On Software Fault Prediction by Mining Software Complexity Data with Dynamically Filtered Training Sets

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

Abstract: - Software fault prediction methods are very appropriate for improving the software reliability. 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. In the paper we present a study on using decision tree classifiers for predicting software faults. A new training set filtering method is presented that should improve the classification performance when mining the software complexity measures data. The classification improvement should be achieved by removing the identified outliers from a training set. 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: - software fault prediction, classification, search-based software engineering, filtering training set, complexity metrics

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.

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 medical data mining with some success [3].
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 the evolutionary algorithm genTrees for the induction of decision trees [4, 5, 6]. One of its greatest advantages, beside its proven efficiency in classifying, is the ability to produce several almost equally 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 (Fig. 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.

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 [6]. With genTrees a set of classifiers (decision trees) are induced. For each training object (data case from the training set) classification \( cc(x) \) 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.

\[
cc_i(x) = \sum_{j=1}^{num\text{-}DTs} \begin{cases} 1; & \text{class } (DT_j, x) = i \\ 0; & \text{otherwise} \end{cases}
\]

\[
CCS(x) = \sum_{i=1}^{num\text{-}classes} \left( \frac{cc_i(x)}{cc_{max}(x)} \right)^2 - 1
\]

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

3 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 [7] – 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 [8], ID3, C4.5 [9, 10], Naïve-Bayes (N-B), instance-based classifier (IB, i.e. k-nearest neighbors), and logistic regression (LogReg) [11, 12]. All the results are the averages of 10-fold cross-validation.

3.1 A dataset
The used software complexity measures dataset (SCM dataset) has been carefully composed from a well-prepared protocol [13]. 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>
<thead>
<tr>
<th>number of cases</th>
<th>904</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of attributes</td>
<td>168</td>
</tr>
<tr>
<td>- nominal</td>
<td>1</td>
</tr>
<tr>
<td>- continuous</td>
<td>167</td>
</tr>
<tr>
<td>number of decision classes</td>
<td>2</td>
</tr>
<tr>
<td>decision classes distribution</td>
<td>76.4%; 23.6%</td>
</tr>
</tbody>
</table>

3.2 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 2 and 3.

For the SCM dataset five classifiers were induced and the tolerance threshold selected \( t=0.2 \); 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>
<thead>
<tr>
<th>classification algorithm</th>
<th>accuracy on the training set [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>original set</td>
</tr>
<tr>
<td>AREX</td>
<td>80.10</td>
</tr>
<tr>
<td>C4.5</td>
<td>95.60</td>
</tr>
<tr>
<td>ID3</td>
<td>100.00</td>
</tr>
<tr>
<td>IB</td>
<td>100.00</td>
</tr>
<tr>
<td>Naïve-Bayes</td>
<td>72.14</td>
</tr>
<tr>
<td>LogReg</td>
<td>92.04</td>
</tr>
</tbody>
</table>
Table 3. Average classification accuracies on the SCM testing dataset.

<table>
<thead>
<tr>
<th>classification algorithm</th>
<th>accuracy on the testing set [%] original set</th>
<th>filtered set</th>
</tr>
</thead>
<tbody>
<tr>
<td>AREX</td>
<td>80.67</td>
<td>83.00</td>
</tr>
<tr>
<td>C4.5</td>
<td>75.00</td>
<td>78.00</td>
</tr>
<tr>
<td>ID3</td>
<td>71.00</td>
<td>75.00</td>
</tr>
<tr>
<td>IB</td>
<td>72.00</td>
<td>73.00</td>
</tr>
<tr>
<td>Naïve-Bayes</td>
<td>79.00</td>
<td>79.00</td>
</tr>
<tr>
<td>LogReg</td>
<td>78.00</td>
<td>79.00</td>
</tr>
</tbody>
</table>

3.3 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 4 there are two decision trees built on the original, non-filtered learning set, and on Figure 5 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
   |--[<1480.04000] StrCtrl_lines
   |   |--[<44.55000] OK
   |   |--[>=44.55000] DANGEROUS
   |--[>=1480.04000] selection_instr
   |   |--[<3.79900] OK
   |   |--[>=3.79900] DANGEROUS
   |--[>=19.16200] formal_function_params
   |   |--[<19.16200] DANGEROUS
   |--[>=19.16200] signif_of_comments
   |   |--[<3026.50800] OK
   |   |--[>=3026.50800] DANGEROUS

words_of_comments
|--[<188.76900] break
  |--[<25.08800] function_calls_to_funcs
    |   |--[<11.98500] vect_function_args
    |   |   |--[<25.08800] const_with_#define
    |   |   |--[<8.13400] formal_func_pars
    |   |   |   |--[<8.13400] OK
    |   |   |--[>=8.13400] DANGEROUS
    |   |--[>=8.13400] DANGEROUS
    |   |--[>=25.08800] fileType
    |   |   |--[<11.98500] StrCtrl_lines
    |   |   |--[<10.61063] DANGEROUS
    |   |   |--[>=10.61063] OK
    |   |--[>=12.28200] DANGEROUS
    |   |--[>=188.76900] DANGEROUS

Fig. 4. Two of the induced decision trees for predicting dangerous software modules on the original, non-filtered learning set.
Considering the qualitative results, the most obvious difference of the knowledge models induced on original and on filtered dataset is the lower complexity of the models trained on the filtered dataset (with the same or better accuracy). This fact speaks in favor of filtered training set when considering the generality of the induced knowledge models – in this way also a more general “knowledge” of software reliability can be learned that applies to more than only one software development project.

It is interesting to look at the induced knowledge models regarding the important software complexity attributes that are used to predict the faults. It can be seen that different measures of comments in the source code is very important (as it has been thought at schools for decades but many times forgotten by the professional software developers). Additionally, the use of the source code structures such as break or case influence the reliability quite considerably.

3 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, 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; 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 outlier prediction and removal method when inducing a fault predictive knowledge 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:

3.4 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%). The Naïve-Bayes and logistic regression classifiers do not benefit considerably from filtering; however, the classification results are not worse either. Generally speaking: 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.

Fig. 5. A few of the induced decision trees for predicting dangerous software modules on the filtered learning set.


