How Discretization Affects Software Defect Detection Tasks: An Experimental Study

L. Akritidis[1], P. Bozanis[1]

[1]School of Science and Technology, International Hellenic University

16th International Conference on Information, Intelligence, Systems and Applications (IISA 2025)

Mytilene, Island of Lesbos, Greece,

July 10-12, 2025

Software Defect Detection

- Software defect detection constitutes a fundamental process in the life cycle of software development.
- Its goal is to **identify errors and problems in the source code** of a program that may affect its functionality or performance.
- Detecting defects early during the development stage is essential to ensure **reliability**, **robustness**, and **overall quality**.
- Therefore, the problem of software defect detection has attracted the attention of numerous researchers.
- In most cases, **software defect detection is treated as a typical binary classification problem.**

Discretization

- Discretization is the process of **transforming continuous variables into discrete ones**.
- Such a transformation introduces the risk of information loss.
- However, it is frequently proved beneficial, due to its **noise reduction** and **outlier absorption** capabilities.
- Previous works in the relevant literature have demonstrated that discretization may have beneficial effects in classification performance.

Our research

- This paper is the first to examine the magnitude of these effects in software defect detection tasks.
- We conducted a large scale experimental survey with:
  - 7 supervised and unsupervised discretization techniques,
  - 26 datasets and,
  - 6 classifiers.

Our proposed heuristic

- We also **introduce a simple heuristic method that automatically determines the most suitable number of bins per feature column**.
- The proposed heuristic **utilizes the Bayesian Gaussian mixture (BGM)** model, **aiming at capturing different modes in the value distribution of each feature column**.
- The experiments proved that in several scenarios, this approach **can indeed improve** the ability of a discretizer to enhance detection performance.

Discretization Methods

- Unsupervised techniques:
  - Equal width/frequency binning: distributes the values into a set of bins of equal widths of frequencies (each bin has the same number of values).
  - k-Means/GMM binning: groups similar continuous values into bins by executing the k-Means/GMM clustering algorithms.
- Supervised Techniques:
  - CAIM: Performs discretization by applying a class interdependency maximization criterion. It identifies the smallest number of intervals.
  - ChiMerge: Merges adjacent intervals by estimating how similar their relevant class frequencies are. This is achieved by computing their χ^2 similarity with the class labels; if they are found to have similar class frequencies, they are merged.

The proposed method: BGM-ChiMerge

- We proposed an improvement to ChiMerge **with the aim of limiting the maximum number of intervals to be created**.
- More specifically, before executing ChiMerge, **we first fit a Bayesian Gaussian Mixture (BGM) model on the column values** in order to capture different modes in their distribution.
- Subsequently, **the maximum number of bins is determined by the non-zero weights of the BGM modes**.

Datasets & Classifiers

- 26 benchmark datasets have been used for performance evaluation.
  - Origin: NASA MDP and PROMISE collections.
  - They contain healthy and defective code examples.
  - They include numerical attributes that quantify software quality measures.
- We used 6 classifiers:
  - Logistic Regression (LR), Multilayer Perceptron (MLP), C4.5, Random Forest (RF), Support Vector Machines (SVM), and XGBoost.
- Evaluation measures: Accuracy, Balanced Accuracy, F1 score.

TABLE I: Software Defect Detection Datasets

#	Dataset	m	n	$\|Y\|$
1	AR1	121	29	2
2	AR4	107	29	2
3	CM1	498	21	2
4	KC1	2109	21	2
5	KC3	458	39	2
6	MC2	161	39	2
7	PC1	1109	21	2
8	PC3	1563	37	2
9	ANT-1.3	125	20	2
10	ANT-1.5	293	20	2
11	ANT-1.7	745	20	2
12	CAMEL-1.2	608	20	2
13	CAMEL-1.4	872	20	2
14	CAMEL-1.6	965	20	2
15	IVY-1.1	111	20	2
16	IVY-1.4	241	20	2
17	IVY-2.0	352	20	2
18	JEDIT-3.2	272	20	2
19	JEDIT-4.1	312	20	2
20	JEDIT-4.2	367	20	2
21	JEDIT-4.3	492	20	2
22	LOG4J-1.0	135	20	2
23	LOG4J-1.2	205	20	2
24	SYNAPSE-1.0	157	20	2
25	SYNAPSE-1.1	222	20	2
26	SYNAPSE-1.2	256	20	2

Results

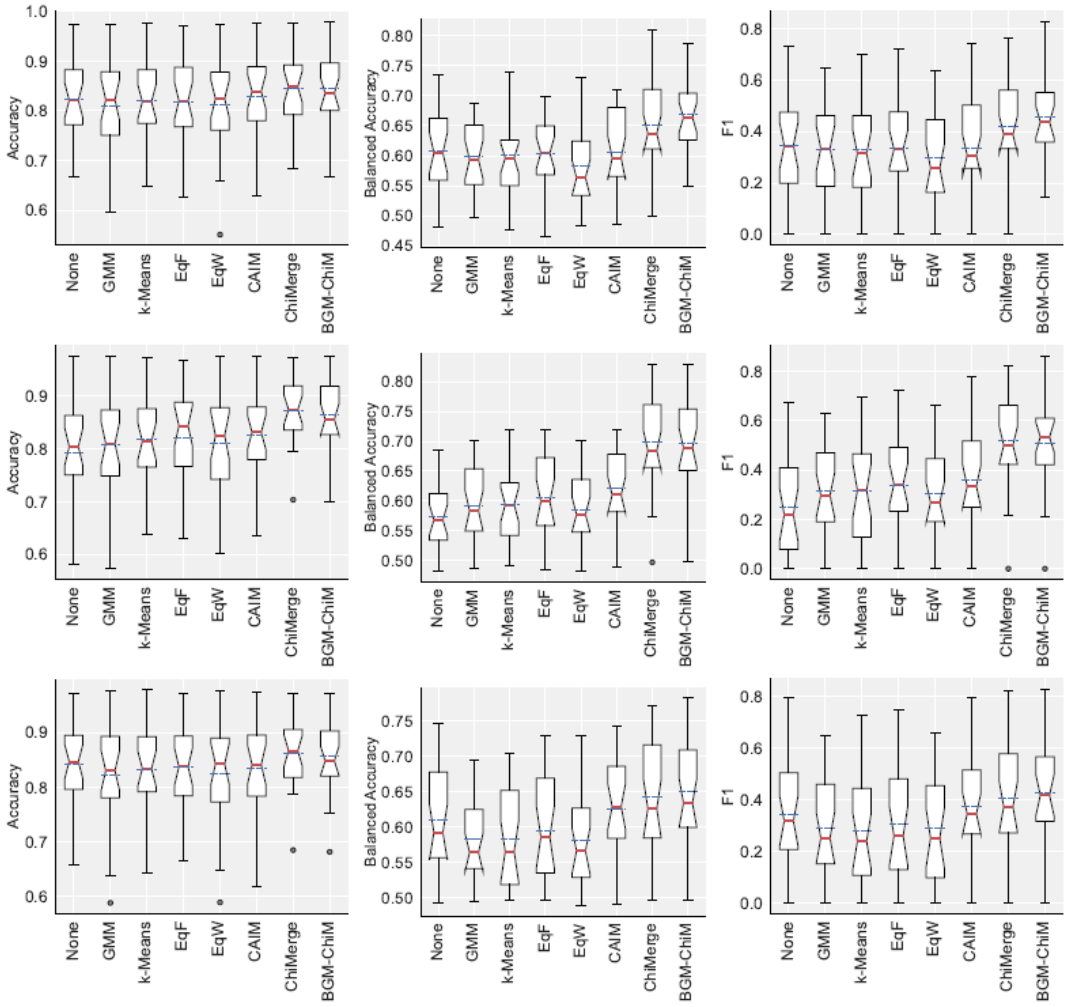

Fig. 1: Software defect detection performance of various discretization methods combined with the XGBoost (top), Multilayer Perceptron (middle), and Random Forest (bottom) classifiers in terms of Accuracy (left), Balanced Accuracy (Center) and F1 score (right).

Results

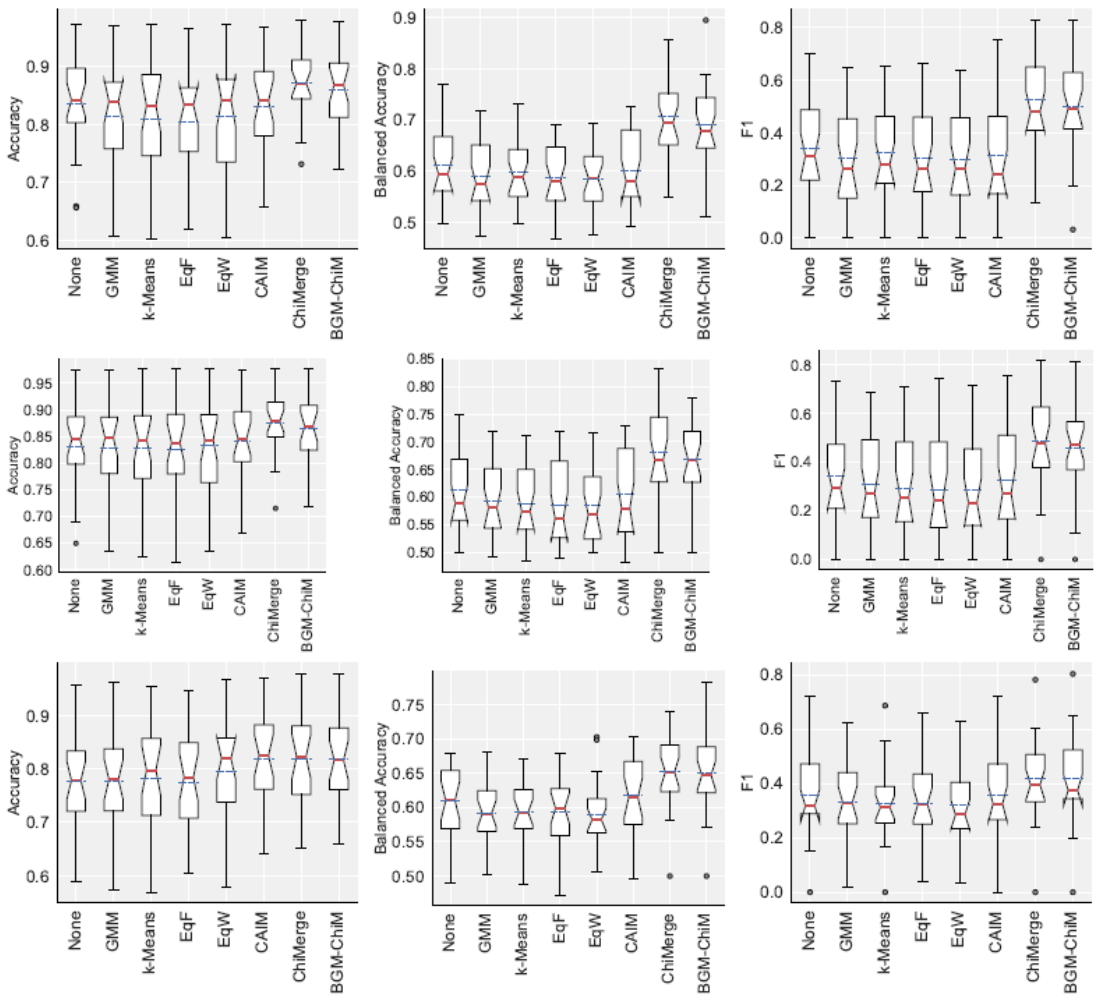

Fig. 2: Software defect detection performance of various discretization methods combined with Linear SVM (top), Logistic Regression (middle), and Decision Tree (bottom) classifiers in terms of Accuracy (left), Balanced Accuracy (Center) and F1 score (right).

Statistical Significance

- The statistical significance of the aforementioned experiments was confirmed by running a series of Friedman post-hoc non-parametric tests.

TABLE II: Post-hoc Statistical Significance Analysis with the Friedman Non-Parametric Test

Classifier	Accuracy	Bal. Accuracy	F1
XGBoost	$3.212 \cdot 10^{-10}$	$2.941 \cdot 10^{-15}$	$7.163 \cdot 10^{-15}$
MLP	$1.248 \cdot 10^{-15}$	$1.306 \cdot 10^{-18}$	$1.052 \cdot 10^{-21}$
Random Forest	$5.739 \cdot 10^{-12}$	$1.288 \cdot 10^{-13}$	$2.241 \cdot 10^{-14}$
SVM	$4.175 \cdot 10^{-20}$	$1.105 \cdot 10^{-17}$	$6.245 \cdot 10^{-18}$
C4.5	$4.862 \cdot 10^{-18}$	$2.869 \cdot 10^{-12}$	$8.424 \cdot 10^{-13}$
Log. Regression	$8.789 \cdot 10^{-18}$	$2.802 \cdot 10^{-17}$	$7.622 \cdot 10^{-18}$

Discussion

- Each diagram displays 8 Whisker plots that correspond to the 7 discretization methods plus the None case.
- Each box is created by considering 26 values (one per dataset) of one evaluation measure. The lower and upper edges of each box represent the 25th and 75th percentiles, respectively. That is, the median values of the lower and upper halves of the set of the 26 measurements. The continuous and dashed lines inside each box denote the median and mean values of the evaluation measure, respectively.
- On average, the only two methods that consistently improve classification performance are ChiMerge and our proposed BGM-ChiM.
- Regarding the other discretizers, their performance against the None scenario (i.e., when no discretization was applied) was rather blurry. On several datasets, classifiers and evaluation measures they were found beneficial, whereas in some others, they significantly degraded the quality of detection.
- The general outcome of this study is that, on average, ChiMerge and BGM-Chi can better capture the underlying distribution of the continuous features.

Conclusions & Future Work

- In this study, we examined the effects of discretization in software defect detection tasks.
- We designed a thorough experimental procedure and verified the performance of 7 such techniques on 26 datasets with 6 classification models.
- We introduced an improved version of ChiMerge, called BGM-ChiM, with the aim of setting an upper limit to the number of the created intervals. Initially, BGM-ChiM fits a Bayesian Gaussian Mixture to the values to be discretized, and determines the upper limit of the intervals from the number of the non-zero mode weights.
- The results indicated that the simple unsupervised techniques, such as equal-width and equal-frequency binning, are in general harmful for detection accuracy, as they miss critical information about the class-dependent distributions of the feature values.
- In contrast, several supervised discretization techniques, and especially BGM-ChiM and ChiMerge, may consistently improve the classification performance.

Thank you for watching

I would be happy to answer your questions.

Please send them to lakritidis@ihu.gr