A Machine Learning-based Impact Analysis Tool and its Improvement Using Co-occurrence Relationships

Teppei Kawabata
Shibaura Institute of Technology
Tokyo, Japan
ma22045@shibaura-it.ac.jp

Tsuyoshi Nakajima
Shibaura Institute of Technology
Tokyo, Japan
tsnaka@shibaura-it.ac.jp

Shuichi Tokumoto
Information Technology R&D Center, Mitsubishi Electric Corporation
Kanagawa, Japan
Tokumoto.Shuichi@dr.mitsubishielectric.co.jp

Ryota Tsukamoto
Information Technology R&D Center, Mitsubishi Electric Corporation
Kanagawa, Japan
Tsukamoto.Ryota@dy.mitsubishielectric.co.jp

Kazuko Takahashi
Information Technology R&D Center, Mitsubishi Electric Corporation Kanagawa, Japan
Takahashi.Kazuko@dx.mitsubishielectric.co.jp

Abstract— **In the development of diverted software, impact analysis, which determines the extent of software impact on change requests, is an important task because it greatly affects the quality and efficiency of the development. We proposed a method that machine-learns the modification histories of the projects using word-embedding techniques and multi-label classifiers to accurately generate a ranking list of modification candidates of the software components in order of their sigmoid values. To improve accuracy of the method, this paper proposes to use the multi-label classifier algorism to take co-occurrence between labels into account because of the assumption of the dependencies between the components. Experiments were conducted on actual project data to compare the accuracy of the four algorisms: Convolutional neural networks, BR method, LP method, and RAkEL method. The result shows that RAkEL method, which takes co-occurrence relationships into account and does not over-learn, has the best accuracy among them.**

Keywords—impact analysis; change requests; machine learning; co-occurrence relationships;

I. Introduction

Software impact analysis is the task of determining the extent to which a change request affects when implemented [1]. Its failure may result in incomplete implementation of change requests or degrades on existing functionalities. Therefore, when many small projects that put small changes on a large source code base concurrently and continuously run, the accuracy and efficiency of impact analysis is crucially important for their development productivity and quality [2].

When conducting an impact analysis to identify the components to be fixed in a source code base for a change request (modification targets), the following two tasks are required: selecting modification candidates and determining modification targets from them. Since the latter task can only be performed by the developers by reviewing the modification candidates, it is important how well the former task can be performed with complete coverage and without waste in order to increase the accuracy and efficiency of the impact analysis.

Requirements traceability is commonly used to seek for modification candidates. It is a discernible association between a requirement and its relating requirements, generally difficult to establish and maintain traceability continuously with a high degree of accuracy. Moreover, it is required to correctly identify the existing requirements affected by the change request before applying the traceability.

Iwasaki et al. [4] proposed and implemented a method for estimating a list of components of a source program as modification candidates directly from a change request by machine-learning the history of changes for the change requests. The implemented tool has two components, one is word embedding part which translates a change request text into a vector form, and the other is machine-learning part which estimates modification candidates from the change request vector. In paper [4], the machine-learning part was implemented using a convolutional neural network (CNN) and evaluated using real project data. The results showed that the method works effectively when there exist many projects each of which implements a small set of change requests for the same source code base.

The tool provides the ranking list of components most likely to be modified (modification candidates) in descendant order of sigmoid value, however it does not provide how to determine the range of modification candidates from which the reviewer determines the modification targets. To determine the range of modification candidates, we set the threshold from the actual data so that it can narrow the range to around 30%. As a result, the rate of missing modification targets for the candidate range was around 23% in case of the above implementation.

In this paper, to improve the accuracy of the tool, we propose to change machine-learning part of the tool from CNN to the other multi- label classification algorithms that take into account label correlations: LP (Label Powerset) and RAkEL (RAndom k-labELsets) [12]. In addition, for comparison of performance, CNN, and BR (Binary Relevance) are used in the experiment. As a result, he RAkEL shows the best results among them.

Section 2 describes the proposed method and its implementation presented in paper [4], Section 3 introduces the algorithms for multi-label classification, and Section 4 describes the experiment to compare the four implementations and show its evaluation results.

II. PROPOSED METHOD AND ITS CNN IMPLEMENTATION

This section describes the experimental data treated in this study and the algorithmic structure of previous studies.

A. Characteristics of the Target Project and its Deliverables

In the software development for diverse and continuously evolving products (multi-product, small-change development [5]), a large number of changes have been made to a source code base to periodically add new features, customizing it for different sets of hardware and various shipping destinations. This type of development often causes many small projects with multiple change requests running in parallel without sufficient human resource with sufficient knowledge on the source code infrastructure to perform impact analysis accurately and efficiently.

The target project group adopts a derivative development method called XDDP (eXtreme Derivative Development Process) [6], in which one change design document is supposed to be created for each change request.

This document describes the following items.

- Change Request ID
- Requirements (natural language)
- Mounting
- Details of changes to software method design specifications
- Modification details regarding the module of the software detailed design specification
- Description of changes made to the source code, including names of components and modules that have been modified

About 30 projects occur every year, and about 10 change requests are made per project in average.

The input change request text is written in Japanese, having 20 to 400 characters, and the output is 32 components, which the source code based has.

B. Proposed Method

The proposed method learns a large number of change design documents to estimate modification candidates directly from new change requests [4].

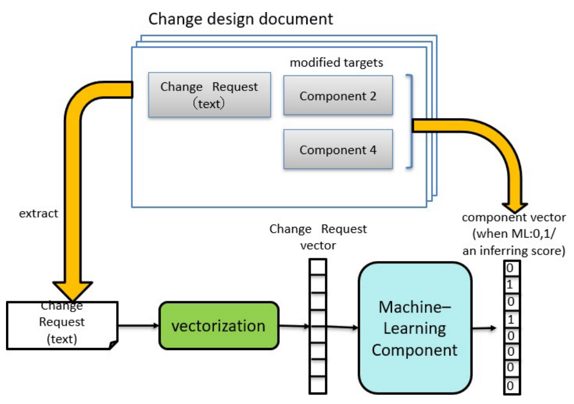
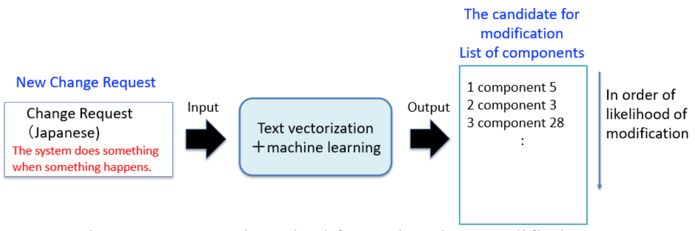


Figure 1 Configuration of the implemented tool.

As shown in Figure 1, for each change request document, a change request text is extracted, and then a vector of change request text is created using a word-embedding technology. On the other hand, a component vector is created from the information on the modified modules in the source code base corresponding to the change request. Each index of the vector is uniquely corresponding to some component in the source code base, whose value is either 0 or 1 (1 means modified and 0 means unmodified).

At the time of estimation, the proposed method outputs the ranking list of components most likely to be modified for a new change request text.

Compared to the impact analysis using traceability, the proposed method has the advantage of being able to select modification candidates directly from a new change request without burdening development activities.


The candidate for modification
List of components
New Change Request
Change Request (Japanese)
The system does something when something happens.
Input
Text vectorization ＋machine learning
Output
1 component 5
2 component 3
3 component 28
:
In order of likelihood of modification
Figure 2 Proposed Method for Estimating Modified Candidates.

C. Algorithm Structure of Previous Studies

1) Text vectorization

When vectorizing a text, the text is decomposed into words by applying the morphological analyzer mecab [7]. The decomposed words are processed in three stages: extraction of words to be used from all the words (word extraction), vectorization of the words, and vector integration. The resulting vector has 100 dimensions.

In a previous study, we tried three implementation methods shown in Table 1 and as a result found out that noun selection + doc2vec [8] (Implementation 3) produced the best results.

TABLE I: IMPLEMENTATIONS EVALUATED IN PREVIOUS STUDY

Implementation No	Word extraction	Word vectorization	Vector integration
I 1	Noun selection	word2vec (skip-gram)	Simple averaging
I 2	Full selection	doc2vec	
I 3	Noun selection	doc2vec	

2) Machine learning

The machine-learning part can be seen as the multi-label classifier since it determines whether the 32 vectors of values are 0s or 1s for a vector of change request text. To implement the tool, convolutional neural networks (CNNs) have been used as a multi-label classifier.

Figure 3 shows the structure of the implemented CNN. The input is a 100-dimensional vector of change request text, and the output is a 32-dimensional vector of component lists. The reason the number of components output is 32 is that the number of components in the data used in the experiment is fixed at 32.

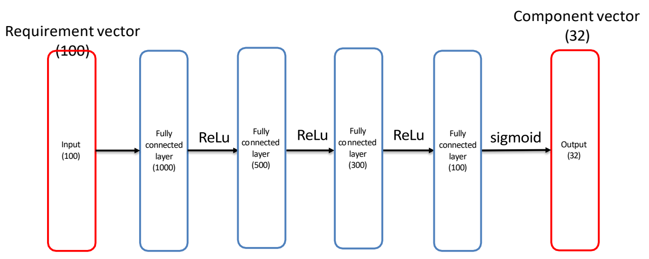

Requirement vector (100)
Component vector (32)
Input (100)
Fully connected layer (1000)
ReLu
Fully connected layer (500)
ReLu
Fully connected layer (300)
ReLu
Fully connected layer (100)
sigmoid
Output (32)

Figure 3 Adopted structure of CNN.

The parameters of the CNN are as follows.

- Intermediate layer: 4 (1000, 500, 300, 100)
- Number of epochs: 50
- Batch size: 50
- Learning rate: 0.1
- Function on output: sigmoid

D. Reassessment of Previous Studies and Issues

In the previous study [4], a ranking list of modification candidates is ordered by the likelihood to be modified to a change request. The output list contains a mixture of modification targets and the others. The modification targets are the components that need to be modified by the change request.

The previous study used two metrics to evaluate its performance: coverage range ratio and accuracy in the coverage range, where the coverage range is up to the position where the last modification target appears in the list.

However, prior research has not provided a method for determining which components need to be reviewed. To do this, we devise a method that determines a threshold on the sigmoid value to determine the range to be reviewed, where the threshold is to be determined from the actual data to be a specified range ratio. In addition, we defined three metrics shown in Figure 4 to evaluate the performance of the method.

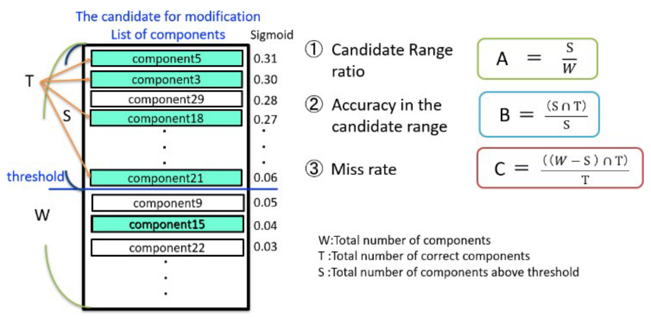


Figure 4 Measurements to evaluate effect on impact Analysis.

1. Candidate range ratio: percentage of components with higher sigmoid value than the threshold out of the number of all components.
2. Accuracy in the candidate range: percentage of modification targets out of components in the candidate range.
3. Missing rate: percentage of the modification targets beyond the candidate range out of all the modification targets.

Table 2 shows the metrics measured for the data of the previous study.

TABLE II THREE METRICS FOR THE PREVIOUS STUDY

Threshold	A	B	C
0.06	30.0%	18.0%	23.0%

In the Table 2, the threshold is 0.06 when A is set to 30%. In case, the tool of the previous study resulted in B of 35% and C of 23%. The problem is that C is considerably high. A higher missing rate may cause bugs in the program because it increases the likelihood of leakage of reviewing. Therefore, it is necessary to reduce the missing rate for practical use.

E. Improvement Targets

The goal of this study is that the candidate range ratio is less than 30% and the missing rate is 5% or less. The reason we set the goal is that we want to keep the missing rate within the 2σ interval from the viewpoint of quality assurance, obtaining a certain level of effort reduction of reviewing tasks.

III. Algorithms for Multi-Label Classification and Co-Occurrence Relationships

This section explains the reasons for focusing on co-occurrence relationships and the methods that take co-occurrence relationships into account.

A. Co-occurrence relationships in the source code base

To improve the missing rate in the previous study, we focus on the architectural dependencies between the components in the source code base. Such architectural dependencies include:

- Call Relationships
- Resource sharing relationships (communication, memory, I/O)
- File read/write relationships
- Inheritance relationships
- Include relationships

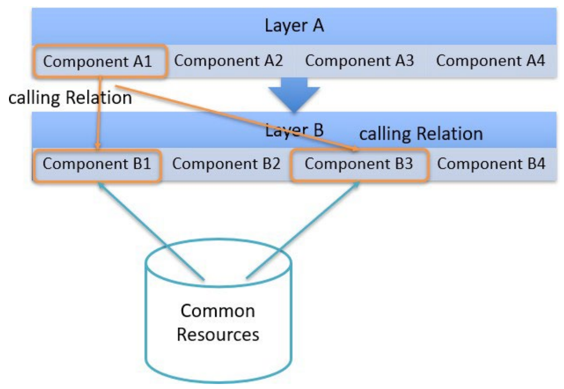


Figure 5 Dependencies between components.

In particular, the target source code base has a layer structure as illustrated in Figure 5, having:

- Calling relationships may occur between the components of adjacent layers.

- Components in a specific layer (Layer B in Figure 5) handle a common resource, causing indirect dependencies.

Concerning this observation, we hypothesize that components having dependencies are often modified together. If it is true, we can improve the machine-learning part by applying multi-label classification algorithms that take into account label correlations to it, which may increase its performance.

B. Algorithms for handling multi-label classification

As mentioned earlier, machine-learning in this study is attributed to the problem of multi-label classification, in which multiple labels are assigned to a single object [7].

The major difference between multi-label classification and single-label classification is that it is expected to improve accuracy by using the co-occurrence relationships between labels in the prediction process.

In this study, in order to incorporate co-occurrence relationships among outputs, several multi-label classifiers are examined, with a combination with the Support Vector Machine (SVM). SVM is a supervised learning algorithm [8] that can be used for classification and regression problems such as natural language processing and speech recognition.

The Binary Relevance (BR) method is one of the representative methods for multi-label classification (without considering correlations between labels), predicting labels by transforming a multi-label classification into multiple single-label classification. In detail, the BR method creates a binary classifier for each label and outputs the sum of the classifier results [9]. For our problem, each element of the component vector is trained with the input sentences vector (Figure 6).

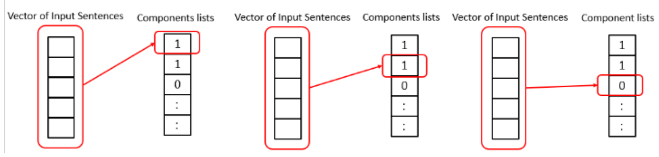

Vector of Input Sentences
Components lists
1
1
0
:
:
Vector of Input Sentences
Components lists
1
1
0
:
:
Vector of Input Sentences
Component lists
1
1
0
:
:

Figure 6 Configuration of the learning algorithm for the BR method.

C. Algorithm to model co-occurrence relationships

1) Label Powersets method (LP method)

Label Powersets method (LP) is one of the basic algorisms for multi-label classification considering the correlation between labels.

The LP method treats each element of the power set of labels as a class, transforming multi-label classification into multi-class classification. A power set is all possible Combinations, for example, a power set of labels 1, 2, and 3 are ϕ, $\{1\}$, $\{2\}$, $\{3\}$, $\{1, 2\}$, $\{1, 3\}$, $\{2, 3\}$, $\{1, 2, 3\}$. The LP method use all the set except ϕ as classes, classifying an input into one class.Figure 7 illustrates what pattens are to be learned.

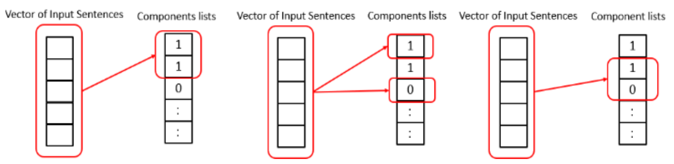
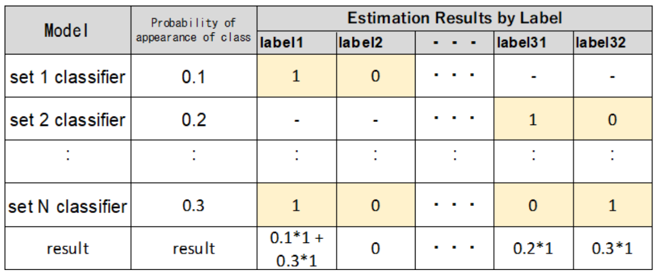

Vector of Input Sentences
Components lists
1
1
0
:
:
Vector of Input Sentences
Components lists
1
1
0
:
:
Vector of Input Sentences
Component lists
1
1
0
:
:

Figure 7 Configuration of the learning algorithm for the LP method.

The LP method calculates the probability of occurrence of a label from the sum of the probability of occurrence of the classes in which the label appears, shown in Figure 8.



Model	Probability of appearance of class	Estimation Results by Label				
		label1	label2	・・・	label31	label32
set 1 classifier	0.1	1	0	・・・	-	-
set 2 classifier	0.2	-	-	・・・	1	0
:	:	:	:	:	:	:
set N classifier	0.3	1	0	・・・	0	1
result	result	0.1*1 + 0.3*1	0	・・・	0.2*1	0.3*1

Figure 8 Estimation results for each label.

Although the LP method has the advantage of prediction based on co-occurrence relationships among labels, it has some disadvantages:

- The computational complexity increases exponentially with the number of labels.
- The number of classes increases, resulting in overlearning when the number of data is small.

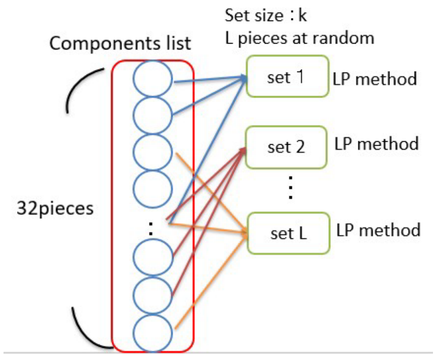
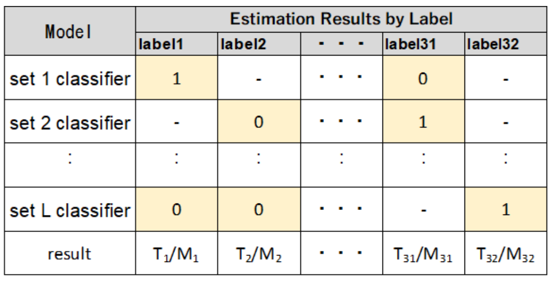


Figure 9 Creating a subset from a set of labels.

2) Random k Labelsets method (RAkEL method)

As mentioned above, the LP method has some disadvantages when the number of labels increases. The RAkEL method [10] was proposed to conquer them.



Model	Estimation Results by Label				
	label1	label2	・・・	label31	label32
set 1 classifier	1	-	・・・	0	-
set 2 classifier	-	0	・・・	1	-
:	:	:	:	:	:
set L classifier	0	0	・・・	-	1
result	T_1/M_1	T_2/M_2	・・・	T_{31}/M_{31}	T_{32}/M_{32}

T1: Number of cells whose estimated result is 1， Mi: Number of cells with estimated resul

Figure 10 Estimation results by label.

The RAkEL method first randomly creates label subsets of size *k* for the input label set, and then applies the LP method to each subset, as illustrated in Figure 9.

The classification results for each subset are then integrated to predict the label, as shown in Figure 10.

The RAkEL method is an algorithm with high potential for improving accuracy compared to the LP method because it can significantly reduce the computational complexity of LP calculations for subsets compared to the LP method for the whole set, and it can also reduce the bias in the distribution of each class value.

IV. EVALUATION AND EXPERIMENT

This section describes the experimental results and evaluation of the proposed method.

A. *Purpose of the experiment*

We consider that the applicability and accuracy of the algorithms described in previous section will differ depending on the nature of the problem domain and the number of available training data. To examine them, we apply the BR, LP, and RAkEL methods to the machine learning part of the proposed method and conduct an experiment to compare their accuracy using the same project data.

Our research question is whether the algorithms that take co-occurrence relationships into account may improve the accuracy for this problem or not. To answer this question, we select the following four implementation of the machine-learning part for comparison:

1. CNN (implementation in [4], already shown in Table 2)
2. BR with SVM (no consideration on co-occurrence)
3. LP with SVM
4. RAkEL with SVM

This will allow us to evaluate whether algorisms considering co-occurrence relationships improve accuracy, investigating effects of the LP method's disadvantages. Furthermore, by presenting the measurement results of the CNN-based classifier, we will evaluate tow what extent they improve the accuracy from the previous study.

B. *Experimental data*

This experiment uses data from 405 change design documents provided. The data was divided into training and test data at a ratio of 4:1, with 324 books used as training data and 81 books used as test data (Figure 11). This sequence of

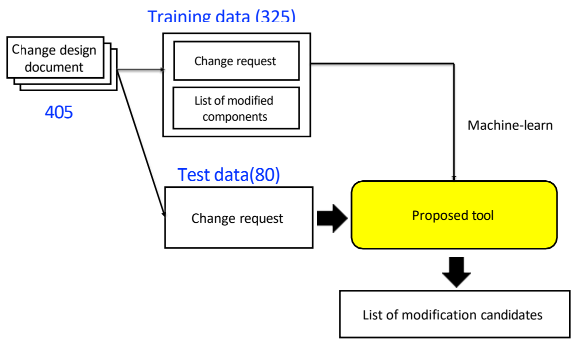


Figure 11 Data size used for the experiment.

experiments were conducted five times with other combinations, and the average of the results was calculated as the experimental result.

C. *Experimental Methods*

The experiment was conducted according to the following procedure.

1. Change request sentences are vectorized by noun extraction + doc2vec and features are extracted.
2. The machine learning component is configured using Scikit-learn. BR/LP/RAkEL methods as multi-label classifiers (k=3) ＋Implemented three SVMs: + SVM.
3. For the five sets of experimental data shown in Figure 11, the following were performed and averaged to calculate the accuracy:
   a) Create a training model from training data.
   b) Estimation of labels is performed on the remaining test data.

TABLE III: BR METHOD + SVM Values

Threshold	Percentage of candidate Range	Accuracy in the candidate range	Missing rate
0.04	38.1%	14.7%	12.3%
0.05	33.5%	17.1%	15.2%
0.06	*29.4%*	*19.1%*	*17.1%*
0.07	25.7%	21.0%	20.5%
0.08	23.1%	22.4%	23.7%
0.09	21.1%	23.1%	28.1%
0.1	19.4%	24.5%	29.9%

TABLE IV: LP METHOD + SVM

Threshold	Percentage of candidate Range	Accuracy in the candidate range	Missing rate
0.04	46.0%	16.8%	10.0%
0.05	39.9%	18.7%	13.4%
0.06	35.8%	19.9%	16.7%
0.07	32.8%	21.0%	19.7%
0.08	*29.7%*	*22.2%*	*23.0%*
0.09	26.6%	23.7%	26.4%
0.1	24.1%	25.0%	29.9%

TABLE V: RAkEL method ＋SVM

Threshold	Percentage of candidate Range	Accuracy in the candidate range	Missing rate
0.04	41.4%	18.7%	9.6%
0.05	36.5%	20.8%	11.3%
0.06	32.6%	22.7%	13.5%
0.07	*29.5%*	*24.5%*	*15.6%*
0.08	27.2%	26.2%	16.7%
0.09	25.2%	27.6%	18.8%
0.1	23.1%	29.3%	20.9%

D. Evaluation methods

Three measures were obtained: the candidate range ratio, which indicates effectiveness of narrowing the review range; the accuracy in the candidate range, which indicates amount of waste in the review process; and the missing rate, which indicates adequacy of determining candidate range.

The sigmoid threshold was moved in 0.01 increments until the candidate range ratio over 30 percent. The threshold value where it is the closest to 30 for each algorism is selected for comparison.

E. Experimental results

Table 3-5 show the values of three measures respectively for method 2-4. The values are shown in the range of 0.04 to 0.1 for sigmoid values, in line with previous studies. The value when it is closest to 30% is shown.

Table 6 compares the accuracy of four methods (including CNN's in Table 4) around a candidate range ratio of 30%.

TABLE VI: COMPARISON RESULTS OF ALL METHODS

method	Candidate range ratio (threshold)	Accuracy in the candidate range	Missing rate
CNN(Previous research)	30.00% (0.06)	18.00%	23.00%
BR＋SVM	29.10% (0.06)	19.10%	17.10%
LP+SVM	29.70% (0.08)	22.20%	23.00%
RAkEL+SVM	29.50% (0.07)	24.50%	15.60%

The results of our analysis are:

- The accuracy of the error rate was improved by 5.9% when comparing the BR method + SVM with the conventional method (CNN). This result indicates that SVM is more accurate than CNN for this problem.
- When comparing the BR and LP methods, the LP method was less accurate than the BR method, which is not the expected result because it must have superiority of the method that takes co-occurrence relationships into account. This is most likely due to overlearning, as the number of output labels is as large as 32, resulting in a huge number of combinations.
- The RAkEL + SVM method is the most accurate of the above methods, improving the missing rate by 1.5 points and the accuracy in the candidate range by 5.4 points compared to the BR method (improving 7.4 and 6.5 to CNN respectively). This result indicates that the RAkEL method did not cause overlearning problems, showing that the co-occurrence relationship is effective in improving accuracy to some extent.
- The highest accuracy of the RAkEL method was 15.6%, and the target accuracy of 5.0% or less could not be achieved.

V. Conclusion

In this paper, we compared and evaluated a total of four algorithms: two that take co-occurrence relationships into account (LP and RAkEL) and two that do not (CNN and BR). The results of the experiment show that the RAkEL method considerably improves the accuracy from the CNN in the previous study.

Future work includes further improvement of algorisms such as the application of the improved RAkEL method (overlapping version). Further validation of the effectiveness of the proposed method by applying it to another dataset is also needed, including exploring the necessary size of historical data to obtain a certain accuracy.

References

[1] S. Sikka, A. Dhamija, Software Change Impact Analysis, BookRix, 2020.

[2] Bohner, Impact analysis in the software change process, a year 2000 perspective, Proceedings of International Conference on Software Maintenance, 1996, pp. 42-51.

[3] ISO/IEC/IEEE 24765, 2017 Systems and software engineering - Vocabulary.

[4] H. Iwasaki, et al, A Software Impact Analysis Tool based on Change History Learning and its Evaluation, ICSE-SEIP '22, May 21 – 29, 2022, Pittsburgh, PA, USA.

[5] N. Motoi, T. Nakajima, and N. Kuno, A case study of applying software product line engineering to the air conditioner domain, Proceedings of the 20th International Systems and Software Product Line Conference, 2016, pp.220-226.

[6] K. Kobata, E. Nakai, and T. Tsuda, Process Improvement using XDDP - Application of XDDP to the Car Navigation System, 5th World Congress for Software Quality, Shanghai, China, November 2011.

[7] T. Kudo, K. Yamamoto, and Y. Matsumoto, 2004 Applying conditional random fields to Japanese morphological analysis, In Proceedings of the Conference on Empirical Methods in Natural Language Processing, volume 2004.

[8] Q. Le, T. Milkolov, Distributed representations of sentences and documents, International conference on machine learning, pp.1186-1196, 2014.

[9] Y, Kosuke, et al, "Interdependence Model for Multi-label Classification." International Conference on Artificial Neural Network*s*. Springer, Cham, 2019.

[10] V. Vapnik, The nature of statistical learning theory, Springer science & business media, 1999.

[11] G. Tsoumakas, and I. Katakis, Multilabel classification, An overview, Int J Data Warehousing and Mining, Vol. 2007, pp.1–13.

[12] G. Tsoumakas, I. Katakis, and I. Vlahavas, "Random k-labelsets for multi-label classification," IEEE Transactions on Knowledge and Data Engineering, vol. 99, no. 1, 201.