Improved Mining of Software Complexity Data on Evolutionary Filtered Training Sets

VILI PODGORELEC
Institute of Informatics, FERI
University of Maribor
Smetanova ulica 17, SI-2000 Maribor
SLOVENIA
vili.podgorelec@uni-mb.si http://lisa.uni-mb.si/vili/

Abstract: - With the evolution of information technology and software systems, software reliability has become one of the most important topics of software engineering. As the dependency of society on software systems increase, so increases also the importance of efficient software fault prediction. In this paper we present a new approach to improving the classification of faulty software modules. The proposed approach is based on filtering training sets with the introduction of data outliers identification and removal method. The method uses an ensemble of evolutionary induced decision trees to identify the outliers. We argue that a classifier trained by a filtered dataset captures a more general knowledge model and should therefore perform better also on unseen cases. The proposed method is applied on a real-world software reliability analysis dataset and the obtained results are discussed.

Key-Words: - data mining, classification, evolutionary decision trees, filtering training sets, software fault prediction, search-based software engineering

1 Introduction

The early and accurate identification of potentially dangerous (faulty) software modules is of vital importance for better software reliability. Reliability is one of the most important aspects of software systems of any kind (information systems, embedded systems, etc.). The size and complexity of software is growing dramatically during last decades. The demand for highly complex software systems is increasing more rapidly than the ability to design, implement and maintain them.

When the requirements for and dependencies of computers increase, the possibility of crises from failures also increases. The impact of these failures ranges from inconvenience to economic damages to loss of lives – therefore it is clear that software reliability is becoming a major concern not only for software engineers and computer scientists, but also for the society as a whole. Therefore, the employment of an efficient fault predictive technique to foresee dangerous software modules is essential.

With the creation of large empirical databases of software projects, as a result of stimulated research on estimation models, metrics and methods for measuring and improving processes and products, intelligent mining of these datasets can largely add to the improvement of software reliability [1]. In order to help the software engineers in predicting the faulty software modules, computerized data mining and decision support tools can be used which are able to help software engineers to process a huge amount of data available from previous software projects and suggest the probable prediction based on the values of several important attributes. Black-box classification methods (neural networks for example) are not very appropriate for this kind of task, because the software experts want to evaluate and validate the decision making process induced by those tools, before there is enough trust to use the tools in practice.

On the other hand, the evaluation of the induced classification rules produced by the computerized tools by a software expert can be an important source of new knowledge on the associations of the available attributes and new “laws” of software reliability engineering. In order to achieve this goal, the classification process should be easily understandable, interpretable and straightforward. One of the most popular and proven-useful approach are decision trees. However, it has been shown that decision trees are a weak classifier, prone to produce very different solutions based on an even small change in input (training) data. Therefore, inaccuracies and noise in training data can easily lead to an inaccurate result.

The idea that we present in this paper is to construct an outlier prediction method that filters out so-called data outliers, i.e. data items that fall outside the boundaries that enclose most other data items in the data set [1, 2]. When a filtered dataset is used to train a classifier (a decision tree) it should produce a better and more reliable classification result. Furthermore, as a consequence of increased homogeneity of the data, the results should be more general and thus simpler, less complex and easier to apply in general. Similar method has already been applied to data mining in medicine with some success [3].

1.1 The aim and scope of the paper

The main objective of this paper is to investigate whether it is possible to improve the classification performance of several machine learning algorithms when the outliers are filtered out of the training set. For the outlier prediction method a set of evolutionary induced decision trees is used. The proposed approach is applied to a software engineering problem of predicting potentially dangerous (containing errors and faults) software modules. The data mining results (both quantitative classification results and qualitative assessment of induced models) of different machine learning algorithms are presented and discussed.

2 Filtering the dataset by outlier prediction method

The basic idea of outlier prediction is to define a criterion or criteria, upon which for each data case from a dataset can be determined whether it belongs to the majority of the cases or not. If the specific data case regarding the defined criteria does not belong to the majority, then it is called an outlier (regarding specific criteria). How the criteria are defined determines the outlier prediction method. In general, defining outlier criteria is not trivial, whereas the identification of outliers based on these criteria is.

In our previous work we presented how to use evolutionary algorithm for the induction of decision trees [4, 5, 6]. One of the greatest advantage of evolutionary construction of decision trees, beside the proven efficiency in classifying, is the ability to produce several comparably accurate classifiers for the same dataset. Having this in mind, it is possible to get a decision for the same data case based on different classifiers (using different attributes and/or different relations).

Our proposition for the prediction of outliers is the following: if a single (known) data case is classified differently by different accurate classifiers, it potentially contains contradictory information (Figure 1). Although this contradiction is not necessarily an error in data, a usual decision tree classifier is not able to correctly construct a general model based on such data. Therefore, our proposition is that the general knowledge model built by a decision tree (or some other induction method in that matter) would be more efficient when a classifier would be trained without such misleading data.

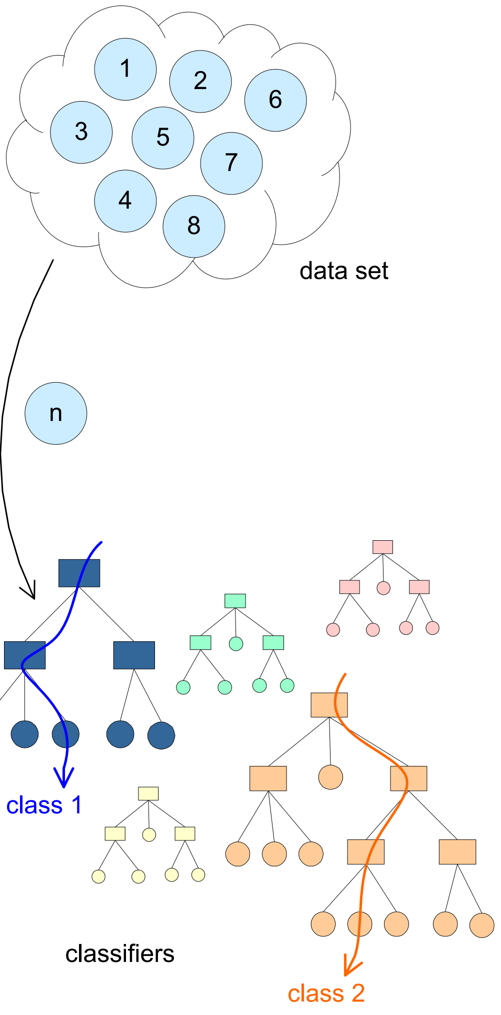

Fig 1. When a single data record is classified differently by different classifiers, it potentially contains contradictory information.

In our method, we first define an approach to the identification of outliers. For this purpose the algorithm for the construction of decision trees *genTrees* is used (described in the next section). With *genTrees* a set of classifiers (decision trees) are induced. For each training object (data case from the training set) classification $cc_i(x)$ is calculated for all decision classes (Eq. 1) – the resulting value represents the number of classifiers that classified object *x* with the decision class *i*. Then classification confusion score *CCS(x)* is calculated for each training object x (Eq. 2) – the result represents the confusion score of a set of classifiers when classifying object *x*; if all DTs give the same classification, then the result is *0*; higher numbers represent less homogeneous objects for classification – possible outliers. The higher is the number *CCS(x)*, more probably lies the object *x* outside the majority area. Based on the *CCS* (Eq. 2) it is determined which objects should be filtered out from the dataset for the classification process. For this purpose a tolerance threshold *tt* is defined for a dataset; if *CCS(x)>tt* then object *x* is filtered out.

$$cc_i(x) = \sum_{j=1}^{num\_DTs} \begin{cases} 1; class(DT_j, x) = i \\ 0; otherwise \end{cases} \quad ;\forall i, i \in (1..num\_classes) \tag{1}$$

$$CCS(x) = \sum_{i=1}^{num\_classes} \left(\frac{cc_i(x)}{cc_{\max}(x)} \right)^2 - 1 \tag{2}$$

3 Backend technologies: evolutionary algorithms and decision trees

The central point of our approach, as has been said above, is the use of evolutionary induced decision trees which are being used to construct a set of classifiers for identifying outliers in a training set. Both backend technologies enabling our approach are briefly presented first, following by the description of our algorithm for inducing decision trees.

3.1 Evolutionary algorithms

Genetic algorithms are adaptive heuristic search methods which may be used to solve all kinds of complex search and optimisation problems [7, 8]. They are based on the evolutionary ideas of natural selection and genetic processes of biological organisms. Like natural populations evolve according to the principles of natural selection and "survival of the fittest", first laid down by Charles Darwin, so by simulating this process, genetic algorithms are able to evolve solutions to real-world problems, if they have been suitably encoded [9]. They are often capable of finding optimal solutions even in the most complex of search spaces or at least they offer significant benefits over other search and optimisation techniques. The basic mechanism of evolutionary algorithms is presented in Figure 2.

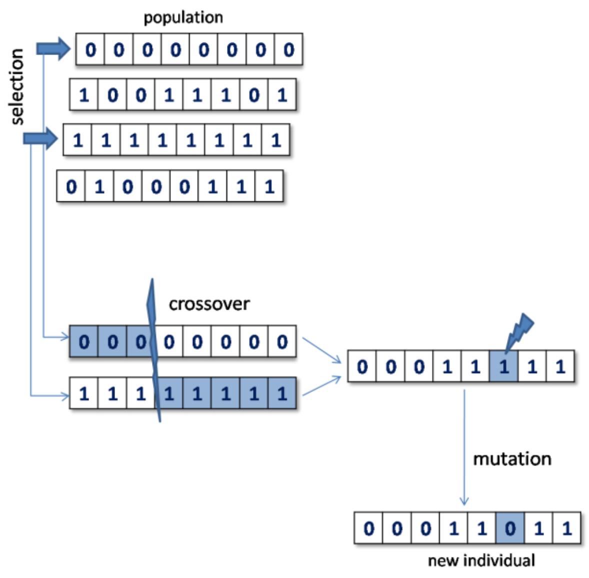

Fig 2. The basic mechanism of evolutionary algorithms: selection, crossover and mutation.

A typical genetic algorithm operates on a set of solutions (population) within the search space. The search space represents all the possible solutions which can be obtained for the given problem, and is usually very complex or even infinite. Every point of the search space is one of the possible solutions and therefore the aim of the genetic algorithm is to find an optimal point or at least come as close to it as possible.

The genetic algorithm consists of three genetic operators: selection, crossover (recombination), and mutation. Selection is the survival of the fittest individuals within the genetic algorithm with the aim of giving the preference to the best ones. For this purpose all solutions have to be evaluated, which is done with the use of the evaluation function. Selection determines individuals to be used for the second genetic operator - crossover or recombination, where from two good individuals a new, even better one is constructed. The crossover process is repeated until the whole new population is completed with the offspring produced by the crossover. All constructed individuals have to preserve the feasibility regarding the given problem; in this manner it is important to coordinate internal representation of individuals with genetic operators. The last genetic operator is mutation, which is an occasional (low probability) alteration of an individual that helps to find an optimal solution to the given problem faster and more reliably.

3.2 Decision trees

Inductive inference is the process of moving from concrete examples to general models, where the goal is to learn how to classify objects by analysing a set of instances (already solved cases) whose classes are known. Instances are typically represented as attribute-value vectors. Training input consists of a set of such vectors, each belonging to a known class, and the output consists of a mapping from attribute values to classes. This mapping should accurately classify both the given instances and other unseen instances.

A decision tree (DT) [5, 9] is a formalism for expressing such mappings and consists of tests or attribute nodes linked to two or more sub-trees and leafs or decision nodes labeled with a class which means the decision. A test node computes some outcome based on the attribute values of an instance, where each possible outcome is associated with one of the sub-trees. An instance is classified by starting at the root node of the tree. If this node is a test, the outcome for the instance is determined and the process continues using the appropriate sub-tree. When a leaf is eventually encountered, its label gives the predicted class of the instance. An example of a DT is presented in Figure 3.

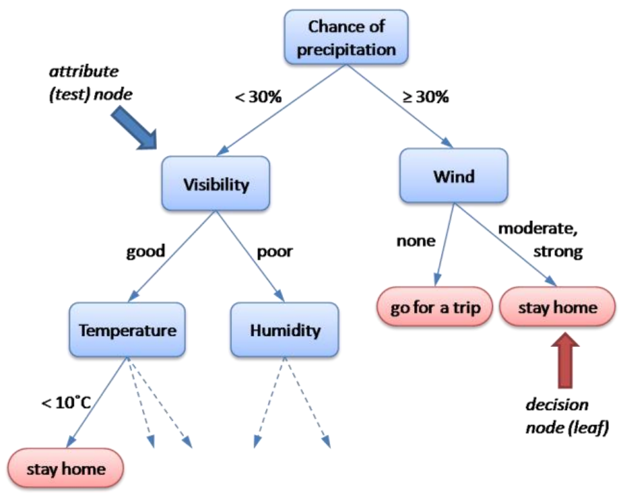

Fig 3. An example of a decision tree.

DTs have been already used in a large amount of different applications: in medicine [5], finance [11], environmental studies [12], etc. Their greatest advantage is comparatively high classification accuracy while preserving the transparent knowledge model which can be easily interpreted and validated by domain experts. This has made DTs one of the most popular classifiers.

3.3 Evolutionary induction of decision trees: the *genTrees* algorithm

Evolutionary algorithms (EAs) are generally used for very complex optimization tasks [8], for which no efficient deterministic or heuristic method is developed. Construction of DTs is a complex task, but an exact method exists, that in general works efficiently and reliably [5, 9]. At a first glance there is no reason to use EAs. Nevertheless, there are some objective reasons that justify our evolutionary approach. First, EAs provide a very general concept that can be used in all kinds of decision making problems. Because of their robustness they can be used also on incomplete, noisy data (which often happens because of measurement errors, unavailability of proper instruments, etc.). This is not very successfully solvable by traditional techniques of DT construction. Furthermore, EAs use evolutionary principles to evolve solutions, therefore solutions can be found that can be easily overlooked otherwise. Another important advantage of EAs is the possibility of optimizing the decision tree's topology and adapting class intervals for numeric attributes, within the evolution process. And the most important advantage of the evolutionary approach in our case is that not only one but several equally qualitative solutions are obtained for the same dataset. In this way the same decision can be made based on different attributes and/or combination of the attributes. By different settings of parameters in EA runs, searching can be directed to different situations that give us different solutions for our final decision forest.

Of course there are also some drawbacks of our method regarding the heuristic induction, the most obvious one being the higher induction time complexity.

When defining the internal representation of individuals within the genetic population, together with the appropriate genetic operators that will work upon the population, it is important to assure the feasibility of all solutions during the whole evolution process. We decided to present an individual directly as a DT. In this manner all intermediate solutions are feasible, no information is lost because of the conversion between internal representation and the DT, the fitness function (FF) can be straightforward, etc. The problem with direct coding of solution may bring some problems in defining of genetic operators. As DTs may be seen as a kind of simple computer programs (with attribute nodes being conditional clauses and decision nodes being assignments) we decided to define genetic operators similar to those used in genetic programming where individuals are computer program trees [10].

The induction of DT-like classifiers with the use of evolutionary algorithms has been already used in several cases [5, 14, 15], also applied to software engineering [6].

3.3.1 Induction of DTs for the initial population

First step of the genetic algorithm is the induction of the initial population. A random decision tree is constructed based on the following algorithm:

1. input: number of attribute nodes *V* that will be in the tree
2. select an attribute A_i from the set of all possible *K* attributes and define it a root node *tn*
3. in accordance with the selected attribute's A_i type (discrete or continuous) define a test for this node *tn*:
    a. for continuous attributes in a form of $f_{tn}(A_i) < \phi_i$, where $f_{tn}(A_i)$ is the attribute value for a data object and ϕ_i is a split constant
    b. for discrete attributes two disjunctive sets of all possible attribute values are randomly defined
4. connect empty leaves to both new branches from node *tn*
5. randomly select an empty leaf node *tn* (note below)
6. randomly select an attribute A_i from the set of all possible *K* attributes (note below)
7. replace the selected leaf node *tn* with the attribute A_i and go to step 3
8. finish when *V* attribute nodes has been created

In the above algorithm:

- the probability of selecting an empty leaf is decreased with the depth of the leaf in a growing tree, and
- the probability of choosing an attribute depends on a number of previous uses of that attribute in a tree – in this manner unused attributes have better chances to be selected.

A simplified pseudo-code of the above algorithm is presented at Figure 4.

```
// initial decision trees
Select number of nodes N
repeat
  Select an attribute Xi
  Define $f_t(Xi)<\Phi_i$ or $f_t(X_i) \in V$
  Connect empty leaves to node
  Randomly select an empty leaf node
until N nodes are created
```

Fig 4. The basic algorithm for constructing random DTs.

For each empty leaf the following algorithm determines the appropriate decision class: let *S* be the training set of all training objects *N* with *M* possible decision classes ω_1, .., ω_M and N_i is the number of objects within *S* of a class ω_i. Let S^{tn} be the sample set at node *tn* (an empty leaf for which we are trying to select a decision class) with N^{tn} objects; N_i^{tn} is the number of objects within S^{tn} of a decision class ω_i. Now we can define a function that measures a potential percentage of correctly classified objects of a class ω_i:

$$F(tn,i) = \frac{N_i^{tn}}{N_i} \qquad (3)$$

Decision ω_i^{tn} for the leaf node *tn* is then marked as the decision ωi, for which *F*(*tn*,*i*) is maximal.

The ranking of an individual DT within a population is based on the FF:

$$FF = \sum_{i=1}^{M} w_i \cdot (1 - acc_i) + \sum_{i=1}^{V} c(tn_i) + w_u \cdot nu \qquad (4)$$

where *M* is the number of decision classes, *V* is the number of attribute nodes in a tree, acc_i is the accuracy of classification of objects of a specific decision class ω_i, w_i is the importance weight for classifying the objects of the decision class ω_i, $c(tn_i)$ is the cost of using the attribute in a node tn_i, *nu* is number of unused decision (leaf) nodes, i.e. where no object from the training set fall into, and w_u is the weight of the presence of unused decision nodes in a tree.

According to FF the best trees (the most fit ones) have the lowest function values – the aim of the evolutionary process is to minimize the value of FF for the best tree. A near optimal DT would: 1) classify all training objects with accuracy in accordance with importance weights w_i (some decision classes may be more important than the others), 2) have very little unused decision nodes (there is no evaluation possible for this kind of decision nodes regarding the training set), and 3) consist of low-cost attribute nodes (in this manner the desirable/undesirable attributes can be prioritized).

3.3.2 Crossover and mutation

Crossover works on two selected individuals as an exchange of two randomly selected sub-trees. In this manner a randomly selected training object is selected first which is used to determine paths (by finding a decision through the tree) in both selected trees. Then an attribute node is randomly selected on a path in the first tree and an attribute is randomly selected on a path in the second tree. Finally, the sub-tree from a selected attribute node in the first tree is replaced with the sub-tree from a selected attribute node in the second tree and in this manner an offspring is created which is put into a new population.

Mutation consists of several parts: 1) one randomly selected attribute node is replaced with another attribute, randomly chosen from the set of all attributes; 2) a test in a randomly selected attribute node is changed, i.e. the split constant is mutated; 3) a randomly selected decision (leaf) node is replaced by an attribute node; 4) a randomly selected attribute node is replaced by a decision node.

```
// evolving DTs
repeat
  tournament selection
  crossover
  mutation
  evaluate all DTs with FF
until termination criteria reached
```

Fig 5. The basic algorithm for evolving DTs.

With the combination of presented crossover that works as a constructive operator towards local optimums, and mutation that works as a destructive operator in order to keep the needed genetic diversity, the searching for the solution tends to be directed toward the globally optimal solution. In this manner the optimal solution is the most appropriate DT regarding our specific needs (expressed in the form of FF). As the evolution repeats, more qualitative solutions are obtained regarding the chosen FF. The evolution stops when an optimal or at least an acceptable solution is found or if the fitness score of the best individual does not change for a predefined (large) number of generations.

A simplified pseudo-code algorithm for the evolutionary process of improving the quality of a DT (in accordance with FF) is presented at Figure 5.

4 Application of the method

The described training set filtering method has been applied to a real-world software reliability analysis dataset, composed at the University of Udine, Italy, from the software development project of a hospital information system – the whole medical software system consists of 904 modules in C programming language representing more than 2.000.000 lines of code.

First the modules have been identified either as OK or DANGEROUS by applying the model developed by Pighin [11] – the modules that contain less than 5 errors were set as OK and the others as DANGEROUS. A set of 168 attributes, containing various software complexity measures, has been determined for each software module. From all 904 modules 804 have been randomly selected for the training set, and the remaining 100 modules have been selected for the testing set.

A set of classifiers were induced with *genTrees* based on the training set. Then outliers were identified using the class confusion score metrics (Eq. 2), which were removed from the original training set in order to get a filtered training set. Finally, some well-known classification algorithms were used on both original and filtered training set in order to compare classification results. The following classification algorithms have been used: AREX [12], ID3, C4.5 [13, 14], Naïve-Bayes (N-B), instance-based classifier (IB, i.e. *k*-nearest neighbors), and logistic regression (LogReg) [15, 16]. All the results are the averages of 10-fold cross-validation.

4.1 A Dataset

The used software complexity measures dataset (SCM dataset) has been carefully composed from a well-prepared protocol [17]. The starting point for the analysis was the definition and measurement of a set of experimental attributes connected to the structure of software products after the code phase. Such parameters may, for example, be the total number of lines of code and lines of comments, the occurrence of various types of instructions, the operators and the types of data used. For each software module, moreover, the fault signals up to the moment when measurement started were considered. By faults all the malfunctions encountered during the internal test phase and after the release of the software are meant. In our experimental environment the code phase included a preliminary test of modules. After this phase the modules went on to the real test session, in which faults were signaled and measurement started.

What had to be dealt with was whether the chosen set of parameters would be sufficiently large to identify the structure of a program. Accordingly it has been started with a very large set of parameters. These parameters were affected by the persistence of multicollinearity. It was necessary to reduce the total number of parameters to a smaller set of independent parameters. This was achieved by statistical procedures eliminating those which were heavily dependent on other parameters in explaining the presence of faults in a code, or which were completely irrelevant. A subset of 168 structural parameters was defined which the factorial analysis identified as being reasonably free from multicollinearity, plus the dependent variable, the number of faults. The basic properties of the SCM dataset are presented in Table 1.

Table 1. The basic properties of the SCM dataset.

number of cases	904
number of attributes	168
- nominal	1
- continuous	167
number of decision classes	2
decision classes distribution	76.4%; 23.6%

4.2 Quantitative results

As described above, after filtering out the outliers by the proposed outlier prediction method, the reduced data set has been used by some well-known classification methods. When using the same training set and the same parameter setting, the algorithms produce the same – deterministic results (for example C4.5 algorithm uses entropy measures for the induction of decision trees, etc). In this manner, the difference in achieved classification effectiveness on a testing set can be objectively compared between the induced classifiers trained by either the original, non-filtered training set or the filtered training set. The classification results are presented in tables 2 and 3 and on figures 4 and 5.

For the SCM dataset five classifiers were induced and the tolerance threshold selected *tt=0.2*; if none or only one classifier (out of five) misclassified an object, then the object was not identified as an outlier. Altogether 29 objects (from 804 in the original training set) were removed from the training set (3.6% removal).

Table 2. Average classification accuracies on the SCM training dataset.

classification algorithm	accuracy on the training set [%]	
	original set	*filtered set*
AREX	80.10	82.20
C4.5	95.60	96.50
ID3	100.00	100.00
IB	100.00	100.00
Naïve-Bayes	72.14	73.16
LogReg	92.04	91.61

Table 3. Average classification accuracies on the SCM testing dataset.

classification algorithm	accuracy on the training set [%]	
	original set	*filtered set*
AREX	80.10	82.20
C4.5	95.60	96.50
ID3	100.00	100.00
IB	100.00	100.00
Naïve-Bayes	72.14	73.16
LogReg	92.04	91.61

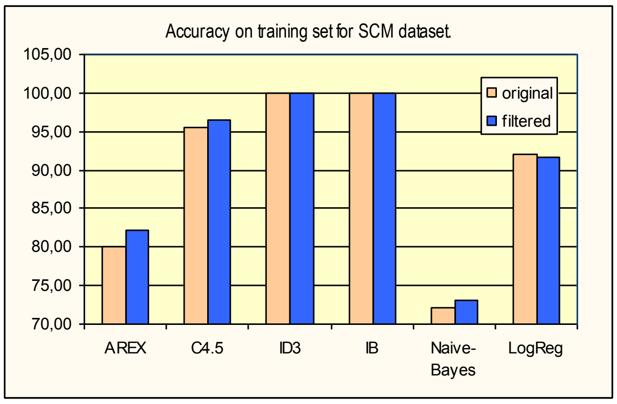

Fig. 4. Classification results on the SCM training dataset.

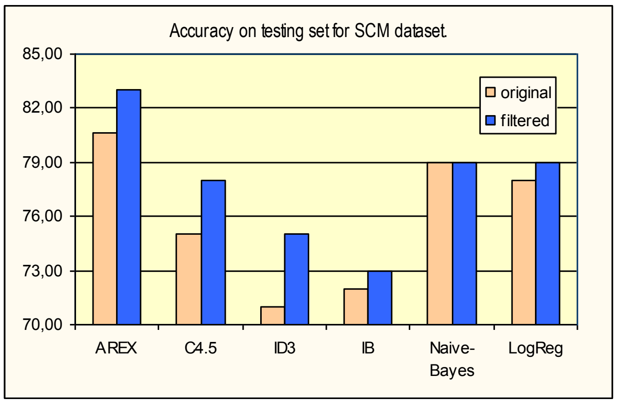

Fig. 5. Classification results on the SCM testing dataset

All the results presented are based on the 10-fold cross validation for each algorithm on a training set and a testing set. The classification results of the method AREX are based on 10 independent evolutionary runs for each fold.

4.3 Qualitative results

The possibility of accurate predictions of the potentially dangerous software modules based on the software complexity attributes is very important for software engineers in order to improve the software reliability. The proposed classification method proves to be effective in performing this task. However, the "knowledge" (i.e. combinations of the used attributes) used to make the predictions would be of an immense importance in order to decrease the possibility of the problems to arise in the first place. Therefore, the built knowledge models (like decision trees or decision rules) should be studied to find this knowledge.

An interesting phenomenon that arose with the filtering of the identified outliers from the original training set is the fact, that the built knowledge models based upon the filtered learning set were much less complex than those built on the original, non-filtered learning set. This means that less attributes were used to predict the faulty modules and the overall models were simpler, smaller and less complex – easier to interpret. For the comparison on Figure 6 there are two decision trees built on the original, non-filtered learning set, and on Figure 7 there are a few decision trees built on the filtered learning set; the accuracy of all the decision models are pretty much the same. We can see that in the case of filtered training set, the resulting classifiers are almost as simple as simple rules, whereas the classifiers induced on the original training set are much more demanding to interpret.

```
alpha
  |--[<0.63233] OK
  |--[>=0.63233] signif_of_comments
     |--[<1480.0400] StrCtrl_lines
     |  |--[<44.5500] OK
     |  |--[>=44.5500] DANGEROUS
     |--[>=1480.0400] selection_instr
        |--[<3.799] DANGEROUS
        |--[>=3.799] formal_function_params
           |--[<19.162] DANGEROUS
           |--[>=19.162] signif_of_comments
              |--[<3026.5080] OK
              |--[>=3026.5080] DANGEROUS
```

```
words_of_comments
  |--[<188.769] break
  |  |--[<12.282] function_calls_to_funcs
  |  |  |--[<11.985] vect_function_args
  |  |  |  |--[<25.088] const_with_#define
  |  |  |  |  |--[<41.584] formal_func_pars
  |  |  |  |  |  |--[<8.134] OK
  |  |  |  |  |  |--[>=8.134] DANGEROUS
  |  |  |  |  |--[>=41.584] DANGEROUS
  |  |  |  |--[>=25.088] fileType
  |  |  |     |--[c,r,h] OK
  |  |  |     |--[s] DANGEROUS
  |  |  |--[>=11.985] StrCtrl_lines
  |  |     |--[<10.611] DANGEROUS
  |  |     |--[>=10.611] OK
  |  |--[>=12.282] DANGEROUS
  |--[>=188.769] DANGEROUS
```

Fig. 6. Two of the induced decision trees for predicting dangerous software modules on the original, non-filtered learning set.

```
tot_of_comments
  |--[<5099.142000] break
  |  |--[<12.903000] OK
  |  |--[>=12.903000] DANGER
  |--[>=5099.142000] DANGER
```

```
declared_vect
  |--[<23.902000] solo_comment_lines
  |  |--[<111.570000] OK
  |  |--[>=111.570000] DANGER
  |--[>=23.902000] DANGER
```

```
StrCtrl_lines
  |--[<25.257927] signif_of_comments
  |  |--[<2484.224000] OK
  |  |--[>=2484.224000] DANGER
  |--[>=25.257927] DANGER
```

```
case
  |--[<20.648000] words_of_comments
  |  |--[<216.804000] OK
  |  |--[>=216.804000] DANGER
  |--[>=20.648000] DANGER
```

Fig. 7. A few of the induced decision trees for predicting dangerous software modules on the filtered learning set.

4.4 Discussion

The classification results on the SCM dataset show that all the decision tree based classifiers (AREX, ID3, C4.5) improved on the training set (as expected) and also improved considerably on the testing set. Whereas the improvement was expected on the training set, as the outliers are more difficult to place in the general model, the improvement on the testing set was only hoped for. The Naïve-Bayes classifier improved on the training set, but scored the same on the testing set. The IB and LogReg classifiers scored practically the same results on the training set and show some improvements on the testing set.

From the above results it can be concluded that the filtering of the training dataset (by removing the outliers) has only a minor effect on IB classifiers; as they search for the most similar objects in the dataset practically the outliers have no effect on the search.

The decision tree classifiers benefit from filtering – the removal of outliers helps to reduce the uncertainty that is introduced by outliers, even as the percentage of the removed objects is rather small (3.6%). As DTs are very weak classifiers, where even a change of a single training case can result in a completely different solution, the removal of identified outliers (which are selected based on their complexity of being correctly classified) substantially improves the classification results.

The Naïve-Bayes and logistic regression classifiers do not benefit considerably from filtering; however, the classification results are not worse either. We must know that statistics, upon which these classifiers are based, normally works towards generalization, neglecting minor (boundary) cases. For this reason, in our opinion, the removal of outliers from the training set does not influence the results considerably.

Generally speaking for all the used classification methods: when using the filtered training set, the quantitative classification results on the testing set are at least as good as when trained on the original training set, and in many cases better.

5 Conclusion

A fault predictive technique to foresee dangerous software modules has been presented in the paper. It is based on a new outlier prediction and removal technique, that is used to filter an original training set, which is then used to train various classifiers. Our proposition was that the filtered training datasets used to train the classifiers can improve the classification results regarding the original, non-filtered datasets.

The obtained results show an improvement of the classification performance with the decision tree classifiers (which are weak classifiers), where outliers have rather negative effect on the training process. The proposed approach has only a minor effect on the instance based, Naïve-Bayes and logistic regression classifiers – in these cases the outliers do not represent a big issue as statistical methods already work primarily in a generalization manner, neglecting minor (boundary) cases. However, all the classification results are at least as good as with the original, non-filtered dataset. This fact speaks in favor of using the proposed outlier prediction and removal method when inducing a fault predictive knowledge model. In particular, because trained knowledge models (in the case of inducing DT classifiers), at least as it turned out in our experiment, are substantially less complex when trained upon filtered training sets. In this manner, an expert can draw some general knowledge out of the induced model.

As even the smallest improvement is of vital importance when mining the software reliability measures data, we plan to further explore the possibilities that the proposed approach offers.

References:

[1] P.K. Kapur, O. Shatnawi, A.G. Aggarwal, R. Kumar, Unified framework for developing testing effort dependent software reliability growth models, *WSEAS Transactions on Systems*, vol. 8, num. 4, 2009, pp. 521-531.

[2] M.M. Breunig, H.-P. Kriegel, R.T. Ng, J. Sander, LOF: Identifying density-based local outliers, *Proceedings of ACM SIGMOD*, 2000.

[3] E.M. Knorr, R.T. Ng, Algorithms for mining distance-based outliers in large datasets, *Proceedings of the 24th VLDB Conference*, 1998.

[4] V. Podgorelec, M. Hericko, I. Rozman, Improving mining of medical data by outliers prediction, *Proceedings of the 18th IEEE Symposium on Computer-based Medical Systems CBMS'2005*, 2005.

[5] V. Podgorelec, P. Kokol, Towards more optimal medical diagnosing with evolutionary algorithms, *Journal of Medical Systems*, Kluwer Academic/Plenum Press, vol. 25, num. 3, 2001, pp. 195-220.

[6] V. Podgorelec, P. Kokol, B. Stiglic, I. Rozman, Decision trees: an overview and their use in medicine, *Journal of Medical Systems*, Kluwer Academic/Plenum Press, vol. 26, num. 5, 2002, pp. 445-463.

[7] V. Podgorelec, P. Kokol, Evolutionary induced decision trees for dangerous software modules prediction, *Information Processing Letters*, vol. 82, num. 1, 2002, pp. 31-38.

[8] T. Baeck, *Evolutionary Algorithms in Theory and Practice*, Oxford University Press, Inc., 1996.

[9] D.E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Reading MA: Addison Wesley, 1989.

[10] J.H. Holland, *Adaptation in natural and artificial systems*, Cambridge, MA: MIT Press, 1975.

[11] A. Rotshtein, M. Posner, H. Rakytyanska, Fuzzy IF-THEN Rules Extraction for Medical Diagnosis Using Genetic Algorithm, *WSEAS Transactions on Systems*, vol. 3, num. 2, 2004, pp. 995-1000.

[12] C.-S. Huang, Y.-J. Lin, C. Lin, Implementation of classifiers for choosing insurance policy using decision trees: a case study, *WSEAS Transactions on Computers*, vol. 7, num. 10, 2008, pp. 1679-1689.

[13] J.R. Koza, *Genetic Programming: On the Programming of Computers by Natural Selection*, Cambridge, MA: MIT Press, 1992.

[14] M. Kasparova, J. Krupka, P. Jirava, Application of decision trees in problem of air quality modelling in the Czech Republic locality, *WSEAS Transactions on Systems*, vol. 7, num. 10, 2008, pp. 1166-1175.

[15] M. Zorman, B. Cernohorski, G. Gorsek, M. Ojstersek, Hybrid evolutionary built decision trees for prediction of perspective cross-country skiers, *WSEAS Transactions on Information Science and Applications*, vol. 2, num. 1, 2005, pp. 32-37.

[16] M. Pighin, P. Kokol, RPSM: A risk-predictive structural experimental metric, *Proceedings of FESMA'99*, pp. 459–464, 1999.

[17] V. Podgorelec, P. Kokol, I. Rozman, AREX - Classification rules extracting algorithm based on automatic programming, *Proceedings of the 15th European Conference on Artificial Intelligence ECAI 2002*, pp. 330-334, 2002.

[18] J.R. Quinlan, *C4.5: Programs for Machine Learning*, Morgan Kaufmann, 1993.

[19] —, RuleQuest Research Data Mining Tools, http://www.rulequest.com, 2001.

[20] —, Machine Learning in C++, MLC++ library, http://www.sgi.com/tech/mlc, 2001.

[21] J. Demsar, B. Zupan, Orange: From Experimental Machine Learning to Interactive Data Mining, White Paper (www.ailab.si/orange), Faculty of Computer and Information Science, University of Ljubljana, 2004.

[22] V. Podgorelec, P. Kokol, M. Zorman, M. Sprogar, M. Pighin, The Operative Constraints of Software Reliability Prediction Methods, *Proceedings of the World Multiconference SCI'2001*, 2001.