2017 IEEE/ACM 39th International Conference on Software Engineering

Machine Learning-Based Detection of Open Source License Exceptions

Christopher Vendome*, Mario Linares-Vásquez†, Gabriele Bavota‡,
Massimiliano Di Penta§, Daniel German¶, Denys Poshyvanyk*
*College of William and Mary, Williamsburg, VA, USA — †Universidad de los Andes, Bogotá, Colombia
‡Università della Svizzera italiana (USI), Lugano, Switzerland — §University of Sannio, Benevento, Italy
¶University of Victoria, BC, Canada

***Abstract*—From a legal perspective, software licenses govern the redistribution, reuse, and modification of software as both source and binary code. Free and Open Source Software (FOSS) licenses vary in the degree to which they are permissive or restrictive in allowing redistribution or modification under licenses different from the original one(s). In certain cases, developers may modify the license by appending to it an exception to specifically allow reuse or modification under a particular condition. These exceptions are an important factor to consider for license compliance analysis since they modify the standard (and widely understood) terms of the original license. In this work, we first perform a large-scale empirical study on the change history of over 51K FOSS systems aimed at quantitatively investigating the prevalence of known license exceptions and identifying new ones. Subsequently, we performed a study on the detection of license exceptions by relying on machine learning. We evaluated the license exception classification with four different supervised learners and sensitivity analysis. Finally, we present a categorization of license exceptions and explain their implications.**

***Keywords*-Software Licenses, Empirical Studies, Classifiers**

I. Introduction

Software licenses have been designed to facilitate and define the conditions under which software projects (open source and commercial) can be copied, modified, and distributed. Also, software licenses are a way to promote and support the philosophy and goals of specific communities of developers (*e.g.,* Apache and Eclipse) as well as of the open source community in general. As described by the License Proliferation Report by the Open Source Initiative [1] and reported by empirical studies from the research community [2], [3], [4], [5], [6], [7], [8], proliferation of FOSS licenses implies that (i) it is hard for developers to choose the right license for their projects and goals; (ii) there are incompatibilities between some licenses that might be a threat for developers' goals; and (iii) reuse of FOSS can lead to projects with multi-license distributions.

One partial solution to the aforementioned issues is the definition and usage of license exceptions that, when attached to a license, modify it without changing the text of the license itself [7]. For example, the MySQL database management system faced a challenging problem: It needed to stop commercial companies from reusing the MySQL connectors library (required to connect to the database) while still allowing other FOSS projects to continue using it. The first issue was easily addressed by using the GPL (since it would require those companies to also release their products under the GPL, unless they bought a commercial license). However, doing so would also stop projects that were not GPL licensed (such as PHP) to continue connecting to the database. The solution MySQL AB chose was to add an exception to the GPL (the MySQL FOSS License Exception) that allows software under certain FOSS licenses (including the PHP license) to use the connector libraries, effectively altering the scope and impact of the GPL on such software projects [9].

Licensing usage in FOSS has been investigated in several studies, mainly focused on identifying the prevalence and adoption of FOSS licenses and the developers' rationale under their licensing choices [2], [3], [4], [5], [6]. However, no previous effort has been devoted to analyze the prevalence/adoption of license exceptions. Given the large volume of FOSS projects available in forges (not only for reuse but also for direct usage), license exceptions might result in legal issues when developers/users are neither knowledgeable of the exceptions declared in the FOSS nor of the implications. While reliable tools for license identification and classification exist [10], [11], [12], [13], and some of these tools can be used to check for licensing inconsistencies [8], no tool is available to support the identification and analysis of license exceptions.

In this work, we aim to address two main questions:

- *Are the exceptions to FOSS licenses used by the community, which exceptions are used, and to what extent?*
- *Can we automatically detect license exceptions with a high precision and recall?*

We address these questions by first performing a large scale mining-based study in which we analyze the phenomenon of license exceptions from both a quantitative and a qualitative perspectives. In particular, we analyzed the source code of 51,754 projects written in six different programming languages (Ruby, Javascript, Python, C, C++, & C#) and hosted on GitHub, looking for usages of the license exceptions reported in the SPDX list [14]. By using defined heuristics, we found files identified as potentially containing license exceptions. Then, we manually inspected the files to (i) remove false positives, and (ii) categorize the exceptions according to their purpose.

Having assessed the magnitude of the license exception phenomenon used in FOSS projects, we devised an approach aimed at automatically identifying exceptions (if any) declared in a license.

1558-1225/17 $31.00 © 2017 IEEE
DOI 10.1109/ICSE.2017.19

Given a licensing statement, our approach exploits the words in its text as predictor variables for a machine learner that classifies the type of exception (categorical dependent variable) reported in the statement or marks it as "not reporting exceptions". We evaluated the precision and recall of our approach using Decision Trees, Naive Bayes, Random Forest, and Support Vector Machine (SVM). SVM outperformed the other classifiers, achieving an F-Measure higher than 95%. Since this is the first work to tackle the identification of license exceptions, we compared our approach with a baseline using template-matching techniques to identify license exceptions, showing the superiority of the SVM-based solution.

To the best of our knowledge, this is **the first work analyzing the usage and adoption of FOSS license exceptions** in open source projects and **to address the automatic identification of exceptions in licensing statements**.

II. The Impact of License Exceptions

A license exception, when attached to a license, changes its meaning and implication. Specifically, an exception can change the requirements of the license (expand them or narrow them) and/or the grants of the license (again, expand them or narrow them). A license with an exception effectively becomes another license. However, license identification tools have focused on identifying the licenses, and while they identify some exceptions, there has been no research in this area.

A recent discussion in the SPDX mailing list (SPDX, the Standard Package Description Language, is intended to create a standard format to document licensing information in software) focused on a relatively unknown exception created by Sun Microsystems and used along with variants of the BSD and the MIT license. This exception [15] states (emphasis is ours) [16]:

> You acknowledge that **this software is not** designed, **licensed** or intended **for use in** the design, construction, operation or maintenance of **any nuclear facility**.

One of the outcomes of this discussion is that this exception appears to restrict the original license to non-nuclear facilities ("is not [...] licensed [...] for use [...] any nuclear facility"). While the original license (BSD-like) is open source, the addition of this exception potentially turns it into non-open source because it appears to contravene clause 6 of the Open Source Definition that requires a license not to discriminate against fields of Endeavor [17].

License identification tools, Ninka [18] and Fossology [10] are capable to identify and classify licensing in software projects, but are oblivious to information that they do not recognize as a license. For instance, Fossology identifies the aforementioned Sun license—**including the exception**—as *BSD-style*. This implies that the license is open source, even though the exception potentially makes it non-open source.

We searched Debian source files to determine whether this license was being used in open source systems. We identified two main variations of this exception, and we found it in two packages of Debian. If this license is considered non-open source, that contravenes the Debian Guidelines and these packages need to be reviewed for their inclusion in Debian (we have reported this issue to Debian).

Other exceptions, such as the MySQL FOSS License Exception, and the Java CLASSPATH Exception [19] soften the restrictions of the GPL regarding derivative works. A library that is licensed under the GPL requires any derivative works that use it to be also licensed under the GPL. The MySQL FOSS License Exception when attached to the GPL allows the creation of derivative works that link to a library without also having to be licensed under the GPL as long as the derivative work is licensed under one of the FOSS licenses the exception lists. The Java CLASSPATH Exception is broader and allows the creation of derivative works under any license (including commercial) that link to libraries (licensed under the GPL plus the Java CLASSPATH Exception). If a library was licensed under the GPL plus either exception, but the exception is not identified, potential users of the library would not use it because they would not be willing to license their software under the GPL (something that the exception allows them to do).

III. Empirical Investigation on License Exceptions in GitHub Projects

The *goal* of this study is to investigate the phenomenon of license exceptions in FOSS projects hosted on GitHub. The *purpose* is to understand the relevance of license exception usage, identify what kinds of exceptions are being used, and understand implications of FOSS license exceptions. The *perspective* is of researchers interested in supporting developers with respect to license compliance and verification. The *context* consists of the change history of 17,984 Ruby, 14,161 JavaScript, 9,349 Python, 4,671 C, 3,690 C++, and 1,902 C# FOSS projects mined from GitHub.

A. Research Questions (RQs)

We aim at answering the following research question:

- **RQ$_1$:** *How prevalent are license exceptions in FOSS systems?* This research question analyzes the prevalence of different types of exceptions to FOSS licenses for projects hosted in GitHub and written in the six programming languages we considered. The goal is to understand license exceptions in practice, since they have not been investigated in prior studies. Besides *quantitatively* measuring the diffusion of different types of license exceptions, we also contacted the developers of the systems in which we identified exceptions to understand whether they are aware of the license exceptions. Additionally, we *qualitatively* discuss prominent cases that we found in order to better understand the context in which license exceptions are used.

B. Data Extraction Process

In order to identify license exceptions, we analyzed the commit history of 51,754 projects hosted on GitHub and developed in six of the most popular programming languages on GitHub [20]. We leveraged the project metadata to filter the repositories such that they had at least one star, watcher, or fork

and were not themselves a fork (*i.e.,* removing abandoned or personal repositories, and preventing duplication). We locally cloned the 51,754 project repositories to perform our analysis.

For each file f_i in the locally cloned repositories, we used *Ninka* [18] to extract a *comment file* C_{f_i} containing all source code comments in f_i (and, therefore, the license header with the exception text, if any). Then, we defined a set of heuristics to identify (candidate) license exceptions in each comment $c_j \in C_{f_i}$. The authors defined these heuristics by manually inspecting the description of the known/accepted license exceptions listed in SPDX[21]. In particular, we looked for sentences and keywords representing "markers" for the presence of a (specific) license exception. In the end, we defined the following heuristics, assuming that a comment c_j reports a license exception le_k if:

- $\mathbf{H}_1$ c_j contains the exact text (*i.e.,* definition) of a license exception le_k;
- $\mathbf{H}_2$ c_j contains the le_k's exception name (e.g., "autoconf" for the *Autoconf Exception*) and the token "exception";
- $\mathbf{H}_3$ c_j contains the string "as a special exception", a quite common pattern across the exceptions listed in SPDX.

It is important to note that these three heuristics are purposefully designed to address recall of license exceptions in order to identify the possible presence of license exceptions that were not listed by SPDX. $\mathbf{H}_3$ was designed such that it might be able to identify license exceptions not reported in SPDX. Also, as with any heuristic-based approach, our heuristics can lead to the identification of false positives. We deal with such limitations by manually analyzing every comment identified by our heuristics as reporting a license exception. In particular, our manual analysis (i) validated the presence of the license exception (*i.e.,* discarded false positives) and (ii) assigned a license exception name. If a comment reported a license exception without a known name, we assigned a custom name to the exception. Overall, our heuristics identified 728 files reporting candidate license exceptions for $\mathbf{RQ}_1$; then, we manually analyzed the files, getting 298 (40.9%) files with license exceptions (true positives). As previously stated, this true positive ratio is expected, since the heuristics were designed to capture exceptions that were not included by SPDX (while this reduced precision, we minimized the impact on our findings through the manual validation). It is possible that some license exceptions were not detected by the heuristics (false-negatives), but we tried to mitigate this by designing the heuristics from existing license exceptions. From the 298 files, we identified fourteen unique exception types, six of which are not documented/reported in the SPDX list.

To answer $\mathbf{RQ}_1$, we report the diffusion of the different exceptions in the mined repositories when considering them together and separately by different languages. It is important to note that the results report only systems with license exceptions and not the diffusion of licenses (*i.e.,* we do not consider all licensing of the 51,754 projects, but the subset that are licensed and have a license exception). Therefore, after the manual validation, we obtained a set E of tuples $E_j = \langle f_i, except, lang \rangle$ with $except$ being the license exception name, and $lang$ the programming language used in f_i.

TABLE I: Frequencies of license exceptions by language at file and system (in parenthesis) granularity.

Exception	Ruby	Py.	C	C++	C#	Total
Autoconf	11 **(3)**	20 **(7)**	11 **(1)**	30 **(3)**	0 **(0)**	72 **(14)**
Libtool	3 **(1)**	0 **(0)**	2 **(1)**	3 **(2)**	0 **(0)**	8 **(4)**
dh-Make	0 **(0)**	3 **(3)**	2 **(1)**	0 **(0)**	0 **(0)**	5 **(4)**
TexInfo	1 **(1)**	0 **(0)**	0 **(0)**	0 **(0)**	0 **(0)**	1 **(1)**
RACC	22 **(20)**	1 **(1)**	0 **(0)**	0 **(0)**	0 **(0)**	23 **(21)**
Bison	6 **(2)**	0 **(0)**	0 **(0)**	2 **(1)**	0 **(0)**	8 **(3)**
Nokia QT LGPL	0 **(0)**	0 **(0)**	0 **(0)**	2 **(1)**	0 **(0)**	2 **(1)**
Nokia QT GPL	0 **(0)**	0 **(0)**	0 **(0)**	49 **(1)**	0 **(0)**	49 **(1)**
Classpath	0 **(0)**	0 **(0)**	0 **(0)**	0 **(0)**	10 **(1)**	10 **(1)**
GUILE	0 **(0)**	2 **(2)**	0 **(0)**	0 **(0)**	0 **(0)**	2 **(2)**
Rails usage	19 **(18)**	0 **(0)**	0 **(0)**	0 **(0)**	0 **(0)**	19 **(18)**
MIF	0 **(0)**	0 **(0)**	8 **(1)**	1 **(1)**	0 **(0)**	9 **(2)**
OpenSSL	0 **(0)**	88 **(1)**	0 **(0)**	1 **(1)**	0 **(0)**	89 **(2)**
WxWin. Lib. 3.1	0 **(0)**	1 **(1)**	0 **(0)**	0 **(0)**	0 **(0)**	1 **(1)**
Total Files	62 **(22)**	115 **(13)**	23 **(2)**	88 **(7)**	10 **(1)**	298 **(45)**

C. Results for RQ_1: How prevalent are license exceptions in FOSS systems?

Table I shows the number of files reporting each of the 14 identified exception types across five programming languages (JavaScript is omitted, since we did not identify any exception in projects written in this language) as well as the number of projects (in parenthesis) containing each exception type.

1) Diffusion of different license exceptions: The *OpenSSL Exception* was the most prevalent, with 89 files having the exception across two systems. The *OpenSSL Exception* facilitates linking the licensed code to OpenSSL or derivative work that maintain the OpenSSL licensing terms. The *Autoconf Exception* was the second most prevalent (72 files containing the exception). This particular exception removes the copyleft requirement of the GPL when it is being used with a configuration script generated by Autoconf. The diffusion of the two exceptions above is not surprising, as OpenSSL is a widely diffused library, whereas Autoconf is a popular (and language independent) tool to generate configure scripts.

The *Nokia GPL Exception v1.3*, created to govern the redistribution of the Nokia Qt library, was the third most prevalent (49 files); however, all files reporting it were from the same project, *qtablet*. These 49 files were part of the *qtanimation framework-1.0-opensource*, which is a third party library for cross-platform software development in C++.

The *RACC Exception* (23 files) and *Rails Exception* (19 files) were the fourth and fifth most prevalent exception types, respectively. The similarity in frequency is due to these files being components of the same reused library. Therefore, these two exceptions were often found in tandem in the files `parser.rb` and `format.rb` belonging to *Action Mailer* [22], which facilitates sending and receiving emails in Rails applications. However, `parser.rb` was also utilized independently of `format.rb`. Racc is a parser generator for Ruby and the *RACC Exception* excludes the parsers that are generated by Racc from being licensed under the Ruby license (not yet approved by the Open Source Initiative). The *Rails Exception* allows for the usage of a MIT-like alternative license

when the source code is used with the official Rails or systems built upon the official Rails.

The *Classpath Exception* was the sixth most prevalent (ten files in a single system). It allows for linking a library to independent modules with requiring the generated binary from being licensed under the terms of the GPL.

The *Macros and Inline Functions Exception* (*MIF Exception*), which allows unrestricted reuse of executable that utilize macros, inline functions, or instantiate a template from the file containing the *MIF Exception*, was the seventh most prevalent with nine files containing the exception.

The *Bison Exception* and *Libtools Exception* were found in eight files each. The *Bison Exception* allows for unrestricted reuse of the Bison skeleton as long as the system is functionally different (*i.e.,* not a parser generator), while the *Libtools Exception* allows for unrestricted distribution of the file if it belongs to a system built by Libtools. Subsequently, we observed *dh-make Exception*, which resembles the *RACC Exception* differing in that it applies to dh-make output files instead of Racc output files, and occurs in five files.

The *Nokia Qt LGPL Exception* and *GUILE exception* tied with two files. The *Nokia Qt LGPL Exception allows* for unrestricted reuse of binary code that (i) utilizes only unmodified header files, modified code impacting numeric parameters, data structure layout, or (ii) the modification adheres to the MIF Exception, and (iii) adheres to the LGPL's Section 6 (facilitating reuse of the work as a library). The *GUILE Exception* is an exception for the executable generated by linking GUILE Library to other source files to be exempt from the terms of the GPL (*i.e.,* the generated binary does need to be releases under the GPL). Finally, there was one file with the *TeX Exception*, which excludes LaTeX files generated by *texinfo* from being licensed under GPL, and one file with the *WxWindows Library Exception 3.1*, allowing for unrestricted reuse of binary code based on the library that contains this particular exception instead of enforcing the terms of the GPL on the binary.

2) Distribution of license exceptions across programming languages: For C, we observed 23 exception instances made up of four exception types. The *Autoconf Exception* was most prevalent in C (11 files). The second most prevalent was the *MIF Exception* (eight files), while both the *dh-make Exception* and the *Libtool Exception* were present in two files.

For C++, we had the greatest variability with eight different types of exceptions and the highest overall number of files reporting exceptions (88). The most prevalent was the *Nokia GPL Exception v1.3* with 49 files from the reused Qt Animation Framework. The *Autoconf Exception* was second most prevalent in C++ (30 files), followed by the *Libtools Exception* (three files). The *Nokia Qt LGPL Exception* and *Bison Exception* tied for fourth (two files), while the *OpenSSL Exception* and *MIF Exception* tied for fifth each with one file.

For C#, we only observed the *Classpath Exception* in ten files and only from the system *Chefrate*. The C# files were reused components for the system's Png Encoder within the cross-platform and cross-browser API. Interestingly, the license header of these files also indicate that they were translated from Java to C# and thus inherited the *Classpath Exception* from the original Java implementation.

For Python, we observed five different types of exceptions resulting in 115 license exceptions. The *OpenSSL Exception* was most prevalent with 88 files containing the exception. The second was the *Autoconf Exception* with 20 files, while *dh-make Exception* was the third (three files). Additionally, we found the *GUILE Exception* once in two different projects, and we observed one file with the *WxWindows Library Exception v3.1* and one with the *RACC Exception*. The latter was the Ruby `parser.rb` file nested under a directory of external libraries.

For Ruby, we found six different types of exceptions across 62 different files. The *RACC Exception* was most prevalent license exception (22 files) and was closely followed by the *Rails Exception* (19 files). The *Autoconf Exception* was the third most prevalent exception with 11 files. The fourth was the *Bison Exception* (six files). The *Libtool Exception* had three files containing the license, while the *TeX Exception* was only attributed to a single file.

It should be noted that the *Autoconf Exception* was found in systems written in four languages with a relatively high prevalence (*i.e.,* top-3 across all four languages). Also, it seems that the programming language may influence the types of found exceptions. Indeed, the *RACC Exception* and *Rails Exception* are inherently coupled to Ruby files, which explains their isolation to Ruby (although a Ruby file did contain it in a python project). Similarly, Nokia's Qt Framework supports C++ development and contains an extensive API, impacting both the frequency of the exceptions when the libraries are reused and the isolation to C++.

D. An Initial Discussion and Learned Lessons

In this section, we first present feedback collected by surveying developers. Then, we present a manual categorization of license exceptions by similar features or properties.

1) Developer Awareness of License Exceptions: After identifying the license exceptions, we contacted the developers of the systems and asked them if they were aware of the license exceptions and if they understood the exception text (we provided paths to the files with exceptions and the exception name to the developers). We received feedback from seven developers contributing to five of the 45 projects reporting license exceptions. While the low response rate limits the ability to draw conclusions from the developer survey, the responses are still useful to understand the perspective from at least a subset of developers.

Interestingly, five developers were unaware of the license exceptions. One respondent thanked us for bringing the license exception to his attention and he expressed his intention to fix the licensing statement. This case is particularly interesting since it demonstrates the difficulty that developers may have tracing the licensing constraints of third-party code and reinforces the need for an automated tool to support the identification of license exceptions.

Additionally, we asked the developers whether they were able to easily understand the particular license exception.

While only two respondents replied that they do not understand (and one indicated that licensing is troubling in general), three respondents expressed their understanding and a potential ambivalence regarding their understanding (*i.e.,* they indicated they were uncertain, but provided their interpretation). The responses suggest the difficulty that developers have when licensing extends to license exceptions, even in cases of more straightforward exceptions (*e.g.,* the *Autoconf Exception*). Additionally, it demonstrates that certain developers would benefit from licensing tools that provide more contextualized licensing analysis, especially when exceptions are present.

The developers' responses demonstrate that license exceptions may not be easily identified within reused third-party code. Additionally, the implications of the license exceptions may not be easy for developers to truly understand. While the sample is small and not generalizable, it does suggest that there are developers, such as package managers, who would benefit from tools to identify license exceptions and determine license compatibility.

2) Categorization of License Exceptions: Based on their purpose, we can classify the exceptions found in our study into three major categories.

Added by a third party and applicable to reused components embedded into the client software. In this category, we found the *Autoconf, Libtool, dh-Make, TexInfo, RACC,* and *Bison Exceptions*. In all these cases, exceptions were found in source code that has been generated by another tool (the tools have the same name as the exception and they are licensed under the GPL). The goal of these exceptions is to clarify that, even if the generated output might contain copies of GPL licensed-software, the GPL license does not affect the software that is using the generated code. For example, Bison, a well-known parser generator, embeds into its output source of Bison itself. Since it is licensed as GPL, and the parser must be compiled within the client program, this would require any client program also to be GPL. The *Bison Exception* removes this requirement: *As a special exception, you may create a larger work that contains part or all of the Bison parser skeleton and distribute that work under terms of your choice, so long as that work isn't itself a parser generator using the skeleton or a modified version thereof as a parser skeleton ...*

In these cases, the text of the exception is embedded into the code generated by the tool and is not added by the authors of the software where they have been found. This is the main reason why these license exceptions are the most prevalent.

Added to clarify or expand use of the software. These exceptions are used as part of the license of the software where it is found and has been explicitly added by the author of the software. This includes the *Nokia QT LGPL/GPL, WxWindows Lib. 3.1, Classpath, GUILE, Rails*, and the *MIF Exceptions*. In these cases, the owner of the product is using the exception to either modify the GPL or to clarify when the terms of the GPL do not apply. For example, the goal of the *ClassPath* and of the *GUILE Exceptions* is to indicate that anybody is free to link to the (unmodified) library without having to release the code also as GPL.

The *QT Exception* is more limiting, since it only allows the linking with software that is licensed under specific open source licenses. The *MIF Exception* (Macros and Inline Functions Exception) clarifies that reusing templates and inlined macros (creating a copy of the original source code as part of the template and functions instantiation) is not a violation of the terms of the GPL:

> *[...] Specifically, if other files instantiate templates or use macros or inline functions from this file, or you compile this file and link it with other files to produce an executable, this file does not by itself cause the resulting executable to be covered by the GNU General Public License. [...]*

Added to allow linking to a library under a license that is not GPL compatible. This category contains only one exception: *OpenSSL*. OpenSSL is considered to be the best library in its kind, but its license is not compatible with the GPL. The exception allows explicitly the authors of GPL code to link OpenSSL even thought the license of OpenSSL is not compatible with the GPL:

> *[...] the copyright holders give permission to link the code of portions of this program with the OpenSSL library under certain conditions as described in each individual source file, and distribute linked combinations including the two [...].*

Given the importance of OpenSSL, its prevalence is not surprising.

E. Threats to Validity

Threats to *construct validity* relate to the relationship between theory and observation, and can be mainly due to imprecision when detecting license exceptions with the textual heuristics. To mitigate this threat, we manually removed the false positives from the automatically identified candidate exceptions. In that sense, the results we report are based on true positives only. However, it is possible that some license exceptions were not detected by the heuristics.

Threats to *external validity* relate to the ability to generalize the study results. We do not assert that the results reported in this paper are representative of the whole FOSS community. We only analyzed projects written in Ruby, JavaScript, Python, C, C++, and C#. Other languages and forges as well as commercial systems may exhibit different frequencies in terms of license exceptions. However, GitHub is the most popular forge with a large number of public repositories. Developers in other languages and utilizing other forges may have other perspectives related to license exceptions.

Threats to *internal validity* relate to internal, confounding factors that would bias the results of our study. We selected all projects from GitHub meeting the filtering criteria; therefore, we did not have a bias while selecting projects from a specific domain; however, because of time considerations we focused only on six programming languages. In the future work, we

aim to expand our analysis to other programming languages, forges, and specifically investigate libraries.

IV. Using Machine Learning to Identify License Exceptions

The study in Section III suggested that license exception identification is a cumbersome task as it requires developers to sift through a high number of potential false positives when using heuristics. Additionally, feedback from the developers suggests that an automated approach would be useful, especially for package maintainers. Automated approaches for text categorization/classification often rely on supervised learning to derive a model from labeled data (*i.e.,* text with labels) that can be used to categorize unseen data [23]. Text categorization has been successfully used in software engineering tasks such as software categorization [24], [25], [26], defect prediction [27], and developer recommendation [28], [29]. However, the task of automated classification of software licenses and license exceptions has not been solved using a machine learning-based approach. As a matter of fact, while tools to identify licenses exist [10], [11], [12], [18], no technique to automatically classify license exceptions has been proposed so far.

As we showed in our first study, keyword-based identification of license exceptions is highly prone to false positives, which suggests that "smarter" approaches should be used. One first option could be to use template-based identification (similar to Ninka [18]), or to use predefined queries to detect the license exceptions within a software license text. Our hypothesis here is that text categorization techniques can be used to detect license exceptions by ensuring higher accuracy with respect to techniques based on template-matching. Therefore, to assist developers in automatically identifying license exceptions, we implemented a text categorization approach. To validate the accuracy of the approach when classifying license exceptions automatically, we conducted an empirical study in which we compare the accuracy of supervised classifiers against a baseline representative of template-matching techniques.

More formally, the *goal* of this study is to evaluate a machine learning-based approach that we devised for license exception identification and compare it against a template-matching baseline for license exception identification. The *perspective* is of practitioners interested in ensuring license compliance of their systems, and the *context* consists of real license exceptions from our preliminary study and a synthetic dataset covering all license exception types.

A. Research Questions (RQs)

We aim at answering the following two research questions:

- **RQ$_2$:** *What classifier provides the best accuracy for license exception identification relying on machine learning?* We compare the performances of different classifiers for identifying license exceptions in licenses text.
- **RQ$_3$:** *Can our machine learning-based approach beat a baseline approach matching the license exception text?* RQ$_3$ aims at understanding whether a machine learning-based approach provides sufficient benefits to solve the problem of license exception identification. Thus, we compare it with a baseline approach (BL) that searches for the license exception text in the licensing statement.

B. Dataset Construction

In our first study, we did not find real examples for all the types of exceptions listed in the SPDX list [14]. Also, in some cases, we only found very few instances for a given exception (*e.g.,* the *Texinfo Exception* only had a single instance in our dataset). To train our machine learning (ML) algorithms by avoiding the "class imbalance problem" [30], [31], we created a dataset composed of real and synthetic license exception instances. In particular, given a set of software licenses text L, and a set of classes[1] representing the possible exceptions in licenses, we define a data instance for the classification process as a couple $d_i = \langle l_i, e_j \rangle$ where l_i represents a license text and e_j the specific license exception declared in l_i. We consider 33 license exception types, including the ones in the SPDX index [14], the ones in Table I, and two variants (one of the *FLTK* and one of the *Nokia QT GPL* exception).

As the classifiers should be able to distinguish when a license text l_i does not contain an exception, we included a negative class *Not-an-exception* to describe the case in which a data instance d_i is not representative of any exception (*e.g.,* the canonical text of GPL does not include a license exception). Therefore, our classifiers consider 34 possible classes to which a license text l_i can be assigned - 33 for the exceptions plus the *Not-an-exception* class.

Also, note that a license text l_i is assigned to only one e_j, and the motivation for this is that in our first study we did not find licenses with more than one exception. Therefore, the classification process we are conducting is single-label.

The procedure we followed to build the dataset for evaluating the proposed approach is the following:

1) For each e_j (*i.e.,* for each possible class) we build an empty bucket $B[e_j]$; each bucket contains the corresponding d_i instances from the real examples and the generated synthetic instances.
2) We assign the empirically found license exceptions (*i.e.,* $d_i = \langle l_i, e_j \rangle, e_j \neq Not\text{-}an\text{-}exception$) to the corresponding bucket $B[e_j]$.
3) Given a target sample size s to achieve within all the buckets, we fill each bucket $B[e_j]$ with synthetic instances until $|B[e_j]| = s$. A synthetic example for $B[e_j]$ is generated by randomly picking a canonical license (as indicated by the Open Source Initiative [32]) and appending the exception text of e_j after the license. This decision is based on the fact that in our previous study we did not find cases with an exception preceding a license; additionally, an exception applies to a license so it is reasonable to expect the license attribution prior to issuing an exception to the license.
4) For the examples in the negative class (*i.e.,* $e_j = Not\text{-}an\text{-}exception$), we fill the corresponding bucket by randomly

[1] "Class" refers to the target of the classification process (exception type).

TABLE II: The distribution of the unique license exception instances in our "real data" for the evaluation of RQ_3. Asterisks signify non-SPDX exceptions.

Exception	Unique Instances	Exception	Unique Instances
Autoconf 2.0	22	Libtool	1
OpenSSL	11	MIF	1
Bison 2.2	2	Nokia Qt GPL*	1
Nokia QT LGPL	2	RACC*	1
Classpath 2.0	1	Rails Usage*	1
dh-make*	1	TexInfo*	1
GUILE*	1	WxWindows	1

picking canonical licenses, without adding the exception at the end. We do not aim at identifying the type of a license, since existing license identification tools can deal with this task.

5) To ensure diversity of the licenses' text (including the exception text), each element in $B[e_j], \forall j \in [1, 34]$, is perturbed/mutated by randomly injecting typographical errors. We set a threshold of 1% of the words to be mutated, while also ensuring that at least one word was mutated. Prior work shows that scholarly writing texts can achieve an error rate of 0.2% [33], or 1.1% per word [34]. Therefore, to be conservative, we picked 1% as for the mutation rate in the texts (a developer is also less likely to review a header comment with the same detail as a publish work). These text mutations simulate slight changes that may occur in real data (*e.g.,* different copyright years or different copyright owner). Additionally, typographical errors are reasonable, as in the case of a license added by the National Institute of Standards and Technology in which there is a typo having "Untied Stated" instead of "United States" [35]. Finally, these mutations also allowed us to verify that our dataset did not have duplicated samples in the training and validation sets by computing SHA1 checksums of the files in both the training and test data.
6) Finally we split the data from each bucket into training and validation sets, by assigning 70% for training and 30% for validation.

In addition to the synthetic dataset, we built a second dataset to be used as test set. Note that this test set was unseen data and not part of the training and validation sets. This test set is composed only of real examples of canonical licenses and exceptions that we found in our preliminary study in the analyzed GitHub projects. Our goal with this dataset is to measure the generalization error [36] of the classifiers and of the baseline on "real data". To construct this dataset, we identified the unique instances from the results of our first study by computing the SHA1 (Secure Hash Algorithm 1) checksum. We only included one representative example of each unique instance. Table II shows the breakdown of the dataset consisting of real exceptions we used for evaluating the classifiers and baseline accuracy. For the canonical licenses, we added an instance of each canonical license to avoid any bias induced by choosing a subset and represent the $Not\text{-}an\text{-}exception$ class.

C. Building the Classifier

To build a classifier, we first extract terms from the licensing statements of the files under analysis. We perform a preprocessing in which we (i) remove English stop words, and (ii) weight the terms using the *tf-idf* weighting scheme [37].

Then, we use the data from our training set to build a machine learning classifier, using the license words as features (weighted by their *tf-idf*) and as a dependent variable the (manually labeled) kind of exception contained in the license.

We consider four machine learning classifiers that have been widely used for text categorization [23], [38]: decision trees (DT), Naive Bayes (NB), Random Forest (RF), and Support Vector Machine (SVM). To build the classifiers, we relied on the Weka [39] data mining library. The machine learners aim to classify the dataset into 34 classes: six new exceptions from our preliminary study (marked with an asterisk in Table II), 25 from SPDX's index [14], one *FLTK Exception* variant, *Nokia QT GPL Exception* variant, and the *Not-an-exception* class.

D. Analysis Method

To answer $\mathbf{RQ}_2$, we compared the accuracy of the classifiers in terms of the F-1 score [40], which is the harmonic mean of precision and recall, and the Receiver Operating Characteristic (ROC) area [41]; both metrics are widely used in the machine learning community to evaluate classifiers with a unified metric [40], [41]. The accuracy of the classifiers was measured with the synthetic dataset described in Section IV-B and with a dataset composed only of the real examples we found in our preliminary study. The evaluation on both datasets has the objective to verify whether our results are an artifact of the synthetic dataset. To measure the sensitivity of the classifiers to the sample size, we trained/tested the classifiers with samples sizes of 100, 200, 300, 400, 500, 1K, 2K, 3K, 4K, 5K, and 10K for each class (*e.g.,* $|B[e_j]| = 10,000, \forall j \in [1, 34]$). The results for $\mathbf{RQ}_2$ are reported in Section IV-E.

Concerning $\mathbf{RQ}_3$, since there is no existing approach for identifying license exceptions, we constructed a baseline (BL) to compare against the machine learning classifiers. The baseline approach attempts to search for the license exception text of each license exception. Since we already demonstrated in our preliminary study that a keyword-matching approach is prone to low precision (see Section III—40.9% of precision), we considered a more conservative baseline that matches the entire exception text. As for the target text to match, for each exception, we extracted a canonical example from the real data, and from the SPDX list when no instance was available in the real data. Subsequently, the approach matches text of the license exception to ensure the entire text is contained in the test file. If the match is confirmed, the text under test is tagged as containing the exception. As for $\mathbf{RQ}_2$, the comparison is done in terms on the F1-score achieved by the baseline and the classifiers, and by performing sensitivity analysis with different sample sizes. The results for $\mathbf{RQ}_3$ are reported in Section IV-F.

For both **RQs**, we use statistical tests to measure the significance of the achieved results. In particular, we use the Wilcoxon signed-rank test [42] (with $\alpha = 0.05$) in order to statistically compare each approach with the baseline across the different sample sizes. We use the (paired) Wilcoxon test as the comparisons (*e.g.,* SVM *vs.* BL) are performed between paired

samples. Since we perform multiple pairwise comparisons, we adjust p-values using the Holm's correction procedure [43].

In addition, we estimate the magnitude of the observed differences by using the Cliff's Delta (d), a non-parametric effect size measure for ordinal data [44]. Cliff's d is considered negligible for $d < 0.148$ (positive as well as negative values), small for $0.148 \leq d < 0.33$, medium for $0.33 \leq d < 0.474$, and large for $d \geq 0.474$ [44]. Finally, to visually corroborate the significance of the differences and the overlapping between the accuracies achieved by each approach, we computed the confidence intervals of the accuracies with 95% of confidence.

Replication: The dataset and results are available in our online appendix [45].

E. Results for RQ_2: What classifier provides the best accuracy for license exception identification relying on ML?

Synthetic data. Fig. 1a shows the F-1 scores achieved across different sample sizes for each classifier, and the confidence intervals around the means of the F-1 scores, on the validation set from the synthetic data. Table III reports the results of the Wilcoxon tests (adjusted p-values and Cliff's d) for each pairwise comparison. In general, there are statistically significant differences between NB and the other classifiers; this is confirmed by the F-1 curves, the ROC areas, the Wilcoxon tests, and the confidence intervals. Concerning the other three classifiers (*i.e.,* RF, SVM, and DT), the results suggest that SVM and DT, in general, outperform RF when training with samples sizes per class less than 1k. DT and SVM are close in terms of performance for all the sample sizes and their confidence intervals completely overlap each other and the difference is marginally significant (p-value=0.049). In the dataset with 1k samples per class, the difference between SVM, decision trees, and random forests become minimal. The ROC area is always $> 90\%$ for all the classifiers except for NB when the dataset has 100 instances per class.

Both DT and SVM exhibit a high precision and recall for all sample sizes. For a sample size of 200 DT outperforms SVM, and they are basically equivalent for a sample size of 5k. While random forests initially ranked third, it begins outperforming decision trees at sample size of 2k, excluding a performance drop in performance for a sample size of 5k. NB always exhibited the lowest precision and recall.

SVM is able to correctly classify 96.28% of the validation set in the worst case, and classifies 99.99% of the validation data correctly in the best case. The lowest precision achieved by SVM is 0.974 and the highest is 1 (a value of 1 is achieved when the incorrect classification was between 0.01% and 0.04%). We observe this drop in terms of magnitude (from 0.20% to 0.02%) for a sample size of 500. The best results for SVM are for a sample size of 2k when it only misclassifies 2 license exceptions. In fact, we observe the number of incorrectly classified exceptions decreases from sample size 100 until 1k when it rises before dropping again at a sample size 2k. After sample size 2k, the percentage of incorrect license exceptions rises from 0.01% to 0.04% of the test set.

We observe that decision trees, random forests, and SVM have a relatively similar accuracy in classification starting at a sample size of 2k. While SVM outperforms both random forest and decision trees. Decision trees incorrectly classify between 0.03% and 0.08% of the test set, while random forests incorrectly classify between 0.05% and 0.06% of the testing data, while SVM incorrectly classifies between 0.01% and 0.04% of the test set.

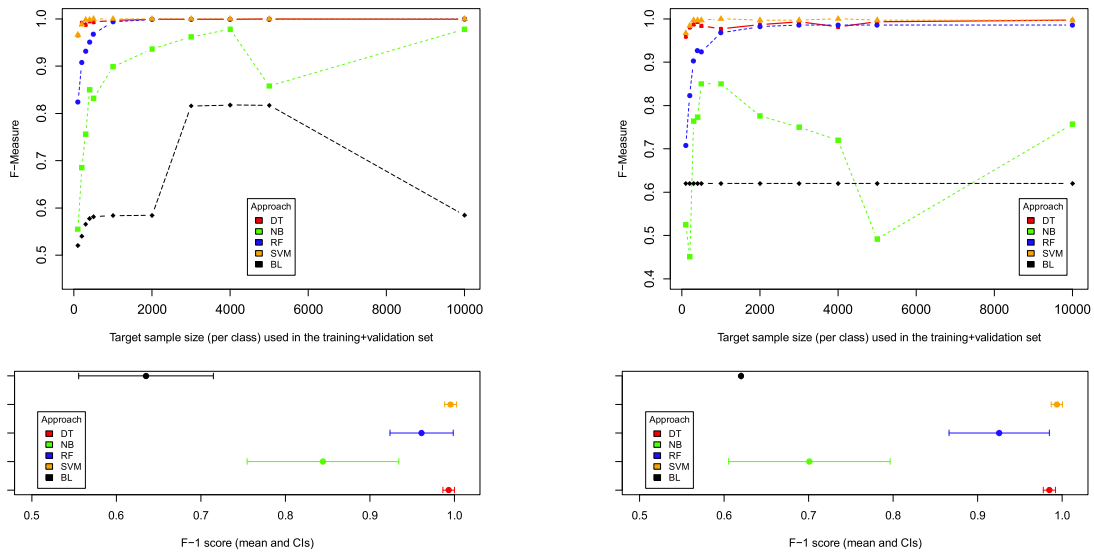
Real data. The results of the supervised learners on the real dataset (see Fig. 1b and Table III) demonstrate a similar behavior to their results on the validation dataset. In general, SVM outperforms the other supervised learners on the real data (although DTs marginally outperforms SVM for sample sizes of 100 and 200). SVM achieves an F-1 score of 0.997 or 1 for sample sizes 300 and larger (the precision is also 1 for each of these evaluations). In smaller sample sizes, we observe that DTs outperform RF, but this behavior switches above training sample size of 2k (inclusive of 2k). The ROC area is always $> 90\%$ for all the classifiers except NB when the dataset has 100, 300, 4k, and 5k instances per class.

Summary for RQ_2. *The results demonstrate that SVM, DT, and RF outperform NB with statistical significance in both validation and testing set. SVM also attains a higher precision and recall than DT and RF. In terms of the best accuracy, in the validation set, SVM is the first to achieve a ROC area of 99.99% at sample size of 2k, and a F-1 measure of 1 for datasets of at least 500 instances per class. In the real test set, SVM always exhibits a F-1 scores greater than 0.95, and a ROC area greater than 97%.*

F. Results for RQ_3: Can our machine learning-based approach beat a baseline approach matching the license exception text?

Synthetic data. The baseline approach exhibits high precision for all sample sizes in the dataset, ranging between 0.958 and 0.976. This result is expected since the approach relies on matching the text of the exception. Thus, it should find the correct match when the exception is directly copy-pasted. The negative class suffers from a high number of false positives, which harms the overall precision of the baseline approach (it wrongfully marks the license exceptions as licenses). The recall suffers since this type of baseline does not account for changes like modifications to a copyright year, copyright holder, or typographical errors in the exception. We observe a recall as low as 0.361 for a sample size of 100, and it achieves a maximum value of 0.677 for sample size 4k. However, we observe a decrease in precision at sample size 5k and 10k; the latter case drops to 0.418.

SVM outperforms the baseline in terms of both precision and recall. The precision of SVM and the baseline approach at sample size 100 is very close, only differing by 0.016, while the largest difference in precision is 0.0529 at sample size 4k. However, the recall is much lower for the baseline approach, never able to outperform any of the classifiers in terms of precision and recall for any of the sample sizes. SVM has more than twice the recall as compared to the baseline for samples sizes of under 2k (inclusive of 2k) and sample size



(a) Synthetic data (Validation set)

(b) Real data from GitHub projects (Test set)

Fig. 1: F-1 scores and confidence intervals achieved across different sample sizes when testing the analyzed approaches with a) synthetic data (*i.e.,* validation set), and b) license exceptions extracted from GitHub projects (*i.e.,* test set).

10k. In the best case, SVM still outperforms the baseline in terms of recall with a difference of approximately 0.3. In terms, of statistical tests, SVM has a significantly higher classification performance when compared to the baseline (p-value < 0.05 with a large—0.909—effect size).

Real data. In addition to the synthetic comparison, we compare the baseline and the supervised learners on the real dataset (see Fig. 1b and Table III). The real dataset consists of the unique license exceptions identified in the preliminary as well as an example of each canonical license (canonical licenses are for the negative class). The baseline (textual-based matching) achieves a precision and recall of 0.73 and 0.54, respectively. We observe that the baseline is able to outperform NB in terms of recall and F-1 score for the NB classifier trained on sample sizes of 100 and 200, and the baseline is able to outperform NB in terms of precision for sample size of 5,000, which is the only case of the baseline achieving a better precision than a supervised learner.

Fig. 1b depicts the F-1 score and confidence intervals for supervised learners and baseline when classifying the real data. We observe that the DT and SVM have similar confidence intervals that are tight around the F-1 score. Only NB and the baseline do not exhibit a significant difference in terms of performance. These results are confirmed by the confidence intervals shown in Fig. 1b. These results indicate that the supervised learners trained on synthetic data can achieve a high precision and recall with respect to classifying license exceptions and identifying the absence of an exception (*i.e.,* the negative class). Therefore, a supervised learner integrated after the initial license identification stage would enable a license compliance engine to determine if the particular license includes some additional exception to its terms.

Summary for RQ$_3$. *The results show that supervised learners trained on synthetic data were able to outperform a baseline approach when classifying synthetic and real data. Specifically, supervised learners are able to handle variations, which occur in practice. Furthermore, the supervised learners outperform the baseline approach on real data.*

TABLE III: Results of the Wilcoxon test (adjusted p-values and Cliff's d effect size) for the pairwise comparisons between the classifiers (DT, NB, RF, SVM) and the baseline (BL), when using the synthetic data (*i.e.,* validation set), first table, and when using real license exceptions found in GitHub project (*i.e.,* test set), second table.

Synthetic Data					
Comparison	**p-value**	**Cliff's d**	**Comparison**	**p-value**	**Cliff's d**
DT vs. NB	0.019	0.884	NB vs. RF	0.010	-0.652
DT vs. RF	0.049	0.214	NB vs. SVM	0.019	-0.966
DT vs. SVM	0.049	-0.454	NB vs. BL	0.010	0.686
DT vs. BL	0.009	0.909	RF vs. SVM	0.019	-0.561
SVM vs. BL	0.010	0.909	RF vs. BL	0.010	0.909
Real Data					
Comparison	**p-value**	**Cliff's d**	**Comparison**	**p-value**	**Cliff's d**
DT vs. NB	0.010	0.909	NB vs. RF	0.010	-0.835
DT vs. RF	0.022	0.595	NB vs. SVM	0.010	-1.000
DT vs. SVM	0.022	-0.669	NB vs. BL	0.083	0.455
DT vs. BL	0.022	0.909	RF vs. SVM	0.022	-0.835
SVM vs. BL	0.021	0.909	RF vs. BL	0.022	0.909

G. Threats to Validity

Construct validity threats can be mainly due to bias when sampling our datasets. We balanced the classes of our dataset and generated synthetic license and license exception pairs randomly. Additionally, we performed sensitivity analysis by varying sample sizes between 100 instances per class to 10k instances per class. *Internal validity* threats can occur in the creation of the training and test sets. When evaluating the performance of the classifiers, we considered 70% of the dataset for training and 30% for validation, which is an accepted practice for evaluating supervised learners. Additionally, we

had a separate testing dataset with real data. For what concerns *external validity* threats, it is possible that the performance of the classifiers on the synthetic dataset does not generalize. Such a dataset was designed to replicate user errors and aimed to ameliorate the problem of limited real data.

The comparison with the real data suggests that the perturbation rate of the dataset may inflate the differences between to licenses with exceptions. It is possible that results might vary using different datasets (*e.g.,* from other projects).

H. Discussion

This study addressed a novel problem of detecting license exceptions. Our results indicate the effectiveness of supervised learners in identifying (when present) different types of license exceptions reported in licensing statements. In general, SVM seems to be the most applicable supervised learner, since it outperformed the other algorithms. The results from the evaluation on the real data mimics our results on the synthetic data, which contains perturbations. The findings suggest that supervised learners are able to learn relationships between the terms of a license exception, while allowing for variations.

Results also indicate that keyword-based identification of license exceptions generate a large number of false positives (59.1%), which is not acceptable for an automated tool aimed at supporting software developers, especially in large software systems. Similarly, the recall of a baseline approach by matching the text of the license exception results in a low recall so it is likely to miss license exceptions. It is important to note that no other approaches exist and the baseline was designed to represent how a developer could identify license exceptions (*i.e.,* by pattern matching) without having to devise a sophisticated approach.

V. Related Work

Our work is related to prior approaches on license identification and previous empirical studies on software licenses.

License Identification. Several approaches exist to identify the license type and version of source code and jars, but these approaches do not address license exceptions. The FOSSology project [10] first utilized machine learning to classify licenses in order to solve the challenge of license identification. ASLA was also proposed by Tuunanen *et al.* [11] with a high accuracy of 89%. *Ninka*, the state-of-the-art approach proposed by German *et al.* [18] utilized sentence matching and was empirically shown to have a precision of 95%. Di Penta *et al.* [12] sought to identify the licensing of jars and proposed an approach employing code search (Google Code, which is no longer available). Lastly, German *et al.* [13] investigated the impact of propriety licensing, when used with FOSS licenses, on the ability to accurately identify the FOSS license by analyzing 523,930 archives.

These previous approaches focused on the identification of licenses, while not being capable to deal with license exceptions. By identifying license exceptions, our work represent a natural complement to previous work.

Empirical Studies. German *et al.* [7] identified exceptions as a viable method for licensors to modify a certain license. In that work, the authors proposed a model, based on their investigation of 124 FOSS packages, to illustrate the applicability of particular licenses. However, the work does not empirically investigate the existence of license exceptions. German *et al.* [8] studied license inconsistencies in Fedora-12 distribution. Importantly, this work demonstrated the importance of studying license exceptions, since the results showed that license exceptions are an important factor for validating potential license inconsistencies.

Additionally, Manabe *et al.* [46] investigated license changes in FreeBSD, OpenBSD, Eclipse, and ArgoUML and found the change patterns were project specific. Vendome *et al.* [3] investigated 16,221 FOSS Java systems to understand the license usage and changes in licensing. Additionally, the work investigated the reason for potential usage and changes by analyzing commits and issues trackers. Vendome *et al.* [2] also performed a survey involving software developers to investigate *when* developers pick a particular license or changes the license(s) and to understand the underlying reasons *why* developers choose or change licensing of their system.

Other empirical studies focused on license inconsistencies in code clones between Linux and either OpenBSD or FreeBSD [47], inconsistencies between the licensing of code clones in Debian 7.5, suggesting potential violations [48], and the presence of license violations in android applications [49].

VI. Conclusions

In this paper, we studied—for the first time, to the best of our knowledge—the presence of license exceptions in 51,754 FOSS systems from six languages. We found 14 different license exception types across 298 files in five of the six languages. While we observed that certain license exceptions are more prevalent in projects written in specific languages, we also found that the *Autoconf Exception* was within the top-3 most prevalent license exceptions for all of the languages (being Autoconf, a cross-language tool). Additionally, we also observed the frequent coexistence of the *RACC Exception* and *Rails Exception* due their presence in the same library. These license exceptions directly impact the way in which software can be reused and are critical for understanding license compliance. Specifically, these exceptions ameliorate inconsistencies under particular circumstances. We have sent the new license exceptions for consideration to the SPDX team.

After that, we evaluated the applicability and effectiveness of supervised learners for the identification of license exceptions. The results indicate that machine learning classifiers—and specifically SVM and Random Forests—are able to achieve high precision and recall when identifying the type of license exception as well as determining the lack of a license exception on real and synthetic data. A license exception classifier can be integrated into a license compliance engine after the initial license identification. In our future work, we aim to create a solution that integrates an SVM-based classifier for classifying exceptions into a license identification tool.

This research was supported in part via NSF CAREER CCF-1253837 and CCF-1525902.

REFERENCES

[1] The Open Source Initiative, "Report of license proliferation committee and draft faq. http://opensource.org/proliferation-report."

[2] C. Vendome, M. Linares-Vásquez, G. Bavota, M. Di Penta, D. M. German, and D. Poshyvanyk, "When and why developers adopt and change software licenses," in *The 31st IEEE International Conference on Software Maintenance and Evolution, ICSME 2015, Bremen, Germany, September 29 - October 1, 2015*. IEEE, 2015, pp. 31–40.

[3] C. Vendome, M. Linares-Vásquez, G. Bavota, M. Di Penta, D. M. Germán, and D. Poshyvanyk, "License usage and changes: A large-scale study of Java projects on GitHub," in *The 23rd IEEE International Conference on Program Comprehension, ICPC 2015, Florence, Italy, May 18-19, 2015*. IEEE, 2015, pp. 31–40.

[4] P. Singh and C. Phelps, "Networks, social influence, and the choice among competing innovations: Insights from open source software licenses," *Information Systems Research*, vol. 24, no. 3, pp. 539–560, 2009.

[5] M. Sojer and J. Henkel, "Code reuse in open source software development: Quantitative evidence, drivers, and impediments." *Journal of the Association for Information Systems*, vol. 11, no. 12, pp. 868–901, 2010.

[6] M. Sojer, O. Alexy, S. Kleinknecht, and J. Henkel, "Understanding the drivers of unethical programming behavior: The inappropriate reuse of internet-accessible code," *J. of Management Information Systems*, vol. 31, no. 3, pp. 287–325, 2014.

[7] D. M. Germán and A. E. Hassan, "License integration patterns: Addressing license mismatches in component-based development," in *31st International Conference on Software Engineering, ICSE 2009, May 16-24, 2009, Vancouver, Canada, Proceedings*, 2009, pp. 188–198.

[8] D. M. Germán, M. Di Penta, and J. Davies, "Understanding and auditing the licensing of open source software distributions," in *The 18th IEEE International Conference on Program Comprehension, ICPC 2010, Braga, Minho, Portugal, June 30-July 2, 2010*, 2010, pp. 84–93.

[9] Oracle, "Mysql connectors. http://dev.mysql.com/downloads/connector/."

[10] R. Gobeille, "The FOSSology project," in *Proceedings of the 2008 International Working Conference on Mining Software Repositories, MSR 2008 (Co-located with ICSE), Leipzig, Germany, May 10-11, 2008, Proceedings*, 2008, pp. 47–50.

[11] T. Tuunanen, J. Koskinen, and T. Kärkkäinen, "Automated software license analysis," *Autom. Softw. Eng.*, vol. 16, no. 3-4, pp. 455–490, 2009.

[12] M. Di Penta, D. M. Germán, and G. Antoniol, "Identifying licensing of jar archives using a code-search approach," in *Proceedings of the 7th International Working Conference on Mining Software Repositories, MSR 2010 (Co-located with ICSE), Cape Town, South Africa, May 2-3, 2010, Proceedings*, 2010, pp. 151–160.

[13] D. M. Germán and M. Di Penta, "A method for open source license compliance of java applications," *IEEE Software*, vol. 29, no. 3, pp. 58–63, 2012.

[14] The Linux Foundation, "Software package data exchange - license exceptions. https://spdx.org/licenses/exceptions-index.html."

[15] "Sun microsystems - bsd license," http://bit.ly/1SDwnNL.

[16] "New license/exception request: Bsd-3-clause-nonuclear," http://bit.ly/1rFJBmT.

[17] "Open Source Definition http://opensource.org/osd."

[18] D. M. Germán, Y. Manabe, and K. Inoue, "A sentence-matching method for automatic license identification of source code files," in *ASE 2010, 25th IEEE/ACM International Conference on Automated Software Engineering, Antwerp, Belgium, September 20-24, 2010*, 2010, pp. 437–446.

[19] The Free Software Foundation, "Gnu general public license, version 2, with the classpath exception http://openjdk.java.net/legal/gplv2+ce.html."

[20] http://githut.info/.

[21] The Linux Foundation, "Software package data exchange - fltk exception. https://spdx.org/licenses/FLTK-exception.html."

[22] "Action mailer basics. http://guides.rubyonrails.org/ action_mailer_basics.html."

[23] F. Sebastiani, "Machine learning in automated text categorization," *ACM Comput. Surv.*, vol. 34, no. 1, pp. 1–47, Mar. 2002.

[24] S. Ugurel, R. Krovetz, and C. L. Giles, "What's the code?: Automatic classification of source code archives," in *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '02. New York, NY, USA: ACM, 2002, pp. 632–638.

[25] C. McMillan, M. Linares-Vásquez, D. Poshyvanyk, and M. Grechanik, "Categorizing software applications for maintenance," in *Software Maintenance (ICSM), 2011 27th IEEE International Conference on*, Sept 2011, pp. 343–352.

[26] M. Linares-Vásquez, C. Mcmillan, D. Poshyvanyk, and M. Grechanik, "On using machine learning to automatically classify software applications into domain categories," *Empirical Softw. Engg.*, vol. 19, no. 3, pp. 582–618, Jun. 2014.

[27] T. Menzies and A. Marcus, "Automated severity assessment of software defect reports," in *Software Maintenance, 2008. ICSM 2008. IEEE International Conference on*, Sept 2008, pp. 346–355.

[28] J. Anvik, L. Hiew, and G. C. Murphy, "Who should fix this bug?" in *Proceedings of the 28th International Conference on Software Engineering*, ser. ICSE '06. New York, NY, USA: ACM, 2006, pp. 361–370.

[29] M. Linares-Vásquez, K. Hossen, H. Dang, H. Kagdi, M. Gethers, and D. Poshyvanyk, "Triaging incoming change requests: Bug or commit history, or code authorship?" in *Software Maintenance (ICSM), 2012 28th IEEE International Conference on*, Sept 2012, pp. 451–460.

[30] N. Japkowicz and S. Stephen, "The class imbalance problem: A systematic study," *Intell. Data Anal.*, vol. 6, no. 5, pp. 429–449, Oct. 2002.

[31] C. Lemnaru and R. Potolea, *Enterprise Information Systems: 13th International Conference, ICEIS 2011, Beijing, China, June 8-11, 2011, Revised Selected Papers*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, ch. Imbalanced Classification Problems: Systematic Study, Issues and Best Practices, pp. 35–50.

[32] The Open Source Initiative, "Open source licenses. http://opensource.org/licenses/category."

[33] J. J. Pollock and A. Zamora, "Collection and characterization of spelling errors in scientific and scholarly text," *Journal of the American Society for Information Science*, vol. 34, no. 1, pp. 51–58, 1983.

[34] F. Chédru and N. Geschwind, "Writing disturbances in acute confusional states," *Neuropsychologia*, vol. 10, no. 3, pp. 343 – 353, 1972.

[35] "Mediadescriptionimpl.java https://java.net/projects/jsip/sources/svn/ content/trunk/src/gov/nist/javax/sdp/MediaDescriptionImpl.java?rev=2364."

[36] Y. S. Abu-Mostafa, M.Magdon-Ismail, and H.-T. Lin, *Learning from Data*. AMLBook Press, 2012.

[37] R. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval*. Addison-Wesley, 1999.

[38] C. C. Aggarwal and C. Zhai, *Mining Text Data*. Boston, MA: Springer US, 2012, ch. A Survey of Text Classification Algorithms, pp. 163–222.

[39] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: An update," *SIGKDD Explor. Newsl.*, vol. 11, no. 1, pp. 10–18, Nov. 2009.

[40] M. Sokolova and G. Lapalme, "A systematic analysis of performance measures for classification tasks," *Information Processing & Management*, vol. 45, no. 4, pp. 427 – 437, 2009. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0306457309000259

[41] T. Fawcett, "An introduction to roc analysis," *Pattern Recogn. Lett.*, vol. 27, no. 8, pp. 861–874, Jun. 2006.

[42] S. D.J., *Handbook of Parametric and Nonparametric Statistical Procedures (fourth edition)*. Chapman & All, 2007.

[43] S. Holm, "A simple sequentially rejective Bonferroni test procedure," *Scandinavian Journal on Statistics*, vol. 6, pp. 65–70, 1979.

[44] R. J. Grissom and J. J. Kim, *Effect sizes for research: A broad practical approach*, 2nd ed. Lawrence Earlbaum Associates, 2005.

[45] "Online appendix: http://www.cs.wm.edu/semeru/data/ ICSE17_LicenseExceptions/."

[46] Y. Manabe, Y. Hayase, and K. Inoue, "Evolutional analysis of licenses in FOSS," in *Proceedings of the Joint ERCIM Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution (IWPSE), Antwerp, Belgium, September 20-21, 2010.*, 2010, pp. 83–87.

[47] D. M. Germán, M. Di Penta, Y. Guéhéneuc, and G. Antoniol, "Code siblings: Technical and legal implications of copying code between applications," in *Proceedings of the 6th International Working Conference on Mining Software Repositories, MSR 2009 (Co-located with ICSE), Vancouver, BC, Canada, May 16-17, 2009, Proceedings*, 2009, pp. 81–90.

[48] Y. Wu, Y. Manabe, T. Kanda, D. M. Germán, and K. Inoue, "A method to detect license inconsistencies in large-scale open source projects," in *The 12th Working Conference on Mining Software Repositories MSR 2015, Florence, Italy, May 16-17, 2015*. IEEE, 2015.

[49] O. Mlouki, F. Khomh, and G. Antoniol, "On the detection of licenses violations in the android ecosystem," in *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, vol. 1, March 2016, pp. 382–392.