Applying Machine Learning to Solve an Estimation Problem in Software Inspections

Thomas Ragg [1,2], Frank Padberg [2], and Ralf Schoknecht [2]

[1] phase-it AG, Vangerowstraße 20, 69115 Heidelberg, Germany
[2] Fakultät für Informatik, Universität Karlsruhe,
Am Fasanengarten 5, 76131 Karlsruhe, Germany
{ragg, padberg, schokn}@ira.uka.de

Abstract. We use Bayesian neural network techniques to estimate the number of defects in a software document based on the outcome of an inspection of the document. Our neural networks clearly outperform standard methods from software engineering for estimating the defect content. We also show that selecting the right subset of features largely improves the predictive performance of the networks.

1 Introduction

Inspections are used with great success to detect defects in different kinds of software documents such as designs, specifications, or source code [7]. In an inspection, several reviewers independently inspect the same document. The outcome of an inspection is a zero-one matrix showing which reviewer detected which defect. Some defects will be detected by more than one reviewer, but usually not all the defects contained in a document are detected. How many defects actually are contained in a document is unknown. To have a basis for management decisions such as whether to re-inspect a document or to pass it on, it is important in software engineering practice to *reliably estimate* the number of defects in a document from the outcome of an inspection.

Defect content estimation methods for software inspections currently fall into two categories: capture-recapture methods [6, 12] and curve-fitting methods [15]. Both approaches use the zero-one matrix of the inspection as the only input to compute the estimate. Several studies show that the defect content estimates computed by these methods are much too unreliable to be used in practice [2, 4, 12, 14]. Both methods show extreme outliers and a high variation in the error of the estimates. A possible explanation is that these methods do not take into account the experience made in past inspections (no learning).

In this paper, we view defect content estimation for software inspections as a *machine learning* problem: the goal is to learn from empirical data collected during *past* inspections the relationship between certain observable features of an inspection and the true number of defects in the document being inspected. A typical example of an observable feature is the total number of different defects detected in an inspection. With our approach, knowledge gained in the past is exploited in the estimation process.

To solve the machine learning problem, we apply the following techniques. We use feature selection based on *mutual information.* As estimation models, we train *neural networks.* For training, we take a *Bayesian* approach and use an error function with a *regularization* term. The selection of the final model is based on the *model evidence.* We have applied this framework successfully in other application domains [10].

The application of neural networks to defect content estimation in software inspections is novel, although neural networks have previously been used in software reliability. Closest to our work are Khoshgoftaar and Szabo [8], but their use of neural network techniques is improper. To estimate the number of defects in software modules, they use 10 different static code metrics as input features. Even after having reduced the number of features using principal component analysis, their training dataset is too small to avoid overfitting a non-linear model. In addition, when training a network they keep adding hidden units and layers until the network achieves a prescribed error bound on the training data, which occurs not until the network has 24 hidden units. No regularization or model selection techniques are used. As a result, the predictive performance of their networks is completely insufficient for software engineering practice.

We validate our machine learning approach applying a jackknife technique to an empirical inspection dataset. We also compare against a particular capture-recapture method (Mt(MLE) [6]) and against a particular curve-fitting method (DPM [15]). The machine learning approach achieves a mean of absolute relative errors of 5 percent. This is an improvement by a factor of 4, respectively, 7, as compared to the other approaches. In addition, no outlier estimates occur for this dataset when using machine learning.

2 Feature Selection and Bayesian Learning

2.1 Data Collection and Feature Generation

The machine learning approach requires a database containing empirical data about past inspections. For each inspection in the database, we need to know the zero-one matrix of the inspection and the true number num of defects in the inspected document. The zero-one matrix is compiled during any inspection. The true number of defects in a document can be approximated by adding up all the defects detected in the document during development and maintenance. This is sufficient for practical purposes; defects which are not detected even during deployment of the software are not relevant for its operational profile.

From the zero-one matrix of an inspection, we generate a set of five candidate features:

- the total number tdd of different defects detected in the inspection;
- the average, maximum, and minimum number ave, max, min of defects detected by a reviewer;
- the standard deviation std of the number of defects detected by a reviewer.

These features are measures for the overall success of an inspection, respectively, for the performance of the individual reviewers.

Our empirical dataset consists of 16 inspections which were conducted on different specification documents during controlled experiments [1]. For each inspection in the dataset, we know its zero-one matrix as well as the true number of defects contained in the document, because the defects had been seeded into the documents.

2.2 Feature Ranking and Subset Selection

Estimating a non-linear dependency between the input features and the target will be more robust when the input-target space is low-dimensional ("empty space phenomenon"). As a rule of thumb, we deduce from Table 4.2 in [13] that for a dataset of size 16 at most two features should be used as input.

To select the two most promising features from our five candidate features, we use a forward selection procedure based on *mutual information.* The mutual information $\mathrm{MI}(X;T)$ of two random vectors X and T is defined as

$$\mathrm{MI}(X;T) \;=\; \iint p(x,t) \cdot \log \frac{p(x,t)}{p(x)\,p(t)}\,.$$

The mutual information measures the degree of stochastic dependence between the two random vectors [5]. To compute the required densities from the empirical data, we use Epanechnikov kernel estimators [10, 13].

As the first step in the selection procedure, that feature f is selected from the set of candidate features which has maximal mutual information with the target for the given dataset. For example, to maximize $\mathrm{MI}(f;\mathsf{num})$ the total number tdd of different defects detected in the inspection is selected. In each subsequent step, that one of the remaining features is selected which maximizes the mutual information with the target when *added* to the already selected features. For example, to maximize $\mathrm{MI}((\mathsf{tdd},f);\mathsf{num})$ in the second step, the standard deviation std of the reviewers' inspection results is selected. This way the features are ranked by the amount of information which they add about the target. In our example, the ranking is $(\mathsf{tdd},\mathsf{std},\mathsf{max},\mathsf{min},\mathsf{ave})$. As input to the neural networks, we use the two features tdd and std.

2.3 Regularization and Bayesian Learning

The functional relationship between the input features and the target which has been learned from the empirical data must generalize to previously unseen data points. The theory of regularization shows that approximating the target as good as possible on the training data, for example, by minimizing the mean squared error E_D on the training data, is not sufficient: it is crucial to balance the training error against the model complexity [3]. Therefore, we train the neural networks to minimize the *regularized error* $E = \beta \cdot E_D + \alpha \cdot E_R$. The regularization term E_R measures the model complexity, taking into account the weights w_k in the network. We choose the weight-decay $\frac{1}{2}\sum {w_k}^2$ as the regularization term.

The factors α and β are additional parameters. Instead of using cross-validation, we take a Bayesian approach to determine the weights w_k and the parameters α and β during training [9, 10]. This is done iteratively: we fix α and β and optimize the weights w_k using the fast gradient descent algorithm Rprop [11]. Afterwards, we update α and β; see [10] for the update rule. We alternate several times between optimizing the weights and updating α and β.

2.4 Model Evidence and Selection

To find the model which best explains the given dataset, we systematically vary the number h of hidden units in the networks from 1 to 10. Since the dataset is small, we put all hidden units in a single layer, restricting the search space to models with moderate non-linearity.

For each network topology, that is, for each number of hidden units, we train 50 networks. As the final model, we select the network which maximizes the *posterior probability* $\mathrm{P}(\theta \,|\, D)$, where $\Theta = (\underline{w}, \alpha, \beta, h)$ is the parameter vector and $\underline{w}$ is the weight vector of the network. To determine the final model, we again use a Bayesian approach. We assign the same prior to each topology h, integrate out α, β, and $\underline{w}$, and estimate the posterior probability by the *model evidence* $\mathrm{P}(D \,|\, h)$ for which a closed expression can be given [10]. The model evidence is known to be in good correlation with the generalization error as long as the number of hidden units is not too large [3]. Therefore, instead of choosing the network with the best evidence from all 500 trained networks, we first choose the topology which has the best *average* evidence. From the 50 networks with the best-on-average topology we select the network with the best evidence as the final model.

Table 1 shows the model selection when the second datapoint is left out from our dataset as the test pattern and the remaining 15 datapoints are used for training. For 1 to 5 hidden units, the correlation between the model evidence and the test error of the corresponding 50 networks is strongly negative, whereas for larger networks the correlation is positive (or absent). The test error grows as

Table 1. Example for model selection.

hidden units	1	2	3	4	5	6	7	8	9	10
mean evidence	-22.6	**-22.5**	-23.6	-24.0	-25.7	-34.3	-35.3	-40.2	-40.7	-40.8
best evidence	-19.5	**-20.0**	-19.2	-19.3	-18.6	-18.5	-18.5	-19.9	-15.9	-16.3
rel. test error	3.5	**3.5**	3.7	3.7	4.0	4.3	5.0	4.2	3.9	5.2
correlation	-0.46	**-0.86**	-0.67	-0.89	-0.77	0.55	0.44	0.25	0.05	-0.02

the number of hidden units increases. Since the networks with two hidden units show the best average evidence, the selection procedure chooses the best model with two hidden units as the final model. The minimal test error is reached with one or two hidden units; thus, the selection procedure yields a network with good predictive performance.

3 Validation and Results

To validate the machine learning approach, we apply a jackknife to our empirical dataset. One by one, we leave out an inspection from the dataset as the test pattern, use the remaining 15 inspections as the training patterns, and compute the relative estimation error for the test pattern. For the inspection left out, we also compute the error for the linear regression model, the capture-recapture method Mt(MLE) [6] and the curve-fitting method DPM [15]. Recall that the last two methods take as input only the zero-one matrix of the inspection which is to be estimated.

The results are given in Table 2. The neural network approach achieves a mean of absolute relative errors of 5.3 percent. The other methods show high

Table 2. Relative estimation errors for the 16 test patterns.

pattern	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Mt	-20	-27	-17	-24	-22	-25	-14	-25	-39	-67	**0**	**0**	-47	-33	-7	-13
DPM	-7	-20	**-3**	-17	-11	-29	18	**-7**	-34	-61	73	27	-40	-27	113	87
linear	**3**	-20	**3**	**-3**	**4**	-14	14	-18	**0**	-22	33	33	-13	7	**0**	7
NN	**-3**	**-3**	**-3**	**-3**	**4**	**4**	**4**	-14	-17	**-17**	13	**0**	**0**	**0**	**0**	**0**

estimation errors and a high error variation. The mean errors are 12.1 percent for the linear model, 23.7 percent for Mt, and 35.8 percent for DPM. Clearly, the machine learning approach yields a strong improvement over the standard methods from software engineering. The improvement over the linear model shows that the function to be learned has a non-linear component.

Using more input features for fitting the unknown function *decreases* the performance of the models significantly. In Figure 1, the number of input features is varied from 1 to 5 according to the ranking of the features given in subsection 2.2. For each set of features, the model selection procedure of subsection 2.4 is applied to each of the 16 jackknife datasets; then, the average model evidence and testing error over the 16 resulting models is computed. The average model evidence is highest when only two features are used, namely, tdd and std. The average testing error is minimal with these two input features. This result experimentally justifies having selected two input features as was suggested by the rule of thumb in subsection 2.2.

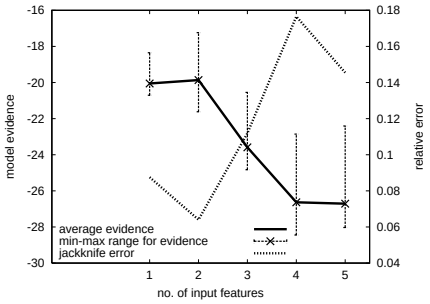

Fig. 1. The jackknife error of the machine learning approach when using input vectors of increasing dimensionality.

The feature tdd is an important input for the estimation because it gives a lower bound for the number of defects in the document. Yet, two rather different inspections can lead to the same total number of different defects detected. For example, in one inspection some reviewers might detect a large number of defects while others detect only a few defects; in some other inspection, each reviewer might detect about the same number of defects. The feature std distinguishes between two such cases, thus being an important supplement to the feature tdd.

The process we have described for building defect content estimation models for software inspections can easily be deployed in a business environment. The process can run automatically without constant interaction by a machine learning specialist. In particular, the estimation models can automatically adapt to new empirical data.

References

1. Basili, Green, Laitenberger, Lanubile, Shull, Sorumgard, Zelkowitz: "The Empirical Investigation of Perspective-Based Reading", Empirical Software Engineering 1:2 (1996) 133-164
2. Biffl, Grossmann: "Evaluating the Accuracy of Defect Estimation Models Based on Inspection Data From Two Inspection Cycles", Proceedings International Conference on Software Engineering ICSE 23 (2001) 145-154
3. Bishop: *Neural Networks for Pattern Recognition*. Oxford Press, 1995
4. Briand, El-Emam, Freimut, Laitenberger: "A Comprehensive Evaluation of Capture-Recapture Models for Estimating Software Defect Content", IEEE Transactions on Software Engineering 26:6 (2000) 518-540
5. Cover, Thomas: *Elements of Information Theory*. Wiley, 1991
6. Eick, Loader, Long, Votta, Vander Wiel: "Estimating Software Fault Content Before Coding", Proceedings International Conference on Software Engineering ICSE 14 (1992) 59-65
7. Gilb, Graham: *Software Inspection*. Addison-Wesley, 1993
8. Khoshgoftaar, Szabo: "Using Neural Networks to Predict Software Faults During Testing", IEEE Transactions on Reliability 45:3 (1996) 456-462
9. MacKay: "A practical bayesian framework for backpropagation networks", Neural Computation 4:3 (1992) 448-472
10. Ragg, Menzel, Baum, Wigbers: "Bayesian learning for sales rate prediction for thousands of retailers", Neurocomputing 43 (2002) 127-144
11. Riedmiller: "Supervised learning in multilayer perceptrons – from backpropagation to adaptive learning techniques", International Journal of Computer Standards and Interfaces 16 (1994) 265-278
12. Runeson, Wohlin: "An Experimental Evaluation of an Experience-Based Capture-Recapture Method in Software Code Inspections", Empirical Software Engineering 3:3 (1998) 381-406
13. Silverman: *Density Estimation for Statistics and Data Analysis*. Chapman and Hall, 1986
14. Vander Wiel, Votta: "Assessing Software Designs Using Capture-Recapture Methods", IEEE Transactions on Software Engineering 19:11 (1993) 1045-1054
15. Wohlin, Runeson: "Defect Content Estimations from Review Data", Proceedings International Conference on Software Engineering ICSE 20 (1998) 400-409