DETEKCIJA EKTREMNIH VREDNOSTI POMOĆU SATURACIONOG FILTRA U CILJU POBOLJŠANJA KVALITETA SOFTVERA
SATURATION FILTER APPROACH TO OUTLIER DETECTION IN SOFTWARE QUALITY IMPROVEMENT DOMAIN

Srdjan Sladojević, Igor Šetrajčić, Dubravko Ćulibrk, Fakultet tehničkih nauka, Novi sad

Sadržaj – *Efikasnost i uspešnost tehnika istraživanja podataka mnogo zavise od kvaliteta podataka koji se koriste za izradu modela koji će se koristiti. To čini kvalitet podataka važnim pitanjem u praksi. Činjenica da šum i u trening i u test podacima može negativno uticati na performanse algoritama mašinskog učenja opravdava povećanje istraživačkih napora posvećenih istraživanju robusnih i tačnih algoritama detekcije šuma. Ovaj rad prezentuje unapređenu metodologiju saturcionog filtra za detekciju šuma u domenu detekcije grešaka u softveru. Na osnovu analize rezultata, preporučuje se modifikovani filtar koji vrši eliminaciju šuma po slojevima. Filtrirani podaci su korisćeni za treniranje robusnog klasifikatora na bazi C4.5 stabla odlučivanja, a rezultati upoređeni sa rezultatima dobijeni3m bez filtriranja. Performanse klasifikacije se povećavaju sa eliminisanjem šuma iz trening podataka, pokazujući da je preporučeni saturacioni filtar sposoban da eliminiše šum.*

Abstract - *The effectiveness and efficiency of data mining techniques largely depends on the quality of the data used to build the data mining models employed. This makes the quality of the data a significant issue in practice. The fact that noise in both training and test data set can adversely affect the performance of machine learning algorithms justifies the increasing research effort dedicated to the investigation of robust, accurate noise detection algorithms. The paper presents an application of a saturation filter methodology to detect noisy examples in the domain of software fault detection. Based on the analysis of the results, a modified saturation filter is proposed, which performs stratified noise elimination. Filtered data is used to train robust C4.5 decision-tree classifiers and the prediction results are compared to those achieved by classifiers trained with noisy data. The classifier performance is increased when noise is eliminated from the dataset, showing that the proposed saturation filter is able to eliminate the noise and preserve the data relevant to the concept.*

1. INTRODUCTION

Advances in technology over the past decades have resulted in production of large amounts of data containing all sorts of potentially useful information. Extracting the information of interest from this large bulk is by no means a trivial task. In a strive to use the information gathered, data mining techniques have come to be employed in all areas of human activity. They are used to solve a variety of problems in fields such as business, finance, software quality [3], network intrusion detection and Internet search engines design. The use of data mining techniques has led to the automation of many processes and procedures that once solely depended on human expertise, leading to more cost-effective and time-effective processes.

The effectiveness and efficiency of data mining techniques largely depends on the quality of the data used to build the data mining models employed. This makes the quality of the data a significant issue in practice. Formally, the data is considered clean if it incorporates only the data stemming from the target concept, which the data mining techniques are trying to learn. All other data obscure the target concept and can be considered noise. In practice, and in this work, the term noise is used in a broader sense [4] denoting all examples that do not follow the same model as the rest of data. This definition "includes not only erroneous data but also surprising veridical data" [5]. Such data items are referred to as outliers.

Recently, several studies have been carried out on outlier detection in the data mining community. These studies broadly classify outliers into four categories. The first category is distribution-based outlier, which is defined based on the probability distribution. A standard distribution is used to fit the dataset. For example, a Gaussian mixture model is used to present the normal behaviors of the dataset [6], and each data item is given a score according to the changes in the model. High score indicates a high probability that the data item is an outlier. Distance-based outlier is the second category, which was presented by Ramasway et al. [7]. Outliers are identified by this top k data points after all data points in dataset are ranked by the distance to their kth nearest neighbor. Ramasway et al. gave effective algorithms for mining top k outliers. Jiang et al. [8] addressed cluster-based outlier detection, which is the third category. They considered the observations in small clusters as outliers. However, He et al. [9] proposed another technique, which found outliers based on clustering algorithm by incorporating semantic knowledge such as the class label of each data point in the dataset. From the class information viewpoint, the observations whose class labels are different from that of the majority of the cluster are considered semantic outliers. In a recent study [11], an outlier was defined from software engineering point of view by an expert in a clustering approach. For a given cluster, if a majority of software modules are classified as not fault-prone, the expert can make an educated assessment that the few fault-prone modules in the cluster are likely noisy data items and could be eliminated from the software measurement data. The forth category is rule-based outlier, [10]. An observation can be assigned to one and only one rule. Each rule is classified as FP (fault-prone) or NFP (not fault-prone), based on the probability of the instances associated with that rule being FP or NFP. If an instance is associated with a rule that has a different class label (FP or NFP) then that instance is likely to be an outlier

candidate. The work presented in this paper builds on the work presented in [1], where each single data item that contributes largely to the complexity of the learned hypothesis is considered an outlier.

The presence of noisy data in the training data set has been shown to hurt the predictive accuracy of data mining classifiers [3].

Generally speaking, there are three ways that data mining algorithms can handle noise. First, algorithms can be designed to be robust in the presence of noise. In this case, no specific noise detection steps are necessary; accuracy of the classifier on test data will not be significantly affected by the noise in the training data set. In the presence of significant amounts of noise, however, performance of these techniques may suffer regardless. In the second case, polishing techniques are designed to detect and correct noise in the data prior to training of the classifier [12]. Finally, noise can be detected and removed from the data prior to training of the classifier by using filtering techniques [3]. Noise handling m the second and third cases is a separate and distinct process from training the final classifier, with the major difference being that the second process will correct the cause of the noise, while the third will simply remove the noisy instance from the data set. The second method is preferred when the training data set has a small number of observations. In this case, the elimination of any observations will significantly reduce the size of the training data set with potentially adverse effects on the final classifier. The technique described in this paper fall in the third general class of noise detection algorithms.

The rest of this paper is organized as follows: Section 2 presents the methodology. Experimental evaluation is described in Section 3. Finally, conclusions are drawn in Section 4.

2. SATURATION FILTER

The saturation approach to noise filtering has its theoretical foundation in the saturation property of training data [14]. The algorithm, described in detail in [13], is outlined here for the sake of completeness. The description is based on that of the authors of the approach, presented in [2].

Suppose that a complexity measure c is defined and that for any hypothesis H its complexity $c(H)$ can be determined. Based on this complexity measure, for a training set E one can determine the complexity of the least complex hypothesis correct for all the examples in E; this complexity, denoted by $g(E)$ is called the CLCH value (Complexity of the Least Complex Hypothesis, correct for all the examples in E).

In [24] the authors show that if E is noiseless and saturated (containing enough training examples to find a correct target hypothesis), then $g(E) < g(E_n)$, where $E_n = E \cup \{e_n\}$ and e_n is a noisy example for which the target hypothesis is not correct. The property $g(E) < g(E_n)$ means that noisy examples can be detected as those that enable CLCH value reduction. The approach in an iterative form is applicable also when E_n includes more than one noisy example.

It must be noted that the saturation property of a training set is the main theoretical condition for the presented filter. In practice many domains have a restricted number of training examples and hence it can be assumed that these domains do not satisfy the saturation condition. Notice, however, that the described algorithm is applicable without changes also in this case. The reason is that for some sub-concept of the domain there may still be enough training examples so that this subpart of the domain is saturated; hence, a sub-theory description is induced by the learner whereas all other examples are eliminated since the learner will treat them as being erroneous or exceptions of the sub-theory description being learned.

The greatest practical problem of the saturation filter is the computation of the CLCH value $g(E)$ for a training set E. In rule-based induction, the hypothesis complexity measure $c(H)$ can be defined as the number of attribute value tests (literals) used in the hypothesis H. In this case, the corresponding $g(E)$ value can be defined as the minimal number of literals that are necessary to build a hypothesis that is correct for all the examples in E.

Suppose that the training set is contradiction free (there are no examples that differ only in their class value), and that the set of literals L defined in the hypothesis language is sufficient for finding a hypothesis H which is correct for all examples in E. Then the necessary and sufficient condition for a subset $L' \subseteq L$ *to* have this same property is that for every possible example pair, such that the first example in the pair is a positive example from E and the second one is negative, there must be at least one literal in L' which covers the pair. A literal covers a pair if it is true for the positive and false for the negative example in the pair. This fact enables that the $g(E)$ value defined by the minimal number of literals can be computed by any minimal covering algorithm over the set of example pairs. The ILLM heuristic minimal covering algorithm, presented here as Procedure 1 in Figure 1, is used. The advantages of this approach are: $g(E)$ computation does not require the actual construction of a hypothesis and the $g(E)$ value can be determined relatively fast. This approach presents the heart of the saturation noise filtering method used in this work.

The procedure starts with the empty set of selected literals L' (step 1) and the set U' of yet uncovered example pairs equal to all possible pairs of one positive and one negative example from the training set (step 3), for which in (step 2) weights $v(ei,ej)$ have been computed. The weight of a pair is high if the pair is covered by a small number of distinct literals from L. The meaning of this measure is that for a pair with a large weight it will be more difficult to find an appropriate literal which will cover this pair than for a pair with a small weight.

Each iteration of the main algorithm loop (steps 4 -11) adds one literal to the minimal set L' (step 9). At the same time, all example pairs covered by the selected literal are eliminated from U' (step 10). The algorithm terminates when U' remains empty. In each iteration, the algorithm tries to select the literal, which covers a maximal number of 'heavy' example

pairs (pairs with large weight). This is done so that a pair (e_a,e_b) is detected which is covered by the least number of literals (step 5).

Procedure 1: MINIMAL COVER
Input: U (set of example pairs), L (set of literals)
Output: L' (minimal set of literals)
(1) $L' \leftarrow \emptyset$
(2) **for** every $(e_i,e_j) \in U$ compute weights
$v(e_i,e_j) = 1/z$, where z is number of literals $l \in L$ that cover (e_i,e_j)
(3) $U' \leftarrow U$
(4) **while** $U' \neq 0$ **do**
(5) select (e_a,e_b)
$(e_a,e_b) \in U'$: $(e_a,e_b) = arg\ max\ v(e_i,e_j)$,
where *max* is over all $(e_i,e_j) \in U'$
(6) $L_{ab} \leftarrow \{l \mid l \in L$ covering $(e_a,e_b)\}$
(7) **for** every $l \in L_{ab}$ compute
$w(l) = \Sigma\ v(e_i,e_j)$, where *sum* is over all $(e_i,e_j) \in U'$ covered by l
(8) select literal l_s: $l_s = arg\ max\ w(l)$, where *max* is over all $l \in Lab$
(9) $L' \leftarrow L' \cup \{l_s\}$
(10) $U' \leftarrow U' \setminus \{$all (e_i,e_j) covered by $l_s\}$
(11) **end while**

Figure 1 Heuristic minimal covering algorithm

At least one of the literals from the set L_{ab} with literals that cover this pair (step 6), must be included into the minimal set L'. To determine which literal, weight $w(l)$ is computed for each of them (step 7) and the literal with the maximal weight is selected (step 8). The weight of a literal is the sum of the weights of example pairs that are covered by the literal.

Algorithm 1 in Figure 2 presents the saturation filter. It begins with the reduced training set E' equal the input training set E (step 1) and an empty set of detected noisy examples A (step 2). The algorithm supposes that the set of all appropriate literals L for the domain is defined. U represents a set of all possible example pairs where the first example in the pair is from the set of all positive training examples P' in the reduced set E', and the second example is from the set N' of all negative examples in the reduced training set E'. The algorithm detects one noisy example per iteration. The base for noise detection are weights $w(e)$ which are computed for each example e from E'. Initially all $w(e)$ values are initialized to 0 (step 6). At the end, the example with maximum weight $w(e)$ is selected (step 17). If the maximum $w(e)$ value is greater than the parameter ε_h predefined value then the corresponding training example is included into the set A (step 19) and eliminated from the reduced training set E' (step 20). The new iteration of noise detection begins with this reduced training set (steps 3-22). The algorithm terminates when in the last iteration no example has $w(e)$ greater than ε_h. Noisy examples in A and the noiseless E' are the output of the algorithm.

Computations in each iteration begin with the search for the minimal set of literals L' that cover all example pairs in U (calling Procedure 1 in step 5). A pair of examples is covered by a literal I if the literal is evaluated *true* for the positive example and evaluated *false* for the negative example in the pair. This step represents the computation of the $g(E')$ value. Next, a heuristic approach is used to compute weights $w(e)$ that measures the possibility that the elimination of an example e would enable $g(E')$ reduction.

Algorithm 1: *SaturationFilter(E)*
Input: E (training set), L (set of literals)
Parameter: ε_h (noise sensitivity parameter)
Output: A (detected noisy subset of E)

(1) $E' \leftarrow E$
(2) $A \leftarrow \emptyset$
(3) **while** $E' \neq \emptyset$ **do**
(4) find U, set of all possible example pairs for examples in E'
(5) call **Proc.1** to find min. L', $L' \subseteq L$ *so* that $\forall (e_i,e_j) \in U\ \exists l \in$ L' with the property l covers (e_i,e_j)
(6) initialize w(e) $\leftarrow 0$ for all e $\in E'$
(7) **for** every $l \in L'$ do
(8) $P^* \leftarrow \emptyset$, $N^* \leftarrow \emptyset$
(9) **for** every $(e_i,e_j) \in$ U
(10) **if** (e_i,e_j) covered by l and no other literal from L' **then**
(11) P* $\leftarrow$ P* $\cup \{e_i\}$, $N^* \leftarrow$ N $\cup \{e_j\}$
(12) **end for**
(13) **if** P* $= \emptyset$ **then** $L' \leftarrow$ L' $\setminus \{l\}$ and **goto** step 6
(14) **if** |P*| < |N*|
for every e $\in P^*$ **do**
$w(e) \leftarrow w(e) + \frac{1}{|P^*|}$
(15) **else**
for every e $\in$ N* **do**
$w(e) \leftarrow w(e) + \frac{1}{|N^*|}$
(16) **end for**
(17) select example e_s: e_s = arg max w(e),
where max; is computed over all $e \in E'$
(18) if $w(e_s) > \varepsilon_h$ **then**
(19) A $\leftarrow$ A $\cup \{e_s\}$
(20) $E' \leftarrow E' \setminus \{e_s\}$
(21) **else** exit with generated sets A and E'
(22) **end while**

Figure 2 Saturation filter algorithm

Weights $w(e)$ are computed so that for every literal I from L', minimal sets of positive (P*) and negative examples (N^*) are determined, such that if P* or N^* are eliminated from E' then I becomes unnecessary in L'. This is done in a loop (steps 9-12) in which every example pair is tested to determine if it is covered by a single literal I. If such a pair is detected (step 10) then its positive example is included into the set P* and its negative example into the set N^* (step 11). Literal elimination from L' presents the reduction of the $g(E')$ value. If a literal can be made unnecessary by elimination of a very small subset of training examples, then this indicates that these examples might be noisy. In steps 14 and 15, the $w(e)$ weights are incremented only for the examples which are the members of the P* and N^* sets. The weights are incremented by the inverse of the total number of examples in these sets. Weights are summed over all literals in L'. Step 13 is necessary because of the imperfectness of the heuristic minimal cover algorithm. Namely, if some $I \subseteq L'$ exists for which there is no example pair that is covered only by this literal (i.e., for which either P* $= \emptyset$ or $N^* = \emptyset$), this means that L' is actually not the minimal set because $L' \setminus \{I\}$ also covers all example pairs in U. In such case L' is substituted by $L' \setminus \{I\}$.

The presented saturation filter uses the parameter ε_h, that determines noise sensitivity of the algorithm. The parameter can be adjusted by the user in order to tune the algorithm to

the domain characteristics. Reasonable values are between 0.25 and 2. For instance, the value 1.0 guarantees the elimination of every such example by whose elimination the set L' will be reduced for at least one literal. Lower ε_h values mean greater sensitivity of the algorithm (i.e., elimination of more examples): lower ε_h values should be used when the domain noise is not completely random, and when dealing with large training sets (since statistical properties of noise distribution in large training sets can have similar effects). In ILLM the default values of ε_h are between 0.5 and 1.5, depending on the number of training examples in the smaller of the two subsets of *E:* the set of positive examples *P* or the set of negative examples *N*. Default values for the saturation filter's noise sensitivity parameter ε_h are: 1.5 for training sets with 2-50 examples, 1.0 for 51-100 examples, 0.75 for 101-200 examples, and 0.5 for more than 200 examples.

The approach of selecting whether to eliminate the positive or the negative example in the pair based on the number of literals which cover them, proved to lead to biased behavior of the filter in the case study presented in the next section. The examples pertaining to the minority class will more likely form a smaller set of examples that can be eliminated to make the selected literal unnecessary. Thus, it is more likely that the condition in line 14 will cause the members of the minority class to be selected for removal from the dataset. As the case study shows, when the dataset is unbalanced the filter tends to concentrate only on removing the members of the minority class. Since this is not consistent with the preposition that the dataset incorporates random noise, the criterion for selecting whether the positive or negative set of examples should be eliminated has been revised to accommodate unbalanced datasets. Instead of merely comparing the sizes of the of P* and N* datasets, their values were normalized using the prior probabilities of an example being positive or negative and a scaling constant which was used as an additional parameter of the filter. The criterion in line 14 of Figure 2 has been changed to:

$$\frac{|P^*|}{\pi_p} \cdot C < \frac{|N^*|}{\pi_n}$$

where:

π_p – prior probability of the sample being positive,

π_n – prior probability of the sample being negative,

C – scaling constant.

3. EXPERIMENTAL EVALUATION

The software metrics and quality data used on our study is that of a NASA software project, JM1, written in C++.

The software metrics and quality data used in our study is that of a NASA software project, JM1, written m C++. The data was made available through the Metrics Data Program (MDP) at NASA, and included software measurement data and associated error (fault) data collected at the function / subroutine / method level. The dataset used consisted of 188 modules, of which 55 modules contained errors (ranging from 1 to 13) while the remaining 414 modules were error-free, i.e., had no software faults. A module with no faults was considered NFP, and FP otherwise. Each module in the JM1 project was characterized by 21 software measurements: four McCabe metrics (Cyclomatic_Complexity, Essential_Complexity, Design_Comlexity, and Loc_Total); eight derived Halstead metrics (Halstead_Length, Halstead_Volume, Halstead_Level, Halstead_Difficulty, Halstead_content, Halstead_Effort, Halstead_Error_Est, and Halstead_Prog_Time); four metrics of Line Count (Loc_Executable, Loc_Comment, Loc_Blank, and Loc_Code_And_Comment); four basic Halstead metrics (Unmque_0perators, Unique_0perands, Total_0perators, and Total_0perands); and one metric for Branch Count. The quality of the modules was described by their Error Rate, i.e., number of defects in the modules, and Defect, whether or not the module has any defects. The latter was used as the class label. We only used the 13 primitive metrics and module class was used in the analysis, all of which were of quantitative type. The eight derived Halstead metrics were not used.

The filtering has been applied a number of times, with different values of the filter sensitivity parameter. The datasets obtained in this way have been used to train the C4.5 classifier. The classifier obtained for each dataset has then been evaluated both by 10 fold cross-validation and witholding half of the dataset as test dataset, which has not been used to train the classifier.

The results obtained through cross-validation are presented in Table 2:

Th	Overall Error	Type I Error (%)	Type II Error (%)	Leaves	Nodes	NFP elim. (%)	FP elim. (%)
0.5	9.7222	2.2556	100	4	7	0	94.5454
0.6	9.7222	2.2556	100	4	7	0	80
0.7	9.7222	2.2556	100	4	7	0	80
0.8	9.7222	2.2556	100	4	7	0	80
0.9	13.0719	5.2631	65	5	9	0	80
1	14.3678	11.2781	24.3902	2	3	0	63.6363
1.1	14.3678	11.2781	24.3902	2	3	0	25.4545
1.2	18.0791	17.2932	20.4545	9	17	0	25.4545
1.3	16.1111	12.7819	25.5319	10	19	0	23.6363
1.4	16.2162	13.5338	23.0769	2	3	0	14.5454
1.5	13.369	12.0300	16.6666	11	21	0	5.45454
1.6	13.369	12.0300	16.6666	11	21	0	1.81818
1.7	13.369	12.0300	16.6666	11	21	0	1.81818
1.8	13.369	12.0300	16.6666	11	21	0	1.81818
1.9	13.369	12.0300	16.6666	11	21	0	1.81818
2	15.4255	14.2857	18.1818	8	15	0	0

Table 2 Cross-validation results obtained for the dataset containing 188 instances.

The results show the overall misclassification error of the classifier as well as the rate of misclassification of NFP modules as FP (Type I error) and misclassification of FP modules as NFP (Type II error). The latter being more serious misclassification. The table also shows the complexity of the generated C4.5 tree, listing the number of nodes in it as well as the number of leaves of the tree. The table also shows the portion of FP and NFP modules eliminated.

The results obtained in this way showed that C4.5 models built based on the filtered dataset behaved better only when the filter removed a single instance from the dataset. It is unlikely that the dataset in question contains such a small amount of outliers. However the C4.5 algorithm did actually generate a much smaller decision tree for the case when 20% of FP modules have been eliminated from the dataset. Such a model is much less likely to be over-fitted than the one generated using the initial dataset.

The experiments also showed a distinct inclination of the saturation filter to the removal of the FP instances, as member of a minority (FP) class. This is inconsistent with the hypothesis that the dataset contains random noise. Such bias of the filter cannot be attributed to the failure of the dataset to

incorporate the data allowing for the forming of complete and consistent hypothesis, rather it is likely that it is a result of the bias of the heuristic algorithm used to determine which data items should be eliminated.

Separate test dataset based validation results are shown in Table 3.

Threshold	Overall Error (%)	Type I Error (%)	Type II Error (%)
0.5	20.2128	10.60606	42.85714
0.6	20.2128	10.60606	42.85714
0.7	20.2128	10.60606	42.85714
0.8	20.2128	10.60606	42.85714
0.9	19.1489	7.575758	46.42857
1	17.0213	21.21212	7.142857
1.1	17.0213	21.21212	7.142857
1.2	14.8936	3.030303	42.85714
1.3	18.0851	9.090909	39.28571
1.4	18.0851	24.24242	3.571429
1.5	17.0213	7.575758	39.28571
1.6	17.0213	7.575758	39.28571
1.7	17.0213	7.575758	39.28571
1.8	17.0213	7.575758	39.28571
1.9	17.0213	7.575758	39.28571
2	14.8936	7.575758	32.14286

Table 3 Test dataset based results obtained for the dataset with 188 instances

The approach of modifying the selection criterion by adding a scaling constant, which would be an additional parameter of the algorithm allowed for the possibility of making the filter more balanced, however it was then necessary to vary two parameters to achieve the desired results (the ratio of positive and negative examples eliminated should be in the vicinity of ratio of the prior probabilities of an example being positive or negative, respectively). For the threshold of Th = 0.3975 and the scaling constant C = 54.9 a a balanced filter was achieved by removing 13 positive and 31 negative examples. The overall, Type I and Type II error obtained through cross-validation were 20.14, 22.55 and 14.29, respectively. The results for the test dataset were 21.28, 28.79 and 3.6.

The model constructed for this dataset achieved significantly lower Type II error rate than the model built based on the initial dataset. Also the model is significantly simpler having only tree nodes instead of fifteen, which is a very desirable characteristic. The test dataset validation results show that the model generated by the C4.5 algorithm, although much simpler, achieved far better Type II error rate, which is a significant result bearing in mind that the C4.5 algorithm is a robust algorithm designed to handle noise well.

4. CONCLUSION

In the work presented, the saturation filter approach to noise elimination has been applied to a domain it has not been used in before. Specifically, it has been applied to the software quality improvement domain. An inherent bias of the original approach has been observed and an approach devised to alleviate the problem. The modified saturation filter was able to exhibit balanced behavior and improve the performance of a robust algorithm such as C4.5.

An improvement of the saturation filter approach has been proposed. The parameters of the new filter have been determined manually to achieve the desired behavior. Presently, there is no mechanism of adjusting the parameters of the filter automatically. This is the direction in which further research is possible.

It has been demonstrated that the modified saturation filter is capable of achieving results in the case study at hand, something that the original saturation filter was not capable.

REFERENCES

[1] D. Gamberger; N. Lavrac; S. Dzeroski: Noise elimination in inductive concept learning: A case study in medical diagnosis. In *Proc. of the 7' International Workshop on Algorithmic Learning Theory*. Springer, Berlin, 1996, pp. 199 - 212.

[2] D. Gamberger; N. Lavrac; C. Groselj: Experiments with noise filtering in a medical domain. In *Proc. of the 16th International Conference on Machine Learning*, Morgan Kaufmann, 1999, pp. 143 - 153

[3] T.M. Khoshgoftaar; L.A.Bullard; K. Gao: Detecting Outliers Using Rule-Based Modeling for Improving CBR-Based Software Quality Classification Models. *Cased-Based Reasoning Research and Development, LNAI 1689,* Springer-Verlag, 2003, pp. 216-230.

[4] Weisberg, S. (1985). *Applied Linear Regression.* John Wiley &: Sons.

[5] John, G.H. (1995). Robust decision trees: Removing outliers from data. In *Proc. of the 1st Int. Conference on Knowledge Discovery and Data Mining,* 174-179. AAI Press.

[6] K. Yamanish, J. Takeuchi, G. Williams, *On-line unsupervised outlier detection using finite mixtures with discounting learning algorithm.* In Proceedings of KDD'00, Boston, MA, USA, 2000. pp.320-325.

[7] S. Ramasway, R. Rastogi, S. Kyuseok, *Efficient algorithms for mining outliers form large data sets.* In Proceedings ofSIGMOD'00, Dallas, Texas, 2002, pp. 93-104.

[8] M. Jiang, S. Tseng, C. M. Su. *Two-phase clustering process for outliers detection.* Pattern Recognition Letters, 22(6/7), 2001, pp.691-700.

[9] Z. He, S. Deng, X. Xu. *Outlier detection integrating semantic knowledge.* In Proc. of the 3th Int'1 Conf. On Web-Age Information management, Beijing, China, 2002, pp.126-131.

[10] T. M. Khoshgoftaar, L. A. Bullard, K. Gao. *Detecting outliers using rule-based modeling for improving CBR-based software quality classification models.* Cased-Based Reasoning Research and Development, Springer-Verlag, 2003, pp.216-230.

[11] S. Zhong, T. M. Khoshgoftaar, N. Seliya. *Analyzing software measurement data with clustering techniques.* IEEE Intelligent Systems, vol. 19 Issue 2, March 2004. pp.20 – 27

[12] C.M. Teng. Correcting Noisy Data. *Proceedings 16th International Conference on Machine Learning.* Morgan Kaufmann, 1999, pp. 239 - 248.

[13] Gamberger, D., Lavrac, N. & Dzeroski, S. (1996). Noise elimination in inductive concept learning: A case study in medical diagnosis. In *Proc. of the 7th International Workshop on Algorithmic Learning Theory,* 199-212. Springer, Berlin.

[14] Gamberger, D. & Lavrac, N. (1997). Conditions for Occam's razor applicability and noise elimination. In *Proc. of the 9th European Conference on Machine Learning,* 108-123. Springer, Berlin.