Clustering OSS License Statements Toward Automatic Generation of License Rules

Yunosuke Higashi
Graduate School of System Engineering, Wakayama University
Email: s151039@sys.wakayama-u.ac.jp

Yuki Manabe
Graduate School of Science and Technology, Kumamoto University
Email: y-manabe@cs.kumamoto-u.ac.jp

Masao Ohira
Graduate School of System Engineering, Wakayama University
Email: masao@sys.wakayama-uc.ac.jp

***Abstract*—Reusing open source software (OSS) components for own software products has become common in the modern software development. Automated license identification tools has been proposed to help developers identify OSS licenses, since a large number of licenses sometimes must be checked to be reused. Of the existing tools, Ninka [1] can most correctly identify licenses of each source file by using regular expressions. In case Ninka does not have license identification rules for unknown licenses, Ninka reports they are "unknown licenses" which must be checked by developers manually. Since completely-new or derived OSS licenses appear nearly every year, a license identification tool should be appropriately maintained by adding regular expressions corresponding to the new licenses. The final goal of our study is to construct a method to automatically create candidates of license rules to be added to a license identification tool such as Ninka. Toward achieving the goal, files identified as unknown licenses must be classified by license firstly. In this paper, we propose a hierarchical clustering which divides unknown licenses into clusters of files with a single license. We conduct a case study to confirm the usefulness of our clustering method when it is applied for classifying 2,838 unknown license files of Debian v7.8.0. As a result, it is confirmed that our method can create clusters which are suitable as candidates for generating license rules automatically.**

I. Introduction

Utilizing Open Source Software (OSS) is a means of reducing production costs in the modern software development. OSS is also reusable as part of a proprietary software product if it strictly complies with OSS licenses described in source files [1]. In general, an OSS license is declared in a header part of each source file as license statements. Before reusing OSS for own products, all license statements must be confirmed to avoid inappropriate, illegal reuse. Identifying OSS licenses manually is a time-consuming task if there are a large number of source files to be reused. In order to help identify OSS licenses, license identification tools [1]–[5] have been proposed.

Of the existing tools, Ninka [1] and FOSSology [2] which are rule-based license identification tools correctly identifies OSS licenses. Rule-based license identification tools can discriminate between known and unknown licenses by using regular expressions to identify OSS licenses, while the other existing tools do not have such a mechanism and often make a misjudgment that leads to lower accuracy for the license identification. However, the biggest weakness of rule-based license identification tools such as Ninka is that it will demand to manually and constantly add new regular expressions to the tools, every time when the tools encounter new OSS licenses (i.e., the licenses is really unknown to the tools) or the tools unexpectedly judges some licenses as unknown due to notation variants in license statements. In case there are "unknown" licenses for rule-based license identification tools, the manual identification of OSS licenses is required eventually.

The goal of this study is to construct a method to automatically generate candidates of license rules to be incorporated into rule-based license identification tools in order to address the issue above. To achieve the goal, we are developing a method which consists of the following three steps for creating license rules.

1) **Grouping source files with unknown licenses: Source files which are not identified by the license identification tools are reviewed and grouped by license.**
2) Checking notation variants for a single license: Each group of source files with a single license is checked to extract expression patterns for a single license.
3) Creating license rules: License statements are tokenized as regular expressions and license rules are created to be able to match new licenses.

In this paper, we focus on the automation of Step 1. To automate Step 1, we introduce a method using grep and hierarchical clustering. Grep is used to extract GPL/BSD-related licenses which are known by a rule-based license identification tool but are not completely identified due to notation variants After filtering out source files with GPL/BSD-related licenses using grep, a hierarchical clustering is used to group the rest of "really-unknown" license files by license. Modifying the common hierarchical clustering, our clustering method tries to divides a set of files with unknown licenses into clusters of files with a single license.

A case study is conducted to confirm the usefulness of using grep and our clustering method when they are applied for classifying 2,838 unknown license files of Debian v7.8.0. As a result, it is confirmed that grep can extract 89% of files with GPL/BSD-related licenses and the clustering method can create clusters which are suitable as candidates for generating license rules automatically.

The rest of the paper is organized as follows. Section II discusses the problems in manually creating license rules for rule-based license identification tools and technical challenges

978-1-5090-1851-2/16 $31.00 © 2016 IEEE
DOI 10.1109/IWESEP.2016.20

in this paper. Section III describes the usage of grep and our clustering method. Section IV describes a case study where software packages of Debian v7.8.0 is used to evaluate our method. Section V discusses results of the case study. Section VI introduces related work and Section VII concludes the paper and describes our future work.

II. Toward Automatic Generation of License Rules

This section describes current problems in using a license identification tool and technical challenges in this paper.

A. Current Problems in Using a License Identification Tool

Automatic license identification tools such as Ninka [1] and FOSSology [2] have matching rules to check and determine if license statements declared in the header part of each source file are known or unknown. In order to do so, each statement in OSS licenses is manually analyzed and tokenized as regular expressions in advance. However, completely-new or derived OSS licenses appear nearly every year. If the existing tools do not have matching rules for newly emerging licenses, license rules must be manually added to the tools again thorough manually creating matching rules.

The process of creating license matching rules follows the steps 1 to 3 below.

1) **Grouping source files with unknown licenses:** Source files which are not identified by the license identification tools must be reviewed and grouped by license manually.
2) **Checking notation variants for a single license:** Even if the source files are grouped by license, there often exist notation variants (e.g., misspelling, small modification and different expressions for the original license) in license statements for a single license. Each group of source files with a single license must be reviewed manually again to know expression patterns for a single license.
3) **Creating license rules:** Based on the reviews in Step 2, license statements are tokenized as regular expressions and license rules are created to be able to match new licenses. Note that a "new" license connotes a license with different variations in license statements even for the same license.

These tasks are time-consuming especially when the rule-based license identification tools cannot determine licenses for a large number of source files. Therefor it is desired to automate the process of creating license rules.

B. Technical Challenges

The final goal of our study is to construct a method to automatically generate candidates of license rules to be incorporated into rule-based license identification tools in order to address the issue above. In this paper, we try to tackle with the issue in Step 1 where all source files with unknown licenses are reviewed and grouped by license manually. Automating Step 1 requires to address at least two technical challenges as follows.

TABLE I
GPL/BSD-related licenses

license family	licenses
GPL	AGPLv3(+) GPLv1(+), GPLv2(+), GPLv3(+) LGPLv2.1(+), LGPLv3(+) LibraryGPLv2.0(+)
BSD	BSD2, BSD3, BSD4

1) Discriminating GPL/BSD-related licenses: In our pilot study where Ninka was used to detect licenses of source files in Debian v.7.8.0, we found that Ninka judged many source files as "unknown" licenses even for very popular licenses such as GPLv2, although Ninka had regular expressions to detect such the licenses. As a result of analysis of the cause, it was because regular expressions used in Ninka were rigidly implemented not to misjudge a license and our dataset had license statements with misspelling and/or notation variances that were judged as "unknown" by Ninka.

In this paper, we try to correctly discriminate GPL/BSD-related licenses shown in Table I, since GPL/BSD-related licenses have a long history (the first version of GPL and BSD were released in the late 1980s) and are widely used in Linux and FreeBSD distributions. In Table I, "(+)" indicates an option to specify the existence of an "or later" clause. For instance, license statements for AGPLv3 can be expressed in two ways (i.e., "AGPLv3" and "AGPLv3 or later"). The GPL/BSD-related licenses could be distinguishable not by using rigid combinations of regular expressions but by finding license names and versions including "or later" options in license statements.

2) Creating groups of source files with unknown licenses: To automate the manual review and grouping in Step 1 described earlier, it is ideal to create groups of source files only with a single license. That is, the number of groups should correspond to the number of licenses existed in software components being reused. And, toward Step 2 and Step 3, groups should not be divided due to notation variations for a single license. If groups are divided by different notations for a single license, groups consisting of files with different license statements will be created and/or files with a single license will spread among different groups. This will lead to extra reviews of license statements in Step 2 and incorrect regular expressions in Step 3.

III. A clustering method toward the automated license rule generation

In this section, we introduce a method to automate Step 1 discussed in the previous section, which is consisted of using grep and hierarchical clustering to create clusters which would be appropriate as candidates for automating the license rule creation.

A. An overview of the clustering method

Figure 1 shows an overview of the proposed method in this paper. According to the following procedure, the method

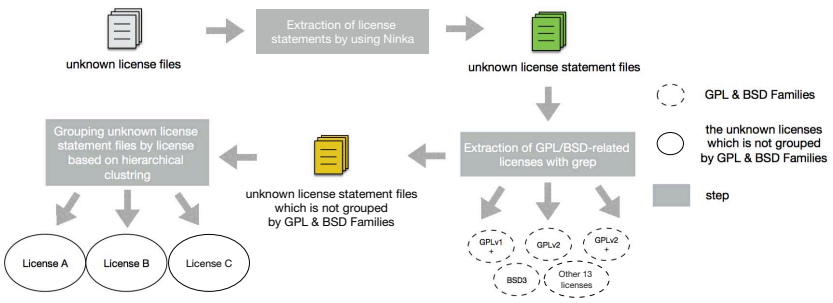

Fig. 1. An overview of the proposed method

creates clusters of unknown license files which are reported by rule-based license identification tools.

1) License statements are extracted from unknown license files by using a rule-based license identification tool such as Ninka.
2) The extracted license statements are separately grouped so that similar but discriminable licenses such as GPL version 1, 2 and 3 can be correctly identified. In this paper, license statements belonging to the GPL license family and the BSD license family are grouped by each license.
3) The rest of unknown license files which is not grouped in the previous step are divided into clusters of unknown license files, based on a dendrogram created by hierarchical clustering.

In what follows, we describe the above procedure in detail.

B. The clustering method

1) Extracting license statements with Ninka: Ninka (as of version 1.1) can extract license statements with one or more of frequently-used 177 words (e.g., "warranties" and "copyright") from comments in source code. Using Ninka, license statements in a set of unknown license files are extracted as a set of unknown license statements files.

2) Using grep to discriminate GPL/BSD-related licenses from others: Using grep, the seventeen GPL/BSD-related licenses shown in Table I are grouped by license. For the GPL license family, the discriminant conditions in Table II are used to *know* licenses. Using a combination of a key phrase and a version name, one of the GPL-related license is detected by grep. For the BSD license family, the discriminant conditions and the flow conditions in Figure 2 are used. These conditions above are created based on examples of license statements which are provided from Open Source Initiative (OSI)[1].

In case there exist license statements which meet multi-licensing for GPL and BSD, it could be determined as both one of the GPL licenses and one of the BSD licenses. Therefore, in this step, such unknown license statements are treated later in the entire process to newly create license rules for them.

[1]http://opensource.org

TABLE II
DISCRIMINANT CONDITIONS FOR GPL LICENSE FAMILY

GPL Family	Keyphrase	versions
GPLv1	GNU General Public License	version 1
GPLv2	GNU General Public License	version 2
GPLv3	GNU General Public License	version 3
LibraryGPLv2	GNU Library General Public License	version 2
LGPLv2.1	GNU Lesser General Public License	version 2.1
LGPLv3	GNU Lesser General Public License	version 3
AGPLv3	GNU Affero General Public License	version 3

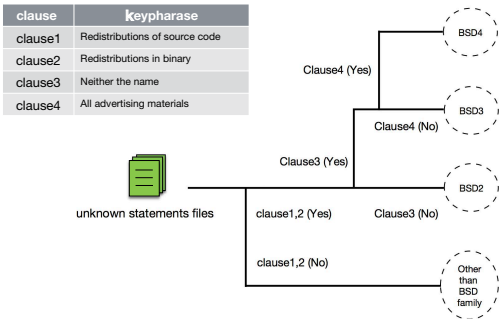

Fig. 2. Discriminant conditions and flow for BSD license family

C. Grouping unknown license statements files by license

A dendrogram is created by Ward's method [6] from the unknown license statements files which are not grouped with grep in the previous step. Before creating a dendrogram, the unknown license statements files are converted into Bag of Words vectors. Using the obtained dendrogram, the unknown license statements files are divided into clusters. Here, it is ideal to contain the same type of license statements files in each cluster.

In the general usage of a dendrogram, clusters are obtained by only once cutting the tree at any height. In our study, cutting the tree at a lower place would yield large clusters which would contain different kinds of license statements, while cutting the tree at a higher place would yield many small clusters that means the same type of license statements would be scattered across different clusters.

Since in this study the same type of license statements should become a single cluster, we introduce two kinds of

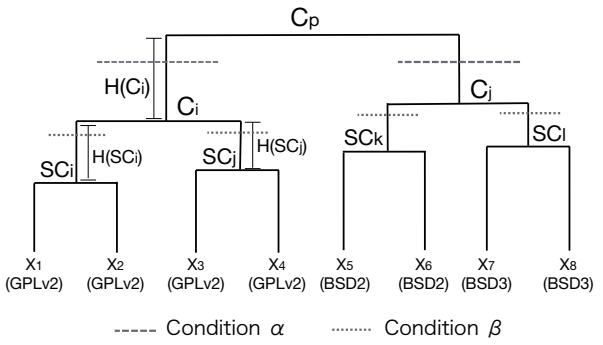

Fig. 3. Hierarchical clustering used in this study

conditions for cutting the tree in a dendrogram multiple times. Suppose a dendrogram in Figure 3 is obtained. In the figure, X_x represents one data (i.e., license statements in a source file) and C_x represents one cluster which is created by merging a pair of two data (X_x) or a pair of two clusters. Based on Ward's method, smallest clusters such as SC_i and SC_j are created using the dissimilarity (i.e., the distance in the bag-of-words vector space) between two data. In the same manner, clusters such as C_i and C_j are created using the dissimilarity between two clusters or between one cluster and one data. The dissimilarity ($d(C_i, C_j)$) between C_i and C_j is formulated as

$$d(C_i, C_j) = E(C_i \cup C_j) - E(C_i) - E(C_j) \qquad (1)$$

where $E(C_x)$ represents a sum of squared Euclidean distances between the center of C_x and elements in C_x. The dissimilarity is represented as height in a dendrogram. The height of a smallest cluster ($H(SC_x)$) is uniquely-determined from the dissimilarity between a pair of two data (X_x) included in the cluster. The height of a cluster ($H(C_x)$) is also determined by the dissimilarity ($d(C_i, C_j)$) between following two clusters. Here we introduce two conditions for creating clusters by cutting multiple trees in a dendrogram.

1) Condition α: The condition α is used to determine whether to create a large cluster. For instance, as shown in Figure 3, the cluster C_i should be isolated from C_j since C_i includes license statements files with a single license (i.e., GPLv2). C_i can be created by cutting the tree between C_p and C_i. In order to do so, the tree will be cut if the following equations are met.

$$H(C_i) > H(SC_i) \ \ and \ \ H(C_i) > H(SC_j) \qquad (2)$$

It is assumed that SC_i and SC_j should be merged as a cluster (C_i) but should not merged to the next level cluster (i.e., C_p) because the cluster SC_i and the cluster SC_j are semantically closer than C_i and C_j.

If the equations are not met, the same processing are applied to the next level of clusters (i.e., C_i and C_j).

2) Condition β: Since a cluster is only created by merging two data or two clusters in the condition α, it cannot be created from a cluster and one data such as C_j in the right side of the dendrogram in Figure 3. The condition β is an extension of the condition α that allows a cluster to be created by merging a cluster and one data. Regarding a datum in a dendrogram as a cluster only with one data, the conditional equation 2 is used to create a cluster such as C_j. The condition β is intended to create more clusters and more isolated data so that license statements files are much more correctly grouped by license.

TABLE III
DISCRIMINATION ACCURACY OF GREP FOR GPL/BSD-RELATED LICENSES

License	TP	FP	FN	Precision	Recall	F-measure
GPLv1+	6	0	0	1.00	1.00	1.00
GPLv1	–	–	–	–	–	–
GPLv2+	96	1	53	0.99	0.64	0.78
GPLv2	74	3	71	0.96	0.51	0.67
GPLv3+	21	2	14	0.91	0.60	0.72
GPLv3	24	3	22	0.89	0.52	0.66
LibraryGPLv2+	13	1	2	0.93	0.87	0.90
LibraryGPLv2	6	0	0	1.00	1.00	1.00
LGPLv2.1+	18	1	13	0.95	0.58	0.72
LGPLv2.1	13	1	33	0.93	0.28	0.43
LGPLv3+	8	0	6	1.00	0.57	0.73
LGPLv3	13	3	8	0.81	0.62	0.70
AGPLv3+	11	0	0	1.00	1.00	1.00
AGPLv3	4	0	1	1.00	0.8	0.89
BSD2	17	27	3	0.39	0.85	0.53
BSD3	16	1	34	0.94	0.32	0.48
BSD4	2	0	7	1.00	0.22	0.36
all	342	43	195	0.89	0.64	0.74

IV. A CASE STUDY

This section describes a case study to investigate how well our clustering method can create clusters which could be candidates for generating license rules.

A. Dataset

In the case study, 12,725 software packages[2] of Debian v7.8.0 were used to obtain a set of unknown license files. In order to do so, compressed files with the file extensions of .zip, .tar.gz, .tar.xz, and .tar.bz2 were regarded as source code archives and were uncompressed. Of the uncompressed files, 12,725 source files written in Java, C, C++, Lisp, Perl, and Python were randomly sampled so that one source file for each software package can be selected.

B. Step 1: License Statements Extraction with Ninka

Ninka was used to identify OSS licenses of the sampled source files. As a result, 2,838 unknown license files were obtained as a set of unknown license statements files. All the unknown license statements files were manually reviewed and were confirmed that they consisted of 145 different OSS licenses.

C. Step 2: GPL/BSD-Related License Extraction with grep

1) Purpose: In step 2, GPL/BSD-related license statements files are extracted and grouped by license, using grep. We

[2]http://ftp.riken.jp/pub/Linux/debian/debian-cd/7.8.0/source/iso-dvd/

TABLE IV
METRICS TO EVALUATE CREATED CLUSTERS AND THEIR FEATURES

Metrics	Description
C	The number of created clusters.
SLC (Single License Clusters)	The number of clusters with a single license. Creating more SLC is desired in this study.
SF (Single Files)	The number of single files which are not clustered. SF are not desired because creating a license rule for each single file is not efficient.
LSLC (Licenses in SLC)	The number of licenses in SLC. It is ideal that LSLC is same as SLC.
RSLC (Ratio of SLC)	The ratio of the number of SLC to the number of C (SLC/C). A higher ratio is desired.
RSF (Ratio of SF)	The ratio of the number of SF to the number of C (SF/C). A lower ratio is desired.
RLSLC (Ratio of LSLC)	The ratio of the number of LSLC to the number of SLC. A higher ratio is desired.

analyze and evaluate that grep can correctly find GPL/BSD-related licenses from the unknown license statements files (i.e., 2,838 files) obtained in step 1.

2) Approach: Since the discrimination accuracy of using grep is of interest, the result of using grep is evaluated with precision ($\frac{TruePositive(TP)}{TruePositive(TP)+FalsePositive(FP)}$) and recall ($\frac{TruePositive(TP)}{TruePositive(TP)+FalseNegative(FN)}$) through comparing with the result of the manual inspection for 2,838 files in step 1. In this study we emphasize a value of precision, since it is well-known that precision and recall have a trade-off relationship and higher recall means that more licenses can be extracted but would be less correct (in case lower precision). Creating license rules correctly is much important in this context. Emphasizing a value of precision in this study is because of safety in creating license rules, though licenses for many files might not identified in this step.

3) Result: Of 2,838 unknown license statements files, 385 files were extracted and grouped by GPL/BSD-related license. Figure III shows the result of the discrimination accuracy of grep for each GPL/BSD-related license. "–" in the figure means that license statements for GPLv1 did not exist in the dataset. From Table III, we can confirm that the discrimination accuracy of grep in all was 0.89 of precision and 0.64 of recall.

D. Step 3: Grouping unknown license statements files by license

1) Purpose: In step 3, our hierarchical clustering method are applied to 2,453 unknown license statements files which are not extracted in Step 2. We analyze and evaluate that the method can create groups with a single license which would be better candidates for automating the license rule generation.

2) Approach: First, a dendrogram is created using the unknown license statements files. Then, clusters of the unknown license statements files are created according to the condition α and β for grouping the unknown license statements files.

The ideal clusters should only consist of groups with a single license. Through reviewing all the unknown license statements files in Step 3 for evaluation, 81 clusters and 37 single files (i.e., 118 licenses exist in Step 3.) should be created in the ideal case. Based on the ideal case, clusters created by our clustering method are evaluated.

3) Result: Table V shows the evaluation results. The condition β for the cluster division creates more SLC than the condition α. The condition α marks higher RSLC and lower RSF than the condition β. The condition α also marks higher RLSLC than the condition β[3] .

TABLE V
CLUSTERING CONDITIONS AND EVALUATION METRICS

	C	SLC	SF	LSLC	RSLC	RSF	RLSLC
cond. α	247	153	**48**	35	**0.62**	**0.19**	**0.23**
cond. β	719	**409**	207	42	0.52	0.29	0.10
ideal	81	31	37	17	0.36	0.46	0.55

V. DISCUSSIONS

This section discusses the results of our case study.

A. grep for GPL/BSD-related licenses

From the result of III, the precision in all was 0.89 that is not so high. In particular, the precision for BSD2 was the worst (0.39). These results indicate that it is still insufficient to address the technical challenge described in II-B1. In order to understand the reason why the precision for BSD2 was worst, we conducted an additional analysis by manually reviewing the unknown license statements files which failed to be discriminated. For BSD2, mis-discriminations often occurred to Apache License v1.1. Since Apache License v1.1 is created by adding clauses to and modifying BSD2, the discrimination condition for BSD2 in Table 2 mistakenly corresponds to Apache License v1.1. For this case, the precision can be improved by excluding license statements files which include the term "Apache". In the future, we also need to expand key phrases used in this study to handle licenses created by adding to and/or modifying existing licenses.

We also found other licenses were mis-categorized to BSD2 due to notation variants (e.g., "The name of " and "Neither name of") for clause3 used to detect BSD3 or BSD4. To detect BSD2 more correctly, additional patterns for clause3 are also needed to resolve the problem. For the GPL license family, the discrimination failed due to notation variants for "or later" which was expressed as "or above" or "or any newer version" in license statements. Additional discriminant conditions for the notation variants must be added to improve the precision of

[3]Note that the case study did not care if a license has exception clauses or not. It means that a license with exception clauses and a license without exception clauses could be categorized to the same cluster. The value of #SLC might be lower than that in Table V. Since discriminating licenses with or without exception clauses by clustering would be difficult, additional key phrases are needed for licenses with exception clauses in the future.

the GPL license family. In this manner, we observed notation variants caused many discrimination errors. However, in other words, it might indicate that our method is useful to detect notation variants which are unknown by license identification tools such as Ninka.

B. The Two Conditions for Clustering

From the result of V, the clustering condition α is better than the condition β. Using the condition β produces more single license clusters (SLC) than α, but it also produces much more single files (SF) than α. In contrast to β, the condition α creates clusters with the higher ratio of SLC to created clusters (RSLC) and with the lower ratio of SF to created clusters (RSF). This means that the condition α is more resistant to create a cluster which includes license statements files with different licenses. In addition, the ratio of the number of licenses in clusters (LSLC) to the number of (SLC) (i.e., RLSLC) is higher when the condition α is used. This means that the condition α is also more resistant to create multiple clusters for the same license and that the condition α is better to allow a cluster to include license statements files with notation variants.

The ideal state of clusters cannot be obtained unless the number of existed licenses are known. Anyone cannot know it before manually reviewing all unknown license statements files reported by a license identification tool. All the scores of RSLC, RSF, and RLSLC with the condition α are better if it is compared to the ideal state of clusters. Based on the discussions above, we can conclude that using the condition α for clustering creates clusters which are suitable as candidates for generating license rules automatically.

VI. Related Work

A. License compliance

In these days, many studies are tackling with the issue on the assurance of OSS license compliance. Sojer et al. [7] investigated commercial software' knowledge on OSS licenses. As a result, they found that commerce developer only have limited knowledge on OSS licenses and acquire the knowledge from unofficial sources of information. It is important for developers to learn right information about OSS licenses in the right place. German et al. [8] proposed a method so called Kenen that semi-automatically discriminates required licenses when reusing Java software components. Vendome et al. [9] studied on common understandings among developers on the reason and timing to change a license, through interviews of developers. Wu et al. [10] proposed a method to detect license inconsistencies in a large-scale OSS. As a result of an experiment using Debian v7.5.0, license inconsistencies were detected for Debian v7.5.0. The final goal of our study is to support rigorous compliance with OSS licenses by helping developers to easily create license rules.

B. License identification tools

Beside Ninka [1] and FOSSology [2], Tuunanen et al. [11] also proposed a rule-based license identification tool. Kapitsaki et al. [12] compared functions between license identification tools and showed advantages and disadvantages of the tools. We used Ninka for our case study, but our approach to automating the license rule generation would be applicable to other tools in the future.

VII. Conclusion and Future Work

Toward the automated license rule generation, in this paper we proposed a method to automatically classify unknown license statements files which are reported by rule-based license identification tools. Our method consists of (1) using grep to discriminate GPL/BSD-related licenses from others and (2) clustering unknown licenses into groups with a single license. A case study was conducted for licenses of 12,725 source files randomly sampled from software packages of Debian v7.8.0. Our method was applied to 2,838 source files which were identified as "unknown" by Ninka. As a result, we found that (1) 89% of GPL/BSD-related licenses were correctly discriminated by using grep and (2) our clustering method created clusters which are suitable as candidates for generating license rules automatically.

However, the discriminant conditions for GPL/BSD-related licenses should be improved based on the findings discussed in V-A, since they were still not perfect. The proposed clustering method also must be improved in the near future, since many clusters had license statements files with different licenses (i.e., our method still creates clusters which are not appropriate candidates for the license rule generation.

Acknowlegement

This work is conducted as part of Grant-in-Aid for Scientific Research: (C) 24500041 by Japan Society for the Promotion of Science.

References

[1] D. M. German, Y. Manabe, and K. Inoue, "A sentence-matching method for automatic license identification of source code files," in *Proc. of ASE '10*, 2010, pp. 437–446.
[2] R. Gobeille, "The fossology project," in *Proc. of MSR'08*, 2008, pp. 47–50.
[3] OSLC, http://forge.ow2.org/projects/oslcv3/.
[4] what license, http://www.what-license.com/.
[5] Ohcount, https://github.com/blackducksw/ohcount.
[6] J. H. Ward, "Hierarchical grouping to optimize an objective function," *Journal of the American Statistical Association*, vol. 58, no. 301, pp. 236–244, 1963.
[7] M. Sojer and J. Henkel, "License risks from ad hoc reuse of code from the internet," *Communications of the ACM*, vol. 54, no. 12, pp. 74–81, 2011.
[8] D. German and M. D. Penta, "A method for open source license compliance of java applications," *IEEE Software*, vol. 29, no. 3, pp. 58–63, 2012.
[9] C. Vendome, M. Linares-Vsquez, G. Bavota, M. D. Penta, D. M. German, and D. Poshyvanyk, "When and why developers adopt and change software licenses," in *Proc. ICSME*. IEEE, 2015, pp. 31–40.
[10] Y. Wu, Y. Manabe, T. Kanda, D. M. German, and K. Inoue, "A method to detect license inconsistencies in large-scale open source projects," in *Proc. of MSR '15*, 2015, pp. 324–333.
[11] T. Tuunanen, J. Koskinen, and T. Kärkkäinen, "Automated software license analysis," *Automated Software Engg.*, vol. 16, no. 3-4, pp. 455–490, 2009.
[12] G. M. Kapitsaki, N. D. Tselikas, and I. E. Foukarakis, "An insight into license tools for open source software systems," *Journal of Systems and Software*, vol. 102, pp. 72 – 87, 2015.