Multi-label Software Behavior Learning

Yang Feng, Zhenyu Chen∗
State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210093, China
Software Institute, Nanjing University, Nanjing 210093, China
∗corresponding author: zychen@software.nju.edu.cn

Abstract—Software behavior learning is an important task in software engineering. Software behavior is usually represented as a program execution. It is expected that similar executions have similar behavior, i.e. revealing the same faults. Single-label learning has been used to assign a single label (fault) to a failing execution in the existing efforts. However, a failing execution may be caused by several faults simultaneously. Hence, it needs to assign multiple labels to support software engineering tasks in practice. In this paper, we present multi-label software behavior learning. A well-known multi-label learning algorithm ML-KNN [14] is introduced to achieve comprehensive learning of software behavior. We conducted a preliminary experiment on two industrial programs: flex and grep. The experimental results show that multi-label learning can produce more precise and complete results than single-label learning.

Keywords—Software behavior learning; multi-label learning; F-measure; failure report classification; failure prediction

I. INTRODUCTION

Software behavior learning is one of the most important tasks in all stages of software development lifecycle [2]. Software behavior is usually represented as a program execution, and then similar executions are identified and grouped into a category. This grouping could help understand software behavior and assist software engineering tasks. Many applications of software behavior learning have been studied, including test case selection [5][13], failure report classification [10], failure prediction [2][7], and debugging [8][9].

A fundamental idea behind software behavior learning is failure proximity, i.e. the similarity of two failing executions due to a same fault [9]. The successes of these applications should be based on well-established results of failure proximity. That is, labels (faults) should be properly assigned to failing executions. However, an ideal learning result of software behavior is difficult to be obtained in practice. This inspires us to present new models and introduce new learning algorithms to improve software behavior learning.

In all these existing efforts of software behavior learning, an execution is only assigned with a single label. This is limited by single-label learning. The assumption under these applications is that a failing execution only corresponds to a fault [9]. The strong assumption could simplify solutions of software behavior learning. However, this simplification increases risks of incomplete or even wrong software behavior learning, which will affect software engineering tasks finally. Multi-label learning was initially motivated by the challenges in text categorization, in which each document may belong to several topics simultaneously [11]. Multi-label learning have been applied in many real world problems. A failing execution may be caused by several faults simultaneously in practice. That is, a failing execution should be assigned with multiple labels (faults). As much as we know, there is no effort on multi-label learning of software behavior. This paper firstly present multi-label software behavior learning and its preliminary studies.

A. Motivation

In this paper, We use multi-label failure report classification as an example to explain our motivations, but multi-label learning is not restricted to it. It can assist to other software engineering tasks, such as testing, debugging, etc. Many modern software products could automatically collect the execution information of failure and report it to developers. A main challenge of these systems is that too many failure reports must be diagnosed by developers. It is necessary to simplify the work by grouping failure reports. Please notice that a failure report may be caused by multiple faults.

Single-label learning may produce incomplete learning results and then damage software engineering tasks. For example, some new failures caused by both fault $F_A$ and fault $F_B$ are reported. These failure reports are assigned to fault $F_A$ by single-label learning. As a result, the failure reports cannot be processed completely and fault $F_B$ is ignored. On the other hand, if the failure reports are assigned to fault $F_B$, then the failure reports also cannot be processed completely and fault $F_A$ is difficult to diagnose. Furthermore, single-label learning in a multi-label task may damage the training process. Finally it may produce some wrong learning results.

B. Contribution

The main contributions of this paper are as follows.

• We firstly present the multi-label learning tasks in software behavior learning. We formalize it as a multi-label model and introduce a well-known multi-label learning algorithm ML-KNN [14] into the solution.
• A preliminary experiment was conducted. We provided insight into the risks of single-label learning in software engineering. The experimental results show that
Multi-label software behavior learning totally outperform single-label learning.

II. Approach

A. Multi-label Model of Software Behavior

We adopt the failure indexing model [9] and extend it to a multi-label model of software behavior. Each fault is denoted by a label \( l_i \). To facilitate the discussion, we use fault(s) and label(s) interchangeably. \( L \) is a set of labels \( \{l_1, l_2, \ldots, l_m\} \). Given a set of failing inputs(tests) \( X = \{x_1, x_2, \ldots, x_n\} \) caused by \( L \), we assume that an oracle model \( \Phi : X \rightarrow 2^L \). Each input \( x_i \) in \( X \) is mapped to a subset \( L_i \subseteq L \), in which \( x_i \) is caused by faults in \( L_i \). We denote \( \Phi(x_i) = L_i \).

\( \Phi \) is unknown in practice. Hence, it is expected to obtain a training model \( C \) by learning the execution information of \( X \). The problem of software behavior learning is to obtain \( C \), such that the difference between \( C(x) \) and \( \Phi(x) \) is minimized for each \( x \). For each label \( l \), \( \Phi_l = \{x|l \in \Phi(x)\} \) and \( C_l = \{x|l \in C(x)\} \). In this paper, we use \( C_S \) and \( C^M \) to denote the single-label model and the multi-label model, respectively.

In different application scenarios, \( L \) may be known or not. It can use supervised learning (classification) or unsupervised learning (clustering). In this paper, we will focus on supervised learning. A recent effort on semi-supervised learning of software behavior is proposed to deal with both label and unlabel data simultaneously [3]. Multi-label learning could also be generalized to unsupervised learning, and even semi-supervised learning. But we cannot discuss them in detail due to the limited pages.

B. Multi-label Learning

The task of supervised learning is to classify new instances (inputs) based on learning from a training set of instances \( X^T \) that have been properly labeled. The \( k \)-nearest neighbor (KNN) is one of the most popular and simplest supervised learning algorithms. An unknown instance will be assigned to the group most common amongst its \( k \) nearest neighbors by KNN [1].

Single-label learning is limited by one instance with only one label. Multi-label learning is a non-trivial generalization by removing the restriction and it has been a hot topic in machine learning [11][14][12]. We introduce a well-known multi-label learning algorithm ML-KNN [14] into the solution. The most important difference between ML-KNN and KNN is the aggregation of the label sets of these instances. For each \( x_i \), KNN outputs \( L_i \) containing only one label and ML-KNN outputs \( L_i \) containing one or more labels. KNN and ML-KNN are used in our experiment for single-label learning and multi-label learning, respectively.

C. Framework

The framework of software behavior learning is shown in Figure 1. In order to collect the execution information, the program \( P \) is instrumented in advance. Each input \( x_i \) in \( X^T \) is run on \( P \) and the corresponding execution of \( x_i \) is collected. The label information of each \( x_i \) in \( X^T \) will be obtained by diagnosing the failures or other ways. The classifier \( C \) is trained by a certain algorithm with the executions and their labels. \( C \) is used to predict the labels of new instances. The label information of \( X^E \) could be used to assist software engineering tasks.

In single-label software behavior learning, \( X^T \) with its single-label information will be used to train \( C_S \). However, multi-label data are omnipresent in real applications. Hence, it usually randomly select one label to train \( C_S \). For a new input \( x \), \( C_S \) outputs one label for \( x \). Multi-label learning is a generalization of single-label learning. It uses multi-label data to train \( C^M \), which can output one or more labels for new \( x \).

D. Evaluation Metric

In order to evaluate \( C_S \) and \( C^M \), we adopt a widely-used metric F-measure in machine learning [11]. For each label \( l \), \( |\Phi_l \cap C_l| \) is the number of instances assigned correctly to \( l \). Recall \( (r_l) \) and precision \( (p_l) \) are defined as follows:

\[
  r_l = \frac{|\Phi_l \cap C_l|}{|\Phi_l|}, \quad p_l = \frac{|\Phi_l \cap C_l|}{|C_l|},
\]

following by the concept of information retrieval. F-measure is the harmonic mean of recall and precision as follows:

\[
  F\text{-measure} = \frac{1}{|L|} \sum_{l \in L} \frac{2r_l p_l}{r_l + p_l} \quad (1)
\]

The F-measure values are in \((0,1]\). Larger F-measure values correspond to higher quality of software behavior learning.

III. Preliminary Experiment

A. Subject Programs

We used two subject programs, flex and grep, and their test sets from the “Software-artifact Infrastructure Repository”(SIR) [6]. In order to compare multi-label learning and single-label learning, we selected the versions of programs which have more than 3 revealed faults. Hence 7 versions from the initial 10 versions were used in the experiment.

In order to evaluate the experimental results, we obtained the truth of label information. A version of program was seeded with a fault \( l \in L \) in each time. Each test in \( X \) was run to determine whether it can reveal the fault \( l \). In this step, we obtained the label information \( \Phi : X \rightarrow L \).
We do not consider the interference of multiple faults [4] in this experiment. It might increase the threats of validity. However, the culpability determination of interference of multiple faults will result in a combination challenge [9]. We would design and conduct more comprehensive experiments in the future.

### Table I

**Table I**

**BASIC INFORMATION OF SUBJECT PROGRAMS**

<table>
<thead>
<tr>
<th>ID</th>
<th>ID F1</th>
<th>F2</th>
<th>F3</th>
<th>F4</th>
<th>G1</th>
<th>G2</th>
<th>G3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ver</td>
<td>2.5.1</td>
<td>2.5.2</td>
<td>2.5.3</td>
<td>2.5.4</td>
<td>2.5.5</td>
<td>2.5.6</td>
<td>2.5.7</td>
</tr>
<tr>
<td></td>
<td>14</td>
<td>9</td>
<td>11</td>
<td>5</td>
<td>7</td>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>552</td>
<td>237</td>
<td>542</td>
<td>560</td>
<td>781</td>
<td>158</td>
<td>784</td>
</tr>
<tr>
<td></td>
<td>544</td>
<td>113</td>
<td>542</td>
<td>559</td>
<td>324</td>
<td>29</td>
<td>675</td>
</tr>
<tr>
<td>AL</td>
<td>6.68</td>
<td>1.98</td>
<td>5.22</td>
<td>4.89</td>
<td>1.42</td>
<td>1.18</td>
<td>3.54</td>
</tr>
</tbody>
</table>

We summarize the results of subject programs in Table I. We use “ID” to facilitate discussion. “F*” and “G*” denote the version number. |L| and |X| denote the numbers of faults and the number of all failing inputs, respectively. |L|s in Table I were modest. We believe that applications with large |L| may be more suitable for multi-label learning. “NM” denotes the number of multi-label failing inputs. The results in Table I show that multi-label inputs are common cases. In some cases (F1, F3, F4, G3), almost all failing inputs are multi-label ones. In these cases, the averages of |L|s (“AL’s”) are modest, because |L| is not large.

### B. Experiment Setup

X was partitioned into two disjoint sets X^T and X^E for each version. X^T is a training set and X^E is an evaluation set. |X^T| : |X^E| = 1 : 4 in the experiment. We repeated the experiment 30 times by randomly selecting different instances to build X^T.

In the training stage, a set of labels L_i was assigned to x_i properly for each x_i ∈ X^T based on Φ in advance. X^T with its label information was used to train a multi-label classifier C^M. In order to train a single-label classifier C^S, the multi-label data must be reduced to single-label data. Hence, one label in L_i was randomly selected for each x_i. C^S was trained by the single-label data.

In the application stage, X^E was input to C^S and C^M to obtain single-label information and multi-label information, respectively. The label information could be used to assist software engineering tasks. These tasks should be based on high quality label information.

In order to evaluate the quality of learning results, a set of labels L_i was assigned to x_i properly for each x_i ∈ X^E based on Φ in advance. X^E with its label information was used to evaluate both C^S and C^M. For each label l, we calculated |Φ_l Π C^S_l| and |Φ_l Π C^M_l|, respectively. Then the recall, precision and F-measure were calculated for single-label learning and multi-label learning, respectively.

### C. Experimental Results

The experimental results is shown in Figure 2. “S” and “M” denote the results of single-label learning and multi-label learning, respectively. It was obvious that multi-label learning outperformed single-label learning in all cases. The improvement of multi-label learning was significant in most cases, except F2 and F4. For F4, the F-measure value of F4S was already very high. Hence there was little space for improvement. For F2, the low “AL” values might be one reason. However, G1 and G2 with low “AL’s” still achieved significant improvement. The influence of “AL” should be studied further by more empirical studies.

To confirm the results in a more rigorous way, we applied one-tail t-test for the improvement of F-measure, as shown in Table II. All of these improvements were statistically significant (“h=1”). All of p-values were much less that the standard value 0.05. The experimental results indicated that multi-label learning could improve software behavior learning significantly, with regard to F-measure. Therefore, multi-label software behavior learning could assist software engineering tasks more effectively.

### Table II

**Table II**

**RESULTS OF T-TEST**

<table>
<thead>
<tr>
<th>ID</th>
<th>F1</th>
<th>F2</th>
<th>F3</th>
<th>F4</th>
<th>G1</th>
<th>G2</th>
<th>G3</th>
</tr>
</thead>
<tbody>
<tr>
<td>p-value</td>
<td>1e-28</td>
<td>4e-36</td>
<td>3e-16</td>
<td>8e-20</td>
<td>3e-28</td>
<td>2e-30</td>
<td>7e-9</td>
</tr>
</tbody>
</table>

![Figure 2. Experimental Results](image-url)
IV. RELATED WORK AND DISCUSSION

The related work could be mainly classified into two types: software behavior learning and multi-label learning.

There are many related efforts on software behavior learning in the recent years. In [2][7], supervised learning are used to predict failing executions. There are only two labels: “pass” and “fail”, and each input only has one label in their works. In [5][13][8][9], unsupervised learning are used to cluster tests. There are implicit labels for clusters and each test belongs to only one cluster. In [10], unsupervised learning is used for failure report classification and supervised learning is used for feature selection to improve unsupervised learning. Recently, semi-supervised learning is introduced to assist test case selection [3]. As much as we know, all of the existing efforts of software behavior learning are based on single-label learning. A novel contribution of this paper is firstly presenting multi-label software behavior learning and its preliminary studies.

Multi-label learning are useful in many real world problems. For example, a picture may belong to several semantics class: beach, sun and tree. A gene may be associated with several functional classes: metabolism, transcription and protein synthesis. More recent efforts on multi-label learning could be found in [12]. In this paper, we provide a novel application of multi-label learning on software engineering. An intuition behind our approach is the observation that a failure may be caused by several faults simultaneously in real applications.

In this paper, we use labels to denote faults. Multi-label learning could be applied in other software engineering tasks by extending the meanings of labels. We can use implicit meanings of labels, such that fuzzy clustering could be used to deal with the problems in [5][13][8][9]. The meanings of labels could be extended to performance bottlenecks and security vulnerabilities, and then multi-label learning could be used in performance tuning and security analysis.

V. CONCLUSION AND FUTURE WORK

Software behavior learning is an important task in software engineering. The main challenges include modeling software behavior and identifying similar software behavior. A natural idea is to improve software behavior learning by some features of real applications. We observe that a failing execution may be caused by several faults simultaneously. Following by the observation, we firstly present multi-label software behavior learning. A well-known multi-label learning algorithm ML-KNN is introduced to produce more comprehensive results. The experimental results show that multi-label learning can improve the effectiveness of software behavior learning significantly. Furthermore, we discuss the broad impact of multi-label learning on software engineering tasks.

Multi-label software behavior learning is still preliminary. There are many aspects that could be improved or studied in the future. A number of multi-label learning algorithms could be used and they can optimize the solution. The implementation of our approach would be improved and a series of comprehensive experiments would be conducted. We also would studied concrete software engineering tasks based on multi-label learning in the future.

VI. ACKNOWLEDGMENTS

The work described in this article was partially supported by the National Natural Science Foundation of China (90818027, 61003024, 61170067). The authors would like to thank Yufeng Li and Dr. Liming Zhang for their valuable discussions on multi-label learning.

REFERENCES