FreeMeLegal - A Recommender System for Open Source License for a software project

Final Project Report

Vineet Kumar
School of Computer Science
McGill University
vineet.kumar@mail.mcgill.ca

ABSTRACT

Software licensing is a very important part of any software. Big corporations spend millions of dollars to carefully carve their licenses and to protect them. However for open-source software most of the developers are non-funded and work for interest and fun. In such a scenario and specially when corporations have started playing big roles in open-source development it becomes really important that the developers get the exact control they require on their software and get due credit and profit if they want. However most of the developers are not interested in, have poor knowledge of and do not spend time on these issues. In this project I have developed a semi-automated recommender system that, based on developer's requirements of control on his software and a description of the software, it suggests 1-3 open-source software licenses that the developer may choose to use. FreeMeLegal is based on two recommendation engines. First one is a knowledge-based recommender system. It takes users' intents about the degree of freedom that they would like to provide with their software. Second recommender is a kNN algorithm based recommender.It takes as input the attributes of the software like its category, programming language, supported operating systems and intended audience and finds the nearest neighbours from some of the popular open source software. The licenses of these neigbours are used to fine-tune (filter and/or sort) the results from the first intent based recommender engine.

Categories and Subject Descriptors

p.2 [**Recommender Systems for Software engineering**]: Project

General Terms

Design, Documentation

Keywords

Recommender Systems, Open Source, License

1. MOTIVATION

Most of the Open source software developers do not really like spending a lot of time deciding which license is most suitable (and in some cases allowed) for their software. The Open Source Initiative (OSI) has listed about 70 approved open source licenses that can be used for open source software (http://www.opensource.org/licenses/category).This number of choices makes it even more difficult to choose a correct license for a software product and it is nearly impossible to read and understand the legalities in these licenses. As a result, developers often end up using a commonly used license like GPL or BSD without even understanding the implications they may have. Sometimes developers reuse code from a software protected by a particular license like Apache which makes it mandatory for the developers to release their code under the same license and developers often unknowingly ignore this binding. With the amount of open source development done today and the increased participation of companies like IBM, Google, Red Hat, etc. in the open source community it has become even more important for independent and non-funded developers to use a license that protects their rights and makes sure that they do not lose due credit and/or profit for their work. For an example, Sun Microsystems released parts of their implementation of Java under GNU Public License which is a commercial-friendly license which allows Java to be used for products that are sold commercially. For an enterprise, this was probably an informed decision but for an independent developer who unknowingly uses GPL would not be able to make money for his work even though, using a license like MySQL license or sleepyCat license would allow him to open his code yet earn for its commercial use by others. FreeMeLegal is an attempt to help developers easily and quickly choose correct license that would fit to their requirements.

2. HIGH LEVEL DESIGN

Figure 1 shows the high level architecture of FreeMeLegal recommender system. It is a web-based application. In the first page it shows a web-form that contains questions related to the "intent" of the developers regarding the degree of freedom they want to give to the users of their open code. These questions are presented as radio-buttons and checkboxes to embed some constraints in the user interface itself. This user input first goes through a set of filters that handle the highly-restrictive constraints such as the use of certain copyleft licenses which bind the developers to release their work under those licenses only. If there are no such

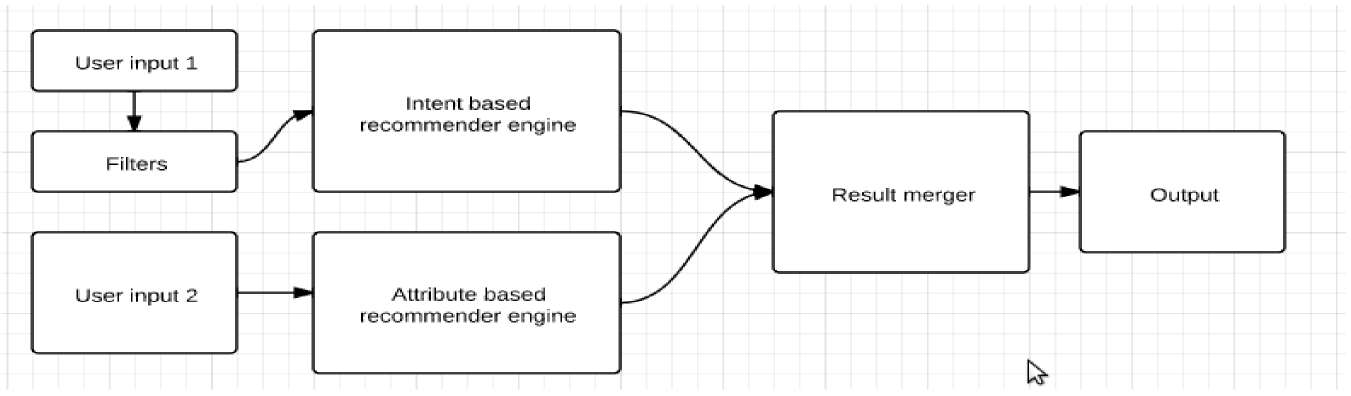

User input 1
Filters
Intent based recommender engine
User input 2
Attribute based recommender engine
Result merger
Output

Figure 1: High level design of the recommender

highly-restrictive constraints then the user input is processed by the intent-based recommender engine which is based on the constraint-based type of knowledge-based recommendation technique as described in [9]. The knowledgebase for this recommender consists of a database table of licenses and their properties. This table is prepared by parsing the wikipedia page listing a comparison of various open source licenses and corresponding wikipedia pages for each of the licenses listed in this comparison. This recommender engine is discussed in more detail in section 2.1. This intent based recommender suggests a list of open source licenses based solely on the permissions granted by the licenses that match best with the users' intents and does not have any means of sorting the results or removing an extra number of suggestions from the result set or to decide how suitable each suggestion is. To solve this problem FreeMeLegal uses a second fine-tuning recommender engine that takes into account the nature of the software, for example whether it is a library or a database software or a multimedia software, etc. and finds existing popular projects that are similar in nature and alters the results from the intent recommender based on the licenses of these similar projects. The line of reasoning being that if a software is related to a particular group of projects or will be used mostly by a particular community then it is a good idea to use a license used most by that community since it will reduce the chances of potential license incompatibility issues in future. For example if your project is a Java project closely related to Eclipse IDE it may be a good idea to use the Eclipse license since a lot of your project's users might be eclipse developers or use your work to develop eclipse plugins so by using a license same as most of those projects there is a less chance that your work will run into any license incompatibility issues. This second attributes-based recommender engine uses the kNN algorithm to find the existing popular projects based on sourceforge.net data which are similar to the attributes as described by the user. The database for this recommender consists of a set of equal number of projects for each license and for each category. This recommender is described in more detail in section 2.2. Please note that one of the assumptions that this tool makes is that the users have read and agreed to the terms of licenses of any software that they might be using in their project.

2.1 Intent-based classifier (Primary recommendation engine)

Intent-based classifier serves as the primary recommendation engine that recommends a list of open source licenses that a user may use based on the degree of freedom (or permissions) granted by a license that best suit the degree of freedom that the software developer wants to provide with his/her software. This recommender works in following three steps 1)Getting input from the user about how much degree of freedom is he/she willing to provide. This is important since some of the constraints of the system are captured by using checkboxes and radio-buttons in the HTML form itself. 2)Filters that check for highly restrictive constraints and 3)The actual constraint-based recommender core. These three parts are described below in detail.

2.1.1 User input for intent-based classifier

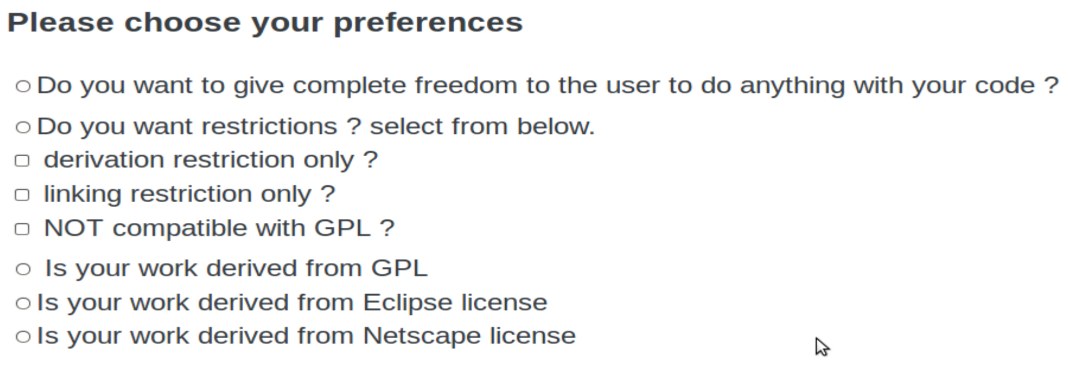
Figure 2 (screenshot from the HTML page) shows the intent questions that the user needs to answer to show his/her intended degree of freedom. Some of the constraints are embeddedin the HTML form itself. For example if the user wants to give complete freedom for his software that implies that there can be no other restrictions hence the radio buttons for these two options. The last three radio buttons correspond to the major copyleft licenses and serve as the input for the filters. Please note that compatibility to GPL is considered important since it is the most widely used open source license and some of the major software like GNU/Linux and GCC use this license.

2.1.2 Filters

Figure 3 depicts the logic for a filter for copyleft derivative work. The idea is basically that if there is some kind of superceding restrictions caused by the use of a software with a restrictive license then the user has to legally abide by them and has little choice over the license for his/her product. This part of the recommender is majorly influenced by software licensing restriction issues discussed in [7]

2.1.3 constraint-based recommender core

After applying these filters recommender engine follows the constraint-based approach of solving knowledge-based recommender problems as described in chapter 4 of [9]. In the first step, License data (V_P) is collected and stored in the


Please choose your preferences

○ Do you want to give complete freedom to the user to do anything with your code ?

○ Do you want restrictions ? select from below.

▢ derivation restriction only ?

▢ linking restriction only ?

▢ NOT compatible with GPL ?

○ Is your work derived from GPL

○ Is your work derived from Eclipse license

○ Is your work derived from Netscape license

Figure 2: Questions presented to the user for intent-based recommender

Table 1: Sample License data table

License	link with diffт. license	changes under diffт. license	GPL compatible	Copyleft	Copyfree
Apache License	Yes	Yes	Yes	No	No
Apple Public Source License	Yes	No	No	Yes	No
BSD license	Yes	Yes	No	No	No
GNU General Public License	No	No	Yes	Yes	No
GNU Lesser General Public License	Yes	No	Yes	Yes	No

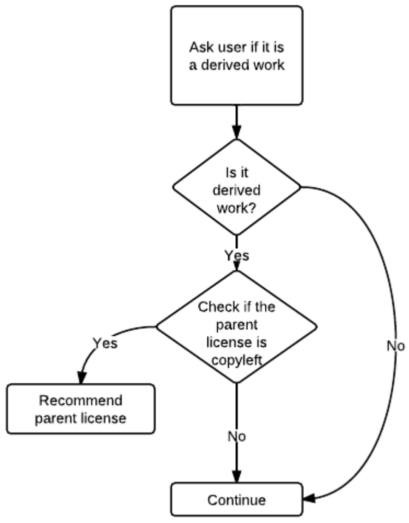

Ask user if it is a derived work
Is it derived work?
Yes
No
Check if the parent license is copyleft
Yes
Recommend parent license
No
Continue

Figure 3: filter for copyleft derivative work

form of a database table where each row corresponds to a license and each column corresponds to a property. We consider five important permission properties of each license. Table 1 depicts the entries for 5 popular open source licenses in this table. This data is collected and stored in a database by a PHP script that parses HTML for [13] and the corresponding wikipedia entry for each of the listed licenses. Please note that it was still necessary to manually verify and prune the generated dataset. As described in section 2 chapter 4 of [9] we define the constraint satisfaction problem by a tuple (V, D, C) where V is a set of variables, D is a set of domains for V and C is a set of constraints that define the combinations of values V can simultaneously take. For this classifier we define V as $V = (V_C \cup V_P)$ where V_C (shown in Table 2) is the set of answers given by the users and V_P (Shown in Table 3)is the set of license properties stored in the data table. We describe the following constraints

C_F as the relationship between user properties and license properties for example a user's intent to not allow changes under a different license implies license property of copyleft.(Shown in Table 4)

C_R as the compatibility constraints between user requirements. For example a user cannot give full freedom and yet restrict on releasing changes under different licenses. Please note that this set of constraints is embedded in the user interface itself and thus not implemented separately.

C_P as the combination of properties of available assortment of licenses.

REQ as the set of answers given by a particular user or the actual values of V_C.

Finally we define the set of constraints C as $C = (C_F \cup C_R \cup C_P \cup REQ)$ In case a user does not select anything

Table 2: List of customer variables

V_C
Free
Restricted
–Linking restrictions only
–Derivative restrictions only
–Both restrictions
GPL compatibility
Copyleft derived

Table 3: List of product variables

V_P
Linking allowed
GPL compatible
Copyleft
Copyfree

the default is taken as ”complete freedom” since the system assumes that the user does not care about the restrictions. Since the data for this recommender is binary in nature (i.e it can take only YES or NO) the obvious approach to calculating answers was to generate database queries that represented the constraints on the data.

2.2 Project attributes-based classifier (Secondary classifier)

Project attributes-based classifier is the second stage recommendation engine which sorts and fine tunes the results generated by the primary classifier on the basis of project attributes like area of application of project, programming language/s used for implementationt,intended audience for the project and supported operating systems. This recommender uses the kNN algorithm to find similarity between user's project and the projects in the dataset.

2.2.1 dataset

The dataset for this part of FreeMeLegal uses sourceforge.net as its source of data primarily because it is one of the largest and most widely used open source software repository and it has a very good categorization of the projects based on various attributes of the projects. A PHP script is used to scrape the HTML pages of sourceforge.net and stored in a database table. Figure 4 shows a snapshot of this database table. This table stores six most popular projects for each license for each category. Currently it has 13 licenses and 10 categories. Please note that the size of data is limited due to restricted resources and time (in comparison to the number of pages to be fetched and parsed).

2.2.2 recommender logic

For this part of the recommender, users have to first fill a HTML form where they provide the various attributes of their project - Category (based on the area of application), supported Operating system platforms, programming languages used for implementation and intended audience for the project. Figure 5 shows this form.

These selections/answers are used to calculate the similarity between the user's project and the projects in the database. First a query like

```
SELECT * FROM ‘TABLE 2‘ WHERE category=$category
```

is used to select only those projects that belong to user's category, then for each entry in the dataset a distance is measured by calculating the euclidean distance of weighted Jaccard index for each of the three remaining attributes of the project.Weighted Jaccard index is defined as follows

$$J(A,B) = W_a \frac{|A \cap B|}{|A \cup B|}$$

where W_a is the weight assigned to the attribute which is 2 for the programming language used, 1 for the intended audience and 1 for the supported operating systems. A is the set of attribute values of the user project and B is the set of the attribute values of the data entry. The euclidean distance D between two projects is calculated as follows

$$D = \sqrt{\sum J^2(A_i, B_i)}$$

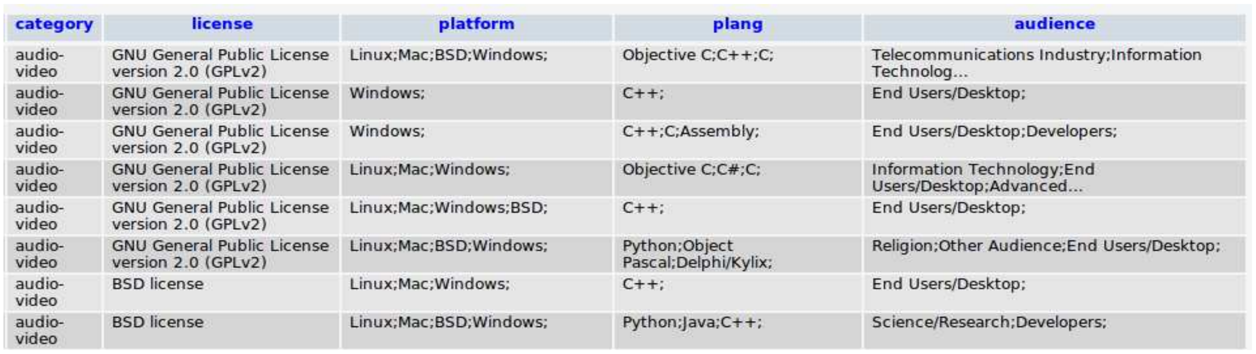
The euclidean distance in this case actually represents the similarity between user's project and the project in the data. The list of projects in the dataset with their corresponding license is stored in an array and is sorted by the similarity from higher similarity to lower similarity. Two projects with same similarity are implicitly sorted in the order of their popularity. Value of 'k' is chosen to be a maximum of 5. 5 is chosen since we recommend a maximum of 3 licenses as the final result after merging the results of the two recommender engines. Since the merge operation is basically an intersection of two sets of results, a value less than 5 usually does not give 3 recommendations after merging and a value larger than 5 will include not-so-near neighbours too. An array of unique licenses corresponding to the resultant set of neighbours is the final output of the attribute-based recommender.

2.3 Merge operation

The merge operation is the set intersection of the licenses recommended by intent-based recommender and those recommended by the attribute based recommender engine. If

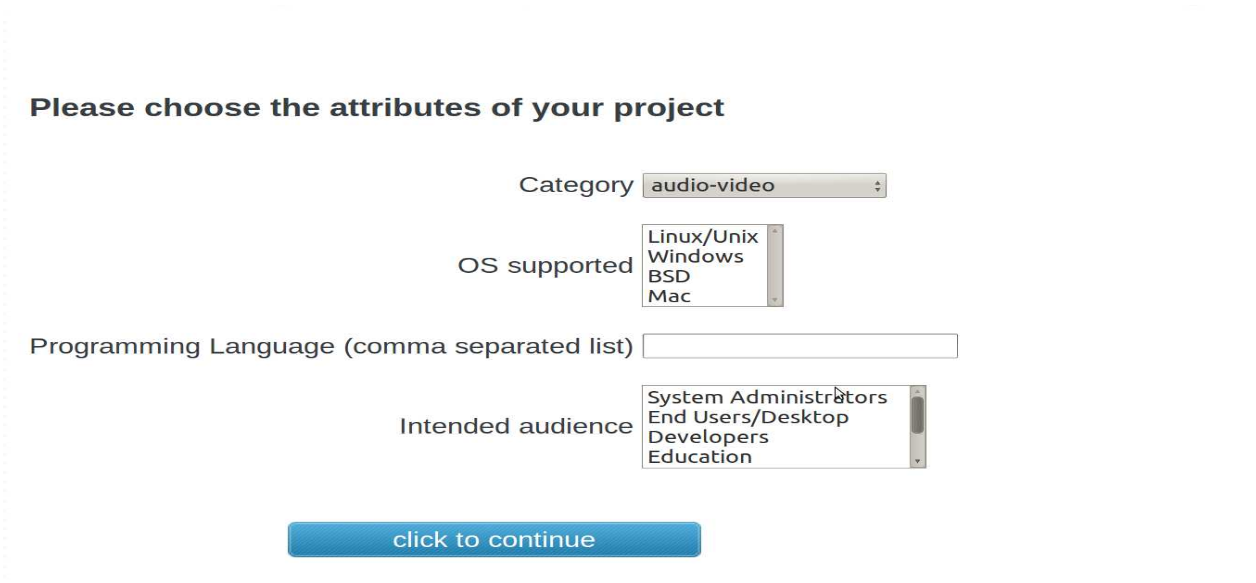
Table 4: Relation between user properties and license properties

C_F	
User properties	License properties
Free	Copyfree = Y
restricted	
–Linking only	Linking allowed = Y and Copyleft = N
–Derivative only	Copyleft =Y and Linking allowed = N
–Both	Copyleft = Y and Linking allowed = Y
GPL compatibility	GPL compatible = Y



category	license	platform	plang	audience
audio-video	GNU General Public License version 2.0 (GPLv2)	Linux;Mac;BSD;Windows;	Objective C;C++;C;	Telecommunications Industry;Information Technolog...
audio-video	GNU General Public License version 2.0 (GPLv2)	Windows;	C++;	End Users/Desktop;
audio-video	GNU General Public License version 2.0 (GPLv2)	Windows;	C++;C;Assembly;	End Users/Desktop;Developers;
audio-video	GNU General Public License version 2.0 (GPLv2)	Linux;Mac;Windows;	Objective C;C#;C;	Information Technology;End Users/Desktop;Advanced...
audio-video	GNU General Public License version 2.0 (GPLv2)	Linux;Mac;Windows;BSD;	C++;	End Users/Desktop;
audio-video	GNU General Public License version 2.0 (GPLv2)	Linux;Mac;BSD;Windows;	Python;Object Pascal;Delphi/Kylix;	Religion;Other Audience;End Users/Desktop;
audio-video	BSD license	Linux;Mac;Windows;	C++;	End Users/Desktop;
audio-video	BSD license	Linux;Mac;BSD;Windows;	Python;Java;C++;	Science/Research;Developers;

Figure 4: Snapshot of the dataset for attribute based recommender



Please choose the attributes of your project

Category	audio-video
OS supported	Linux/Unix Windows BSD Mac
Programming Language (comma separated list)	
Intended audience	System Administrators End Users/Desktop Developers Education

Figure 5: attributes form

this results in three, two or one recommendations then these results are presented to the user as the final results. If it results in more than three recommendations then the tp three are chosen in the order of the results in the attribute-based recommender. If the intersection results in no recommendations then the result of the intent-based recommender is shown as the final result. I would like to emphasize here that intent-based recommender is the primary recommender since the intent with regard to degree of freedom is the key factor in making a choice for the license to be used and that attributes-based recommender just helps to "fine tune" these results.

2.4 Explanations

[8] and [11] discuss importance and effects of providing exxplanations to the users of a recommender system. For a recommender system like FreeMeLegal where users might actually use the result recommendations and the results are supposed to be accurate it is highly useful to provide an explanation to the users as to why a particular result was recommended. This is required to increase users' trust in the system and to let them know that the results are not magically generated. Also it is important to give explanations about the inputs that the users provide. These explanations tell the user "what" information they need to provide, "why" is this information needed to help choosing a license and "how important" is this information for getting an accurate recommendation. The explanations related to input are mostly static and do not change with the user's input. They just help the user to provide more precise information about their project. The explanations for the results are however dynamic in nature and change depending on the choices made by the user in the input form for intent-based recommender. For example if a user tells that his/her project is derived from GPL license then he gets only GPL as the result - here the user must be given an explanation that since GPL is a strong copyleft license and any project derived from it or even linking to it must be released under GPL. Similarly for each choice the user makes on the input page, an explanation should be provided as to how does that choice affects the resultant recommendations. For the attribute-based recommender an explanation can be given to clarify on what basis is the recommendation made and why is it important, for example an explanation like "Your project uses Java as the programming language and it is intended for developers and it's category is development which suggests that your project is similar to the following popular projects and since they use the suggested license it is a good idea for you to use them to reduce any probable license incompatibilities in future". Users can get these explanations by clicking a link beside the results.

3. IMPLEMENTATION

This recommender system project is implemented as a web-based application. Backend implementation is done using PHP. MySQL is used as the database to store all the datasets. User interface consists of simple HTML forms. The user can see explanations about each choice that he may make or the results he/she gets by clicking on a link beside the field that will open a pop-up giving the explanation. First page presents user with the intent-based questions discussed above. Upon submission of answers to these questions the primary recommender will be invoked and results are stored in a database table and user is presented with the page that asks him/her to answer questions about project attributes. These questions are implemented as drop-downs and multiple selects for users to select from the given options. On submission of this form the secondary recommender is invoked and the user is finally navigated to the page which presents him/her the final results. Figure 6 shows a schematic diagram of the user interface. System is currently online at http://cloudmedo.com/762

4. EVALUATION

The evaluation of the whole system is done in three phases. In the first phase only intent based recommender is evaluated. Here the aim is to find how accurate the results are. This evaluation is done manually on different sets of possible choices on the form. First a set of intent values are submitted and then each of the licenses in the results is manually compared to the actual values corresponding to those licenses. If these values match and for all the licenses in the result set then it can be concluded that the particular choice of intent values gives accurate results. In the second phase of evaluation the attribute-based recommender is evaluated. In this a project is randomly selected on sourceforge.net which is not present in our database. The input given is the same as that listed on the sourceforge.net page for the project. The output recommendations list should contain the actual license of this project The third phase of evaluation is the black-box evaluation which also implicitly evaluates the merger. Here, like in phase two, a project is selected at random. for the input to the first form choices are selected that represent the actual license used by this project. The input to the second form are given as in second phase. The final result should include the license that is actually used by this project. If the actual license is not included in the result then we need to see if the recommended license also fulfills all the required criteria. The aim of this project is not just to recommend a license that is more famous but a license that is correct. The whole evaluation is repeated for 10 different projects with varying characteristics.

5. THREATS TO VALIDITY

FreeMeLegal is still a very basic system that tries to combine two approaches to recommend an open source license which are based on users' inputs that are orthogonal in nature. These approaches are based on very high level properties of the system and the licenses. Software licenses can not always be fully contained into a combination of these properties. For example, even if two licenses allow linking with a code under different license they may have unique special conditions applied to this clause. The users must still review the license thoroughly before using it. This project's aim is to reduce the number of licenses that the user may have to review and specially for individual developers who are not affected to a great extent by these special clauses. Another important issue in software licensing as discussed in is about licensing restrictions when a project derives from or uses or even links to another project or library and it becomes even harder when the used software itself uses a project under some other license. This may lead to incompatibilities that are hard to detect. Resolving this issue is still a huge topic for research. FreeMeLegal is still very naive at handling these issues and addresses very limited and major restric-

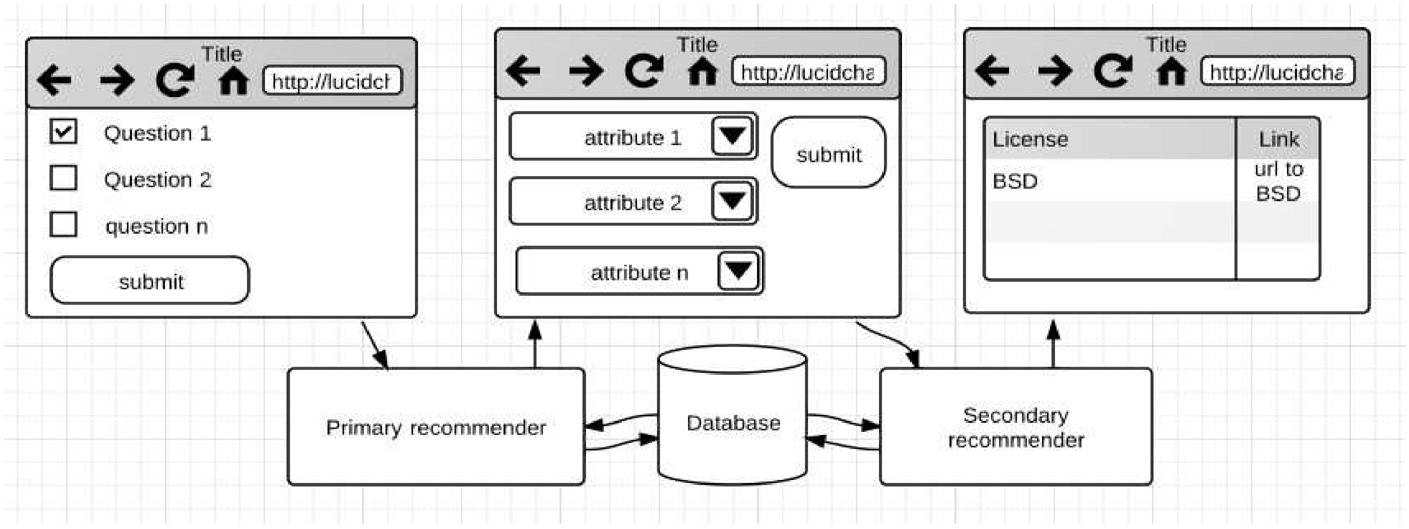


Figure 6: UI scheme of the recommender system

tions only through explicit filters. The database used by FreeMeLegal still needs some manual tuning and an increase in size of the data for attribute-based recommender. Size of the database is currently limited due to limited resources to scrape the data. Gathering the current data itself required to send a request to sourceforge.net every 3 seconds continuously for more than forty eight hours and several restarts of the script.The project also needs more extensive evaluation and possibly some feedback from the users themselves specially about explanations.

6. FUTURE WORK

In short term there are three major things that need to be done. First, a through manual review of the database and increase it to include more licenses, more categories and more projects. Second, an extensive evaluation with more number of projects and more number of combination of choices. Third, develop a more explicit, informative and dynamic explanations for every choice and every result. In long term, constraints based on finer details of specific licenses also need to be taken into account specially those that affect compatibility with software under other licenses.

7. RELATED WORK

One of the first works on knowledge-based recommender systems was presented by Burke (2000)[1] and Burke et al. (1997)[2]. They discussed knowledge-based recommendation systems for cars, movies, restaurants and consumer electronics. Felfernig and Burke (2008)[5] present constraint-based recommender systems in the domain of financial servces. Felfernig et al. (2006-2007) in [6] discuss in detail the VITA Financial Services Sales Support Environment which is a recommender system that helps sales representatives to suggest financial products to customers. VITA is a classical example of constraint-based recommender systems which has been discussed in [9] and the intent-based recommender engine is highly influenced by it. Felfernig and Burke (2008)[5] introduce a categorization of principal recommendation approaches and provide a detailed overview of constraint-based recommendation technologies and their applications. Zanker et al. (2010)[14] formalize different variants of constraint-based recommendation problems and empirically compare the performance of the solving mechanisms. This project is highly motivated by the works of Burke and Felfernig specially. In the field of open source licenses there is a lot of structured documentation available on various websites. Ed Burnette explains a simple process to choose an open source license among a few famous licenses in his two-part article on howto pick an open source license at [3] and [4] respectively. This method is also the basis of filtering algorithms for intent-based classifier in this project. German and Hassan have done some interesting research on license incompatibility issues. In [7] they have discussed how much a problem can license mismatching be and suggest ways to deal with it. They have also presented a mathematical model for representing license properties and their relations with other licenses. Wikipedia has a well structured comparison of a large number of open source licenses at [13] which is also a main source of data for this project. Wikipedia also has detailed information page about each of these licenses. [12] has a big repository of very popular open source projects and has a very detailed categorization of projects based on various attributes. This project uses the categories and attributes defined there. Last but not the least, User interface is a very important part of this project since capturing user requirements properly influences the results to a great extent. [10] and discuss the importance and affects of User interfaces and describe various types of user interfaces.

8. REFERENCES

[1] R. Burke. Knowledge-based recommender systems. *Encyclopedia of Library and Information Science*, 69(32):180–200, 2000.

[2] R. Burke, K. Hammond, and B. Young. The findme approach to assisted browsing. *IEEE Expert*, 4(12):32–40, 1997.

[3] E. Burnette. Howto: Pick an open source license (part 1). `http://www.zdnet.com/blog/burnette/howto-pick-an-open-source-license-part-1/130`, June 2006.

[4] E. Burnette. Howto: Pick an open source license (part 2). `http://www.zdnet.com/blog/burnette/howto-pick-an-open-source-license-part-2/131`, June 2006.

[5] A. Felfernig and R. Burke. Constraint-based recommender systems: technologies and research issues. In *Proceedings of the 10th International Conference on Electronic Commerce (ICEC '08) (Innsbruck, Austria)*, pages 1–10. ACM, 2008.

[6] A. Felfernig, K. Isak, K. Szabo, and P. Zachar. The vita financial services sales support environment. In *Proceedings of the 22nd National Conference on Artificial Intelligence (AAAI '07)*, pages 1692–1699. AAAI, 2007.

[7] D. German and A. Hassan. License integration patterns: Dealing with licenses mismatches in component-based development. In *International Conference of Software Engineering (ICSE) 2009*. ICSE, may 2009.

[8] J. L. Herlocker, J. A. Konstan, and J. Riedl. Explaining collaborative filtering recommendations. In *Proceedings of the 2000 ACM Conference on Computer Supported Cooperative Work*, pages 241–250. ACM, 2000.

[9] D. Jannach, M. Zanker, A. Felfernig, and G. Friedrich. *Recommender Systems: An Introduction*. Cambridge Univ Press, 2010.

[10] B. P. Knijnenburg, N. J. M. Reijmer, and M. C. Willemsen. Each to his own: how different users call for different interaction methods in recommender systems. In *Proceedings of the 5th ACM Conference on Recommender systems*, pages 141–148. ACM, 2011.

[11] J. V. S. Sen and J. Riedl. Tagsplanations: explaining recommendations using tags. In *Proceedings of the 14th International Conference on Intelligent User Interfaces*, pages 47–56, 2009.

[12] sourceforge.net. Sourceforge. `http://sourceforge.net`.

[13] wikipedia. Comparison of free and open source software licenses. `http://en.wikipedia.org/wiki/Comparison_of_free_software_licences`.

[14] M. Zanker, M. Jessenitschnig, and W. Schmid. Constraints in constraint-based recommender systems. *Constraints Springer*, 15(4):574–595, 2010.