IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received September 19, 2019, accepted September 29, 2019, date of publication October 10, 2019,
date of current version October 21, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2945608

Automatic Real-Time Mining Software Process Activities From SVN Logs Using a Naive Bayes Classifier

RUI ZHU[1,2], YICHAO DAI[4], TONG LI[2,3], ZIFEI MA[1], MING ZHENG[1], YAHUI TANG[1], JIAYI YUAN[1], AND YUE HUANG[1]

[1]School of Software, Yunnan University, Kunming 650091, China
[2]Key Laboratory in Software Engineering of Yunnan Province, Kunming 650091, China
[3]College of Big Data, Yunnan Agricultural University, Kunming 650091, China
[4]College of Computer, National University of Defense Technology, Changsha 410073, China

Corresponding authors: Tong Li (tli@yun.edu.cn) and Yichao Dai (daiyichaowing@163.com)

This work was supported in part by the National Natural Science Foundation of China under Grant 61662085 and Grant 61862065, in part by the Yunnan Provincial Natural Science Foundation Fundamental Research Project under Grant 2019FB-16, in part by the Project of Yunnan Provincial Department of Education Science Research Fund under Grant 2017ZZX227, in part by the Yunnan University Data-Driven Software Engineering Provincial Science and Technology Innovation Team Project under Grant 2017HC012, in part by the Yunnan University Dong Lu Young-backbone Teacher Training Program, and in part by the Yunnan University Education Department Science Research Fund Graduate Program under Grant 2019Y0008 and Grant 2019Y0010.

ABSTRACT The abundance of event data in current software configuration management systems makes it possible to discover software process models automatically by using actual observed behavior. However, traditional process mining algorithms cannot be applied to event logs recorded in software configuration management (SCM) systems, such as SVN, because of missing activity attributes. To address this problem, a software process activity classifier is proposed to build event-activity mapping relationships from software development event streams, revealing activity attributes and associating the activity to the original SVN log. The proposed approach extracts activity from the SVN log based on semantic features and introduces a novel technique based on a naive Bayes approach to associate event activities dynamically. The approach has been applied to two real-world software development process logs, *ArgoUML* and *jEdit*, consisting of more than 80,000 events, covering development information from 1998 to 2015. With the application of our approach to such data, activities can be extracted from event logs and a classifier can be constructed for adding activity attributes to new events. The results of the classification are evaluated in terms of *precision rate*, *recall rate,* and the *F-measure*. Overall, two real-world software development process logs are used to validate the method, and the experimental results show that the approach can mine software process activities from SVN log events automatically and in real-time.

INDEX TERMS Activity classifier, machine learning, software process activity, SVN log.

I. INTRODUCTION

Nowadays, it is widely accepted that the quality of software is not only related to the product, but to the organization and to the production process that is carried out [1]–[3]. Software process modeling helps to create process descriptions that correspond to processes actually performed during software development or maintenance. Process models can be used to visualize tacit knowledge, roles, and information flows in the processes, identifying points for improvement and optimization [4], [5].

However, with the deepening of research on software development, problems associated with traditional subjective modeling methods have become apparent. These arise because the task of designing a software process model is complex and error prone, and because the life cycle of the model is short, individuals are not sensitive to differences between actual processes and the process model, and there are increasing requirements for process engineers, all while the software process is still evolving [6]–[8]. As software systems become more and more complex, the establishment

The associate editor coordinating the review of this manuscript and approving it for publication was Weizhi Meng.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see http://creativecommons.org/licenses/by/4.0/

of a sound process model is becoming more like "an art rather than a science" [9]. Therefore, current modeling methods cannot effectively meet the needs of modern software engineering [10]. In the era of big data, how to automatically extract a process model from the massive process data generated by existing software development organizations has become a focus of current software process research.

To automatically extract process models from software development log systems, some software process mining methods [6], [11]–[14] have been proposed. However, these works simply attempt to apply the traditional business process mining approach [15] to the software process domain, but ignore the structure of software development logs, which are significantly different from the business process. The most significant difference is the lack of activity attributes; this problem can be solved using an activity classifier.

Activity extraction (discovery) in software process mining is analogous to classification in data mining and it is essentially a mapping problem. This mapping is between log events and SPAs (Software Process Activities) based on their associations. However, although activity extraction is very important for software process mining, little attention has been paid to build appropriate SPAs classifiers. Aalst [9] proposed a classifier that is commonly used in the field of process mining, in which activity is determined by its attributes. This classifier is widely used in business process research, but it cannot be applied to software processes that only have a single case. Yang *et al.* [16] proposed a business process mining method appropriate for a range of diverse environments, and which classified logs based on domain knowledge. However, it focused on the classification of business process logs based on cases and subdivided logs into sub-logs based on case classification. Therefore, it would not be suitable for software process mining. Rubin *et al.* [6] proposed an activities-extraction method by filtering keywords in paths. This method can rapidly generate mapping results but is not very accurate and the activity discovery is granular. Rui *et al.* [17] proposed an activities-mapping method based on entropy, and which can extract activities chronologically. However, there is no comparison test, no consideration of semantic information in the event log, and no specific criteria for determining the optimal number of clusters. In traditional process mining, the process consists of cases and cases consist of events that are ordered in time and have attributes such as activities, timestamps, and so on [9]. However, software process mining depends on the software process log, which has only one case, and there are no activities in the events of the case. This study uses events in the software development process log as the research object to investigate the feasibility of extracting activities based on the log. The purpose of this research is to improve existing methods of activity mining, to improve the efficiency and accuracy of activity mining, and to further discover software process models from development events.

The main contributions of this study are as follows:

(1) We proposed *K-means* clustering, building on *Word2Vec* vectorization, to address the problem of a lack of initial activity labeling in software process logs. This enables the creation of initial labels, where those whose semantic information is similar are situated close together and labels whose semantic information is different are far apart.

(2) We combine supervised learning with unsupervised learning to solve the problem of a lack of activity information in software process log events and effectively classify these activities.

(3) In unsupervised learning, we propose a method to determine the optimal number of clusters by taking the second derivative, and introducing a method of calculating a plurality of harmonic averages for accuracy and a recall rate for assessing the classifier. An evaluation of the classifier is implemented so that the results are more objective.

The remainder of this article is organized as follows.

Section II discusses the background and some preliminaries. Section III presents the general framework of the paper. Section IV proposes the method to extract activity from SVN logs based on semantic features. Section V establishes a dynamic incremental method of event-activity mapping based on a naive Bayes classifier. Section VI presents the experimental results of mining two real software development logs. Finally, the conclusions and future work are described in Section VII.

II. BACKGROUND

The software process refers to a series of activities to develop and maintain software products. The attributes of each activity involve relating products (artifacts), resources (people or other resources), organizational structure, and constraints [18]. Research on traditional software processes is divided into two categories: (1) process evaluation and improvement, represented by Capability Maturity Model Integration (CMMI) [19], [20]; and (2) software process modeling [21], [22], which abstracts the software processes through specific methods to increase understanding of the software development process and is used to guide the activities of software development [23].

The Software Process Model is also known as the software life cycle or software development model; it describes how various activities in the software process are executed and it also forms a structural framework for the whole process. The software process model establishes the order and boundaries of each stage of software development, as well as the activities of each stage, which can intuitively express the whole process of software development, define the main activities and tasks to be completed, and serve as the basis for project implementation [24].

Software process activities (SPAs) represent the behavioral information of the software development process, where similar behavior is represented as a kind of activity. Activity data is necessary for software process mining [12], [25]. By using the process mining algorithm, a sequence composed of

activities is divided into cases and then the model is mined. The ultimate goal is to rapidly extract a simple, reasonable, and high-quality process model to support software development activities.

Software Process Mining [12], [13], [17] refers to revealing the software process model automatically using data from the software process itself in order to help software engineers better identify, understand, analyze, and optimize software development and to ultimately improve the quality of software products.

In software process mining, software process data is the foundation of mining. It is generated by activities such as software development, evolution, maintenance, and testing [26]. The management system for maintaining these data includes the software configuration management system (SCM), project management software (PMS), defect tracking system (DTS), etc. In addition to managing the documentation, these systems can also collect and store behavioral information about software processes, such as who created it, accessed or changed the documents, and the time that tasks were submitted and completed [6].

A software process event log is a two-dimensional table that is used in process mining to describe the information in the event log. The event log is the beginning of the process mining. A traditional event log is defined as follows.

Definition 1 [Event, Attribute [9]]: ξ represents the event space, which is the collection of all possible event identifiers. Events are described by attributes, where AN is a collection of attribute names. For any event $e \in \xi$, the attribute name $an \in AN$, $\#an(e)$ represents the value of attribute an for event e. If event e does not contain any attribute names, then $\#an(e) = \perp$(null).

Definition 2 [Case, Trace, Event Log [9]]: ζ represents the case space, which is the collection of all possible case identifiers. The case is related to events and has attributes in which the trace is a special mandatory attribute of the case, expressed as $\#trace(c) \in \xi^*$. A trace is a finite sequence of events, denoted by $\sigma, \sigma \in \xi^*$. An event log is a collection of cases, denoted as $L, L \subseteq \zeta$.

On the basis of the traditional event log, the definition of the software development process event log is given below.

Definition 3 [Software Development Process Event, Attribute [17]]: ξ is the event space of the software development process; that is, the collection of all possible event identifiers. Events are described by attributes, where AN is the collection of attribute names. For any event $le \in \xi$, attribute name $an \in AN$, $\#an(le)$ represents the value of attribute an of event le. If event le does not contain any attribute names, then $\#an(le) = \perp$(null).

Definition 4 [Software Development Process Trace, Software Development Process Log [17]]: Let A be a set of activities, then a sequence $\sigma \in A^*$ is a trace of the software development process and$L \in P(A^*)$ is a software development process log, where $P(A^*)$ is a power set of A^*.

The attributes of software development process events are abstracted from the process data and the determination of the attribute is dependent on whether it can effectively support the subsequent mining work. This article defines the attribute set $AN = \{id, date, paths, msg\}$ of software development process event le. The id attribute is the unique identifier of the event, and the $date$ attribute specifies the moment when the event occurs. The $paths$ attribute specifies which files are affected by the development activity and the msg attribute is a description of the event. These attributes can be nested, such as the $paths$ attribute which is composed of multiple $path$ attributes, and the $path$ is composed of $action$ and $address$, in which $action$ indicates file operations such as the creation, deletion, or modification of a file. The $address$ attribute is a specific path.

The software development process log L is simply referred to as the process log and the software development process trace σ can simply be referred to as the process trace. The definition is defined from the point of view of the activity, while in reality there is no activity and case information in the process data, only events. The event log contains a lot of information about the whole event, but in the theoretical aspect of process mining, most of the time we are only concerned with the name of the activity, case, event, and so on. Therefore, in order to simplify the event log, the concept of a simple event log is given below.

Definition 5 [Simple Process Log [17]]: A simple development process log L is a collection of traces that is defined as a multiple set of traces on A; that is, $L \in B(A^*)$. For example, $L_1 = [< a, b, c, d >_4, < a, e, f, g >_8, < a, e, d >]$, from which it can be seen that the simple process log L_1 contains 13 traces, of which trace $< a, b, c, d >$ occurs four times, trace $< a, e, f, g >$ occurs eight times, and trace $< a, e, d >$ occurs once. In the simple process log, there are no attributes, time stamps, or resource information, and the order of events from left to right represents the order in which the events occurred.

Definition 6 [Association Between Event and Activity [17]]: In process mining, processes are composed of activities and activities consist of tasks. A log is a record or a true reflection of a set of tasks and is also a collection of multiple event traces. Here, a case is an event trace that manifests itself as a form of trace consisting of activities, such as $< a, b, c, d >$. The relationship between activities and events is a mapping relationship. The association between event le and activity a is record $leRa$, which is a subset of cartesian set $E \times A$ in event log spaces E and A among software activity sets; that is, $leRa \in \{<le, a> | le \in E \wedge a \in A\}$, and the correlation function $R(le) = a$.

Definition 7 [Activity Classifier) [17]]: Let L be a process log, A is a collection of activities on L. The correlation between process events and activities is R, then the event sequence $<le_1, le_2, \ldots, le_n>$ is transformed into a trace where $\sigma = < R(le_1), R(le_2), \ldots, R(le_n) >$. For example, suppose that there are event sequences $<le_1, le_2, le_3, le_4, le_5, le_6, le_7>$; among them, $\{le_1, le_2, le_3\}$ Ra, le_4Rb, $\{le_5, le_6\}$ Rc, le_7Rd. The event sequence can be transformed into trace $< a, b, c, d >$ by the activity classifier.

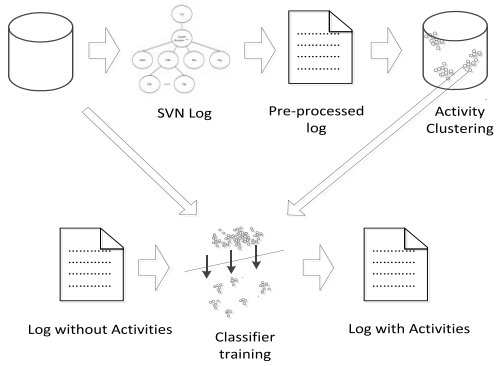

FIGURE 1. Flow diagram of activity mapping.

III. GENERAL FRAMEWORK

There is only one case in the software process log and there are no activities in the events contained in the case, and this is an important reason for why we cannot directly apply business process mining algorithms. Therefore, it is necessary to extract activities from the case and classify them to carry out subsequent software process mining. The events of the software process log are contained within implicit semantic information; we do not know what each event did directly, but we can find out which events are doing similar or different things. Based on this idea, this study conducted activity discovery and classification on the software process logs using the techniques of clustering and classification. The overall flow of this method is divided into three main parts (Fig. 1).

Firstly, software development process events are vectorized. The vectors can be used to compute the similarity between events. We preprocess the software log to extract the *path* and *msg* attributes of each event. Then we organize these attributes using natural language processing. The preprocessing includes the removal of noise, extraction of word stems and selection of speech. Then, we process the structured data to obtain a corpus to complete the vectorization of data based on semantic features. In natural language processing, word vectorization is a way to create a vocabulary library where every word in the sequence is numbered. In practical application, sparse coding is widely used, based on the number of words. One of the biggest problems for this method of representation is that it cannot capture the similarity between words, it ignores the semantic relevance of words and it is prone to dimensionality problems. To address this problem, this study, inspired by the *Distributed representation* method, maps every word onto a K-dimensional real number vector by training (K is generally a hyper-parameter in the model), and the semantic similarity between words is represented by the distance between them. We use the *Word2Vec* model for this task. *Word2Vec* is an open source tool, developed by Google, that obtains word vectors by training on a corpus and then evaluates them [23]. The word vector obtained by training the *Word2Vec* model contains semantic information, retains semantic similarity between words, and overcomes the problem that traditional text-feature-representation methods do not represent semantic information and context. In *Word2Vec*, the degree of similarity between word vectors is found by computing a distance, which represents the similarity between words.

Secondly, the *K-means* approach is used to cluster events as activities based on parts of datasets and then use these labeled datasets to train the Activity Classifier. We cluster the vectored data based on the distance between them and determine the optimal number of clusters by quadratic differential derivation. After training, we extract the most relevant events by clustering, using semantic mapping, and then relate them to the same activity. By extracting activities in this way, it is possible to represent the internal cohesion of events in the software process log and so discover the relationship between each event and activity. The *K-means* algorithm is a widely used clustering algorithm that expresses classes of centroids and clusters by digital attributes. The *K-means* is an iterative process, which first selects k objects in the data space as initial cluster centers, where each object represents one class center [27]. For other data objects, the *K-means* algorithm calculates the Euclidean distance between them and the cluster center, then assigns them to the most similar cluster center according to their distance from it. Next, it calculates the average value of all the objects in each class as a new clustering center and then calculates the sum of squares distance from all the data to their cluster center until the clustering center and the sum of squares distance does not change. The clustering process is complete when the sum of squares of distance is minimized.

In *K-means* clustering, it is necessary to define the number of cluster categories beforehand and the choice has a significant effect on the results. We can set a reasonable range of clusters by comparing the clustering results and then base subsequent work on those results. Since software process logs are constantly updated, a lot of new event data is added after the relationship between events and activities has been found. To associate events in the new software process log with activities, we can construct an activity classifier and predict the activity class of the new data by using data in the training set.

Thirdly, the trained Activity Classifier is used to discover the activities from the new data based on a naive Bayes approach, thereby achieving a dynamic incremental method of event-activity mapping. The clustering results are obtained as the training sample to construct the classifier and complete the mapping between activities and classes. The classification is divided into four stages; namely, preparation, training, application, and evaluation.

IV. ACTIVITY EXTRACTION FROM THE SVN LOG BASED ON SEMANTIC FEATURES

A. VECTORIZATION OF THE SVN LOG

SVN (Subversion) is a common software configuration management (*SCM*) system whose log is a record of software development activities. The *SVN* log can be used to analyze and mine the software development process. It can be exported as an *XML* document. The *SVN* log is composed of

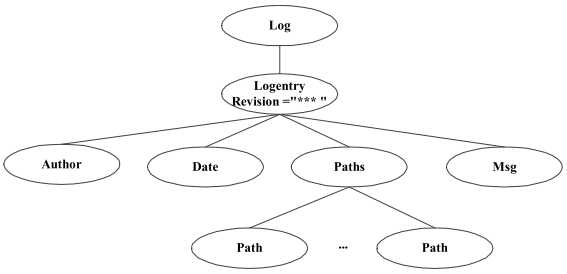

FIGURE 2. Tree structure of an *SVN* log.

entries in the log and each entry represents an event. These records hold the version (*Revision*), the author (*Author*), the date (*Date*), the path (*Path*), and the behavior (*Msg*). They can be represented as a tree structure, as shown in Fig. 2.

The record of events in the *SVN* log does not contain any case information and does not have any associated activity information. The structural comparison between the traditional event log and the *SVN* log data is shown in Fig. 3.

The information we use for event mining is presented as text in the *SVN* log; the natural language information in the text cannot be used for mining activities directly. Text must be converted for machine processing. The software process log has only one case, and the case information is presented as natural language text. Unstructured data is a common format for storing information in natural language processing. This study first realizes the conversion from unstructured to structured text. The significance of preprocessing is that noise can be removed, and the textual data can be structured to represent semantic information more effectively. The traditional method of text processing is to quantify words to create a thesaurus, but this ignores the semantic information and context; thus, it does not convey the meaning of words. If we cannot reflect the word meaning, natural language processing becomes purely statistical. The *Word2Vec*[1] open source tool can learn high-dimensional word vectors from a large corpus and has been used to represent linear semantic relationships among word vectors. In this study, *Paths* and *Msg* information are extracted from the *SVN* log and vectorized. The result of the quantification will be used as input to the clustering activities to further study the effect of clustering based on semantic relationships in activity mining. This study uses the *skip_gram* training framework, which has three layers: the input layer, the projection layer, and the output layer, as showed in Fig. 4.

For the training samples, we predict the contextual information of the word based on knowledge of the current word. First, the word vector corresponding to the current word w is input, and the word vector corresponding to the current word is projected. A binary frequency tree is constructed using the word frequency of the corpus where the word is located as the weight, and the leaf node corresponds to the word in the vocabulary. Suppose the number of leaf nodes is N and the

[1]https://code.google.com/archive/p/word2vec/

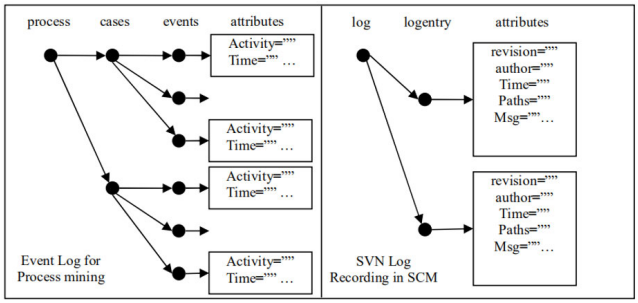

FIGURE 3. Comparison of structures for traditional event logs and *SVN* log data.

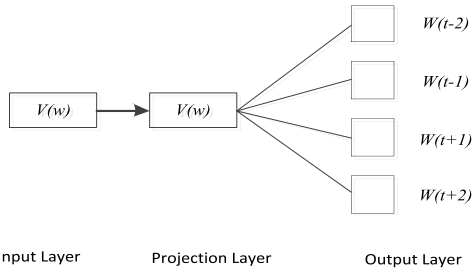

FIGURE 4. Framework for *Skip_gram* training.

number of non-leaf nodes is N-1. For the sample [w, context (w)], when its context word $w' = w$, the tag is set to 1, otherwise the tag is set to 0. The word vectors in the vocabulary and auxiliary vectors corresponding to the output non-leaf nodes are taken as training parameters. Finally, we output the corresponding vector of leaf nodes and the auxiliary vector corresponding to non-leaf nodes.

B. ACTIVITY CLUSTERING BASED ON K-MEANS

After the case text is structured, this paper uses *K-means* clustering of the word vectors with the semantic features contained in the event information obtained by the *Word2Vec* method to complete the clustering of the texts of each event in the case and then obtains categories. The labels serve as the training set for the activity classifier. Here, we use the *K-means* clustering algorithm to cluster log activities as follows.

First, the eigenvectors of k structured log text messages are randomly assigned as the initial vector coordinates of k centroids. Then, the sequence number and representation vectors of short text M_i are extracted from the structured log information $M = \{M_1, M_2, \ldots M_n\}$. Next, we calculate the distance from M_i to the k centroids and select the center of mass nearest to the centroid. Finally, we output the sequence number of the cluster center and representation of vectors M_i. It can be seen from the above steps that the number of *K-means* clusters directly affects the results and they are not consistent owing to the random element in the initial centroid determination process. Therefore, this study presents a method of determining the optimal number of activities based on a quadratic differential.

C. DETERMINING THE OPTIMAL NUMBER OF ACTIVITIES

As the number of clusters affects the results and the value of k needs to be set in advance, we can iterate over a range to determine the optimal value of k using a *Cost Function*. The minimum value of the corresponding k indicates how many clusters of K to select. In practice, since *K-means* clustering is generally used for data preprocessing or for assortment classification labeling, k is usually set to a small number. It is possible to enumerate k from 2 to a fixed value such as 10 and run *K-means* repeatedly on each value of k to find the best solution and then calculate the average contour coefficient of the current k, and finally to select it. The value of the largest contour coefficient corresponds to k as the final number of clusters. This paper adopts the method of setting the number of clusters within a certain range. In order to avoid finding a local rather than a global solution, this paper uses the second derivative method to choose the optimal value of k. In the second derivative formula, x represents the argument, y is the dependent variable, and h is the step size:

$$y'' = [y(x_0 + h) - 2 * y(x_0) + y(x_0 - h)]/h^2 \quad (1)$$

The *K-means* clustering method can find the average distance between the clustering result and the cluster center. In theory, the smaller the value, the better the clustering effect. However, as the number of clusters increases, the distance will also decrease. Therefore, distance alone is not a reliable estimate of the optimum number of clusters. However, when the value decreases slowly, increasing the number of clusters further does not enhance the clustering effect significantly. To address this problem, this study uses the method of calculating the rate of change and selects the number of clusters corresponding to the inflection point of the fastest-growing clustering effect. Since our data are discrete, we can use the method of function fitting to form these data into functions, and then take the derivative of the function. However, the process is complex, and so we use the second derivative of the difference formula.

V. DYNAMIC INCREMENTAL METHOD OF EVENT-ACTIVITY MAPPING

A. CREATING A NAIVE BAYES CLASSIFICATION MODEL

The software process development log is not tagged with activities. Constructing a classifier is applied as a supervised learning method so that we can assess the classification results by mapping activities from unsupervised learning to supervised learning. We need to generate a more reliable label, and so we propose a *K-means* clustering method based on the *Word2Vec* vectorization method, and which considers semantic information, because there is no real label from the beginning. The sample set is divided into K clusters according to the distance between the samples, and the points in the cluster are kept as close together as possible to make the distance between clusters as large as possible. What we obtain will be semantically similar text content for the label. One of the advantages of machine learning is its ability to handle new data. We cannot put the new data into the training data because the number of clusters may be affected. Hence, we build a classifier to handle the new data. The purpose of clustering is to obtain classes for training purposes, and then the classifier is used to categorize new data into these tags.

We study software development process log attributes through keyword extraction and analysis, and then process the new activity information. The log of the software development process facilitates the subsequent process mining research. Based on this condition, naive Bayesian classification is applied to classify log events to avoid the problem of an over simplistic assumption of independence. In practice, the classification information contained in some features of the naive Bayesian classifier can provide higher accuracy. Therefore, we use the previously obtained clustering categories to train the naive Bayesian classifier; that is, to classify the object class mark obtained by clustering a known class mark in the naive Bayesian classification problem in order to complete event-to-activity category mapping. In the classification problem, the category is the target variable and a *k-fold* cross validation method is used to test the accuracy and recall rate of the classification results.

We assume that $X = \{M_1, M_2, \ldots, M_n\}$ is a set of n messages to be classified and that they are mutually independent. Let $C = \{Y_1, Y_2, \ldots, Y_n\}$ be a collection of categories in the software development process. The mapping rule $Y = f(M)$ is determined such that any $M_i \in X$ has only one $Y_i \in C$ such that $Y_i \in f(M_i)$ holds where f is the classifier. The training dataset is set to $T = \{(M_1, Y_1), (M_2, Y_2), \ldots, (M_t, Y_t)\}$. We choose the class with the largest posterior probability as the final classification of the message and achieve expectation risk minimization. The derivation of maximizing the posterior probability according to the expected risk minimization is explained in detail by Li [28]. The work involved in this section is divided into four stages: preparation, training, application, and evaluation.

1) PREPARATION

During preparation, the main task is to determine the characteristics of the attributes, and some of the data to be classified are labeled for training. We have used clustering annotation instead of manual annotation. The input data is classified and then output as feature attributes and training sample data. The quality of the results of this stage is dependent upon the training sample data by the characterization of attributes. Feature attribute classification is often more important than classifier selection.

2) TRAINING

The main aim of the training phase is frequency calculation. We calculate the frequency and feature attributes that appear in the training sample data for the category. Then, we divide the conditional probability estimates for each category. In this stage, the input is the attributes and training sample data. The output is the classifier.

3) APPLICATION

In the application stage, we use the classifier. Here, the input is a classifier and the items to be classified. The output is the mapping between the item to be classified and the category.

4) EVALUATION

In this phase, the original test data is input as labels. The label of the cluster corresponds to the test set data. The classifier label of the test set corresponds to the test data, and the evaluation index is output. From the literature on existing classification evaluation methods, we can see that the generally accepted method is to calculate the accuracy, the recall, and the harmonic mean. Hence, we compare the classification results with the original clustering based on the accuracy P, the recall rate R, and the harmonic mean F of each classification index:

$$P = A/B \tag{2}$$

$$R = A/C \tag{3}$$

$$F - measure = (\alpha^2 + 1) \times P \times R / \alpha^2 \times (P + R) \tag{4}$$

where A = the number of correct classifications, B = the number of classifications, and C = the number of items in the sample.

B. EVENT-ACTIVITY MAPPING AND DYNAMIC UPDATES

As the software development and maintenance work progresses, new data will be appended to the software process log. Our goal is to map the activities of newly added events. In order to conduct real-time mining of the whole process, we update and control the activity mining process dynamically by regularly updating the logs as follows: the training data set is updated once every 3 months, the event data in the most recent software process log is marked as a training set by clustering. The new log events added each time will be used as the test set and the naive Bayesian classification model will be used for activity mapping before the next update of the training set. The result of the mapping will be marked as the current activity of adding the event information. After the next training set is updated, the log event information that has already been marked by the activity will be added as a new training set to the original training set and re-labeled through clustering. This dynamic update cycle is represented in Fig. 5. In each training set update, the initial activity label will be updated accordingly. The best activity number is also updated to improve the activity map. For example, for a newly added event e, it needs to be classified according to the existing activity classifier C, which is done in real time. But in order to increase the accuracy of the classifier model, it is necessary to re-update the classifier as the training data after a period of time to obtain classifier C'.

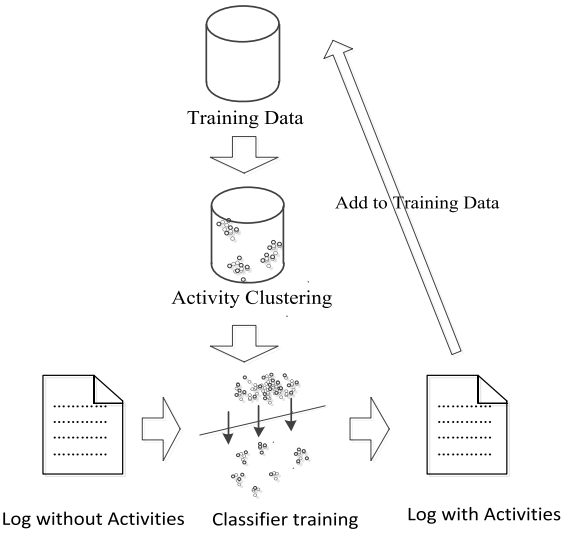

FIGURE 5. Real-time dynamic mining log activity cycle.

```
<?xml version="1.0" encoding="UTF-8"?>
<log>
<logentry
  revision="19961">
<author>bobtarling</author>
<date>2015-01-11T22:13:53.949437Z</date>
<paths>
<path
  action="M"
  kind="">/trunk/src/argouml-app/src/org/argouml/uml/util/ModelUtil.java</path>
</paths>
<msg>Only use navigable associations to generate package dependencies</msg>
</logentry>
<logentry
  revision="19960">
<author>bobtarling</author>
<date>2015-01-11T18:10:42.554995Z</date>
<paths>
<path
  action="A"
  kind="">/trunk/src/argouml-app/src/org/argouml/uml/util/ModelUtil.java</path>
</paths>
<msg>Move method to generate package from classes out of the Java RE module and into core argo
to allow reuse elsewhere</msg>
</logentry>
……
</log>
```

FIGURE 6. Part of the argolog.xml software process log document.

VI. EXPERIMENTATION

A. INSTANCE SPECIFICATION

This experiment uses the software development process logs as the experimental data set. We extract *ArgoUML* software development process log information and *jEdit* software development process log information to conduct the experiment. *ArgoUML* is a well-known open source *UML* modeling tool. It can support all the diagrams of the latest *UML* standards, has good cross-platform features and can run on all Java platforms; *jEdit* is a mature programmer's text editor with hundreds of person-years of development behind it. Figure 6 shows parts of the data in the software process log document argolog.xml. It contains 85,792 events, covering development information from 1998 to 2015, which is sufficient for our purpose.

We extract two attributes for each event, *msg* and *path*, to support activity discovery. The pretreatment is carried out according to three steps.

1) NOISE REMOVAL

We remove non-text data from the original content (e.g., digital information and punctuation) to avoid potential problems

```
trunk src argouml app src org argouml modelutil java use associ generat packag depend
trunk src argouml app src org argouml modelutil java move method generat class java modul core
argo reus elsewh
trunk src argouml app src org argouml cmd relationshipactionfactori java depend ad diagram
element
trunk modul dev argouml modul dev share project argouml dev argouml tigri org svn trunk modul
dev
trunk src argouml core umlpropertypanel src argouml core propertypanel model metamodel xml issu
show packag
trunk src argouml app src org argouml cognit clattributecompart java fix dash line test issu
trunk histori html ad date relea
```

FIGURE 7. Preprocessed software process log document argolog_preprocess.txt.

encountered in data processing. The result after processing is a string that only consists of words and spaces.

2) SELECTION OF PARTS-OF-SPEECH

The activities of each event in the case are mainly expressed by verbs but individual verbs do not convey the complete meaning. An event is composed of the verb and a corresponding noun. This study uses the natural language processing tool *NLTK* to filter the part-of-speech of the content. Finally, we obtain the gerund (a verb that acts as a noun) of the attributes *msg* and *path* and show them in a row.

3) STEM EXTRACTION

This article considers software development process logs in English. Here, it is necessary to have words with the same general form. The same English word exists in many forms, with different singular and plural tenses. For example, the word *make* can take the forms *makes*, *made*, *making*, etc. We would like all of them to be converted to the prototype stem "*make*". After the above processing, the event log is expressed as a collection of verbs, nouns, and space symbols, all of which have value in representing the meaning of the text. Figure 7 is a partial representation of the preprocessed data.

B. SETTING TRAINING PARAMETERS

To allow for variance in the experimental data, we divide the entire dataset into two parts: training and test data. Considering the impact of different parts of the datasets on the overall results, we perform 10 experiments using k-fold cross validation. The principle is that the data are divided into k sub-samples. In each experiment, a different sample is taken as the verification data and the remaining $k-1$ samples act as the training data.

C. ANALYSIS OF EXPERIMENTAL RESULTS

Sufficient data are collected from three randomized experimental data sets to conduct Word2Vec vectoring and K-means clustering. We set the number of clusters from 1 to 100 to find the relationship between the number of clusters k and the *Average Distance* of the corresponding points for each category label relative to the clustering center. After creating the relationship graph, we take the second derivative of the difference formula to obtain a new graph that reflects the relationship between the number of clusters k and the rate of change in an average distance of the corresponding points

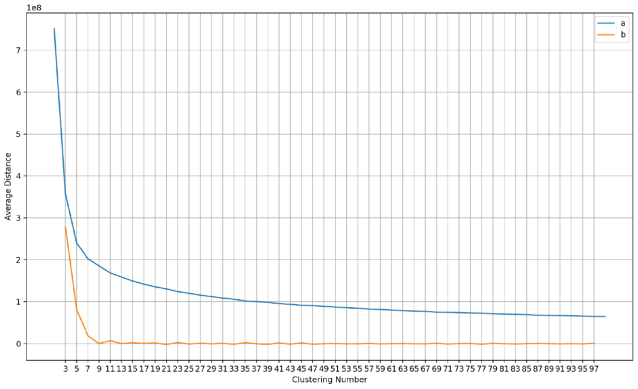

FIGURE 8. Function curve of clustering.

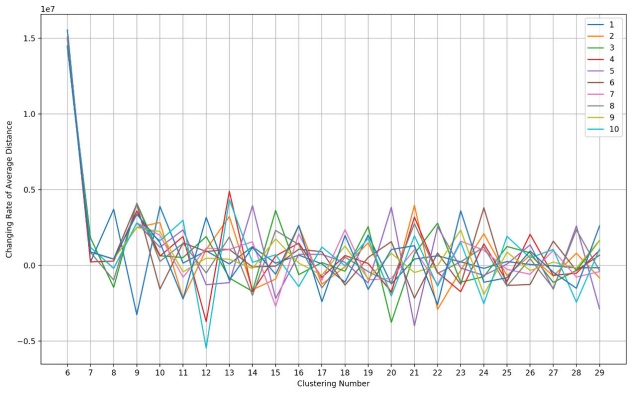

FIGURE 9. Selecting the number of clusters.

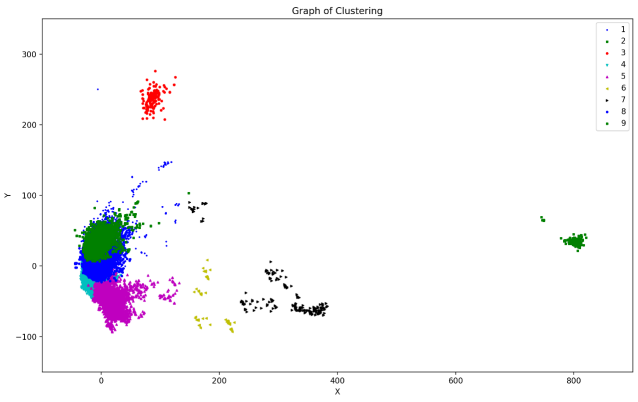

FIGURE 10. Effect of optimal cluster number.

for each category label relative to the cluster center. Take the *ArgoUML* log experiment as an example (Fig. 8), curve *a* is the function between k and the average distance, and curve *b* is the corresponding function of k and the average rate of change in distance.

As each initial point in the *K-means* clustering is set at random while the training set in the experiments is fixed, we conducted 20 repetitions of the clustering and selected the average of the number of clusters as the final optimal number. Figure 9 is a composite of 20 experiments showing the second

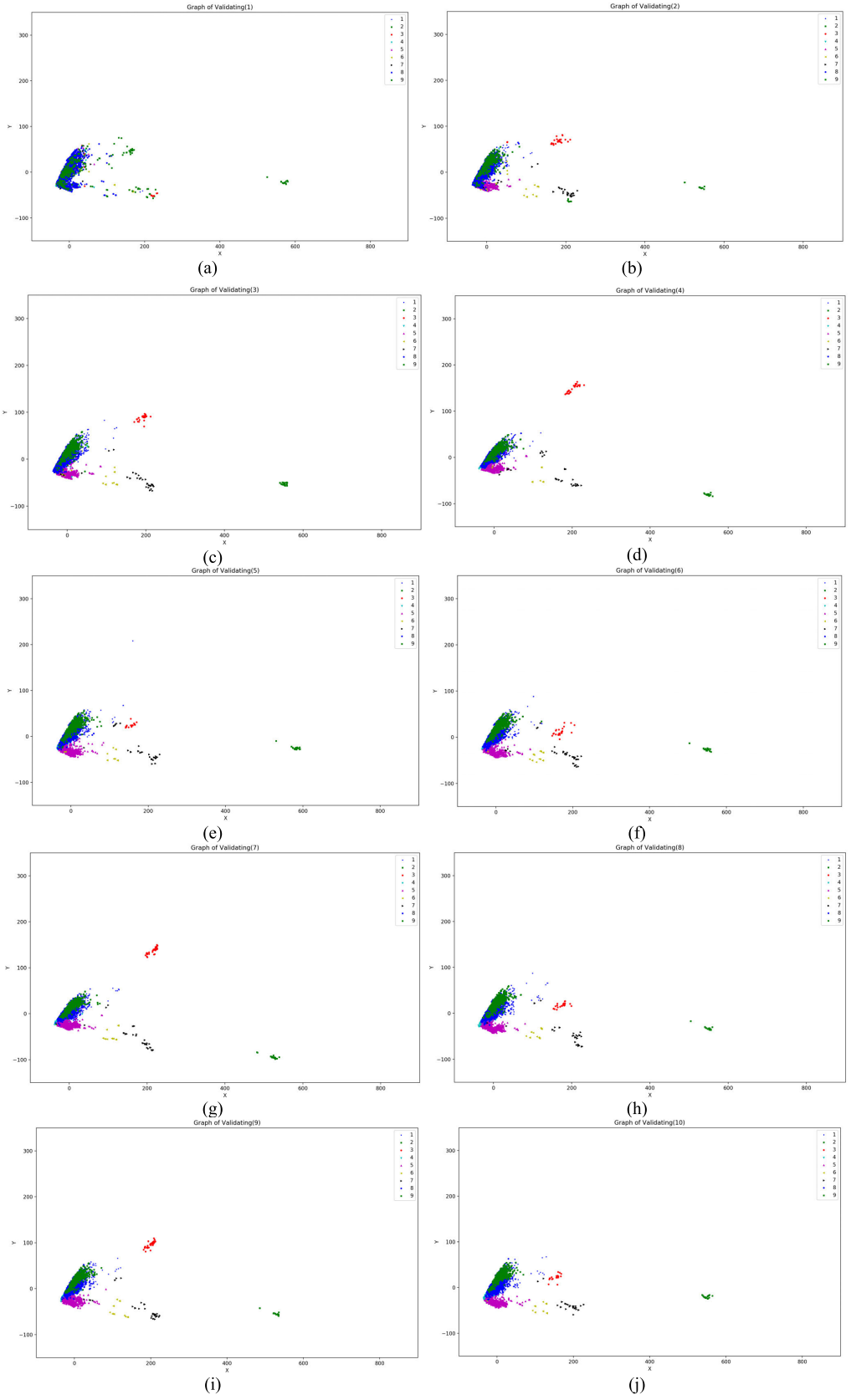

FIGURE 11. Validation of the classifier.

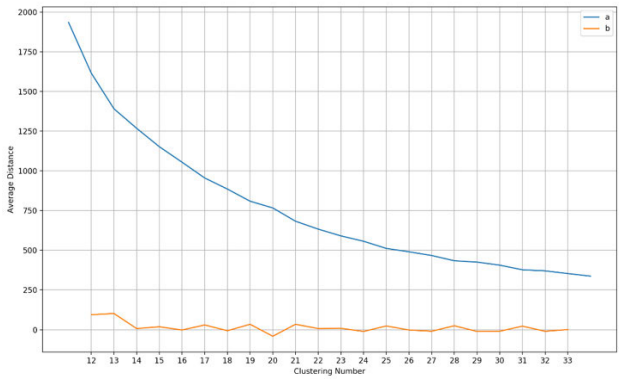

FIGURE 12. Average distances using two contrasting methods.

TABLE 1. Manual verification of partial clustering tags.

Cluster tags	Context of paths and msg
8	/trunk/src/argouml-app/src/org/argouml/uml/util/ModelUtil.java Only use navigable associations to generate package dependencies
8	/trunk/src/argouml-app/src/org/argouml/uml/util/ModelUtil.java Move method to generate package from classes out of the Java RE module and into core argo to allow reuse elsewhere
8	/trunk/src/argouml-app/src/org/argouml/ui/cmd/RelationshipActionFactory.java Allow dependencies to be added to diagram on any element
0	/trunk/modules/dev/argouml-module-dev Share project "argouml-module-dev" into "http://argouml.tigris.org/svn/argouml/trunk/modules/dev"
0	/trunk/src/argouml-core-umlpropertypanels/src/org/argouml/core/propertypanels/model/metamodel.xml Issue 6500 - Show dependencies for packages
8	/trunk/src/argouml-app/src/org/argouml/uml/cognitive/critics/ClAttributeCompartment.java Fix an exception when clicking on the dashed line of an associationclass. Discovered when testing issue 4463
5	/trunk/www/history.html Added some dates of releases.
8	/trunk/src/argouml-app/src/org/argouml/ui/cmd/RelationshipActionFactory.java Fix some exceptions when right-clicking on elements in an activity diagram
8	/trunk/src/argouml-app/src/org/argouml/ui/explorer/ExplorerPopup.java Fix for issue 5237: Allow creation of association class via explorer Thanks for the patch, Bob! I added a limit to creating an association class between 2 classes at maximum.
8	/trunk/src/argouml-app/src/org/argouml/uml/diagram/ui/FigAssociationClass.java Do not show a red wavy line in the top left corner of the DIAGRAM when selecting an unnamed association class
8	/trunk/src/argouml-app/src/org/argouml/uml/cognitive/critics/ClAttributeCompartment.java Fix for issue 6252: Adding associationclass does not add node. I am still unsure about the architecture, but this is a severe bug and the fix cannot wait longer. Tested again - it works.
3	/trunk/pom.xml; /trunk/src/argouml-app/pom.xml; /trunk/src/argouml-core-diagrams-activity2/pom.xml; /trunk/src/argouml-app/src/org/argouml/application/ArgoVersion.java Updated poms to start work towards 0.35.2.
3	/trunk/www; /trunk/tools; /trunk/src/argouml-core-umlpropertypanels; /trunk/src/argouml-core-transformer; /trunk/src/argouml-core-diagrams-activity2 Updated the subclipse:tags tag.

derivative relational graph of the difference between formula clustering number k and the *Average Distance*. We take the average of the corresponding k in 20 experiments as the final number of clusters when the second derivative drops to zero, as shown in Fig. 10.

After automatic clustering, we extract part of the event information and conduct manual confirmation to verify tag partition appropriateness, as shown in Table 1, which depicts some paths and message information content of *logentry* as well as their corresponding cluster tags. This part of the data covers four categories. Among them, updating scale and

TABLE 2. Results of classifier validation for the *ArgoUML* log (units: centesimal system).

Class name	Precision	Recall	F-measure θ = 0.5	F-measure θ = 1	F-measure θ = 1.5
class 1	93.55	79.37	74.47	76.05	77.24
class 2	87.61	100	88.55	90.46	92.31
class 3	83.19	90.27	84.21	86.00	87.39
class 4	82.05	75.05	80.20	77.99	76.77
class 5	82.28	85.77	82.87	83.88	84.58
class 6	84.17	93.56	85.46	87.77	89.59
class 7	82.84	89.42	84.00	85.87	87.16
class 8	87.50	88.87	87.54	87.77	88.07
class 9	80.53	78.96	79.85	79.29	79.10

TABLE 3. Results of classifier validation for the *jEdit* log (units: centesimal system).

Class name	Precision	Recall	F-measure θ = 0.5	F-measure θ = 1	F-measure θ = 1.5
class 1	97.33	86.48	94.93	91.57	89.54
class 2	60	95	63.21	70	76.51
class 3	49.05	73.33	50.05	53.95	58.34
class 4	90	85	86.11	83.33	83.02
class 5	49.58	84.88	53.84	62.02	68.94
class 6	61.97	81.2	64.92	70.04	73.87

TABLE 4. SVN log revision number bound to active Class 6.

Activity revision	C6 23820	C6 23335	C6 23298	C6 23166	C6 23165	C6 23017	C6 23011	C6 22935	C6 22931	C6 20635
Activity revision	C6 20470	C6 20428	C6 20212	C6 20132	C6 20130	C6 20126	C6 20125	C6 20124	C6 19975	C6 20428

updating label is a class, adding and deleting information is a class, repairing events is a class, and authorized displaying events is a class, all of which appear plausible.

After determining the number of clusters, we divide the data into a validation set and a training set and extract 1/10 of the training set each time. After 10 experiments, we calculate the *precision rate*, *recall rate,* and their harmonic mean, known as the *F-Measure* using parameters of 0.5, 1.0, and 1.5. Then, we conduct *10-fold cross validation*. The results of the 10 experiments and of the *10-fold cross validation* are shown in Fig. 11a–j and Table 2, respectively. The results of 10-*fold cross validation* for the *jEdit* log is shown in Table 3.

The results show that training a data set based on our improved naive Bayesian classification algorithm for activity mapping of *SVN* log events is more accurate than the fuzzy clustering method. Take the *ArgoUML* log experiment as an example, by using the fuzzy clustering method for the same data set, we find that the optimum number of clusters is $k = 31$ and the average distance between cluster centers is 383.77. The optimum number of clusters obtained using the proposed method is $k = 14$ and the average distance between cluster centers is 1267.42. Compared with the rate of change of the evaluation distance between the two, as shown by curve *b* of Fig. 12, when $k = 14$, it has reached the inflection point

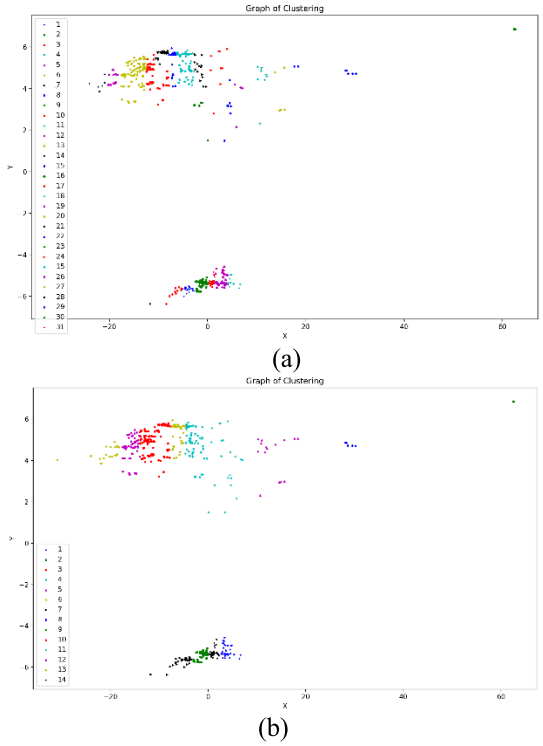

FIGURE 13. Clustering effect diagram using two contrasting methods.

FIGURE 14. Adding activity categories tags to the *SVN* log.

Document Log

Document	Date	Author
project1/models/design.mdl	01.01.05 14:30	designer
project1/src/Code.java	01.01.05 15:00	developer
project1/tests/testPlan.xml	05.01.05 10:00	qaengineer
project1/docs/review.pdf	07.01.05 11:00	manager
project2/models/design.mdl	01.02.05 11:00	designer
project2/tests/testPlan.xml	15.02.05 17:00	qaengineer
project2/src/NewCode.java	20.02.05 09:00	developer
project2/docs/review.pdf	28.02.05 18:45	designer
project3/models/design.mdl	01.03.05 11:00	designer
project3/models/verification.xml	15.03.05 17:00	qaengineer
project3/src/GenCode.java	20.03.05 09:00	designer
project3/review/Areview.pdf	28.03.05 18:45	manager

Filtered Log

Document
DES
CODE
TEST
REV
DES
TEST
CODE
REV
DES
VER
CODE
REV

FIGURE 15. Example of the mapping method based on document path name.

and the subsequent change has little effect on curve *a*. At the same time, compared with the rate of change of the evaluation distance between the two methods, we find that using the proposed method is better for mapping the results.

The clustering map does not change significantly when the number of clusters is reduced from 31 (Fig. 13a) to 14 (Fig. 13b).

This experiment is based on the software development process log information of *ArgoUML* and realizes a method of extracting activities from the software process log. In order to address the problem of a lack of activity attributes in the log, we propose a method to extract software process activities based on the correlation between events and activities. We extract each record of events in the *SVN* log and map its contents. Then, we move from unsupervised learning to supervised learning by constructing a classifier to map new activities based on the tag of activities by clustering. Finally, we assessed the results of the classification using *precision rate*, *recall rate,* and the *F-measure*.

The results show that this method can effectively extract activity information from the log and solve the problem of the lack of initial activity tags, which makes the initial label partitioning plausible. For the *ArgoUML* log, on using the test data sets, we find that the average *precision rate*, *recall rate,* and *F-measure* (using parameters of 0.5, 1 and 1.5) produce values of 0.85, 0.87, 0.83, 0.84, and 0.85 respectively. For the *jEdit* log, on using the test data sets, we find that the average *precision rate*, *recall rate,* and *F-measure* (using parameters of 0.5, 1 and 1.5) produce values of 0.68, 0.84, 0.69, 0.72, and 0.75 respectively. These results illustrate the effectiveness of this method in multiple dimensions of evaluation.

The final goal is to insert the activity categories tag by mapping into the original *SVN* log without activity categories tags. As shown in Fig. 14, the original *SVN* log (Fig. 14a) becomes a new *SVN* log (Fig. 14b) to begin a follow-up that generates the single trigger sequence in process mining.

To verify the validity of our results, they are compared with those from the commonly used mapping method. The mapping method first defines several activities, such as the four activities defined in Fig. 15 (Design, DES; Code, CODE; Test, TEST; and Review, REV), based on the correlation between the name of the document path involved and the defined activity, and log filtering.

The mapping method [8], [6], [29] supports the naming of the document, but ignores the semantic information of each modification, which makes the approach less applicable and less accurate. Table 3 shows the revision number of *logentry* in the SVN log bound to active Class 6 after

<logentry revision="23820">
<author>ezust</author>
<date>2015-01-04T01:22:00.224072Z</date>
<paths>
<path text-mods="true" kind="file" action="M" prop-mods="false">/jEdit/trunk/doc/FAQ/faq-install.xml</path>
<path text-mods="true" kind="file" action="M" prop-mods="false">/jEdit/trunk/doc/FAQ/faq-use.xml</path>
</paths>
<msg>FAQ: Reformatted whitespace and made small updates.
</msg>
</logentry>

<logentry revision="23298">
<author>ezust</author>
<date>2013-10-29T15:41:30.563399Z</date>
<paths>
<path prop-mods="false" text-mods="true" kind="file" action="M">/jEdit/trunk/doc/FAQ/faq-use.xml</path>
</paths>
<msg>How to toggle auto-indent FAQ.
</msg>
</logentry>

FIGURE 16. Binding two *logentry* in the same activity.

mining by the method proposed in this study. At the same time, the relationship can be further viewed between each *logentry*. The records for revision number 23820 and revision number 23298 are shown in Fig. 16; we find that both have modified FAQ information. Through this comparison, it can be easily found that the proposed method not only considers the semantic information of the path, but also increases the accuracy of the activity discovery, taking into account the message information of each log.

VII. CONCLUSION AND FURTHER WORK

In this study, we developed a method to extract software process activities based on the correlation between events and activities. This method extracts each record of events in the log and processes its contents. Then, it uses unsupervised and supervised learning by constructing a classifier based on clustering to extract software process activities. We assess the result of the classification in terms of *precision rate*, *recall rate*, and the *F-measure*. The method addresses the problem that software process logs do not contain initial activity labels. It enables the classification of initial labels and represents labels whose semantic information is similar to that of labels whose semantic information is different. At the same time, this study effectively addressed the problem of determining the optimum number of clusters. The optimal clustering number occurs when the average distance between cluster centers decreases at the fastest rate. This paper also enables the mapping of activities in the software process log and introduces a method of calculating multiple harmonic averages for the accuracy and recall rate in the evaluation of the classifier.

However, there remain areas for improvement: The size of the existing database sample of events in the software process log is limited. However, with the continued use of the software development process, the log will be continually updated. At the same time, we will obtain more data for clustering, which will improve the accuracy of the classifier. In addition, the weight of certain indicators used in event mapping is uncertain, and there is no unified standard of measurement. Finally, since there is no standard method of evaluating the results of unsupervised learning and the environment that it is used in is complex, there is more scope for development in this aspect of unsupervised learning.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviews and editors for their suggestions.

REFERENCES

[1] R. Bendraou, J.-M. Jezequel, M.-P. Gervais, and X. Blanc, "A comparison of six UML-based languages for software process modeling," *IEEE Trans. Softw. Eng.*, vol. 36, no. 5, pp. 662–675, Sep./Oct. 2010.
[2] G. A. García-Mireles, M. Á. Moraga, F. García, and M. Piattini, "Towards the harmonization of process and product oriented software quality approaches," in *Proc. Eur. Conf. Softw. Process Improvement*. Springer, 2012, pp. 133–144.
[3] A. Hachemi and M. Ahmed-Nacer, "Reusing process patterns in software process models modification," *J. Softw., Evol. Process*, vol. 30, no. 8, 2018, Art. no. e1938.
[4] M. Unterkalmsteiner, T. Gorschek, A. K. M. M. Islam, C. K. Cheng, R. B. Permadi, and R. Feldt, "Evaluation and measurement of software process improvement—A systematic literature review," *IEEE Trans. Softw. Eng.*, vol. 38, no. 2, pp. 398–424, Mar./Apr. 2012.
[5] T. Allweyer, *BPMN 2.0: Introduction to the Standard for Business Process Modeling*. Norderstedt, Germany: Books on Demand GmbH, 2016.
[6] V. Rubin, C. W. Günther, W. M. P. van der Aalst, E. Kindler, B. F. van Dongen, and W. Schäfer, "Process mining framework for software processes," presented at the Int. Conf. Softw. Process, Minneapolis, MN, USA, May 2007.
[7] V. Rubin, I. Lomazova, and W. M. P. van der Aalst, "Agile development with software process mining," presented at the Int. Conf. Softw. Syst. Process, Nanjing, China, May 2014, pp. 26–28.
[8] T. Gürgen, A. Tarhan, and N. A. Karagöz, "An integrated infrastructure using process mining techniques for software process verification," in *Computer Systems and Software Engineering: Concepts, Methodologies, Tools, and Applications*. Hershey, PA, USA: IGI Global, 2018, pp. 1503–1522.
[9] W. M. P. van der Aalst, *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Berlin, Germany: Springer-Verlag, 2011.
[10] A. M. Valle, E. A. P. Santos, and E. R. Loures, "Applying process mining techniques in software process appraisals," *Inf. Softw. Technol.*, vol. 87, pp. 19–31, Jul. 2017.
[11] S. Bala and J. Mendling, "Monitoring the software development process with process mining," presented at the Int. Symp. Bus. Modeling Softw. Design, 2018.
[12] L. Dong, B. Liu, Z. Li, O. Wu, M. A. Babar, and B. Xue, "A mapping study on mining software process," presented at the 24th Asia–Pacific Softw. Eng. Conf. (APSEC), 2018.
[13] J. Caldeira and F. B. E. Abreu, "Software development process mining: Discovery, conformance checking and enhancement," presented at the Qual. Inf. Commun. Technol., 2017.
[14] J. E. Cook and A. L. Wolf, "Discovering models of software processes from event-based data," *ACM Trans. Softw. Eng. Methodol.*, vol. 7, no. 3, pp. 215–249, 1998.
[15] W. M. P. van der Aalst, *Process Mining: Data Science in Action*. Berlin, Germany: Springer, 2016.
[16] L. Q. Yang, G. S. Kang, L. P. Guo, L. Zhang, X. N. Zhang, and X. Gao, "Process mining approach for diverse application environments," (in Chinese), *J. Softw.*, vol. 26, no. 3, pp. 550–561, 2015.
[17] Z. Rui, L. Tong, M. Qi, H. Zhenli, Y. Qian, and W. Yiquan, "Data-driven bilayer software process mining," *Ruan Jian Xue Bao/J. Softw.*, vol. 29, no. 11, pp. 3455–3483, 2018.
[18] A. Fuggetta and E. Di Nitto, "Software process," in *Proc. Future Softw. Eng.*, 2014, pp. 1–12.
[19] D. Falessi, M. Shaw, and K. Mullen, "Achieving and maintaining CMMI maturity level 5 in a small organization," *IEEE Softw.*, vol. 31, no. 5, pp. 80–86, Sep./Oct. 2014.
[20] D. Doss, R. Henley, D. McElreath, B. Gokaraju, R. Goza, and G. Lusk, "Process improvement: Urban vs. rural personnel perspectives of a derivative CMMi process maturity framework," in *Proc. Southwest Acad. Manage.*, 2017, pp. 44–51.

[21] L. García-Borgñón, M. A. Barcelona, J. A. García-García, M. Alba, and M. J. Escalona, "Software process modeling languages: A systematic literature review," *Inf. Softw. Technol.*, vol. 56, no. 2, pp. 103–116, 2014.
[22] A. A. Khan, J. W. Keung, Fazal-E-Amin, and M. Abdullah-Al-Wadud, "SPIIMM: Toward a model for software process improvement implementation and management in global software development," *IEEE Access*, vol. 5, pp. 13720–13741, 2017.
[23] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," Feb. 2013, *arXiv:1301.3781*. [Online]. Available: https://arxiv.org/abs/1301.3781
[24] T. Li, W. Wang, and Y. Yu, *Introduction to Software Engineering*. Beijing, China: The Science, 2012.
[25] A. Joonbakhsh and A. Sami, "Mining and extraction of personal software process measures through IDE interaction logs," presented at the 15th Int. Conf. Mining Softw. Repositories, 2018.
[26] J. Carmona and J. Cortadella, "Process discovery algorithms using numerical abstract domains," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 12, pp. 3064–3076, Dec. 2014.
[27] M. B. Cohen, S. Elder, C. Musco, C. Musco, and M. Persu, "Dimensionality reduction for k-means clustering and low rank approximation," in *Proc. 47th Annu. ACM Symp. Theory Comput.*, 2015, pp. 163–172.
[28] H. Li, *Statistical Learning Method*. Beijing, China: Tsinghua Univ. Press, 2012.
[29] M. Leemans and W. M. P. van der Aalst, "Process mining in software systems: Discovering real-life business transactions and process models from distributed systems," presented at the ACM/IEEE 18th Int. Conf. Model Driven Eng. Lang. Syst. (MODELS), Sep./Oct. 2015.

RUI ZHU was born in Kaifeng, Henan, China, in 1987. He received the Ph.D. degree in software engineering from Yunnan University, Kunming, China, in 2016, where he is currently a Lecturer with the School of Software, Yunnan University. He is also a Scholar of the Ali Living Water Project. He is an important member of the Data-Driven Software Engineering Department, Science and Technology Innovation Team, Yunnan University. His current research interest includes process mining. He was a recipient of the Yunnan Province Ph.D. Scholar Newcomer Award and a Secretary of the Yunnan Provincial Software Engineering Teaching Committee.

YICHAO DAI was born in Hunan, China. She received the master's degree in software engineering from Yunnan University, Kunming, China, in 2018. She is currently with the College of Computer, National University of Defense Technology. Her current research interest includes process mining.

TONG LI was born in Kunming, Yunnan, in 1963. He received the Ph.D. degree in software engineering from the School of Computer Science and Engineering, De Montfort University, U.K. He is currently a Professor and a Doctoral Tutor with the Young and Middle-Aged Academic, a Technical Leader, a Famous Teacher, and a Leader of the Teaching Team, Yunnan. He is also a Professor with the College of Big Data, Yunnan Agricultural University, the Director of the National Linux Technology Training and Promotion Center and the Key Laboratory in Software Engineering of Yunnan Province, and the Principal of psychology in Yunnan cloud computing engineering research.

ZIFEI MA was born in Kunming, Yunnan, in 1990. He is currently pursuing the Ph.D. degree in system analysis and integration with Yunnan University. His current research interests include machine learning and data mining.

MING ZHENG was born in Anqing, Anhui, in 1992. He received the B.S. degree in information and computing science from the School of Mathematics and Statistics, Yunnan University, Kunming, China, where he is currently pursuing the Ph.D. degree in information and communication engineering with the School of Information Science and Engineering. His current research interests include data mining and artificial intelligence.

YAHUI TANG was born in Lanzhou, Gansu, in 1995. She received the degree in software engineering technology from Yunnan University. Her current research interests include software process mining, especially on genetic mining algorithms. She was a recipient of the 2016 Yunnan Excellent Graduate and also received several scholarships during the school.

JIAYI YUAN was born in Hejin, Shanxi, in 1995. She received the bachelor's degree from the School of Software, North University of China, Taiyuan. She is currently pursuing the master's degree with the School of Software, Yunnan University. Her current research interest includes process mining.

YUE HUANG was born in Yuxi, Yunnan, in 1996. She received the bachelor's degree from the School of Software, Wuhan Polytechnic University of China. She is currently pursuing the master's degree with the School of Software, Yunnan University. Her current research interest includes data mining.