Using Machine Learning for Estimating the Defect Content After an Inspection

Frank Padberg
Fakultät für Informatik
Universität Karlsruhe
Am Fasanengarten 5
76131 Karlsruhe
Germany
padberg@ira.uka.de

Thomas Ragg
quantiom bioinformatics GmbH
Ringstraße 61
76356 Weingarten
Germany
thomas.ragg@quantiom.de

Ralf Schoknecht
Fakultät für Informatik
Universität Karlsruhe
Am Fasanengarten 5
76131 Karlsruhe
Germany
schokn@ira.uka.de

Abstract— **We view the problem of estimating the defect content of a document after an inspection as a machine learning problem: The goal is to learn from empirical data the relationship between certain observable features of an inspection (such as the total number of different defects detected) and the number of defects actually contained in the document. We show that some features can carry significant non-linear information about the defect content. Therefore, we use a non-linear regression technique, neural networks, to solve the learning problem. To select the best among all neural networks trained on a given dataset, one usually reserves part of the dataset for later cross-validation; in contrast, we use a technique which leaves the full dataset for training. This is an advantage when the dataset is small. We validate our approach on a known empirical inspection dataset. For that benchmark, our novel approach clearly outperforms both linear regression and the current standard methods in software engineering for estimating the defect content, such as capture-recapture. The validation also shows that our machine learning approach can be successful even when the empirical inspection dataset is small.**

Keywords— **Defect Content Estimation, Software Inspections, Non-linear Regression, Neural Networks, Empirical Methods.**

I. Introduction

Software inspections are being applied with great success to detect defects in different kinds of software documents such as specifications, designs, test plans, or source code [10]. In an inspection, several reviewers independently inspect the same document. Afterwards, all defects detected by the inspection team are collected. Some defects will be detected by more than one reviewer; hence, the outcome of an inspection is a zero-one matrix showing which reviewer detected which defect.

Usually, not all the defects contained in a document are detected during an inspection. After the inspection, management must decide whether to re-inspect the document to find more defects or pass the document on to the next development step. A common way to reach a decision is to prescribe a certain level of defect-freeness of the document. For example, management could demand that a document be 95 percent defect-free before it is released. In reality, it is unknown how many defects actually are contained in a document. Therefore, it is an important problem in software engineering practice to *reliably estimate* the number of defects in a document from the outcome of an inspection. In the example, if the defect content estimate indicates that more than 5 percent of the defects are still hidden, additional quality assurance is required before the document can be released.

The current standard techniques for estimating the defect content after an inspection fall into two categories: the capture-recapture methods [7], [8], [22] and the curve-fitting methods [25]. We shall give a brief account of these methods in section VI. The standard methods use the zero-one matrix of the current inspection as the only input for computing the estimate. Several studies show that the capture-recapture and curve-fitting estimates are much too unreliable to be used in practice [2], [4], [5], [22], [24]. The methods show extreme outliers and a high variation in the estimation error. A likely explanation is that the standard methods do not take into account the experience made in past inspections.

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 in 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 by the inspection team, which is a lower bound for the true number of defects. With a machine learning approach, knowledge gained in the past is exploited in the estimation process.

To solve our particular machine learning problem, we use *neural networks.* Neural networks should be viewed as a mature non-linear statistical regression technique which is highly suitable for the given kind of estimation problem [3]. Using neural network terminology, a regression function is called a *model,* the input variables are called *features,* the output variable is called the *target,* and a pair consisting of a feature vector and the corresponding target value is called a *pattern.* Some patterns are used for *training,* that is, for iteratively computing (or *learning*) the regression function; other patterns are used for *testing* the performance of the regression function.

The regression function which has been learned from the empirical data is useful only when it is good at predicting the target value for previously unseen input. This property is called *generalization.* Neural network techniques put a strong emphasis on achieving generalization by focusing on features which carry much information about the target, avoiding complicated regression functions during training, and evaluating the performance of the regression models before putting them into use. As with any regression technique, there is no guarantee though that a neural network will yield accurate predictions on new input; the best one can do is to properly apply the techniques and validate the models on empirical inspection datasets.

It seems to be a widespread belief that neural networks are applicable only when a large dataset is available for training. This is *not* the case [3]. Rather, it is essential that the number of features used as input to a neural network stands in a reasonable relation to the size of the database in order to avoid the *empty space phenomenon.* The empty space phenomenon describes the situation that the available training patterns are very sparse in input space [3], [23]. The number of patterns required for training grows quickly with the number of input features. We apply a known practical rule [23] to determine how many input features should be used with a given dataset size, see subsection IV.B. Clearly, the larger the dataset which is available for training the more features, and hence, the more information about the target can be used as input for estimating.

Organizations differ in the way in which inspections are carried out. Also, different reviewers have different skills and experience. Such differences are implicitly reflected in any empirical inspection dataset. To take such differences into account, we do not present just one fixed estimation model; rather, we present a *methodology* which yields for each empirical dataset a regression model which is adapted to that dataset. We have already applied this methodology successfully in other application domains [17], [18]. Our present paper shows how to apply the methodology to the defect content estimation problem in the context of software inspections. Besides neural networks, we use other techniques from machine learning, including a generally applicable feature-selection scheme and a Bayesian methods-based model selection technique. More specifically, our methodology has the following steps:

- *Determine a set of candidate features.*
  The zero-one matrix of an inspection is not suitable as input for the estimation. The matrix is too large, and its dimensions vary from inspection to inspection. Thus, one must compute features (such as the total number of different defects detected in the inspection) from the zero-one matrix, see subsection II.C. Recall that the true number of defects in a document is not a feature, but the target.
- *Select an appropriate subset of the features.*
  How many of the candidate features should finally be selected as input to the neural networks depends on the size of the database, see the discussion above. The selected features should carry as much information about the inspection process as possible. We rank the candidate features using a general scheme which computes the *mutual information* between the features and the target, see subsection IV.A. As opposed to other techniques such as linear correlation analysis, mutual information can detect non-linear dependencies between the features and the target. After having ranked the features, we select the most promising features as input. Note that the result of feature selection depends on the dataset.
- *Train different neural networks on the dataset.*
  Training a neural network is a non-linear optimization process which aims at fitting the regression function to the training data. Because of the non-linearity, training bears the risk of getting caught in a local optimum of the error function which is far from being a global optimum. Thus, we train many different networks with randomly chosen weight initializations. In addition, we train networks with different topologies (different numbers of hidden units) to allow for varying degrees of non-linearity of the learned function. To avoid *overfitting,* we use an error function with a *regularization term,* see subsection IV.B. Overfitting means to memorize the training data by learning a highly complicated function which almost perfectly fits the data but which does not recognize the underlying process. Overfitting usually leads to poor generalization. Regularization during training penalizes complicated functions.
- *Select the best neural network.*
  One way to select the best from all the trained networks is to evaluate the predictive performance of each network on an independent validation dataset (cross-validation). In order to get a validation dataset, one must split the given dataset into two parts, one of which is exclusively reserved for cross-validation. We use a different, Bayesian methods-based approach to model selection which leaves the *whole* dataset for training. We select the best model based on the *model evidence* of each network, see subsection IV.D. The model evidence is known to be well-correlated with the generalization ability as long as the network size is reasonable compared to the size of the dataset [3]. Having the full dataset available for training is a great advantage when the dataset is small, as is often the case with inspection data.

We validate our machine learning approach by applying a jackknife to a well-known empirical inspection dataset, see section V. That dataset serves as a standard benchmark for defect content estimation techniques in software inspections since the true number of defects in each document is known exactly. Our approach achieves a mean of absolute relative errors (jackknife error) of 5 percent on this dataset; in addition, no outlier estimates occur. Compared against the curve-fitting method DPM [25] and the capture-recapture method Mt(MLE) [8], our machine learning approach achieves an improvement by a factor of 7 and 4, respectively, and clearly outperforms the standard

methods on this dataset.

Prior to applying neural network techniques, we check how good a *linear* model would fit our empirical dataset, see section III. In general, one should spend the effort of using advanced non-linear statistical estimation techniques only if there is substantial evidence in the data that the process sampled has a significant non-linear component. For our validation dataset, we show in subsection IV.B that some features carry significant non-linear information about the defect content, which strengthens the case for the application of non-linear regression techniques. Interestingly, linear regression already yields a significant improvement over the standard estimates for our validation dataset, see Table VII in section VI. Exploiting the non-linear dependencies in our dataset between some features and the defect content with neural networks yields a further improvement by a factor of 2 over linear regression, with respect to both the mean and the max relative error.

The application of neural networks to estimating the defect content after software inspections is novel, although neural networks have previously been applied to problems in software reliability, see for example [11], [14]. Closest to our current work are Khoshgoftaar and Szabo [12], [13], but the way in which they use neural network techniques is improper (this is also true for other papers, for example [11]). To estimate the number of defects in already coded software modules, Khoshgoftaar and Szabo use 10 different static source code metrics as input features. Even after having reduced the number of features to 6 by principal component analysis, which is a linear technique, their training dataset is too small to avoid overfitting a non-linear model. When training their networks they start with 12 hidden units and 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. Using too many hidden units and layers again leads to overfitting. As a result, the generalization power of the networks generated by Khoshgoftaar and Szabo is much too low for software engineering practice.

The machine learning approach taken in this paper improves upon of the *interval estimate method* developed by one of the authors [16]. The interval estimate method also uses data from past inspections, but largely differs from neural network-based regression in the way in which the input is processed and the estimates are computed. In particular, the interval estimate method yields an *interval* estimate, that is, a whole range of values which is likely to contain the true value of the number of defects. From the interval estimate, a point estimate can be derived. Each interval estimate comes with a *confidence level* which indicates whether the estimate should be considered reliable or not. When performing a jackknife on the validation dataset, the interval estimate in some cases must be discarded due to a low confidence level, see section VI. The main advantage of the machine learning approach as compared to the interval estimate method is that it provides a good estimate for *each* datapoint in the validation dataset.

This paper is a largely revised and expanded version of a paper which we have presented at a machine learning conference [19].

II. Empirical Data

A. Data Collection

Our machine learning approach requires a database with 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. Entry $a_{j,k}$ in the zero-one matrix is equal to one if reviewer j detected defect k, and zero otherwise. The zero-one matrix is compiled at the end of an inspection.

The true number of defects in a document is hardly ever known exactly, but can be approximated by adding up all defects which were detected in the document after the inspection, that is, during later development phases and maintenance. This gives a lower bound for the true number of defects which is sufficient for practical purposes; defects which are not detected even during deployment of the software are not relevant for its operational profile. In order to be able to compute these approximate defect counts, the software organization must run a defect tracking system which allows to trace each defect back to the documents where the defect was introduced.

B. Validation Dataset

To validate the machine learning approach, we shall use a known empirical inspection dataset. This dataset consists of 16 inspections which were conducted during two controlled experiments in 1994 and 1995 [1]. For each inspection in the dataset, we know the zero-one matrix as well as the true number of defects contained in the inspected document. The true number of defects is known because the experiments seeded typical defects into documents which were considered defect-free otherwise, see [1].

The inspected documents were specifications of different sizes. Two documents came from NASA and were labelled NASA-A and NASA-B. Two documents were generic and were labelled PG and ATM. The number of reviewers in the inspections varied between six and eight; in most inspections, there were six reviewers. The inspections were conducted using two different reading techniques, namely, perspective-based reading and ad-hoc reading. The reviewers were software professionals from the NASA Software Engineering Laboratory. For later use, Table I labels the 16 inspections of the sample dataset.

C. Features

Given an inspection, we compute the following candidate features from its zero-one matrix:

- the total number tdd of *different* defects detected by the inspection team;
- the minimum, maximum, and average number min, max, ave of defects detected by a single reviewer;

- the standard deviation std for the number of defects detected by a single reviewer.

These five features are natural statistical measures for the overall efficiency of an inspection, for the performance of the individual reviewers during the inspection, and for the variation in the performance of the reviewers. Our candidate features do not depend on the size of the zero-one matrix, nor on the number of reviewers, and hence can be used for a variety of different inspections.

For example, the six reviewers in one particular inspection of the sample dataset detected 9, 7, 6, 13, 9, and 6 defects, respectively. Some defects were detected by more than one reviewer. The total number of different defects detected by the team was 23; the overlap between the reviewers was read off the zero-one matrix. The outcome of the inspection yields the following values of the candidate features: tdd = 23, min = 6, max = 13, ave = 8.3, and std = 2.4.

Clearly, one can think of computing other features from the zero-one matrix instead of the five statistical measures we have chosen. For example, our candidate features resemble the input data for the capture-recapture methods more closely than the input data for the curve-fitting methods. Choosing "good" candidate features is *not* a mathematical problem; whether the choice of candidate features is good or not must be judged in the aftermath by looking at the predictive performance of the resulting estimation models.

III. Linear Models

A. Correlation Analysis

We have determined a set of five candidate features as potential input variables for the defect content estimation problem, see section II. As we have already discussed in the introduction (see generalization and empty space phenomenon), it may have a negative impact on the accuracy of the predictions of a regression model to take all available features as input. This is true also for linear models. In general, adding an input variable which carries much noise or contains outliers can distort the linear regression curve and lead to a worsened prediction of the output. When the number of input variables is large compared to the number of given "training" datapoints, linear regression tends to have a small approximation error on these datapoints; yet, a good approximation of the training datapoints does *not* automatically lead to a good prediction of the output for previously unseen input. In other words, the generalization ability of a linear model might become worse as the number of input variables increases.

The objective of *feature selection* is to reduce the set of input variables to a "good" subset which contains only variables with much predictive power with regard to the output. For linear models, we use correlation analysis to select a suitable subset of the input variables for the regression. If two input variables are strongly correlated, one of them can be omitted because the prediction of the output does not improve much when having both variables as input. Thus, we are looking for input variables that are strongly correlated with the output but weakly correlated among each other.

Table II shows the correlation matrix for our empirical dataset. The coefficients in the second last column show the correlation of the input variables with the output.

For each input variable, we perform a two-sided Pearson test to check whether the variable is correlated with the output or not (null hypothesis). The last column in Table II shows the p-values for the tests.

To rank the input variables we proceed as follows. As the first step, we omit all input variables for which the Pearson test yields a p-value of more than 10 percent. Thus, we omit std and max because there is not enough evidence that these variables are *linearly* correlated with the output. In the correlation matrix, the corresponding columns and rows are deleted. As the second step, among the remaining input variables we search for that pair of variables which has the largest correlation coefficient. In our example, the two most strongly correlated input variables are tdd and min (0.838). From that pair of variables, we eliminate the variable which has the smaller correlation with the output. In our case, min has a smaller correlation with num (0.683) than tdd (0.839). Thus, min is eliminated from the input vector. In the correlation matrix, the column and row corresponding to min are deleted. This elimination procedure is repeated until only one input variable is left. The order in which the variables have been eliminated gives a ranking. For our dataset, the ranking is

$$\mathsf{std} < \mathsf{max} < \mathsf{min} < \mathsf{ave} < \mathsf{tdd}.$$

B. Evaluation

Starting with the vector (min, max, tdd, ave, std), the elimination procedure yields a sequence of input vectors of decreasing dimension. To each input vector corresponds a linear regression model. To asses the quality of these linear models, we perform a jackknife validation on our dataset for each input vector. Suppose for a moment that the input vector has been fixed. One by one, we omit one inspection from the dataset and use the remaining 15 inspections to compute the linear regression for the given input vector. Afterwards, we use the linear model to estimate the inspection which was left out, and compute the relative estimation error. Table III shows for each input vector the mean of the absolute relative estimation errors (*jackknife error*).

The most important input variable tdd alone yields a model with a mean estimation error of about 19 percent. Adding the variable ave to the input vector considerably decreases the estimation error of the linear model to about 11 percent. As can be seen from Table VII in section VI, this particular linear model already largely improves upon the standard estimates for our validation dataset. To get some visual impression for the quality of the linear regression based on the features tdd and ave, Figure 1 shows the regression plane corresponding to the full dataset (all 16 datapoints). The datapoints are shown as solid dots.

TABLE I
SAMPLE INSPECTION DATASET.

inspection	document	num
A1	ATM	30
A2	ATM	30
A3	ATM	30
A4	ATM	30
B1	PG	28
B2	PG	28
B3	PG	28
B4	PG	28

inspection	document	num
C1	NASA-A	18
C2	NASA-A	18
C3	NASA-A	15
C4	NASA-A	15
D1	NASA-B	15
D2	NASA-B	15
D3	NASA-B	15
D4	NASA-B	15

TABLE II
CORRELATION MATRIX.

	min	max	tdd	ave	std	num	p-value
min	1.000	0.703	0.838	0.824	0.248	0.683	0.4 %
max	–	1.000	0.759	0.956	0.842	0.384	14.2 %
tdd	–	–	1.000	0.832	0.426	0.839	0.005 %
ave	–	–	–	1.000	0.701	0.464	7.0 %
std	–	–	–	–	1.000	0.034	90.0 %

TABLE III
LINEAR ESTIMATION ERRORS.

input vector	jackknife error	approximation error
min, max, tdd, ave, std	12.1 %	7.6 %
min, max, tdd, ave	10.1 %	7.6 %
min, tdd, ave	10.7 %	7.8 %
tdd, ave	10.8 %	10.1 %
tdd	18.9 %	15.8 %

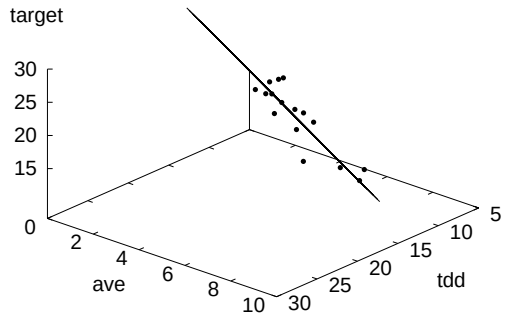

Fig. 1. Linear regression plane.

Table III also shows the approximation error (*training error*) for each of the linear models derived from the correlation ranking. The approximation error is obtained by using all 16 datapoints when computing the linear regression model for a given input vector. With increasing size of the input vector the approximation error decreases, because there are more degrees of freedom to fit the linear model to the data. However, using more than two input variables does *not* significantly improve the jackknife error, which is a measure for the generalization ability of the model. It follows that the two-dimensional vector (tdd, ave) is most appropriate as input when performing linear regression on this dataset.

IV. NON-LINEAR MODELS

In this section, we describe one by one the techniques needed for neural network-based non-linear regression. Subsection A shows how a neural network works as a non-linear function approximator. Subsection B explains our general feature selection scheme based on mutual information. Subsection C describes regularization as a means to avoid overfitting a network during training. Subsection D provides details about the training procedure, including weight optimization with gradient descent and Bayesian methods-based learning of the regularization parameters. Subsection E explains how to use the model evidence instead of cross-validation for selecting the neural network which is finally used for estimating. The exposition in this section follows the description of our methodology given in the introduction.

A. Function Approximation

To approximate a non-linear process, a general function approximator – such as a neural network – is required. A neural network is built from neurons and connections between the neurons as illustrated in Figure 2. The connection between the neurons s_j and s_i carries the weight w_{ji}. Each neuron computes the *weighted sum* over its inputs and applies an *activation function* f_{act} to the result:

$$s_i \;=\; f_{\text{act}}\Big(\sum_j w_{ji}\cdot s_j\Big).$$

When solving regression problems with neural networks, the activation function usually is the logistic function

$$f_{\text{act}}(x) \;=\; \frac{1}{1+e^{-x}}$$

for the hidden neurons and the identity function for the output neurons. Thus, the function of interest is approximated by squasing and stretching logistic functions which are then combined linearly by the output neuron. Figure 3 shows how a neural network with two hidden neurons and one output neuron approximates data which was generated from a polynomial. The two logistic functions corresponding to the hidden neurons are shown on the bottom.

B. Feature Ranking

To avoid the empty space phenomenon discussed in the introduction, the number of input features must be adapted to the number of training patterns available. Table IV is taken from [23] and specifies how many samples are required for estimating the value at the origin of the standard multivariate normal density of a particular dimension such that the relative mean squared error is less than ten percent.

One can consider the table as a practical rule for how many input features should be used for a given set of training patterns. The required number of training patterns quickly increases with the number of input features. For example, the training dataset used in [13] has size 99; thus, only three or four input features should have been used instead of six.

The empirical dataset that we shall use for validation has size 16, see subsection II.B. According to Table IV, we may use only 2 features as input to our networks. To select the two most promising features from our set of five candidates tdd, min, max, ave, and std, it does not make much sense to use the ranking obtained from the correlation analysis in section III, because correlation analysis is a linear technique, and we aim at computing *non*-linear regression functions. We use a feature selection procedure which is based on the concept of *mutual information.* The mutual information $\mathrm{MI}(X;T)$ of two random vectors X and T is defined using the entropy H as

$$\begin{aligned}\mathrm{MI}(X;T) &= \mathrm{H}(T) - \mathrm{H}(T \mid X) \\ &= \iint p(x,t) \cdot \log \frac{p(x,t)}{p(x)\,p(t)}.\end{aligned}$$

The mutual information measures the degree of stochastic dependence between two random vectors X and T [6]. If the mutual information value is high, X carries much information about T. To compute the joint density $p(x,t)$ and the marginal densities $p(x)$ and $p(t)$ from the empirical data, we use Epanechnikov kernel estimators [18], [23]. Kernel estimation is a non-parametric technique for computing a probability density from sample data which improves upon using a histogram.

As the first step in the feature selection procedure, that feature F is selected from the set of candidate features which has maximal mutual information $\mathrm{MI}(\mathsf{F};\mathsf{num})$ with the target, the defect content num. For our dataset, the feature tdd (the total number of different defects detected by the inspection team) is selected since it carries the most information about the defect content among the five candidate features.

In each subsequent step, that one from the remaining features is selected which maximizes the mutual information with the target when *added* to the already selected features. For example, in order to maximize $\mathrm{MI}((\mathsf{tdd},\mathsf{F});\mathsf{num})$ in the second step of the selection procedure, the feature std (the standard deviation for the number of defects detected by a single reviewer during the inspection) is selected. Continuing this way, the features are ranked by the amount of information which they add about the target. For our validation dataset, the final ranking is

$$\mathsf{tdd} > \mathsf{std} > \mathsf{max} > \mathsf{min} > \mathsf{ave}.$$

As input to the neural networks, we use the two features which rank highest, tdd and std.

The feature ranking based on mutual information differs from the ranking based on linear correlation analysis. In particular, the feature std ranked last in the linear case but ranks second with mutual information. The opposite is true for the feature ave. Although there is only a weak *linear* correlation between std and num, mutual information clearly indicates that std carries significant information about num. Consequently, the information carried by std about the defect content must be mainly non-linear. An opposite statement applies to the feature ave. Therefore, mutual information provides strong evidence that using a non-linear regression function instead of a linear one might considerably improve the predictive performance of the regression model.

C. Regularization

The function from the input features to the target which was learned from the empirical data by the neural networks should generalize to new input. Empirical data usually is noisy, that is, the data shows small random deviations from the underlying process. Similar to linear regression, approximating the target as good as possible on the training data (for example, by minimizing the mean squared error on the training data) does not automatically lead to good generalization. Figure 4 illustrates the noisy data problem for the polynomial from Figure 3 with Gaussian noise added to the original data. When trying to minimize the training error by using a network with many hidden neurons, *overfitting* occurs: the output function almost perfectly matches the training data but is very complicated, resulting in a poor approximation of the underlying smooth polynomial.

In order to achieve good generalization, it is crucial to balance the training error against the model complexity [3]. Thus, we train the neural networks to minimize the *regularized error*

$$E = \beta \cdot E_D + \alpha \cdot E_R.$$

The term E_D is the mean squared error on the training data. The factors α and β are additional model parameters, see the next subsection. The regularization term E_R measures the model complexity, taking into account the weights w_k in the network. We choose the weight-decay $\frac{1}{2} \cdot \sum {w_k}^2$ as the regularization term. The weight decay penalizes large weights and (indirectly) large numbers of hidden units. Large weights still are possible, but there must be considerable evidence in the data for such weights to occur in the network after training.

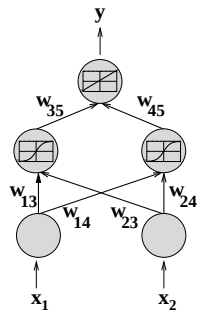

Fig. 2. Architecture of a neural network.

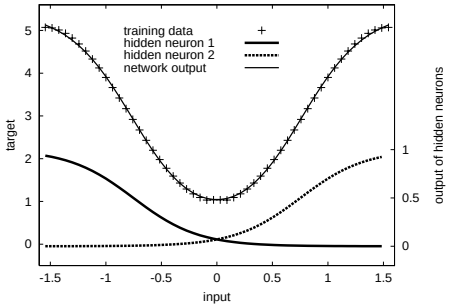

Fig. 3. Estimating a non-linear function.

TABLE IV
FEATURES VS. TRAINING PATTERNS.

features	patterns
1	4
2	19
3	67
4	223
5	768
6	2790

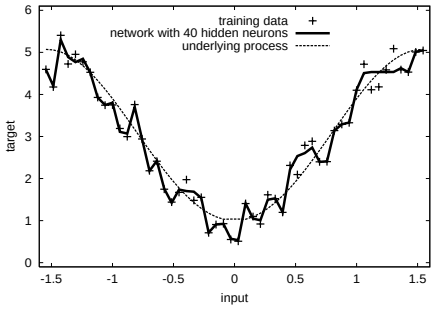

Fig. 4. Overfitting to noisy data.

D. Bayesian Learning

Training a network aims at adjusting the weights w_k of the network in such a way that the regularized error E is minimized (non-linear optimization). When the parameters α and β are fixed, we optimize the weights using the fast gradient descent algorithm Rprop [20]. The gradient of the error E with respect to the weights is computed by backpropagation [21].

Instead of using up part of our empirical dataset for determining α and β before training a network, which is the standard approach, we apply Bayesian techniques to determine the weights w_k and the parameters α and β *simultaneously* during training [15], [18]. The Bayesian approach leaves the whole dataset for training. The idea is to maximize during training the posterior probability $\mathrm{P}(\alpha, \beta \mid D)$ of α and β given the training data D. The posterior probability is computed from the so-called *evidence* $\mathrm{P}(D \mid \alpha, \beta)$ for α and β and from prior assumptions about their joint density $p(\alpha, \beta)$ by using Bayes' rule:

$$\mathrm{P}(\alpha, \beta \mid D) = \frac{\mathrm{P}(D \mid \alpha, \beta) \cdot p(\alpha, \beta)}{p(D)}.$$

Maximization of the posterior probability can be transformed into an *update rule* for α and β, see [18] for the technical details.

We finally get the following iterative procedure for training a network: We randomly choose α and β and optimize the weights using gradient descent. Afterwards, we update α and β using the update rule. Then, we again optimize the weights, now using the updated values for α and β in the error function. We keep alternating between optimizing the weights and updating α and β until a prescribed number of training cycles (*epochs*) has been reached.

Figure 5 shows a sample error curve (*learning curve*) for one neural network during training. When optimizing the network weights the training error goes down quickly. Each time α and β are updated the training error increases, but the overall trend is for the training error to decrease.

E. Model Evidence

To try out different degrees of non-linearity for the regression function, we systematically vary the number h of hidden units in the neural networks from 1 to 10. Since our empirical dataset is small, we put all hidden units in a single layer, restricting the regression models to moderate non-linearity. For each network topology, that is, for each number h of hidden units, we train 50 networks with random weight initializations. Each of these networks is trained using the alternating procedure described in the previous subsection to optimize its weights and determine the model parameters α and β.

After training, we have a pool of 500 networks (50 networks for each of the 10 topologies) which fit the training data. To choose the network which is best suited for estimating the defect content, one usually must reserve part

of the empirical dataset for cross-validation *before* training begins, using only the other part of the dataset for training. Again, we use a Bayesian approach instead which leaves the whole dataset for training.

For each of the trained networks, we compute its *model evidence* $P(D \mid \alpha, \beta, h, w_1, w_2 \ldots)$. The model evidence is known to be in good correlation with the generalization error as long as the number of hidden units is reasonable compared to the size of the dataset [3]. 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 then select the network with the best evidence as the final model. Practical experience shows that first choosing a good topology is more robust and can be expected to yield better generalization than immediately choosing the network with the best evidence.

Table V shows our model selection procedure for an example. The second datapoint has been left out from our dataset as the test pattern

and the remaining 15 datapoints were used for training. For 1 to 5 hidden units, the correlation between (the logarithm of) the mean evidence and the test error of the corresponding 50 networks is strongly negative, whereas for larger networks the correlation is positive (or absent). Although the test error does not change much as the number of hidden units increases, the correlation clearly indicates that one should better select a network with a small number of hidden units. 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.

V. Validation

To validate our machine learning approach, we apply a jackknife to the empirical dataset described in section II. Recall that the true number of defects for each inspected document in this dataset is exactly known since the defects had been seeded into the documents. One by one, we leave out an inspection from the dataset as the test pattern and use the remaining 15 inspections as the training patterns. For each of the 16 datasets constructed this way, the training and model selection procedure described in the previous section yields a neural network (with the features tdd and std as input) which is used to estimate the test pattern.

The estimation results for the 16 test patterns are given in Table VI. The table also shows the number of different defects detected during the inspection (which is a lower bound for the true number of defects). Our machine learning approach achieves a mean of absolute relative errors (jackknife error) of only 5.3 percent. In particular, no outlier estimates occur. The individual estimates show that on this dataset our novel approach provides a *highly reliable estimator* for the defect content.

To get some visual impression how the non-linear regression functions learned by the neural networks look like, Figure 6 shows the regression surface corresponding to the "best" neural network with two hidden units when using the full dataset (all 16 datapoints) for training. The regression surface closely approximates the datapoints, which are shown as solid dots. The approximation is better than for the linear regression plane shown in Figure 1. As a consequence, our machine learning approach outperforms linear regression on the sample dataset with respect to both the mean and the max relative error.

The feature tdd is 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 particular 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. For the sample dataset, a large value for std at the bottom level of the surface in Figure 6 indicates that tdd is close to the true number of defects in the document; a small value for std indicates that tdd is significantly off. Note that the separation of dots in Figure 6 into a "bottom level" and a "top level" is special for this dataset.

Using more than two features for the regression *decreases* the performance of the models significantly. In Figure 7, the number of features increases from 1 to 5 according to the ranking computed in subsection IV.B. For each set of features and each of the 16 jackknife datasets, we repeat the training and model selection procedure to get one neural network. Figure 7 shows for each set of features the model evidence and test error averaged over the corresponding 16 networks. The average model evidence is highest when only two features are used, namely, tdd and std. The jackknife error is minimal with these two input features. This result experimentally justifies having selected just two input features as was suggested by the practical rule given in subsection IV.B.

VI. Comparison

In this section, we compare our machine learning approach with other methods for estimating the defect content: the capture-recapture method Mt(MLE), the detection profile method DPM, and the interval estimate method IEM. Table VII shows for each method discussed in this paper the individual estimates and the estimation errors for the 16 inspections (test patterns) from the empirical dataset described in section II. The table also shows the maximum and mean absolute relative estimation error for each method.

A. Capture-Recapture Methods

Capture-recapture methods use stochastic models which were developed in biology to estimate the size of an animal population based on repeated captures of animals. When applying capture-recapture to software inspections, the animals are replaced by defects and each capture is replaced by the inspection performed by one of the reviewers. For the comparison, we use the method Mt(MLE)

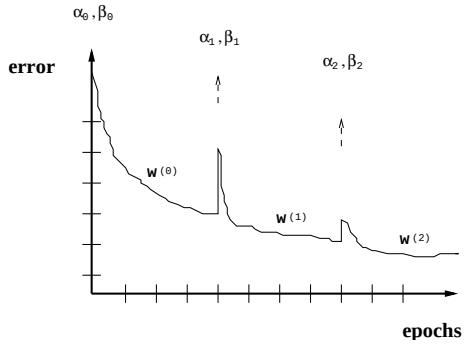

Fig. 5. Iterative training procedure.

TABLE VI
LEARNING ESTIMATES FOR THE 16 TEST PATTERNS.

inspection	tdd	num	estimate	error
A1	23	30	29	-3 %
A2	20	30	29	-3 %
A3	24	30	29	-3 %
A4	22	30	29	-3 %
B1	20	28	29	4 %
B2	17	28	29	4 %
B3	24	28	29	4 %
B4	21	28	24	-14 %
C1	10	18	15	-17 %
C2	6	18	15	-17 %
C3	15	15	17	13 %
C4	15	15	15	0
D1	6	15	15	0
D2	9	15	15	0
D3	14	15	15	0
D4	13	15	15	0

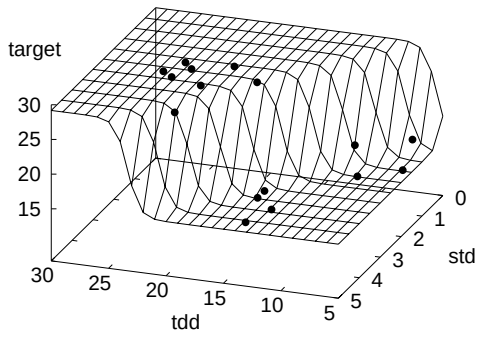

Fig. 6. Learned non-linear regression surface.

[8]. Other capture-recapture methods show a similar performance [5].

The method Mt (MLE) uses only the zero-one matrix of the inspection to be estimated as input, but no data about past inspections. From the zero-one matrix, the total number of different defects detected during the inspection is computed, as well as the number of defects detected by each individual reviewer. The probabilistic model underlying Mt (MLE) assumes that defects are probabilistically identical, but different reviewers may have different probabilities of detecting defects. The probabilistic model is parameterized by the unknown true number N of defects contained in the document. From the observed input data, a maximum likelihood estimate for N is computed.

Table VII shows that the capture-recapture estimates have a strong tendency to underestimate the true number of defects in the document. The estimation error has a high variation. The mean of the absolute relative errors is 23.7 percent. In Figure 8, the true number of defects in each document is shown as a solid dot, our corresponding machine learning estimate as a hollow dot, and the capture-recapture estimate as a small cross. The capture-recapture method is clearly outperformed by our machine learning approach on this dataset.

B. Detection Profile Method

The detection profile method DPM is a curve-fitting method [25]. Like the capture-recapture methods, DPM uses only the zero-one matrix of the inspection to be estimated as input, but no data about past inspections. From the zero-one matrix, for each defect j the number n_j of reviewers who have detected that particular defect is computed. The numbers n_j are sorted according to their size; then, an exponential curve is fitted through the sorted datapoints using linear regression on the log values. The estimate is the x-value of the intersection point of the exponential curve with the horizontal line $y = 0.5$.

Table VII shows that the estimation error for the detection profile method has an extremely high variation. The mean of the absolute relative errors is equal to 35.8 percent. The curve-fitting method DPM is clearly outperformed by our machine learning approach on this dataset.

C. Interval Estimate Method

The interval estimate method IEM [16] uses the same empirical database about past inspections as the machine learning approach. Yet, the interval estimate method largely differs from the machine learning approach in the way in which the input data is processed and the estimates are computed.

For each inspection in the database, the *signature* of the inspection is computed from the zero-one matrix and the corresponding true number of defects. The signature of an inspection combines a measure for the overall efficiency of the inspection (*efficiency class*) with a measure for the variation among the inspection results of the individual reviewers (*span*). The signature carries information which

is similar to the features tdd and std used as input to the neural networks. The computation of the signature also involves an *equivalence relation* to bundle up the datapoints and make it easier to compare different inspections. From the signatures of the inspections in the empirical database a *probabilistic model* for the outcome of an inspection is constructed. Since by construction the signature of an inspection depends on the true number N of defects in the corresponding document, the probabilistic model is parameterized by N, similar to the capture-recapture models.

Given the outcome of an inspection, a maximum likelihood estimation for the defect content of the document is performed. Since there is an equivalence relation involved in computing signatures, the interval estimate method yields a whole range of values which is likely to contain the true value of the number of defects in the document (*interval estimate*). From the interval estimate, various point estimates can be derived. For example, the lower boundary of the interval estimate can serve as a point estimate. Each interval estimate comes with a *confidence level* which indicates whether the estimate should be considered reliable or not.

It can be necessary to subdivide the dataset according to, for example, the document type, and use only the relevant part of the dataset as input for constructing the probabilistic model. The need to subdivide the dataset is indicated by the confidence levels of the interval estimates when performing a jackknife on the full dataset. There is no need to subdivide the empirical dataset when using our machine learning approach: The regression functions represented by the neural networks *automatically* adapt to small and inhomogeneous datasets.

When performing a jackknife on the empirical dataset from section II, the confidence levels indicate that this dataset must be split according to the document type (generic versus NASA). The confidence levels also show that the NASA half of the dataset is too inhomogeneous to derive valid interval estimates, see [16] for the details. Therefore, IEM provides estimates only for the inspections A1 to B4. We take the lower boundary of an interval estimate as the corresponding point estimate.

Table VII shows that the mean of the absolute relative errors for IEM is 6.6 percent for the inspections A1 to B4. The interval estimate method clearly outperforms the capture-recapture and detection profile method on the generic documents. The interval estimate method and our machine learning approach have a similar performance for the generic documents. The main difference is, though, that our novel machine learning approach also provides reliable estimates for the NASA documents, for which the interval estimate method does not provide estimates.

VII. Conclusions

In this paper, we have shown how machine learning can be used to solve the defect content estimation problem for software inspections. As opposed to the standard methods for estimating the defect content, our novel approach not only uses the zero-one matrix of an inspection, but also an empirical database collected during past inspections as input for computing the estimate.

Our approach identifies defect content estimation for software inspections as a *non-linear regression problem.* A major motivation for our approach was the discovery that some features of an inspection can carry significant non-linear information about the defect content of the inspected document. Using this information can greatly improve the accuracy of the estimates.

In order to exploit the non-linear information carried by some features one must go beyond the scope of linear regression. We have deliberately chosen one particular machine learning technique, neural networks, for solving the regression problem. Neural networks are a mature technique that we are familiar with and that we have already successfully applied elsewhere. Other non-linear regression techniques might do as well, but this is subject to further research.

Besides neural networks, we apply other techniques from machine learning, including a general feature selection scheme based on mutual information and a Bayesian methods-based model selection technique. The final regression model depends on the concrete dataset in two ways: which features are selected as input depends on the data, as well as which function is learned by the networks. Selecting a promising set of features as input is important in order to gain a strong predictive performance of the regression model.

We have validated our machine learning approach for estimating the defect content after an inspection using a known empirical inspection dataset. The main advantages of our machine learning approach which we have observed *on this dataset* include:

- Our approach yielded much more accurate estimates than the standard estimation methods such as capture-recapture and detection profile.
- Our approach gave clear guidelines which information about an inspection to select as input for estimating.
- Our approach detected and exploited non-linear relationships between the features of an inspection and the defect content of the document.
- Our approach worked well with a small dataset, leaving the whole dataset for training.
- Our approach automatically adapted to different document types and sizes, reading techniques, and team sizes, as represented in the dataset.

More validation is needed using other empirical datasets, preferably industry-scale datasets. In order to compile such a dataset, a defect tracking system must be in use to trace each defect back to the documents where the defect was introduced.

The estimated number of residual defects in *code* often is used as a predictor for the future reliability of the software in operation. This is problematic and has been critized, see the overview article [9]. The difficulty is to predict which defects are likely to lead to failures; experience suggests that a small number of the defects leads to a large fraction of the observed failures. Despite this problem, the defect

content estimate remains an important management tool for making informed decisions about the next development steps, especially for documents other than code.

The methodology we have described in this paper for building defect content estimation models for software inspections can be deployed with little effort in a business environment. The whole estimation process can run automatically without constant interaction by a machine learning specialist. In particular, the neural network models can automatically adapt to new empirical data.

VIII. Acknowledgements

We would like to thank Vic Basili and Forrest Shull from the University of Maryland who generously provided the zero-one matrices from their 1994 and 1995 inspection experiments.

References

[1] Basili, Green, Laitenberger, Lanubile, Shull, Sorumgard, and Zelkowitz. The empirical investigation of perspective-based reading. *Empirical Software Engineering*, 1(2):133–164, 1996.

[2] Biffl and Grossmann. Evaluating the accuracy of defect estimation models based on inspection data from two inspection cycles. In *Proceedings International Conference on Software Engineering ICSE*, volume 23, pages 145–154, 2001.

[3] Bishop. *Neural Networks for Pattern Recognition*. Oxford Press, 1995.

[4] Briand, El-Emam, and Freimut. A comparison and integration of capture-recapture models and the detection profile method. In *Proceedings International Symposium on Software Reliability Engineering ISSRE*, volume 9, pages 32–41, 1998.

[5] Briand, El-Emam, Freimut, and Laitenberger. A comprehensive evaluation of capture-recapture models for estimating software defect content. *IEEE Transactions on Software Engineering*, 26(6):518–540, 2000.

[6] Cover and Thomas. *Elements of Information Theory*. Wiley, 1991.

[7] Ebrahimi. On the statistical analysis of the number of errors remaining in a software design document after inspection. *IEEE Transactions on Software Engineering*, 23(8):529–532, 1997.

[8] Eick, Loader, Long, Votta, and Vander Wiel. Estimating software fault content before coding. In *Proceedings International Conference on Software Engineering ICSE*, volume 14, pages 59–65, 1992.

[9] Fenton and Neil. A critique of software defect prediction models. *IEEE Transactions on Software Engineering*, 25(5):675–689, 1999.

[10] Gilb and Graham. *Software Inspection*. Addison-Wesley, 1993.

[11] Karunanithi, Whitley, and Malaiya. Prediction of software reliability using connectionist models. *IEEE Transactions on Software Engineering*, 18(7):563–574, 1992.

[12] Khoshgoftaar, Pandya, and More. A neural network approach for predicting software development faults. In *Proceedings International Symposium on Software Reliability Engineering ISSRE*, volume 3, pages 83–89, 1992.

[13] Khoshgoftaar and Szabo. Using neural networks to predict software faults during testing. *IEEE Transactions on Reliability*, 45(3):456–462, 1996.

[14] Lanubile and Visaggio. Evaluating predictive quality models derived from software measures: Lessons learned. *Journal of Systems and Software*, 38:225–234, 1997.

[15] MacKay. A practical bayesian framework for backpropagation networks. *Neural Computation*, 4(3):448–472, 1992.

[16] Padberg. Empirical interval estimates for the defect content after an inspection. In *International Conference on Software Engineering ICSE*, volume 24, pages 58–68, 2002.

[17] Ragg. Bayesian learning and evolutionary parameter optimization. In *Proceedings KI 2001: Advances in Artifical Intelligence*, pages 48–62. Springer LNAI 2174, 2001.

[18] Ragg, Menzel, Baum, and Wigbers. Bayesian learning for sales rate prediction for thousands of retailers. *Neurocomputing*, 43:127–144, 2002.

[19] Ragg, Padberg, and Schoknecht. Applying machine learning to solve an estimation problem in software inspections. In *Proceedings International Conference on Artificial Neural Networks ICANN*, pages 516–521. Springer LNCS 2415, 2002.

[20] Riedmiller. Supervised learning in multilayer perceptrons – from backpropagation to adaptive learning techniques. *International Journal of Computer Standards and Interfaces*, 16:265–278, 1994.

[21] Rumelhart, Hinton, and Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[22] Runeson and Wohlin. An experimental evaluation of an experience-based capture-recapture method in software code inspections. *Empirical Software Engineering*, 3(3):381–406, 1998.

[23] Silverman. *Density Estimation for Statistics and Data Analysis*. Chapman and Hall, 1986.

[24] Vander Wiel and Votta. Assessing software designs using capture-recapture methods. *IEEE Transactions on Software Engineering*, 19(11):1045–1054, 1993.

[25] Wohlin and Runeson. Defect content estimations from review data. In *Proceedings International Conference on Software Engineering ICSE*, volume 20, pages 400–409, 1998.

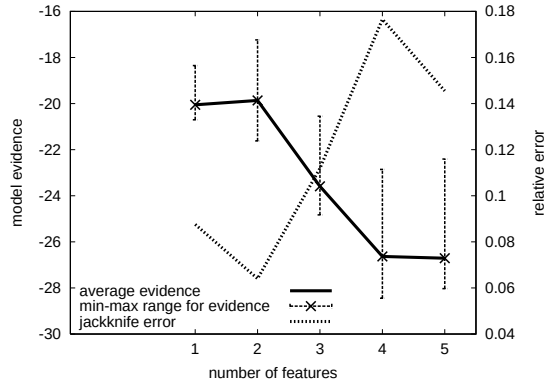

Fig. 7. Estimation with different feature vectors.

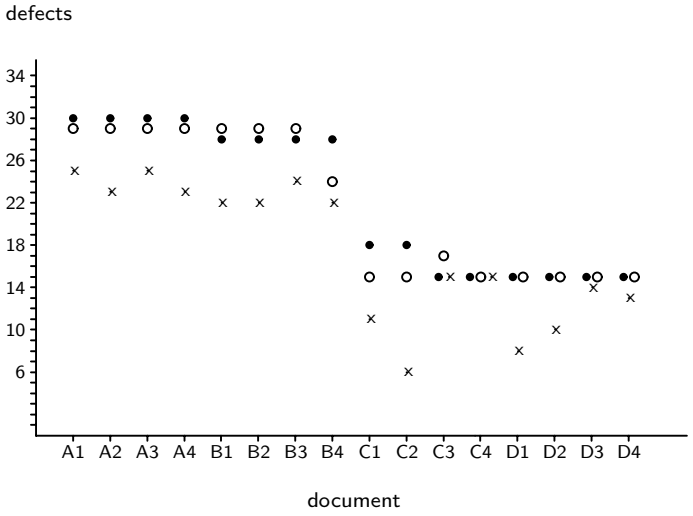

Fig. 8. Capture-recapture method vs. learning.

TABLE V
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
best test error	-3 %	**-3 %**	-3 %	-3 %	-3 %	-3 %	-7 %	-3 %	-3 %	-7 %
correlation	-0.46	**-0.86**	-0.67	-0.89	-0.77	0.55	0.44	0.25	0.05	-0.02

TABLE VII
ESTIMATES AND RELATIVE ESTIMATION ERRORS FOR THE 16 TEST PATTERNS USING DIFFERENT ESTIMATION TECHNIQUES.

pattern	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	max	mean
Mt (MLE)	24	22	25	23	22	21	24	21	11	6	15	15	8	10	14	13		
error in %	-20	-27	-17	-24	-22	-25	-14	-25	-39	-67	**0**	**0**	-47	-33	-7	-13	67 %	23.7 %
DPM	28	24	29	25	25	20	33	26	12	7	26	19	9	11	32	28		
error in %	-7	-20	-3	-17	-11	-29	18	**-7**	-34	-61	73	27	-40	-27	113	87	113 %	35.8 %
IEM	29	28	30	34	27	27	30	32	–	–	–	–	–	–	–	–		
error in %	**-3**	-7	**0**	13	**-4**	**-4**	7	14	–	–	–	–	–	–	–	–	14 %	6.6 %
linear	29	28	28	32	29	25	30	26	18	13	16	21	15	18	17	13		
error in %	**-3**	-7	-7	7	**4**	-11	7	**-7**	**0**	-28	7	40	**0**	20	13	-13	40 %	10.8 %
learning	29	29	29	29	29	29	29	24	15	15	17	15	15	15	15	15		
error in %	**-3**	**-3**	-3	**-3**	**4**	**4**	**4**	-14	-17	**-17**	13	**0**	**0**	**0**	**0**	**0**	17 %	5.3 %