STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LXI**, Number 2, 2016

A STUDY ON SOFTWARE DEFECT PREDICTION USING FUZZY DECISION TREES

ZSUZSANNA MARIAN, ISTVÁN-GERGELY CZIBULA, IOAN-GABRIEL MIRCEA AND VLAD-SEBASTIAN IONESCU

Abstract. In this paper we conduct a study on applying fuzzy decision trees for software defect prediction, investigating the results of varying different parameters, for the FuzzyDT method, introduced in a previous paper. The proposed method uses software metrics and fuzzy decision trees to identify potentially faulty software entities like components, modules, methods, etc. Experiments are performed on five open-source case studies in order to analyze the effect of using different thresholds for the software metrics used to define the fuzzy membership functions as well as using different *impurity* functions in building the fuzzy decision tree. We also analyse whether using only certain selected software metrics leads to a better performance than using all the software metrics from the data sets. The obtained results confirm that the *fuzzy* approach outperforms the crisp one and the results are better than most of the results already reported in the literature for the data sets considered in our evaluation.

1. Introduction

Software defect prediction represents the activity of identifying software modules which are likely to develop errors in a forthcoming version of a software system, being of major importance for software testing and for assuring the software quality as well. The methods for detecting faulty software entities are useful for suggesting to developers the software modules that should be rigorously tested. These software entities can be software components, modules, packages, classes, methods, functions or other software artifacts.

The software maintenance process represents a major part of a software life cycle, requiring a large software engineering effort. The software engineering

Received by the editors: September 20, 2016.

2010 *Mathematics Subject Classification.* 68N99, 68T05.

1998 *CR Categories and Descriptors.* D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement - *Restructuring, reverse engineering, and reengineering*; D.2.8 [**Software Engineering**]: Metrics – *Product metrics*; I.2.6 [**Artificial Intelligence**]: Learning – *Induction*;

Key words and phrases. software defect prediction, software metrics, fuzzy decision trees.

literature reveals that understanding the software represents about half of the amount of effort allocated to the maintenance activity. Fixing defects represents one of the main software maintenance activities, also being referred to as *corrective maintenance* [8].

For increasing the efficiency of the software defect-fixing process, defect prediction models are useful for anticipating locations in a software system where future defects may appear. Identifying software defects is difficult, mainly for complex software projects. The main difficulty related to building supervised defect predictors is the fact that the number of defects in software projects is much smaller than the number of non-defective entities and thus, the training data is highly imbalanced [2].

In [10] we have introduced a novel method for software defect detection using *fuzzy decision trees* (*FuzzyDT*). The proposed method uses software metrics, which are often used for software defect prediction. We have provided in [10] experiments on *JEdit* and *Ant* open source systems, to show the effectiveness of our method. In this paper, we further investigate the *FuzzyDT* method on other five publicly available data sets [4] called *Ar1*, *Ar3*, *Ar4*, *Ar5* and *Ar6*. We also analyse different parameter settings for our method. The following criteria are used in the performed analysis:

- Using different threshold values for the software metrics needed to define the *fuzzy membership functions*.
- Using only some selected software metrics or using all of the software metrics from the data sets for building the *fuzzy* decision trees.
- Using different *impurity functions* in building the *fuzzy* decision tree.

The remainder of the paper is structured as follows. Section 2 presents the background of the *FuzzyDT* method for software defect prediction, while Section 3 describes the criteria used in our further study. Section 4 presents the experimental results obtained on several case studies, as well as an analysis of the obtained results and a comparison to related work. Section 5 contains the conclusions of the paper.

2. Background

In this section we present the main background of the *FuzzyDT* method we have previously introduced in [10] for software defect prediction.

A fuzzy decision tree appears to be an effective choice for solving the software defect prediction problem for the following reasons. Most importantly, the nature of the data concerning software metrics makes a clear differentiation between the defective and non-defective classes virtually impossible and therefore a certain degree of uncertainty must be taken into account in the decision process. That is why it is important that accurate fuzzy functions are defined and incorporated in the classical decision tree paradigm, thus transforming it into a fuzzy decision tree.

A fuzzy decision tree [7] follows the classical decision tree paradigm for classification in the sense that, starting from the entire data set, a tree is constructed by selecting at any decision step the most relevant attribute with respect to an impurity measure and splitting the remaining attribute information on several branches according to the distinct values that underlie the chosen attribute. The internal nodes of the fuzzy tree contain all the instances from the data set, but each instance has a membership degree to each class. A leaf node from the fuzzy tree, instead of indicating a single classification as in the classical approach, contains cumulative membership values to each of the classes.

However, in the case of the fuzzy approach, the distinct values that enable the decision branching process of the tree are replaced by fuzzy functions concerning the attribute. The *entropy* and the *information gain* measures, which play a fundamental part in the decision process, are strongly dependent not only on the balance in size between the target classes used in training, in the current case the classes of defective or non-defective software components, but also on the construction of the fuzzy membership functions concerning each attribute since these functions need to be established in such a way that they better enable the defect classification process.

3. Comparison criteria

The FuzzyDT method for software defect prediction presented in Section 2 depends on different parameters, whose selection can influence the accuracy of the obtained results. In this section we present the comparison criteria that are used in our experiments and serves as the basis for the study performed on the FuzzyDT method.

3.1. **Thresholds for the software metrics.**

Our first action is to investigate how different software metrics thresholds for the fuzzy membership function influence the results of the algorithm.

All the data sets used for the experiments in this paper contain 29 software metrics and the class label (*defective* or *non-defective*). In order to build the *fuzzy* decision trees, we define for each software metric two trapezoidal fuzzy functions: the first fuzzy function determines the membership degree of a software metric value to the class of *defective* entities and the second fuzzy function determines the membership degree to the class of *non-defective* entities.

For defining the fuzzy functions we take inspiration from the work presented in [6]. The authors have created a large data set by computing the value

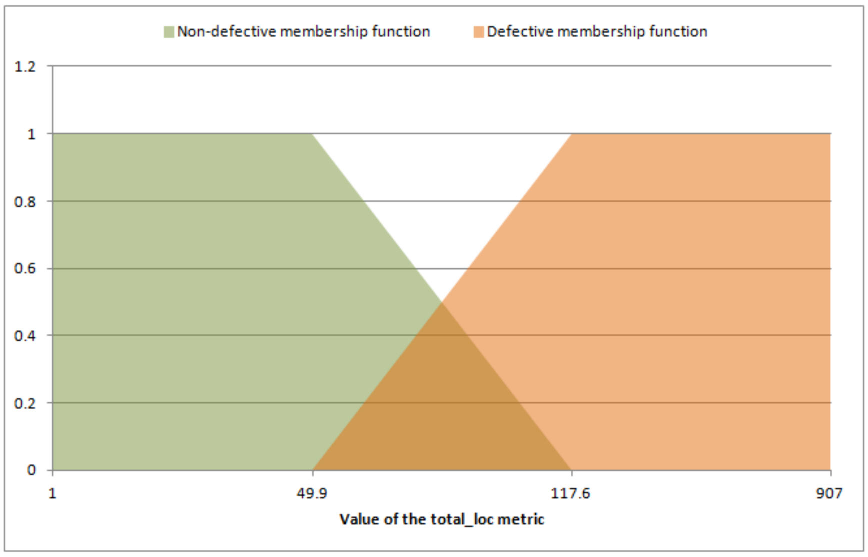

FIGURE 1. The two fuzzy functions defined for the *total_loc* software metric.

of different software metrics for 111 software systems and identified thresholds to group the value of these software metrics into three categories: *Good*, *Regular* and *Bad*. They defined the threshold between the first two categories at the 70^{th} percentile and between the second and the third categories at the 90^{th} percentile in the data. Similar to this work, we merge the five *Ar* data sets used for the experimental evaluation and for each software metric we compute the value of 70 and 90 percentile and use these values for defining the fuzzy functions. For example, the two *fuzzy* functions defined for the *total_loc* software metric, where the two percentile values are 49.9 and 117.6, are presented on Figure 1.

Denoting the two threshold values used to define the fuzzy functions as a and b, the membership degree of a software metric value x to the *non-defective* class can be computed using Formula (1). Similarly, the membership degree to the defective class can be computed using Formula (2).

$$(1) \qquad \mu_{non-defect}(x) = \begin{cases} 1, & x < a \\ \frac{b-x}{b-a} & a \le x \le b \\ 0, & x > b \end{cases}$$

$$\mu_{defect}(x) = \begin{cases} 0, & x < a \\ \frac{x-a}{b-a} & a \leq x \leq b \\ 1, & x > b \end{cases} \tag{2}$$

Besides using the value of percentiles 70 and 90 as thresholds for defining the *fuzzy* functions, we use two other pairs of thresholds as well. These thresholds are presented in Table 1. By modifying the threshold value, we are actually modifying the section where the two functions overlap. We consider the first threshold pair *Regular* and we defined one where the overlap section is narrower (second row of Table 1), and one where the overlap section is wider (third row of Table 1).

Name	**Percentile threshold for a**	**Percentile threshold for b**
Regular	70	90
Narrow overlap	75	85
Wide overlap	65	95

TABLE 1. Different percentile thresholds used for defining the *fuzzy* functions.

3.2. **Software metrics.**

In this section investigate the effect of using different *software metrics* in building the *fuzzy decision tree* for the software defect prediction task.

3.2.1. *All 29 software metrics.*

We use the values of 29 different McCabe and Halstead software metrics: *halstead_vocabulary*, *unique_operators*, *unique_operands*, *total_operands*, *total_operators*, *executable_loc*, *halstead_length*, *total_loc*, *halstead_volume*, *halstead_error*, *halstead_difficulty*, *halstead_effort*, *halstead_time*, *blank_loc*, *condition_count*, *multiple_condition_count*, *branch_count*, *decision_count*, *cyclomatic_complexity*, *halstead_level*, *comment_loc*, *code_and_ comment_loc*, *decision_density*, *call_pairs*, *design_complexity*, *cyclomatic_density*, *normalized_ cyclomatic_complexity*, *design_density formal_parameters.*

3.2.2. *A subset of 9 software metrics.*

For reducing the dimensionality of the feature set characterizing the software entities, we use the analysis performed in [11] on the *Ar3*, *Ar4* and *Ar5* data sets for selecting relevant software metrics for the software defect prediction task. For determining the importance of the software metrics *information gain* (IG) measure was used. From the software metrics having IG values higher than a given threshold, 9 software metrics measuring different characteristics of the software system

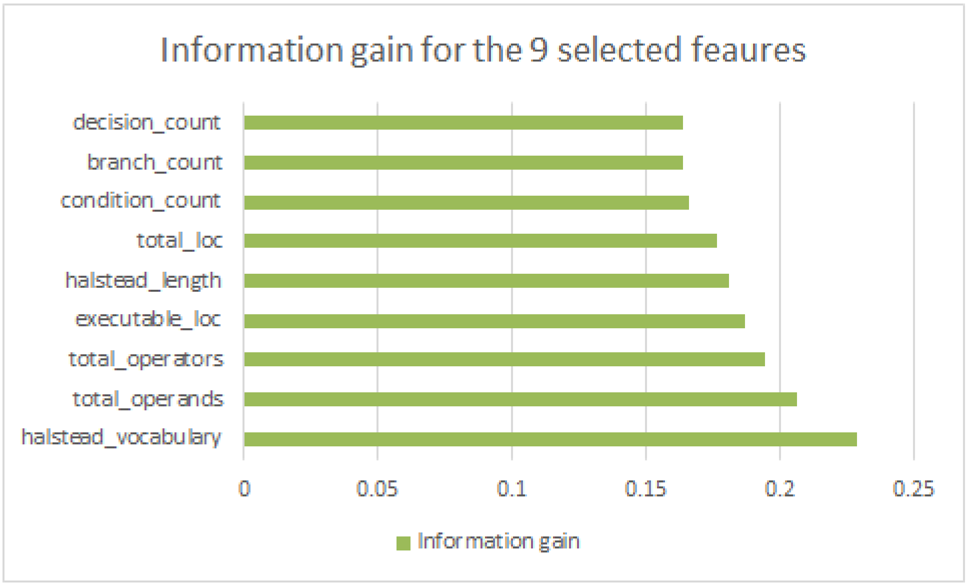

FIGURE 2. Selected software metrics

were selected in [11] being representative for the software defect detection process: *halstead_vocabulary*, *total_operands*, *total_operators*, *executable_loc*, *halstead_length*, *total_loc*, *condition_count*, *branch_count*, *decision_count* [11]. These software metrics are used as features in our classification task.

The selected software metrics are shown in Figure 2.

3.3. **Impurity functions.**

In this section we investigate the effect of using different *impurity functions* in building the *fuzzy decision tree*.

For building the *fuzzy decision tree*, two impurity functions are used to measure the heterogeneity of a set of l software entities labeled as *defects* and *non-defects*. As we have described in Section 2, each internal node from the fuzzy decision tree stores all the instances from the training data set (let us denote t by $\mathcal{D}$), but each instance has a certain membership degree to each class.

The first impurity function is the one usually used when building decision trees, namely the *entropy*.

The *entropy* measure at a *node* from the *fuzzy* decision tree is computed as in Formula (3) and it represents a generalization for the *entropy* from the *crisp* case.

$$Entropy(node) = -\frac{n_{defect}}{n_{defect} + n_{non-defect}} \cdot \log \frac{n_{defect}}{n_{defect} + n_{non-defect}} -$$

$$\frac{n_{non-defect}}{n_{defect} + n_{non-defect}} \cdot \log \frac{n_{non-defect}}{n_{defect} + n_{non-defect}} \tag{3}$$

where n_{defect} sums the membership degrees for the defective entities from $\mathcal{D}$, $n_{non-defect}$ sums the membership degrees for the non-defective entities from $\mathcal{D}$.

The second impurity function we use is the *misclassification* function. The *misclassification* at a certain *node* from the tree is computed as shown in Formula (4) and generalizes the definition of the *misclassification* function for the *crisp* case.

$$misclassification(node) = \begin{cases} \frac{n_{non-defect}}{n_{defect}+n_{non-defect}} & if \quad n_{defect} > n_{non-defect} \\ \frac{n_{defect}}{n_{defect}+n_{non-defect}} & otherwise \end{cases} \tag{4}$$

The notations in Formula (4) are the same as in Formula (3).

4. Experimental results

In this section we provide an experimental evaluation of the FuzzyDT model (described in Section 2) on five open-source data sets previously used in the software defect detection literature. For each case study, the comparison criteria presented in Section 3 is applied.

First, the data sets used in our case studies are described, then the obtained experimental results are provided. An analysis of the obtained results and their comparison to related work is provided in Section 4.3.

4.1. Data sets.

The data sets used in our experiments are called *Ar1*, *Ar3*, *Ar4*, *Ar5* and *Ar6*, they are open-source and available at [4]. These data sets were obtained from a Turkish white-goods manufacturer embedded software implemented in C [11]. From these software products the functions and methods were extracted and these entities are represented as 29-dimensional vectors containing the value of different McCabe and Halstead software metrics. The data sets used in our case studies are composed of these high-dimensional representations. For each software entity from the data sets, the class label denoting whether the entity is *defective* or *not* is known.

Table 2 gives the description of the *Ar1*-*Ar6* datasets used in our case studies. For each data set, its *difficulty*, as well as the number of *defects* and *non-defects* are shown. The measure of *difficulty* for a data set was proposed

Data set	**Defective**	**Non-defective**	**Difficulty**
Ar1	9	112	0.666
Ar3	8	55	0.625
Ar4	20	87	0.7
Ar5	8	28	0.375
Ar6	15	86	0.666

TABLE 2. Description of the *Ar1-Ar6* data sets.

in [3] by Boetticher and represents the percentage of software entities from the data set for which the nearest neighbor has a different class label. For computing the difficulty of the data sets we considered only the percentage of software defects (defective entities) for which the nearest neighbor is non-defective.

One can observe from Table 2 the imbalanced nature of the data sets, with much smaller number of *defective* entities than *non-defective* ones. We also observe large values for the *difficulty* measure, which confirm the complexity of the *defect* classification task.

In order to make the data sets less imbalanced, we decided to add to each data set more defective entities, but instead of oversampling or creating synthetic instances, we use the actual defective instances from the other data sets. For example, to the 9 defective entities from *Ar1* we add all the defective entities from the other four data sets. In this way, all data sets contain the same 60 defective entities, while the number of non-defective entities remains the same as in Table 2. The only exception is the *Ar5* data set, which has only 28 non-defective entities and to which we only add the 20 defective entities from *Ar4*, making it perfectly balanced.

Figure 3 shows the *Ar1-Ar6* data sets reduced to two dimensions using t-SNE [13], after the aforementioned transformations. It can be seen that the defective and non-defective instance are clustered very close together, with no clear way to separate them in two dimensions. This is another proof of the problem's intrinsic difficulty.

4.2. **Results.** For evaluating the performance of the *fuzzy* decision tree, we have used a *leave-one-out* cross validation technique [14]. For each data set a *fuzzy* decision tree is built using all but one instances and the tree is tested on the instance not used for the training. This process is repeated until every instance from the data set was used once for testing.

During the cross validation process, the confusion matrix is computed. The confusion matrix contains the number of *true positives* (TP; defective instances classified as defective), *true negatives* (TN; non-defective instances classified as

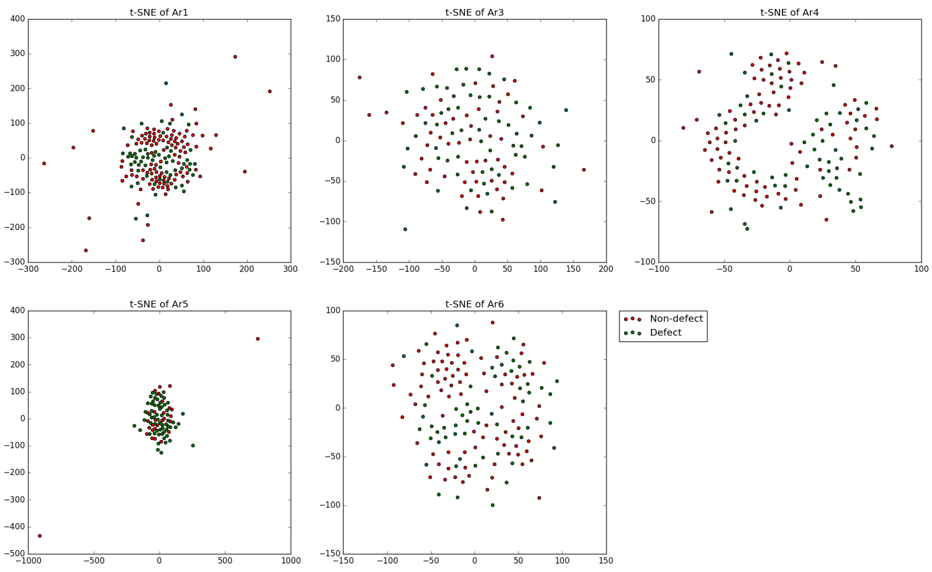

FIGURE 3. Two dimensional representations using t-SNE of our transformed data sets.

non-defective), *false positives* (FP; non-defective entities classified as defective) and *false negatives* (FN; defective entities classified as non-defective).

In the literature there is a large number of different performance metrics that can be computed from the confusion matrix. While *accuracy* (Formula 5) is often used, it is not suitable in the case of imbalanced data sets. A more relevant evaluation measure for the performance of the software defect classifier is the *Area under the ROC curve* (AUC) measure [5]. This measure is usually used in case of classifiers that, instead of returning directly the class of the tested instance, return a single value, which is transformed into the class label using a threshold. For such approaches modifying the value of this threshold can lead to different values for the *Probability of detection* (Formula 6) and the *Probability of false alarm* (Formula 7) measures. For each threshold, the point (Pf, Pd) is represented on a plot, and AUC measures the area under this curve.

$$Acc = \frac{TP + TN}{TP + TN + FP + FN} \tag{5}$$

$$Pd = \frac{TP}{TP + FN} \tag{6}$$

$$Pf = \frac{FP}{FP + TN} \tag{7}$$

In case of the approaches where the output is directly the class label, like our approach, there is one single (Pf, Pd) point, but it can be linked to the $(0, 0)$ and $(1, 1)$ points and the area under this curve can be computed using Formula 8.

$$AUC = (1 - Pf) * Pd + \frac{Pf * Pd}{2} + \frac{(1 - Pf) * (1 - Pd)}{2} \tag{8}$$

The results achieved for the five data sets used for experimental evaluation for all the parameter combinations presented in Section 3 are presented in Tables 3, 4, 5, 6, 7. In these tables, besides the value of the *Acc* and *AUC* metrics, we provide the complete confusion matrices as well.

Thresholds	**#Metrics**	**Impurity function**	**TP**	**FP**	**TN**	**FN**	**Acc**	**AUC**
a=70, b=90	29	Entropy	38	8	104	22	0.826	0.781
		Misclassification	38	8	104	22	0.826	0.781
	9	Entropy	35	5	107	25	0.826	0.769
		Misclassification	34	5	107	26	0.820	0.761
a=75, b=85	29	Entropy	36	11	101	24	0.797	0.751
		Misclassification	39	12	100	21	0.808	0.771
	9	Entropy	34	5	107	26	0.820	0.761
		Misclassification	34	5	107	26	0.820	0.761
a=65, b=95	29	Entropy	39	4	108	21	0.855	0.807
		Misclassification	36	6	106	24	0.826	0.773
	9	Entropy	31	3	109	29	0.814	0.745
		Misclassification	31	3	109	29	0.814	0.745

TABLE 3. Detailed results obtained for the *Ar*1 data set.

4.3. **Discussion and comparison to Related Work.**

To get an overall view of the results, Table 8 presents the minimum, maximum, average and population standard deviation of the *Acc* and *AUC* values across each configuration for each data set. It can be seen that the best average accuracy is obtained on *Ar1*, while the best average *AUC* is obtained on *Ar5*.

We also record, in Table 9, for each data set the configurations for which the highest *AUC* values are achieved. The column *T* contains the percentile thresholds, *M* contains the number of metrics, and *I* contains the impurity function of each configuration.

Thresholds	**#Metrics**	**Impurity function**	**TP**	**FP**	**TN**	**FN**	**Acc**	**AUC**
a=70, b=90	29	Entropy	38	26	29	22	0.583	0.580
		Misclassification	40	20	35	20	0.652	0.652
	9	Entropy	36	12	43	24	0.687	0.691
		Misclassification	36	15	40	24	0.661	0.664
a=75, b=85	29	Entropy	42	26	29	18	0.617	0.614
		Misclassification	42	26	29	18	0.617	0.614
	9	Entropy	40	14	41	20	0.704	0.706
		Misclassification	37	18	37	23	0.644	0.645
a=65, b=95	29	Entropy	44	24	31	16	0.652	0.649
		Misclassification	40	23	32	20	0.626	0.624
	9	Entropy	31	15	40	29	0.617	0.622
		Misclassification	33	18	37	27	0.609	0.611

TABLE 4. Detailed results obtained for the *Ar*3 data set.

Thresholds	**#Metrics**	**Impurity function**	**TP**	**FP**	**TN**	**FN**	**Acc**	**AUC**
a=70, b=90	29	Entropy	40	10	77	20	0.796	0.776
		Misclassification	39	15	72	21	0.755	0.739
	9	Entropy	27	7	80	33	0.728	0.685
		Misclassification	26	7	80	34	0.721	0.676
a=75, b=85	29	Entropy	32	15	72	28	0.708	0.681
		Misclassification	41	13	74	19	0.782	0.767
	9	Entropy	29	11	76	31	0.714	0.678
		Misclassification	30	13	74	30	0.708	0.675
a=65, b=95	29	Entropy	36	6	81	24	0.796	0.766
		Misclassification	34	9	78	26	0.762	0.732
	9	Entropy	25	5	82	35	0.728	0.680
		Misclassification	26	5	82	34	0.735	0.688

TABLE 5. Detailed results obtained for the *Ar*4 data set.

From Table 9 we can see that in case of each data set the highest AUC value was achieved for a different configuration.

Since looking at the whole configuration does not lead to a conclusion regarding the best configuration, in the following we compare the results for each of the three comparison criteria presented in Section 3 separately. Table 10 shows the results of the comparison.

Thresholds for the fuzzy functions. The second column from Table 10 contains for each of the three thresholds used for the fuzzy functions the number of cases when the highest AUC is achieved for those threshold values, the other parameters having the same value. For example, for the *Ar1* data set,

Thresholds	#Metrics	Impurity function	TP	FP	TN	FN	Acc	AUC
a=70, b=90	29	Entropy	24	4	24	4	0.857	0.857
		Misclassification	24	2	26	4	0.893	0.893
	9	Entropy	21	6	22	7	0.768	0.768
		Misclassification	22	6	22	6	0.786	0.786
a=75, b=85	29	Entropy	22	5	23	6	0.804	0.804
		Misclassification	19	5	23	9	0.750	0.750
	9	Entropy	21	6	22	7	0.768	0.768
		Misclassification	21	6	22	6	0.768	0.768
a=65, b=95	29	Entropy	21	5	23	7	0.786	0.786
		Misclassification	22	4	24	6	0.821	0.821
	9	Entropy	20	6	22	8	0.750	0.750
		Misclassification	22	6	22	6	0.786	0.786

TABLE 6. Detailed results obtained for the *Ar*5 data set.

Thresholds	#Metrics	Impurity function	TP	FP	TN	FN	Acc	AUC
a=70, b=90	29	Entropy	40	13	73	20	0.774	0.758
		Misclassification	42	11	75	18	0.801	0.786
	9	Entropy	39	6	80	21	0.815	0.790
		Misclassification	39	6	80	21	0.815	0.790
a=75, b=85	29	Entropy	35	16	70	25	0.719	0.699
		Misclassification	37	14	72	23	0.747	0.727
	9	Entropy	38	6	80	22	0.808	0.782
		Misclassification	38	4	82	22	0.822	0.793
a=65, b=95	29	Entropy	39	5	81	21	0.822	0.796
		Misclassification	40	5	81	20	0.829	0.804
	9	Entropy	35	5	81	25	0.795	0.763
		Misclassification	35	5	81	25	0.795	0.763

TABLE 7. Detailed results obtained for the *Ar*6 data set.

Data set	Acc				AUC			
	Min	Max	Avg	Stdev	Min	Max	Avg	Stdev
Ar1	0.797	0.855	0.821	0.013	0.745	0.807	0.767	0.017
Ar3	0.583	0.704	0.639	0.033	0.580	0.706	0.639	0.034
Ar4	0.708	0.796	0.744	0.032	0.675	0.776	0.712	0.039
Ar5	0.750	0.893	0.795	0.042	0.750	0.893	0.795	0.042
Ar6	0.719	0.829	0.795	0.032	0.699	0.804	0.771	0.030

TABLE 8. Minimum, maximum, average and population standard deviations of the obtained values on each data set.

Data set	T	M	I
Ar1	65-95	29	Entropy
Ar3	75-85	9	Entropy
Ar4	70-90	29	Entropy
Ar5	70-90	29	Misclassification
Ar6	65-95	29	Misclassification

TABLE 9. The configurations for which the highest AUC values are achieved.

we compare the AUC values achieved for 9 software metrics with Entropy, for the three possible threshold values. When two thresholds have the same maximum AUC value, both are considered.

Number of software metrics used. Table 10 contains on the third column for both values for the number of software metrics used the number of cases when the value of the AUC measure is higher for that software metric number, the other parameters having the same value.

The impurity function used. The last column from Table 10 contains for both impurity functions used the number of cases when the value of the AUC measure was higher for that impurity function, the other parameters having the same value. The last column, *Ties*, counts the number of cases when the AUC value is the same for the two impurity functions. From Table 10 we can see that, even if Misclassification has the highest number of wins, there is no significant difference between the two impurity functions. While in case of the other two criteria one of the parameter values always has about twice as much wins as the other(s), in this case the difference between the number of wins for the two impurity functions is only one and there are also 7 ties. What is interesting is how these wins are achieved: on the *Ar1*, *Ar3* and *Ar4* data sets together Entropy has 10 wins (from a total of 11 wins) and Misclassification only 4 (and there are 4 ties), while on the other two data sets, *Ar5* and *Ar6*, Misclassification has 8 wins and Entropy only 1.

	Threshold value			**# Software metrics**		**Impurity function**	
	70-90	**75-85**	**65-95**	**9**	**29**	**Entropy**	**Misclassification**
Number of wins	12	5	6	10	20	11	12
Ties	–			–		7	

TABLE 10. Comparison of our results based on the considered comparison criteria.

Approach	**Ar1**	**Ar3**	**Ar4**	**Ar5**	**Ar6**
Our FuzzyDT	0.807	0.706	0.776	0.893	0.804
Genetic Programming [1]	0.530	0.67	0.65	0.67	0.630
Multiple Linear Regression [1]	0.550	0.61	0.62	0.55	0.590
Binary Logistic Regression [15]	0.551	**0.87**	0.73	0.39	0.722
Logistic Regression [12]	0.734	0.82	**0.82**	**0.91**	0.640
Logistic Regression [9]	0.494	n/a	n/a	n/a	0.538
Artificial Neural Networks [9]	0.711	n/a	n/a	n/a	0.774
Support Vector Machines [9]	0.717	n/a	n/a	n/a	0.721
Decision Trees [9]	**0.865**	n/a	n/a	n/a	**0.948**
Cascade Correlation Networks [9]	0.786	n/a	n/a	n/a	0.758
GMDH Network [9]	0.744	n/a	n/a	n/a	0.702
Gene Expression Programming [9]	0.547	n/a	n/a	n/a	0.688

Table 11. Comparison of our average AUC with related work on the same data sets.

Comparison to related work. Table 11 compares our best AUC values with supervised learning methods from the literature. It can be seen that our *fuzzy decision tree* approach leads to better results than most of the other approaches. Out of 11 other approaches, our approach is the second best on four of the data sets (*Ar1*, *Ar4*, *Ar5* and *Ar6*), and third best on the remaining *Ar3*.

Figure 4 presents, for each data set, how many of the other approaches our method outperforms. One can observe that the FuzzyDT method presented in this paper outperforms most of the approaches considered for comparison.

We note that some of the authors we compare ourselves to report average AUCs, while others, such as [1], report the best values. Since our standard deviations are small, we consider our comparisons to still be relevant and insightful.

5. Conclusions and Further Work

In this paper we have presented a study on the effect of changing different parameters for the FuzzyDT method we have previously introduced in [10] for *software defect prediction*. We have considered three possible variations for the FuzzyDT, and reported and analysed the results on the *Ar* open-source data sets. We showed that all variations can perform well, depending on the data set we are working with. This is why we recommend to try multiple settings and choose the best performing one for the problem at hand.

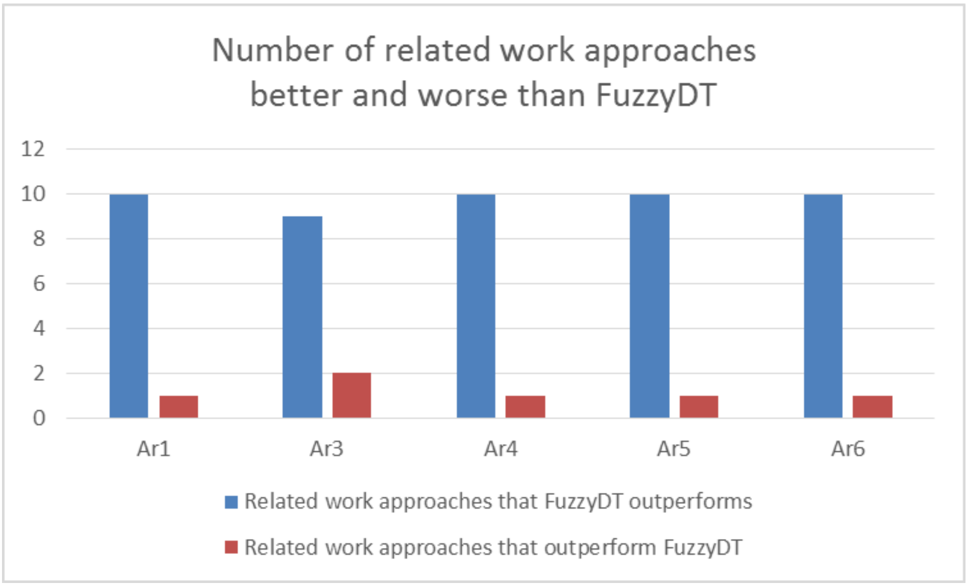

Figure 4. Counts of related work methods that are better and worse than *FuzzyDT* on the considered data sets.

Considering the thresholds for the fuzzy functions comparison criterion we can observe that the best threshold for defining the fuzzy functions seems to be 70-90. It provides higher AUC than the other two thresholds 12 times, which is slightly more than half of the cases. From the number of software metrics point of view we can see that using all the software metrics from the data set leads to better results than using only the 9 software metrics selected in [11]. The best impurity function which should be used can depend on the exact data set. Therefore, it is impossible to choose the best impurity function for a data set without performing experiments that consider both Entropy and Misclassification.

The experimental results we have obtained for the best parameter setting show that the *fuzzy* decision tree approach performs better than most of the existing approaches for the *software defect prediction* task. Further work will be done to use function approximation methods (like neural networks, radial basis function networks, etc.) to learn the fuzzy functions.

Acknowledgements

This work was supported by a grant of the Romanian National Authority for Scientific Research and Innovation, CNCS–UEFISCDI, project number PN-II-RU-TE-2014-4-0082.

References

[1] Wasif Afzal, Richard Torkar, and Robert Feldt. Resampling methods in software quality classification. *International Journal of Software Engineering and Knowledge Engineering*, 22(2):203–223, 2-12.

[2] Ishani Arora, Vivek Tetarwal, and Anju Saha. Open issues in software defect prediction. *Procedia Computer Science*, 46:906 – 912, 2015.

[3] Gary D. Boetticher. *Advances in Machine Learning Applications in Software Engineering*, chapter Improving the Credibility of Machine Learner Models in Software Engineering. IGI Global, 2007.

[4] Tera-promise repository. http://openscience.us/repo/.

[5] Tom Fawcett. An introduction to ROC analysis. *Pattern Recogn. Lett.*, 27(8):861–874, 2006.

[6] Tarcísio G. S. Filó, Mariza A. S. Bigonha, and Kecia A. M. Ferreira. A catalogue of thresholds for object-oriented software metrics. In *First International Conference on Advances and Trends in Software Engineering*, pages 48–55, 2015.

[7] C. Z. Janikow. Fuzzy decision trees: issues and methods. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 28(1):1–14, 1998.

[8] Mira Kajko-Mattsson, Stefan Forssander, and Ulf Olsson. Corrective maintenance maturity model (cm3): Maintainer's education and training. In *Proceedings of the 23rd International Conference on Software Engineering*, ICSE '01, pages 610–619, Washington, DC, USA, 2001. IEEE Computer Society.

[9] Ruchika Malhotra. Comparative analysis of statistical and machine learning methods for predicting faulty modules. *Applied Soft Computing*, 21:286–297, 2014.

[10] Z. Marian, I.G. Mircea, I.G. Czibula, and G. Czibula. A novel approach for software defect prediction using fuzzy decision trees. page to be published, Timisoara, Romania, 2016. IEEE Computer Science.

[11] Zsuzsanna Marian, Gabriela Czibula, Istvan-Gergley Czibula, and Sergiu Sotoc. Software defect detection using self-organizing maps. *Studia Universitatis Babes-Bolyai, Informatica*, LX(2):55 – 69, 2015.

[12] Jaechang Nam and Sunghun Kim. Heterogeneous defect prediction. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, pages 508–519. ACM, 2015.

[13] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

[14] G. Wahba, Y. Lin, and H. Zhang. GACV for support vector machines, or, another way to look at margin-like quantities. *Advances in Large Margin classifiers*, pages 297–309, 2000.

[15] Liguo Yu and Alok Mishra. Experience in predicting fault-prone software modules using complexity metrics. *Quality Technology & Quantitative Management*, 9(4):421–433, 2012.

Department of Computer Science,, Faculty of Mathematics and Computer Science,, Babeş-Bolyai University, Kogălniceanu 1, Cluj-Napoca, 400084, Romania.

E-mail address: {marianzsu, istvanc, mircea, ivlad}@cs.ubbcluj.ro