Chapter 25
Counting the Hidden Defects in Software Documents

Frank Padberg
Saarland University, Germany

ABSTRACT

The author uses neural networks to estimate how many defects are hidden in a software document. Input for the models are metrics that get collected when effecting a standard quality assurance technique on the document, a software inspection. For inspections, the empirical data sets typically are small. The author identifies two key ingredients for a successful application of neural networks to small data sets: Adapting the size, complexity, and input dimension of the networks to the amount of information available for training; and using Bayesian techniques instead of cross-validation for determining model parameters and selecting the final model. For inspections, the machine learning approach is highly successful and outperforms the previously existing defect estimation methods in software engineering by a factor of 4 in accuracy on the standard benchmark. The author's approach is well applicable in other contexts that are subject to small training data sets.

INTRODUCTION

This chapter describes a novel application of machine learning to an important estimation problem in software engineering – estimating the number of hidden defects in software artifacts. The number of defects is a software metric that is indispensable for guiding decisions about the software quality assurance during development. In engineering processes, management usually demands that a certain quality level be met for the products at each production step, for instance, that each product be 98 percent defect-free. Software can never be assumed defect-free. In order to assess whether additional quality assurance is required before a prescibed quality level is met, the software engineers must *reliably estimate* the

DOI: 10.4018/978-1-60566-766-9.ch025

defect content of their software document, be it code or some other intermediate software product, such as a requirements specification or a design artifact. This is a hard problem with substantial economic significance in software practice.

Software companies use defect classification schemes and possess broad practical experience about the kind of errors typically committed by their developers, but software engineering does not have a general theory available that would explain how, when, and where defects get inserted into software artifacts: Currently, no model explains the generating process of the software defects. As a consequence, any estimates must be based on (secondary) defect data that is observed during development and deployment of the software.

During development, defect data emerges mainly during software testing and software inspections. Software testing requires that executable code is available. A typical defect metric collected during testing is the number of defects detected in each test. Software reliability engineering uses this kind of data to predict the total number of defects (including the hidden ones) by extrapolating the cumulative number of defects found during testing. The rationale is that the quality of the code increases as testing progresses and defects get fixed, hence the growth of the defect curve should eventually flatten. Reliability models do not explain the defect generating process, nonetheless they can yield good estimates for the defect content of code.

For software documents other than code, *inspections* are the main quality assurance technique (Gilb & Graham, 1993). During an inspection, reviewers independently find defects in individual detection phases and group meetings in a structured way, using special reading techniques. Inspections are highly effective where testing is not possible, including textual specifications and design documents. Defect data that emerge during an inspection of a document include the number of defects found by each individual reviewer, or the number of different defects detected by the inspection team as a whole.

In principle, software reliability models can also be applied to inspections by replacing the code tests with the detection phases of the individual reviewers. Empirical studies show, though, that reliability models do not work well when transferred to inspections (see the next section). The models generally exhibit a high variation in their estimation error. In particular, the estimates can be extreme outliers without warning during the estimation procedure. As a result, there is a compelling need in software engineering for a reliable defect content estimation technique for inspections.

In this chapter, we view the defect content estimation problem for inspected documents as a *machine learning* problem. The goal is to learn the relationship between certain observable features of an inspection and the true number of defects in the inspected document, including the hidden ones. A typical feature that can be observed during an inspection is the total number of different defects detected by the inspection team; this feature provides a lower bound for the defect content of the document. Some inspection features may carry valuable *nonlinear* information about the defect content of the document; an example is the variation of the reviewers' performance during the inspection. We identify such features using the information-theoretic concept of mutual information. In order to be able to capture any nonlinear relationships between the features and the target, we use *neural networks* for learning from the data.

The use of machine learning in our context is motivated by the key observation that all existing defect estimation techniques for inspections restrict themselves solely to data coming from *just the current* inspection, that is, the inspection effected on the document whose defect content is to be estimated. Hence, the existing techniques miss the opportunity to exploit the knowledge gained in past inspections of similar documents: There is no learning involved in these techniques. However, the way in which the defect content problem presents itself – an estimation problem where a quantitative model must be