Article

A Novel Data Analytics Method for Predicting the Delivery Speed of Software Enhancement Projects

Elías Ventura-Molina ¹, Cuauhtémoc López-Martín ^{2,*}, Itzamá López-Yáñez ^{1,*}
and Cornelio Yáñez-Márquez ^{3,*}

¹ Centro de Innovación y Desarrollo Tecnológico en Cómputo, Instituto Politécnico Nacional, Ciudad de México 07700, Mexico; eventuram@ipn.mx

² Department of Information Systems, Universidad de Guadalajara, Zapopan, Jalisco 45100, Mexico

³ Centro de Investigación en Computación, Instituto Politécnico Nacional, Ciudad de México 07738, Mexico

* Correspondence: cuauhtemoc@cucea.udg.mx (C.L.-M.); ilopez@ipn.mx (I.L.-Y.); cyanez@cic.ipn.mx (C.Y.-M.)

Received: 11 September 2020; Accepted: 5 November 2020; Published: 10 November 2020

Abstract: A fundamental issue of the software engineering economics is productivity. In this regard, one measure of software productivity is delivery speed. Software productivity prediction is useful to determine corrective activities, as well as to identify improvement alternatives. A type of software maintenance is enhancement. In this paper, we propose a data analytics-based software engineering algorithm called search method based on feature construction (SMFC) for predicting the delivery speed of software enhancement projects. The SMFC belongs to the minimalist machine learning paradigm, and as such it always generates a two-dimensional model. Unlike the usual data analytics methods, SMFC includes an original algorithmic training procedure, in which both the independent and dependent variables are considered for transformation. SMFC prediction performance is compared to those of statistical regression, neural networks, support vector regression, and fuzzy regression. To do this, seven datasets of software enhancement projects obtained from the International Software Benchmarking Standards Group (ISBSG) Release 2017 were used. The validation method is leave-one-out cross validation, whereas absolute residuals have been chosen as the performance measure. The results indicate that the SMFC is statistically better than statistical regression. This fact represents an obvious advantage in favor of SMFC, because the other two methods are not statistically better than SMFC.

Keywords: data analytics; software enhancement projects; delivery speed prediction; feature construction; search methods; simulated annealing; ISBSG

1. Introduction

Economics is the study of value, costs, resources, and their relationship in a given context or situation, whereas software engineering economics involves decision making related to software engineering in a business context. In spite of software engineering economics being concerned with aligning software technical decisions with the business goals of the organization, in many companies, software business relationships to software development and engineering remain vague.

The software engineering economics fundamentals are finance, accounting, controlling, cash flow, decision-making process, valuation, inflation, depreciation, taxation, time-value of money, efficiency, effectiveness, and productivity. Productivity has been defined as the ratio of output over input from an economic perspective (i.e., maximizing productivity is about generating the highest value with the lowest resource consumption). Output is the value delivered, whereas input covers all resources spent to generate the output [1].

The economic interest regarding the study and measurement of productivity has allowed for comparing and proposing public policies aimed at industry sectors [2]. In the software engineering field, a software productivity measure is delivery speed, which measures the speed achieved in delivering a quantity of software (i.e., size) over a period of time [3]. Software size has mainly been measured either in lines of code or function points [4]. Our study is related to the software productivity prediction, which is useful to determine corrective activities, as well as to identify improvement alternatives [5].

Developer human factors refer to the considerations of human factors taken when developing software [1]. In our study, the type of productivity to be predicted is delivery speed, which is affected by the human resource management of an organization [6].

The types of development of software projects can be classified into new and maintenance [7]. Corrective, adaptive, reengineering, and enhancement are types of software maintenance. The type of projects predicted in our study is enhancement, which has been defined as “changes made to an existing application where new functionality has been added, or existing functionality has been changed or deleted” [8]. Since productivity rates are different between new and enhancement software projects, software productivity prediction should then be separately performed.

Prediction techniques in the software engineering field have been used for predicting size, effort, duration, and costs [9]. However, in the extensive documentary research carried out for the development of this article, not one study proposing the application of any technique for delivery speed prediction was found. This lack of previous work fully justifies the current investigation.

Search-based software engineering consists on the application of computational intelligence algorithms to hard optimization problems in software engineering [10]. Algorithms of this type have recently been applied to software product and process [11–18]. In this context and as a contribution to the state of the art in the area of search-based software engineering, this article proposes and describes a new and original search-based software engineering method, and it is applied to the prediction of delivery speed. The new proposal is called search method based on feature construction (SMFC) and belongs to the minimalist machine learning paradigm [19].

A very relevant part of this article is the application of the new SMFC to delivery speed prediction. Since the prediction performance of any new proposal should at least outperform that of a statistical regression [20], the SMFC prediction performance is compared to that of a multiple linear regression model (MLR) with logarithmic transformation. In addition, the SMFC is compared against three of the most important standard machine learning models: feedforward neural networks, specifically a multilayer perceptron (MLP); two support vector regression (SVR) models; fuzzy regression (FR).

The software enhancement projects used for training and testing the models were obtained from the International Software Benchmarking Standards Group (ISBSG) Release from May 2017, which consists of 8012 projects implemented between 1989 and 2016 [3]. The patterns are formed with two independent variables: the unadjusted function points (UFP) and the maximum number of participants in each project, which is referred to as max team size (MTS). Each of the patterns is associated with a specific value of the dependent variable delivery speed (DS). More details are presented in Section 4.

The hypothesis investigated in this study is the following:

Hypothesis 1 (H1). *The delivery speed (DS) prediction performance of software enhancement projects with SMFC is statistically better than the accuracies obtained with MLR, MLP, SVR, and FR when UFP and MTS are used as the independent variables.*

The rest of this paper is organized as follows: in Section 2, some of the articles dealing with software delivery speed are briefly mentioned, and the different approaches are clearly specified, emphasizing the fact that in the extensive documentary research carried out during the development of this paper, no studies related to delivery speed prediction were found. In addition, MLR, FR, MLP, and SVR models are briefly described. Section 3 includes basic elements of the minimalist

machine learning paradigm, while Section 4 describes in detail the central proposal of this article (SMFC), carefully exemplifying each of the four steps of the new model. Some considerations about the complexity of the algorithm are also included. Section 5 is of utmost importance because it describes the criteria observed to select the data sets of software enhancement projects. Section 6 presents the experimental results, whereas Section 7 presents the discussion, conclusions, and the limitations of our proposal, as well as the future work.

2. Related Work

This section briefly mentions some of the few articles dealing with software delivery speed, and the different topics that are covered in relation to this topic. In addition, very brief descriptions of the three models mentioned in the introduction are included, against which our proposal is compared.

2.1. Delivery Speed

Delivery speed is a subject that is rarely covered in the scientific literature of software engineering. This topic has recently been studied from several approaches, such as its influence on globally distributed projects [21], its relationship with quality improvement [22], as well as the value aspect in agile software development organizations [23].

Moreover, the influence of some factors on delivery speed has been analyzed such as reusing [24], the application of automated toolchains, and of agile practices such as Kanban, Scrum, and Extreme Programming (XP) [25].

It is highly pertinent to emphasize the fact that in the extensive documentary research carried out during the development of this paper, the authors did not find any studies related to delivery speed prediction. The relevance of this fact is that this completely justifies the originality of the approach given to the research reported herein. The good results presented in this paper open up a new vein of scientific research: delivery speed prediction using data analytics methods.

In accordance with a systematic review of studies on effort prediction published in 2012 based on machine learning models, neural networks and support vector regression (SVR) reported the best prediction performance [26], whereas according to another systematic review recently published in 2018, their application remains of relevance [27]. Considering this, and in addition to the application of MLR, in this article a multilayer feedforward perceptron (MLP) neural network, and two types of SVR have also been applied to be compared to SMFC performance.

2.2. MLR and FR

Statistical regression is a very popular technique that is used in applications in many and diverse areas of science and engineering. Specifically in software engineering, statistical regression is the usual technique when it is required to perform regression of functions. In fact, one of the conditions to take into account any new proposal in software engineering is that the performance should at least outperform that of statistical regression [20].

If when solving a regression problem there is a dependent variable y depending on two or more independent variables $x_1, x_2, \dots, x_k$, statistical regression is called multiple linear regression (MLR). It is assumed that variable y is a linear function of the independent variables $x_1, x_2, \dots, x_k$. The linear relationship between the independent variables and the dependent variable is modeled, and this modeling is represented by Expression (1) [28]:

$$y = b_0 + b_1x_1 + \dots + b_kx_k \quad (1)$$

where $b_0, b_1, \dots, b_k$ are constants whose value must be adjusted according to the data of the problem under study.

If we restrict ourselves to the case where there are two independent variables and apply a logarithmic transformation, Expression (1) becomes Expression (2), where the new constants are a, b, c :

$$\ln(y) = a + b \ln(x_1) + c \ln(x_2) \quad (2)$$

In Equation (1), which represents the MLR model, it is considered that $b_0, b_1, \dots, b_k$ are constants whose value must be adjusted. If these parameters are substituted with fuzzy intervals, then the model becomes fuzzy regression (FR), which is the fuzzy version of MLR. FR determines a fuzzy linear relationship between the independent variables and the dependent variable [28].

2.3. MLP

A neural network (NN) deals with real-world problems that are nonlinear. It is defined from its learning paradigm, learning algorithm, and topology. A learning paradigm can either be supervised, unsupervised, or a reinforcement process; where the supervised one is commonly used for classification and prediction applications, the unsupervised one for data clustering and segmentation, and reinforcement is usually applied in optimization over time and adaptive control. In the supervised paradigm, the learning algorithm calculates the difference between the correct output and the actual prediction generated from the neural network, then this difference is used for adjusting the weights of the NN such that next time, the prediction is closer to the desired output [29].

The manner of how the neural processing units named neurons and their interconnections are related influences on the NN performance. An NN has a set of neurons which receive inputs from the outside world. These neurons are known as input units. Moreover, an NN has one or more hidden layers also consisting of neurons that receive inputs from other neurons. Each layer receives a vector of data or the outputs of a previous layer of neurons for processing them in parallel. The neuron representing the final result of the NN is called output unit [26].

Regarding topology, feedforward, limited recurrent, and fully recurrent network are three types of connection topologies that define how the data flow between the input, hidden, and output neurons. A connection topology does not refer to any specific type of activation function or training paradigm. In our study, a multilayer feedforward perceptron (MLP) with back propagation learning algorithm is applied since it has been the most commonly used algorithm and successfully applied to effort prediction [27]. An MLP uses a feedforward topology, supervised learning, and the back-propagation learning algorithm.

An MLP can have a layer of input neurons, one or more layers of hidden neurons, and finally a layer of output neurons. In this study, an MLP with a single hidden layer of neurons is used since it can model any continuous function to any degree of performance.

In an MLP, the data flow through it in one direction, and the response is based on the current set of inputs. For the current study, the size of projects and the number of developers by project enter the MLP through the input neurons. The input values of software projects are assigned to the input neurons as the unit activation values. The output value of the neuron is modulated by means of connection weights. A threshold value is used by each neuron to combine all of the input signals. This input signal is passed through an activation function to determine the actual output of the neuron.

The type of activation function suggested for hidden layer neurons has been non-linear because of its capacity for learning non-linear relationships among variables. The alternative most frequently used function in literature for such problems is the sigmoid function, which converts an input value to an output ranging from 0.0 to 1.0 [29]. Since this study is related to prediction, the type of activation

function for output layer is linear. The sigmoid and piecewise-linear functions used here are described in the following equations.

$$\Phi(v) = \begin{cases} \frac{1}{1+e^{-av}} & \text{for } (0, 1) \\ \tanh(v) & \text{for } (-1, 1) \\ \begin{cases} -1 & v \leq v_1 \\ a_1v + a_0 & v_1 < v \leq v_2 \\ 1 & v_2 < v \end{cases} & \end{cases} \quad (3)$$

where v is the internal state of the neuron, which is calculated by summing the inner product of the input vector, the weight vector, and a bias value, whereas $y = \Phi(v)$ corresponds to the output of the neuron.

2.4. SVR

An SVM has a main concept to distinguish items into two groups. This model seeks to find the optimal hyperplanes determined by support vectors for linearly separable classes [30]. Equation (4) describes an SVM in the plane, where w is the normal vector to the separation line satisfying for the point x , whereas b refers to bias (a measure of offset of the separation line from the origin).

$$wx - b = 0 \quad (4)$$

Equation (5) is used to find the optimal line from the Lagrange multipliers indicated by α_i , where the training observation values y_i are either 1 or -1 .

$$\sum_{i=1}^n \alpha_i y_i = 0 \quad (5)$$

The result is shown below, with a w vector corresponding to a linear combination of the support vectors x_i [31].

$$w = \sum_{i=1}^n \alpha_i y_i x_i \quad (6)$$

When the classes are not linearly separable, that is, in cases where there is no hyperplane that separates the two classes, a “soft margin” is proposed by seeking to minimize the number of errors in the two splits and maximizing the margin between these splits. Slack variables are used to measure the degree of misclassification by data point through the training phase, a penalty function is used, and Lagrange multipliers are restricted by a parameter C ($0 \leq \alpha_i \leq C$). An SVM uses kernels assuming a linear separation by mapping data into a higher-dimensional space. The preferred ones in the state of the art are linear, polynomial, radial basis function, and Sigmoid function.

According to [32], a support vector regression (SVR) is a type of SVM applicable to regression tasks. There are two types of SVRs: ε -support vector regression (ε -SVR) and ν -support vector regression (ν -SVR). The former is trained based on a symmetrical loss function (i.e., ε -insensitive) that penalizes high and low misestimates. An SVR looks for a function $f(x)$ having the most ε deviation from the target y_i for the data x_i . As for ν -SVR, it minimizes the ε -insensitive loss function and it uses new ν parameter (between 0 and 1) instead. This ν parameter controls the number of support vectors by allowing data compression and generalizing the prediction error bounds.

3. Basic Elements of the Minimalist Machine Learning Paradigm

The minimalist machine learning paradigm was recently presented as a response to a problem that afflicts the areas of machine learning and artificial intelligence [19]. Many of the most effective models used in these areas, especially the intelligent pattern classification models, are not explainable. That is, in order to achieve good performance in the classification of patterns, the models are less

transparent and less explainable, because complicated algorithmic steps are included. Specialists say that these models and algorithms behave like “black boxes” [33].

The SVM model is a clear example, because its good performance is based on the kernel trick to achieve separability of the classes [30]. However, the use of the kernel brings problems: the patterns are transformed to be represented in a space of greater dimension than the original patterns, which obviously decreases the explainability and increases the complication, at the cost of better results.

With the new minimalist machine learning paradigm, it has been achieved that the models of this paradigm are capable of minimizing classification errors, but without becoming “black boxes”.

The new paradigm is based on the strong assumption that it is possible to reduce any problem of pattern classification to a graphical problem on the Cartesian plane. This holds, regardless of how large the patterns dimension is.

The algorithms of this new paradigm are effective, transparent and explainable. Additionally, it should be noted that its high efficiency and effectiveness is due to only a few simple operations being used for both phases: learning and classification. Obviously, the interpretation of the results is immediate, because the user unambiguously and immediately detects the way in which the classification is carried out.

The idea is to convert, through a simple operation, all the features of a given pattern into a single real value, which will be located on the horizontal axis of the Cartesian plane. Then, through another simple operation, convert all the features of that same pattern into another real value, which will be located on the vertical axis of the Cartesian plane. Both values form an ordered pair whose graph is a point in the plane.

After performing the steps described in all the patterns of the classification problem to be solved, it is expected that two lines separated by a horizontal line will be formed.

The ideal case is one in which all the points of one class (C1) are above that horizontal line, while all the points of the other class (C2) appear below that same horizontal line.

Figure 1 illustrates the case in which the two simple operations involved are the standard deviation and the mean.

Figure 1. The two classes are separated by a horizontal line.

The reader can find in [19] an example developed from beginning to end. The example applies these two simple operations to the patterns of a real cancer-related dataset. The results can be replicated with the support of a pocket calculator or modest computer equipment.

The reader will witness the power of the minimalist machine learning paradigm, when verifying the results. While two of the best classifiers in the state of the art show excellent performances (SVM: 92.85%, and MLP: 96.42%), with the minimalist machine learning paradigm model, 100% accuracy was obtained.

One might wonder if, besides standard deviation and mean, there will be other operations that are useful for this paradigm. The answer is that any operation that converts an array of numbers to a real number could be useful. In fact, some combinations of operations on subsets of features may also be useful. The minimalist machine learning paradigm has just been born and has opened up a host of novel veins of scientific research.

One might also wonder if for any dataset there is a horizontal line that separates the classes. The answer in this case is a resounding no. This would be a contradiction to the No Free Lunch Theorem [31]. Although for many datasets this separation line does not exist, in these cases it is no longer sought to achieve zero errors, but rather to minimize the number of errors, which opens up other scientific research topics.

4. Our Proposal: Search Method Based on Feature Construction (SMFC)

The proposal introduced in this paper is a data-analytics-based method for predicting the delivery speed of software enhancement projects, which has three parts. The first part is the most important, because it gives the SMFC the character of a model belonging to the minimalist machine learning paradigm. This first part consists of a set of variable transformations, which allow the generation of a two-dimensional model. The importance of this model is that it describes the problem to be solved. The second part is a simple linear regression (SLR), while the third part consists of applying a metaheuristic search to optimize the parameters of the SLR model.

Section 4.1 describes five illustrative cases to exemplify the process of transforming variables. It also includes all the variable transformations that are used in the proposal. Then, Sections 4.2 and 4.3 describe the SLR model and metaheuristics, respectively. In Section 4.4, the SMFC model as a four-step integral whole is explained, and also includes some considerations on the complexity of the algorithm.

4.1. Variable Transformation and Overview

The first novel characteristic of SMFC is that it always generates a two-dimensional model, in contrast to the behavior of machine learning techniques such as SVM, in which the input vectors are mapped into a highly dimensional space [34]. In this sense, the basic assumption is that as an advantage evident of this proposal, regardless of the number of predictors (independent variables), it is always possible to find a transformation that generates a two-dimensional model whose representation space is the Cartesian plane.

An additional original characteristic of SMFC consists of that, for its training, both independent and dependent variables are considered for transformation. This is remarkable due to the novelty of including the dependent variable into the transformed independent variable. Currently, the authors do not have any knowledge about any study taking advantage of this issue.

In the acronym SMFC, FC means “Feature Construction”. In this model, those features are built by means of elemental transformations from the two types of variables: independent and dependent.

Let a problem in the software engineering field, whose set of involved variables, V , includes (without loss of generality), a dependent variable v_d and a set V_i of independent variables (i.e., predictors), whose cardinality can be higher than one as follows:

$$V = \{v_d\} \cup V_i \quad (7)$$

The transformations are next described by means of illustrative cases.

First illustrative case: The problem regarding the delivery speed (DS) of software enhancement projects of our study involves a dependent variable $v_d = DS$, as well as a set of two independent variables $V_i = \{UFP, MTS\}$, thus, the cardinality for the set of independent variables is two.

Let us now consider a finite set of n elemental transformations $T = \{\tau_1, \tau_2, \dots, \tau_n\}$ with $n \in \mathbb{Z}^+$, where each $\tau_1 \in T$ can be either an arithmetic operation (involving the problem variables, and even some other real parameters), a linear function, a nonlinear function (such as trigonometric, logarithmic, or exponential functions), an elemental statistical operation, or another option.

The main objective of the FC is to select a collection of elemental transformations τ_i and apply each of them to specific values of elements from the power set $2^{|V|}$, such that a collection of points on the plane is obtained. This collection corresponds to pairs of specific values that involve dependent and independent variables of the problem.

Second illustrative case: The following power set is obtained from the first illustrative case:

$$2^{|V|} = \{\emptyset, \{DS\}, \{UFP\}, \{MTS\}, \{DS, UFP\}, \{DS, MTS\}, \{UFP, MTS\}, V\} \quad (8)$$

Third illustrative case: There exists an infinite quantity of possibilities for selecting transformation combinations τ_i with elements from $2^{|V|}$, which can be combined from either real parameters or results of other transformations τ_i applied to other elements from $2^{|V|}$, such that a pair of specific values is obtained by case. For instance, if τ_1 was the sum of real numbers and then that τ_1 is applied to the two independent variables, then we would have $\tau_1(UFP, MTS) = UFP + MTS$; if it also happens that τ_2 is the power function of real numbers, we could apply τ_2 to the following two arguments: dependent variable (i.e., DS), and the result obtained from $\tau_1(UFP, MTS) = UFP + MTS$, which would result in:

$$\tau_2(DS, \tau_1(UFP, MTS)) = DS^{\tau_1(UFP, MTS)} = DS^{(UFP+MTS)} \quad (9)$$

Fourth illustrative case: Now we must select those combinations that have the best fit regarding the specific problem we want to solve. This selection is achieved from that infinite quantity of transformation combinations τ_i with elements from $2^{|V|}$, described in the third illustrative case. SMFC includes the application of the selected transformations to those specific values corresponding to the selected variables under a convenient order, such as a set of pairs of values is obtained. An SLR model is then applied to this data set of pairs. Since the training data set of the problem to be solved consists of N software enhancement projects:

$$(UFP^\mu, MTS^\mu, DS^\mu), \mu \in \{1, \dots, N\} \quad (10)$$

then each set of specific values corresponds to one of the projects. For instance, consider the first software project having as specific values the following ones: $DS^1 = 2.04$; $UFP^1 = 4.34$ and $MTS^1 = 1.10$. The resulting values are obtained after applying the transformations described in the third illustrative case as follows:

$$\tau_1(UFP^1, MTS^1) = \tau_1(4.34, 1.10) = 5.44 \quad (11)$$

and

$$\tau_2(DS^1, \tau_1(UFP^1, MTS^1)) = (DS^1)^{\tau_1(UFP^1, MTS^1)} = (2.04)^{5.44} = 48.35 \quad (12)$$

This procedure is then performed over each project in the rest of the training data set.

In accordance with the concepts described in the previous four illustrative cases, the SMFC general algorithm consists of applying all the transformations $\tau_1 \in T$ to those specific values of the selected elements $2^{|V|}$, as well as to the obtained values from some transformations in the proper order; that is, the transformation $\tau_k \in T$ will be applied either to an element $V_k \in 2^{|V|}$, or to any value $\arg_k$ obtained from either the application of one or more transformations $\{\tau_1, \tau_2, \dots, \tau_{k-1}\}$ or from:

$$\tau_k(V_k, \arg_k) \quad (13)$$

This procedure is iteratively performed for all k values.

So far, we have obtained the values for all the training data set in the transformation space. The fifth illustrative case will describe how the mentioned values are converted in a problem that allows one to apply an SLR model.

Fifth illustrative case: N real values are obtained once the application of all the transformations defined in the third illustrative case are performed to the N software enhancement projects included in the training data set of the fourth illustrative case. It means that a specific result:

$$r^\mu = (DS^\mu)^{(UFP^\mu + MTS^\mu)} \quad (14)$$

is obtained for each project μ , where $\mu \in \{1, \dots, N\}$.

As example, a value of:

$$48.35 = (2.04)^{(5.44)} = (2.04)^{(4.34+1.10)} \quad (15)$$

was obtained for the first software project.

The transformations τ_1 and τ_2 described in the third, fourth, and fifth illustrative cases were presented with an explanatory objective. The original model introduced in our study corresponds to a particular case of the SMFC general algorithm. SMFC is applied to the solution of the described problem, that is, to delivery speed prediction of software enhancement projects with *UFP* and *MTS* as the independent variables.

SMFC includes the following five transformations (two of them correspond to the “product of a real parameter by a variable” type):

- τ_1 : Product of the real parameter a by a variable;
- τ_2 : Product of the real parameter b by a variable;
- τ_3 : Arithmetic addition operation;
- τ_4 : Natural logarithm function ($\ln$);
- τ_5 : Arithmetic product operation.

These five transformations are applied to each $\mu \in \{1, \dots, N\}$ in the following order:

- τ_1 is applied to MTS^μ for obtaining $a \cdot MTS^\mu$;
 - τ_2 is applied to DS^μ for obtaining $b \cdot DS^\mu$;
 - τ_3 is applied to $a \cdot MTS^\mu$ and $b \cdot DS^\mu$ for obtaining $a \cdot MTS^\mu + b \cdot DS^\mu$;
 - τ_4 is applied to $a \cdot MTS^\mu + b \cdot DS^\mu$ result for obtaining $\ln(a \cdot MTS^\mu + b \cdot DS^\mu)$;
 - τ_5 is applied to both UFP^μ and $\ln(a \cdot MTS^\mu + b \cdot DS^\mu)$ for obtaining $UFP^\mu \cdot \ln(a \cdot MTS^\mu + b \cdot DS^\mu)$.
- Finally, the following r^μ result is obtained for each $\mu \in \{1, \dots, N\}$:

$$r^\mu = UFP^\mu \cdot \ln(a \cdot MTS^\mu + b \cdot DS^\mu) \quad (16)$$

4.2. Simple Linear Regression

One of the advantages of our proposal is that the application of the transformations leads to a two-dimensional model. This advantage is clearly reflected in the following fact: it is possible to apply an SLR model in order to fit the data to a line in the plane [35]. It contrasts with studies which implement multiple linear regressions due to the existence of multiple predictor variables.

The following step has high relevance for the SMFC: an independent variable is selected and its values are graphically represented on the X-axis of the transformation space, whereas the corresponding N values r^μ are graphically represented on the Y-axis. In our study, the N values UFP^μ correspond to those ones on the X-axis.

There is not a general rule to select any of the independent variables to be graphically represented on the X-axis from the transformation space, because of this, selection is one of the decisions to be taken into account for tuning the model parameters for the specific problem to be solved. This issue is similarly presented by tuning the parameters in SVM, neural networks or other models [32].

When an SLR model is applied to the following expression, the real parameters a and b are optimized by means of a metaheuristic technique:

$$r^\mu = UFP^\mu \cdot \ln(a \cdot MTS^\mu + b \cdot DS^\mu), \mu \in \{1, \dots, N\} \quad (17)$$

We emphasize the following feature as an SMFC relevant feature on other recent common models used for prediction in the software engineering field: on the X-axis is graphically represented those specific values of any of the independent variables; however, on the Y-axis of the SMFC transformation space occurs a special feature: the dependent variable values do not explicitly appear, but they are implicitly contained in the values represented on the Y-axis. Thus, the application of elemental algebraic operations allows for explicitly representing the DS value from r . In this final step, a DS predicted

value is generated for a software enhanced project contained in the testing data set. This procedure is described in the Section 4.4 of this article.

4.3. Metaheuristic Search

SMFC uses a metaheuristic optimization technique for finding the best parameters for the SLR model. The prediction problem is now seen as an optimization problem regarding the transformation parameters with the objective of obtaining the best results for the SLR model.

To tackle optimization problems, almost any search heuristic could have been used. However, metaheuristic approaches are able to approximate a solution close to the global optimum in a relatively brief time because of their ability to stave off stagnation at local minima by accepting worse solutions with a non-zero probability. The metaheuristic technique chosen for our proposal is simulated annealing, whose basic algorithm for a minimization problem is the following [36]:

1. Create a starting candidate solution randomly. Then, evaluate its worth as a measure of its energy E . The system is then heated up to a starting temperature T , which is usually a high value.
2. Move the search along by slightly modifying the starting candidate solution and evaluate its energy E_{new} . The process for generating the new solution is domain-specific, and the search will greatly depend on a proper neighbor generation method.
3. Calculate the probability that the modified solution is accepted:

$$P(E, E_{new}) = \begin{cases} 1, & E_{new} < E \\ e^{-\left(\frac{E_{new}-E}{T}\right)}, & E_{new} \geq E \end{cases} \quad (18)$$

that is, always accept a solution that descends down the gradient and therefore is better. Otherwise, the probability of the system accepting a solution worse than the current one depends on the temperature of the system. As the system lowers its temperature, this probability becomes smaller.

4. Cool down the temperature T according to the previously specified cooling schedule. This scheduling determines how soon the algorithm will likely stop accepting worse solutions. This, in turn, has the consequence of determining how soon it will start performing local rather than global exploration. Different cooling schedules have been proposed such as exponential and linear descents of temperature.
5. Check whether the algorithm needs to stop depending on the pre-defined stopping conditions. If this is not the case, a new iteration is necessary, and the algorithm returns to Step 2. Often used conditions include reaching a target fitness value or exceeding an established limit of iterations.

4.4. The SMFC Model

Let us start with a training data set of N software enhancement projects:

$$(UFP^\mu, MTS^\mu, DS^\mu), \mu \in \{1, \dots, N\} \quad (19)$$

Each training software enhancement project contains two independent variables (UFP^μ, MTS^μ) and a dependent variable DS^μ (i.e., UFP^μ and MTS^μ act as predictors for DS^μ).

Note that the graph of this problem lies in the three-dimensional space, because three variables are involved. To generate a graph, we should first graph the ordered pair (UFP^μ, MTS^μ) on an X–Y plane and then locate the value of DS^μ on the Z axis.

If we were to consider the values with which the fourth illustrative case was exemplified, we would have that the pair $(UFP^1, MTS^1) = (4.34, 1.10)$ would be plotted on the X–Y plane, and for that point the value $DS^1 = 2.04$ would be located on the Z axis.

One of the great advantages of models that belong to minimalist machine learning was emphasized in Section 3; with the new paradigm it is possible to reduce any problem of pattern classification to a graphical problem on the Cartesian plane.

Since the SMFC model belongs to the new paradigm because it is an adaptation to the regression task, the graph of the values of the variables of each project can already be expressed in the Cartesian plane with all the advantages that this brings. The application of expression 20 allows us to work in the Cartesian plane, no longer in three-dimensional space, as it is mandatory to do with the original data, without transforming.

The four algorithmic steps including the three SMFC parts described in the three previous subsections are described and exemplified next:

Step 1:

The five transformations described in Section 4.1 are applied to each software enhancement project μ with $\mu \in \{1, \dots, N\}$ being taken from the training set, such that the following variable transformation is obtained:

$$r^\mu = UFP^\mu \cdot \ln(a \cdot MTS^\mu + b \cdot DS^\mu), \mu \in \{1, \dots, N\} \quad (20)$$

where a and b are the transformation parameters, and r^μ is the resulting transformed variable.

Note that r^μ is a real value, and all original variables intervene in the creation of the transformed variable: both independent and dependent.

In expression (20), the value of r^μ for each project is obtained through a small number of elemental operations. Consequently, the processing of the N projects of the training set has a running time complexity $O(N)$.

This means that in Step 1 corresponding to the learning phase, the part of our proposal related to minimalist machine learning has linear complexity.

Step 2:

An SLR is applied to the variables using the UFP^μ values to be graphically represented on the X-axis and the values of r^μ to be graphically represented on the Y-axis. This is done to fit the values into a linear function.

After completing this step, it is now possible to graph the problem on the Cartesian plane.

Step 3:

An r^μ value implicitly having the DS_{pred}^μ predicted value is obtained by each pair of a and b parameter values, as well as to each software enhancement project μ with $\mu \in \{1, \dots, N\}$ taken from the training set. The term DS_{pred}^μ can be algebraically expressed in an explicit manner for the r^μ expression.

Since the software enhancement projects belong to the training set, to each project μ with $\mu \in \{1, \dots, N\}$ is previously known its correct value for DS^μ ; therefore, it is possible to calculate the absolute residual (AR) generated from the SLR to that μ :

$$AR^\mu = \left| DS_{pred}^\mu - DS^\mu \right| \quad (21)$$

In this article, absolute residuals are used as prediction criterion to evaluate the performance.

Now, simulated annealing is applied for finding the a and b parameter values minimizing the mean of the absolute residuals (MAR):

$$MAR = \frac{1}{N} \sum_{\mu=1}^N \left| DS_{pred}^\mu - DS^\mu \right| \quad (22)$$

After applying the previous three steps to the problem data, the SLR model has been generated having the a and b optimized parameters: a_{opt} and b_{opt} .

Step 4 is of great importance because it consists of applying the SLR model obtained with the three previous steps to the test patterns. For each project, with the application of Step 4, a value for delivery speed can be estimated. It is now possible to estimate the delivery speed for testing patterns.

Step 4:

Let t be the index of a software enhancement project belonging to the testing data set.

The UFP^t value is localized on the X-axis of the SLR optimized by the a_{opt} and b_{opt} parameters. Then, its corresponding r^t is obtained.

The DS_{pred}^t predicted value can be implicitly expressed from the r^t expression as follows:

$$r^t = UFP^t \cdot \ln(a_{opt} \cdot MTS^t + b_{opt} \cdot DS_{pred}^t) \quad (23)$$

Note that in Expression (23) all the values are known, except for DS_{pred}^t . By means of elementary algebraic operations, this value is obtained:

$$DS_{pred}^t = \frac{e^{\frac{r^t}{UFP^t}} - a_{opt} \cdot MTS^t}{b_{opt}} \quad (24)$$

In Expression (24), the value of DS_{pred}^t for each testing project is obtained through a small number of elemental operations. Consequently, the processing of each of the testing projects has a running time complexity $O(1)$.

This means that in Step 4 corresponding to the operation phase, the part of our proposal related to minimalist machine learning has constant complexity.

From the value obtained in (24) and the DS^t value that is known from the formulation of the problem, the absolute error is calculated for project t :

$$E^t = |DS_{pred}^t - DS^t| \quad (25)$$

Finally, with all E^t values, the performance of the SMFC model is estimated by calculating the mean of the absolute error:

$$\frac{1}{N} \sum_{\mu=1}^N |DS_{pred}^t - DS^t| \quad (26)$$

5. Data Sets Used for Training and Testing

The ISBSG Release May 2017 data set includes data from 8012 projects implemented between 1989 and 2016. ISBSG includes four types of development: new, enhancement, migration, and re-development. In our study, enhancement projects were selected.

The data sets of enhancement projects for our study were selected observing the “Guidelines for use of the ISBSG data”, that is, taking into account data quality, sizing method, development platform, and programming language generation [37].

In accordance with their data quality, sizing method, development platform, and programming language generation, data sets of enhancement projects were selected. The ISBSG reports the delivery speed as functional size units by elapsed month (i.e., UFP/month).

The UFP value is a composite value calculated from five independent variables (inputs, outputs, inquiries, internal files, and external files), whereas the number of participants is termed as max team size (MTS), which is defined by the ISBSG as “The maximum number of people during each component of the work breakdown who are simultaneously assigned to work full-time on the project at least one elapsed month” [3].

The counting for UFP consists of a process that involves two data functions (i.e., internal logical file, and external interface files), and three transactional functions (i.e., external inputs, external outputs, and external inquiries) [9]. Table 1 shows the number of projects by applying the two first mentioned criteria for this study. ISBSG classifies *data quality*, and *function point* rating quality from “A” to “D”.

Our study considered only those “A” and “B” software projects once they were suitable for statistical analysis. As for *functional sizing methods*, ISBSG reports several ways to measure functional

size such as COSMIC, Dreger, Feature Points, FiSMA, Fuzzy Logic, Gartner FFP, IFPUG, Lines of code, Mark II, and NESMA [3].

Since pre-IFPUG V4 projects should not be mixed with V4 and post V4, we only selected projects whose count approach were IFPUG V4+. NESMA was also considered once it could be mixed with IFPUG V4+. In total, 3521 of the 3986 projects of Table 1 were excluded for having empty values in any of their following fields: *development platform* (1719 projects), *max team size* (1255), *speed of delivery* (278), and *language type* (269).

With the goal of proposing a model for larger projects, of the remaining 465 projects, only those projects having a value higher than or equal to three for both *speed of delivery* and *max team size*, were only selected for our study (a total of 65 projects were excluded).

In accordance with the ISBSG, the development platform is classified based on the operating system used: personal computer (PC), mid-range (MR), mainframe (MF), or multiplatform (Multi); (2) programming language generation: second (2GL), third (3GL), fourth (4GL) generation, and application generator (ApG); (3) relative size measured in UFP, XS: 10 and <30, S: 30 and <100, M1: 100 and <300, M2: 300 and <1000, L: 1000 and <3000.

Table 1. Criteria for selecting software enhancement projects from the International Software Benchmarking Standards Group (ISBSG) data set.

Attribute	Selected Value(s)	Projects
Data quality rating	A, B	7533
Unadjusted function points rating	A, B	6184
Functional sizing methods	IFPUG V4+, NESMA	5322
Type of development	Enhancement	3986

Table 2 shows the 400 enhancement projects classified by (1) development platform. Since our objective is to propose a model that has a better generalization, those data sets that have less than 30 enhancement projects in Table 2 were excluded. Thus, seven data sets were used in our study for training and testing the models.

Table 2. Enhancement projects classified by relative size.

Development Platform	Programming Language Generation	Total by Relative Size					Total
		XS	S	M1	M2	L	
MF	2GL	—	1	2	—	—	3
	3GL	5	29	49	36	6	125
	4GL	—	2	3	1	1	7
	ApG	—	—	1	1	—	2
MR	3GL	—	10	16	9	—	35
	4GL	1	7	13	15	—	36
Multi	3GL	—	15	34	29	—	78
	4GL	—	2	14	13	—	29
PC	3GL	—	7	20	20	8	55
	4GL	—	6	11	11	2	30

A regression analysis for the seven data sets was performed. Scatter plots were generated by correlating DS and UFP, and DS and MTS. The fourteen obtained scatter plots showed skewness (they showed fewer large projects than small projects), heteroscedasticity (the variability of DS increased with either UFP or MTS), and outliers (they presented extremely large data values). Given that these three features were presented, each data set was normalized applying the natural logarithm (ln) [20].

The ANOVA *p*-values for the seven MLR equations were equal to 0.000, that is, there was a statistically significant relationship between the variables at the 99% confidence level for the seven equations showed in Table 3.

Table 3. Coefficient of determination r^2 by multiple linear regression (MLR) by data set (NPL: number of projects and PL).

Platform	NPL	MLR Equation	r^2
MF	125 - 3GL	$\ln(DS) = -0.484 + 0.73 \ln(UFP) - 0.028 \ln(MTS)$	0.6241
MR	35 - 3GL	$\ln(DS) = 0.003 + 0.67 \ln(UFP) - 0.131 \ln(MTS)$	0.5911
	36 - 4GL	$\ln(DS) = -0.879 + 0.84 \ln(UFP) + 0.008 \ln(MTS)$	0.7593
Multi	78 - 3GL	$\ln(DS) = -1.106 + 0.84 \ln(UFP) - 0.090 \ln(MTS)$	0.6040
	29 - 4GL	$\ln(DS) = -0.442 + 0.81 \ln(UFP) - 0.309 \ln(MTS)$	0.6890
PC	55 - 3GL	$\ln(DS) = -0.710 + 0.81 \ln(UFP) + 0.150 \ln(MTS)$	0.6257
	30 - 4GL	$\ln(DS) = -0.768 + 0.91 \ln(UFP) - 0.322 \ln(MTS)$	0.7631

Table 3 also shows the coefficient of determination (r^2) by MLR. This coefficient indicates the proportion of the variance in DS that is explained from the independent variables (i.e., UFP and MTS). In accordance with r^2 values of Table 3, the two independent variables explained the proportion of the variance in DS in more than 59% of the seven data sets.

Prior to displaying the results, some of the notation is summarized in Table 4.

Table 4. Table of notation.

SMFC	Search Method Based on Feature Construction	MAR	Mean of the Absolute Residuals
ISBSG	International Software Benchmarking Standards Group	PC	Personal computer
MLR	Multiple linear regression	MR	Mid-range
MLP	Multilayer perceptron	MF	Mainframe
SVR	Support vector regression	Multi	Multiplatform
UFP	Unadjusted function points	NI	Natural logarithm
MTS	Max team size	LOOCV	Leave-one-out cross-validation
DS	Delivery speed	SA	Standardized performance
XP	Extreme Programming	Δ	Effect size
NN	Neural network	NMLP	Number of neurons in MLP hidden layer
SLR	Simple linear regression	PM	Performance measure
AR	Absolute residuals	ID	Insufficient data

6. Experimental Results

A leave-one-out cross-validation (LOOCV) method was applied to train and test the MLR, MLP, the two types of SVR, FR, and the SMFC model, because it leaves out nondeterministic selection for training and testing sets, whereas the prediction performance for the models was calculated from absolute residuals (ARs) since ARs are an unbiased measure [9].

For each project i , with $i \in \{1, \dots, N\}$, the AR^i is obtained as follows:

$$AR^i = \left| DS_{pred}^i - DS^i \right| \quad (27)$$

where DS_{pred}^i is the predicted delivery speed and DS^i is the actual delivery speed for project i .

The mean (MAR) of the N software enhancement projects was obtained as follows:

$$MAR = \frac{1}{N} \sum_{i=1}^N AR^i \quad (28)$$

The median of all the AR^i is represented as MdAR. The performance of a prediction model is inversely proportional to the MAR and MdAR.

In addition, standardized performance (SA) and effect size (Δ) performance measures were used. The SA examines whether the prediction model generates predictions better than random guessing. The Δ examines that the predictions are not produced by chance. The value of Δ is recommended to be larger or equal to 0.5.

SA and Δ are calculated as follows [38]:

$$SA = 1 - \frac{MAR}{\overline{MAR_{P_0}}} \quad (29)$$

where $\overline{MAR_{P_0}}$ is the mean value of a large number, typically 1000, runs of random guessing.

$$\Delta = \frac{MAR - \overline{MAR_{P_0}}}{S_{P_0}} \quad (30)$$

where S_{P_0} is the sample standard deviation of the random guessing strategy.

As for the MLR model, a LOOCV was performed by data set from Table 3, and an MAR by data set was calculated. Thus, a total of 125, 35, 36, 78, 29, 55, and 30 MLR equations of type

$$\ln(DS) = a + b \cdot \ln(UFP) + c \cdot \ln(MTS) \quad (31)$$

were generated by each data set.

The number of neurons for the hidden layer in the MLP, as well as the kernel, was changed for each type of SVR until obtaining the best MAR. Table 5 contains the final values for MLP and SVR having the best prediction performance.

The math expression by SVR kernel of Table 5 is the following, where x, y are data patterns:

- Radial basis function: $K(x, y) = e^{(-\gamma|x-y|^2)}$, where the γ parameter controls the radial base function spread;
- Linear: $K(x, y) = xy$;
- Polynomial: $K(x, y) = (\gamma(xy) + c_0)^d$, where γ is a slope parameter, c_0 is a trade-off between major terms and minor terms of the generated polynomials, and d is the polynomial degree.

Table 5. Parameter values for MLP and support vector regression (SVR) having the best prediction performance (NMLP: number of neurons in MLP hidden layer).

Platform	NPL	NMLP	SVR Type	SVR Kernel	Values by SVR
MF	125-3GL	4	v - SVR	Radial basis function	$v = 0.5, \gamma = 0.0001$
MR	35-3GL	2	v - SVR	Linear	$v = 0.5$
	36-4GL	3	v - SVR	Linear	$v = 0.5$
Multi	78-3GL	3	ε - SVR	Polynomial	$\varepsilon = 0.001, \gamma = 0.001, c_0 = 0, d = 3$
	29-4GL	2	ε - SVR	Linear	$\varepsilon = 0.01$
PC	55-3GL	3	ε - SVR	Linear	$\varepsilon = 0.005$
	30-4GL	2	v - SVR	Polynomial	$v = 0.5, \gamma = 0.001, c_0 = 0, d = 3$

Regarding the training and testing of SMFC, the LOOCV process consisted of simulated annealing finding the transformation coefficients a and b in the expression:

$$r^\mu = UFP^\mu \cdot \ln(a \cdot MTS^\mu + b \cdot DS^\mu), \mu \in \{1, \dots, N\} \quad (32)$$

that minimize the training error for the corresponding iteration. Then, those same coefficients are used to predict the DS of the test project. The absolute difference between the predicted and actual DS of the project currently serving as test pattern is saved. This is repeated until all test patterns have acted as test inputs, and the mean of the errors is reported.

Table 6 shows the MAR and MdAR obtained when SMFC and the other four models were applied to the seven data sets included in Table 3.

Table 6. Prediction performance by model (PM: performance measure).

Platform	NPL	PM	SMFC	MLR	MLP	SVR	FR
MF	125-3GL	MAR	0.46	0.47	0.47	0.48	0.50
		MdAR	0.39	0.43	0.41	0.38	0.40
		SA	70.1	68.3	67.9	65.8	63.2
		Δ	0.89	0.82	0.84	0.85	0.79
MR	35-3GL	MAR	0.35	0.35	0.36	0.34	0.38
		MdAR	0.33	0.31	0.32	0.33	0.37
		SA	72.6	73.1	70.8	73.2	69.3
		Δ	0.82	0.85	0.79	0.89	0.77
	36-4GL	MAR	0.37	0.41	0.39	0.39	0.41
		MdAR	0.30	0.34	0.31	0.33	0.35
		SA	75.8	69.8	73.2	74.3	66.9
		Δ	0.91	0.86	0.88	0.90	0.82
Multi	78-3GL	MAR	0.36	0.38	0.38	0.35	0.39
		MdAR	0.29	0.30	0.29	0.26	0.30
		SA	78.9	74.3	75.8	79.5	72.4
		Δ	0.89	0.84	0.86	0.91	0.83
	29-4GL	MAR	0.28	0.30	0.30	0.27	0.30
		MdAR	0.16	0.19	0.19	0.17	0.22
		SA	81.2	78.9	77.5	82.3	79.5
		Δ	0.91	0.85	0.87	0.93	0.82
PC	55-3GL	MAR	0.51	0.53	0.54	0.48	0.63
		MdAR	0.38	0.41	0.44	0.30	0.55
		SA	83.2	79.3	78.4	85.3	75.3
		Δ	0.89	0.86	0.82	0.91	0.78
	30-4GL	MAR	0.32	0.38	0.35	0.30	0.47
		MdAR	0.29	0.31	0.24	0.23	0.38
		SA	92.1	87.6	86.3	93.3	83.5
		Δ	0.91	0.81	0.84	0.92	0.77

Since results should be reported based upon statistical significance [20], the selection of a statistical test to compare the prediction performance between SMFC and each of the other four models (MLR, MLP, SVR, and FR) was based on the number of data sets to be compared, data dependence, and data distribution.

Since the five models were applied to each enhancement project, data are dependent; therefore, an additional data set of differences calculated between each pair of data sets (ARs from the SMFC and ARs from the other models) was compiled.

If this additional data set resulted normally distributed after applying the Chi-squared χ^2 , Shapiro-Wilk, skewness, and kurtosis statistical tests, a parametric t -paired test (based on means) was applied.

Otherwise, a non-parametric Wilcoxon test (based on medians) was applied to statistically compare the performance between SMFC and each other model. Both Wilcoxon and t -paired are statistical tests used when two data sets are compared.

Table 7 contains the p -value by data set after applying the corresponding statistical test (the χ^2 was not performed in some data sets since it needs at least thirty data to be applied).

Table 8 contains the p -value by data set after applying the Wilcoxon or t -paired test between SMFC and models MLR, MLP, SVR, and FR.

Table 7. *p*-value by data set after applying the corresponding normality statistical test obtained from a data set of differences of absolute residuals (ARs) between search method based on feature construction (SMFC) and each model (ID: insufficient data).

Statistical Test	MLR	MLP	SVR	FR
Data set size: 125				
χ^2	0.0214	0.3103	0.0005	0.0000
Shapiro-Wilk	0.0006	0.3156	0.0000	0.0000
Skewness	0.0749	0.6418	0.0000	0.0154
Kurtosis	0.0006	0.0157	0.0000	0.0000
Data set size: 35				
χ^2	0.5206	0.2154	0.3799	0.0531
Shapiro-Wilk	0.5912	0.1492	0.2534	0.1654
Skewness	0.8687	0.8714	0.7919	0.8483
Kurtosis	0.5581	0.1672	0.7749	0.4951
Data set size: 36				
χ^2	0.9185	0.6015	0.0000	0.6756
Shapiro-Wilk	0.3039	0.6341	0.0000	0.9364
Skewness	0.2617	0.5755	0.8621	0.5257
Kurtosis	0.8626	0.0570	0.0000	0.3277
Data set size: 78				
χ^2	0.0033	0.0000	0.0000	0.8476
Shapiro-Wilk	0.0005	0.0000	0.0000	0.0739
Skewness	0.3483	0.7263	0.3758	0.0971
Kurtosis	0.6158	0.0000	0.0000	0.1312
Data set size: 29				
χ^2	ID	ID	ID	ID
Shapiro-Wilk	0.0937	0.8460	0.4901	0.1105
Skewness	0.1405	0.6296	0.9893	0.1941
Kurtosis	0.0139	0.7155	0.1363	0.6162
Data set size: 55				
χ^2	0.5153	0.5153	0.2547	0.0356
Shapiro-Wilk	0.1494	0.8524	0.1546	0.0844
Skewness	0.1268	0.7744	0.3164	0.2853
Kurtosis	0.2506	0.4994	0.0447	0.7346
Data set size: 30				
χ^2	0.5289	0.5289	0.4456	0.4456
Shapiro-Wilk	0.1258	0.3450	0.2603	0.1430
Skewness	0.4062	0.7738	0.5691	0.2735
Kurtosis	0.1635	0.0925	0.1425	0.9485

One of the most important results of this article that shows the superiority of the proposed model is that in accordance with the prediction performance values of Table 6 and *p*-values of Table 8, the SMFC resulted statistically better than MLR in four data sets at 95% of confidence (data sets containing 36, 78, 29, and 30 projects).

Furthermore, it is important to emphasize the fact that in all other cases, the methods with which our proposal is compared are not statistically better than SMFC. In those cases of Table 8 where there was statistical difference between SMFC and FR, it was always in favor of SMFC.

Table 8. Statistical comparison p -values based on Wilcoxon test or t -paired test, as appropriate, between SMFC and each of the other three models MLR, MLP, and SVR.

Data Set Size	Model			
	MLR	MLP	SVR	FR
125	0.5341	0.8512	0.7002	0.0351
35	0.9828	0.8057	0.6372	0.4648
36	0.0458	0.5374	0.1316	0.1468
78	0.0380	0.6765	0.0988	0.0473
29	0.0143	0.3081	0.6071	0.0645
55	0.3550	0.5619	0.9999	0.0076
30	0.0150	0.4703	0.4267	0.0050

7. Discussion, Conclusion, and Future Work

Although software development has evolved into a high-paced business tackling challenges of demanding contexts, such as those pertaining to the need for short development cycles and fast time-to-market [25], still several organizations show an unclear relationship between software business and software development [1].

Our article proposed a model named SMFC to predict a type of productivity in software organizations (i.e., software enhancement delivery speed). The SMFC prediction performance was compared to those of MLR, MLP, two types of SVR (i.e., ε -SVR and ν -SVR), and FR. They were trained and tested using seven data sets obtained from ISBSG, which is a repository widely used in the software prediction field [7,39]. These data sets were selected based upon their data quality, sizing method, development platform, and programming language generation as suggested in the guidelines of the ISBSG Release May 2017.

A large number of authors have taken on the task of estimating the computational complexity of the algorithms against which our proposal was compared in the experiments in Section 6. In this short discussion on complexity, we will assume that the training set consists of N projects, and that each project consists of p features, where for the particular case of the sets of projects included in this paper, the value of p is fixed at 2. In addition, for all the algorithms we have taken the corresponding state of the art data to the worst case run-time complexity.

Under these assumptions, the complexity of the SVR algorithm in its learning phase is $O(N^2p + N^3)$, while the prediction is made in $O(pn_{sv})$, where n_{sv} is the number of support vectors [40,41]. For this paper, the expressions for the complexity of the SVR algorithm are reduced to $O(2N^2 + N^3)$ and $O(2n_{sv})$, respectively.

The case of MLP is remarkable, because the complexity of the algorithm, both for learning and for prediction, depends on the topology of the neural network, in addition to the implementation. Some authors have estimated the complexity in the learning phase as $O(NpHeE)$, where H is the number of hidden neurons, E is the number of output values, and e is the number of epochs [42]. Depending on the problem and the number of layers in the network topology, the E and e values can be very large. In many cases of MLP applications, the magnitude of these values causes the execution time of the MLP learning phase to be really long. Consequently, sometimes this process takes weeks and even months to run, even though the computer equipment is high-performance [43]. Additionally, it is necessary to emphasize that the described situation corresponds to the best of the cases, when the network topology allows convergence; otherwise, the learning never ends because the network does not converge to some valid result.

Due to the fixed value of p , in this paper the expression for the learning complexity of the MLP algorithm is reduced to $O(2NHeE)$. Assuming that the convergence of the MLP was achieved, the prediction is made with an estimated complexity as $O(s_1s_2 + s_2s_3 + \dots + s_{k-1}s_k)$, where k is the number of layers, and s_j is the size of the j -th layer.

From the perspective of computational complexity, the MLR algorithm is more suitable than the SVR and MLP algorithms when the number of features is small [28]. Given that the complexity of the learning phase of the MLR algorithm is $O(p^2N^2 + p^3)$, for the experimental data of this paper this complexity is reduced to $O(4N^2 + 8)$, thus making the learning complexity of the MLR algorithm linear with respect to the number of projects N .

Let us now analyze what happens with the computational complexity of our proposal, the SMFC, considering that in the learning phase the algorithm consists of three parts (MML transformations, SLR, and simulated annealing), and that the prediction complexity for a pattern of test is constant.

Since there is only one feature in the SLR (an advantage that is achieved as a consequence of the MML transformations) the complexity of the SLR as part of the SMFC is less than the MLR. When taking into account that the complexity of the MML transformations is linear, as established when analyzing Expression (20) in Section 4.4, the complexity of these two parts is less than the MLR and the FR. Here, it is clear that the complexity of the first two parts of the SMFC is largely less than the complexities of both remaining models: SVR and MLP.

Regarding the third part of the SMFC, which consists of applying a metaheuristic search to optimize the parameters of the SLR model, the estimation of the complexity is not so straightforward. In this paper, we have selected simulated annealing to optimize the parameters of the SLR model [44]. However, the authors of the state of the art in metaheuristics agree that the complexity of this algorithm depends entirely on the problem to be solved. Therefore, it is not possible to speak of a generic expression for complexity.

The spectrum of types of computational complexity thrown up by the algorithms where simulated annealing is involved to solve problems is very broad. It is possible to easily verify that the complexity depends entirely on the area of application and the problem being tackled. In order to illustrate this great diversity, it is pertinent to mention that when an efficient version of the simulated annealing method has been applied to a variant of the bin-packing problem, the computational complexity of the method is linear on input size [45]. When applying simulated annealing to the problem of finding the maximum cardinality matching in a graph, it is shown for arbitrary graphs that a degenerate form of the basic annealing algorithm produces matchings with nearly maximum cardinality in polynomial average time [46]. Additionally, in the matter of computing the volume of a convex body in $\mathbb{R}^n$, when applying a variant of simulated annealing, the complexity is $O(n^4)$, where n is the dimension of the hyperspace where the convex body is immersed [47].

As regards to the problem that has been addressed in this paper, something is worth mentioning. When analyzing the execution times, a remarkable fact is clearly exhibited which is explained here. It turns out that when predicting the delivery speed of software enhancement projects, the complexity estimate is irrelevant, because in software engineering project sets are typically small. In the implementation that was designed to carry out all the experiments in Section 6, the MLP algorithm took less than two seconds, while the SVR algorithm took less than a second to complete both phases. Our proposal, SMFC, and regression algorithms (both MLR and FR) took less than half a second. In other words, the times are extremely small, and the differences are minimal, so in this case it is not productive to take into account the complexities of the algorithms.

After a statistical prediction performance comparison between SMFC, MLR, MLP, SVR, and FR was performed on the seven data sets (Tables 6 and 8), we can accept the following hypothesis derived from that formulated in the introduction for four of the seven data sets:

H1: The delivery speed prediction performance of software enhancement projects with SMFC is statistically better than the performance obtained with MLR when the UFP and number of practitioners are used as the independent variables.

We conclude that SMFC can be used for predicting the DS of the following types of software enhancement projects:

- Mid-range, and coded in 4GL;
- Multi-platform, and coded in 3GL;

- Multi-platform, and coded in 4GL;
- Personal computer, and coded in 4GL.

Since the SMFC resulted statistically equal than MLR in the resting three data sets, we can also suggest as alternative the use of SMFC for predicting the DS of the following types:

- Mainframe, and coded in 3GL;
- Mid-range, and coded in 3GL;
- Personal computer, and coded in 3GL.

The prediction performance of SMFC depends on the high prediction performance on its independent variable values: the UFP as well as the availability of that number of practitioners who would participate in the project, thus, it could be considered as an external validation threat of our study.

A relevant task that remains to be carried out in the immediate future is to use machine learning algorithms in projects similar to the one developed and presented here [48–51]. Future work will be related to the following issues to be applied to delivery speed prediction of software enhancement projects: (1) automatic design of transformation functions, (2) the application of alternative search methods to find the transformation coefficients, (3) intelligent selection of the independent variable graphically represented on the X-axis, (4) intelligent algorithm that selects the final prediction of an ensemble of simple linear models, and (5) the application of genetic programming for selecting the transformations to be applied to variables.

Author Contributions: Conceptualization, C.L.-M. and C.Y.-M.; methodology, E.V.-M., C.L.-M. and C.Y.-M.; software, E.V.-M. and I.L.-Y.; validation, E.V.-M. and C.L.-M.; formal analysis, C.Y.-M. and C.L.-M.; investigation, E.V.-M. and I.L.-Y.; writing—original draft preparation, C.L.-M.; writing—review and editing, C.Y.-M. and I.L.-Y.; visualization, E.V.-M. and I.L.-Y. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Acknowledgments: The authors want to thank the Instituto Politécnico Nacional of Mexico (Secretaría Académica, CIC, SIP and CIDETEC), the Universidad de Guadalajara, the CONACyT, and the SNI for their support to develop this work.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Guide to the Software Engineering Body of Knowledge, SWEBOK V3.0. Available online: <https://www.computer.org/education/bodies-of-knowledge/software-engineering/v3> (accessed on 11 March 2020).
2. Duarte, C.H.C. Productivity paradoxes revisited: Assessing the relationship between quality maturity levels and labor productivity in brazilian software companies. *Empir. Softw. Eng.* **2017**, *22*, 818–847. [CrossRef]
3. ISBSG Field Descriptions ISBSG D&E Repository Release May 2017. Available online: <http://isbsg.org/wp-content/uploads/2017/05/e.-ISBSG-Release-2017-R1-Field-Descriptions.pdf> (accessed on 11 March 2020).
4. Sheetz, S.D.; Henderson, D.; Wallace, L. Understanding developer and manager perceptions of function points and source lines of code. *J. Syst. Softw.* **2009**, *82*, 1540–1549. [CrossRef]
5. Petersen, K. Measuring and predicting software productivity: A systematic map and review. *Inf. Softw. Technol.* **2011**, *53*, 317–343. [CrossRef]
6. Paul, A.K.; Anantharaman, R.N. Impact of people management practices on organizational performance: Analysis of a causal model. *Int. J. Hum. Resour. Manag.* **2003**, *14*, 1246–1266. [CrossRef]
7. Gonzalez-Ladron-de-Guevara, F.; Fernandez-Diego, M.; Lokan, C. The usage of ISBSG data fields in software effort estimation: A systematic mapping study. *J. Syst. Softw.* **2016**, *113*, 188–215. [CrossRef]
8. ISBSG Glossary of Terms for Software Project Development and Enhancement. Available online: https://isbsg.org/wp-content/uploads/2016/10/ISBSG-Glossary_of_Terms-for-DE-and-MS.pdf (accessed on 11 March 2020).

9. Ferreira-Santiago, A.; López-Martín, C.; Yáñez-Márquez, C. Metaheuristic Optimization of Multivariate Adaptive Regression Splines for Predicting the Schedule of Software Projects. *Neural Comput. Appl.* **2016**, *27*, 2229–2240. [\[CrossRef\]](#)
10. Jiang, H.; Tang, K.; Petke, J.; Harman, M. Search Based Software Engineering (Guest Editorial). *IEEE Comput. Intell. Mag.* **2017**, *12*, 23–71. [\[CrossRef\]](#)
11. Chi, Z.; Xuan, J.; Ren, Z.; Xie, X.; Guo, H. Multi-Level Random Walk for Software Test Suite Reduction. *IEEE Comput. Intell. Mag.* **2017**, *12*, 24–33. [\[CrossRef\]](#)
12. Ferreira, T.N.; Lima, J.A.P.; Strickler, A.; Kuk, J.N.; Vergilio, S.R.; Pozo, A. Hyper-Heuristic Based Product Selection for Software Product Line Testing. *IEEE Comput. Intell. Mag.* **2017**, *12*, 34–45. [\[CrossRef\]](#)
13. Huang, H.; Liu, F.; Zhuo, X.; Hao, Z. Differential Evolution Based on Self-Adaptive Fitness Function for Automated Test Case Generation. *IEEE Comput. Intell. Mag.* **2017**, *12*, 46–55. [\[CrossRef\]](#)
14. Pitangueira, A.M.; Maciel, R.S.P.; Barros, M. Software requirements selection and prioritization using SBSE approaches: A systematic review and mapping of the literature. *J. Syst. Softw.* **2015**, *103*, 267–280. [\[CrossRef\]](#)
15. Paixao, M.; Souza, J. A robust optimization approach to the next release problem in the presence of uncertainties. *J. Syst. Softw.* **2015**, *103*, 281–295. [\[CrossRef\]](#)
16. Lopez-Herrejon, R.E.; Linsbauer, L.; Galindo, J.A.; Parejo, J.A.; Benavides, D.; Segura, S.; Egyed, A. An assessment of search-based techniques for reverse engineering feature models. *J. Syst. Softw.* **2015**, *103*, 353–369. [\[CrossRef\]](#)
17. Pascual, G.G.; Lopez-Herrejon, R.E.; Pinto, M.; Fuentes, L.; Egyed, A. Applying multiobjective evolutionary algorithms to dynamic software product lines for reconfiguring mobile applications. *J. Syst. Softw.* **2015**, *103*, 392–411. [\[CrossRef\]](#)
18. Smith, J.; Simons, C. The influence of search components and problem characteristics in early life cycle class modelling. *J. Syst. Softw.* **2015**, *103*, 440–451. [\[CrossRef\]](#)
19. Yáñez-Márquez, C. Toward the Bleaching of the Black Boxes: Minimalist Machine Learning. *IT Prof.* **2020**, *22*, 51–56. [\[CrossRef\]](#)
20. Kitchenham, B.; Mendes, E. Why comparative effort prediction studies may be invalid. In *ACM International Conference Proceeding Series (ICPS), Proceedings of the 5th International Conference on Predictor Models in Software Engineering, PROMISE'09, Vancouver, BC, Canada, 18–19 May 2009*; ACM International Conference Proceeding Series: Vancouver, BC, Canada, 2009.
21. Manoj Ray, D.; Samuel, P. Improving the Productivity in Global Software Development. In *Innovations in Bio-Inspired Computing and Applications, Proceedings of the 6th International Conference on Innovations in Bio-Inspired Computing and Applications IBICA 2015, Kochi, India, 16–18 December 2015*; Snášel, V., Abraham, A., Krömer, P., Pant, M., Muda, A., Eds.; Springer: Kochi, India, 2016.
22. Khomh, F.; Adams, B.; Dhaliwal, T.; Zou, Y. Understanding the impact of rapid releases on software quality. *Empir. Softw. Eng.* **2015**, *20*, 3336–3373. [\[CrossRef\]](#)
23. Alahyari, H.; Svensson, R.B.; Gorschek, T. A study of value in agile software development organizations. *J. Syst. Softw.* **2017**, *125*, 271–288. [\[CrossRef\]](#)
24. Pacheco, C.; Garcia, I.; Calvo-Manzano, J.A.; Arcilla, M. Reusing functional software requirements in small-sized software enterprises: A model oriented to the catalog of requirements. *Requir. Eng.* **2017**, *22*, 275–287. [\[CrossRef\]](#)
25. Mäkinen, S.; Leppänen, M.; Kilamo, T.; Mattila, A.L.; Laukkanen, E.; Pagels, M.; Männistö, T. Improving the delivery cycle: A multiple-case study of the toolchains in Finnish software intensive enterprises. *Inf. Softw. Technol.* **2016**, *80*, 175–194. [\[CrossRef\]](#)
26. Wen, J.; Li, S.; Lin, Z.; Hu, Y.; Huang, C. Systematic literature review of machine learning based software development effort estimation models. *Inf. Softw. Technol.* **2012**, *54*, 41–59. [\[CrossRef\]](#)
27. Gautam, S.S.; Singh, V. The state-of-the-art in software development effort estimation. *J. Softw. Evol. Process* **2018**, *30*, e1983. [\[CrossRef\]](#)
28. Bhavyashree, S.; Mishra, M.; Girisha, G.C. Fuzzy regression and multiple linear regression models for predicting mulberry leaf yield: A comparative study. *Int. J. Agric. Stat. Sci.* **2017**, *13*, 149–152.
29. Bigus, J.P. *Data Mining with Neural Networks: Solving Business Problems—From Application Development to Decision Support*, 1st ed.; McGraw-Hill: New York, NY, USA, 1996; pp. 61–94.
30. Cortes, C.; Vapnik, V. Support vector networks. *Mach. Learn.* **1995**, *20*, 273–297. [\[CrossRef\]](#)

31. Sewell, M.; Shawe-Taylor, J. Forecasting foreign exchange rates using kernel methods. *Expert Syst. Appl.* **2012**, *39*, 7652–7662. [\[CrossRef\]](#)
32. García-Florian, A.; López-Martín, C.; Yáñez-Márquez, C.; Abran, A. Support Vector Regression for Predicting Software Enhancement Effort. *Inf. Softw. Technol.* **2018**, *97*, 99–109. [\[CrossRef\]](#)
33. Adadi, A.; Berrada, M. Peeking inside the black-box: A survey on Explainable Artificial Intelligence (XAI). *IEEE Access* **2018**, *6*, 52138–52160. [\[CrossRef\]](#)
34. Paul, S.; Magdon-Ismael, M.; Drineas, P. Feature selection for linear SVM with provable guarantees. *Pattern Recognit.* **2016**, *60*, 205–214. [\[CrossRef\]](#)
35. Pan, G.; Zhou, Y.; Sun, H.; Guo, W. Linear observation based total least squares. *Surv. Rev.* **2015**, *47*, 18–27. [\[CrossRef\]](#)
36. Ezugwu, A.E.S.; Adewumi, A.O.; Frincu, M.E. Simulated annealing based symbiotic organisms search optimization algorithm for traveling salesman problem. *Expert Syst. Appl.* **2017**, *77*, 189–210. [\[CrossRef\]](#)
37. ISBSG Guidelines for Use of the ISBSG Data, Release May 2017. Available online: <https://isbsg.org> (accessed on 11 March 2020).
38. Shepperd, M.; MacDonell, S. Evaluating prediction systems in software project estimation. *Inf. Softw. Technol.* **2012**, *54*, 820–827. [\[CrossRef\]](#)
39. Fernández-Diego, M.; González-Ladrón-de-Guevara, F. Potential and limitations of the ISBSG dataset in enhancing software engineering research: A mapping review. *Inf. Softw. Technol.* **2014**, *56*, 527–544. [\[CrossRef\]](#)
40. Abdiansah, A.; Wardoyo, R. Time complexity analysis of support vector machines (SVM) in LibSVM. *Int. J. Comput. Appl.* **2015**, *128*, 28–34. [\[CrossRef\]](#)
41. Kaneda, Y.; Mineno, H. Sliding window-based support vector regression for predicting micrometeorological data. *Expert Syst. Appl.* **2016**, *59*, 217–225. [\[CrossRef\]](#)
42. Nicolas, P.R. *Scala for Machine Learning*, 2nd ed.; Packt Publishing Ltd.: Birmingham, UK, 2017; pp. 576–579.
43. Mizutani, E.; Dreyfus, S.E. On complexity analysis of supervised MLP-learning for algorithmic comparisons. In Proceedings of the International Joint Conference on Neural Networks, Washington, DC, USA, 15–19 July 2001; Kenneth, M., Paul, W., Eds.; IEEE: Washington, DC, USA, 2001; Cat. No. 01CH37222, Volume 1, pp. 347–352.
44. Kirkpatrick, S.; Gelatt, C.D.; Vecchi, M.P. Optimization by simulated annealing. *Science* **1983**, *220*, 671–680. [\[CrossRef\]](#)
45. Rao, R.L.; Iyengar, S.S. Bin-packing by simulated annealing. *Comput. Math. Appl.* **1994**, *27*, 71–82. [\[CrossRef\]](#)
46. Sasaki, G.H.; Hajek, B. The time complexity of maximum matching by simulated annealing. *J. ACM.* **1988**, *35*, 387–403. [\[CrossRef\]](#)
47. Lovász, L.; Vempala, S. Simulated annealing in convex bodies and an $O^*(n^4)$ volume algorithm. *J. Comput. Syst. Sci.* **2006**, *72*, 392–417. [\[CrossRef\]](#)
48. Huang, X.L.; Ma, X.; Hu, F. Machine learning and intelligent communications. *Mob. Netw. Appl.* **2018**, *23*, 68–70. [\[CrossRef\]](#)
49. Li, Z.; Chen, J.; Fu, Y.; Hu, G.; Pan, Z.; Zhang, L. Community detection based on regularized semi-nonnegative matrix tri-factorization in signed networks. *Mob. Netw. Appl.* **2018**, *23*, 71–79. [\[CrossRef\]](#)
50. Vamvakas, P.; Tsiropoulou, E.E.; Papavassiliou, S. Dynamic provider selection & power resource management in competitive wireless communication markets. *Mob. Netw. Appl.* **2018**, *23*, 86–99.
51. Huang, X.L.; Tang, X.; Huan, X.; Wang, P.; Wu, J. Improved KMV-cast with BM3D denoising. *Mob. Netw. Appl.* **2018**, *23*, 100–107. [\[CrossRef\]](#)

Publisher’s Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

© 2020 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<http://creativecommons.org/licenses/by/4.0/>).