Computers, Materials & Continua CMC, vol.64, no.3, pp.1999-2012, 2020

A Recommendation Approach Based on Bayesian Networks for Clone Refactor

Ye Zhai[1, *], Dongsheng Liu[1], Celimuge Wu[2] and Rongrong She[1]

Abstract: Reusing code fragments by copying and pasting them with or without minor adaptation is a common activity in software development. As a result, software systems often contain sections of code that are very similar, called code clones. Code clones are beneficial in reducing software development costs and development risks. However, recent studies have indicated some negative impacts as a result. In order to effectively manage and utilize the clones, we design an approach for recommending refactoring clones based on a Bayesian network. Firstly, clone codes are detected from the source code. Secondly, the clones that need to be refactored are identified, and the static and evolutions features are extracted to build the feature database. Finally, the Bayesian network classifier is used for training and evaluating the classification results. Based on more than 640 refactor examples of five open source software developed in C, we observe a considerable enhancement. The results show that the accuracy of the approach is larger than 90%. We believe our approach will provide a more accurate and reasonable code refactoring and maintenance advice for software developers.

Keywords: Clone code, clone refactor, feature extraction, Bayesian network.

1 Introduction

Code clone has become a common method during software development. If two or more code fragments in a software system's codebase are exactly or nearly similar to one another, we call them code clones [Kim, Sazawal and Notkin (2005)]. A group of similar code fragments forms a clone class. Code clones are mainly created because of the frequent copy/paste activities of the programmers during software development and maintenance. Whatever may be the reasons behind cloning, code clones are of great importance from the perspectives of software maintenance and evolution [Roy, Zibran and Koschke (2014); Roy (2009)].

A large number of identical or similar code clones brought difficulties for the software maintenance. For example, if a bug is detected in a code fragment, all fragments similar to it should be checked for the same bug [Li, Lu, Myagmar et al. (2006)]. Duplicated fragments can also significantly increase the work to be done when enhancing or adapting code. Many

[1] Inner Mongolia Normal University, Hohhot, 010022, China.

[2] Graduate School of Informatics and Engineering, The University of Electro-Communications, Tokyo, 182-8585, Japan.

[*] Corresponding Author: Ye Zhai. Email: cieczy@imnu.edu.cn.

Received: 31 January 2020; Accepted: 09 May 2020.

other software engineering tasks, such as program understanding, code quality analysis (fewer clones may mean better quality code), aspect mining (clones may indicate the presence of an aspect), plagiarism detection, copyright infringement investigation, software evolution analysis, code compaction (for example, in mobile devices), virus detection and bug detection may require necessary and proper management of the cloned code. Many researchers consider code clones should be removed. However, sometimes there are dependencies relations among each of which belong to the different clones, and simple removal of the code may cause more serious errors [Yoshida, Higo, Kamiya et al. (2005)].

Existing research shows that it is impractical to refactor all clones in software system, and not all clones need to be refactored [Higo, Ueda, Kusumoto et al. (2007)]. Blindly refactoring may affect other useful codes in the software, and lead to software quality degradation. An empirical study of Kim et al. [Kim, Sazawal and Notkin (2005)] revealed two points: first one is that some clones are short-lived, and merging them wouldn't improve the maintainability; second one is that most of long-living clones are not suited to be refactored because there is no abstraction function of the programming language. Therefore, it is especially critical to identify the clones suitable for refactoring before effective maintenance of the clones.

In this paper, we provide an approach based on Bayesian to refactor clone code for software developers. The main purpose of our study is to provide valuable reference information for software development and maintenance, so as to reduce maintenance costs, improve software code quality and obtain greater economic benefits.

The rest of the paper is organized as follows. Section II contains the terminology, Section III discusses the experimental steps, Section IV analyzes the experimental results, and Section V concludes the paper by mentioning possible future work.

2 Related works

2.1 Clone code definition and classification

Since the extensive use of clones has an important impact on the quality of software products, the related research on clone has become a more active branch in the field of code analysis in recent years. A code clone is a set of source code fragments identical or similar to each other from the viewpoint of software maintainability. The types of clone can be divided from different perspectives. Currently, there are two main perspectives of classification. One is to divide code into Type-1, Type-2, Type-3 and Type-4 according to code similarity [Kim, Sazawal and Notkin (2005)], Tab. 1 shows the definition; the other is to divide code into file clone, class clone, function clone, block clone and statement clone according to detection granularity.

Table 1: Definition of clone type

Type	Description
Type-1	The exact same code fragment except the space and comment changes
Type-2	The code fragment with the same syntax structure except the blanks, comments, identifiers, and type substitutions
Type-3	Code fragments with the same syntactic structure except blanks, comments, identifiers, and type substitutions, but with added, deleted, or modified with a small number of statements
Type-4	Code fragments with the same function but different syntax structures

2.2 Clone detection

Clone detection can find clones in the source code and giving feedback in the form of a clone pair or a clone group [Kamiya, Kusumoto and Inoue (2002)]. At present, the research of clone detection has been attracting great interests, and many detection technologies have been proposed, including text-based clone detection [Johnson (1993)], Token-based clone detection [Li, Lu, Myagmar et al. (2006)], Abstract Syntax Trees (AST) detection [Koschke, Falke and Frenzel (2006)], Program Dependence Graphs (PDG) detection [Roy (2009)], low-level language-based detection [Davis and Godfrey (2010)], Metrics-based detection [Abd-El-Hafiz (2012)].

2.3 Clone refactor

The concept of refactor is proposed by Opdyke [Opdyke (1992)] to improve the quality of the code by changing the internal structure of the program code without changing the external behavior of the program code. At present, there are many studies on clone refactor. Bian [Bian (2014)] extracted a clone suitable for refactor with a simple calculation method. Bakota [Bakota (2011)] extracted clones suitable for refactoring based on evolutionary analysis, which is suitable to Type-1 and Type-2 clone. Mondal et al. [Mondal, Roy and Schneider (2015)] divided the SPCP clone into refactored data set and a tracking data set according to the location of the clone fragments. Higo et al. [Higo, Ueda, Kusumoto et al. (2007)] proposed an approach for extracting clones based on clone metrics for refactor. Meng [Meng (2014)] proposes a method based on SOM (Self Organized Mapping) clustering to seek refactorable code clone.

Liu et al. [Liu, Liu, Zhang et al. (2016)] proposed an approach to evaluate the reconfigure ability of clone code. This approach systematically evaluates the reconfigure ability of clones in current version, and ranks it from low to high according to the refactor level of clones. But the effort laid the foundation for recommending refactor clone.

2.4 Bayesian network

Bayesian network is also called belief network. Bayesian classifier is a simple and powerful classification method [Oliver, Patrick and Bruce (2009)]. It is one of the most effective theoretical models in the field of reasoning. Shen et al. [Shen, Nagai and Gao (2019)] improve computer visualization of architecture based on the Bayesian network. Burak et al. [Burak, Turhan, Ayse et al. (2009)] used Bayesian network to predict the code that needs to be refactored in Java system, and achieved certain results. The team used the decision tree to recommend refactor clones in the early stage, but the prediction effect is not very good, so this paper uses Bayesian network to recommend refactor clones.

There are two main steps in constructing or training Bayesian network:

Step 1: Determine the topological relationship between random variables to form DGA, which is to find a set of conditional probabilities. The calculation formula is as follows：

$$p(x_1, x_2, \ldots, x_n) = p(x_1)p(x_2|x_1)p(x_3|x_1, x_2)\ldots p(x_n|x_1, x_2, \ldots, x_{n-1}) \quad (1)$$

Step 2: The Bayesian network is trained to complete the construction of conditional probability table. The calculation formula is as follows:

$$p(x_1, x_2, \ldots, x_n) = \prod_{i=1}^{n} p(x_i|Parents(x_i)) \quad (2)$$

where Parents represent the union of direct precursor nodes of x_i.

3 Recommend clone based on Bayesian network

This paper proposes an approach for recommending clone code, which is divided into five steps, as shown in Fig. 1.

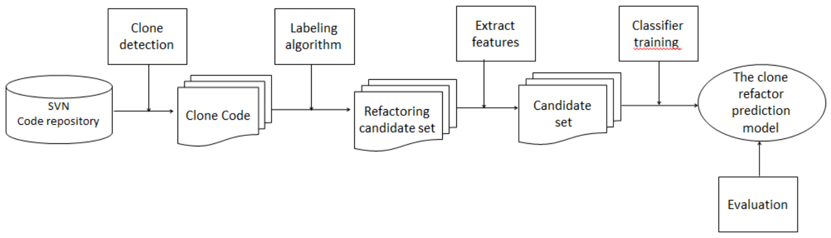
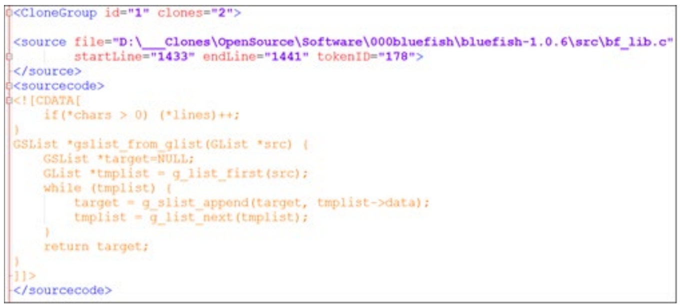


Figure 1: Overall analysis process

3.1 Clone detection

We use Nicad [Roy and Cordy (2008)] for clone detection, which can detect Type-1, Type-2 and Type-3 clones more accurately. This tool takes multiple versions of software as input to get clone group information, which includes file name, function name, start line, end line and other information as well as clone fragment related information. The data are stored in the form of Extensible Markup Language (XML) format to facilitate the extraction and carry out follow-up research. The Fig. 2 shows the details.



```
<CloneGroup id="1" clones="2">

<source file="D:\___Clones\OpenSource\Software\000bluefish\bluefish-1.0.6\src\bf_lib.c"
        startLine="1433" endLine="1441" tokenID="178">
</source>
<sourcecode>
<![CDATA[
    if(*chars > 0) (*lines)++;
}
GSList *gslist_from_glist(GList *src) {
    GSList *target=NULL;
    GList *tmplist = g_list_first(src);
    while (tmplist) {
        target = g_slist_append(target, tmplist->data);
        tmplist = g_list_next(tmplist);
    }
    return target;
}
]]>
</sourcecode>
```

Figure 2: Clone code detection results

3.2 Identify the refactor clone code

In this paper, the specific process of identifying refactor instances of clone is divided into the following three steps:

Step 1. Refactor code instances are defined based on adjacent version keys. If any two clone fragments are refactored, they are refactored instance.

Step 2. Generate refactoring candidate set. In this experiment, five famous open source systems are selected and all public versions of these systems are collected. To identify clone refactoring instances in the selected target system, we use the measurement tool Moose\Metrics [Demeyer, Ducasse and Nierstrasz (2005)] to extract classes, methods, attributes and calculate necessary metrics from the source code to produce refactoring candidate sets.

Step 3. Optimize refactoring candidate set.

After generating the refactoring candidate set, it will be further filtered and optimized according to the following three conditions:

1. Two siblings of the clone class between adjacent versions disappear
2. For methods within the clone fragment, the number of source code lines is reduced, and at least one method is added only in this method
3. A new source class in the raw clone is created and used as a parent class.

These three conditions are selected because they contain all the refactoring patterns associated with clone refactor. By manually checking clone refactor candidates with this metric-based approach, Demeyer et al. [Demeyer, Ducasse and Nierstrasz (2005)] found that this method can detect all fowler refactor modes suitable for clone refactor, including Extract Method, Extract Superclass, Pull-Up Method, Replace Method with Method Object, Template Method et al. If any of these conditions is met, then a clone class C is considered as a candidate.

3.3 Extract feature

Feature extraction is a common data preprocessing method in the field of machine learning and pattern recognition. It is a necessary step to train machine learning models. Through the study of two large industrial software, Wang et al. [Wang, Dang and Zhang (2012)] proposed that the harmfulness of using clones could be related to the characteristics of clone fragment and the characteristics of content. Steidl et al. [Steidl and Gode (2013)] used the characteristics of clone fragment and clone relationships to automatically identify the bug fixes of clones and verified them. These features are related to the harmfulness of the clone, and can also be an important basis for recommending refactor.

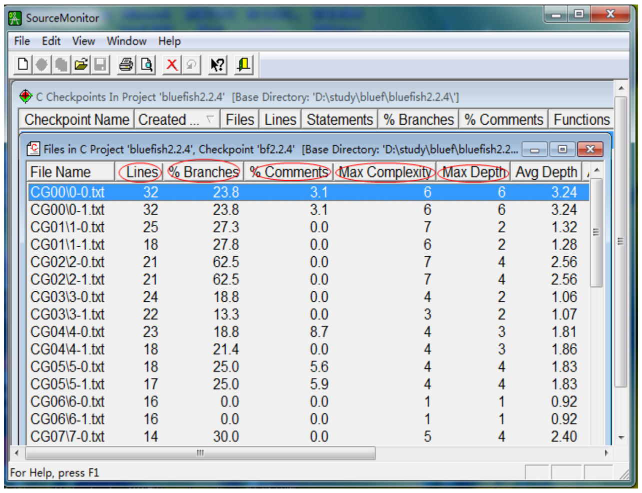
Clone code is not static, it is constantly evolving, and it is not enough to reflect the refactor of clones from a single version of the feature. Therefore, this paper reflects the possibility of refactor clones from the four dimensions of clone relationship, clone context, clone fragment and clone evolution. The clone evolution includes clone life, clone evolution frequency, and clone evolution mode. These three characteristics reflect the impact of changes in the evolution of the clone on the quality of the software, which will help to analyze the possibility that the clones need to be refactored. Detailed characteristics information is shown in Tab. 2.

Table 2: Detailed feature description

Feature type	Feature
Clone relationship	1. Number of clone fragments in the clone class
	2. Edit distance between clone method names
	3. Whether the clone group is a clone of Type-3
	4. Whether the clone fragment is in the same file, or an adjacent file
Clone context	5. Whether it is a clone fragment following the flow control statement
	6. The life cycle of the original file containing the clone fragment
	7. The number of lines of code that contain the method of the clone code fragment
	8. Size of clone code fragment out of size of a method
Clone fragment	9. Number of lines of the clone fragment
	10. The size of the token of the clone fragment
	11. Whether a clone fragment contains a complete control block
	12. The complexity of clone code
	13. The percentage of method call statements in the clone fragment
	14. The percentage of the calculated statement in the clone fragment
	15. Whether the clone fragment starts with a control flow statement
Clone evolution	16. Life of clone
	17. Clone evolution frequency
	18. Clone evolution model

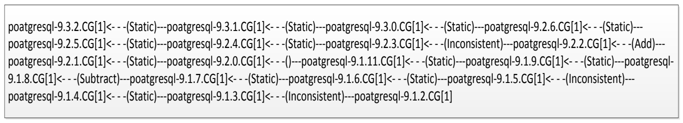
Extract feature is based on clone detection and refactor annotation. Clone detection result is the basic clone data in this study, including the number and location of the clone. Whether the clone group is Type-3 or the token size of the clone fragment is directly extracted from the clone detection result. We use the code tool SourceMonitor to extract other features. SourceMonitor is a tool that measures code written in multiple languages (C++, C, C#, VB.net, Java, Visual, Basic, and HTML) and outputs different code metrics for different languages. Taking the clone code as input, SourceMonitor will output the static characteristics value of the corresponding clone code. Fig. 3 shows the results of SourceMonitor's measurement of partial clone code. The <Lines>, <Max Complexity> and <%Comments> tag of the SourceMonitor tool indicated code lines, cyclomatic complexity and percentage of comments. But you need to manually extract for feature 11 (whether a clone fragment contains a complete control block).

The evolutionary features extracted in this paper are three categories: clone life, frequency of change, and evolution mode. The extraction method is mainly extracted by the clone function extractor FCGE, which is developed by our team. The extraction results are shown in Fig. 4.


SourceMonitor
File Edit View Window Help
C Checkpoints In Project 'bluefish2.2.4' [Base Directory: 'D:\study\bluef\bluefish2.2.4\']
Checkpoint Name | Created ... | Files | Lines | Statements | % Branches | % Comments | Functions
Files in C Project 'bluefish2.2.4', Checkpoint 'bf2.2.4' [Base Directory: 'D:\study\bluef\bluefish2.2...
For Help, press F1

File Name	Lines	% Branches	% Comments	Max Complexity	Max Depth	Avg Depth
CG00\0-0.txt	32	23.8	3.1	6	6	3.24
CG00\0-1.txt	32	23.8	3.1	6	6	3.24
CG01\1-0.txt	25	27.3	0.0	7	2	1.32
CG01\1-1.txt	18	27.8	0.0	6	2	1.28
CG02\2-0.txt	21	62.5	0.0	7	4	2.56
CG02\2-1.txt	21	62.5	0.0	7	4	2.56
CG03\3-0.txt	24	18.8	0.0	4	2	1.06
CG03\3-1.txt	22	13.3	0.0	3	2	1.07
CG04\4-0.txt	23	18.8	8.7	4	3	1.81
CG04\4-1.txt	18	21.4	0.0	4	3	1.86
CG05\5-0.txt	18	25.0	5.6	4	4	1.83
CG05\5-1.txt	17	25.0	5.9	4	4	1.83
CG06\6-0.txt	16	0.0	0.0	1	1	0.92
CG06\6-1.txt	16	0.0	0.0	1	1	0.92
CG07\7-0.txt	14	30.0	0.0	5	4	2.40

Figure 3: Partial clone code measurement results



```
poatgresql-9.3.2.CG[1]<- - -(Static)---poatgresql-9.3.1.CG[1]<- - -(Static)---poatgresql-9.3.0.CG[1]<- - -(Static)---poatgresql-9.2.6.CG[1]<- - -(Static)---
poatgresql-9.2.5.CG[1]<- - -(Static)---poatgresql-9.2.4.CG[1]<- - -(Static)---poatgresql-9.2.3.CG[1]<- - -(Inconsistent)---poatgresql-9.2.2.CG[1]<- - -(Add)---
poatgresql-9.2.1.CG[1]<- - -(Static)---poatgresql-9.2.0.CG[1]<- - -()---poatgresql-9.1.11.CG[1]<- - -(Static)---poatgresql-9.1.9.CG[1]<- - -(Static)---poatgresql-
9.1.8.CG[1]<- - -(Subtract)---poatgresql-9.1.7.CG[1]<- - -(Static)---poatgresql-9.1.6.CG[1]<- - -(Static)---poatgresql-9.1.5.CG[1]<- - -(Inconsistent)---
poatgresql-9.1.4.CG[1]<- - -(Static)---poatgresql-9.1.3.CG[1]<- - -(Inconsistent)---poatgresql-9.1.2.CG[1]
```

Figure 4: Clone evolution feature extraction results

3.4 Training classifier

After extracting features and completing the construction of feature datasets, we select the machine learning model of Bayesian network to recommend clone refactor code. Dependency analysis and search-based scoring methods are the two main types of methods commonly used in Bayesian networks. In this study, a score-based search method is used, which can search the exact network structure. Because of the large architectural space, heuristic algorithms are needed to search for the best Bayesian network architecture. We use a heuristic algorithm to represent the K2 algorithm. The pseudo code of the K2 algorithm is shown in Tab. 3, where μ represents the upper bound of the number of variable nodes and ν represents a complete set of data.

Table 3: K2 algorithm

```
input: X={X1, X2, …..Xn}
output: Bayesian network
1.  £← Borderless graph consisting of nodes X1, X2, ..., Xn
2. for j=1 to n
3.  $\pi \leftarrow \Phi$
4. $V_{old} \leftarrow CH(<X_j,\pi_j>|\nu)$
5. while(true)
6. $i \leftarrow \mathrm{argmax}_{1\le i\le j}, Xi \in \pi j$ CH（$<X_j,\pi_j \cup \{X_i\}>|\nu$）
7. $V_{new} \leftarrow CH(<X_j, \pi_j \cup \{X_i\}>|\nu)$
8.  if ($V_{new}>V_{old}$ and $|\pi_j|<\mu$)
9. $V_{old} \leftarrow V_{new}$
10. $\pi_j \leftarrow \pi_i \cup \{X_i\}$
11.  Adding edges in $\nu$ to $X_i \rightarrow X_j$
12. else
13. break
14. end if
15. end while
16. end for
17. estimated £ parameter $\theta$
18. return (£, $\theta$)
```

3.5 Evaluation of classification results

When predicting unknown data samples, some are correctly classified and some are misclassified. Therefore, in order to evaluate the performance of the clone refactor prediction model, we use the recall rate, precision, and F commonly as evaluation indicators. The mixing matrix corresponding to the values and the like is as shown in Tab. 4.

Table 4: Mixed matrix

Forecast result	**Positive class**	**Anti-class**
Positive class	True Positive (TP)	False Negative (FN)
Anti-class	False Positive (FP)	True Negative (TN)

The definitions of precision, recall and F are as follows:

$$Precision=\frac{TP}{TP+FP} \tag{3}$$

$$\mathrm{Re}call=\frac{TP}{TP+FN} \tag{4}$$

$$F=\frac{2*\mathrm{Pr}ecision*\mathrm{Re}call}{\mathrm{Pr}ecision+\mathrm{Re}call} \tag{5}$$

Among them, recall measure classifiers correctly predict the proportion of clone refactor, and recall is a common index in classification problems, which reflects the overall classification performance of classifiers to data sets.

4 Experiment and analysis

4.1 Experimental data selection

The five open source software are selected in this experiment, FFmpeg, Smalltalk, Claws-mail, Fdisk and Lighttpd. Tab. 5 shows the basic information.

Table 5: Basic information of the experimental software

Software	language	Features	Lines of code
FFmpeg	C	Multimedia tool software	543560
Smalltalk	C	Program integration development environment	3078858
Claws-mail	C	Mail client	220235
Fdisk	C	Disk management tool	187958
Lighttpd	C	Web server	53378

4.2 Experimental results

In this paper, different feature subsets are selected and each subject is used to train a new classifier. This experiment selects the characteristics of four subsets. These subsets are clone relationship, clone context, clone fragment and clone evolution. The experimental results are shown in Tabs. 6 and 7.

Table 6: Results of classification training for clone refactor instances

Software	All features		Static feature						Evolution information	
			Clone relationship		Clone fragment		Clone context			
	Precision	recall	Precision	Recall	precision	recall	precision	Recall	precision	recall
FFmpeg	0.925	0.920	0.715	0.804	0.920	0.913	0.865	0.856	0.801	0.795
Smalltalk	0.905	0.897	0.685	0.640	0.895	0.885	0.842	0.815	0.812	0.734
Claws-mail	0.935	0.918	0.729	0.801	0.928	0.909	0.870	0.819	0.785	0.834
Fdisk	0.928	0.925	0.699	0.621	0.920	0.918	0.867	0.899	0.821	0.835
Lighttpd	0.945	0.930	0.805	0.563	0.930	0.908	0.898	0.889	0.818	0.858

From these two tables, the following conclusions can be drawn:

- Classifiers without any type of feature training in the five experiment softwares are always superior to classifiers trained by other feature types.

Table 7: Results of classification training for non-refactored instances

Software	**All features**		**Static feature**						**Evolution information**	
			Clone relationship		**Clone fragment**		**Clone context**			
	Precision	**recall**	**Precision**	**Recall**	**precision**	**recall**	**precision**	**Recall**	**precision**	**recall**
FFmpeg	0.930	0.925	0.625	0.701	0.927	0.913	0.875	0.863	0.873	0.855
Smalltalk	0.912	0.905	0.798	0.686	0.910	0.905	0.875	0.889	0.862	0.878
Claws-mail	0.928	0.913	0.798	0.585	0..920	0.907	0.901	0.894	0.896	0.883
Fdisk	0.917	0.914	0.805	0.425	0.911	0.905	0.885	0.895	0.881	0.890
Lighttpd	0.945	0.925	0.585	0.861	0.941	0.919	0.909	0.905	0.901	0.884

- Compared with the classifier trained by all feature types, the result of the classifier trained by clone fragment type features is the most similar.
- The classifier of the clone context feature type and the classifier training of the evolution feature type training are slightly lower than the classifiers trained by all feature types.
- The classifier of clone relationship feature type training, which makes the performance of classifier get a small part of improvement.

Based on the above conclusions, it is concluded that the characteristics of the clone fragment are independent of the clone relationship, clone content and evolution information. The generated result is used to recommend the refactored clone.

In this experiment, K-fold cross-validation is selected to evaluate the verification. The basic idea of this method is to divide the input data set into training set and test set. Because the test set and the training set are invisible to the classifier, the object of cross-validation is performed. The object is the result of the training set output. The experimental steps are as follows:

Step 1. Divide the data set D into k mutually similar reciprocal subsets, namely D=D_1 and D_2 and D_3... and D_k, and there is no intersection between each subset.

Step 2. Use the union of k-1 subsets as the training set each time, and the remaining one as the test set, thus obtaining the k group training/test set.

Step 3. Perform k training and testing, and finally return the mean of the k results.

Step 4. Use different partitions multiple times at random.

This experiment uses K-fold cross-validation to evaluate the classifier prediction model, where K=10. We found that the error obtained by 10% is the smallest after many experiments on large data sets and using different technologies, so the experiment divides the data set into 10 parts. The verification results are shown in Tabs. 8 and 9. The accuracy, recall, and F metric of the refactored and non-refactored instances are shown in the two tables.

Table 8: Results of the refactor group test

Software	Precision	Recall	*F*-Measure
FFmpeg	0.885	0.897	0.901
Smalltalk	0.921	0.935	0.938
Claws-mail	0.912	0.919	0.921
Fdisk	0.901	0.915	0.920
Lighttpd	0.899	0.913	0.918

Table 9: Results of the non-refactor group test

Software	Precision	Recall	*F*-Measure
FFmpeg	0.879	0.902	0.911
Smalltalk	0.895	0.899	0.902
Claws-mail	0.918	0.920	0.923
Fdisk	0.901	0.909	0.913
Lighttpd	0.928	0.930	0.931

From Tab. 8, it can be concluded that the accuracy of the ten-fold cross-validation of refactor example has increased from 88.5% to 92.1%, which is 3.6 percentage points higher. The recall rate increased from 89.7% to 93.5%, up 3.8 percentage points. The F value increased from 90.1 to 93.8%, 3.2 percentage points higher. From Tab. 9, it can be concluded that the accuracy of the ten-fold cross-validation of non-refactored instance increased from 87.9% to 92.8%, growth rate is 4.9 percentage points. The recall rate increased from 89.9% to 93.0%, up 3.1 percentage points. The *F*-value increased from 90.2% to 93.1%, 2.9 percentage points higher. Therefore, it is concluded that the classifier constructed in this study is better than the random selection.

4.3 Comparative analysis of similar experiments

At present, there are few researches on using machine learning methods to recommend and predict the refactor clone code. Wang et al. [Wang and Godfrey (2014)] proposed a representative method use machine learning methods to predict the clones to be refactored. They used Iclones detection and propose an automated approach to recommend clones for refactoring by training a decision tree-based classifier. In this study, Nicad is used to detect the clone code. At the same time, we extracted the features are not completely consistent with Wang's. The project studied in this paper is the C project and Wang used Java.

Liu et al. [Liu, Liu, Zhang et al. (2016)] of the team used the Bayesian network to predict the clones to be refactored and evaluated it using the quality model EMISQ. The clone detection is Fclones, and its feature selection is based on ISO software quality standards. The characteristics of this paper are extracted from the static and evolution categories, so only the recommended clone refactor is compared. The projects used in this paper and Liu et al. [Liu, Liu, Zhang et al. (2016)] are all C projects, and all used Bayesian network model for prediction. The experimental platform is the same operating system: Ubuntu14.04 64-bit, memory: 8 GB, CPU:2 cores.

In order to prove the effectiveness of the experimental results, we chose three source software for comparison experiments, which are same as used in Liu et al. [Liu, Liu, Zhang et al. (2016)]. The version information is shown in Tab. 10.

Table 10: Recommended software information for cloning refactored experiments

Software	Start version	End version	Commits	Refactoring instance
Xorriso	1.4.15	1.4.39	53378	308
Smalltalk	2.0.11	3.2.5	3078858	381
Bison	1.9.100	3.9.3	220235	289

Through the comparative experiment, the results are shown in Tab. 11. Compared with the EMISQ method, the precision, recall and *f*-value of Xoriso are increased by 9%, 11.4% and 8.4% respectively. Compared with the EMISQ method, the precision of the proposed clone refactor software Smalltalk is increased by 9.1%, 7% and 5.2% respectively. Compared with the EMISQ method, the precision of the software, it is increased by 11.7%, 7% and 5.9% respectively. In conclusion, the recall, precision and *F*-value of clone refactor method are higher than those of EMISQ method.

Table 11: Recommended cloning and refactor of similar experimental results

Software	Approach	Precision	recall	*F*
Xorriso	EMISQ	0.926	0.882	0.904
	Method of this paper	0.975	0.996	0.988
Smalltalk	EMISQ	0.889	0.990	0.934
	Method of this paper	0.980	0.997	0.986
Bison	EMISQ	0.865	0.991	0.927
	Method of this paper	0.972	0.998	0.986

5 Conclusions and future work

The difficulty of software maintenance and the complexity of the project can be reduced by refactoring. This paper proposes an approach to recommend refactoring clone based on Bayesian network. The clones in the project are detected by clone detection tool, the clone that needs to be refactored are found in three steps and then are labeled. At the same time, the refactoring tool SourceMonitor and FCGE are used to extract features and build sample data set. Finally, more than 640 clone instances are tested by using ten-fold cross-validation. It is found that the accuracy, recall rate and *F*-value of the proposed classifier based on Bayesian network can reach more than 90%, which proves the effectiveness of this approach. The findings from our study are important for better management of code clones as well as for better maintenance of software systems.

There are still some shortcomings in the research content and experiment of this paper, for example, we can only recommend the refactoring clones in the C-language development project at present. In future research, we will continue to improve this work. We are targeting at recommending not only C-language development projects, but also Java and python projects.

Funding Statement: This work was supported by the National Natural Science Foundation (61363017) of China. The author is Liu, D. S. and the website is https://isisn.nsfc.gov.cn.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

Abd-El-Hafiz, S. K. (2012): A metrics-based data mining approach for software clone detection. *International Computer Software and Applications Conference*, pp. 35-41.

Bakota, T. (2011): Tracking the evolution of code clones. *SOFSEM: Theory and Practice of Computer Science*, vol. 6543, pp. 86-98.

Bian, Y. X. (2014): *Research on Process Extraction Method of Reconfigurable Clone code (Ph.D. Thesis)*. Harbin Institute of Technology, Harbin.

Cordy, J. R.; Roy, C. K. (2011): Tuning research tools for scalability and performance: The NiCad experience, *Science of Computer Programming*, pp.158-171.

Davis, I. J.; Godfrey, M. W. (2010): From whence it came: detecting source code clones by analyzing assembler. *Proceedings of 17th Working Conference on Reverse Engineering*, pp. 242-246.

Demeyer, S.; Ducasse, S.; Nierstrasz, O. (2005): Object-oriented reengineering: patterns and techniques. *21st IEEE International Conference on Software Maintenance*, pp. 723-724.

Higo, Y.; Ueda, Y.; Kusumoto, S.; Inoue, K. (2007): Simultaneous modification support based on code clone analysis. *14th Asia-Pacific Software Engineering Conference*, pp. 262-269.

Johnson, J. H. (1993): Identifying redundancy in source code using fingerprints. *Proceedings of the Conference of the Centre for Advanced Studies on Collaborative Research: Software Engineering*, vol. 1, pp. 171-183.

Kamiya, T.; Kusumoto, S.; Inoue, K. (2002): CCFinder: a multilinguistic token-based code clone detection system for large scale source code. *IEEE Transactions on Software Engineering*, vol. 28, no. 7, pp. 654-670.

Kim, M.; Sazawal, V.; Notkin, D. (2005): An empirical study of code clone genealogies. *13th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pp. 187-196.

Koschke, R.; Falke, R.; Frenzel, P. (2006): Clone detection using abstract syntax trees. *Proceedings of 13th Working Conference on Reverse Engineering*, pp. 253-262.

Li, Z.; Lu, S.; Myagmar, S.; Zhou, Y. (2006): CP-miner: finding copy-paste and related bugs in large-scale software code. *IEEE Transactions on Software Engineering*, vol. 32, no. 3, pp. 176-192.

Liu, D.; Liu, D.; Zhang, L.; Hou, M.; Wang, C. (2016): The prediction of code clone quality based on Bayesian network. *International Journal of Software Engineering and its Applications*, vol. 10, no. 4, pp. 47-56.

Meng, F. (2014): Using self-organized mapping to seek refactorable code clone. *Fourth International Conference on Communication Systems and Network Technologies*, pp. 851-855.

Mondal, M.; Roy, C. K.; Schneider, K. A. (2015): SPCP-Miner: a tool for mining code clones that are important for refactor or tracking. *IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineerin*, pp. 484-488.

Oliver, P.; Patrick, N.; Bruce, M. (2009): *Bayesian Networks: A Practical Guide to Applications*, John Wiley and Sons, Ltd.

Opdyke, W. F. (1992): *Refactor Object Frame Works.* Illinois: University of Illinois at Urban-Champaign.

Roy, C. K. (2009): Detection and analysis of near-miss software clones. *25th IEEE International Conference on Software Maintenance*, pp. 447-450.

Roy, C. K.; Zibran, M. F.; Koschke, R. (2014): The vision of software clone management: past, present, and future. *Software Evolution Week-IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering*, pp. 18-33.

Roy, C. K.; Cordy, J. R. (2008): NICAD: Accurate detection of near-miss intentional clones using flexible pretty-printing and code normalization. *16th IEEE International Conference on Program Comprehension*, pp. 172-181.

Shen, T.; Nagai, Y. K.; Gao, C. (2019): Improve computer visualization of architecture based on the Bayesian network. *Computers, Materials & Continua*, vol. 58, no. 2, pp. 307-318.

Steidl, D.; Gode, N. (2013): Feature-based detection of bugs in clones. *7th International Workshop on Software Clones*, pp. 76-82.

Turhan, B.; Bener, A. (2009): Analysis of naive Bayes' assumptions on software fault data: an empirical study. *Data & Knowledge Engineering*, vol. 68, no. 2, pp. 278-290.

Wang, W.; Godfrey, M. W. (2014): Recommending clones for refactoring using design, context, and history. *IEEE International Conference on Software Maintenance and Evolution*, pp. 331-340.

Wang, X. Y.; Dang, Y. N.; Zhang, L. (2012): Can I clone this piece of code here? *Proceedings of the IEEE: ACM International Conference on Automated Software Engineering*, pp. 170-179.

Yoshida, N.; Higo, Y.; Kamiya, T.; Kusumoto, S.; Inoue, K. (2005): On refactoring support based on code clone dependency relation. *11th IEEE International Software Metrics Symposium*, pp. 10-16.