Machine Learning based on Software Metrics for Process Assessment

Steffen Herbold
Georg-August-University of Göttingen,
Institute of Computer Science,
Software Engineering / Theoretical Computer Science,
Goldschmittstr. 7, 37077 Göttingen, Germany.
herbold@cs.uni-goettingen.de

Abstract

The improvement of the currently used processes and quality assurance mechanisms is an important part of software engineering. In our work, we apply machine learning techniques to metric data with the aim to provide techniques that improve the state of the art. Machine learning has the advantage of being unbiased, whereas experts instinctively use their intuition and expertise, which may be biased.

1. Introduction

In software development, the quality of the development process is important for the quality of software products. A high-quality development process leads more likely to high-quality products. The assessment of a development process is a complicated task, usually performed by experts. The aim of our research is to use machine learning techniques to analyse specific features of software processes. Since learning algorithms have been successfully applied in many different fields of research, including software engineering (for example, for defect prediction), we are confident that they will provide valuable results in our setting, too.

For our research, we use software metric data as input for learning algorithms. Our work has two different aspects. The first aspect is the learning and verification of thresholds for software metrics and the analysis of metric sets. These can be used to efficiently locate problematic sections of source code. The second is to analyse software development processes. For this, we use metric data measured at different points of time during the execution of software projects. In the following, these two aspects are introduced in greater detail.

2. Thresholds and metric sets

Software metrics are often used in combination with thresholds. If the value of a metric violates a threshold, some kind of problem with the measured entity is indicated. With sets of software metrics $\{m_1, \ldots, m_n\}$, it is possible to discriminate measured entities into good and bad ones. If the entity violates a threshold of a metric set, it is bad, otherwise it is good. The metric values measured over an entity can be interpreted as a vector in the n-dimensional real-space. The combination of the classification in good and bad entities with their corresponding metric values results in a *supervised* learning sample.

An algorithm to learn n-dimensional rectangles can be applied to such a learning sample. The algorithm determines a rectangle, such that the good entities are inside the rectangle and the bad ones outside. The borders of the rectangle are interpreted as thresholds. This way, the original thresholds, which were used to classify the entities, can be verified. Another approach is to take only a subset of the metrics as input for the learning algorithm while keeping the classification defined by the whole set. With this approach, it is possible to determine whether thresholds for a subset exist that yield a classification that is sufficient for the whole metric set.

A variation of the approach is to consider other means to discriminate the entities in good and bad ones. Above, one violation is enough to classify an entity as bad. However, it is possible that a project manager is content if there are less than k violations, which means that the programmers have a larger scope of coding possibilities. A more complex approach would be to use more than one threshold value, like a soft and a hard threshold. Here the meaning would be as follows: if less than a specified number of the soft thresholds are violated, the code section unproblematic. However, if one of the hard thresholds is violated, it is an indicator for a huge flaw and instantly classifies the entity as bad. A similar idea is to compare threshold violations with violations

against coding standards, like certain structures of variable names. The idea behind this is that a person who does not abide to one rule, often violates other rules, too.

The general ideas presented above can be applied in various settings. For example, the measured entities can vary between methods or classes. The approach should work independently of the programming language or even its paradigm. The approach was successfully applied to TTCN-3 [3]. Currently, we are working on adapting this approach to Java, C, C++ and C#, where we consider both methods and classes and use a deeper algorithmic background. The metrics we use are, for example, from the Chidamber and Kemerer metrics suite. The metric data is obtained from several large open source projects from different domains. Depending on the results, this could lead to a general approach on how to select and improve a set of of software metrics and their threshold values.

3. Process data

Using finished projects for the analysis of the process used is a commonly applied practice. Software metrics are often the basis for such an analysis. For example, higher Capability Maturity Model (CMM) [2] levels require a managed quantitative analysis, i.e., measurement of the process. Usually, only metrics measured at the end of a project are considered. The general idea behind the analysis that is part of our work, is to use metric data that has been measured at different points in time during the execution of a project. Thus, we consider not only the status quo, but the development that led to this point. Only looking at the current status could mask problems that are hidden in the history. The aim is to detect different phases of software development projects with machine learning algorithm based on the metric data.

In a first experiment using open source data, *feature freezes* have been successfully detected using the *k-means clustering algorithm* [1]. For that work, we collected metric data at the milestones of the development of the Eclipse Platform Project 3.2 and the Eclipse Java Development Tools 3.2, using both the CVS repository of the Eclipse Foundation as well as a dump of their Bugzilla database. As input for the k-means algorithm, we used the metrics Lines of Code and Number of Bugs. The algorithm divided the milestones into two sets, which were consistent with the API freeze, that was declared in the project plan.

There are many possible software metrics that can be used for such an analysis, depending on the exact application. Some examples are the size of specification documents, the lines of code, the lines of comments, test case coverage, the reported number of bugs, or the passed and failed test cases. All of these metrics represent certain features of a project, for example, the size of a product. How the values of these develop, should depend on the current phase of a project. For example, the size of the specification documents should only change considerably during the design phase. A general problem is the quality of the data: since projects are usually imperfect the data will be imperfect as well. By using learning algorithms that can cope with noise, we implicitly solve this problem. The imperfectness of the data simply results in noise.

Depending on the learning algorithm and the available data, various approaches are possible. Since our research is in an early phase, it has yet to be determined which metrics and algorithms will ultimatively be used. One interesting approach is to use Conditional Random Fields (CRF). A CRF can than be used to label sequences of data. In this setting, the labels would be the different phases of a project, the data would be software metrics. This could either be used to label projects retrospectively or during their execution. Deviations of the labels with reference to the real phase that took place according to the project plan are indicators for problems.

4. Summary

In our work, we try to define machine learning based methods, to improve existing concepts. As basis for these improvements, we consider whole processes and not only the current status. This should perform better than methods that only take the status quo as basis for their hypothesis, since the history often contains valuable information.

References

[1] S. Herbold. Detection of feature freezes using clustering algorithms. Master's thesis, Masterarbeit im Studiengang Angewandte Informatik am Institut für Informatik, ZFI-BM-2008-12, ISSN 1612-6793, Sept. 2008.

[2] C. Team. Capability Maturity Model® Integration (CMMI SM), Version 1.1. *Pitsburg, Software Engineering Institute*, 2001.

[3] E. Werner, J. Grabowski, H. Neukirchen, N. Röttger, S. Waack, and B. Zeiss. TTCN-3 Quality Engineering: Using Learning Techniques to Evaluate Metric Sets. In E. Gaudin, E. Najm, and R. Reed, editors, *SDL 2007: Design for Dependable Systems, 13th International SDL Forum, Paris, France, September 18–21, 2007, Proceedings*, volume 4745 of *Lecture Notes in Computer Science (LNCS)*, pages 54–68, Berlin, Sept. 2007. Springer.