卒 業 研 究 報 告 書

題　目　　RBMを用いたソフトウェアメトリクスの
特徴抽出手法の提案
(Unsupervised Defect Prediction
with Restricted Boltzmann Machine)

指導教員　　　水野　修　教授

京都工芸繊維大学 工芸科学部 情報工学課程

学生番号　　13122502

氏　　名　　近藤　将成

平成29年2月14日提出

RBM を用いたソフトウェアメトリクスの 特徴抽出手法の提案 (Unsupervised Defect Prediction with Restricted Boltzmann Machine)

平成 29 年 2 月 14 日

13122502　近藤　将成

Abstract

The defect prediction is one of the important tasks to preserve an assurance of software quality in the software engineering. In previous works of the defect predicion, two issues are identified. First, there is an issue of a heterogeneous metrics set. Many researchers use a supervised learning approach as to generate a defect prediction model. Then, they have to collect a training dataset that has same metrics with an objective dataset. This reduces the amount of available data, and thus should be improved. Second, it is difficult to choose the fittest model since there is an issue of differences with accuracy of a model between datasets. Various solutions have been reported regarding the issue of a heterogeneous metrics set. For instance, Some researchers apply an unsupervised learning method to an object dataset since unsupervised learning methods have an advantage that they do not need a training dataset. However, less research has focused the second issue. In this paper, we propose an unsupervised learning approach using Restricted Boltzmann Machine as preprocessing of metrics to solve the second issue. We conduct experiments on three empirical datasets. These results show that differences between five unsupervised learning methods are reduced, and all of them belong to the group which has the best AUC values. Furthermore, we confirm that unsupervised learning methods with Restricted Boltzmann Machine as preprocessing of metrics are effective on the source code complexity metrics.

Index

1. Introduction

The defect prediction is one of the important tasks in the software engineering. For instance, in the test task, software project managers have to manage limited resources to allocate to tested files. If they can detect defects of an early stage, it is possible to reduce the cost of development[1]. Thus, effective defect prediction approaches will contribute the improvement of software development.

To solve the defect prediction problem, many researchers have built defect prediction models and almost adopt the supervised learning (SVL). (e.g. neural network, logistic regression, etc.) to their datasets [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]. New metrics for the defect prediction, which are used to extract features from original information (e.g. source code, commit log, etc.) are also reported in many research works[14, 15, 16, 17, 18].

However, some approaches are not applied to industrial situations since it is difficult to collect datasets that have enough samples to build a model in industries[19]. To solve the problem of the SVL approach applied to the defect prediction, we can adopt cross-company datasets[20, 21, 22, 23, 24, 25]. Cross-company datasets contain data collected from multiple companies. In this case, we can use more data than that from single company dataset. However, there is also a problem of a heterogeneous metrics set, which derives inconsistent metrics of data between training datasets and test datasets[20].

To avoid the training data issue, unsupervised leaning (USVL) methods can be used[19, 26, 27, 28, 29, 30]. Since the USVL methods do not need training datasets, the problem of a heterogeneous metrics set between training dataset and test dataset does not happen. However, previous research works are reported that accuracy of classification methods have a variation[31, 32]. Since previous research works reported that the best model may change by an object dataset, finding the best model for a certain situation is also an important issue[32]. Results of previous research work using USVL methods also show that the variation in performance of these methods on different datasets is considerable[19].

Our hypothesis is that the valiation in performance is caused by the selection of metrics that are used to do the defect prediction. In this paper, we adopt USVL approach instead of SVL approach to avoid the training data issue, and the Restricted Boltzmann Machine (RBM)[33] to automatically extract features from datasets, which can be applied to unsupervised situations since RBM can extract features from original data in USVL. We call this approach as USVL with RBM. RBM can extract features that are clearly separable in two classes. Therefore, RBM improve stability since all USVL with RBM methods can use features that are easy to classify.

Our major findings in this work are as follows:

- All USVL methods have same AUC values when we applied features extracted from source code metrics by RBM to USVL methods. This result is better than the previous USVL approach from the viewpoint of having a small variation between the performance of defect prediction approaches.
- These AUC values are small better than previous USVL methods.
- Feature extraction approach of RBM is better than traditional feature extraction approach of PCA in this experimental setup.

The organization of our paper is as follows: Section 2 introduces related work and another related method. Section 2.2 indicates datasets and a modeling approach. Section 2.3 presents the experiment, its results, and discussions. Section 2.4 describe threats to validity. Section 2.5 presents conclusion.

2. Background

In this section, we introduce related work such as the defect prediction and the model selection, the Boltzmann machine, and the restricted Boltzmann machine.

2.1 Related Work

2.1.1 Defect Prediction

The Software Quality Assurance (SQA) is an activity of improving software quality prior to release. For instance, activities such as the code inspection, the unit testing, and the source code control, are included. Among them, the testing activity needs much effort in a huge software project since the objective testing code becomes huge, and there are limited resources to test them. If they can detect defects in an early stage of development, it is possible to reduce the cost of development[1]. Therefore effective defect prediction approaches will contribute the improvement of the software development.

In the SVL, they need training data and test data. The training data and test data are collected from the same project or a similar project from target projects in the same organization, which is called Within Project Defect Prediction (WPDP). However, sometimes, it is difficult to collect sufficient training data in the same organization.

Thus, the Cross Project Defect Prediction (CPDP) is researched and applied to the SVL. In the CPDP, datasets contain data collected from multiple companies as a training dataset in prediction. The CPDP gives a solution for problems of the WPDP since a dataset has enough data in general. Thus, the CPDP is one of important research themes in the software engineering. However, the CPDP has some issues reported in previous research works[34, 35, 36, 37, 20]. For instance, the CPDP has a problem of a heterogeneity[20]. We cannot converge datasets collected from multiple companies to a dataset or apply datasets multiple companies to an object dataset since they sometimes contain different metrics set[20].

The other hand, the USVL can be applied to the defect prediction to solve problems of CPDP[19, 26, 27, 28, 29, 30]. The USVL has an advantage that they do not need a training dataset. Thus, they are not affected by the heterogeneous metrics set, and we can apply the

USVL to the defect prediction[19, 26, 27, 28, 29, 30]. Recently, Zhang et al. have summarized the accuracy of several USVL and SVL methods on the defect prediction[19]. They concluded that the connectivity-based USVL has as good accuracy as SVL methods. Therefore, it is possible to solve the problem of CPDP by USVL methods.

2.1.2 Model Selection

It is difficult to select the model fitted to the objective data since the best model may change by objective datasets[31, 32]. Thus, we have to solve the problem selecting a model fitted an objective dataset.

For instance, Ghotra et al. [31] reported that the defect prediction model groups have statistical distance between defect prediction models in cleaned version of the NASA dataset and the PROMISE dataset. The cleaned version means a dataset that noises or biases on the original datasets are removed. In this case, when they use the not cleaned NASA dataset, each defect prediction model has the same accuracy. However, when they use the cleaned version NASA dataset, each defect prediction model has a different accuracy. It means that it is difficult to select a better or the best prediction model to get the best accuracy since they depend on an objective dataset.

Moreover, in the effort estimation, Gray et al. [32] reported that three regression models (i.e. least-squares, least-median-squares, and least-trimmed-squared) and the neural network are applied to three datasets. Their result showed that the best model is changed by datasets. It means that we have to focus on the dataset to select the best prediction model.

Thus, it is a problem for us to select the model fitted an objective data. We aim to have a small variation between the performance of models by applying USVL methods to different datasets.

2.1.3 Neural Network and RBM

Many researchers reported applying neural network to the defect prediction[38, 39, 40, 41, 42, 43] since neural network is one of the good machine learning approach, which can learn arbitrary functions[44]. Recently, deep neural network (DNN) is also applied to the defect prediction, and deep belief network (DBN) is applied to extracting feature from source code in the defect prediction[3, 5]. DNN and DBN adopt RBM as preprocessing for parameters of a network as extracting features. However, there are no works applying RBM to the USVL as preprocessing of extracting features. Thus, we apply RBM to the USVL task as preprocessing of metrics.

2.1.4 Boltzmann Machine

We assume that the observation variables $x_1, \cdots, x_N$ are generated by the true probability distribution $p_g(\boldsymbol{x})$. Then, in a certain probability distribution $p(\boldsymbol{x}|\boldsymbol{\theta})$, $\boldsymbol{\theta}$ is parameters of a probability distribution, $\boldsymbol{\theta}$ is adjusted and, a certain probability distribution is closed to the true probability distribution by a maximum likelihood estimation. The likelihood function is as follow:

$$L(\boldsymbol{\theta}) = \prod_{n=1}^{N} p(\boldsymbol{x}_n|\boldsymbol{\theta}). \tag{2.1}$$

Then, the model is expressed by a Boltzmann Machine (BM). BM is a network that consists of a non-directed graph, and a state of each unit expressed by a stochastic variable is given by the certain probability distribution[45, 46].

When unit i is calculated to update its binary state (each unit has a binary state) by other connected units and bias of unit i, First the following equation is calculated:

$$z_i = b_i + \sum_j s_j w_{ij}. \tag{2.2}$$

where z_i is a total input to unit i, b_i is its bias, w_{ij} is a weight between unit i and active unit j, and s_j is a binary term that if unit j turn on, s_j is 1 and 0 otherwise. Second if unit i turns on, a probability distribution is given by following:

$$p(s_i = 1) = \frac{1}{1 + \mathrm{e}^{-z_i}} \tag{2.3}$$

The network eventually follows a Boltzmann distribution:

$$P(\mathbf{v}) = e^{-E(\mathbf{v})} / \sum_{\mathbf{v}} e^{-E(\mathbf{v})} \tag{2.4}$$

where $\mathbf{v}$ is the probability of a state vector, and energy function, $E(\mathbf{v})$, is as follow:

$$E(\mathbf{v}) = -\sum_i s_i^{\mathbf{v}} b_i - \sum_{i<j} s_i^{\mathbf{v}} s_j^{\mathbf{v}} w_{ij} \tag{2.5}$$

where $s_i^{\mathbf{v}}$ is the binary state of unit i. It is assigned by a state vector.

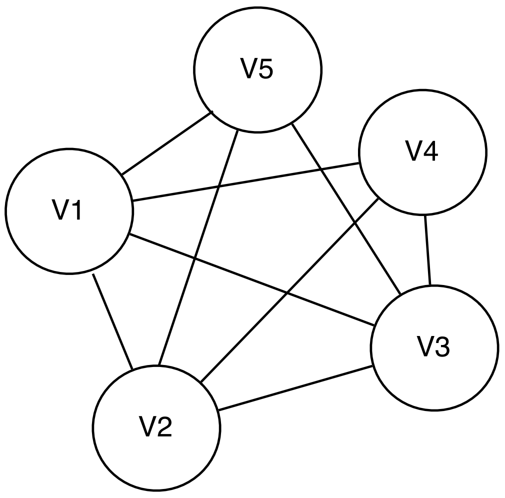


Figure 2.1 An example of Boltzmann machine

The weights can be represented in a matrix, which is a symmetric matrix and has zeros diagonal. Also, BM can have hidden units. Then, units are divided to visible units and hidden units. We can observe only visible units. BM with hidden units can express all distribution on the theory.

Figure 2.1 shows an example of a Boltzmann machine. A circle is a unit, and an edge is a connection between units.

2.1.5 Restricted Boltzmann Machine

The Restricted Boltzmann Machine (RBM) is also a probability graphical model[47]. The RBM is a BM that has hidden units and has a restriction of connection between units. It creates the probability occurrence distribution from data. The RBM add a restriction that connections are only between visible units and hidden units, and connections are not between units of same layers. Also, the energy function is as follow:

$$E(\mathbf{v}, \mathbf{h}) = -\sum_{i=1}^{n} \sum_{j=1}^{m} w_{ij} h_i v_j - \sum_{j=1}^{m} b_j v_j - \sum_{i=1}^{n} c_i h_i \tag{2.6}$$

where $\mathbf{v}$ is a vector of visible units, $\mathbf{h}$ is a vector of hidden units, m is the number of visible units, n is the number of hidden units, b and c are bias terms. The network will also be a Boltzmann distribution, eventually:

$$P(\mathbf{v}, \mathbf{h}) = e^{-E(\mathbf{v},\mathbf{h})} / \sum_{\mathbf{v},\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})} \tag{2.7}$$

For BM consisted of visible units and hidden units, it is hard to calculate a marginal likelihood estimation. The RBM is easier to calculate marginal likelihood estimation than BM with hidden units since it has restrictions of a connection. Also, it can express all distribution.

Recent years, RBM is applied to preprocessing of a classification task[48]. Also, the Deep Belief Network (DBN), which consist of a multiple RBM is applied to defect prediction. For instance, Wang et al. reported a defect prediction approach of generating new features from source code by DBN[3]. Also, Yang et al. reported a defect prediction approach of generating new features from change metrics by DBN[5]. However, in fields of bug prediction, RBM have not been applied to preprocessing of an unsupervised classification task. Thus, we apply RBM to a defect prediction task using the USVL.

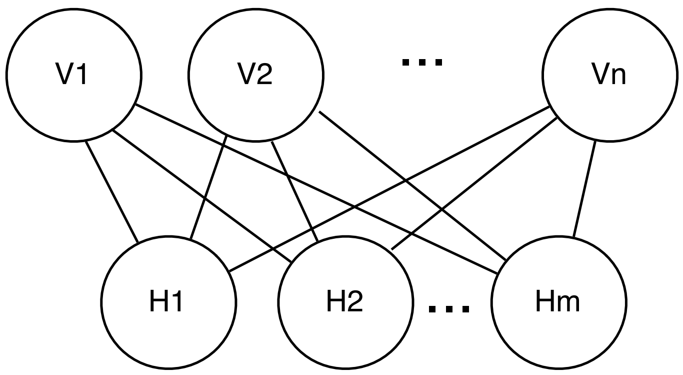


Figure 2.2 An example of restricted Boltzmann machine

Figure 2.2 shows the example of RBM. A circle labeled "V" is a visible unit, labeled "H" is a hidden unit.

2.2 Experimental Setup

In this section, we describe about the experimental setup. The organization of this section is as follows: Section 2.2.1 introduces RQ and what kind of competitions we conduct. Section 2.2.2 and 2.2.3 show datasets and metrics used in this paper. Section 2.2.4 indicates how to execute preprocessing of metrics. Section 2.2.5 shows the models used in this paper. Section 2.2.6 shows the evaluation measures and statistical tests. Section 2.2.7 introduces overview of experiment.

2.2.1 Research Questions and Competitions

To tackle problems of defect prediction, we have risen two research questions.

RQ1: How well do RBM preprocessing affect the unsupervised defect prediction using source code complexity metrics?

RQ2: Which is a better preprocessing between RBM and a traditional feature extraction approach?

To solve these research question, we need to conduct two competitions respectively by the statistical method:

1. **USVL with RBM vs. USVL:** We compare the performance of predictors based on either RBM or normalized metrics.
2. **USVL with RBM vs. USVL with the principal component analysis (PCA):** We select the PCA as representation of the traditional feature extraction approach.

2.2.2 Dataset

We applied our proposed approach to public repository datasets, since we want to compare our proposed approach with the conventional approach, and it is easy to conduct replication for future research. These datasets have been used in the previous research work[19]. Datasets used in this study are indicated in Table 2.1. Types of used metrics for RQ1 and RQ2 are indicated in Table 2.2. Each metrics is collected from an original source code entities (e.g. files or classes).

(1) PROMISE

The public repository PROMISE dataset is collected by Jureczko et al. from open-source and academic projects[49]. PROMISE dataset has different the number of metrics for projects. Thus, we select the projects following the previous research work[19].

(2) NASA

The public repository NASA dataset is partial of PROMISE dataset. NASA dataset is collected by Shepperd et al. from the original publicly available NASA datasets[50].

The project of NASA dataset consists of source code metrics (i.e. LOC, cyclomatic metrics, etc.). Each project has different number of metrics. For example, the project of CM1 has 38 metrics including defect metrics. The other hand, the project of JM1 has 22 metrics including defect metrics.

(3) AEEEM

The public AEEEM dataset consists of metrics data collected from five software systems by D'Ambros et al. [18]. They applied calculations of metrics to these five software systems and collect metrics data. Thus, they consist of various metrics data that are CK and OO metrics, number of previous defects, change metrics churn of CK and OO, and entropy of CK and OO. In our experiments, we use a Chidember-Kemerer (CK) and object-oriented (OO) metrics since NASA dataset and PROMISE dataset are complexity metrics datasets.

2.2.3 Metrics

To solve RQ1 and RQ2, we chose complexity metrics of the source code only as building a model since previous research work also has used complexity metrics of the source code[19]. The selected metrics are summarized in Table 2.2. In PROMISE dataset, we use all complexity

Table 2.1 Description of analyzed projects

Dataset	Project	# Entities	# Defective	(% Defective)	# Metrics	# Used Metrics
PROMISE	Ant v1.7	745	166	(22.3)	24	20
	Camel v1.6	965	188	(19.5)	24	20
	Ivy v1.4	241	16	(6.6)	24	20
	Jedit v4.0	306	75	(24.5)	24	20
	Log4j v1.0	135	34	(25.2)	24	20
	Lucene v2.4	340	203	(59.7)	24	20
	POI v3.0	442	281	(63.6)	24	20
	Tomcat v6.0	858	77	(9.0)	24	20
	Xalan v2.6	885	411	(46.4)	24	20
	Xerces v1.3	453	69	(15.2)	24	20
NASA	CM1	327	42	(12.8)	38	37
	JM1	7,782	1,672	(21.5)	22	21
	KC3	194	36	(18.6)	40	39
	MC1	1,988	46	(2.3)	39	38
	MC2	125	44	(35.2)	40	39
	MW1	253	27	(10.7)	38	37
	PC1	705	61	(8.7)	38	37
	PC2	745	16	(2.1)	37	36
	PC3	1,077	134	(12.4)	38	37
	PC4	1,287	177	(13.8)	38	37
	PC5	1,711	471	(27.5)	39	38
AEEEM	Eclipse JDT Core	997	206	(20.7)	291	17
	Equinox	324	129	(39.8)	291	17
	Apache Lucene	691	64	(9.3)	291	17
	Mylyn	1,862	245	(13.2)	291	17
	Eclipse PDE UI	1,497	209	(14.0)	291	17

Table 2.2 Metrics used

Dataset	Metrics	Reference
PROMISE, AEEEM	CK and OO	Basili et al.[14]
NASA	McCabe and Halsted	Shepperd et al.[50]

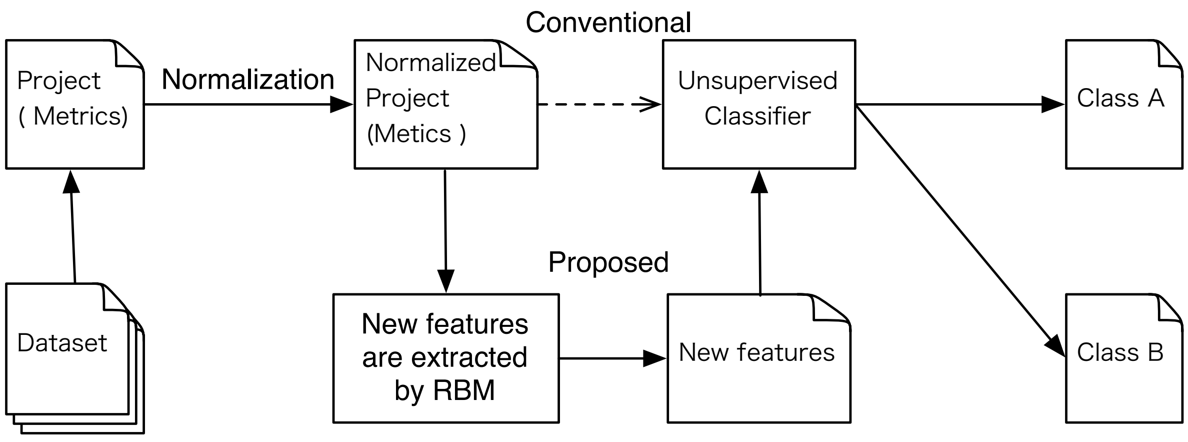


Figure 2.3 Overview of experiments

metrics for the analysis except a project name, a version name, a package name, and the number of bug in each project. For instance, Ant v1.7 has 24 complexity metrics including a project name, a version name, a package name, and the number of bug. The number of used metrics is 20 since we use all metrics except four metrics that are a project name, a version name, a package name, and the number of bug. In NASA dataset, we also use all metrics except the metrics of buggy or not buggy. In AEEEM dataset, we adopt the CK and OO metrics except a class name. Other metrics (i.e. number of previous defects, change metrics churn of CK and OO, and entropy of CK and OO) are not used since we use complexity metrics of source code in PROMISE and NASA datasets.

2.2.4 Preprocessing

The metrics used by RBM are normalized to a 0-1 scale since Bernoulli RBM of scikit-learn needs either binary inputs or 0-1 scale inputs.In 0-1 scale, we calculate as follow:

$$\boldsymbol{X} = \frac{\boldsymbol{X} - X_{\mathrm{min}}}{X_{\mathrm{max}} - X_{\mathrm{min}}} \tag{2.8}$$

where $\boldsymbol{X}$ is a vector of a feature, X_{min} is the smallest number in $\boldsymbol{X}$, X_{max} is the largest number in $\boldsymbol{X}$. The other hand, the metrics used by USVL and USVL with PCA are normalized to a mean 0 and variance 1 by z-score since previous research work has used a mean 0 and variance 1. The equation of z-score is as follow:

$$\boldsymbol{X} = \frac{\boldsymbol{X} - \mu}{\sigma} \tag{2.9}$$

where μ is a mean of a feature, σ is a variance of a feature.

Figure 2.4 shows the example of extracting features from 0-1 scale complexity metrics of source code by RBM. Normalized metrics to a 0-1 scale are applied to RBM that has ten hidden layers, and we get the new ten features. The new features extracted by RBM are applied to the USVL methods, and we refer them as USVL with RBM. The normalized metrics to z-score are applied to the USVL methods, and we refer them as USVL. Then, in both cases, the USVL methods divide original data to two classes. Since labeling task is not the main topic in our research, we label these classes using a correct data. Appropriate labeling tasks are left as a future work. Previous research works using USVL have used the qualitative information of original metrics data to docket the label[19, 26, 27, 28, 29, 30].

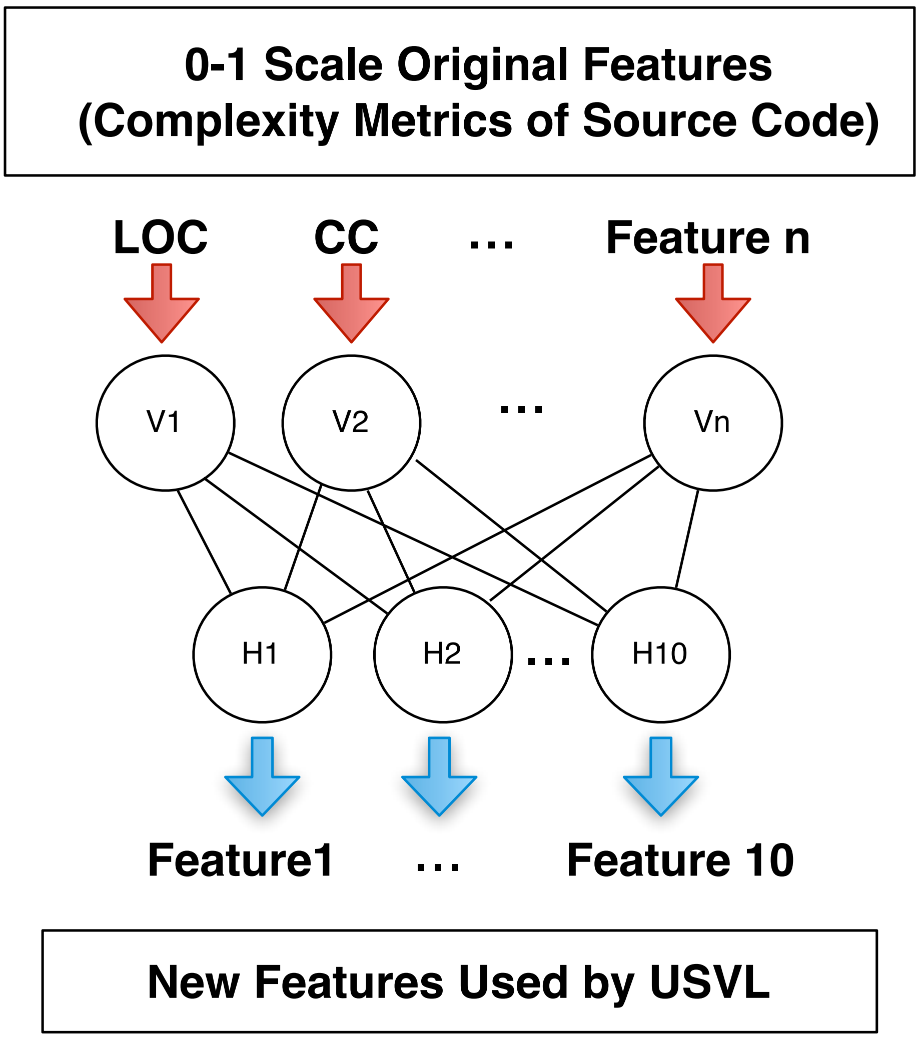

0-1 Scale Original Features
(Complexity Metrics of Source Code)
LOC
CC
...
Feature n
V1
V2
...
Vn
H1
H2
...
H10
Feature1
...
Feature 10
New Features Used by USVL

Figure 2.4 Example of preprocessing of RBM

2.2.5 Modeling Methods

We select five USVL methods: spectral clustering (SC), k-means (KM), partition around medoids (PAM), fuzzy C-means (FCM), and neural-gas (NG). These modeling methods have been used in previous research work[19].

In each modeling method, we use libraries of R and Python to calculate models except SC since we want reputability, and Zhang et al. creates function of SC in R[19]. R and Python are very simple and useful script to arrange data. Also, they have a lot of libraries to calculate and test models. Table 2.3 shows the library used by our experiments. We almost use default values of parameters except the number of clusters as two in USVL methods, the name of a method (e.g. neural-gas), and random state.

2.2.6 Evaluation Measures and Statistical Tests

(1) Evaluation Measures

We use the Area Under the receiver operating characteristic Curve (AUC) as the evaluation measure since AUC is not affected by the skewness of defect data[19]. In addition, we use an essential statistics of a variance as evaluate a variation between the performance among datasets and methods in each approach. High variance means that an approach has a variation between the performance, and we have to select a model, which is fitted a dataset since other models have low performances.

(2) Statistical Tests

We conduct two statistical test to compare performances of approaches (i.e. USVL with RBM, USVL, and USVL with PCA) that are the Scott-Knott test[51] and the Wilcoxon singed-rank test.

We adopt the Scott-Knott test using the 95% significance level, and have confirmed whether performances among methods have a variation between the performance or not. The Scott-Knott test procedure is a hierarchical clustering algorithm that finds out some groups. We describe the procedure of the Scott-Knott test as follows:

Table 2.3 Library used for experiments

Modeling Approach	Library
SC	Appendix[19]
KM	Package ’cclust’ CRAN (R)
PAM	Package ’cluster’ CRAN (R)
FCM	Package ’e1071’ CRAN (R)
NG	Package ’cclust’ CRAN (R)
RBM	scikit-learn neural_network Bernoulli-RBM

1. Evaluated methods (e.g. USVL with RBM and USVL methods) are arranged in the descending order of mean values.
2. Evaluated methods are separated into two groups based on the evaluation measure (e.g. AUC).
3. The statistics λ is calculated, and is tested by the chi square statistic.
4. If two groups have significant difference, the Scott-Knott test executes above procedures within each group, and the algorithm is stopped otherwise,

More details about the Scott-Knott test can be found on the article[51].

We adopt the Wilcoxon signed-rank test using the 95% significance level to compare the distribution of the approaches (i.e. USVL with RBM, USVL, and USVL with PCA) are significantly different. The Wilcoxon signed-rank test is a paired, non-parametric test. Thus, we do not need to suppose a normality and an equality of variance.

The Wilcoxon signed-rank test determines only whether two distribution are different or not, but not the effect size of the difference. Thus, we use the Cliff's delta[52] to measure the effect size like Lin et al.[53]. The threshold of effect size is proposed by Romano et al.[54] as follow:

$$\text{EffectSize} = \begin{cases} \text{negligible}(N), & \text{if } |d| \leq 0.147. \\ \text{small}(S), & \text{if } 0.147 < |d| \leq 0.33. \\ \text{medium}(M), & \text{if } 0.33 < |d| \leq 0.474. \\ \text{large}(L), & \text{if } 0.474 < |d| \leq 1. \end{cases} \tag{2.10}$$

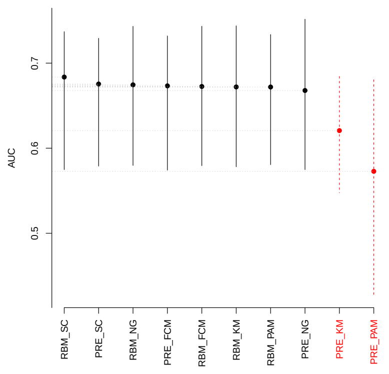
2.2.7 Overview of Experiment

Figure 2.3 shows the experiment overviews. We conduct ten times 10-fold cross validation (10-fold CV) to compare performances. Thus, the training data and test data are extracted from the original each project that is extracted from the original dataset (i.e. NASA dataset, PROMISE dataset and AEEEM dataset). The USVL uses only test data and do not use training data.

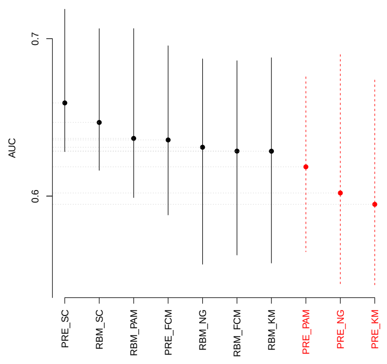
Table 2.4 All AUC values of USVL methods in 10 times 10-fold CV

		USVL with RBM						USVL					
Dataset	Project	SC	PAM	NG	FCM	KM	δ	SC	PAM	NG	FCM	KM	δ
PROMISE	ant v1.7	0.715	0.711	0.709	0.711	0.707	**0.008**	0.721	0.530	0.724	0.718	0.674	0.194
	camel v1.6	0.575	0.581	0.580	0.579	0.578	**0.006**	0.581	0.491	0.575	0.576	0.548	0.090
	ivy v1.4	0.671	0.630	0.658	0.661	0.652	**0.041**	0.683	0.569	0.654	0.683	0.603	0.114
	jedit v4.0	0.675	0.677	0.665	0.659	0.658	**0.019**	0.687	0.609	0.682	0.707	0.629	0.098
	log4j v1.0	0.737	0.711	0.726	0.721	0.721	**0.026**	0.725	0.647	0.722	0.732	0.647	0.085
	lucene v2.4	0.697	0.690	0.676	0.674	0.671	**0.026**	0.626	0.597	0.625	0.625	0.587	0.039
	poi v3.0	0.692	0.637	0.623	0.616	0.630	**0.076**	0.729	0.680	0.669	0.697	0.616	0.113
	tomcat v6.0	0.715	0.734	0.743	0.743	0.744	**0.029**	0.724	0.426	0.752	0.699	0.684	0.326
	xalan v2.6	0.632	0.631	0.627	0.626	0.626	**0.006**	0.579	0.570	0.596	0.574	0.587	0.026
	xerces v1.3	0.726	0.717	0.738	0.735	0.734	**0.021**	0.699	0.608	0.682	0.720	0.631	0.112
NASA	cm1	0.627	0.606	0.616	0.611	0.611	**0.021**	0.637	0.598	0.612	0.614	0.607	0.039
	jm1	0.616	0.609	0.557	0.563	0.557	**0.059**	0.628	0.619	0.543	0.588	0.544	0.085
	kc3	0.626	0.602	0.629	0.613	0.619	0.027	0.628	0.613	0.622	0.605	0.606	**0.023**
	mc1	0.669	0.654	0.653	0.654	0.649	**0.020**	0.674	0.565	0.592	0.651	0.588	0.109
	mc2	0.662	0.673	0.670	0.667	0.671	**0.011**	0.689	0.639	0.668	0.655	0.654	0.050
	mw1	0.649	0.653	0.657	0.650	0.647	**0.010**	0.668	0.676	0.690	0.682	0.674	0.022
	pc1	0.625	0.599	0.610	0.610	0.610	**0.026**	0.650	0.631	0.620	0.625	0.614	0.036
	pc2	0.706	0.706	0.687	0.686	0.688	**0.020**	0.719	0.667	0.617	0.696	0.603	0.116
	pc3	0.685	0.652	0.623	0.619	0.619	**0.066**	0.679	0.601	0.545	0.643	0.544	0.135
	pc4	0.631	0.629	0.620	0.622	0.622	**0.011**	0.647	0.624	0.567	0.630	0.566	0.081
	pc5	0.619	0.619	0.621	0.621	0.622	**0.003**	0.632	0.574	0.545	0.604	0.544	0.088
AEEEM	Eclipse JDT Core	0.700	0.697	0.673	0.668	0.668	**0.032**	0.700	0.558	0.644	0.685	0.600	0.142
	Equinox	0.712	0.724	0.718	0.711	0.715	**0.013**	0.702	0.643	0.657	0.661	0.625	0.077
	Apache Lucene	0.652	0.644	0.622	0.625	0.620	**0.032**	0.631	0.583	0.593	0.629	0.567	0.064
	Mylyn	0.627	0.640	0.627	0.628	0.628	**0.013**	0.640	0.605	0.586	0.634	0.568	0.072
	Eclipse PDE UI	0.625	0.630	0.598	0.603	0.600	**0.032**	0.655	0.627	0.604	0.629	0.587	0.068

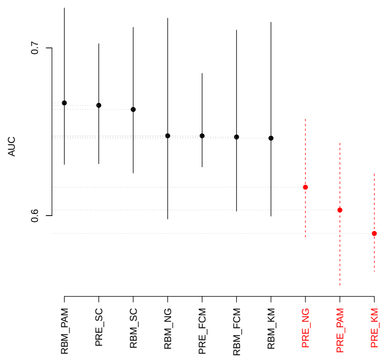
δ is the difference between maximum and minimum AUC values among five learning/learning methods in an approach. The smaller δ means that the approach can reduce the difference between methods. As for δ, a gray bold cell shows the smallest δ among two approaches, USVL with RBM and USVL.



(a) PROMISE



(b) NASA



(c) AEEEM

Figure 2.5 The max,min and average plots of AUC values of our USVL with RBM and USVL methods (RBM_SC, RBM_PAM, RBM_NG, RBM_FCM, RBM_KM, PRE_SC, PRE_PAM, PRE_NG, PRE_FCM, PRE_KM) in datasets by Scott-Knott Test. Different colors/lines show different classes by Scott-Knott Test.

Table 2.5 Statistic values: a p-value of Wilcoxon signed rank test and Cliff's delta d between USVL with RBM and USVL. Means, Medians, and Variances of USVL with RBM and USVL about AUCs.

Statistics	Values	
p-value	2.49e-06	
$\|d\|$	0.243	
	USVL with RBM	USVL
Means	0.654	0.630
Medians	0.650	0.628
Variance	0.00207	0.00304

Table 2.6 Statistic values: a p-value of Wilcoxon signed rank test and Cliff's delta d between δ.

Statistics	Values
p-value	5.96e-08
$\|d\|$	0.843

2.3 Results

2.3.1 RQ1:How well do RBM preprocessing affect the unsupervised defect prediction using source code complexity metrics?

(1) Motivation

The RBM is a popular preprocessing approach for learning technique in the machine learning. Thus, the RBM has already been applied to preprocessing of supervised learning, DNN in the defect prediction since the DNN needs preprocessing[5]. However, the RBM has not been applied yet to preprocessing of USVL methods and has not been reported in the defect prediction.

In general, the SVL requires the training data. However, less training data is available in new projects. This is a big problem to apply defect prediction approaches to industrial situations. If the accuracy of the USVL is as good as that of the SVL, the USVL can solve this problem since it does not require the training data. Zhang et al. tackle this problem, and have found that SC is as good as SVL[19].

However, there are significance differences among USVL methods. In addition, we do not know that SC can be the best performance in all other datasets. This motivates us to apply the USVL with RBM to the defect prediction expecting to improve the accuracy, and make all methods into flat by RBM.

(2) Approach

To compare the performances of predictors, we use two statistical tests that are the Scott-Knott test with the 0.05 significance level and the Wilcoxon signed-rank test with the 0.05 significance level. We use the Scott-Knott test as comparing the USVL with RBM vs. the USVL and a variation between the performance among methods for USVL with RBM and USVL approach, and use the Wilcoxon signed-rank test as comparing the accuracy of the USVL with RBM vs. the USVL. We use Cliff's delta as measuring an effect size for the USVL with RBM vs. the USVL. In addition, we use variances among each leaning approach. This statistic indicates that RBM has a small variation between the performance among methods for USVL with RBM and USVL approach.

(3) Results

Table 2.4 shows average AUC values calculated by USVL with RBM, and USVL in PROMISE, NASA, and AEEEM dataset. Each cell indicates average AUC values for ten times 10-fold CV in an approach. Also, δ shows the difference between maximum and minimum AUC values among five learning methods in an approach. The smaller δ means that the approach can have a small variation between the performance among methods. As for δ, a gray bold cell shows the smallest δ between two approaches, USVL with RBM, and USVL.

Table 2.4 indicates that the δs of USVL with RBM are almost a gray bold cell. Therefore, this result indicates that USVL with RBM has a small variation between the performance among methods. For instance, the project "ant v1.7", the highest difference of the AUC value between learning methods in proposed approach is 0.008 between SC and KM. On the other hand, in the conventional approach of USVL, the highest difference of value is 0.194 between NG and PAM. Thus, the δ of USVL with RBM is a gray bold cell.

Figure 2.5 shows the result by the Scott-Knott test applied to AUC values extracted from the PROMISE, NASA and AEEEM dataset, respectively. The figures indicate a variation between the performance (AUC) among USVL with RBM methods (with the prefix RBM_), and USVL methods (PRE_). Each learning has AUC values of max, minimum, and average. Also, the same color/line indicates, which they have nothing significant differences between AUC values of learning methods in the 0.05 significance level.

Figure 2.5 implies that the USVL with RBM can provide not only flatten accuracy but also improved accuracy of the defect prediction. Figure 2.5(a) shows that all learning methods belong to same groups except conventional KM and PAM. Figure 2.5(b) shows that the conventional SC has the best accuracy. The USVL with RBM methods belong to the same group. Figure 2.5(c) shows that SC with RBM has the best accuracy. Other USVL with RBM methods belong to the same group. Thus, the methods of USVL with RBM are as good as the best conventional USVL methods in PROMISE, NASA, and AEEEM dataset. In addition, the methods of USVL with RBM are belong to the same group of conventional SC. As conventional

SC is as good as SVL, the methods of USVL with RBM are also as good as SVL methods in PROMISE, NASA, and AEEEM dataset[19].

Table 2.5 shows the statistic values: a *p*-value of the Wilcoxon signed-rank test, Cliff's delta *d* between USVL with RBM and USVL, and means, medians, and variances of USVL with RBM and USVL about AUCs in Table 2.4.

Table 2.5 indicates that the accuracy of USVL with RBM approach is better than USVL approach. The *p*-value indicates that these approaches have significantly different, effect size is small since Cliff's delta *d* is 0.243, and mean and median AUC values of USVL with RBM approach are larger than USVL approach ones. On the other hand, the variance for USVL with RBM approach is smaller than USVL one. Thus, preprocessing using RBM has a small variation between the performance among datasets.

Table 2.6 indicates that the δ of USVL with RBM approach is better than USVL approach. This table shows the statistic values: a *p*-value of the Wilcoxon signed-rank test, and Cliff's delta *d* between δ of USVL with RBM and USVL in Table 2.4. In addition, δs of USVL with RBM are better than USVL since almost δs of USVL with RBM are the gray bold cell in table 2.4.

These results indicate that USVL with RBM approach have the same accuracy in PROMISE, NASA, and AEEEM dataset, and is better than USVL approach. The USVL approach have different accuracy. Thus, preprocessing using RBM makes the same accuracy of learning methods, and improve the accuracies of prediction.

(4) Discussion

In the experiment, we investigate how good the USVL with RBM as a preprocessing of a metrics be comparing to the USVL. The accuracy of detect prediction in the USVL with RBM approach is better than the USVL approach by the Wilcoxon signed-rank test and the Cliff's delta. The variation between the performance of prediction results among all USVL with RBM methods is smaller than the USVL methods and they become the same group in the Scott-Knott test at 0.05 significance level. Therefore, we can select an USVL approach without considering which USVL methods are fitted in a dataset since there are no significant difference among the

USVL methods. Also, we can solve the heterogeneity problem since we can select an USVL approach. Considering the results, a USVL with RBM as preprocessing of a metrics is an efficient method in a defect prediction.

Considering the results, by the USVL with RBM as preprocessing of metrics, we can have a small variation between the performance of accuracy among methods of learning. Furthermore, the USVL with RBM approach achieve better accuracy than USVL approach using the normalized metrics.

2.3.2 RQ2:Which is a better preprocessing between RBM and a traditional feature extraction approach?

(1) Motivation

In RQ1, we compare USVL based on normalized metrics vs. USVL based on extracting features from metrics by RBM, and get USVL with RBM is better than USVL. A feature extraction from original data is important process for USVL since USVL does not use training data. Thus, we have to compare above results of USVL with RBM vs. USVL with a traditional feature extraction approach. A lot of feature extraction approaches have been used in the defect prediction[55]. In this RQ, we apply one of the popular feature extraction approach of the PCA to USVL approach (USVL with PCA)[56].

(2) Approach

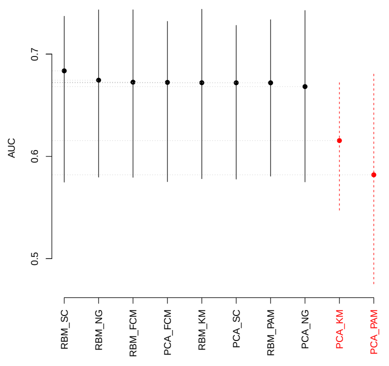
In PCA, we have to decide the number of principal components k. k means the number of new features. In this experiment, we choose k by a variance, which is retained after a feature extraction. We have retained 95% of variance after a feature extraction.

An approach is the same as RQ1. To compare the performances of predictors, we use two statistical tests that are the Scott-Knott test with the 0.05 significance level and the Wilcoxon signed-rank test with the 0.05 significance level. We use the Scott-Knott test as comparing the USVL with RBM vs. the USVL with PCA and a variation between the performance among

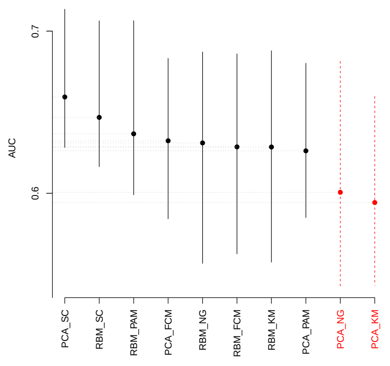
Table 2.7 All AUC values of USVL methods in 10 times 10-fold CV by PCA

		USVL with RBM						USVL with PCA					
Dataset	Project	SC	PAM	NG	FCM	KM	δ	SC	PAM	NG	FCM	KM	δ
PROMISE	ant v1.7	0.715	0.711	0.709	0.711	0.707	**0.008**	0.720	0.562	0.724	0.719	0.672	0.162
	camel v1.6	0.575	0.581	0.580	0.579	0.578	**0.006**	0.578	0.473	0.575	0.575	0.547	0.105
	ivy v1.4	0.671	0.630	0.658	0.661	0.652	**0.041**	0.666	0.582	0.672	0.681	0.561	0.120
	jedit v4.0	0.675	0.677	0.665	0.659	0.658	**0.019**	0.688	0.619	0.683	0.702	0.635	0.083
	log4j v1.0	0.737	0.711	0.726	0.721	0.721	**0.026**	0.724	0.638	0.727	0.732	0.667	0.094
	lucene v2.4	0.697	0.690	0.676	0.674	0.671	**0.026**	0.622	0.601	0.619	0.623	0.565	0.058
	poi v3.0	0.692	0.637	0.623	0.616	0.630	**0.076**	0.728	0.680	0.666	0.697	0.613	0.115
	tomcat v6.0	0.715	0.734	0.743	0.743	0.744	**0.029**	0.715	0.479	0.743	0.703	0.663	0.264
	xalan v2.6	0.632	0.631	0.627	0.626	0.626	**0.006**	0.580	0.569	0.592	0.575	0.577	0.023
	xerces v1.3	0.726	0.717	0.738	0.735	0.734	**0.021**	0.698	0.617	0.682	0.717	0.653	0.100
NASA	cm1	0.627	0.606	0.616	0.611	0.611	**0.021**	0.637	0.606	0.614	0.612	0.604	0.033
	jm1	0.616	0.609	0.557	0.563	0.557	**0.059**	0.628	0.618	0.542	0.584	0.545	0.086
	kc3	0.626	0.602	0.629	0.613	0.619	0.027	0.634	0.618	0.615	0.613	0.612	**0.022**
	mc1	0.669	0.654	0.653	0.654	0.649	**0.020**	0.674	0.591	0.593	0.649	0.583	0.091
	mc2	0.662	0.673	0.670	0.667	0.671	**0.011**	0.688	0.639	0.663	0.655	0.660	0.049
	mw1	0.649	0.653	0.657	0.650	0.647	**0.010**	0.669	0.680	0.681	0.683	0.658	0.025
	pc1	0.625	0.599	0.610	0.610	0.610	**0.026**	0.651	0.628	0.621	0.625	0.610	0.041
	pc2	0.706	0.706	0.687	0.686	0.688	**0.020**	0.714	0.664	0.620	0.680	0.611	0.103
	pc3	0.685	0.652	0.623	0.619	0.619	**0.066**	0.680	0.620	0.544	0.634	0.543	0.137
	pc4	0.631	0.629	0.620	0.622	0.622	**0.011**	0.647	0.639	0.569	0.628	0.567	0.080
	pc5	0.619	0.619	0.621	0.621	0.622	**0.003**	0.632	0.585	0.545	0.592	0.544	0.088
AEEEM	Eclipse JDT Core	0.700	0.697	0.673	0.668	0.668	**0.032**	0.698	0.580	0.643	0.688	0.616	0.118
	Equinox	0.712	0.724	0.718	0.711	0.715	**0.013**	0.705	0.644	0.659	0.659	0.625	0.080
	Apache Lucene	0.652	0.644	0.622	0.625	0.620	**0.032**	0.629	0.575	0.596	0.626	0.571	0.058
	Mylyn	0.627	0.640	0.627	0.628	0.628	**0.013**	0.640	0.604	0.587	0.634	0.578	0.062
	Eclipse PDE UI	0.625	0.630	0.598	0.603	0.600	**0.032**	0.653	0.630	0.603	0.627	0.591	0.062

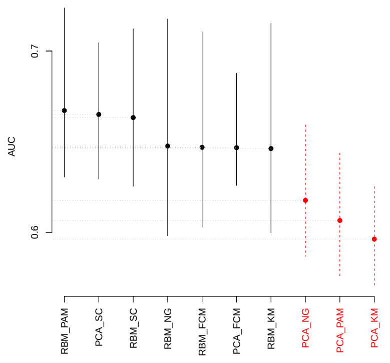
δ is the difference between maximum and minimum AUC values among five learning/learning methods in an approach. The smaller δ means that the approach can reduce the difference between methods. As for δ, a gray bold cell shows the smallest δ among two approaches, USVL with RBM and USVL with PCA.



(a) PROMISE



(b) NASA



(c) AEEEM

Figure 2.6 The max,min and average plots of AUC values of our USVL with RBM and USVL methods (RBM_SC, RBM_PAM, RBM_NG, RBM_FCM, RBM_KM, PRE_SC, PRE_PAM, PRE_NG, PRE_FCM, PRE_KM) in datasets by Scott-Knott Test. Different colors/lines show different classes by Scott-Knott Test.

Table 2.8 Statistic values: a p-value of Wilcoxon signed rank test and Cliff's delta d between USVL with RBM and USVL with PCA. Means, Medians, and Variances of USVL with RBM and USVL with PCA about AUCs.

Statistics	Values	
p-value	1.665e-06	
$\|d\|$	0.239	
	USVL with RBM	USVL with PCA
Means	0.654	0.631
Medians	0.650	0.628
Variance	0.00207	0.00280

Table 2.9 Statistic values: a p-value of Wilcoxon signed rank test and Cliff's delta d between δ.

Statistics	Values
p-value	5.96e-08
$\|d\|$	0.833

methods for USVL with RBM and USVL with PCA approach, and use the Wilcoxon signed-rank test as comparing the accuracy of the USVL with RBM vs. the USVL with PCA. We use Cliff's delta as measuring an effect size for the USVL with RBM vs. the USVL with PCA. In addition, we use variances among each leaning approach. This statistic indicates that RBM has a small variation between the performance among methods for USVL with RBM and USVL with PCA approach.

(3) Results

Table 2.7 indicates that the δs of USVL with RBM are almost a gray bold cell. This table shows average AUC values calculated by USVL with RBM, and USVL with PCA in PROMISE, NASA, and AEEEM dataset same as RQ1. Therefore, USVL with RBM approach has a small difference between the performance than USVL with PCA approach.

Figure 2.6 shows that USVL with PCA cannot provide flatten accuracy. However, figure 2.6(b) show that PAM with PCA improve group than PAM of USVL. These figures show the result by the Scott-Knott test applied to AUC values extracted from the PROMISE, NASA and AEEEM dataset, respectively. The prefix PCA_ indicates USVL with PCA.

Table 2.8 indicates that the accuracy of USVL with RBM approach is better than USVL approach. This table shows the statistic values: a *p* value of the Wilcoxon signed-rank test, Cliff's delta *d* between USVL with RBM and USVL with PCA, and means, medians, and variances of USVL with RBM and USVL with PCA about AUCs in Table 2.7. The *p*-value and Cliff's delta *d* indicate that there is a significance difference, and effect size is small. In addition, other statistics also indicate that USVL with RBM approach is better than USVL with PCA approach.

Table 2.9 indicates that the δ of USVL with RBM approach is better than USVL with PCA approach. This table shows the statistic values: a *p*-value of the Wilcoxon signed-rank test, and Cliff's delta *d* between δ of USVL with RBM and USVL with PCA in Table 2.7. The *p*-value and Cliff's delta *d* indicate that there is a significance difference, and effect size is large. In addition, δs of USVL with RBM are better than USVL with PCA since almost δs of USVL with RBM are the gray bold cell in table 2.7.

These results indicate that USVL with RBM approach is also better than USVL with PCA. Thus, preprocessing using RBM is better than PCA in this experimental setup.

(4) Discussion

In the experiment, we compare RBM feature extraction with PCA feature extraction. The results are similar to RQ1. USVL with RBM approach is better accuracy, and a variation between the performance of prediction results is smaller than USVL with PCA. Thus, RBM feature extraction is better than PCA in this experimental setup.

The reason why results are similar to RQ1 is that PCA focus on the dimensionality reduction. We have to choose *k* principal components in PCA from the viewpoint of how much variance of original features retained. Then, the new features generated by PCA has attribute close original features since PCA try to retain a variance of objective data. Thus, results of USVL with PCA is similar to USVL.

Considering the results, RBM preprocessing is small better accuracy, and has a small variation between the performance of prediction results than PCA preprocessing in defect prediction. Therefore, RBM is better than PCA in the experimental setup.

2.4 Threats To Validity

2.4.1 Construct validity

Learning methods are selected from the previous research works[19]. As the previous research work indicates, these learning methods are off-the-shelf and not state-of-the-art. We can say that the libraries are verified by many users. Since the dataset, PROMISE, NASA, and AEEEM, are used in many research works, we can say that these data are valid and verified ones.

In our experiments, we use information of correct data to add labels to two groups divided by USVL methods. We do not indicate an approach how we judge which cluster is a faulty cluster.

The USVL focuses on dividing data by extracting features from data, but does not focus on docketing labels, so this is a common issue in the USVL.

For example, in previous works using USVL methods[19, 26, 27, 28, 29, 30], the qualitative information of original metrics data is used to docket the label. The task of docketing labels is one of our future works.

2.4.2 External Validity

We apply our approach to three empirical datasets. Since the number of datasets are small, we conduct ten times 10-fold cross validation. This treatment makes our result generic.

2.4.3 Reliability

In this paper, we performed our experiments by existing libraries. The datasets used in the experiments are available publicly. These treatments makes our approach more reliable.

2.4.4 Open questions

Does it work that we use default values of parameters on existed libraries?: In our experiment, we use libraries to implement various modeling methods (e.g. scikit-learn). To guarantee reputability, we use default values of parameters on existed libraries except the number of the cluster in USVL, the name of the method (e.g. neural-gas), and random state. Adjusting parameters is an important task in machine learning since adjusted parameters improve the accuracy of machine learning approaches. Recently, researcher tackles adjusting parameters[57]. We validate this problem from the viewpoints of USVL with RBM as preprocessing vs. both using normalized metrics and metrics extracted by PCA. We describe USVL with RBM vs. both normalized metrics and metrics extracted by PCA. In this comparison, we can compare these methods on the equal condition that parameters of USVL are not adjusted. Thus, our setup of experiment should be fair from the above point.

2.5 Conclusion

Collecting a training dataset and selecting a prediction model for the defect prediction are important tasks in the software engineering. When we use a SVL as the defect prediction, we should collect a matched homogeneous metrics set with test data as a training data of a model. Using within-project data is a good way to collect such matched homogeneous metrics set. However, within-project data sometimes has a problem of less available data. For this reason, cross-project data often is used in defect prediction as a training data of a model. However, in the cross-project data, the accuracy of prediction may not consistent according tot the data used for training and testing.

In this paper, we proposed the USVL with RBM as preprocessing of metrics to solve the above problems. The USVL does not need a training dataset and we can have a small variation between the performance of predicting accuracy between the USVL methods by adopting RBM in the USVL.

In the experiments, three empirical datasets (i.e. PROMISE, NASA, and AEEEM) and various prediction models (e.g. scikit-learn) are used. The results of analysis showed that our USVL with RBM has a small variation between the prediction models and group learning methods to the highest accuracy group.

Contributions of this paper to MSR is as follows:

- **All USVL methods have same AUC values when we applied features extracted from source code metrics by RBM to USVL methods. This result is better than the previous USVL approach from the viewpoint of having a small variation between the performance of defect prediction approaches** Several previous research works showed that the accuracy of defect prediction models differs by the objective dataset, and selecting metrics. This paper indicates a way to solve the problem of selecting the model fitted the objective dataset, and selecting metrics in the defect prediction and show empirical results as the evidence.
- **These AUC values are small better than previous USVL methods** USVL with RBM improve the accuracy of USVL, which have low accuracy (e.g. *k*-means). These low accuracy methods can belong to high accuracy group using RBM. Thus, USVL with

RBM approach is small better accuracy than USVL approach.

- **Feature extraction approach of RBM is better than traditional feature extraction approach of PCA in this experimental setup** Our approach shows that the RBM is effective as a preprocessing technique of software metrics. In particular, RBM is a better feature extraction technique than the traditional feature extraction technique of PCA from the viewpoints of the accuracy and the variation between the performance among models in the defect prediction on this experimental setup.

Acknowledgment

I received a lot of help from many individuals during this work. Especially, I would like to thank Professor Osamu Mizuno. I would not have completed this study without his continuous support and encouragement. I would also like to express great thanks to Associate Professor Yasutaka Kamei, at Kyushu University, Dr. Cor-Paul Bezemer and Professor Ahmed E. Hassan, at Queen's University for their valuable and helpful suggestions about this study. I am deeply grateful to Mr. Keita Mori and Mr. Tomoharu Kitawaki, who discuss with us about this study. I am also very grateful to the members of the Software Engineering Laboratory in Kyoto Institute of Technology. I also thank to my friends and my family.

References

[1] G. Tassey, “The economic impacts of inadequate infrastructure for software testing,” National Institute of Standards and Technology, RTI Project, vol.7007, no.011, pp.••–••, 2002.

[2] C. Catal and B. Diri, “A systematic review of software fault prediction studies,” Expert systems with applications, vol.36, no.4, pp.7346–7354, 2009.

[3] S. Wang, T. Liu, and L. Tan, “Automatically learning semantic features for defect prediction,” Proceedings of the 38th International Conference on Software Engineering, pp.297–308, ACM, 2016.

[4] X.-Y. Jing, S. Ying, Z.-W. Zhang, S.-S. Wu, and J. Liu, “Dictionary learning based software defect prediction,” Proceedings of the 36th International Conference on Software Engineering, pp.414–423, ACM, 2014.

[5] X. Yang, D. Lo, X. Xia, Y. Zhang, and J. Sun, “Deep learning for just-in-time defect prediction,” Software Quality, Reliability and Security (QRS), 2015 IEEE International Conference on, pp.17–26, IEEE, 2015.

[6] K. Gao and T.M. Khoshgoftaar, “A comprehensive empirical study of count models for software fault prediction,” IEEE Transactions on Reliability, vol.56, no.2, pp.223–236, 2007.

[7] Y. Ma, L. Guo, and B. Cukic, “A statistical framework for the prediction of fault-proneness,” Advances in Machine Learning Application in Software Engineering, Idea Group Inc, pp.237–265, 2006.

[8] V.U.B. Challagulla, F.B. Bastani, I.-L. Yen, and R.A. Paul, “Empirical assessment of machine learning based software defect prediction techniques,” International Journal on Artificial Intelligence Tools, vol.17, no.02, pp.389–400, 2008.

[9] A.G. Koru and H. Liu, “An investigation of the effect of module size on defect prediction using static measures,” ACM SIGSOFT Software Engineering Notes, vol.30, no.4, pp.1–5, 2005.

[10] L. Guo, B. Cukic, and H. Singh, “Predicting fault prone modules by the dempster-shafer belief networks,” Automated Software Engineering, 2003. Proceedings. 18th IEEE International Conference on, pp.249–252, IEEE, 2003.

[11] T.M. Khoshgoftaar and N. Seliya, “Tree-based software quality estimation models for fault prediction,” Software Metrics, 2002. Proceedings. Eighth IEEE Symposium on, pp.203–214, IEEE, 2002.

[12] N.F. Schneidewind, “Investigation of logistic regression as a discriminant of software quality,” Software Metrics Symposium, 2001. METRICS 2001. Proceedings. Seventh International, pp.328–337, IEEE, 2001.

[13] T.M. Khoshgoftaar, K. Gao, and R.M. Szabo, “An application of zero-inflated poisson regression for software fault prediction,” Software Reliability Engineering, 2001. ISSRE 2001. Proceedings. 12th International Symposium on, pp.66–73, IEEE, 2001.

[14] V.R. Basili, L.C. Briand, and W.L. Melo, “A validation of object-oriented design metrics as quality indicators,” IEEE Transactions on software engineering, vol.22, no.10, pp.751–761, 1996.

[15] S. Kim, T. Zimmermann, E.J. Whitehead Jr, and A. Zeller, “Predicting faults from cached history,” Proceedings of the 29th international conference on Software Engineering, pp.489–498, IEEE Computer Society, 2007.

[16] R. Moser, W. Pedrycz, and G. Succi, “A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction,” 2008 ACM/IEEE 30th International Conference on Software Engineering, pp.181–190, IEEE, 2008.

[17] A.E. Hassan, “Predicting faults using the complexity of code changes,” Proceedings of the 31st International Conference on Software Engineering, pp.78–88, IEEE Computer Society, 2009.

[18] M. D’Ambros, M. Lanza, and R. Robbes, “An extensive comparison of bug prediction approaches,” 2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010), pp.31–41, IEEE, 2010.

[19] F. Zhang, Q. Zheng, Y. Zou, and A.E. Hassan, “Cross-project defect prediction using a

connectivity-based unsupervised classifier," Proceedings of the 38th International Conference on Software Engineering, pp.309–320, ACM, 2016.

[20] J. Nam and S. Kim, "Heterogeneous defect prediction," Proceedings of the 2015 10th joint meeting on foundations of software engineering, pp.508–519, ACM, 2015.

[21] J. Nam, S.J. Pan, and S. Kim, "Transfer defect learning," Proceedings of the 2013 International Conference on Software Engineering, pp.382–391, IEEE Press, 2013.

[22] F. Zhang, A. Mockus, I. Keivanloo, and Y. Zou, "Towards building a universal defect prediction model," Proceedings of the 11th Working Conference on Mining Software Repositories, pp.182–191, ACM, 2014.

[23] Y. Kamei, T. Fukushima, S. McIntosh, K. Yamashita, N. Ubayashi, and A.E. Hassan, "Studying just-in-time defect prediction using cross-project models," Empirical Software Engineering, vol.21, no.5, pp.2072–2106, 2016.

[24] B. Turhan, T. Menzies, A.B. Bener, and J. Di Stefano, "On the relative value of cross-company and within-company data for defect prediction," Empirical Software Engineering, vol.14, no.5, pp.540–578, 2009.

[25] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, and B. Murphy, "Cross-project defect prediction: a large scale experiment on data vs. domain vs. process," Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering, pp.91–100, ACM, 2009.

[26] J. Nam and S. Kim, "Clami: Defect prediction on unlabeled datasets (t)," Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on, pp.452–463, IEEE, 2015.

[27] G. Abaei, Z. Rezaei, and A. Selamat, "Fault prediction by utilizing self-organizing map and threshold," Control System, Computing and Engineering (ICCSCE), 2013 IEEE International Conference on, pp.465–470, IEEE, 2013.

[28] P.S. Bishnu and V. Bhattacherjee, "Software fault prediction using quad tree-based k-means clustering algorithm," IEEE Transactions on knowledge and data engineering, vol.24, no.6, pp.1146–1150, 2012.

[29] B. Yang, Q. Yin, S. Xu, and P. Guo, “Software quality prediction using affinity propagation algorithm,” 2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence), pp.1891–1896, IEEE, 2008.

[30] S. Zhong, T.M. Khoshgoftaar, and N. Seliya, “Unsupervised learning for expert-based software quality estimation.,” HASE, pp.149–155, Citeseer, 2004.

[31] B. Ghotra, S. McIntosh, and A.E. Hassan, “Revisiting the impact of classification techniques on the performance of defect prediction models,” Proceedings of the 37th International Conference on Software Engineering-Volume 1, pp.789–800, IEEE Press, 2015.

[32] A.R. Gray and S.G. Macdonell, “Software metrics data analysis—exploring the relative performance of some commonly used modeling techniques,” Empirical Software Engineering, vol.4, no.4, pp.297–316, 1999.

[33] G.E. Hinton, S. Osindero, and Y.-W. Teh, “A fast learning algorithm for deep belief nets,” Neural computation, vol.18, no.7, pp.1527–1554, 2006.

[34] I. Arora, V. Tetarwal, and A. Saha, “Open issues in software defect prediction,” Procedia Computer Science, vol.46, pp.906–912, 2015.

[35] J. Nam, “Survey on software defect prediction,” HKUST PhD Qualifying Examination, Department of Compter Science and Engineerning, The Hong Kong University of Science and Technology, Tech. Rep, pp.●●–●●, 2014.

[36] S. Herbold, “Training data selection for cross-project defect prediction,” Proceedings of the 9th International Conference on Predictive Models in Software Engineering, p.6, ACM, 2013.

[37] Z. He, F. Shu, Y. Yang, M. Li, and Q. Wang, “An investigation on the feasibility of cross-project defect prediction,” Automated Software Engineering, vol.19, no.2, pp.167–199, 2012.

[38] A. Mahaweerawat, P. Sophatsathit, C. Lursinsap, and P. Musilek, “Fault prediction in object-oriented software using neural network techniques,” Advanced Virtual and Intelligent Computing Center (AVIC), Department of Mathematics, Faculty of Science, Chulalongkorn University, Bangkok, Thailand, pp.1–8, 2004.

[39] Q. Wang, B. Yu, and J. Zhu, "Extract rules from software quality prediction model based on neural network," Tools with Artificial Intelligence, 2004. ICTAI 2004. 16th IEEE International Conference on, pp.191–195, IEEE, 2004.

[40] S. Kanmani, V.R. Uthariaraj, V. Sankaranarayanan, and P. Thambidurai, "Object oriented software quality prediction using general regression neural networks," ACM SIGSOFT Software Engineering Notes, vol.29, no.5, pp.1–6, 2004.

[41] M.M.T. Thwin and T.-S. Quah, "Application of neural networks for software quality prediction using object-oriented metrics," Journal of systems and software, vol.76, no.2, pp.147–156, 2005.

[42] N.J. Pizzi, R. Summers, and W. Pedrycz, "Software quality prediction using median-adjusted class labels," Proceedings: International Joint Conference on Neural Networks, vol.3, pp.2405–2409, 2002.

[43] T.M. Khoshgoftaar, E.B. Allen, J.P. Hudepohl, and S.J. Aud, "Application of neural networks to software quality modeling of a very large telecommunications system," IEEE Transactions on neural networks, vol.8, no.4, pp.902–909, 1997.

[44] G. Cybenko, "Approximation by superpositions of a sigmoidal function," Mathematics of control, signals and systems, vol.2, no.4, pp.303–314, 1989.

[45] D.H. Ackley, G.E. Hinton, and T.J. Sejnowski, "A learning algorithm for boltzmann machines," Cognitive science, vol.9, no.1, pp.147–169, 1985.

[46] G. Hinton, "Boltzmann machines," Encyclopedia of Machine Learning, pp.132–136, Springer, 2011.

[47] P. Smolensky, "Information processing in dynamical systems: Foundations of harmony theory," Technical report, DTIC Document, 1986.

[48] H. Larochelle and Y. Bengio, "Classification using discriminative restricted boltzmann machines," Proceedings of the 25th international conference on Machine learning, pp.536–543, ACM, 2008.

[49] M. Jureczko and L. Madeyski, "Towards identifying software project clusters with regard

to defect prediction," Proceedings of the 6th International Conference on Predictive Models in Software Engineering, p.9, ACM, 2010.

[50] M. Shepperd, Q. Song, Z. Sun, and C. Mair, "Data quality: Some comments on the nasa software defect datasets," IEEE Transactions on Software Engineering, vol.39, no.9, pp.1208–1215, 2013.

[51] E.G. Jelihovschi, J.C. Faria, and I.B. Allaman, "Scottknott: a package for performing the scott-knott clustering algorithm in r," TEMA (São Carlos), vol.15, no.1, pp.3–17, 2014.

[52] N. Cliff, "Dominance statistics: Ordinal analyses to answer ordinal questions.," Psychological Bulletin, vol.114, no.3, p.494, 1993.

[53] D. Lin, C.-P. Bezemer, and A.E. Hassan, "Studying the urgent updates of popular games on the steam platform," Empirical Software Engineering, pp.1–32, ●●.

[54] J. Romano, J.D. Kromrey, J. Coraggio, J. Skowronek, and L. Devine, "Exploring methods for evaluating group differences on the nsse and other surveys: Are the t-test and cohen ' sd indices the most appropriate choices," annual meeting of the Southern Association for Institutional Research, pp.●●–●●, 2006.

[55] Z. Xu, J. Liu, Z. Yang, G. An, and X. Jia, "The impact of feature selection on defect prediction performance: An empirical comparison," Software Reliability Engineering (ISSRE), 2016 IEEE 27th International Symposium on, pp.309–320, IEEE, 2016.

[56] R. Bro and A.K. Smilde, "Principal component analysis," Analytical Methods, vol.6, no.9, pp.2812–2831, 2014.

[57] C. Tantithamthavorn, S. McIntosh, A.E. Hassan, and K. Matsumoto, "Automated parameter optimization of classification techniques for defect prediction models," Proceedings of the 38th International Conference on Software Engineering, pp.321–332, ICSE '16, ACM, New York, NY, USA, 2016.