Comprehensible Software Fault and Effort Prediction: a Data Mining Approach

Julie Moeyersoms[a,*], Enric Junqué de Fortuny[a], Karel Dejaeger[b], Bart Baesens[b], David Martens[a]

[a] *Faculty of Applied Economics, University of Antwerp, Prinsstraat 13, Antwerp B-2000, Belgium*

[b] *Department of Decision Sciences and Information Management, Katholieke Universiteit Leuven, Naamsestraat 69, Leuven B-3000, Belgium*

Abstract

Software fault and effort prediction are important tasks to minimize costs of a software project. In software effort prediction the aim is to forecast the effort needed to complete a software project, whereas software fault prediction tries to identify fault-prone modules. In this research both tasks are considered, thereby using different data mining techniques. The predictive models not only need to be accurate but also comprehensible, demanding that the user can understand the motivation behind the model's prediction. Unfortunately, to obtain predictive performance, comprehensibility is often sacrificed and vice versa. To overcome this problem, we extract trees from well performing Random Forests (RFs) and Support Vector Machines for regression (SVRs) making use of a rule extraction algorithm ALPA. This method builds trees (using C4.5 and REPTree) that mimic the black-box model (RF, SVR) as closely as possible. The proposed methodology is applied to publicly available datasets, complemented with new datasets that we have put together based on the Android repository. Surprisingly, the trees extracted from the black-box models by ALPA are not only comprehensible and explain how the black-box model makes (most of) its predictions, but are also more accurate than the trees obtained by working directly on the data.

[*] Corresponding author. Tel.: +32 3 265 42 05.
E-mail addresses: julie.moeyersoms@uantwerpen.be (J.Moeyersoms), enric.junquedefortuny@uantwerpen.be (E. Junqué de Fortuny), Karel.Dejaeger@kuleuven.be (K.Dejaeger), Bart.Baesens@kuleuven.be (B.Baesens), david.martens@uantwerpen.be (D. Martens)

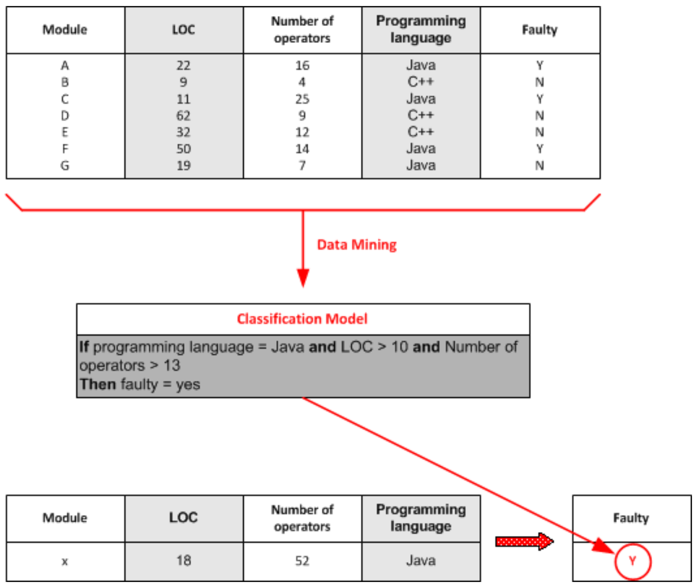
Keywords: Rule extraction, Software fault and effort prediction, Comprehensibility, Data mining

1. Introduction

The worldwide enterprise software development market was valued at $244 billion in 2010 according to information technology research and advisory firm Gartner, which proves the importance of this sector globally [1]. Yet, the software industry suffers from frequent cost overruns [2, 3, 4]. As a consequence this can lead to serious problems for software companies and sometimes even jeopardize their existence [5]. It is therefore important as a software development company to minimize costs as much as possible. In order to do so, activities such as software effort estimation and software fault prediction can be crucial, and constitute the topic of this paper.
Software effort estimation is the basis for project bidding, budgeting and planning [2]. Software fault prediction on the other hand aims to identify error prone software modules in a timely manner [6]. It is crucial to identify faults in the early stages of development since the cost of fixing or reworking software can be surprisingly high if they are detected in the later phases of the software development life cycle [7, 8, 9].
In this paper we predict software faults and effort, making use of different data mining techniques. Data mining entails the process of extracting knowledge from large amounts of data [10]. In the literature (see e.g. [11]) different types of data mining are discussed such as regression, classification and association rule mining. Regression and classification are predictive data mining tasks, where the target variable is continuous and discrete respectively. Association rule mining is a descriptive data mining task and aims at learning frequently occurring patterns [10]. The focus in this research lies on regression for software effort prediction and classification for software fault prediction. In both cases, statistical predictive models are built in order to generate predictions of new observations [12]. A simplified example for both prediction tasks is presented in Fig. 1 and 2, where it is shown that a classification or regression model is built based on historical data in order to generate accurate predictions of new observations. Data mining techniques are applied in many domains. Some well-known examples include credit scoring [13], churn prediction [14] and applications in the medical sector such as the selection of the best in-vitro fertilized embryo [15].

Although research on fault and effort prediction often emphasizes the predictive performance of a model, comprehensibility is an important aspect as well, demanding that the user can understand the rationale behind the model's prediction [16, 17]. Comprehensibility refers to how well humans grasp the classifier induced or how strong the mental fit of the classifier is [18, 19]. There are two main drivers to consider when talking about comprehensibility. A first important aspect is type of output: i.e. although the comprehensibility of a specific output type is largely domain-dependent, in general, rule-based classifiers can be considered as the most comprehensible, and non-linear classifiers as the least comprehensible [18]. A second driver for comprehensibility is the size of the output; i.e. smaller models are preferred [18]. Comprehensible models are often needed in order to inspire confidence in a business setting [7, 17] and improve model acceptance [14, 20]. Also, it facilitates the validation of results with domain knowledge, which is important because intuitiveness will determine whether or not the model will be accepted by the end-user [14]. Unfortunately, predictive performance and comprehensibility often work in a contradictory way and either must be sacrificed for the other [21, 22]. That is, any model that tries to achieve both predictive and explanatory power, will have to compromise somewhat [12]. The purpose of this research is to illuminate whether rule extraction can generate both accurate and meaningful rule-sets for software fault and effort prediction. Applying rule extraction in these specific domains can be particularly useful because previous research on fault and effort prediction shows that non-linear techniques typically give the best results for these problems. As non-linear techniques yield uninterpretable outputs, rule extraction can have a large added value in order to meet the need for comprehensibility in these domains.

Figure 1: Building a Classification Model with Data Mining



Module	LOC	Number of operators	Programming language	Faulty
A	22	16	Java	Y
B	9	4	C++	N
C	11	25	Java	Y
D	62	9	C++	N
E	32	12	C++	N
F	50	14	Java	Y
G	19	7	Java	N

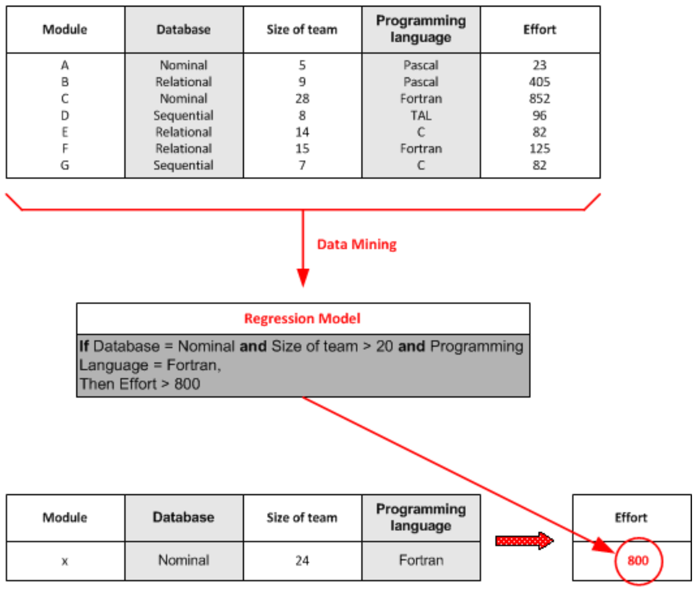
Classification Model

If programming language = Java **and** LOC > 10 **and** Number of operators > 13
Then faulty = yes

Module	LOC	Number of operators	Programming language
x	18	52	Java

Faulty
Y

Figure 2: Building a Regression Model with Data Mining



Module	Database	Size of team	Programming language	Effort
A	Nominal	5	Pascal	23
B	Relational	9	Pascal	405
C	Nominal	28	Fortran	852
D	Sequential	8	TAL	96
E	Relational	14	C	82
F	Relational	15	Fortran	125
G	Sequential	7	C	82

Regression Model

If Database = Nominal **and** Size of team > 20 **and** Programming Language = Fortran,
Then Effort > 800

Module	Database	Size of team	Programming language
x	Nominal	24	Fortran

Effort
800

The remainder of this paper is structured as follows: first, Section 2 describes the two considered prediction tasks in the field of software development. An overview of different data mining techniques applied in these settings is presented in Section 3. Section 4 argues for the application of rule extraction and explains ALPA, the rule extraction technique investigated in this study, in more detail. Next, Section 5 presents the setup of the experiments while Section 6 discusses the results of these experiments. In Section 7, some threats to validity and interesting topics for future research are set out. The main findings of this study are finally summarized in a short conclusion.

2. Software Development

2.1. Software Fault Prediction

Software companies are focusing on delivering quality to their customers. One of the main factors that determine the perception of the end-user of quality is the degree to which the software is free of bugs [23]. Software fault prediction aims to improve software quality by identifying fault prone modules in a timely manner by means of metric-based classification [6]. Early detection is also crucial as the cost of correcting or reworking software is much lower if faults are discovered in the early phases of the software development cycle [8].
In the literature, software fault prediction has been studied from different viewpoints; predicting the number of faults in each segment [24] or using the characteristics of different segments in the software code to identify which segments are most fault prone [25]. In the first one, software fault prediction is considered to be a regression problem while the second case regards it as a classification task [7]. This study considers software fault prediction from a classification point of view in which modules that are fault prone are identified based on their specific metrics. Various techniques have been investigated to this end, including artificial neural networks (ANN) [26], tree-based methods [27, 28, 29], swarm intelligence [10], SVMs [30], Random Forests (RFs) [6, 27] and statistical procedures like discriminant analysis [31] and logistic regression [32]. However, it is often stated that results regarding the superiority of one technique over another are difficult to assess across different studies due to different experimental setups [33]. Table 1 gives a non exhaustive overview concerning the use of different techniques for software fault prediction and in e.g. [7, 34, 35] a more detailed synopsis of the fault prediction landscape can be found.

It is argued that fault prediction techniques should not be based on predictive performance alone but that other aspects such as computational ef-

Table 1: Literature table summarizing related Software Fault Prediction research in recent years.

Author	**Reference**	**Techniques**	**Data source**	**Top performer**
Khoshgoftaar et al. (1997)	[26]	-ANN -Discriminant Analysis	Telco	ANN
Guo et al. (2004)	[27]	-RF -Discriminant Analysis -ROCKY detectors -LogReg -See5 classifiers -Several WEKA classifiers	NASA	RF
Menzies et al. (2007)	[25]	-C4.5 -Naive Bayes -1R	NASA	Naive Bayes
Arisholm et al.(2007)	[36]	-SVM -PART -C4.5 -LogReg -NN	Telco	C4.5
Elish and Elish (2008)	[30]	-SVM -LogReg -k-NN -Multi-layer perceptrons -RBF -Bayesian Belief Network -Naive Bayes -RF -Decision Trees	NASA	SVM
Lessmann et al.(2008)	[6]	-RF, LMT -LDA, QDA, Logit -Naive Bayes, BayesNet -LARS, k-NN, ANN, SVM -C4.5, CART, ADT -Multi-layer perceptron	NASA	RF
Dejaeger et al.(2011)	[7]	-NB -Augmented NB (ANB) -RF -LogReg	NASA ECLIPSE	RF

ficiency, ease of use and especially comprehensibility should also be taken into account [6, 7]. Comprehensible models help us to better understand software failures and provide improved causal insight, which can enable the development of novel predictors of fault-proneness and the development of better quality code.

2.2. Software Effort Prediction

In software effort prediction the aim is to predict the effort needed to complete a software project. For every new project, the software manager has to allocate people, time and money to the project [10] and unfortunately, this often proves to be a challenging task. It has been estimated by the 2004 CHAOS Report that 53 % of the projects turn out to be either more expensive than expected, late on delivery or missing required functionalities [37]. This highlights the need to properly estimate development efforts, and the non-trivial nature of this task [17, 38]. A large number of modeling techniques have been applied on software effort estimation data: Table 2 gives a non exhaustive overview concerning the use of different techniques for software effort prediction.

Table 2: Literature table summarizing related Software Effort Prediction research in recent years

Author	**Reference**	**Techniques**	**Data source**	**Top performer**
Shepperd and Schofield (1997)	[39]	-Case based reasoning -Regression Models	Telco & IT	Case based reasoning
Finnie et al. (1997)	[40]	-Case based reasoning -ANN -Regression models	ASMA	ANN + CBR
Briand et al. (1999)	[41]	-OLS Regression -Stepwise ANOVA -CART -Case based reasoning	Experience	No significant difference
Braga et al. (2007)	[42]	-M5P -SVR -MLP -Bagging predictors	NASA, Desharnais	Bagging with M5P
Park and Baek (2008)	[43]	-ANN -Expert judgment -Regression models	IT	ANN
Kumar et al. (2008)	[44]	-MLP -RBF networks -DENFIS -SVM -WNN	Finance, IBMDPS	WNN
Huang et al. (2008)	[45]	-CART -Grey Relational Anaysis -ANN -Case based reasoning	COCOMO, IBMDPS	GRA with genetic algorithm
Dejaeger et al. (2012)	[17]	-Tree/rule-based models -Linear models -Nonlinear models -Lazy learning approach	Coc81, USP05, Desharnais, Maxwell, Euroclear, Cocomonasa2, Experience, ISBSG, ESA	Log + OLS

Poor effort estimation can lead to a poor budgeting and planning which in turn may have serious consequences [5]. In this context, it is again important to have a predictive model that is both accurate and comprehensible in order to create confidence in a business environment.

3. Considered Classification and Regression Techniques

A myriad of different techniques is proposed in literature for classification and regression [11, 46, 47]. Non-linear techniques are mostly applied when the main purpose is to obtain a high predictive performance [48]. When the aim is to create an output that can easily be understood by the user, rule/tree-based techniques are in general considered to be the better option, generally at the expense of a diminished predictive power [49]. We apply non-linear techniques as well as tree-based techniques for both fault and effort prediction.
The non-linear technique of choice for software fault prediction is Random Forest; a choice motivated by previous research in this domain [6, 27]. Random Forest outputs a set of decision trees whose predictions are combined in an ensemble [50]. Since such a model is hard to understand, also C4.5 is used to induce a more comprehensible (single) tree model. For the experiments related to software effort prediction Support Vector Machine for Regression with a RBF-kernel is adopted since the Support Vector Machine is one of the state-of-art techniques [30, 48]. Second, we apply a regression tree learner in order to create an understandable model offering a clear interpretation.
Let us first consider the techniques covered in our study.

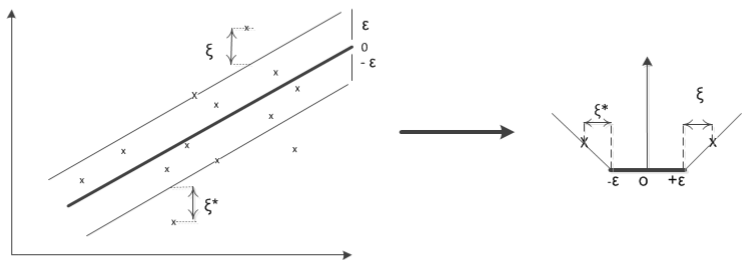
3.1. Support Vector Regression

The Support Vector Machine is a learning procedure based on statistical learning theory. SVMs have originally been developed to solve classification problems but can be extended to regression problems as well [51]. Hereto, an alternative loss function that includes a distance measure is introduced [52]. In this case a ε-insensitive loss function is used where,

$$|\xi|_\varepsilon = \begin{cases} 0 & \text{if } |\xi| \leq \varepsilon, \\ |\xi| - \varepsilon & \text{otherwise.} \end{cases} \quad (1)$$

This means that we do not accept any deviations that are larger than ε. Fig. 3 depicts the situation graphically [52]. In other words, the goal is to find a function that has at most ε deviation from the actual target values y_i for every point x_i in the training set (with n data points and m variables) [53].

Figure 3: SVR using ε-insensitive loss function



The optimization problem can be written as [52],

$$\begin{aligned} \text{minimize} \quad & 1/2||\omega||^2 + C\sum_{i=1}^{n}(\xi_i + \xi_i^*) \\ \text{subject to} \quad & y_i - \langle \omega, x_i \rangle - b \leq \varepsilon + \xi_i, \\ & \langle \omega, x_i \rangle + b - y_i \leq \varepsilon + \xi_i^*, \\ & \xi_i, \xi_i^* \geq 0 \end{aligned} \tag{2}$$

with ω the weight vector in the feature space, b a threshold value, ξ_i and ξ_i^* slack variables measuring the deviation from the boundaries of the ε-insensitive zone, and C the regularization constant. The optimization criterion of equation (2) penalizes training data points where the distance between y and the fitted function $f(x)$ is larger than ε.
A non-linear model is often needed to adequately model data. This non-linear function is approximated by mapping the input data into a dual feature space, induced by a so called kernel function. This study opts to use the RBF-kernel [53],

$$K(\mathbf{x}, \mathbf{x}_i) = e^{\frac{-\|\mathbf{x}-\mathbf{x}_i\|_2^2}{\sigma^2}} \tag{3}$$

with σ the kernel bandwidth hyperparameter, which is tuned on a validation set.

Finally, the general form SVR function can be written as follows with α and α^* the Lagrangians [52] ,

$$f(x, \omega) = \sum_{i=1}^{n} (\alpha_i - \alpha_i^*) K(x, x_i) + b. \tag{4}$$

For more details on SVR, we refer to [52, 53].

3.2. Random Forest

Random Forest (RF) is an ensemble method constructing a collection of univariate tree classifiers [47, 50].
The first step is to take L samples of size n at random from the original data using bootstrap sampling. Next, from every sample, a tree is built by splitting on $k(\leq m)$ input variables randomly selected from the total amount of input variables in the original data. The number of selected input variables is a hyperparameter of the learner and stays constant during the forest growing process [50]. After inducing a large number of trees, a majority voting procedure is performed to decide on the final class.
Although Random Forest builds upon multiple decision trees which are comprehensible, the ensemble loses this advantage and creates an output that is hard to interpret (a set of hundreds of trees).

3.3. C4.5

C4.5 is a popular tree induction technique based on information theoretic concepts. More specifically, it uses entropy to measure how informative an attribute is in splitting the data [54]. Entropy quantifies the order (or disorder) amongst observations with respect to the classes. If we consider p_1 (p_0) to be the proportion of examples of class 1 (0) in sample S, we can state that the entropy equals 1 when $p_1 = p_0 = 0.5$ (maximal disorder, minimal order) and 0 (maximal order, minimal disorder) when all observations belong to the same class, $p_1 = 0$ or $p_0 = 0$. In order to decide upon which attribute to split at a given node, the gain criterion is used [54]. Gain is defined as the expected reduction in entropy due to splitting on attribute x_j. C4.5 uses a gainratio criterion and applies normalization in order to avoid that attributes with many distinct values will be favored [54]. Decision trees are popular due to their simplicity and transparency. They are self-explanatory and easy to interpret if the size is small enough [55, 56].

3.4. Regression Tree

Regression Trees induce a tree by splitting the input data in cuboid regions. The splits are chosen to optimize some well-chosen criterion such that each one of the leaves represents one of those regions and each of them uses a simple model [57].
The REPTree algorithm which optimizes information gain/variance reduction over the remaining data is employed in this study [11]. After tree learning, the tree depth is reduced by using reduced-error pruning (with backfitting) in order to improve generalization.
The output of this tree is considered to be comprehensible (if the depth of the tree is small enough) since the relation between input and output is immediately visible.

4. Rule Extraction

4.1. Rationale behind Rule Extraction

It has been argued that models which are both accurate and comprehensible are to be preferred in the domain of software engineering [6, 7]. The problem is that by using traditional techniques these two requirements often collide [16]. For example, complex, non-linear techniques often perform very well in terms of predictive performance but mostly yield uninterpretable models [21]. Rule induction techniques on the other hand construct very comprehensible results but have the disadvantage of reduced predictive power [58]. Rule extraction is a technique that will compromise between these two requirements by building a simple rule set that mimics the performance of the complex model [20, 58, 59].
There are two reasons why rule-extraction is often used [59]. Firstly to gain better insight into how complex models make their decisions. That is, we want to know whether results are logical and justifiable before implementing them in practice [49]. By building a set of rules mimicking the performance of a complex model, we are able to better comprehend the inner workings of opaque models. The extent to which the complex model and the rule-based model agree on their predictions is measured by the fidelity [58, 59]. This is an important measure since it can be used as an indicator of how similar the extracted rules and the complex model are. If the rules and fidelity are satisfactory, one may decide that the complex model is sufficiently explained and can be used as a decision support model [59]. A second reason for using

rule-extraction is because we want to improve the performance of rule induction techniques [59].
The rule extraction technique that was used in this paper is ALPA, given its superior performance and applicability to both classification [58] and regression [60].

4.2. ALPA

In order to improve a rule set in terms of either predictive power or fidelity, we can use one of the previously trained rule induction techniques (the *white-boxes*) to imitate the output of the more complex model (the *black-box*) that performs better. The way in which this imitation is realized differs between different rule extraction methods, but is crucial to the performance of the techniques. There are three key insights being exploited in ALPA (for more details we refer to [58, 60]).

First, by presenting the *predicted* target values of the training set to the white-box algorithm instead of the original target values associated with the training set, we can improve the similarity between the black-box and the white-box substantially. By doing so, the black-box effectively becomes an oracle for predictions.

Second, since the oracle is only dependent on the black-box, we do not have to sample any new target labels, and thus, we are free to generate new artificial data points and their predictions without restrictions. This is the *active learning* component of the algorithm: at any given point, the algorithm can choose whichever input data vector it wants to get a label for.

Third, by choosing the right (artificial) data vectors, further improvements of fidelity can be achieved as we generate incrementally more data. The data *generation region* differs for classification and regression respectively.

4.2.1. Classification ALPA (ALPAC)

In classification, a model typically induces some kind of decision boundary. This decision boundary marks the transition from one class to another. For classification, this region has been found to be the best region when optimizing fidelity [58]. For Random Forests the exact parametric function of this boundary region is unknown, but this can be circumvented by the generation of data in this region using a proxy based on the so called uncertainty function for Random Forests, defined as:

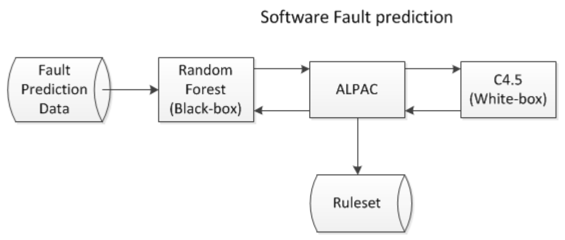
$$\pi(C_i|\mathbf{x}) \;=\; \operatorname*{avg}_k \mathrm{I}(\mathrm{h}_k(\mathbf{x}) = y) - \max_{j \neq y} \operatorname*{avg}_k \mathrm{I}(\mathrm{h}_k(\mathbf{x} = j)),$$

where $\mathbf{x}$ is the input vector, $I()$ the indicator function and $h()$ the previously mentioned trees from the Random Forest model. Knowing this uncertainty function, we can select a few points that are very uncertain (i.e. are located next to the decision boundary) and generate data on the convex combination of these points. In this way, the artificially generated data points are positioned in a relevant region. Note that the convex combination $\mathbf{r}$ between two input vectors is defined as:

$$\mathbf{r} \;=\; \theta\,\mathbf{x}_i + (1-\theta)\,\mathbf{x}_j, \quad \theta \in [0,1]. \tag{5}$$

All of the $\mathbf{r}$ (one for each choice of θ) are located on the line connecting both input vectors in the input space. The operational set-up for ALPA when discussing software fault prediction is shown in Fig. 4.

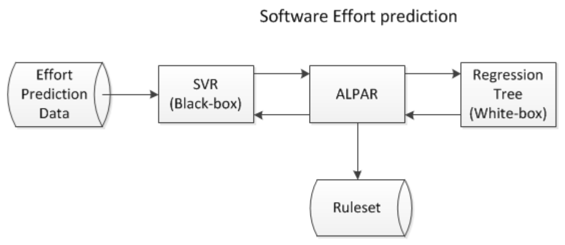
Figure 4: Operational set-up of the rule-extraction technique for software fault prediction.



4.2.2. Regression ALPA (ALPAR)

In regression, the interesting region is that which covers the regression target function. In order to generate data around this region, one wants to prevent generating data near outliers. Unfortunately, we do not know the exact target function (otherwise there would be no point in applying a regression procedure). It can however be approximated by using the same principles as in Section 4.2.1, with the difference that this time, we want to generate data located next to the most *confidently* predicted vectors. By definition, these are sure to appear near the target function (and as an added advantage, being an almost uniform sampling, simulate the underlying distribution well). By using the convex combination as before, additional data points can be generated in the correct region. This leads to the following operational set-up for ALPA for software effort prediction, shown in Fig. 5.

Figure 5: Operational set-up of the rule-extraction technique for software effort prediction.



5. Methodology

5.1. Data Sets

5.1.1. Software Fault Prediction

Software fault prediction data stemming from the NASA MDP and the PROMISE repository [61, 62] (KC1, PC3 and PC4) as well as four new Android data sets collected from the Android repository are investigated in this study. These data sets contain measurements of static code features on a file level granularity as earlier research pointed out that such attributes are useful, easy to use and widely used in the context of software fault prediction [7, 25]. As such, the set of features includes line counts, McCabe and Halstead metrics. Table 3 provides an overview of the number of observations and attributes included in the data sets. For more detailed information about the data sets we refer to [6, 7] and other studies listed in Section 2.1.

Data collection.
The NASA MDP data have previously been used in the domain of software fault prediction and have been discussed in e.g. [6, 7, 34]. As was noticed by Hall et al [35], the NASA data sets are widely used and offer unique advantages in terms of the reproducibility of studies. The Android data have been collected in the context of this study and relate to different releases of the well known open source Android platform. This platform is targeted towards mobile devices such as smartphones and tablet computers and is based on

Table 3: Characteristics of the software fault prediction data sets

Data sets	Android _V2.0	Android _V2.2	Android _V2.3	Android _V4.0	KC1	PC3	PC4
Nr of code attributes	27	27	27	27	21	37	37
Nr of modules	4,388	5,622	4,883	5,753	1,571	1,511	1,347
Nr of fp modules	512	956	1,302	350	319	160	178
% of fp modules	11.7	17	26.7	6.1	20.3	10.6	13.2

the Linux kernel while also drawing upon other open source projects such as squeak (a bluetooth package) and yaffs (Yet Another Flash File System). Evidently, the source code of the Android platform is made available to the public, licensed under the Apache License v2, except for the Linux kernel and its modifications, which are published under the GNU Public License v2. Remark that a 6 month development life cycle is in place for major releases, each receiving a specific nickname, which are followed by several minor releases. Fig. 6 provides a chronological overview on the development of the Android platform.

Considering this 6 month development cycle, data on 4 major releases was collected and matched with 6 months of post release defect data. This matching was done by using regular expression comparison on the commit messages entered into the version management system storing the Android source code. As commit messages were not stored before an adjustment in the version management system, the chronological overview indicates that the first release on which sufficient data is available is Eclair, and also 3 more recent releases were considered; i.e. Froyo, Gingerbread and Icecream sandwich. The source code of all projects listed in Table 4 was downloaded for each release, totalling ~1.3 GB of source code, documentation and support files. A set of static code features was derived from the Java source files, except for those related to testing procedures. An open source tool called ‘Perst’ was adopted hereto, which makes use of a code parser generated by JavaCC [63]. The most recent version was obtained and modified to extend the set of features which can be mined by this tool.

As a similar attribute set is mined compared to the well known NASA data sets, we believe our study (and data sets) can make a valuable addition to the current research landscape. By aligning ourselves with the attribute

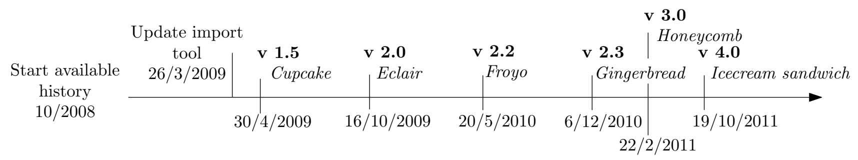


Figure 6: Android platform chronological overview

space from NASA data sets, and positioning our data sets (extracted from an open source project) in the public domain [1], we also answer to the call raised by Hall et al [35].

Project	**Description**
Dalvik	Dalvik is a virtual machine which allows to compile apps to byte code and executing them in this virtual machine; is based on the now defunct Harmony Apache project. Contains both Java and C source code files.
Frameworks base	A collection of smaller parts which has been developed or adapted to the Android project such as telephony and camera services. Contains both Java and C source code files.
Libcore	A library of files which are frequently used by other parts of the Android platform; was initially part of the Dalvik subproject until 30/04/2010. These files have mainly been written in Java.
SDK	The software development kit (SDK) which is offered to Android developers to assist in creating applications for the Android platform.

Table 4: Description of Android sub projects

5.1.2. Software Effort Prediction

For software effort prediction, we used 4 publicly available data sets stemming from PROMISE repository (Coc81, Cocomonasa2, Desharnais, Maxwell) [62] as well as a data set collected from the European Space Agency [64]. Each data set contains a different set of attributes, which can be grouped into the following categories [17]:

[1]http://www.applieddatamining.com/cms/?q=software

- Size: attributes relating to the size of the software project such as LOC counts or function points (FP).

- Environment: attributes containing information regarding the development team, the project itself and the sector of the developing company.

- Project: attributes pertaining to project type and purpose.

- Development: attributes containing information about the technical or managerial aspects of the project such as the programming language or the type of database system.

Table 5 provides an overview on the characteristics of the different data sets while Table 6 shows the number of observations and attributes included. Further information on these data sets can be found in e.g. [17] and other studies in Section 2.2.

Table 5: Characteristics of software effort prediction data sets

Data set	**Single/multi company**	**Application domains**	**Size measure**	**Year collection**
ESA	M	Space/military	Kloc	1983-1996
Coc81	S	Engineering, science, finance, etc	Kloc	1970-1981
Cocomonasa2	M	Space/military	Kloc	1981-1999
Desharnais	S	Unknown	FP	1981-1988
Maxwell	S	Finance	FP	1985-1993

Table 6: Observations and attributes included in the software effort prediction data sets

Data set	**Observations**	**Attributes**	**Nominal**	**Numeric**
ESA	171	16	X	X
Coc81	81	14		X
Cocomonasa2	93	24	X	X
Desharnais	81	12	X	X
Maxwell	62	27	X	X

5.2. Experimental Setup

The ALPA methodology has been applied on the different data sets with the set-up shown in Fig. 4 and 5. For all experiments the Weka workbench

was used [11].
As a first important step we preprocess the data sets and select the data to learn and validate the model. For data sets concerning fault prediction, the ID as well as attributes with zero variance are discarded. In case of effort prediction, only attributes known at the moment of effort estimation are retained; attributes such as duration and costs are not known at the time of estimation and are discarded. Furthermore, in case of missing values, an attribute is removed if more than 25 percent of its values are missing. Otherwise, for continuous attributes, the missing value are replaced with the median value while categorical variables are transformed into binary variables by using dummy encoding, including a missing flag if appropriate.
After preprocessing the data, input selection was performed using the CfsSubsetEval method in WEKA [11]. This evaluator considers the predictive value of each attribute individually along with the degree of redundancy with them [11]. Input selection can be considered as an important step in the model building phase since including irrelevant and noisy attributes can result in poor predictive performance and high computational effort [65].
A 10-fold cross testing is used to randomly split the data into training and test data. Since both the SVR and Random Forest have adjustable parameters, also referred to as hyperparameters, we first tune them in order to adapt the algorithms to the underlying data characteristics. For SVR the complexity parameter C and σ parameter of the RBF Kernel are tuned. In case of the RF two hyperparameters were considered, namely, the number of trees (L) and the number of attributes used to build each individual tree (k). Each time a grid-search procedure on a validation set is adopted in order to tune the hyperparameters. That is, a set of values is defined for each hyperparameter and possible combinations are evaluated using a 2-fold cross validation on the training data, which will split this training data into a learning and validation set. Each time, the combination of hyperparameters leading to the highest AUC value is chosen. For RF we considered a range of [10,50,100,250,500,1000] trees and three different values for parameter k, namely $[0.5, 1, 2].\sqrt(m)$ (see [6]). The C and σ parameter of the SVR model were chosen as follows: $C \in [5 : 10]$ and $\sigma \in [-3 : 2]$.
The predictive performance of the models are evaluated using accuracy and recall [66, 67] for classification and RMSE (Root Mean Square Error) for regression. Accuracy is defined as the number of test data points correctly identified by the rules, divided by the total number of data points in the test set. Recall on the other hand, measures the proportion of positive cases that

are correctly classified as positive. In case of regression we use the RMSE as compared to the true output values in order to measure the performance of the models. Another measure that was used to evaluate the models is fidelity. As explained in Section 4.1, fidelity can be described as the extent to which the rule set obtained from the white-box technique is similar to the complex model [58, 59]. For classification this can be defined as the number of data points where the rule set and the complex model agree, divided by the total number of data points in the test set. In case of regression, fidelity is expressed as the RMSD (Root Mean Square Deviation) of the predictions of the white-box on the one hand and the black-box on the other hand.
Since we apply rule extraction, it is required that the black-box model outperforms the white-box model. That is, we would prefer to use the white-box model if the performance would be even better than that of the black-box model. Consequently, folds for which this requirement is not met were discarded from the results. This procedure is a common practice in the rule extraction literature [58, 59, 60].
Finally, since in this paper we aim for comprehensibility, the rule sets created by C4.5 were pruned making use of the pruning parameters. The maximum depth of the Regression Tree was set to 5. By doing this, the final model is ensured to be comprehensible and easy to interpret.

6. Results

Table 7 shows the detailed results for the experiments on software fault prediction. The performances of the original trees and the ALPAC-trees are shown, where each time the size of the tree (number of leaves) is indicated in the last column. The results of the Random Forest are added in the last column of the table. As mentioned before, accuracy and recall are used to measure the predictive performance of the models. The fidelity measures how often C4.5 and Random Forest agree and so it is used as an indicator of how similar the extracted rules and the complex model are. In other words, the higher the fidelity, the better the rules explain the complex model. The results for fidelity, accuracy and recall represent the average over all 10 fold tests except for the folds where Random Forest performs worse than C4.5. That is, for fidelity and accuracy the folds for which the C4.5 performed better in terms of accuracy were deleted from the results, whereas for recall we only included the folds for which the black-box model performed better in terms of recall. This is a common practice in the rule extraction literature [58, 59, 60] as it would be easier to just use the white-box model if this would give better results. Obviously, this implies that Random Forest performs better than the original tree for all datasets in terms of accuracy and recall. As compared to the ALPAC-tree, Random Forest again performs better for all datasets, which is logical since rules (that have not the same modelling capabilities as Random Forests) are extracted from this black-box, thereby compromising slightly on accuracy. When comparing the new ALPAC tree with the original tree, it can be seen that ALPAC performs better in terms of fidelity as compared to the baseline algorithm C4.5 for 6 out of 7 data sets. In terms of accuracy ALPAC performs better than C4.5 in 5 out of 7 cases. Also for recall, ALPAC is performing better for 6 datasets, yet recall is rather low. This can be explained by the fact that ALPAC optimizes for accuracy instead of recall. If the main goal is to achieve a high recall, ALPAC could be adapted so that the model would be optimized for recall. In our case, however, the focus is not on which performance measure should be used but on the potential of ALPAC to create comprehensible rule sets out of a complex algorithm. The trees are pruned so that the trees produced by C4.5 have approximately the same size than the ones produced by ALPAC. Note that it is very hard to arrive at exactly the same size of the trees due to the nature of the techniques, but a comparison of the median size of the trees shows that the models have approximately similar rule complexity. This allows us to compare the results produced by the models. Figure 7 shows an example of a tree that was built. The tree is small enough to be understood by the user and thereby obtains insight in how the black-box model makes most of its predictions. As such, we effectively combine the good generalization behavior of the RF model with the comprehensibility of the C4.5 tree.

The results of the experiments for software effort prediction are displayed in Table 8. The results of the trees and ALPAR-trees are shown in the first columns and the performances of the SVR are displayed in the last column. For the trees, the size is not mentioned because we used a maximum depth of 5 for both regression tree and ALPAR, which meets our aim to create trees that are small enough to be comprehensible. For this regression task, the fidelity of the white-box regression model to the black-box regression model can be described as the RMSD of their predictions. In other words, the lower the fidelity the better the white-box is able to explain the black-box model. As with ALPAC, the results represent the average over all 10 fold

Table 7: Results C4.5 compared to ALPAC-trees in terms of Fidelity, Accuracy, Recall and Size

	$C4.5_{original}$				$C4.5_{ALPAC}$				RF	
Data set	Fidelity (%)	Accuracy (%)	Recall (%)	Size	Fidelity (%)	Accuracy (%)	Recall (%)	Size	Accuracy (%)	Recall (%)
$KC1$	87.44	84.40	13.19	7	88.02	84.61	17.23	6	86.50	31.61
$PC3$	89.55	89.12	3.75	6	89.47	88.96	10.00	8	90.96	22.50
$PC4$	87.44	89.60	24.08	7	89.12	89.71	27.45	6	91.89	54.51
$Android\_V2\_0$	88.37	88.56	20.45	4	89.03	89.11	22.38	6	89.20	27.24
$Android\_V2\_2$	82.73	83.94	16.78	8	83.20	83.48	13.19	7	85.34	30.33
$Android\_V2\_3$	75.17	75.93	18.52	5	75.56	76.38	24.66	6	78.20	44.70
$Android\_V4\_0$	94.20	93.94	7.14	4	94.63	93.98	16.47	6	94.33	19.64
$Wins$	1	2	1		6	5	6			

tests except for the folds where the regression tree performs better than the SVR. Therefore, it is logical that the SVR performs better than the original trees for all datasets, as can be seen from Table 8. When comparing the performances of the ALPAR-trees with those of the SVR, it can be noticed that although they are lower for most of the datasets, they are still very close to the SVR accuracy. A comparison of the original trees with the ALPAR-trees shows that ALPAR performs better than the regression tree both in terms of accuracy and fidelity for all datasets. Figure 8 shows one of the regression trees that was produced by ALPAR. This tree is interpreted as follows: If the time constraint for the CPU is extra high, the predicted effort will be very high as well. Else, a large equivalent physical KLOC will lead to a higher predicted effort whereas if the equivalent physical KLOC is smaller, the predicted effort depends on the NASA-center concerned. That is, the NASA center number 2 appears to be more efficient. Again, since the relation between input and output is easy to understand from this tree, we can state that the tree is comprehensible. Once more, the user thereby obtains insight into the well performing black-box SVR model.

Table 8: Results REPTree versus ALPAR-trees in terms of Fidelity and Accuracy

Data set	$REPTree_{original}$ Fidelity (RMSD)	$REPTree_{original}$ Accuracy (RMSE)	$REPTree_{ALPAR}$ Fidelity (RMSD)	$REPTree_{ALPAR}$ Accuracy (RMSE)	SVR Accuracy (RMSE)
$Coc81$	309.79	1397.78	9.78	330.91	335.12
$Cocomonasa2$	1174.19	817.58	5.33	459.10	449.10
$Desharnais$	2623.88	2231.22	14.30	1706.71	1691.25
$Maxwell$	2432.04	9412.59	1.24	6383.10	6253.47
ESA	112.96	164.90	14.49	153.61	153.76
$Wins$	0	0	5	5	

Figure 7: Example of ALPAC-tree for fault prediction using the Android_V4_0 data set

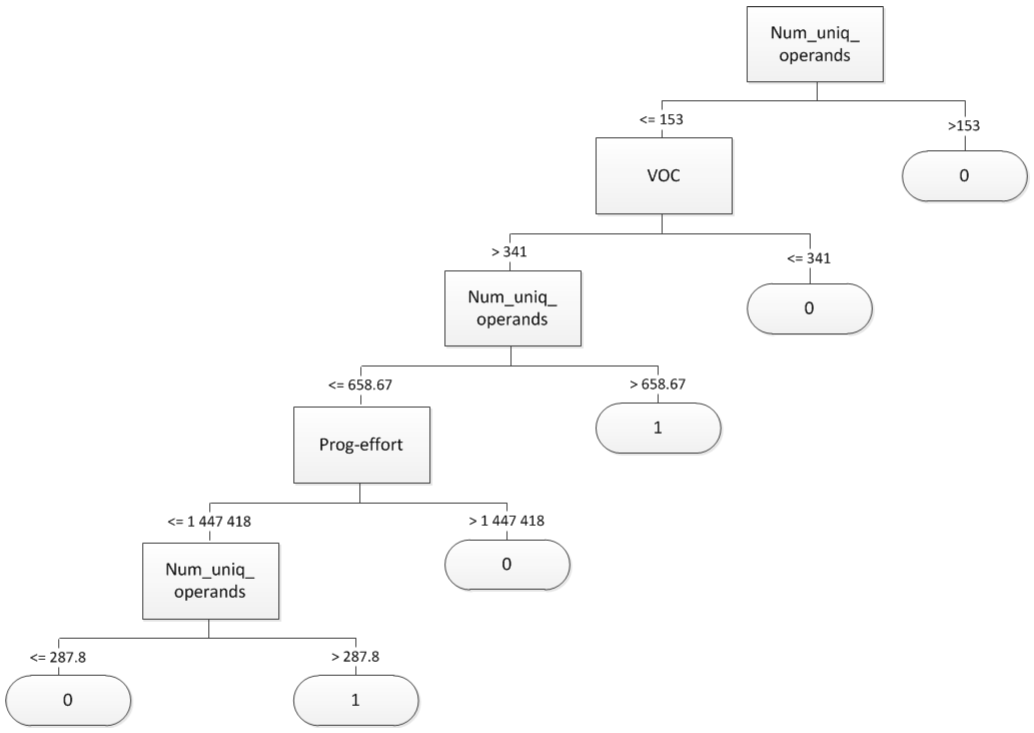
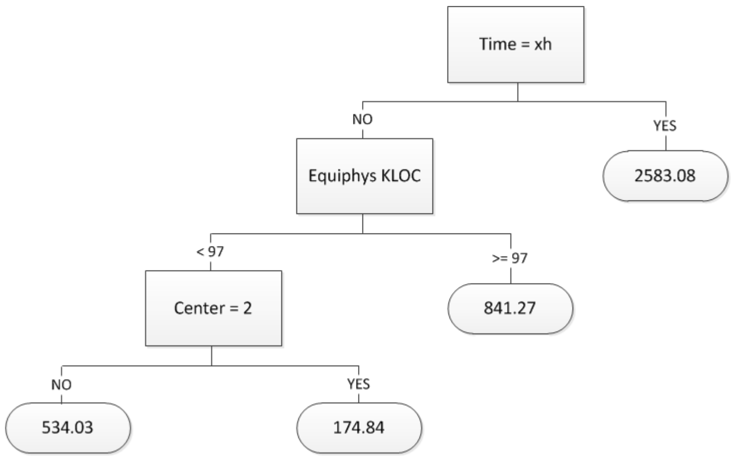


Figure 8: Example of ALPAR Tree for effort prediction using the Cocomonasa2 data set



7. Threats to Validity and Future Research

In an empirical study it is important to take into account potential threats to validity of the results. A first possible source of bias that can be detected are the preprocessing steps that address the issues such as missing value handling and input selection, which can play an important role in the outcomes of the experiments. While the same preprocessing steps are applied on all data sets, further experiments on the impact of the these steps on the results can be interesting for future research. Secondly, the data used in the experiments can be seen as another possible source of bias. That is, questions can be raised about the representativeness and suitability of the data. This issue is partially tackled by the fact that data from the public domain are used in this study. In this way, the results can be verified and compared with similar studies in the domain. Moreover, as many authors have argued in favor of the use of public data sets from the NASA MPD and PROMISE repository [10, 17, 25, 27], we are convinced that the obtained results are useful to the current research landscape of software estimations. The newly generated data sets in this study on the other hand, are aligned with the attribute space from NASA data sets and are made publicly available in order to ensure consistency and replicability. Furthermore, it can be noticed that the choices of algorithms are made based on previous research in the domains [6, 27]. It can be interesting to check if the same results are obtained when extracting rules from other non-linear models. Another important challenge of using rule extraction is that it implies an extra layer of complexity in the model, thereby increasing the running time. Moreover, the ALPA algorithm was implemented in the Weka workbench and therefore the size of the datasets that can be used is limited to the order of about 10.000 data points. Finally, we mentioned already that the ALPA algorithm optimizes for accuracy. Since the choice of performance measure depends largely on the industrial context, it might be useful to adapt the ALPA algorithm so it can optimize for recall, precision or other performance measures. This might be an interesting topic for future research.

8. Conclusion

Software fault and effort prediction are both important tasks in order to minimize costs in a software company. The predictive model used in these cases needs to be both accurate and comprehensible. Unfortunately, to obtain predictive performance, comprehensibility is often sacrificed and visa versa. In this paper we illustrated that rule extraction can tackle this issue and investigated the trade-off between both requirements.
By applying the rule extraction technique ALPA we improve the rule sets in terms of fidelity and, in most cases, accuracy and recall as well. On the other hand our results validate that rule extraction allows us to get more insight in the inner workings of complex models since all extracted trees were easy to understand. This in turn improves the acceptance of the model by the end-user. We thereby hope that this methodology further facilitates the widespread adoption of data mining in software development.

Acknowledgements

We would like to thank the Flemish Research Council (FWO) for financial support (Grant G.0827.12N).

References

[1] K. Dejaeger, Essays on empirical software engineering, Ph.D. thesis, KU Leuven (2012).

[2] M. Jørgensen, K. Moløkken-Østvold, How large are software cost overruns? A review of the 1994 CHAOS report, Information and Software Technology 48 (4) (2006) 297–301.

[3] H. Uwano, Y. Kamei, A. Monden, K.-i. Matsumoto, An analysis of cost-overrun projects using financial data and software metrics, in: Software Measurement, 2011 Joint Conference of the 21st International Workshop on and 6th International Conference on Software Process and Product Measurement, 2011, pp. 227–232.

[4] S. Grimstad, M. Jørgensen, K. Moløkken-Østvold, Software effort estimation terminology: The tower of babel, Information and Software Technology 48 (4) (2006) 302–310.

[5] M. Bloch, S. Blumberg, J. Laartz, Delivering large-scale it projects on time, on budget, and on value, McKinsey on Business Technology (27) (2012) 2–7.

[6] S. Lessmann, B. Baesens, C. Mues, S. Pietsch, Benchmarking classification models for software defect prediction: A proposed framework and novel findings, Software Engineering, IEEE Transactions on 34 (4) (2008) 485–496.

[7] K. Dejaeger, T. Verbraken, B. Baesens, Toward comprehensible software fault prediction models using bayesian network classifiers, IEEE Transactions on Software Engineering 39 (2) (2013) 237–257.

[8] M. Fagan, Design and code inspections to reduce errors in program development, IBM Systems Journal 38 (2.3) (1999) 258–287.

[9] B. Boehm, P. Papaccio, Understanding and controlling software costs, IEEE Transactions on Software Engineering 14 (10) (1988) 1462–1477.

[10] O. Vandecruys, D. Martens, B. Baesens, C. Mues, M. D. Backer, R. Haesen, Mining software repositories for comprehensible software fault prediction models, Journal of Systems and Software 81 (5) (2008) 823–839.

[11] I. Witten, E. Frank, M. Hall, Data Mining: Practical Machine Learning Tools and Techniques, Morgan Kaufmann, 2011.

[12] G. Shmueli, O. Koppius, Predictive analytics in information systems research., MIS Quarterly 35 (3) (2011) 553 – 572.

[13] B. Baesens, R. Setiono, C. Mues, J. Vanthienen, Using neural network rule extraction and decision tables for credit-risk evaluation, Management Science 49 (3) (2003) 312–329.

[14] W. Verbeke, K. Dejaeger, D. Martens, J. Hur, B. Baesens, New insights into churn prediction in the telecommunication sector: A profit driven data mining approach, European Journal of Operational Research 218 (1) (2012) 211 – 229.

[15] L. Passmore, J. Goodside, L. Hamel, L. Gonzales, T. Silberstein, J. Trimarchi, Assessing decision tree models for clinical in-vitro fertilization data, Technical Report TR03-296, Dept. of Computer Science and Statistics, University of Rhode Island (2003).

[16] D. Martens, F. Provost, Explaining data-driven document classifications., MIS Quarterly 38 (1) (2014) 73 – A6.

[17] K. Dejaeger, W. Verbeke, D. Martens, B. Baesens, Data mining techniques for software effort estimation: A comparative study, IEEE Transactions on Software Engineering 38 (2) (2012) 375–397.

[18] D. Martens, J. Vanthienen, W. Verbeke, B. Baesens, Performance of classification models from a user perspective, Decision Support Systems 51 (4) (2011) 782 – 793.

[19] O. Maimon, L. Rokach, Decomposition methodology for knowledge discovery and data mining, in: O. Maimon, L. Rokach (Eds.), Data Mining and Knowledge Discovery Handbook, Springer US, 2005, pp. 981–1003.

[20] M. Craven, J. Shavlik, Extracting tree-structured representations of trained neural networks, in: Advances in Neural Information Processing Systems, MIT Press, Cambridge, MA, USA, 1996, pp. 24–30.

[21] D. Martens, B. Baesens, T. V. Gestel, J. Vanthienen, Comprehensible credit scoring models using rule extraction from support vector machines, European Journal of Operational Research 183 (3) (2007) 1466 – 1476.

[22] U. Johansson, Obtaining accurate and comprehensible data mining models : An evolutionary approach, Ph.D. thesis, Linköping University, Department of Computer and Information Science, The Institute of Technology (2007).

[23] S. Dick, A. Meeks, M. Last, H. Bunke, A. Kandel, Data mining in software metrics databases, Fuzzy Sets and Systems 145 (1) (2004) 81–110.

[24] T. Ostrand, E. Weyuker, R. Bell, Predicting the location and number of faults in large software systems, IEEE Transactions on Software Engineering 31 (4) (2005) 340–355.

[25] T. Menzies, J. Greenwald, A. Frank, Data mining static code attributes to learn defect predictors, IEEE Transactions on Software Engineering 33 (1) (2007) 2–13.

[26] T. Khoshgoftaar, E. Allen, J. Hudepohl, S. Aud, Application of neural networks to software quality modeling of a very large telecommunications system, IEEE Transactions on Neural Networks 8 (4) (1997) 902–909.

[27] L. Guo, Y. Ma, B. Cukic, H. Singh, Robust prediction of fault-proneness by random forests, in: 15th International Symposium on Software Reliability Engineering, 2004, pp. 417–428.

[28] T. Khoshgoftaar, W. Jones, Classification tree models of software quality over multiple releases, IEEE Transactions on Reliability 49 (1) (2000) 4–11.

[29] A. Porter, R. Selby, Evaluating techniques for generating metric-based classification trees, Journal of Systems and Software 12 (3) (1990) 209 –218.

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

[31] J. Munson, T. Khoshgoftaar, The detection of fault-prone programs, Software Engineering, IEEE Transactions on 18 (5) (1992) 423–433.

[32] T. Khoshgoftaar, E. Allen, Logistic regression modeling of software quality, International Journal on Reliability, Quality and Safety Engineering 6 (4) (1999) 303–317.

[33] M. Shepperd, G. Kadoda, Comparing software prediction techniques using simulation, IEEE Transactions on Software Engineering 27 (11) (2001) 1014–1022.

[34] C. Catal, Software fault prediction: A literature review and current trends, Expert Systems with Applications 38 (4) (2011) 4626 – 4636.

[35] T. Hall, S. Beecham, D. Bowes, D. Gray, S. Counsell, A systematic literature review on fault prediction performance in software engineering, Software Engineering, IEEE Transactions on 38 (6) (2012) 1276–1304.

[36] E. Arisholm, L. C. Briand, M. Fuglerud, Data mining techniques for building fault-proneness models in telecom java software, in: Software Reliability, 2007. ISSRE'07. The 18th IEEE International Symposium on, IEEE, 2007, pp. 215–224.

[37] The Standish Group, Chaos report, Tech. rep. (2004).

[38] E. Jun, J. Lee, Quasi-optimal case-selective neural network model for software effort estimation, Expert Systems with Applications 21 (1) (2001) 1 – 14.

[39] M. Shepperd, C. Schofield, Estimating software project effort using analogies, IEEE Transactions on Software Engineering 23 (12) (1997) 736–743.

[40] G. Finnie, G. Wittig, J.-M. Desharnais, A comparison of software effort estimation techniques: Using function points with neural networks, case-based reasoning and regression models, Journal of Systems and Software 39 (3) (1997) 281–289.

[41] L. Briand, K. El Emam, D. Surmann, I. Wieczorek, K. Maxwell, An assessment and comparison of common software cost estimation modeling techniques, in: Proceedings of the 21st International Conference on Software Engineering, ICSE '99, ACM, New York, NY, USA, 1999, pp. 313–322.

[42] P. Braga, A. Oliveira, S. Meira, Software effort estimation using machine learning techniques with robust confidence intervals, in: 19th IEEE International Conference on Tools with Artificial Intelligence, Vol. 1, 2007, pp. 181–185.

[43] H. Park, S. Baek, An empirical validation of a neural network model for software effort estimation, Expert Systems with Applications 35 (3) (2008) 929 – 937.

[44] K. Kumar, V. Ravi, M. Carr, N. Kiran, Software development cost estimation using wavelet neural networks, Journal of Systems and Software 81 (11) (2008) 1853 – 1867.

[45] S.-J. Huang, N.-H. Chiu, L.-W. Chen, Integration of the grey relational analysis with genetic algorithm for software effort estimation, European Journal of Operational Research 188 (3) (2008) 898 – 909.

[46] B. Baesens, Developing intelligent systems for credit scoring using machine learning techniques, Ph.D. thesis, K.U.Leuven (2003).

[47] P. Tan, M. Steinbach, V. Kumar, Introduction to data mining, Pearson Eduction, Boston, USA, 2006.

[48] B. Baesens, T. V. Gestel, S. Viaene, M. Stepanova, J. Suykens, J. Vanthienen, Benchmarking state-of-the-art classification algorithms for credit scoring, Journal of the Operational Research Society 54 (6) (2003) 627–635.

[49] D. Martens, B. Baesens, Building acceptable classification models, in: Data Mining, Vol. 8 of Annals of Information Systems, Springer US, 2010, pp. 53–74.

[50] L. Breiman, Random forests, Machine learning 45 (1) (2001) 5–32.

[51] V. Vapnik, The Nature of Statistical Learning Theory, Springer, N.Y., 1995.

[52] B. Schölkopf, A. J. Smola, Learning with kernels: Support vector machines, regularization, optimization, and beyond, MIT press, 2002.

[53] A. Smola, B. Schölkopf, A tutorial on support vector regression, Statistics and computing 14 (3) (2004) 199–222.

[54] J. Quinlan, C4.5: Programs for Machine Learning, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[55] L. Rokach, O. Maimon, Data Mining with Decision Trees, World Scientific Publishing Co., 2008.

[56] J. Huysmans, K. Dejaeger, C. Mues, J. Vanthienen, B. Baesens, An empirical evaluation of the comprehensibility of decision table, tree and rule based predictive models, Decision Support Systems 51 (1) (2011) 141–154.

[57] T. Minka, Data mining : Regression trees, Lecture notes, Carnegie Mellon University (2003).

[58] E. Junqué de Fortuny, D. Martens, Active learning-based pedagogical rule extraction, University of Antwerp (2014).

[59] D. Martens, B. Baesens, T. Van Gestel, Decompositional Rule Extraction from Support Vector Machines by Active Learning, IEEE Transactions on Knowledge and Data Engineering 21 (2) (2009) 178–191.

[60] E. Junqué de Fortuny, D. Martens, Active learning based rule extraction for regression, in: 2012 IEEE 12th International Conference on Data Mining Workshops (ICDMW), 2012, pp. 926–933.

[61] M. Chapman, P. Callis, W. Jackson, Metrics data program, NASA IV and V Facility (2004).

[62] T. Menzies, B. Caglayan, Z. He, E. Kocaguneli, J. Krall, F. Peters, B. Turhan, The promise repository of empirical software engineering data, http://promisedata.googlecode.com (June 2012).

[63] A. Tosun, A. Bener, B. Turhan, T. Menzies, Practical considerations in deploying statistical methods for defect prediction: A case study within the turkish telecommunications industry, Inf. Softw. Technol. 52 (11) (2010) 1242–1257.

[64] D. Greves, B. Schreiber, The ESA initiative for software prodcutivity benchmarking and effort estimation., http://esapub.esrin.esa.it, ESA Bulletin, no 87 (August 1996).

[65] M. Hall, G. Holmes, Benchmarking attribute selection techniques for discrete class data mining, IEEE Transactions on Knowledge and Data Engineering. 15 (6) (2003) 1437–1447.

[66] X. Zhang, H. Zhang, Comments on data mining static code attributes to learn defect predictors, IEEE Transactions on Software Engineering 33 (9) (2007) 635–636.

[67] T. Menzies, A. Dekhtyar, J. Distefano, J. Greenwald, Problems with precision: a response to comments on data mining static code attributes to learn defect predictors, IEEE Transactions on Software Engineering 33 (9) (2007) 637.