Automated Software License and Copyright Analysis

by

A. R. Bates

B.S., University of Colorado at Boulder, 2006

A thesis submitted to the

Faculty of the Graduate School of the

University of Colorado in partial fulfillment

of the requirements for the degree of

Master of Science

Department of Computer Science

2010

This thesis entitled:
Automated Software License and Copyright Analysis
written by A. R. Bates
has been approved for the Department of Computer Science

_____

James H. Martin

_____

Prof. Elizabeth R. Jessup

_____

Prof. James H. Curry

Date _____

The final copy of this thesis has been examined by the signatories, and we find that both the content and the form meet acceptable presentation standards of scholarly work in the above mentioned discipline.

Bates, A. R. (M.S., Computer Science)

Automated Software License and Copyright Analysis

Thesis directed by Prof. James H. Martin

The complex interactions of software licensing and intellectual property prove daunting hurdles for many individuals and businesses looking to open source software solutions. The financial reproductions for misusing a piece of open source software is high, and require great attention. Many resources are required to determine a software packages copyright holders and licensing information. The cost of such an analysis may become too costly to justify the use of the open source solution. The existing tools for analysing software projects licenses and copyrights are lacking, and much hand vetting is required. If these tool could be improved then free and open source software would be more transparent and less costly to companies and individuals looking for open source alternative.

This thesis describes a new approach to automated software license analysis and copyright analysis, which results show are more accurate and easier to maintain than previous methods. The use of machine learning and information extraction result in algorithms that produce abstract models of software licenses and copyrights based on hand labelled data. We will show that these models are more general and robust than previous techniques, and result in better accuracy.

Dedication

To my mother. If it wasn't for your help I would still me labelling data.

Acknowledgements

Sarah A.M. Atherton, for being more patient than a saint and for all the editing.

Contents

Chapter

Tables

Table

Figures

Figure

Chapter 1

Introduction

In recent year the availability of free and open source software (FOSS) has allowed software developers to easily and quickly add functionality to their projects. Online shopping systems, databases, webservers, and operating systems are just a few examples of software can be found licensed as under a free or open source software. The advantages of using FOSS are that the initial cost is zero and you not only get the software solution, but a community that helps maintain and grow the product.

Although, free and open source software seems like a great option there are small hurdles which must be tackled before a FOSS package can be chosen over a commercial package, or the creation of the need software in house. The developers and contributors of the software have attached a software license that governs what, if anything, you may do with the software. Many of these licenses dictate that the users must state that they are using the software and where to get a copy of the original source code and license. Some of the more strict licenses prevent use in particular applications, such as nuclear facilities and defence projects, or require the release of intellectual property, such as software patents, into the public domain.

The user of the software package may have to make changes to the original source code in order to fit the specifics of the project. Many open source licenses require that the changes be documented and contribute back to the public. This however may introduce a problem if the open source packages is used for encryption, or secure communication. A recent case between the Free Software Foundation and TiVo examined whether TiVo was violating the terms and conditions of

the General Public License by refusing to provide encryption keys used by GLP'd software in the TiVo digital recorder [2]. These types of conflicts have influenced changes to the general culture behind FOSS. With the new version of the General Public License (version 3) brings new verbiage that prevent situations where users are prevented from recreating the functionality of FOSS used in a product, as TiVo did.

Given the importance of the licensing of open source software many developers are not well versed in the differences in the many licenses that exist today. This is only made harder due to license prolification. License prolification is the process where small changes to a software license overtime results in a brand new family of software licenses. Another problem comes from the confusion between copyright and licensing. Many software packages contain a phase such as "...this software is copyright some license..." which is incorrect. Only a person or organization may hold a copyright. The license only governs what extra privileges or restrictions a third party has to use and modify a particular work.

These problems cause companies to spend countless man hours determining what open source software is being used in their products, and ensure that the licenses don't conflict with their business model. This means that the companies avoid software licenses that require patents be relinquished into the public domain, or determining which software must be accessible to the customer and the appropriate method that are allowed for said software's delivery.

The need to locate software licenses is very important, but there are no standard method for declaring the governing license. This is left to the discretion of the developer. Some entities like Debian require software included in there main distribution to adhere to a strict format when declaring copyright and licensing information. Other entities such as Source Forge have no such requirement. Because of this each and every file must be searched for a reference to a license when determining what licenses are present in a software project.

Searching each and every source file requires resources that may not be available to even a fortune 500 company let alone a small business or individuals. This is abundantly true for software licenses that have been stitched together by the developer to define the usage of his or her software.

These types of licenses require the attention of a lawyer and possibly contacting the copyright holders for more clarification of their intentions of the license.

1.1 Software Licenses and Intellectual Propriety

Most large software organization have some sort of review board that determines whether a product entering the consumer world or the public domain contains intellectual propriety which will be lost by moving forward with product release. Many time this process is focused on the open source software licenses that are included in the product. High paid intellectual propriety lawyers spend countless hours along with project managers combing the products for licenses that may unintentionally release IP into the public domain.

This process starts by locating, normally by hand, all software licenses included in the project. These licenses are then classified based on there danger to the companies intellectual property, and the lawyers familiarity with the particular license. License which are never or only frequently encountered will be analysed by the IP lawyers for potential threats. In most cases these licenses are rare and the known licenses can quickly be classified. If a 'bad' license is found the project manager returns to the development team to determine if and alternative solution can be used. If the an alternative does not exist then the project manager may contact the copyright holder of the software and ask for permission to use the software without required adherence to the specific license.

This process is tedious and requires many hours of work to locate software licenses and copyrights contained in the software used in a project. The accuracy is dependant on the person conducting the analysis, and varies on outside factors other than the software package being analysed. This process results in a large hidden overhead to the use of FOSS simply because software licenses are hard to locate. If the software license could be ignored this process would be quite easy, but the stakes are large, many times a large amount of a companies intellectual property is on the line if a misclassification is made. Therefore, many companies forgo the use of FOSS to ensure the protection of their intellectual property and develop custom software in house.

1.2 Automated Solutions

If copyrights and software licenses are readily available then the issue of using FOSS becomes a trivial matter. The licenses must be vetted for terms of use that contradict ones business model. If a license encumbers business then the choice not to use the software is simple, and the cost to investigate a FOSS alternative relatively inexpensive. Many other domains use automated tools to analyse large amounts of data to determine information about products, individuals, and companies. School now use automated software to locate scan term papers for plagiarism. There also exists software to automatically grade term papers, and search source code to copied code. If a tool was available that would automate the process of extracting copyrights and software licenses from software packages it would cut a large amount of the cost out of using FOSS.

To date there are only a handful of automated solutions to tackle the problem of locating the correct software licenses and copyright holders in a software package. These solutions can be broken into two types: the Longest Common Substring approaches and the rule based approaches. The Longest Common Substring (LCS) approaches use a relaxation of the LCS problem to find a large string belonging to a template license in a piece of source code [7] and [19]. This matching string is then given the same name as the corresponding template license. The full text comparison results in an algorithm that is accurate but computationally expensive [9].

Alternatively, the rule based approaches employ a small set of phrases, sequences of unique words, or regular expressions to capture definitive features of the license. Rule based approaches only locate small strings in the license and disregard the rest of the license. The ignored text defines the rights and restrictions that embody the license, thus leaving the algorithm prone to monumental errors when the rule is not present but the basic license text is.

1.3 The creation of an Open Source Software License Analysis tool.

The goal of this work is to create an open source tool for software license analysis. We require that the tool is able to take training data in the form of files previously labelled by lawyers or other

expert as training data. This will allow non technical people to interface with the tool and add knowledge needed to improve performance. This differs from current techniques which rely heavily on customized regular expressions and other hand written rules that proof daunting to a naive user. We also require that the tools be robust and accurate. This will be achieved by creating hand labelled data sets to test the alogorithms performance.

1.4 License Prolification

There are many problems facing the creation of automated tools to locate and classify software licenses. The first is the large number of different software licenses in existence. Many of which are very similar to a single parent license, but contain a small set of important clauses that distinguish it from it parent license. Locating the differences between these licenses is very important [14]. For example Figure 1.4 is an example of the standard BSD license (with advertisement clause). Notice the last paragraph of the license is a non-standard addition to the BSD, that restricts the use of the software in a nuclear facility. This last paragraph is very important in cases where the software is used in a nuclear facility, but otherwise unimportant to the user. Many license analysis tools ignore this last paragraph. The current Fossology implementation of bSAM ignores this last paragraph since there is no matching template for the last paragraph in the database.

Figure 1.4 shows the first page of the General Public License version 2 (GPLv2). The full GPLv2 is over 2400 words long and is a copyrighted document. This means that you are not allowed to change the text of the GPLv2 except with express permission from the Free Software Foundation, Inc. Although the GPLv2 is copyrighted there are many examples of modification to the main text of the GPLv2. Many of these deviations are simple spelling mistakes or formatting error, but a large percent are actual additions or deletions to GPLv2, see [8].

Although the GPLv2 has rules for the way in which to reference the full license text many other none standard references exist. This creates problems determining licensing attributes even with the most commonly used license. The first three paragraphs in Figure 1.4 are the standard referencing text for the GPLv2, the last paragraph is an addition that removes a standard require-

ment that all command line programs must abide by. Figure 1.4 is an example of a non-standard license reference that has become popular because of its small size.

In order to automate the task of license analysis the algorithm must be able to locate all text that looks like a license and report it to the user. If possible the license sections should be match against a standard licenses. When a match is found a report of the additions and deletions should be reported. Since Figure 1.4 has an additional clause, and should be flagged for user evaluation.

```
GNU GENERAL PUBLIC LICENSE Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc., 51 Franklin
Street, Fifth Floor, Boston, MA 02110-1301 USA Everyone is permitted to
copy and distribute verbatim copies of this license document, but changing
it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to
share and change it.  By contrast, the GNU General Public ...
```

Figure 1.1: The first page of the General Public License version 2

```
This file may be distributed and/or modified under the terms of the
GNU General Public License version 2 as published by the Free Software
Foundation and appearing in the file LICENSE.GPL included in the packaging
of this file.
```

Figure 1.2: Non standard GPLv2 reference

1.4.1 Locating Software Licenses

Locating software licenses is the single most import problem to solve when determining what licenses govern the software package being analysed. This is because there are no rules that dictate how a software license should be attached to a copyrighted work. This means that where the software license is define is up to the description of the software developer. Many times the software license is located in text form nested in either a license file or the header of the source files. Some projects push standards to help ease the search for the governing software licenses. The Linux distribution Debian has strict guidelines outlining the correct location of the governing

software licenses and copyrights for all packages included in their distribution. Other products such as Google Code, Source Forge and FreshMeat try to provide licensing information by allowing the developers of the software to enter the licensing information. Google Code does provide some search functionality for software licenses, but these sites rely heavly on the developers to provide licensing information that is both correct and accurate.

1.4.2 Naming

Once the license is located it must be placed in a category. This maybe the parent license or perhaps a category of licenses with similar governing rules. The simplest way to license you source code is to place the software license at the top of your source files. This provides all copyright and licensing information on a per-file basis, thus making it easy to allow multiple licenses per project. Another common method is to place a reference to a software license by either name or point to a "License" file. This lessens the amount of effort that the developer must put into maintaining the licensing information and reduces the size of the source files. This method still allows for the analysis of a single file to determine its governing license. A third method instead places an umbrella license on all files in the project. In this case a single file will hold the license and state that all files belonging to the project are under the spesific license. Searching for a particular files licensing information becomes more difficult in this case, because a source file contains no reference to the licensing file.

This suggests three types of software licensing types. The first is the best case situation, in which each source file contains the full text of the software license. If a file contains a license of this type we will say it contains a "Full-text License". The second type of licensing is the "Reference License" which embodies all files which contain a reference to the full text of the software license, but does not actually contain the text of the software license. The third and final license type is the the "Umbrella License". This license type is retained for those license which claim governance for all files with in a project. This might be distinguished by clauses or additional statements that make claims to governance of all project files.

1.5 Thesis Statement

This thesis will create two new algorithms for analysing software licenses and copyrights. These algorithms will use learning based techniques in-order to allow users to continue to improve the accuracy and results of the algorithms. This will help to combat problems when new licenses are created or when new formats of copyright statements are found.

```
Copyright (C) 1989-2006 Free Software Foundation, Inc.

Bash is free software; you can redistribute it and/or modify it under the
terms of the GNU General Public License as published by the Free Software
Foundation; either version 2, or (at your option) any later version.

Bash is distributed in the hope that it will be useful, but WITHOUT ANY
WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
FOR A PARTICULAR PURPOSE. See the GNU General Public License for more
details.

You should have received a copy of the GNU General Public License with
your Debian GNU/Linux system, in /usr/share/common-licenses/GPL, or with
the Debian GNU/Linux bash source package as the file COPYING. If not,
write to the Free Software Foundation, Inc., 51 Franklin St, Fifth Floor,
Boston, MA 02110-1301, USA.

The Free Software Foundation has exempted Bash from the requirement of
Paragraph 2c of the General Public License.  This is to say, there is no
requirement for Bash to print a notice when it is started interactively in
the usual way.  We made this exception because users and standards expect
shells not to print such messages.  This exception applies to any program
that serves as a shell and that is based primarily on Bash as opposed to
other GNU software.
```

Figure 1.3: Standard GPLv2 reference with an exemption clause.

```
JOGL is released under the BSD license.  The full license terms follow:

Copyright (c) 2003-2007 Sun Microsystems, Inc.  All Rights Reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are
met:

- Redistribution of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.

- Redistribution in binary form must reproduce the above copyright notice,
this list of conditions and the following disclaimer in the documentation
and/or other materials provided with the distribution.

Neither the name of Sun Microsystems, Inc.  or the names of contributors
may be used to endorse or promote products derived from this software
without specific prior written permission.

This software is provided "AS IS," without a warranty of any kind.  ALL
EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING
ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE
OR NON-INFRINGEMENT, ARE HEREBY EXCLUDED. SUN MICROSYSTEMS, INC. ("SUN")
AND ITS LICENSORS SHALL NOT BE LIABLE FOR ANY DAMAGES SUFFERED BY LICENSEE
AS A RESULT OF USING, MODIFYING OR DISTRIBUTING THIS SOFTWARE OR ITS
DERIVATIVES. IN NO EVENT WILL SUN OR ITS LICENSORS BE LIABLE FOR ANY LOST
REVENUE, PROFIT OR DATA, OR FOR DIRECT, INDIRECT, SPECIAL, CONSEQUENTIAL,
INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE
THEORY OF LIABILITY, ARISING OUT OF THE USE OF OR INABILITY TO USE THIS
SOFTWARE, EVEN IF SUN HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

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

Figure 1.4: BSD license with non nuclear clause

Chapter 2

Copyright Analysis

In this chapter we will explore techniques for locating copyright statements. This information is critical when there are questions about the licensing and distribution of the software. The extraction of the copyright along with the license allows intellectual property lawyers to quickly contact the copyright holders with questions. The copyright information is even more important when a critical piece of software does not contain a software license. In these cases the only recourse is to contact the copyright holder and ask permission to use their software. A third possible use for copyright statements, suggested by [6] and [20], is to extract the copyright holders and track an individuals contributions over a set of projects. To our knowledge there is no learning algorithm except ours which performs automatic extracting copyright statements, although the idea is suggested in several papers ([3], [6] and [20]).

We will start this chapter by examining the most common copyright statements. Many of these copyright statements fit a pattern that is easily captured by a regular expression. We will show that the regular expression is too general and does not capture the more general cases. We will show how replacing the regular expression with a naive Bayes classifier trained on hand labelled data can achieve an accuracy of 88.8%. The final algorithm is composed of a naive Bayes classifier that uses word tokens and noun phrase boundaries, which results in an algorithm with an observed accuracy of 99%.

To test our algorithms 200 copyright files were selected from the Debian linux distribution to serve as a training and test set. Each copyright statement was labelled by placing a $< s >$ tag

Token	% Begin
contributed	1
authors	4
author	5
written	6
copyright	83

Table 2.1: Percent first word of copyright statements

at the begining and a $< /s >$ tag at the end of the statement. To determine heuristics and train our algorithms we broke the set of labelled files into two sets. A training set, fully observable, and a test set which was kept separate and only used to determine an estimate on the accuracy of the algorithms on unseen data. The training set consisted of 137 files and the testing set is composed of the rest of the labelled files.

A quick analysis of the training set showed that a small set of keywords defined the start of the copyright statements. Table 2 shows the top five tokens that start copyright statements in the training data. Since nearly 99% of the copyright statements start with one of five words the task of locating copyright statements seems a trivial one, but Table 2 gives the ratio of times that the top five words are found inside vs. outside of a copyright statement. This suggests that even though there are a small number of tokens that define the beginning of a copyright statement it is still relatively hard to create a simple algorithm that has a low false positive rate.

The regular expression has the ability to locate the beginnings of 95% of the copyright statements. This can be interpreted to mean that the start of a copyright statement is very regular. The major problem with this regular expression is that it does not have the ability to determine the

Token	Inside	Outside
author	36	73
copyright	590	946
contributed	7	2
written	44	99
authors	29	107

Table 2.2: Word counts for inside copyrights vs. outside.

Figure 2.1: A regular expression for locating copyrights

ending of the copyright statement. In fact the regular expression only has a classification rate of 15% for copyright endings, if a period or carriage return is used to determine the ending. By looking closer at the ending of the copyright statements we can see that many of these statements are ended by the phrase "All right reserved.". When this check is added the classification rate increases to 35% for endings. After adding more corner case checks the best a simple regular expression can do is catch 65% of the endings of the copyright statements, see Figure 2, and has a final observed accuracy of 78%.

Although a regular expression can't locate the endings of copyright statements with the precision required to automate the search for copyright information it is successful at locating the start of the statements. For this reason we will use the regular expression to help extract useful features, and spend the rest of this section exploring more precise algorithms for locating the ending of a copyright statement.

2.0.1 Naive Bayes classifier

Naive Bayes classifiers are known for there simplicity and easy training and implementation. This makes it a perfect candidate to help solve the problem of locating the ending of copyright statements. The naive Bayes classifier estimaties the probability distribution function $P(F_1 = f_1, F_2 = f_2, ..., F_d = f_d | C = c)$, where the vector F is a feature vector containing information about the textual features in the document and c is the class which in our situation can be in the set 'I': INSIDE, 'O': OUTSIDE, 'B': BEGINNING. Naive Bayes make the simplifying assumption that the features are independent, which allows the pdf to be simplified to $P(F_1 = f_1 | C = c)P(F_2 = f2 | C = c)...P(F_d = f_d | C = c)$. In theory all words are known and have a probability of belonging to each of the classes. Many words such as names, and other proper nouns are never seen in the training data, which means the naive Bayes classifier will produce a zero probability for the event. To

account of this missing data a small amount of noise is added to each word count. We used Laplace smoothing in our baseline algorithm which assumes that each feature has 1 extra observation than truly exists in the training set. This allows the algorithm to easily account for unseen data, because its frequency is now 1 and prevents divide-by-zero errors. Equation 2.3 gives the final formulation of the naive Bayes algorithm used for the baseline, where J is the number of distinct values F_i can take on, N is the number of training examples in the training set, and $count\,(x)$ is the number of observations of x in the training data.

$$\hat{P}\,(C = c_k) = \frac{count\,(C = c_k)}{N} \tag{2.1}$$

$$\hat{P}\,(F_i = f_i | C = c_k) = \frac{count\,(F_i = f_i, C = c_k) + 1}{count\,(C = c_k) + J} \tag{2.2}$$

$$C \leftarrow \underset{c_k}{\operatorname{argmax}}\, \hat{P}\,(C = c_k) \prod_i \hat{P}\,(F_i | C = c_k) \tag{2.3}$$

We created a simple naive Bayes classifier to serve as a baseline to test more advanced algorithms. IOB tagging was used to label the copyright statements. IOB tagging allows the labelling of phrases and chunks for use with classification algorithms, see [10] and [15]. A tri-nary classifier is created which can locate the Beginning, Inside, and Outside of a copyright statement. The previous label is then used as a feature in the classifier to encode information about the previous state into the classifier. The baseline uses the current word, previous word and the previous label as input features.

Figure 2.1 shows that the simple naive Bayes baseline algorithm already out performs the regular expression. The performance of the baseline does depend significantly on the size of the training set, where as the regular expression does not depend on training data, and does not fluctuate with training size. By expanding the feature set of the naive bayes model and implementing more advanced methods for smoothing we will show how the simple naive Bayes model can be leveraged to provide a stable and accurate copyright extraction algorithm.

2.1 Not so naive Bayes

Our first optimization to the simple naive Bayes model was to include bi-grams to the feature set. The addition of bi-gram features allows the naive Bayes model to better fit non-linear trends in data. By adding the bi-grams our classification accuracy went up by 2%. This is a great result for a simple feature addition, but it still doesn't provide a significant increase in f-measure, only 83.34. Since there are only 200 labelled files we added linear bi-gram smoothing [10] to help get a better representation of the true parameter space with out having to label more data. This resulted in a much better approximation of the data and an observed accuracy of 94% and an F-measure of 85.46.

Finally we observed that there exists at least one noun phrase in each of the copyright statements. Further analysis showed that a copyright statement ends just after the end of the noun phrase. This suggests that noun phrases should be included in our model so that a better fit of the data can be made.

To locate noun phrases we first used the part-of-speech tagger included in the Natural Language Toolkit [1] to label the parts of speech of the tokens in the document. Then we used the grammar in Figure 2.1 to extract the noun phrases. A new feature variable was added with values in the set (NP_B, NP_I, NP_O). Using 10-fold cross validation it was found that the optimal feature set consisted of the following features;

$$NP : \{< DT|PP? >? < JJ > * < NN|NNP|NNS|NNPS > + < CD >?\}$$

Figure 2.2: Grammar used to construct noun phrases from parts-of-speech

- Current Token (Un-stemmed)
- Previous Token
- Previous Label
- Current Noun Phrase

- Previous Noun Phrase
- Current Bi-gram
- Previous Bi-gram

Confusion Matrix			
True \ Predicted	B	I	O
B	93.15%	0.68%	6.16%
I	0.60%	67.55%	31.85%
O	0.01%	0.21%	99.77%

Table 2.3: Confusion Matrix Naive Bayes with bi-grams and noun phrases

Accuracy	0.98
Precision	0.95
Recall	0.87
F-measure	0.91

Table 2.4: Results for Naive Bayes classifier using bi-grams and noun phrases

The naive Bayes model with bi-grams and noun phrases produced the best accuracy and F-meausre. It was also able to model copyright statements with only a small number of examples, seen in Figure 2.1. Table 2.3 shows the confusion matrix for the noun phrase model. Many of the misclassification of the 'I' and 'O' tags result from inconsistent human labelling near the end of the copyright statements. Table 2.1 contains the results of final naive Bayes classifier with bi-grams and noun phrases.

Currently, the Fossology Project (http://www.fossology.org) is using the naive Bayes with bi-gram version of the algorithm. This version will be packaged with the 1.2 version of Fossology and the final version of the algorithm with bi-grams and noun phrases will be merged into the testing version later this summer. The current implementation in Fossology can analyze a 1GB DVD ISO in just over an hour. This is faster than the rest of tha analysis tools used in Fossology, and has resulted in special work-a-rounds to prevent the Fossology scheduler from crashing due to the high rate of transactions from the copyright analysis algorithm.

□

Figure 2.3: Copyright algorithms: accuracy vs. training size

Chapter 3

Sentence Based License Analysis

3.1 Introduction

We propose a replacement of the current license analysis tool in the Fossology project, which relies on technology from the 1980s [4] [17], with a novel algorithm that uses machine learning and information extraction techniques. The new algorithm uses software licenses from a database of known and categorized licenses labelled by experts to identify new licenses. Unlike other license analysis tools, we utilized hand-labelled test data to determine the algorithm's accuracy.

Using two hand labelled data sets we will show that the our approach out preforms the competition in the following areas. First, our approach allows users to add new license templates with out need of knowledge of software licensing, machine learning, nature language processing, regular expressions or computer programming languages. This is because the algorithm uses simple plain text formatted license templates to create models of license families, which can be used to identify related licenses. This makes our analysis software more accessible to the people who are most likely to use it, i.e. lawyers and project managers whose time is already spread thin with out having to learn a special tool. Second, our algorithm uses a combination of machine learning and natural language processing to classify both license reference licenses and full license texts correctly. This is something that previous techniques have trouble handling correctly. Finally, We will show how our algorithm is able to cope with license prolification.

We created two datasets and tested against an existing data set from [7]. The first data set was created by the fossology.org team to test the accuracy of new license analysis algorithm for

the 1.3 release candidate of Fossology. It consists of 294 licenses of which there are 132 unique licenses. These tests are not only meant to test the algorithms accuracy, but also the tokenization and general robustness when analysing large and complex files such as dictionaries and spelling databases. The second dataset was created as a training set for the current algorithm used by the Fossology project. It contains 285 license templates found in various software projects. Each license template has been grouped with similar licenses to create families of licenses. This was done by experts in software licensing from the Fossology project. The third test compares the analysis that German et. al. [7] conducted on the prominent license analysis algorithms. These are hand labelled files randomly selected from the main distribution of Debian linux.

Many of the sentences found in software licenses are shared by one or more other licenses. This observation has lead to a unique algorithm that is both fast and accurate. Unlike other algorithms that require a full test search over all template licenses or only searches for keyword and may miss additions or deletions our algorithm uses sentences to quickly compare the whole license text and accurately locate the defining features of the individual licenses.

3.2 The Algorithm

The algorithm proposed to replace the current out dated license analysis algorithm in the Fossology project uses labelled reference licenses to learn important details about the structure and language in families of software licenses. Before our algorithm can be used to classify new software licenses it must be trained on a set of existing labelled licenses. During training the reference licenses are tokenized, split into sentences, and then stored in a heirarchial vector space database for quick retrial. Once in the database the new license can be compared to the vectorized sentences.

To analyse a file it is tokenized and a naive Bayes classifier is used to locate sections of text that have a high probability of coming from a software license. These sections are then passed to the sentence parser and turned into text frequency vectors so they can be compared to the sentences in the database. To compare to the licenses in the database a similarity matrix is created for each reference license and the file being analysed. These matricies are then converted to binary

Figure 3.1: Regular expression used to tokenize the input files.

matching matricies. With these matricies the longest common subsequence of matched sentences can be determined. The section is then classified based on the best match, or marked as "unknown" if no match was found.

The following section delve into the technical details each of the sub-processes. These include tokenization, license location, sentence parsing and sentence matching. Each section has information about the implementation and the accuracy of the algorithm used to accomplish the task.

3.2.1 Tokenization

Figure 3.2.1 contains the regular expression was used to tokenize the licenses. The regular expression used extracts words, number, email address, file paths and URLs. Notice that the regular expression is setup in an if, else if, ..., else fashion to allow the regular expression to be optimized once compiled. Groups, `(?P<name>...)`, are used to easily extract features from the tokenized text. All groups, except 'words', are replaced with 'XXX`groupname`XXX' to allow the algorithms to generalize over features such as emails, urls, and dates. For example, "http://www.fossology.org/" is of class 'URL' and will be generalized as 'XXXurlXXX'. This helps account for software licenses that allow for arbitrary URLs, emails, and version numbers to be referenced.

Figure 3.2.1 shows a small snippet of text from the Apache 2.0 copyright file and the tokens produced by our tokenizer.

3.2.2 License Locator

We begin by creating a general model of a software license. We use a naive Bayes classifier to calculate the probability of a word originating from a software license. Thus, providing a tool to locate any license text in ant text document. Naive Bayes employs the initial labelled data set along with Bayes' theorem to generate two conditional distribution functions (cdf), which can then

```
Copyright:

Licensed to the Apache Software Foundation (ASF) under one or more contributor
license agreements.  The ASF licenses this work to You under the Apache License,
Version 2.0 (the "License"); you may not use this work except in compliance with
the License.  You may obtain a copy of the License at
http://www.apache.org/licenses/LICENSE-2.0

On a Debian system, the license can be found at
/usr/share/common-licenses/Apache-2.0.

tokens = [‘XXXcopyrightXXX’, ‘:’, ‘..’, ‘Licensed’, ‘to’, ‘the’, ‘Apache’,
‘Software’, ‘Foundation’, ‘(’, ‘ASF’, ‘)’, ‘under’, ‘one’, ‘or’, ‘more’,
‘contributor’, ‘license’, ‘agreements’, ‘.’, ‘The’, ‘ASF’, ‘licenses’, ‘this’,
‘work’, ‘to’, ‘You’, ‘under’, ‘the’, ‘Apache’, ‘License’, ‘,’, ‘Version’,
‘XXXnumberXXX’, ‘(’, ‘the’, ‘"’, ‘License’, ‘"’, ‘)’, ‘;’, ‘you’, ‘may’, ‘not’,
‘use’, ‘this’, ‘work’, ‘except’, ‘in’, ‘compliance’, ‘with’, ‘the’, ‘License’,
‘.’, ‘You’, ‘may’, ‘obtain’, ‘a’, ‘copy’, ‘of’, ‘the’, ‘License’, ‘at’, ‘..’,
‘XXXurlXXX’, ‘..’, ‘On’, ‘a’, ‘Debian’, ‘system’, ‘,’, ‘the’, ‘license’, ‘can’,
‘be’, ‘found’, ‘at’, ‘XXXpathXXX’, ‘.’]
```

Figure 3.2: Example of the output of the tokenizor

be utilized to calculate the probability of a word belonging to a software license. The advantage is that it runs in linear time, and is easy to train and implement. Accordingly, we can locate all the licenses in a project in a short time without needing to reference the license templates. We use this technique to get a set of license sections for each file, that are then passed to the next stage of the algorithm.

To train the naive Bayes classifier 500 documents with hand labelled software license sections are used to calculate $Pr(word|L)$ and $Pr(word|\bar{L})$, where L is a license sections and $\bar{L}$ is a non license section. A windowing technique is used to classify sections of text. Window i, of size w, is defined as the set of words $\{word_i, word_{i+1}, \ldots, word_{i+w}\}$. The conditional probabilities, $Pr(L|word_1, \ldots, word_w)$ and $Pr(\bar{L}|word_1, \ldots, word_w)$, are calculated for a window of size w. If we assume that words appear independently of each other, then the conditional probabilities simplify to,

$$\prod_{i=1}^{w} Pr(word_i|L) \tag{3.1}$$

and,

$$\prod_{i=1}^{w} Pr(word_i|\bar{L}) \tag{3.2}$$

.

This simplification is where the "naive" in naive Bayes come from, and allows use to easily create a classifier that will classify future data based on the original 500 labelled files.

$$\sum_{j=i}^{w} \log \frac{Pr(word_j|L)}{Pr(word_j|\bar{L})} \tag{3.3}$$

To classify new files a window of size w is scanned across the document producing $n - w$ samples. Each sample is then labelled based on the larger of the two conditional probabilities in Equation 3.1 and 3.2. Because of numerical instability the log likelihood ratio is used instead (Equation 3.3). The sample is then labelled L if the log likelihood ratio is greater than 0 and $\bar{L}$

Accuracy	99.3%
False Positive Rate	0.7%
False Negative Rate	0.001%

Table 3.1: License section classifier results.

otherwise. Equation 3.4 is used to fill gaps in the labelling of the license sections. The L labelled sections are then passed to the sentence parser.

$$label_i = \begin{cases} L & \text{if } label_{i-1} = label_{i+1} = L \\ \bar{L} & \text{if } label_{i-1} = label_{i+1} = \bar{L} \\ label_i & \text{otherwise} \end{cases} \tag{3.4}$$

Using 10-fold cross validation the 500 training files were used to determine the accuracy of the naive Bayes license section classifier. Table 3.2.2 gives the observed accuracy, precision, and recall.

3.2.3 Sentence Parsing

We found that breaking the license text into sentences allowed for efficient comparisons between different license templates. A family of licenses may share many of the same sentences, but have one defining sentence. By locating this defining sentence, the license may be classified. We used a maximum entropy sentence parsing algorithm [16] to extract sentences from the templates and license sections.

There are existing MaxEnt models, for example [12] which is trained on the text from the Wall Street Journal, but these model do not account for the law speak and ascii decoration that exist in licenses found in source code. Because current models lack domain specific knowledge required for parsing of sentences in licenses and source code, so we created our own MaxEnt sentence parsing model.

To create the model we labelled 176 licenses files. We used a simple markup tagging system to denote the beginning and endings of sentences. Figure 3.2.3 gives a brief example of a labelled

```
<sentence>Academic Free License Version 1.1
</sentence>
<sentence>The Academic Free License applies to any original work
of authorship (the "Original Work") whose owner (the "Licensor")
has placed the following notice immediately following the copyright
notice for the Original Work:</sentence><sentence> "Licensed under
the Academic Free License version 1.1."
</sentence>
```

Figure 3.3: Example of a labelled sentence used for training a Maximum Entropy sentence parsing model.

file.

To train the MaxEnt classifier the license section are tokenized into alpha-numeric and non-alpha-numeric tokens and the following features are extracted:

- stemmed word (f_1).
- first letter capitalized (f_2).
- whole word capitalized (f_3).
- includes numbers (f_4).
- if not alpha-numeric then a hash of the characters is used (f_5).

A windowing technique, as described in the previous section, is used to create a feature vector, i.e. the set of features for window i is $x_i = \left\{f_{1,word_i}, \ldots, f_{5,word_i}, f_{1,word_{i+1}}, \ldots, f_{5,word_{i+w}}\right\}$. We would like to know the conditional distribution $Pr(BREAK|x_i)$. With constraints $\sum_{i=1}^{n} Pr(x_i|BREAK)f_k(x_i) = F_k$ and $\sum_{i=1}^{n} Pr(x_i|BREAK) = 1$. This distribution can be found by maximizing its entropy $Ent(m) = -\sum_x E[x]m(BREAK|x)\log m(BREAK|x)$, where m is a possible conditional distribution function. The optimal m can be found by introducing the Lagrangian and finding the point that satisfies the Karush-Kuhn-Tucker conditions. The solution to the optimal conditional distribution is given in Equation 3.5, with Z as in Equation 3.6. Numerical methods are used to solve Equation 3.7 for the λs.

$$Pr(BREAK|x) = \frac{1}{Z(\lambda_1, \ldots, \lambda_m)} \exp\left[\lambda_1 f_1(x) + \cdots + \lambda_m f_m(x)\right] \tag{3.5}$$

$$Z(\lambda_1, \ldots, \lambda_m) = \exp\left(\sum_i \lambda_i f_i(x)\right) \tag{3.6}$$

$$F_k = \frac{\partial}{\partial \lambda_k} \log Z(\lambda_1, \ldots, \lambda_m) \tag{3.7}$$

Equation 3.5 can then be used to label sentence breaks in unseen sentences. We use an open source implementation for calculating the MaxEnt model written by Zhang Le [12], which includes both Python and C/C++ libraries. To test the resulting classifier we used 10-fold cross validation to get an observed accuracy based on our dataset. The sentence parser has an observed accuracy of 82.5%. Although this accuracy is low it is reliable, and tends to produce consistent parsings of the same classes of licenses.

3.2.4 Sentence Based Classifier

The sentences must be converted into a form suitable for quick comparisons. We use a text frequency vector to capture the structure of the sentences. Each sentence is converted into a text frequency vector and normalized to length 1. Sentences can now be compared with each other using the cosine similarity measure described in Equation 3.8. Notice how normalizing the vectors makes the equation simpler.

$$\cos\theta = \frac{sv_i \cdot sv_j}{\|sv_i\|\|sv_j\|} = sv_i \cdot sv_j \tag{3.8}$$

The vector space model allows for quick comparison of similar sentences. To generate the sentence vectors, we start by splitting the sentence into word tokens, removing "stop words" such as "and", "the", and "a", and converting the remaining words to their respective roots using a Porter stemming algorithm. We then create a vector in the space spanned by all possible roots to represent the sentence, with each component representing the frequency that root appears in the sentence. We define the similarity between sentences to be the cosine-similarity between the respective vectors.

In order to quickly calculate the nearest neighbours of a target sentences we create a database of reference license sentences. Single-link clustering technique described in [13]. New sentences are then compared to the top level cluster nodes which minimize the number of dis-similar sentences that must be compared to. The new sentences is then moved down the tree until a sufficient level of similarity is met(in our case dis-similar by two words). This helps to quickly remove large numbers of sentences that are dissimilar to the new sentence. The comparison can now be calculated without having to calculate distances between all n reference sentences in the database.

To analyse a particular license section, we extract the sentences and convert them into term-frequency vectors, as above, again preserving sentence order within the section. We use the reference license database of sentences to compare unknown licenses to known licenses. D similarity matrices are created between the unknown document and the reference licenses, where D is defined as the number of reference licenses. Similarity matrix i is of size m by n_i, where m is defined as number of sentences in the unknown license and n_i is as the number of sentences in the reference license i. The entries of the similarity matrix correspond to the cosine similarity of sentences in the unknown license and the sentences in the ith reference license. The similarity matrix is converted into a binary matching matrix, where an element of the matrix is 1 if the similarity of the sentences is greater than 0.9, determined by 10-fold cross validation, and 0 otherwise. Each matrix is given a score based on the longest common subsequence of similar sentences, see Equation 3.9, where $matched\,()$ is the length of the longest common subsequence. This score is then normalized to be between 0 and 1. If the largest score is not greater than 0.9 then we flag it for a human to examine. Otherwise we name it based on the reference license with the highest scoring matrix. License sections that do not match any template licenses are reported as "unknown license"s.

$$s = \frac{(}{2} * matched\,(d))(matched\,(d) + n_i) \tag{3.9}$$

3.3 Competing Approaches

Few software solutions that locate licenses in project source code exist, and these can be broken into three types: closed source commercial algorithms, the Longest Common Substring (LCS) approaches and the rule based approaches. Two companies, Palimida (http://palamida.com) and Black Duck Software (http://blackducksoftware.com), provide license analysis services. The Fossology project has considered using these services but determined that the current algorithm, bSAM [8], is comparable to services provided by Black Duck and Palimida.

The LCS approaches use a relaxation of the LCS problem to find a large string belonging to a template license in a piece of source code, such as [9]. This matching string is then given the same name as the corresponding template license. The full text comparison results in an algorithm that is accurate but very computationally expensive, because each reference license comparison is and $O\left(n^2\right)$ operation based on the number of tokens in each licence.

In contrast, the rule based approaches employ small phrases, sequences of unique words, or regular expressions to capture definitive features of a license. Rule based approaches, although fast, only locate a small string and disregard the majority of the license text. The text ignored may define further rights and restrictions that embody similar but different licenses, thus leaving the algorithm prone to substantial errors. `ohcount` is a regular expression algorithm and works well, but maintaining it is set of hand crafted regular expressions is labour intensive and tedious.

Two mature license analysis tools, `bSAM` and `ASLA`, come from the LCS and rule based families, respectively. `bSAM` relies on technology which was designed for protein matching back in the 1980s. Since then, the protein matching community has moved from LCS-type algorithms to more probabilistic models that are faster and more accurate. `ASLA` [19] runs an order of magnitude faster than bSAM, but lacks full text matching, thus limiting its merit, and no tests on labelled data have been conducted on this algorithm. `OSLC` uses a diff technique like `bSAM` to locate software licenses. Table 3.4 shows that both algorithms have the same problem when scaling to large numbers of reference licenses.

3.4 Results

We tested our algorithm on 294 hand labeled license files without source code but including all the comment characters. All test files were marked as licenses with one exception, consisting entirely of "Copyright: GPL," which may not be a valid license. This phrase should have been marked as an unknown license, but GPL alone was not sufficient to trigger the license detector. The classification accuracy when marking unknown licenses as missed was 60%. As we designed the algorithm to refrain from classifying licenses that did not fully match template licenses, we removed these test files for an accuracy of 87%. Because of the large number of difference licenses our small set of reference licenses was missing templates for 30 of the license families present in the test corpus. After the addition of these 30 template license the raw accuracy changed from 60% to 92%.

The majority of the 24 test licenses that were miss classified are from the "GPL", "LGPL", "BSD" and "MIT" families of licenses. The misclassified "GPL" and "LGPL" licenses are reference licenses and only differ by a single sentence near the end of the license. This last sentence is the address of the Free Software Foundation, which changed after the writing of the General Public License version 2. Our license references have a mix of the two addresses in both the "LGPL" and "GPL" license templates. Since the only difference other than the address is the name of the license, which can be "Lesser General Public License" or "General Public License", this final sentence biases the results causing a misclassification. Of the misclassified licenses only 4 are from the "MIT" "BSD" family. These four licenses are so similar that there is still debate among experts from the Fossology project on the correct classification, thus ignored for now.

To capture the true differences between the "LGPL" and the "GPL" reference licenses we extracted noun phrases from the licenses as separate tokens. We used the same noun phrase tagger used in the copyright analysis algorithm from Section 2.1. By extracting noun phrases and treating them as separate tokens, like the ones in Table 3.4, they can be give weights based on there term frequency inverse document frequency, `tf-idf`. Since the phrase "General Public License version

‘On’	NP_O
‘Debian’	NP_B
‘GNU’	NP_I
‘XXXpathXXX’	NP_I
‘systems’	NP_I
‘,’	NP_O
‘the’	NP_B
‘complete’	NP_I
‘text’	NP_I
‘of’	NP_O
‘the’	NP_B
‘GNU’	NP_I
‘General’	NP_I
‘Public’	NP_I
‘License’	NP_I
‘can’	NP_O
‘be’	NP_O
‘found’	NP_O
‘in’	NP_O
‘XXXpathXXX’	NP_B
‘.’	NP_O

Table 3.2: Example of Noun Phrase tags

	Bates	ninka	bSAM	ohcount	OSLC
Correct	221	200	137	83	57
Incorrect	6	7	112	167	193
Unkown	23	43	1	0	0
F-measure	0.938	0.889	0.708	0.498	0.371

Table 3.3: Example of Noun Phrase tags

2" is less frequent and more important than the individual words in the noun phrases it should bias the similarity of sentences more than individual tokens.

We tested against the test set and found that the 20 "LGPL" and "GPL" licenses had been correctly classified correctly. It should also be noted that the address were removed from the template licenses. Although we greatly improved on the test set this result is invalidated, because we peeked at the test set in order to make improvements. Therefore, a new test was conducted on a set of licenses randomly chosen from Debian distribution by D. German et. al. [7]. This data set consists of 250 files which include both source code and license text. The results of D. German et. al. algorithm `ninka`, `bSAM`, `ohcount`, `OSLC`, and our algorithm `Bates`, can be seen in Table 3.4 along with the results of our algorithms analysis. Our algorithm attains a better accuracy and f-measure than the other four algorithms.

The algorithm shows a linear proportionality in time verses amount of license data analysed. Figure 3.4 shows the tested speed on the test set. While the trend is 1.2 minutes per megabyte, the analysis of real source code will be faster since it is less dense in license data.

3.5 Current Status

‘ The current algorithm has been prototyped in 1500 lines of Python code and is being actively converted into C and integrated into the Fossology project. The goal is to replace `bSAM`, the current full text license analysis algorithm, in the 1.3 release of Fossology. The ability to quickly

Figure 3.4: File size vs Performance

add new template licenses and the overall speed and accuracy improvements mark our algorithm as a candidate for inclusion into the Fossology project.

Chapter 4

Conclusion

We have demonstrated a reliable system for automating the task of locating and classifying licenses in software. The naive Bayes classifier used to locate the license sections gives 99.3% accuracy, with low false positive and false negative rates. The sentence parser provides an accuracy of 82.5%, even with the limited number of training files. This provides a very sturdy foundation for the sentence based classifier to determine the correct license(s). With noun phrases the license classifier has the power to correctly classify both full text licenses and reference licenses. The noun phrases also help increase the probability that unknown modified sections of licenses will be labelled as unknown licenses.

The experimental results show that the license classifier is more accurate and robust than the competition. This is the reason that the Fossology project is spending time to integrate our solution into their software. We hope to have our algorithm fully functional by the release of the 1.3 version of Fossology. This is great new for the open source community and users of FOSS, because our algorithm will help reduce cost and risk to the users of FOSS. Thus, allowing more opportunity for people to use the software, and contribute tot eh projects.

The copyright analysis algorithm is also a great addition to the open source community. Copyright analysis was one of the most requested tools on the Fossology mailing list. The current bi-gram classifier being used in Fossology currently is very fast and produces adequate results but returns a large number of false positives. This was done on purpose to help prevent false negative, and missing important copyright information. The copyright analysis algorithm that uses bi-grams

and noun phrases, suffers less effects from false positives and provides tighter fits to the copyright statements. The noun phrase version of the copyright algorithm is already being incorporated into the development branch of Fossology, and should be finished before the 1.3 release.

Chapter 5

Future Work

The copyright analysis algorithm yields an accuracy of 98%. This is very good, but there is still room for improvement. The amount of training and testing data can be increased to provide an even better span of the copyright statements that exist in the wild. This algorithm itself can be improved. The noun phrases that are used to determine the endings of the copyright statements are really named entities. Analysis of the structure of named entites in relation to copyright statements might yield patterns in proximity and frequency relative to copyright statements. This information could be used to create an even better and faster algorithm that could locate copyright statements and copyright holders.

License analysis is very important to the Fossology project. Recently, a new regular expression license analysis algorithm, from Hewlett Packard, has been added to the Fossology project. This algorithm has been in development for over 7 years and has an extensive database of rules for classifying licenses. There maybe advantages to using this new algorithm to classify the licenses that our algorithm does not recognize, thus minimizing the number of unknown licenses returned. The way that the license analysis algorithm matches licenses has potential to help other domains. More work is needed to determine error bounds on the relaxation of the real valued similarities. The method of clamping the similarities may allow for decreasing the error incurred by the relaxation, and help to get better matches.

Bibliography

[1] Steven Abney. Part-of-speech tagging and partial parsing. In Corpus-Based Methods in Language and Speech, pages 118–136. Kluwer Academic Publishers, 1996.

[2] Kirk St. Amant, Kirk St.Amant Still, and Brian. Handbook of Research on Open Source Software: Technological, Economic, and Social Perspectives. IGI Global, 2007.

[3] Christian Bird, David Pattison, Raissa D'Souza, Vladimir Filkov, and Premkumar Devanbu. Latent social structure in open source projects. In SIGSOFT '08/FSE-16: Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering, pages 24–35, New York, NY, USA, 2008. ACM.

[4] M. O. Dayhoff and R. M. Schwartz. Chapter 22: A model of evolutionary change in proteins. In in Atlas of Protein Sequence and Structure, 1978.

[5] M. O. Dayhoff and R. M. Schwartz. Chapter 22: A model of evolutionary change in proteins. In in Atlas of Protein Sequence and Structure, 1978.

[6] Massimiliano Di Penta and Daniel M. German. Who are source code contributors and how do they change? In WCRE '09: Proceedings of the 2009 16th Working Conference on Reverse Engineering, pages 11–20, Washington, DC, USA, 2009. IEEE Computer Society.

[7] Y. Manabe D.M. German and K. Inoue. A sentence-matching method for automatic license identification of source code files. ASE, 2010.

[8] Robert Gobeille. The fossology project. In MSR '08: Proceedings of the 2008 international working conference on Mining software repositories, pages 47–50, New York, NY, USA, 2008. ACM.

[9] Paul Heckel. A technique for isolating differences between files. Commun. ACM, 21(4):264–268, 1978.

[10] Daniel Jurafsky and James H. Martin. Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition. Prentice Hall, second edition, February 2008.

[11] Ron Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. pages 1137–1143. Morgan Kaufmann, 1995.

[12] Zhang Le. Maximum entropy modeling toolkit for python and c++, 2004.

[13] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. Introduction to Information Retrieval. Cambridge University Press, 1 edition, July 2008.

[14] Gueheneuc Y. Massimiliano Di Penta, Daniel M. German and G Antoniol. Tracking the evolution of software licensing: An empirical study, 2010.

[15] Lance Ramshaw and Mitchell Marcus. Text chunking using transformation-based learning, 1995.

[16] Jeffrey C. Reynar and Adwait Ratnaparkhi. A maximum entropy approach to identifying sentence boundaries. In Proceedings of the fifth conference on Applied natural language processing, pages 16–19, Morristown, NJ, USA, 1997. Association for Computational Linguistics.

[17] M.O. Dayhoff R.M. Schwartz and B. C. Orcutt. Chapter 23: Matrices for detecting distant relationships. In in Atlas of Protein Sequence and Structure, 1978.

[18] R. Schwarz and M. Dayhoff. Matrices for detecting distant relationships. page 353358, 1979.

[19] Timo Tuunanen, Jussi Koskinen, and Tommi Karkkainen. Asla: Reverse engineering approach for software license information retrieval. In CSMR '06: Proceedings of the Conference on Software Maintenance and Reengineering, pages 291–294, Washington, DC, USA, 2006. IEEE Computer Society.

[20] Liguo Yu and Srini Ramaswamy. Mining cvs repositories to understand open-source project developer roles. In MSR '07: Proceedings of the Fourth International Workshop on Mining Software Repositories, page 8, Washington, DC, USA, 2007. IEEE Computer Society.