	28	5	23	EI,EO,EQ,ILF,EIF

all MREs. This metric can be considered better for outliers, and it is therefore preferred over the MMRE.

The Pred(%) measures the percentage of the estimates whose error is less than % and it is usually set at 25. Pred is simply the percentage of estimates that are within m% of the actual value (the % of the estimates with MRE $\leq$ 0.25). The indicator reveals what proportion of estimates are within a tolerance of 25% (Dejaeger et al. 2012). This evaluation is performed because the presence of an association does not necessarily imply that an accurate predictive model can be built. It is defined in Eq. 2. Our study compares results across the same empirical setup, data preprocessing and attribute selectors. Typical Pred25 lies in the range of 10 to 60%, and MdMRE between 30% and 100% (Dejaeger et al. 2012). Spearman's rank correlation (SCC) measures the relationship between (e_i) and ($\hat{e}_i$). It is a nonparametric correlation coefficient and is defined in Eq. 3.

Mean of the absolute residuals (MAR) and the standardized accuracy (SA) are new measures recommended to compare the performance of prediction models (Shepperd and MacDonell 2012; Langdon et al. 2016). MAR is unbiased (towards over or underestimation) since it is not based on ratios, but it is hard to interpret and comparisons cannot be made across data sets since the residuals are not standardized. It is defined in Eq. 4.

SA is a standardized accuracy measure for prediction techniques (P_i) based on MAR. SA measures the accuracy as the MAR relative to random guessing P_0. SA is defined in Eq. 5, where MAR_{P_0} is the unbiased exact version of MAR_{P_0} (Shepperd and MacDonell 2012) recommended for small datasets. It is defined in Eq. 6 as recommended by (Langdon et al. 2016). This is a naive approach that provides a relevant baseline irrespective of the exact form of P_i. SA represents how much better P_i is than random guessing. A value close to zero means that the prediction model P_i is practically useless, performing little better than a mere random guess. Effort estimation models should minimize MAE and maximize SA (Shepperd and MacDonell 2012).

4.5 Statistical tests

A comparative procedure is followed to statistically test the performance of the models. The nonparametric Wilcoxon Rank test was applied (Rédei 2008). This test substitutes t-test for paired samples. The desirable minimal number of paired samples is 8-10 and it is expected that the population would have a median, be continuous and symmetrical. The differences between the variates are tabulated and ranked; the largest one receives the highest rank. In the case of ties, each should be assigned to a shared rank. The smallest group of signed-rank values is then summed as the T value. This T value is compared with figures in a statistical table. If the value obtained is smaller than that in the body of the table under probability and on the line corresponding to the number of pairs tested, then the null hypothesis is rejected and the conclusion is justified that the performance of the two samples are different.

4.6 Threads to validity

There are threats to validity that need to be considered in this study. Construct validity is the degree to which the variables used in the study accurately measure the concepts they are supposed to measure. The datasets analyzed have been used in several studies in effort estimation, but the limited size and characteristics of the sub-dataset used in our analyses should be considered. The datasets were preprocessed according some approaches in the literature but organizations should use our approach with subsets of data close to their domain and type of software. Data were filtered to make sure only desirable and high-level quality information were used in the analysis. We used evaluation measures and statistical tests suggested in previous studies and robust statistical techniques were used to analyze the prediction model results.

Internal validity refers to establishing that a certain observable event was responsible for a change in behavior. We discuss internal validity in terms of threats to our experiments designs. In our study, we followed a strong framework with statistical tests to evaluate the performance. In the selection of the learning scheme combination, only the correlation coefficient was used by the elitism technique. Our main issue has been the computational time of the framework with the datasets, mainly the length of execution of the fitness function. The parameters selection for all used techniques were the default configuration because our aim was to validate the genetic framework. Several other range of parameters could have been tuned to obtain the best estimates. However, the computational time makes this difficult in practice.

External validity regards to generalizing the study results to other situations. The results of this study only considered the ISBSG R12 database. The limited datasets caused some difficulties when attempting to generalize our conclusions. It is necessary to perform further experiments with other kinds of databases presenting different structures and characteristics. The software project datasets used in our experiments were based on the business application domain.

5 Results and discussion

This section reports the results of the genetic framework based on techniques discussed in Section 2.3 and presents the results for each RQ.

5.1 RQ1. Which genetic framework configuration (generation and population, mutation levels, crossover levels) did report the best performance when compared to the baseline exhaustive framework?

5.1.1 Generation and population

Table 9 presents a summary of the best performances of the genetic framework (Gen) and the exhaustive framework (Exh). We present the performance results for each dataset (DS), genetic configuration (C), and leaning scheme (LS). We present the results of three levels of generations and populations ([1] 10x10, [2] 20x20, [3] 40x40). For the configurations of population and generation, we conducted three analyses: first, we compared the performance between the genetic framework (population and generation of 10x10) and the exhaustive framework. Second, we compared the performance between the genetic framework (population and generation of 20x20) and the exhaustive framework. Finally, we compared the performance between the genetic framework (population and generation of 40x40) and the exhaustive framework. Our hypotheses states that the performance is similar between the exhaustive framework and each genetic framework configuration.

For the first hypothesis, Wilcoxon Rank test indicated that exhaustive framework ranks were statistically significant higher than genetic framework (10x10). Our result was $p_{value} = 0.01563 < \alpha = 0.05$. This means that we found a statistically significant difference between the exhaustive framework analyzed and the configuration (10x10).

Table 9 Performance of the genetic and exhaustive frameworks

DS	LS	Exh	C	LS	Gen	C	LS	Gen	C	LS	Gen
1	7X2X6	0.79	1	6X3X8	0.72	4	1X1X11	0.79	7	1X1X11	0.79
2	1X1X14	0.77	1	1X2X8	0.71	4	7X3X14	0.77	7	7X3X14	0.77
3	2X1X1	0.85	1	2X4X6	0.84	4	2X5X1	0.85	7	2X5X1	0.85
4	2X2X1	0.83	1	6X5X3	0.80	4	6X5X3	0.80	7	6X5X3	0.80
5	7X1X2	0.70	1	7X1X3	0.69	4	6X5X1	0.69	7	6X5X1	0.69
6	6X5X1	0.70	1	6X3X1	0.71	4	6X4X1	0.71	7	6X4X1	0.71
7	6X4X6	0.78	1	7X5X14	0.75	4	6X4X6	0.76	7	6X4X6	0.76
8	6X4X6	0.75	1	2X3X1	0.74	4	1X5X3	0.71	7	1X5X3	0.71
1	7X2X6	0.79	2	1X1X11	0.79	5	1X5X11	0.79	8	6X2X4	0.77
2	1X1X14	0.77	2	7X3X14	0.77	5	7X1X14	0.77	8	7X3X11	0.74
3	2X1X1	0.85	2	2X5X1	0.85	5	7X4X6	0.84	8	1X4X6	0.84
4	2X2X1	0.83	2	6X5X3	0.80	5	2X2X1	0.83	8	6X1X3	0.81
5	7X1X2	0.70	2	6X5X1	0.69	5	1X3X4	0.69	8	7X2X3	0.69
6	6X5X1	0.70	2	6X4X1	0.71	5	6X2X1	0.71	8	6X4X3	0.68
7	6X4X6	0.78	2	6X4X6	0.76	5	6X2X3	0.75	8	1X3X14	0.74
8	6X4X6	0.75	2	1X5X3	0.71	5	2X1X1	0.75	8	2X2X6	0.72
1	7X2X6	0.79	3	7X5X6	0.79	6	6X2X4	0.78	9	1X1X2	0.78
2	1X1X14	0.77	3	6X2X14	0.78	6	7X1X14	0.77	9	1X5X14	0.77
3	2X1X1	0.85	3	2X5X1	0.85	6	2X5X6	0.84	9	2X4X1	0.85
4	2X2X1	0.83	3	7X1X6	0.82	6	5X2X1	0.79	9	7X5X6	0.81
5	7X1X2	0.70	3	7X1X3	0.70	6	1X1X3	0.70	9	1X1X6	0.69
6	6X5X1	0.70	3	6X1X1	0.71	6	6X5X1	0.71	9	1X2X6	0.68
7	6X4X6	0.78	3	6X2X6	0.78	6	6X2X3	0.76	9	1X3X14	0.76
8	6X4X6	0.75	3	2X5X1	0.74	6	2X4X1	0.75	9	2X5X1	0.75

For the second hypothesis, Wilcoxon Rank test indicated that exhaustive and genetic framework (20x20) ranks were not statistically significant different. Our result was $p_{value} = 0.07813 > \alpha = 0.05$. This means that we did not find a statistically significant difference between the exhaustive framework analyzed and the configuration (20x20).

Finally, for the third hypothesis, Wilcoxon Rank test indicated that exhaustive and genetic framework (40x40) ranks were not statistically significant different. Our result was $p_{value} = 0.7422 > \alpha = 0.05$. This means that we did not find a statistically significant difference between the exhaustive framework analyzed and the configuration (40x40).

According to performance, the best genetic configuration was (40x40) for all datasets and the worst was (10x10) compared with the baseline exhaustive framework.

5.1.2 Mutation level(s)

For the mutation operator, we conducted three analyses: first, we compared the performance of the genetic framework with the exhaustive framework (mutation 0.01). Second, we compared the performance of the genetic framework with the exhaustive framework (mutation 0.033). Finally, we compared the performance of the genetic framework with the exhaustive framework (mutation 0.1). Our hypothesis is that the performance is similar between the exhaustive framework and each genetic framework configuration.

For the first hypothesis, Wilcoxon Rank test indicated that exhaustive and genetic framework (0.01) ranks were not statistically significant different. Our result was $p_{value} = 0.07813 > \alpha = 0.05$. This means that we did not find a statistically significant difference between the exhaustive framework analyzed and the mutation (0.01).

For the second hypothesis, Wilcoxon Rank test indicated that exhaustive and genetic framework (0.033) ranks were not statistically significant different. Our result was $p_{value} = 0.1484 > \alpha = 0.05$. This means that we did not find a statistically significant difference between the exhaustive framework analyzed and the mutation (0.033).

Finally, for the third hypothesis, Wilcoxon Rank test indicated that exhaustive and genetic framework (0.1) ranks were not statistically significant different. Our result was $p_{value} = 0.07813 > \alpha = 0.05$. This means that we did not find a statistically significant difference between the exhaustive framework analyzed and the configuration (0.1).

Table 9 shows the results of different levels of mutation in configuration (C) values [4] 0.01, [5] 0.033 and [6] 0.1. The mutation with the best performance was 0.033 compared with the baseline exhaustive framework.

5.1.3 Crossover level(s)

For the crossover operator, we conducted three analyses: first, we compared the performance of the genetic framework with the exhaustive framework (crossover 0.6). Second, we compared the performance of the genetic framework with the exhaustive framework (crossover 0.7). Finally, we compared the performance of the genetic framework with the exhaustive framework (crossover 0.9). Our hypothesis says that the performance is similar between the exhaustive framework and each genetic framework configuration.

For the first hypothesis, Wilcoxon Rank test indicated that exhaustive and genetic framework (0.6) ranks were not statistically significant different. Our result was $p_{value} = 0.07813 > \alpha = 0.05$. This means that we did not find a statistically significant difference between the exhaustive framework analyzed and the crossover (0.6).

For the second hypothesis, Wilcoxon Rank test indicated that exhaustive and genetic framework (0.7) ranks were not statistically significant different. Our result was $p_{value} = 0.07813 > \alpha = 0.05$. This means that we did not find a statistically significant difference between the exhaustive framework analyzed and the mutation (0.7).

Finally, for the third hypothesis, Wilcoxon Rank test indicated that exhaustive framework ranks were statistically significant different from genetic framework (0.9). Our result was $p_{value} = 0.03906 < \alpha = 0.05$. This means that we found a statistically significant difference between the exhaustive framework analyzed and the crossover (0.9).

Table 9 shows the results of different levels of crossover in configuration (C) values [7] 0.6, [8] 0.7 and [9] 0.9. The crossovers with the best performance were 0.6 and 0.7 compared with the baseline exhaustive framework).

5.1.4 Runtime comparison between the exhaustive framework and the genetic framework

We compared the genetic and exhaustive framework selecting the runtime of the best configuration according to performance on the same computer configuration (hardware conditions). We used the same search space (600 combinations) and all the conditions were equal for both approaches. Table 10 presents the runtimes reported for the genetic and exhaustive framework according to each dataset evaluated. There is a clear difference between the runtimes of the genetic framework when compared to the exhaustive framework. In all cases, the genetic framework reported better runtimes. As expected, the datasets with context variables (3, 4, 7, 8) were computationally more expensive to process, the datasets without context variables (1, 2, 5, 6) reported less runtime difference.

Our null hypotheses stated that runtime is similar between an exhaustive framework and our framework. Our result was $p_{value} = 0.007813 < \alpha = 0.05$. This means

Table 10 Runtime in milliseconds between genetic and exhaustive framework

DS	Genetic	Exhaustive	Difference
1	10,758,772.0	26,896,930.0	16138158.0
2	5,654,127.0	14,385,317.5	8,731,190.5
3	6,931,549.0	17,328,872.5	10,397,323.5
4	3,252,517.0	7,631,292.5	4,378,775.5
5	17,472,918.0	87,364,790.0	69,891,872.0
6	12,307,627.0	61,538,385.0	49,230,758.0
7	34,767,810.0	173,839,200.0	139,071,390.0
8	3,237,250.0	16,186,275.0	12,949,025.0

that we found a statistically significant difference between runtimes, where our genetic framework reports a better runtime than the exhaustive framework.

5.2 RQ2. Is the performance of the genetic framework similar between evaluation and prediction phases?

Table 11 presents a summary of the best performance of our genetic framework for the search space. We present the performance results for each dataset (DS), genetic configuration (C) and leaning scheme (LS). Our hypothesis is that the performance is similar between the evaluation (Eval) and prediction (Pred) phases for both groups of datasets studied (Section 4.2). The Wilcoxon Rank test indicated that evaluation and prediction phases ranks were not statistically significant different. Our result was $p_{value} = 0.1648 > \alpha = 0.05$. This means that we did not find a statistically significant difference between the evaluation and prediction phases for both groups of datasets. The results reported by the genetic framework are very similar and reliable compared to exhaustive framework.

5.3 RQ3. Which are the learning schemes (data preprocessors, attribute selectors, learning algorithms) more frequently selected by the genetic framework?

Figure 3 shows the best learning schemes selected by the genetic framework. Table 12 presents the frequency (F) of the best learning schemes (LS) by dataset (DS). The list of the learning schemes tecnhiques is found in Table 1.

Best learning schemes by dataset were 7x5x6: [BC(1) x FS x SMO], 1x1x11: [None x GS x M5R] and 6x2x4: [BC(0.5) x BF x MP] for DS=1: [72-UFP]; 6X2X14: [BC(0.5) x BF x M5P], 7X3X14: [BC(1) x LFS x M5P], and 7X1X14: [BC(1) x GS x M5P] for DS=2: [72-BFC]; 2X5X1: [Log x FS x GP] and 2X5X6: [Log x FS x SMO] for DS=3: [72-UFP-CTX]; 7X1X6: [BC(1) x GS x SMO], 6X5X3: [BC(0.5) x FS x LR] and 6X1X3: [BC(0.5) x GS x LR] for DS=4: [72-BFC-CTX]; 7X1X3: [BC(1) x GS x LR], 6X5X1: [BC(0.5) x FS x GP] and 1X1X6: [None x GS x SMO] for DS=5: [202-UFP]; 6X1X1: [BC(0.5) x GS x GP], 6X4X1: [BC(0.5) x BE x GP] and 1X2X6: [None x BF x SMO] for DS=6: [202-BFC]; 6X2X6: [BC(0.5) x BF x SMO], 6X4X6: [BC(0.5) x BE x SMO] and 1X3X14: [None x LFS x M5P] for DS=7: [202-UFP-CTX]; and 2X5X1: [Log x FS x GP], 1X5X3: [None x FS x LR] and 2X4X1: [Log x BE x GP] for DS=8: [202-BFC-CTX]. According to the results, the predominant learning scheme for all datasets is the configuration 6x5x6: [BC(0.5) x FS x SMO].

The frequency of the best learning schemes, according to their performance, was computed considering the complete set of genetic configurations. For the group 72 (DS=1-4), the main data preprocessors were BC(1) with 28%, BC(0.5) with 25% and None and Log with 22% each. The main attribute selectors were FS with 36%, GS with 22% and BF with

Table 11 Evaluation and prediction correlation values by framework configuration

DS	C	LS	Eval	Pred	DS	C	LS	Eval	Pred
1	1	6X3X8	0.72	0.70	5	1	7X1X3	0.69	0.67
1	2	1X1X11	0.79	0.80	5	2	6X5X1	0.69	0.69
1	3	7X5X6	0.79	0.79	5	3	7X1X3	0.70	0.70
1	4	1X1X11	0.79	0.80	5	4	6X5X1	0.69	0.69
1	5	1X5X11	0.79	0.80	5	5	1X3X4	0.69	0.71
1	6	6X2X4	0.78	0.79	5	6	1X1X3	0.70	0.71
1	7	1X1X11	0.79	0.80	5	7	6X5X1	0.69	0.69
1	8	6X2X4	0.77	0.78	5	8	7X2X3	0.69	0.68
1	9	1X1X2	0.78	0.78	5	9	1X1X6	0.69	0.69
2	1	1X2X8	0.71	0.72	6	1	6X3X1	0.71	0.72
2	2	7X3X14	0.77	0.78	6	2	6X4X1	0.71	0.72
2	3	6X2X14	0.78	0.78	6	3	6X1X1	0.71	0.71
2	4	7X3X14	0.77	0.78	6	4	6X4X1	0.71	0.72
2	5	7X1X14	0.77	0.79	6	5	6X2X1	0.71	0.70
2	6	7X1X14	0.77	0.79	6	6	6X5X1	0.71	0.70
2	7	7X3X14	0.77	0.78	6	7	6X4X1	0.71	0.72
2	8	7X3X11	0.74	0.73	6	8	6X4X3	0.68	0.67
2	9	1X5X14	0.77	0.76	6	9	1X2X6	0.68	0.67
3	1	2X4X6	0.84	0.83	7	1	7X5X14	0.75	0.77
3	2	2X5X1	0.85	0.85	7	2	6X4X6	0.76	0.76
3	3	2X5X1	0.85	0.85	7	3	6X2X6	0.78	0.78
3	4	2X5X1	0.85	0.85	7	4	6X4X6	0.76	0.76
3	5	7X4X6	0.84	0.85	7	5	6X2X3	0.75	0.75
3	6	2X5X6	0.84	0.83	7	6	6X2X3	0.76	0.76
3	7	2X5X1	0.85	0.85	7	7	6X4X6	0.76	0.76
3	8	1X4X6	0.84	0.85	7	8	1X3X14	0.74	0.74
3	9	2X4X1	0.85	0.85	7	9	1X3X14	0.76	0.77
4	1	6X5X3	0.80	0.79	8	1	2X3X1	0.74	0.73
4	2	6X5X3	0.80	0.81	8	2	1X5X3	0.71	0.70
4	3	7X1X6	0.82	0.80	8	3	2X5X1	0.74	0.73
4	4	6X5X3	0.80	0.81	8	4	1X5X3	0.71	0.70
4	5	2X2X1	0.83	0.83	8	5	2X1X1	0.75	0.76
4	6	5X2X1	0.79	0.79	8	6	2X4X1	0.75	0.76
4	7	6X5X3	0.80	0.79	8	7	1X5X3	0.71	0.70
4	8	6X1X3	0.81	0.81	8	8	2X2X6	0.72	0.72
4	9	7X5X6	0.81	0.81	8	9	2X5X1	0.75	0.76

16%. Finally, the most selected learning algorithms were GP, SMO and M5R with 19% each and LR with 14%. Otherwise, for the group 202 (DS=5-8), the main data preprocessors were BC($\lambda = 0.5$) with 47%, None with 25% and Log with 17%. The main attribute selectors were FS with 28%, BE with 22% and BF with 20%. Finally, the most selected learning algorithms were GP with 42%, LR with 25% and SMO with 19%.

Our results confirm previous results stated in (Dejaeger et al. 2012), the use of Logarithmic and Cox-Box transformations could increase the performance of the effort prediction models. Another factor impacting the performance of software effort prediction models is input selection, for example, Backward Forward. Further, (MacDonell and Shepperd 2003; Minku and Yao 2013) stablish that ensembles of learning machines techniques optimize effort predictions in software project estimation. Examples of this kind of techniques are the SMO, M5P and M5R.

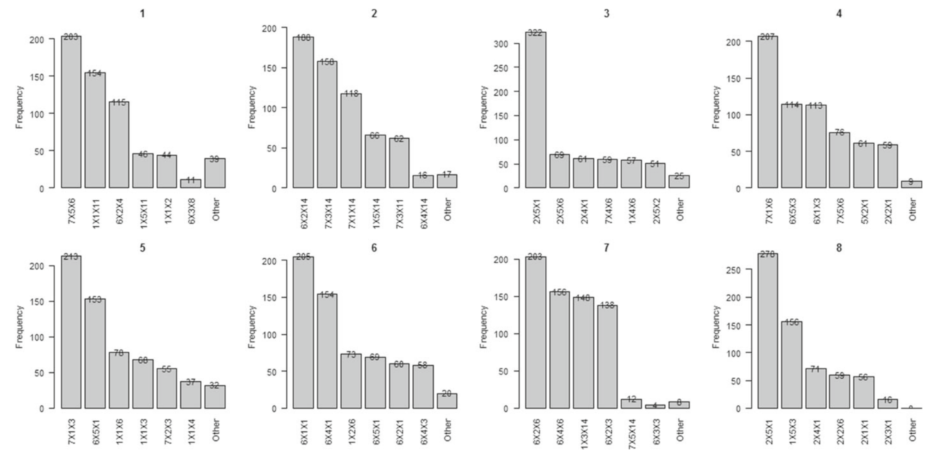

Fig. 3 Learning Schemes Frequency: This figure represents the frequency of the best learning schemes, according to their performance. The results were computed considering the complete set of genetic configurations

5.4 RQ4. Which learning schemes did report the best performance according evaluation criteria metrics?

5.4.1 Learning schemes with the best performance

Tables 15 and 16 in Appendix D present a summary of the best performance of the genetic framework ordered by each dataset (DS), genetic configuration (C), and leaning scheme (LS) that includes the data preprocessing, attribute selector, and learning algorithm (DPx-ASxLA). We present the results for all evaluation metrics detailed in Section 4.4 and the models were ranked according to their performance on each accuracy metric.

In this validation, we investigated if all of the considered learning schemes are more accurate than random guessing. The analysis of the standardized accuracy (SA) measure reveals that all best learning schemes selected by the genetic framework outperform the baseline random guessing (models outperformed random guessing by 32-60% for MSA and 45-80% for MdSA of all evaluations). In our study for the group 72 (DS=1-4)

Table 12 Best learning schemes selected

LS(DS=1)	F	%	LS(DS=2)	F	%	LS(DS=3)	F	%	LS(DS=4)	F	%
7X5X6	203	33%	6X2X14	188	30%	2X5X1	322	50%	7X1X6	207	32%
1X1X11	154	25%	7X3X14	158	25%	2X5X6	69	11%	6X5X3	114	18%
6X2X4	115	19%	7X1X14	118	19%	2X4X1	61	9%	6X1X3	113	18%
1X5X11	46	8%	1X5X14	66	11%	7X4X6	59	9%	7X5X6	76	12%
1X1X2	44	7%	7X3X11	62	10%	1X4X6	57	9%	5X2X1	61	10%
6X3X8	11	2%	6X4X14	16	3%	2X5X2	51	8%	2X2X1	59	9%
Other	39	6%	Other	17	3%	Other	25	4%	Other	9	1%
LS(DS=5)	F	%	LS(DS=6)	F	%	LS(DS=7)	F	%	LS(DS=8)	F	%
7X1X3	213	33%	6X1X1	205	32%	6X2X6	203	30%	2X5X1	278	44%
6X5X1	153	24%	6X4X1	154	24%	6X4X6	156	23%	1X5X3	156	25%
1X1X6	78	12%	1X2X6	73	11%	1X3X14	148	22%	2X4X1	71	11%
1X1X3	68	11%	6X5X1	69	11%	6X2X3	138	21%	2X2X6	59	9%
7X2X3	55	9%	6X2X1	60	9%	7X5X14	12	2%	2X1X1	56	9%
1X1X4	37	6%	6X4X3	58	9%	6X3X3	4	1%	2X3X1	16	3%
Other	32	5%	Other	20	3%	Other	8	1%	Other	0	0%

$MAR_{P_0} = 6406.28$, and for the group 202 (DS=5-8) $MAR_{P_0} = 5284.91$. With the analysis of SA we can conclude that the estimations obtained with the genetic framework are better than those achieved by using random estimates.

Our study compares all performance metrics results across the same empirical setup, data preprocessing and attribute selectors. Our results confirm previous results in literature such as (Dejaeger et al. 2012; Murillo-Morera et al. 2016a; Quesada-Løpez et al. 2016; Dolado et al. 2016; Quesada-López and Jenkins 2016; Lavazza et al. 2013; Minku and Yao 2013). In regards to the prediction accuracy, our approach allows to compare results across different models applying the same empirical setup and data preprocessing. In our study, the models are short of the typical industry target, but similar to reported in previous studies.

5.4.2 Attributes selected by the best learning schemes

In our framework, a set of variables were selected by the attribute selector (AS) component in each model generated by the genetic approach. The frequencies were calculated as the average of the $N - PASS$ for each learning scheme evaluated.

From an effort prediction perspective, the selection of attributes is one of the most important aspect for constructing the model. Therefore, we analyzed the final attribute selection in the construction of the best models. Then, we selected the variables and their frequencies for the best learning scheme per dataset. Finally, we selected the top ten variables.

Datasets DS=1: [72-UFP] and DS=5: [202-UFP] selected functional size with a frequency=100% (only one predictor). For dataset DS=2: [72-BFC] and DS=6: [202-BFC], the variables more selected were all basic functional components (BFC): EI, EO, EQ, EIF, ILF with a frequency=100%. A model based on BFC without context variables is constructed based on all functional components.

For dataset DS=3: [72-UFP-CTX], the variables more selected were: Max Team with a frequency of 75%, functional size with a frequency of 74%, CMMI Level and ISO with a frequency of 45%, Architecture with a frequency of 43%, Client Server with a frequency of 41% and Web Develop with a frequency of 40%.

For dataset DS=4: [72-BFC-CTX], the variables more selected were: Relative Band Size with a frequency of 87%, Develop Type with a frequency of 79%, Language Type with a frequency of 78%. For dataset DS=7: [202-UFP-CTX], the variables more selected were: Develop Type with a frequency of 99% and Language Type with a frequency of 98%. Finally, for dataset DS=8: [202-BFC-CTX], the variables more selected were: EI, EO, EQ, ILF, EIF, and Relative Band Size with a frequency of 100%. Our results confirmed the influence of attributes related with functional size and the context of the development. Further analysis should be conducted on the selection of attributes by the AS component.

6 Conclusions

In this paper, a genetic framework was used to select the best learning scheme for effort prediction per dataset using the elitism technique. The performance of the learning schemes was measured according to recommended metrics in previous studies: Spearman's rank correlation, mean of the magnitude of relative error, (MMRE), median of the magnitude of relative error (MdMRE), mean of the absolute residuals (MMAR), standardized accuracy (SA), and number of predictions within % of the actual ones (Pred25). The models were ranked according to their performance on each accuracy metric.

The final performance per dataset was determined by context configuration (random selection, missing values, imbalance dataset, outliers, and other machine learning strategies). Each dataset had a different context configuration and hence a different learning scheme. Our framework selected data preprocessing, attribute selection technique, and learning algorithms automatically according to characteristics of the specific data set. The main goal was to increase prediction performance optimizing processing time.

We investigated the results of the best learning schemes performance, compared our genetic framework results with an exhaustive framework (baseline), and conducted a sensitivity analysis with the objective to find which genetic configurations reported best performance.

The results of this study confirm the reported results of previous studies in the field. Simple, understandable regression techniques with log transformation of attributes perform as well as nonlinear techniques. Moreover, the SMOreg (a support vector machine for regression) and the M5P or M5R (a reconstruction of Quinlan's M5 algorithm for inducing trees of regression models that combines a conventional decision tree with the possibility of linear regression functions at the nodes) present an opportunity to improve some prediction models.

We agree with (Dejaeger et al. 2012) on the fact that although data mining and artificial intelligence techniques can make a contribution to the set of software effort estimation models, these still could not replace expert judgment. Both approaches should be seen as complementary to each other. Prediction models can be adopted to check expert estimations. The selection of a proper estimation technique can have a significant impact on performance, and it should be selected according the specific set of data used for prediction.

In the validation, we investigated if all the selected learning schemes are more accurate than random guessing. The analysis of the standardized accuracy (SA) measure revealed that all best learning schemes selected by the genetic framework outperform the baseline random guessing (models outperformed random guessing by 32-60% for MSA and 45-80% for MdSA of all evaluations). With the analysis of SA we can conclude that the estimations obtained with the genetic framework are better than those achieved by using random estimates.

Furthermore, we concluded that the exhaustive and the genetic framework (20x20 and 40x40) ranks were not statistically significant different. The genetic framework performs as good as an exhaustive framework. However, we did find a statistically significant difference between the exhaustive framework analyzed and the configuration (10x10).

Further, we did not find a statistically significant difference between the evaluation and prediction phases for both groups of datasets (72 and 202). This means that the results reported by the genetic framework were very similar and reliable. We did not find a statistically significant difference between the exhaustive framework analyzed and the mutations (0.01, 0.033 and 0.1). The mutation with the best performance was 0.033. We did not find a statistically significant difference between the exhaustive framework analyzed and the crossovers (0.6 and 0.7) except (0.9). The crossovers with the best performance were 0.6 and 0.7. This mean that our genetic approach is stable between different configurations and we can make the model's predictions use less computational resources.

Our results showed that the best models depend on the data set being analyzed. Our approach automated this selection based on the specific data set. From a practical point of view, an approach that generates specific models according to changing data of projects is an advantage. In that sense, our framework is adaptive to the specific data with which the model is built. The main advantage of our genetic framework is to obtain learning schemes with performance as good as those that could be obtained with a comprehensive framework; but with a shorter processing time. Contrary to an exhaustive approach, our genetic approach is heuristic and only runs a subset of the entire search space. Finally, this tool could collaborate in predicting the effort in software projects saving time and costs in future developments.

For future work, additional learning machine algorithms and data sets should be investigated. First, we would like to integrate new recommended techniques to the framework. After that, parameter tuning should be implemented automatically in the framework for each learning algorithm in order to improve the performance. The framework could also be enhanced by automating additional statistical analysis as a part of the evaluation process. Finally, novel unbiased error metric, such as standardized accuracy (SA), should be used for comparison of models.

Appendix A: dataset variables

The variables selected in this study are reported in Table 13.

Table 13 Dataset variables

Variable	Type	Description
Unadjusted Function Points	Numeric	The unadjusted function point count (before any adjustment).
Input count	Numeric	FPA External Input.
Output count	Numeric	FPA External Output.
Enquiry count	Numeric	FPA External Enquiry.
File count	Numeric	FPA Internal Logical Files.
Interface count	Numeric	FPA External Interface.
Relative Size	Nominal	XXS, XS, S, M1, M2, L, XL, XXL, XXXL.
Year	Nominal	Year of Project, derived from implementation date.
Development Type	Nominal	New development, enhancement or redevelopment.
Language Type	Nominal	3GL, 4GL, Application Generator, etc.
Program Language	Nominal	Primary technology programming language used.
Development Platform	Nominal	PC, Mid Range, Main Frame or Multi platform.
Architecture	Nominal	Stand alone, Multi-tier, Client server, or Multi-tier.
Client Server?	Nominal	Yes, No or Don't Know.
Web Development	Nominal	Yes, No or Don't Know.
Development method	Nominal	Development method.
Manual Count	Nominal	Yes, No.
FP Standard	Nominal	Function Size Metric Used.
Team Size Group	Numeric	Development Team size.
Max Team Size	Nominal	The maximum number of people that worked.
Average Team Size	Nominal	The average number of people that worked on the project.
Work Effort in man-hours	Numeric	The development team full life-cycle work effort in hours.

Table 14 Datasets descriptive statistics

DS	N	Variable	Mean	Min	5Q	10Q	25Q	Median	75Q	90Q	95Q	Max	Range	Std.D.	Skew	Kurt
1-4	72	UFP	240.42	24	54	70.9	93.25	183.5	323.5	483.4	600.15	1337	1313	202.3	2.66	11.27
	72	EI	88.78	0	1.95	6.6	26.25	71.5	108.75	179.6	320.55	551	551	95.26	2.64	8.94
	72	EO	46.4	0	0	0	7	26	62	130.9	172	287	287	59.7	2.12	5.04
	72	EQ	58.72	0	0	6	18.25	43.5	77.75	125.8	195.4	275	275	57	1.76	3.57
	72	ILF	39.71	0	0	7	7	26	51.25	95.8	137.75	252	252	48.18	2.5	7.58
	72	EIF	6.81	0	0	0	0	0	10	29.1	36.75	54	54	12.62	2.16	4.3
	72	Effort	6134.29	167	837.8	1150.7	2006.75	3347.5	7386.25	12850.4	19641	71729	71562	9135.85	5.52	37.96
5-8	202	UFP	246.9	6	38.45	53.3	90	183.5	312.25	552.1	731.8	1337	1331	224.02	2	4.67
	202	EI	96.99	0	4.3	8.3	26.75	66	131.25	235.6	363.05	551	551	100.02	1.9	3.91
	202	EO	37.56	0	0	0	5	21	49	94	137.25	287	287	50.46	2.64	8.2
	202	EQ	65.08	0	0	4	14	42.5	90.25	148.9	220.1	419	419	70.87	1.99	5.02
	202	ILF	37.07	0	0	0	7	21	51.25	84	123.55	403	403	52.75	3.65	18.31
	202	EIF	10.21	0	0	0	0	0	10	30	53.85	261	261	26.84	5.43	40.83
	202	Effort	4726.56	167	600.7	773.8	1288.75	2752	4999.75	9478.1	13299.75	71729	71562	7921.06	5.74	40.35

Table 15 Best learning schemes models performance

DS	C	LS	S	MMRE	MdMRE	MMAR	MdMAR	MSA	MdSA	Pred25
1	1	2X3X8	0.70	1.11	0.89	3,379	1,665	47.26	59.70	0.28
1	1	6X3X8	0.71	1.09	0.86	3,348	1,628	47.73	60.24	0.28
1	2	1X1X11	0.75	0.55	0.63	2,809	1,063	56.15	67.83	0.32
1	2	1X4X11	0.74	0.55	0.63	2,835	1,072	55.74	67.21	0.32
1	3	1X5X6	0.74	0.57	0.58	2,810	1,027	56.13	69.47	0.38
1	3	6X1X3	0.74	1.65	1.24	4,217	2,325	34.17	45.10	0.17
1	3	6X1X6	0.73	0.66	0.59	2,946	1,079	54.01	68.70	0.39
1	3	7X4X6	0.73	0.57	0.58	2,807	1,036	56.18	69.99	0.38
1	3	7X5X6	0.74	0.57	0.57	2,797	994	56.34	70.44	0.38
1	4	1X1X11	0.75	0.55	0.63	2,809	1,063	56.15	67.83	0.32
1	4	1X4X11	0.74	0.55	0.63	2,835	1,072	55.74	67.21	0.32
1	5	1X1X11	0.73	0.57	0.64	2,880	1,078	55.05	66.68	0.31
1	5	1X4X11	0.74	0.56	0.63	2,845	1,088	55.59	67.11	0.31
1	5	1X4X3	0.74	1.07	1.02	3,735	1,860	41.69	53.87	0.19
1	5	1X5X11	0.75	0.55	0.63	2,810	1,056	56.14	67.92	0.32
1	5	7X4X3	0.74	1.08	1.02	3,747	1,863	41.51	53.54	0.19
1	5	7X5X11	0.74	0.55	0.63	2,842	1,073	55.64	67.50	0.31
1	6	6X2X4	0.74	1.13	1.03	3,899	1,889	39.14	53.94	0.26
1	6	6X3X4	0.72	1.23	1.13	4,010	2,036	37.41	51.15	0.25
1	6	7X5X11	0.73	0.61	0.65	2,917	1,166	54.46	65.77	0.31
1	7	1X1X11	0.75	0.55	0.63	2,809	1,063	56.15	67.83	0.32
1	7	1X4X11	0.74	0.55	0.63	2,835	1,072	55.74	67.21	0.32
1	8	6X2X4	0.74	1.13	1.03	3,899	1,889	39.14	53.94	0.26
1	8	6X3X4	0.72	1.23	1.13	4,010	2,036	37.41	51.15	0.25
1	8	7X5X11	0.73	0.61	0.65	2,917	1,166	54.46	65.77	0.31
1	9	1X1X2	0.74	0.56	0.59	2,856	1,029	55.42	69.80	0.34
1	9	1X2X2	0.74	0.56	0.59	2,866	1,051	55.26	68.64	0.33
1	9	1X2X6	0.74	0.55	0.58	2,711	1,018	57.68	69.66	0.38
1	9	1X4X2	0.74	0.57	0.59	2,876	1,050	55.10	68.69	0.33
1	9	6X2X6	0.74	0.64	0.60	2,872	1,113	55.16	68.69	0.38
1	9	7X4X2	0.73	0.56	0.60	2,877	1,055	55.10	68.96	0.33
2	1	1X2X8	0.69	1.28	0.93	3,648	1,781	43.05	56.92	0.27
2	2	7X3X14	0.72	0.70	0.63	3,270	1,113	48.96	63.97	0.35
2	3	6X2X14	0.72	0.79	0.72	3,439	1,340	46.31	58.28	0.30
2	3	6X3X14	0.69	0.81	0.74	3,511	1,401	45.19	56.85	0.29
2	3	6X4X14	0.71	0.81	0.74	3,477	1,377	45.73	58.00	0.30
2	4	7X3X14	0.72	0.70	0.63	3,270	1,113	48.96	63.97	0.35
2	5	7X1X14	0.72	0.70	0.63	3,257	1,104	49.16	63.75	0.34
2	5	7X3X14	0.70	0.71	0.64	3,289	1,152	48.65	63.04	0.34
2	6	7X1X14	0.72	0.70	0.63	3,257	1,104	49.16	63.75	0.34
2	6	7X3X14	0.70	0.71	0.64	3,289	1,152	48.65	63.04	0.34
2	7	7X3X14	0.72	0.70	0.63	3,270	1,113	48.96	63.97	0.35
2	8	7X3X11	0.68	0.70	0.63	3,313	1,126	48.28	64.21	0.35
2	9	1X2X14	0.69	0.73	0.65	3,247	1,158	49.32	63.16	0.34
2	9	1X3X14	0.70	0.73	0.65	3,232	1,132	49.55	63.83	0.34
2	9	1X4X14	0.69	0.74	0.66	3,253	1,133	49.23	63.01	0.34
2	9	1X5X14	0.70	0.71	0.63	3,192	1,082	50.17	64.63	0.35
3	1	2X4X6	0.81	0.93	0.54	2,839	1,004	55.69	70.83	0.39
3	2	2X1X1	0.78	0.66	0.50	2,716	921	57.60	72.27	0.42
3	2	2X5X2	0.80	0.66	0.48	2,680	876	58.16	72.81	0.43
3	3	2X5X1	0.80	0.66	0.48	2,674	885	58.26	73.45	0.43

Table 15 Best learning schemes models performance *(Continued)*

DS	C	LS	S	MMRE	MdMRE	MMAR	MdMAR	MSA	MdSA	Pred25
3	4	2X5X1	0.80	0.66	0.48	2,680	876	58.16	72.81	0.43
3	5	7X4X6	0.79	0.55	0.55	2,536	1,021	60.42	72.88	0.39
3	5	7X5X6	0.78	0.56	0.56	2,559	1,027	60.06	72.74	0.38
3	6	2X1X6	0.80	0.94	0.55	2,872	1,017	55.17	70.14	0.38
3	6	2X4X6	0.81	0.94	0.55	2,862	1,040	55.33	70.22	0.38
3	6	2X5X6	0.81	0.93	0.54	2,845	990	55.60	70.88	0.39
3	6	7X2X6	0.78	0.57	0.57	2,586	1,040	59.64	72.03	0.37
3	7	2X1X1	0.78	0.66	0.50	2,716	921	57.60	72.27	0.42
3	7	2X5X1	0.80	0.66	0.48	2,680	876	58.16	72.81	0.43
3	8	1X4X6	0.79	0.55	0.55	2,547	1,023	60.25	72.77	0.39
3	8	1X5X6	0.78	0.57	0.56	2,571	1,050	59.86	72.38	0.38
3	8	2X4X6	0.81	0.94	0.55	2,864	1,029	55.29	70.57	0.39
3	9	2X4X1	0.80	0.66	0.48	2,686	879	58.07	72.75	0.43
3	9	2X5X1	0.79	0.66	0.49	2,691	897	58.00	72.58	0.43
4	1	6X5X3	0.74	0.80	0.65	2,815	1,236	56.06	65.63	0.34
4	2	6X5X3	0.74	0.79	0.68	2,801	1,251	56.28	66.05	0.34
4	2	6X1X3	0.74	0.80	0.65	2,799	1,234	56.32	66.04	0.35
4	3	7X1X6	0.77	0.60	0.56	2,596	988	59.47	71.15	0.39
4	3	7X5X6	0.76	0.60	0.58	2,624	1,026	59.04	70.50	0.38
4	4	6X1X3	0.74	0.79	0.68	2,801	1,251	56.28	66.05	0.34
4	4	6X5X3	0.74	0.80	0.65	2,799	1,234	56.32	66.04	0.35
4	5	2X2X1	0.76	0.56	0.54	2,768	984	56.79	69.85	0.41
4	6	5X2X1	0.71	0.68	0.72	3,143	1,304	50.94	62.65	0.31
3	4	2X1X1	0.78	0.66	0.50	2,716	921	57.60	72.27	0.42

Appendix B: datasets descriptive statistics

Descriptive statistics for used datasets are presented in Table 14.

Appendix C: evaluation criteria and equations

The prediction accuracy of the models was tested following the criteria defined in the following equations:

$$MMRE_i = \frac{100}{N}\sum_{i=1}^{N}\frac{|e_i - \hat{e}_i|}{e_i} \tag{1}$$

$$Pred(n) = \tfrac{100}{N}\sum_{i=1}^{N}\begin{cases}1 \; if MRE_i \leq n/100\\ 0 \; otherwise\end{cases} \tag{2}$$

$$SC_s = 1 - \frac{6\sum_{i=1}^{N} d_i^2}{N(N^2-1)}, \tag{3}$$

where d_i represents the difference between the ordinal ranks assigned to each of the observations.

$$MAR = \frac{1}{N}\sum_{i=1}^{N}|e_i - \hat{e}_i| \tag{4}$$

$$SA_{P_i} = 1 - \frac{MAR_{P_i}}{MAR_{P_0}} * 100 \tag{5}$$

Table 16 Best learning schemes models performance

DS	C	LS	S	MMRE	MdMRE	MMAR	MdMAR	MSA	MdSA	Pred25
4	6	5X3X1	0.70	0.69	0.73	3,163	1,337	50.62	61.96	0.30
4	6	5X4X1	0.70	0.69	0.72	3,164	1,333	50.61	62.62	0.30
4	6	5X5X1	0.70	0.69	0.74	3,182	1,344	50.32	61.89	0.30
4	7	6X1X3	0.74	0.79	0.68	2,801	1,251	56.28	66.05	0.34
4	7	6X5X3	0.74	0.80	0.65	2,799	1,234	56.32	66.04	0.35
4	8	2X5X2	0.74	0.65	0.77	3,532	1,324	44.86	60.85	0.23
4	8	6X1X3	0.74	0.80	0.65	2,807	1,216	56.18	66.04	0.35
4	9	7X5X6	0.76	0.59	0.57	2,608	1,018	59.29	70.95	0.38
5	1	7X1X3	0.72	0.95	1.13	3,083	1,400	41.67	68.85	0.23
5	2	6X5X1	0.72	0.81	0.91	2,859	1,210	45.90	72.33	0.26
5	3	7X1X2	0.72	0.64	0.82	2,660	944	49.66	78.67	0.29
5	3	7X1X3	0.72	0.96	1.14	3,119	1,438	40.98	68.13	0.22
5	3	7X2X3	0.72	0.96	1.15	3,142	1,442	40.54	67.72	0.22
5	4	6X5X1	0.72	0.81	0.91	2,859	1,210	45.90	72.33	0.26
5	5	1X1X4	0.72	1.43	1.64	3,606	2,134	31.77	54.83	0.19
5	5	1X1X6	0.72	0.63	0.81	2,466	969	53.34	77.96	0.30
5	5	1X2X2	0.72	0.63	0.82	2,509	925	52.52	79.04	0.29
5	5	1X3X2	0.72	0.64	0.82	2,510	924	52.51	79.10	0.29
5	5	1X3X4	0.72	1.45	1.63	3,587	2,136	32.13	54.96	0.19
5	6	1X1X2	0.72	0.64	0.81	2,559	945	51.58	78.76	0.29
5	6	1X1X3	0.72	0.91	1.07	2,950	1,356	44.17	69.56	0.23
5	6	1X5X3	0.72	0.91	1.08	2,965	1,374	43.89	69.11	0.22
5	7	6X5X1	0.72	0.81	0.91	2,859	1,210	45.90	72.33	0.26
5	8	7X2X2	0.72	0.63	0.81	2,520	923	52.31	79.11	0.30
5	8	7X2X3	0.72	0.93	1.09	2,926	1,356	44.64	69.76	0.24
5	8	7X3X2	0.72	0.63	0.80	2,520	926	52.33	79.19	0.30
5	8	7X3X3	0.72	0.93	1.10	2,934	1,367	44.48	69.48	0.24
5	9	1X1X6	0.73	0.63	0.79	2,517	948	52.38	78.29	0.31
5	9	1X3X6	0.72	0.64	0.80	2,520	959	52.32	77.83	0.31
6	1	6X3X1	0.70	0.80	0.90	2,677	1,105	49.36	74.96	0.27
6	2	6X4X1	0.71	0.80	0.89	2,688	1,108	49.14	74.84	0.27
6	3	6X1X1	0.71	0.80	0.89	2,684	1,109	49.21	74.96	0.28
6	3	6X2X1	0.70	0.80	0.90	2,696	1,138	48.99	74.37	0.27
6	4	6X4X1	0.71	0.80	0.89	2,688	1,108	49.14	74.84	0.27
6	5	6X2X1	0.71	0.80	0.90	2,736	1,124	48.24	74.65	0.27
6	6	6X3X1	0.70	0.80	0.91	2,733	1,130	48.28	74.49	0.27
6	6	6X4X1	0.70	0.80	0.91	2,734	1,129	48.27	74.54	0.27
6	6	6X5X1	0.71	0.80	0.90	2,718	1,115	48.57	74.75	0.28
6	7	6X4X1	0.71	0.80	0.89	2,688	1,108	49.14	74.84	0.27
6	8	6X4X2	0.70	0.63	0.83	2,748	964	48.00	78.53	0.29
6	8	6X4X3	0.71	1.34	1.48	3,555	2,097	32.74	53.28	0.19
6	9	1X1X6	0.71	0.67	0.79	2,613	973	50.55	77.41	0.31
6	9	1X2X2	0.68	0.67	0.83	2,635	963	50.14	78.00	0.28
6	9	1X2X6	0.71	0.67	0.78	2,605	953	50.72	77.82	0.31
6	9	1X3X6	0.70	0.67	0.79	2,620	970	50.42	77.53	0.30
7	1	7X3X14	0.75	0.72	0.84	2,611	1,028	50.59	76.67	0.30
7	1	7X5X14	0.76	0.71	0.83	2,615	1,018	50.51	76.84	0.30
7	2	6X4X6	0.79	0.61	0.72	2,360	894	55.34	80.20	0.35
7	3	6X2X2	0.79	0.55	0.73	2,295	870	56.58	80.48	0.32
7	3	6X2X6	0.80	0.60	0.70	2,174	881	58.86	80.41	0.35
7	3	6X3X6	0.79	0.61	0.72	2,214	904	58.11	80.03	0.35

Table 16 Best learning schemes models performance *(Continued)*

DS	C	LS	S	MMRE	MdMRE	MMAR	MdMAR	MSA	MdSA	Pred25
7	3	6X5X6	0.79	0.61	0.72	2,206	902	58.25	80.14	0.35
7	4	6X4X6	0.79	0.61	0.72	2,360	894	55.34	80.20	0.35
7	5	6X2X3	0.77	0.84	0.92	2,612	1,238	50.58	72.58	0.28
7	5	6X3X3	0.77	0.86	0.94	2,625	1,259	50.33	72.15	0.27
7	6	6X2X3	0.77	0.84	0.92	2,612	1,238	50.58	72.58	0.28
7	6	6X3X3	0.77	0.86	0.94	2,625	1,259	50.33	72.15	0.27
7	7	6X4X6	0.79	0.61	0.72	2,360	894	55.34	80.20	0.35
7	8	1X3X14	0.77	0.71	0.84	2,655	1,047	49.76	76.19	0.31
7	8	6X2X14	0.78	0.73	0.87	2,674	1,110	49.40	75.31	0.29
7	9	1X3X14	0.77	0.71	0.84	2,655	1,047	49.76	76.19	0.31
7	9	6X2X14	0.78	0.73	0.87	2,674	1,110	49.40	75.31	0.29
8	1	2X3X1	0.73	0.76	0.87	2,691	1,117	49.08	74.22	0.28
8	2	1X5X3	0.69	0.94	1.01	2,941	1,406	44.34	69.05	0.25
8	3	2X4X1	0.73	0.74	0.85	2,563	1,084	51.50	75.00	0.29
8	3	2X5X1	0.73	0.73	0.85	2,549	1,075	51.76	75.35	0.29
8	4	1X5X3	0.69	0.94	1.01	2,941	1,406	44.34	69.05	0.25
8	5	2X1X1	0.74	0.73	0.85	2,558	1,084	51.60	75.20	0.29
8	5	2X5X1	0.73	0.74	0.87	2,570	1,112	51.37	74.66	0.29
8	6	2X4X1	0.73	0.73	0.85	2,577	1,100	51.23	74.76	0.29
8	6	2X5X1	0.73	0.73	0.86	2,591	1,115	50.98	74.41	0.29
8	7	1X5X3	0.69	0.94	1.01	2,941	1,406	44.34	69.05	0.25
8	8	2X2X6	0.75	0.81	0.89	2,547	1,144	51.80	74.77	0.30
8	9	2X3X1	0.73	0.74	0.87	2,643	1,091	50.00	74.58	0.28
8	9	2X5X1	0.74	0.74	0.86	2,631	1,085	50.21	74.91	0.29

where MAR_{P_0} is the unbiased exact version of MAR_{P_0} recommended for small datasets.

$$MAR_{P_0} = \frac{2}{n_2} \sum_{i=1}^{n} \sum_{j=1}^{j<i} \left| e_i - e_j \right| \tag{6}$$

Appendix D: best learning schemes models performance

Best Learning Schemes Models Performance are presented in Tables 15 and 16.

Abbreviations

AR: AdditiveRegression; AS: Attribute Selector; BC: Box-Cox; BF: BestFirst; BE: Backward elimination; BAG: Bagging; CR: ConjunctiveRule; DT: DecisionTable; DS: DecisionStump; FS: Forward Selection; GS: Genetic Search; GP: GaussianProcesses; Log: Logarithmic; LFS: LinearForwardSelection; LA: Learning Algorithm; LMS: LeastMedSq; LR: LinearRegression; MP: MultilayerPerceptron; M5R: M5Rules; M5P: M5P; None: None; PP: Data preprocessing; RBFN: RBFNetwork; RT: REPTree SMO: SMOreg; ZR: ZeroR

Acknowledgments
This research was supported in part by the University of Costa Rica (No. 834-B5-A18), the National University of Costa Rica, and the Ministry of Science, Technology and Telecommunications (MICITT). We thank The International Software Benchmarking Standards Group (ISBSG).

Funding
This research was supported by University of Costa Rica, National University of Costa Rica, and Costa Rica Ministry of Science, Technology and Telecommunications (MICITT).

Availability of data and materials

The genetic experimental package is available at https://goo.gl/v85ddl.

Authors' contributions

All authors are equal contributors for the work and for the definition of the approach and experimental design. First and second authors are the main responsible for the implementation of the framework and they collected the data of the experiments. All authors helped in the analysis of results and writing. All authors read and approved the final manuscript.

Authors' information
Juan Murillo-Morera is a doctoral candidate in the Department of Computer Science at the University of Costa Rica. His research interests are in the application of data mining and machine learning to software engineering problems. Contact him at juan.murillomorera@ucr.ac.cr.
Christian Quesada-López is a doctoral candidate in the Department of Computer Science at the University of Costa Rica. His main research interests are empirical software engineering, measurement, and software quality assurance. Contact him at cristian.quesadalopez@ucr.ac.cr http://citic.ucr.ac.cr/perfil/cristian-quesada-lópez.
Carlos Castro-Herrera received his PhD degree in computer science and MS degree in software engineering from DePaul University, in Chicago, IL. His research interests are in the application of data mining and machine learning to software engineering problems. He has authored over a dozen peer reviewed papers and journals. Professionally, he has worked as a Data Scientist at Intel, an invited professor at the University of Costa Rica, and is currently working as a software engineer at Google, Chicago.
Marcelo Jenkins obtained a B.S. degree in Computer and Information Sciences at the University of Costa Rica in 1986 and a M.Sc. and Ph.D. degrees from the University of Delaware, USA, in 1988 and 1992 respectively. Since 1986 he has been teaching computer science at the University of Costa Rica. His research interests are in empirical software engineering, software quality assurance, project management, and object-oriented programming. He has authored more than 60 technical papers on these subjects. Contact him at Contact him at marcelo.jenkins@ucr.ac.cr
http://www.citic.ucr.ac.cr/perfil/marcelo-jenkins-coronas.

Competing interests
The authors declare that they have no competing interests.

Publisher's Note

Author details
[1]Center for ICT Research, University of Costa Rica, San Pedro de Montes de Oca, San José, Costa Rica. [2]Google, Chicago, 24105 Chicago, USA.

Received: 13 December 2016 Accepted: 3 May 2017
Published online: 31 May 2017

References

Albrecht AJ (1979) Measuring application development productivity. In: Proceedings of the Joint SHARE/GUIDE/IBM Application Development Symposium, vol 10. IBM Press. pp 83–92
Aljahdali S, Sheta A (2013) Evolving software effort estimation models using multigene symbolic regression genetic programming. Int J Adv Res Artif Intell 2:52–57
Arcuri A, Fraser G (2011) On parameter tuning in search based software engineering. In: International Symposium on Search Based Software Engineering. Springer, Berling. pp 33–47
Bala A, Sharma AK (2015) A comparative study of modified crossover operators. In: 2015 Third International Conference on Image Information Processing (ICIIP). IEEE, Waknaghat. pp 281–284. doi:10.1109/ICIIP.2015.7414781
Becker BG (1998) Visualizing decision table classifiers. In: Information Visualization, 1998. Proceedings. IEEE Symposium On. IEEE, Research Triangle. pp 102–105
Boehm B (1981) Software Engineering Economics, Vol. 197. Prentice-hall Englewood Cliffs (NJ), USA
Chen J, Nair V, Menzies T (2017) Beyond evolutionary algorithms for search-based software engineering. arXiv preprint arXiv:1701.07950
Dejaeger K, Verbeke W, Martens D, Baesens B (2012) Data mining techniques for software effort estimation: a comparative study. IEEE Trans Softw Eng 38(2):375–397
Dolado JJ, Rodriguez D, Harman M, Langdon WB, Sarro F (2016) Evaluation of estimation models using the minimum interval of equivalence. Appl Soft Comput 49:956–967. http://dx.doi.org/10.1016/j.asoc.2016.03.026
Ghatasheh N, Faris H, Aljarah I, Al-Sayyed RM, et al. (2015) Optimizing software effort estimation models using firefly algorithm. J Softw Eng Appl 8(03):133
González-Ladrón-de-Guevara F, Fernández-Diego M (2014) Isbsg variables most frequently used for software effort estimation: A mapping review. In: Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. ACM, Torino. p 42
Harman M (2007) The current state and future of search based software engineering. In: 2007 Future of Software Engineering. IEEE Computer Society, Minneapolis. pp 342–357
Harman M, Jones BF (2001) Search-based software engineering. Inf Softw Technol 43(14):833–839
Harman M, Burke E, Clark JA, Yao X (2012) Dynamic adaptive search based software engineering. In: Empirical Software Engineering and Measurement (ESEM), 2012 ACM-IEEE International Symposium On. IEEE, Lund Sweden. pp 1–8
Hill P (2010) Practical Software Project Estimation: A Toolkit for Estimating Software Development Effort & Duration. McGraw Hill Professional
Holmes G, Hall M, Prank E (1999) Generating rule sets from model trees. In: Australasian Joint Conference on Artificial Intelligence. Springer, Berling. pp 1–12
Huang J, Li Y-F, Xie M (2015) An empirical analysis of data preprocessing for machine learning-based software cost estimation. Inf Softw Technol 67:108–127
ISBSG (2015) The isbsg development & enhancement project data r12. International Software Benchmarking Standards Group. http://www.isbsg.org
Jeng B, Yeh D, Wang D, Chu S-L, Chen C-M, et al. (2011) A specific effort estimation method using function point. J Inf Sci Eng 27(4):1363–1376

Jørgensen M (2004) A review of studies on expert estimation of software development effort. J Syst Softw 70(1):37–60
Jorgensen M, Shepperd M (2007) A systematic review of software development cost estimation studies. IEEE Trans Softw Eng 33(1):33–53
Keung J, Kocaguneli E, Menzies T (2013) Finding conclusion stability for selecting the best effort predictor in software effort estimation. Autom Softw Eng 20(4):543–567
Langdon WB, Dolado J, Sarro F, Harman M (2016) Exact mean absolute error of baseline predictor, marp0. Inf Softw Technol 73:16–18
Lavazza L, Morasca S, Robiolo G (2013) Towards a simplified definition of function points. Inf Softw Technol 55(10):1796–1809
Lefley M, Shepperd MJ (2003) Using genetic programming to improve software effort estimation based on general data sets. In: Genetic and Evolutionary Computation Conference. Springer, Berlin. pp 2477–2487
Lokan C, Mendes E (2006) Cross-company and single-company effort models using the isbsg database: a further replicated study. In: Proceedings of the 2006 ACM/IEEE International Symposium on Empirical Software Engineering. ACM, Rio de Janeiro. pp 75–84
MacDonell SG, Shepperd MJ (2003) Combining techniques to optimize effort predictions in software project management. J Syst Softw 66(2):91–98
Mair C, Shepperd M (2005) The consistency of empirical comparisons of regression and analogy-based software project cost prediction. In: Empirical Software Engineering, 2005. 2005 International Symposium On. IEEE, p 10
Malhotra R (2014) Comparative analysis of statistical and machine learning methods for predicting faulty modules. Appl Soft Comput 21:286–297
Meffert K, Rotstan N (2005) Jgap: Java genetic algorithms package. http://jgap.sourceforge.com. Accessed 20 May 2017
Mendes E, Lokan C (2008) Replicating studies on cross-vs single-company effort models using the isbsg database. Empir Softw Eng 13(1):3–37
Mendes E, Lokan C, Harrison R, Triggs C (2005) A replicated comparison of cross-company and within-company effort estimation models using the isbsg database. In: Software Metrics, 2005. 11th IEEE International Symposium. IEEE, Como. pp 10–36
Menzies T, Shepperd M (2012) Special issue on repeatable results in software engineering prediction. Empir Softw Eng 17(1):1–17
Minku LL, Yao X (2013) Ensembles and locality: Insight on improving software effort estimation. Inf Softw Technol 55(8):1512–1528
Mittas N, Angelis L (2008) Comparing cost prediction models by resampling techniques. J Syst Softw 81(5):616–632
Moløkken K, Jørgensen M (2003) A review of software surveys on software effort estimation. In: Proceedings of IEEE International Symposium on Empirical Software Engineering ISESE. IEEE, Rome. pp 223–230
Murillo-Morera J, Quesada-Løpez C, Castro-Herrera C, Jenkins M (2016a) An empirical validation of an automated genetic software effort prediction framework using the isbsg dataset. In: CIBSE 2016 - XIX Ibero-American Conference on Software Engineering. Scopus, Quito. pp 185–199
Murillo-Morera J, Castro-Herrera C, Arroyo J, Fuentes-Fernández R (2016b) An automated defect prediction framework using genetic algorithms: A validation of empirical studies. Intel Artif 19(57):114–137
Murillo-Morera J, Castro-Herrera C, Arroyo J, Fuentes-Fernández R (2016c) An empirical validation of learning schemes using an automated genetic defect prediction framework. In: Ibero-American Conference on Artificial Intelligence. Springer, San José. pp 222–234
Quesada-López C, Jenkins M (2015) An empirical validation of function point structure and applicability: A replication study. In: CIBSE 2015 - XVIII Ibero-American Conference on Software Engineering. Scopus, Quito. pp 418–431
Quesada-Lopez C, Murillo-Morera J, Castro-Herrera C, Jenkins M (2016) Benchmarking Software Effort Prediction Models: An Automated Prototype Tool (Tecnhical Research Report Ref. 01-07-16-001 No. 834-B5-A18) Technical report, University of Costa Rica, 2016. Retrieved from: https://goo.gl/vj1qCa
Quesada-Løpez C, Murillo-Morera J, Castro-Herrera C, Jenkins M (2016) Cosmic base functional components in functional size based effort estimation models. In: Central America and Panama Convention (CONCAPAN XXXVI). IEEE, San José
Quesada-López C, Jenkins M (2014) Function point structure and applicability validation using the isbsg dataset: a replicated study. In: Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. ACM, San José. p 66
Quesada-López, C, Jenkins M (2016) Function point structure and applicability: A replicated study. J Object Technol 15
Quinlan JR, et al. (1992) Learning with continuous classes. In: 5th Australian Joint Conference on Artificial Intelligence, vol 92. World Scientific, Singapore. pp 343–348. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.34.885
Rédei GP (2008) Encyclopedia of Genetics, Genomics, Proteomics, and Informatics. Springer, Germany
Sakia R (1992) The box-cox transformation technique: a review Statistician. J R Stat Soc Ser D Stat 41(2):169–178. http://dx.doi.org/10.2307/2348250
Sayyad AS, Goseva-Popstojanova K, Menzies T, Ammar H (2013) On parameter tuning in search based software engineering: A replicated empirical study. In: Replication in Empirical Software Engineering Research (RESER), 2013 3rd International Workshop On, IEEE. pp 84–90
Seo Y-S, Bae D-H, Jeffery R (2013) Areion: Software effort estimation based on multiple regressions with adaptive recursive data partitioning. Inf Softw Technol 55(10):1710–1725
Shepperd M (2007) Software project economics: a roadmap. In: Future of Software Engineering, 2007. FOSE'07. IEEE, pp 304–315
Shepperd M, MacDonell S (2012) Evaluating prediction systems in software project estimation. Inf Softw Technol 54(8):820–827
Singh BK, Misra A (2012) Software effort estimation by genetic algorithm tuned parameters of modified constructive cost model for nasa software projects. Int J Comput Appl 59(9):22–26. http://research.ijcaonline.org/volume59/number9/pxc3884053.pdf
Song L, Minku LL, Yao X (2013) The impact of parameter tuning on software effort estimation using learning machines. In: Proceedings of the 9th International Conference on Predictive Models in Software Engineering. ACM, p 9

Song Q, Jia Z, Shepperd M, Ying S, Liu J (2011) A general software defect-proneness prediction framework. Softw Eng IEEE Trans 37(3):356–370
Wang Y, Witten IH (1997) Induction of model trees for predicting continuous classes, Proceedings of the poster papers of the European Conference on Machine Learning. University of Economics, Faculty of Informatics and Statistics, Prague
Wen J, Li S, Lin Z, Hu Y, Huang C (2012) Systematic literature review of machine learning based software development effort estimation models. Inf Softw Technol 54(1):41–59
Witten IH, Frank E (2005) Data Mining: Practical Machine Learning Tools and Techniques. Morgan Kaufmann, USA