Expert Systems with Applications 38 (2011) 3440–3445

Contents lists available at ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

Thresholds based outlier detection approach for mining class outliers: An empirical case study on software measurement datasets

Oral Alan, Cagatay Catal [*]

The Scientific and Technological Research Council of Turkey (TUBITAK), The National Research Institute of Electronics and Cryptology (UEKAE), Information Technologies Institute, 41470 Kocaeli, Turkey

ARTICLE INFO

Keywords:
Outlier detection
Software metrics thresholds
Software fault prediction
Empirical software engineering

ABSTRACT

Predicting the fault-proneness labels of software program modules is an emerging software quality assurance activity and the quality of datasets collected from previous software version affects the performance of fault prediction models. In this paper, we propose an outlier detection approach using metrics thresholds and class labels to identify class outliers. We evaluate our approach on public NASA datasets from PROMISE repository. Experiments reveal that this novel outlier detection method improves the performance of robust software fault prediction models based on Naive Bayes and Random Forests machine learning algorithms.

1. Introduction

As software systems are becoming more and more complex, the testing of these systems is getting more complicated. Hence, the testing duration should be shortened or the majority of testing resources should be allocated to fault-prone modules. Quality assurance groups or project managers can detect these fault-prone modules prior to system testing by using software fault prediction methods that provide high-performance (Catal & Diri, 2009a; Menzies, Greenwald, & Frank, 2007). Several fault prediction approaches have been proposed such as Decision Trees, Logistic Regression, Fuzzy Logic, Neural Networks, Genetic Programming, Naive Bayes, Case-based Reasoning, and Dempster-Shafer Networks since 1990s (Catal & Diri, 2009b). However, the performance of fault prediction models decreases when there exist outliers in datasets. Several authors reported that public NASA datasets locating in PROMISE repository include many inconsistent modules (same attribute values but different fault labels) (Zhong, Khoshgoftaar, & Seliya, 2004).

An outlier is "one that appears to deviate markedly from other members of the sample in which it occurs" (Hodge & Austin, 2004). The identification of these outliers before building fault prediction models is a critical issue and several outlier detection methods that can be used as a pre-processing step were suggested in the literature. There can be mentioned algorithms such as statistical approaches, machine learning and neural networks based methods for outlier detection (Hodge & Austin, 2004). In this study, our motivation was to develop an outlier detection approach that uses domain knowledge, specifically software metrics thresholds, which will improve the performance of robust software fault prediction models based on Naive Bayes (Menzies et al., 2007) and Random Forests (Ma, Guo, & Cukic, 2006) machine learning algorithms.

We used public datasets called KC1, KC2, PC1, CM1 and JM1 from PROMISE repository which was created in 2005. These datasets include 21 method-level metrics which was proposed by Halstead (1977) and McCabe (1976). However, derived Halstead metrics are not generally used for software fault prediction (Seliya & Khoshgoftaar, 2007). Munson (2003) explains that four Halstead primitive metrics describe the variation of all of the rest of Halstead metrics and hence, derived Halstead metrics do not provide any additional information. In this study, we used seven method-level metrics because our outlier detection approach uses metrics thresholds and thresholds of these metrics are known from empirical researches. These metrics are lines of code, cyclomatic complexity, essential complexity, unique operators, unique operands, total operators and total operands.

Our approach uses both class labels and software metrics. Before building this outlier detection method, we examined the features of each module that locate in datasets. We noticed that there are several modules which have very low metrics values but have faulty class labels. Furthermore, there exist some modules which have very high metrics values but have not-faulty class labels. These observations helped us to construct three research questions that are shown as follows:

RQ1: Can software metrics thresholds help us to identify the faulty modules that are recorded as not-faulty in the dataset?

* Corresponding author. Tel.: +90 262 677 26 34.
E-mail addresses: oral.alan@bte.mam.gov.tr (O. Alan), cagatay.catal@bte.mam.gov.tr (C. Catal).

0957-4174/$ - see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.eswa.2010.08.130

RQ2: Can software metrics thresholds help us to identify the not-faulty modules that are recorded as faulty in the dataset?
RQ3: Can metrics thresholds based outlier detection approach improve the performance of robust fault prediction algorithms such as Naive Bayes and Random Forests?

This paper is organized as follows: Section 2 presents the related work. Section 3 presents the algorithms. Section 4 describes the experiments, contexts, hypotheses formulation, subjects, and design of the experiments, instrumentation, validity evaluation, and experiment operation. Section 5 shows the analysis of the experiments. Finally, Section 6 presents the conclusions and future works.

2. Related work

There exist numerous studies on outlier detection in the field of statistics and these studies can be classified into two main groups: distribution-based and *depth-based* approaches. Distribution-based approaches use a standard distribution (e.g. Poisson) to fit the dataset into this chosen distribution. However, in the field of software engineering, fitting a large dataset into a distribution is a time-consuming and expensive process (Tang & Khoshgoftaar, 2004). In addition, software quality experts or project managers may not have knowledge about the underlying distribution of datasets (Tang & Khoshgoftaar, 2004). In depth-based approach, *depth* information is assigned to each data object and outliers are assumed to be the objects that have smaller depths (Breunig, Kriegel, Ng, & Jörg Sander, 2000). According to the high computational complexity of depth-based approaches, they are not appropriate to be used for large datasets.

The other approaches are based on distance or similarity measures. Knorr and Ng (1998) proposed a method based on distance calculation to identify outliers. "A point p in a dataset is an outlier with respect to parameters k and d if no more than k points in the dataset are at a distance of d or less from p" (Tang & Khoshgoftaar, 2004). Ramaswamy, Rastogi, and Shim (2000) used K-nearest neighbor (KNN) to identify outliers. They applied BIRCH clustering algorithm to cluster data objects and removed data objects which do not include any outlier. Clusters which still exist were evaluated according to the distance measure in order to find the potential outliers.

Clustering-based outlier detection approaches are also based on distance measures and they use different clustering techniques to find outliers. Clusters are grouped into two categories such as small clusters and large clusters. Jiang, Tseng, and Su (2001) proposed an algorithm which marks small clusters as outliers. However, He, Xu, and Deng (2003) presented a cluster-based local outlier factor (CBLOF) algorithm and stated that some outliers may belong to large clusters too. Tang and Khoshgoftaar (2004) argued that this approach is limited to only categorical attributes and they presented a new noise factor metric to eliminate this limitation.

None of the above mentioned approaches take advantage from software metrics thresholds and in this study, we developed thresholds based outlier detection approach which uses software metrics thresholds.

3. Algorithms

In this study, our motivation was to develop outlier detection methods that depend on the usage of software metrics thresholds and fault data (depicted as faulty or not-faulty in the software measurement dataset). Metrics thresholds used in this study were determined from Integrated Software Metrics, Inc.'s (ISM) Predictive tool documentation. We made experiments on widely used public datasets (Tang & Khoshgoftaar, 2004), called KC1, KC2, PC1, CM1, JM1 that have 21 software metrics including McCabe (1976) and Halstead (1977) metrics. We used seven primitive metrics shown in Table 1 because we have knowledge about these metrics' thresholds from both literature (McCabe, 1976) and Predictive tool's documentation. Furthermore, it has been shown that these metrics are very useful for software fault prediction (Menzies et al., 2007).

In order to explain the methods we developed, two algorithms are presented in Figs. 1 and 2. Because both of these algorithms improved the prediction performance when used standalone on datasets, we also investigated the sequential usage of them. This two-stage approach achieved the best performance on public NASA datasets.

3.1. Algorithm 1 (outlier detection approach)

In this algorithm, we mark the data object as outlier if the majority of seven metrics exceed corresponding metrics thresholds and if the data object's class label is not-faulty. We used seven metrics in our experiments, and hence the majority level was four. That means a data object is outlier if any of its four metrics exceed metrics thresholds and if its class label is not-faulty. Algorithm 1's pseudo code is presented in Fig. 1.

3.2. Algorithm 2 (outlier detection approach)

In this second algorithm, we mark the data objects as outliers if all the metrics are below metrics thresholds and if the class label is faulty. Algorithm 2's pseudo code is presented in Fig. 2.

3.3. Our two-stage outlier detection approach

Our two-stage outlier detection approach is the sequential usage of Algorithms 1 and 2. This approach finds outliers that arise both in faulty and not-faulty modules Fig. 3.

3.4. Classifiers based on machine learning

In order to evaluate the performance of our outlier detection methods, we built fault prediction models by using high-performance machine learners: Naive Bayes (Menzies et al., 2007) and Random Forests (Ma et al., 2006).

3.4.1. Naive Bayes

Naive Bayes is a probabilistic classifier and has strong independent assumptions. Because Naive Bayes is a well-known machine learning algorithm, details are skipped here.

3.4.2. Random Forests

Random Forests is another well-known classification algorithm that depends on the majority votes of trees. Koprinska, Poon, Clark, and Chan (2007) reported that Random Forests provide better performance than Naive Bayes on spam e-mail filtering problem which unbalanced datasets are used as in this study. Details of this algorithm are skipped here.

Table 1
Metrics and thresholds.

Metric	Abbreviation	Threshold value
Lines of code	LoC	65
Cyclomatic complexity	CC	10
Essential complexity	EsC	4
Unique operator	UOpr	40
Unique operand	UOpnd	25
Total operator	TOpr	125
Total operand	TOpnd	70

```
Algorithm 1 (Detection of class outliers labeled as not-faulty)
Inputs: THR_LOC ← 45, THR_CC ←10,THR_ESC ←4,
        THE_UOPR ← 40, THR_UOPND ← 25,  THR_TOPR ←125, THR_TOPND  ← 70,
        MAJORITY_LEVEL ← 4
        DATAS ← (kc1,kc2,pc1,cm1,jm1)
        majorityCounter ← 0
Outputs: DATAS' ← (kc1',kc2',pc1',cm1',jm1')

for data in DATAS
        if data.LOC >=THR_LOC then
                majorityCounter  ←  majorityCounter +1
        if data.CC >=THR_CC then
                majorityCounter  ←  majorityCounter +1
        if data.ESC >=THR_ESC then
                majorityCounter  ←  majorityCounter +1
        if data.UOPR>=THR_UOPR then
                majorityCounter  ←  majorityCounter +1
        if data.UOPND >=THR_UOPND then
                majorityCounter  ← majorityCounter +1
        if data.TOPR >=THR_TOPR then
                majorityCounter  ← majorityCounter +1
        if  data.TOPND >=THR_TOPND then
                majorityCounter  ←  majorityCounter +1
        if  majorityCounter >= MAJORITY_LEVEL AND
          data.CLASSLABEL = NOT-FAULTY then
                remove(data)
end for
End Algorithm
```

Fig. 1. Algorithm 1.

```
Algorithm 2 (Detection of class outliers labeled as faulty)
Inputs: THR_LOC ← 45, THR_CC ← 10, THR_ESC ← 4,
        THE_UOPR ← 40, THR_UOPND ← 25,  THR_TOPR ← 125, THR_TOPND ← 70,
        DATAS ← (kc1,kc2,pc1,cm1,jm1)
Outputs: DATAS' ←(kc1',kc2',pc1',cm1',jm1')

for data in DATAS
        if data.LOC < THR_LOC AND
          data.CC < THR_CC AND
          data.ESC < THR_ESC AND
          data.UOPR < THR_UOPR AND
          data.UOPND < THR_UOPND AND
          data.TOPR < THR_TOPR AND
          data.TOPND < THR_TOPND AND
          data.CLASSLABEL = FAULTY then
                remove(data)
end for
End Algorithm
```

Fig. 2. Algorithm 2.

```
Algorithm 3 (Detection of all the class outliers )
Inputs: DATAS ← (kc1,kc2,pc1,cm1,jm1)
Outputs: DATAS'' ← (kc1',kc2',pc1',cm1',jm1')
    1.  Apply Algorithm 1 to dataset DATAS
    2.  Apply Algorithm 2 to dataset DATAS'
End Algorithm
```

Fig. 3. Our two-stage outlier detection approach.

4. Experiment description

In this section, experiment definition, context and variables selection, hypotheses formulation, selection of subjects, design of the experiment, instrumentation, validity evaluation, and the experiment operation are described. These subsections and the reporting format of this paper are based on the guidelines given in Wohlin et al.'s experimental software engineering book (Wohlin et al., 2000).

4.1. Experiment definition

The objects, purpose, quality focus, perspective, and context of experimentation should be provided in an experimental software engineering study. Software metrics and fault data are the objects of this study. The purpose is to detect class outliers and improve the performance of software fault prediction models. The quality focus is the effectiveness of fault prediction models and the researcher's point of view is the perspective of this study. NASA datasets were used during our experiments and the projects where these datasets come from are the context of this study.

4.2. Context selection

KC1, KC2, PC1, CM1 and JM1 public datasets from PROMISE repository were used during our experiments. Table 2 shows some features of these datasets. The datasets we used are named with their original name and the number of data objects such as JM1-10876.

4.3. Variables selection

The dependent variable is the fault-proneness label and independent variables are seven software metrics. Dependent variable can have only one of two labels: fault-prone or not fault-prone. Independent metrics used in these experiments are described as follows:

- *Lines of code* (*LoC*): The total number of code lines including blank lines and comment lines.
- *Cyclomatic complexity* (*CC*): The total number of independent paths on the flow graph.
- *Essential complexity* (*EsC*): Represents the unstructured nature of the module. "if-then-else" and "while" loops are well structured constructs. Use of "goto" statements increases the EsC value.
- *Unique operator* (*UOp*): The total number of *unique* operators; if the same operator appears several times, it is counted once.
- *Unique operand* (*UOpnd*): The total number of *unique* operands; if the same operand appears several times, it is counted once.
- *Total operator* (*TOp*): The total number of operators appearing in a module. Some examples for operators are logical operations, arithmetical operations, bitwise operations, and every keyword (return, sizeof, etc.) that causes an action to the operand. Function calls are also considered as a single operator.
- *Total operand* (*TOpnd*): The total number of operands appearing in a module. The tokens not identified as operators are considered operands.

4.4. Hypotheses formulation

We defined three hypotheses prior to the execution of our experiments and each hypothesis was identified according to our research questions. Hypotheses are listed as follows:

- **HP1:** When outliers are removed by using Algorithm 1, the performance of fault prediction models increase. (*Null hypothesis*: When outliers are removed by using Algorithm 1, the performance of fault prediction models does not increase.)
- **HP2:** When outliers are removed by using Algorithm 2, the performance of fault prediction models increase. (*Null hypothesis*: When outliers are removed by using Algorithm 2, the performance of fault prediction models does not increase.)
- **HP3:** When outliers are removed by using two-stage outlier detection approach, the performances of fault prediction models increase. (*Null hypothesis*: When outliers are removed by using two-stage outlier detection approach, the performances of fault prediction models do not increase.)

Table 2
Datasets.

Dataset	Lang.	LOC (K)	Project	Fault (%)	#Methods
KC2-520	C++	43	Data processing	21	520
CM1-496	C	20	Instrument	21	496
PC1-1107	C	40	Flight software	7	1107
JM1-10876	C	315	Real time	19	10,876
KC1-2107	C++	43	Storage management	15, 45	2107

4.5. Selection of subjects

Chosen datasets are widely used ones from PROMISE repository and subjects are developers worked on these NASA projects.

4.6. Design of the experiment

Three different experiments have been conducted to answer three research questions and to evaluate three hypotheses explained in previous sections. The design is one factor (fault prediction performance), with four treatments. The first treatment is prediction without any outlier removal process. Second, third, and fourth one is prediction after class outliers are removed by using Algorithms 1 and 2, and two-stage outlier detection approach, respectively.

4.7. Instrumentation and measurement

Datasets from PROMISE repository were used as is and we did not give any material to subjects of experiments.

4.8. Validity evaluation

4.8.1. External validity

This experimental study uses public NASA datasets collected from professional projects, and these projects were developed in an industrial environment. Therefore, results can be generalized. However, the performance of our outlier detection method can change for datasets coming from different sources.

4.8.2. Conclusion validity

In this study, 10-fold cross validation was used to produce statistically sound results.

4.8.3. Internal validity

FPR, FNR, and error parameters were calculated for evaluation.

4.8.4. Construct validity

Validated metrics were used and there is no threat for construct validity.

4.9. Experiment operation

The experiments were done at TUBITAK-Marmara Research Center. Matlab and WEKA tools were used for experiments.

5. Analysis of the experiment

This section shows the analysis of our experiments. We used Naive Bayes and Random Forests algorithms to evaluate the performances of our outlier detection approaches.

In this study, we used FPR (false positive rate), FNR (false negative rate) and error parameters to evaluate the performance. Error is the percentage of mislabeled modules, false positive rate (FPR) is the percentage of not-faulty modules labeled as faulty by the model, and false negative rate (FNR) is the percentage of faulty modules labeled as not-faulty. Confusion matrix is shown in Table 3. Eqs.

Table 3
Confusion matrix.

		Actual labels	
		YES	NO
Predicted labels	YES	True-Positive (TP)	False-Positive (FP)
	NO	False-Negative (FN)	True-Negative (TN)

(1)–(3) calculate FPR, FNR and error values, respectively. A trade-off exists between FPR and FNR values, but FNR value is much more vital. If FNR is high, this means that faulty modules could not be detected and these modules will not be tested rigorously prior to end user delivery.

$$FPR = \frac{FP}{FP + TN}, \tag{1}$$

$$FNR = \frac{FN}{FN + TP}, \tag{2}$$

$$Error = \frac{FN + FP}{TP + FP + FN + TN}. \tag{3}$$

When outliers are removed with our outlier detection approaches, FPR, FNR, and error parameters are expected to decrease. Following experiments showed that these evaluation parameters decreased as expected.

5.1. Experiment 1

Experiment 1 was conducted to evaluate Hypothesis 1 (HP1). We used Algorithm 1 to remove outliers and then, we built fault prediction models on the remaining dataset by using Naive Bayes and Random Forests classifiers. Results of this experiment are shown as "Algorithm 1" from Tables 4–13. "No Method" row, in each table, shows the results of predictors when none of our outlier detection algorithms is applied and when datasets are used as are.

Table 4
Naive Bayes results on kc1-2107.

Naive Bayes	FPR (%)	FNR (%)	Error (%)
No method	8.9	63.1	17.2
Algorithm 1	9.2	56.0	16.6
Algorithm 2	6.7	16.9	7.36
Our approach	6.7	5.9	6.60

Table 5
Random Forests results on kc1-2107.

Random Forests	FPR (%)	FNR (%)	Error (%)
No method	5.5	72..9	15.8
Algorithm 1	4.7	63.7	14.0
Algorithm 2	2.0	50.8	5.05
Our approach	1.0	30.5	2.92

Table 6
Naive Bayes results on kc2-520.

Naive Bayes	FPR (%)	FNR (%)	Error (%)
No method	5.1	52.8	14.8
Algorithm 1	6.6	38.7	13.4
Algorithm 2	5.6	36.1	10.0
Our approach	6.1	9.7	6.66

Table 7
Random Forests results on kc2-520.

Random Forests	FPR (%)	FNR (%)	Error (%)
No method	8.5	52.8	17.5
Algorithm 1	4.6	48.1	13.8
Algorithm 2	5.6	36.1	10.0
Our approach	3.1	23.6	6.23

Table 8
Naive Bayes results on pc1-1107.

Naive Bayes	FPR (%)	FNR (%)	Error (%)
No method	5.4	73.7	10.1
Algorithm 1	3.6	60.5	7.91
Algorithm 2	4.9	58.3	7.32
Our approach	3.4	45.8	5.46

Table 9
Random Forests results on pc1-1107.

Random Forests	FPR (%)	FNR (%)	Error (%)
No method	1.6	75.0	6.68
Algorithm 1	1.0	52.6	4.90
Algorithm 2	1.5	79.2	4.91
Our approach	1.1	41.7	3.09

Table 10
Naive Bayes results on cm1-496.

Naive Bayes	FPR (%)	FNR (%)	Error (%)
No method	8.0	70.8	14.1
Algorithm 1	6.1	52.1	11.1
Algorithm 2	7.4	54.8	10.4
Our approach	5.1	29.0	6.87

Table 11
Random Forests results on cm1-496.

Random Forests	FPR (%)	FNR (%)	Error (%)
No method	2.2	81.3	9.87
Algorithm 1	3.1	60.4	9.33
Algorithm 2	2.0	90.3	7.72
Our approach	1.5	41.9	4.50

Table 12
Naive Bayes results on jm1-10876.

Naive Bayes	FPR (%)	FNR (%)	Error (%)
No method	5.3	78.9	19.4
Algorithm 1	3.7	63.2	16.1
Algorithm 2	4.7	68.6	12.6
Our approach	2.8	41.4	8.00

Table 13
Random Forests results on jm1-10876.

Random Forests	FPR (%)	FNR (%)	Error (%)
No method	6.3	76.6	19.9
Algorithm 1	4.7	57.8	15.8
Algorithm 2	4.5	69.9	12.6
Our approach	3.0	42.4	8.39

For example, Tables 4 and 5 present the results of Experiment 1 on KC1-2107 dataset. For all the datasets, we noticed that Algorithm 1 provides a decrease in FNR and error values. This means that the performance of fault predictors based on Naive Bayes and Random Forests improve when outliers are removed with this Algorithm 1. FPR value mostly decreases with this algorithm, too.

5.2. Experiment 2

Experiment 2 was conducted to evaluate Hypothesis 2 (HP2). We used Algorithm 2 to remove outliers and then, we built fault prediction models on the remaining dataset by using Naive Bayes and Random Forests classifiers. Results of this experiment are shown as "Algorithm 2" from Tables 4–13. For all the datasets, we noticed that Algorithm 2 mostly provides a decrease in FNR and error values. FPR value mostly decreases with this algorithm, too.

5.3. Experiment 3

Experiment 3 was conducted to evaluate Hypothesis 3 (HP3). We used our two-stage approach to remove outliers and then, we built fault prediction models on the remaining dataset by using Naive Bayes and Random Forests classifiers. Results of this experiment are shown as "Our Approach" from Tables 4–13. For all the datasets, we noticed that this approach provides a decrease in FNR, FPR and error values except on KC2-520 dataset as shown in Table 6. For example; as shown in Table 4, on KC1-2107 dataset, FNR value is 63.1% when Naive Bayes algorithm is used. After we remove outliers with our approach, FNR value decreases to 5.9%. Furthermore, FPR and error values decrease from 8.9% to 6.7% and from 17.2% to 6.6%, respectively.

6. Conclusions and future works

In this study, we proposed an outlier detection approach and validated it on public software measurement datasets. Experimental results show that our approach improves the performance of software fault prediction algorithms such as Naive Bayes and Random Forests according to the FPR, FNR, and error parameters after outliers are removed. This method is based on the software metrics thresholds and it can be customized according to the different thresholds. Class labels were used in this study and hence, this approach is suitable for supervised fault prediction which requires priori fault data. The experiments we conducted reveal that our outlier detection approach, a sequential usage of our Algorithms 1 and 2, gives much better results than the standalone usage of Algorithms 1 and 2. Further work will be developing a threshold calculation method to define the metrics thresholds that can be used in this approach. As done in this study, metrics thresholds can be chosen from industrially accepted levels but integrating a threshold calculation method into our outlier detection approach will enable us to extend this approach for different domains where outlier detection is crucial. As a result, this study showed that thresholds based outlier detection approach that uses the industry thresholds levels is an effective and efficient approach to detect class outliers in the field of software engineering.

Acknowledgments

This project is supported by TUBITAK (The Scientific and Technological Research Council of TURKEY) under Grant 107E213. The findings and opinions in this study belong solely to the authors, and are not necessarily those of the sponsor.

References

Breunig, M. M., Kriegel, H. P., Ng, R. T., & Jörg Sander (2000). LOF: Identifying density-based local outliers. In *SIGMOD conference* (pp. 93–104).

Catal, C., & Diri, B. (2009a). Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem. *Information Sciences, 179*(8), 1040–1058.

Catal, C., & Diri, B. (2009b). A systematic review of software fault prediction studies. *Expert Systems with Applications, 36*(4), 7346–7354.

Halstead, M. (1977). *Elements of software science*. New York: Elsevier.

He, Z., Xu, X., & Deng, S. (2003). Discovering clustering-based local outliers. *Pattern Recognition Letters, 24*(9–10), 1641–1650.

Hodge, V. J., & Austin, J. (2004). A survey of outlier detection methodologies. *Artificial Intelligence Review, 22*, 85–126.

Jiang, M., Tseng, S., & Su, C. (2001). Two-phase clustering process for outlier detection. *Pattern Recognition Letters, 22*, 691–700.

Knorr, E. M., & Ng, R. T. (1998). Algorithms for mining distance-based outliers in large datasets. In *Proceeding of 24th international conference on very large data bases, VLDB* (pp. 392–403).

Koprinska, I., Poon, J., Clark, J., & Chan, J. (2007). Learning to classify e-mail. *Information Sciences, 177*(10), 2167–2187.

Ma, Y., Guo, L., & Cukic, B. (2006). *A statistical framework for the prediction of fault-proneness. Advances in machine learning application in software engineering*. Idea Group Inc.. 237-265.

McCabe, T. (1976). A complexity measure. *IEEE Transactions on Software Engineering, 2*(4), 308–320.

Menzies, T., Greenwald, J., & Frank, A. (2007). Data mining static code attributes to learn defect predictors. *IEEE Transactions on Software Engineering, 33*(1), 2–13.

Munson, J. C. (2003). *Software engineering measurement*. Boca Raton, FL: Auerbach Publications.

Ramaswamy, S., Rastogi, R., & Shim, K. (2000). Efficient algorithms for mining outliers from large datasets. In *Proceedings of SIGMOD* (pp. 427–438).

Seliya, N., & Khoshgoftaar, T. M. (2007). Software quality estimation with limited fault data: A semi-supervised learning perspective. *Software Quality Journal, 15*(3), 327–344.

Tang, W., & Khoshgoftaar, T. M. (2004). Noise identification with the *k*-means algorithm. *ICTAI, 2004*, 373–378.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., & Wesslen, A. (2000). *Experimentation in software engineering: An introduction*. Norwell, MA: Kluwer Academic Publishers.

Zhong, S., Khoshgoftaar, T. M., & Seliya, N. (2004). Analyzing software measurement data with clustering techniques. *IEEE Intelligent Systems, 19*(2), 20–27.