Recommending answers to technical forum questions

Gayane Petrosyan
School of Computer Science, McGill University
gayane.petrosyan@mail.mcgill.ca

ABSTRACT

Nowadays, Community Question Answering (CQA) is very popular and requires significant amount of effort to be maintained. The key problem is to satisfy all user requests. That is to say to help users to find answers to their questions. However, there are questions which have answers in publicly available resources, posted on the web. This paper presents a recommendation system that mines possible answers to forum questions. The system is intended to decrease the time to answer user questions, increase user satisfaction, make forum more reliable and responsive. The system mines technical resources such as manuals, user guides, documentations and etc. for some specific product; in order to answer the questions posted in that product support forum using algorithms of natural language processing and data mining. Technical questions and documentations from Eclipse and MSDN websites are considered as data for the recommender system. As the system deals with tasks, which make evaluation of the system with exact models impossible, an empirical model of evaluation is planned.

1. INTRODUCTION

Many software organizations such as Eclipse or Microsoft spend resources on maintaining product support forums. Forums have administrators and moderators. In case of MSDN each category in the forum has an assigned person who takes care of all questions to be answered. For MSDN the total amount of questions is around 1,5 million, with approximately 1000 new questions added each day[1]. For Eclipse, the total amount of questions is close to 800,000[2]. Taking into account these numbers it is apparent that maintaining forums is not an easy task.

On the other hand, product support forums try to use the knowledge of the community to answer the questions of the users. To encourage the community to be active and share their knowledge MSDN introduced some motivating ideas. MSDN motivates users by assigning medals for their achievements, giving them so called 'Recognition Points' for posting and answering questions. With their reward system they try to model the idea of an expert whose response should be more trusted than that of the newcomer. Besides, it is just pleasant to earn medals, even virtual. Therefore, users become enthusiastic to gain new medals and points. MSDN uses one more method to help users to find the answers to their questions. When a user types the title of the question MSDN recommends a set of already posted questions which seem similar to the previously posted questions. In case, when a similar question has been posted the user will get the answer right away and will not post new one. Thought the similar question retrieval algorithm seems not that flexible, it is one more step forward to help users.

Eclipse took a slightly different approach, but the goal is still the same. The company enables users to create their own "buddy list" which is basically the list of people whose response to the user is more reliable.

The reason why both Eclipse and Microsoft try to make user experience in their product support forum better is because they want to give the user a satisfactory service. If the user has unanswered questions about the product then he will be dissatisfied with the product.

The good news is that many technical questions posted in forums actually have answers in technical resources such as manuals, user guides etc. That is why the system presented in this paper is going to use this information to help users with their questions. The general approach of the system is to apply data mining and natural language processing (NLP) algorithms to the forum questions and to the technical resources for finding possible recommendations. Another source of information that is very useful to the algorithm is forum question categorization. Proposed system will use this categorization as a support for NLP algorithms to form the queries to search the technical resources.

The evaluation model for this system is to be an empirical case study with a group of domain aware people who will give a feedback on the suggestions of the system.

Of course the recommender system cannot cover all questions and the idea of community and sharing knowledge is still very important.

---

[1] Statistics taken from (on Feb 2012) http://social.msdn.microsoft.com/Forums/en/categories/

[2] Statistics taken from (on Feb 2012) http://www.eclipse.org/forums/

2. RELATED WORK

During recent years, along with the growth of Community Question Answering (CQA) web sites different issues have been raised which led to many published papers on those issues. Some of the papers try to address question-answer problems in forums. They try to find users who might be experts for particular question. For example, G. Dror et al[1] suggests that "user experience could be significantly improved if it could route the "right question" to the "right user". The method that researchers suggest in this paper is to use so called content signals such as the text and categories of questions and associated answers and social signals, which includes user actions such as posting question, answer or voting a post. The drawback of this method is that it does not guarantee that the recommended expert will answer to the question even if he knows the answer to the question.

Another set of papers try to address this problem by mining previously posted questions. However this raises another set of problems with the reliability of previous answers. The quality of previously submitted the answers varies. Thus, having a reliable set of question-answer to recommend requires manual labeling or other data mining techniques as described in the work of Bian et al.[2] and Agichtein et al.[3]. For instance, Bian et al.[2] suggest a machine learning method, which learns how to mark posts based on the quality of the content and user rating. This algorithm learns on relatively small number of initially labeled posts.

Besides the poor quality of posts there is one more issue with the content on the Internet, which should be eliminated to have better recommendations. Due to the openness of social web sites, they are exposed to spam and malware. Kyumin et al.[4] found the correlation between spam content and the poster's profile. Therefore, for identifying those posts, they suggested machine learning technique based on poster profile features such as friend information, posting patterns and etc. Heymann et al.[5] discuss existing approaches for detection, demotion, and prevention of posts containing spam or harmful data.

The approach discussed in this paper provides a better solution for the simple questions which have answers in technical resources. Suggested system for finding the answers will not have an issue of poor quality answers, as they will be retrieved from reliable sources. The answers will not depend on the other user experience and subjective view. Besides, automatic retrieval of useful resources will shorten user wait time to get the necessary answers and, thus, will overall raise the user satisfaction of the product.

For technical details the insights are taken from following papers. In M. Lipczak et al. [6] paper, where try to recommend tags Lucene engine was used to search for tag-profile. In S.K. Bajracharya et al. [7] try to recommend code examples based on the usage. In this case they index and search code examples by Lucene search engine. Also, in this paper authors well discussed the features and properties of Lucene search engine. Though the problem is not similar, these two papers gave me the insight of using Lucene search engine for this system. Another two papers, which introduced to NLP and the usage of NLP in Software Engineering are H. Zhong et al.[8] and H. Dumitru et al. [9]. H. Shong et al[8] paper describes Doc2Spec system where NLP is applied on documentations to infer specifications for the system. H. Dumitru et al.[9] paper uses NLP to parse and analyze product description to create product profile. Product profiles are then used to cluster products form recommendations for product features to be implemented.

3. OVERVIEW

The approached discussed in this paper is based on natural language processing. It takes into account the specific language users usually use while posting a question at forums. The observation is that the questions posted at forums usually can be fit into one question template. The question template will have following parts

- Title of the question
- Description of the situation
- Code examples
- Possibly description of an error
- Summary of the question

From all these parts system will use only title and the summary of the question. Therefore, the system should distinguish these parts of the questions and process them accordingly. Code examples and exception logs will be removed in message preprocessing phase discussed in Section 4.1. Titles of the questions are well separated by design of the forums. Finally, for separating the summary of the questions another observation is used. Summary of the questions usually contain specific keywords and by that can be distinguished from the rest of the message. The intuition is that most of the questions in the forums have fixed grammar and vocabulary, because all of them ask the same thing. All questions ask for some particular knowledge. For example, many summarized questions in forums start with "want to" (figure 1), "how to" (figure 2) or "Is it possible" (figure3) expressions. In figure 1 can be seen that user includes a lot of details into his message, but actually the marked sentence represents all the user wants to ask.

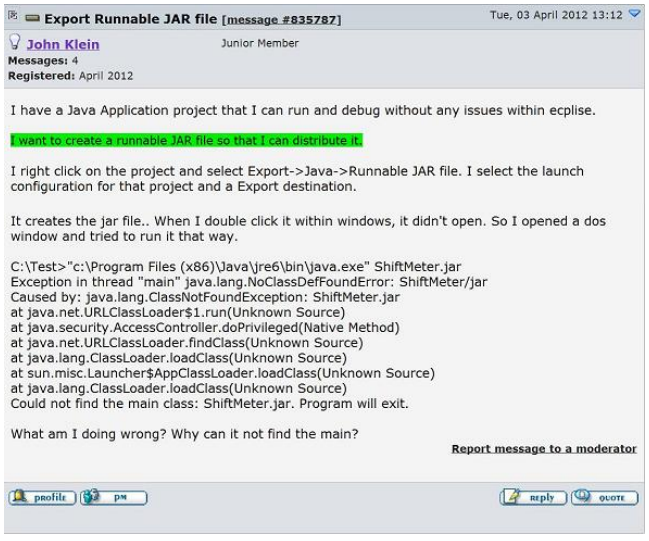


Figure 1: I want to create a runnable JAR file

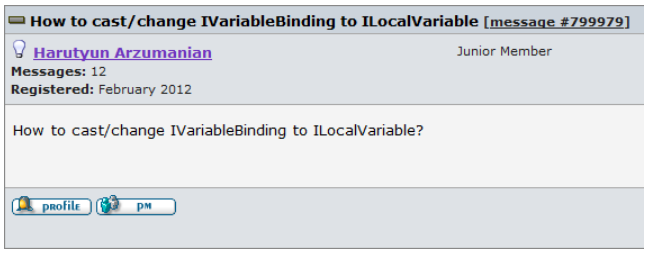


Figure 2: How to cast/change IVariableBinding to ILocalVariable?

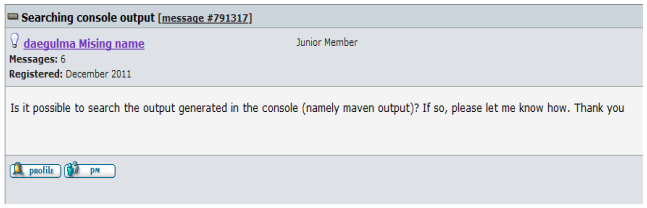


Figure 3: Is it possible to search the output generated in the console?

In all three cases the part of the sentence starting with these keywords, forms the main question of the user. Finding those question keywords is discussed in detail in Section 4.2.

After finding the question keywords system will proceed based on found two pieces of the information: the title and the summary of the question. The next step of the system will be generation of search queries for document name and document content (see Section 4.3). Search queries will be sent to Lucene search engine which will perform the search. The final step of the project will be the macthed terms identification in returned documents and recommendation of the highlighted document lists to users (more in detail see the Section 4.4).

4. IMPLEMENTATION

The first step of the system is creation of the document index. Which is done by DocIndexer once and then is used by the for seacrhing.

Afterwards, the process of the forum message processing can be started. Each forum message passes message preprocessing, keyword matching and question processing steps, after which the queries for searching the documents are ready.

The formed queries are passed to DocSearcher which uses previously built index of the documents to return most related documents. After that some more work need to be done by the highlighter to find the phrases which help document to gain the relevance score. Only after that the results can be delivered to the user.

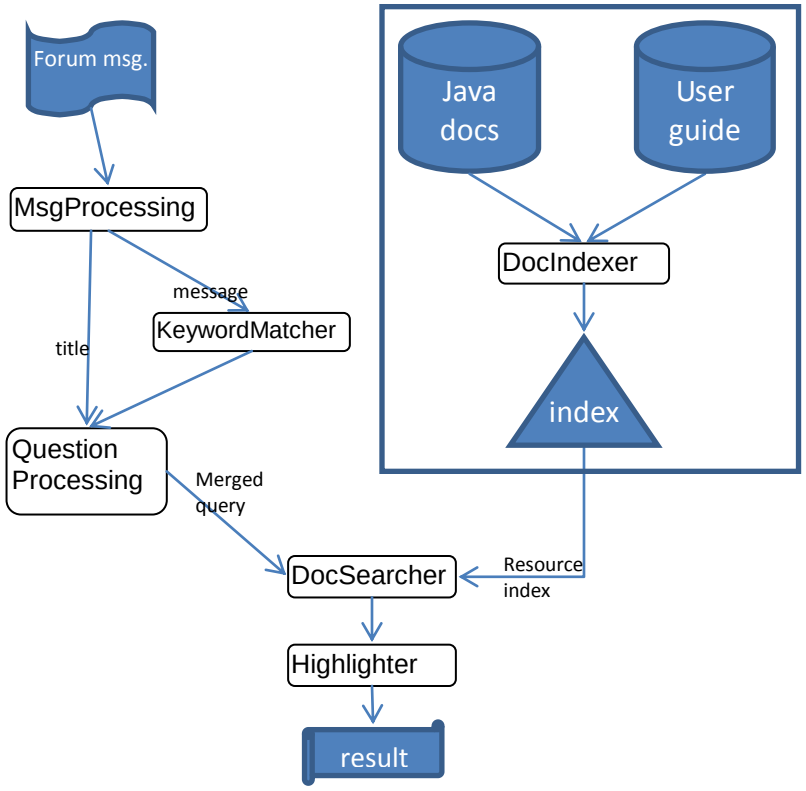


Figure 4: Main architecture

Following sections will discuss in detail all the steps of the implementation. Along with presenting the implementation details all the steps of the system will be explain on one of the real forum question shown below (see Figure 5).

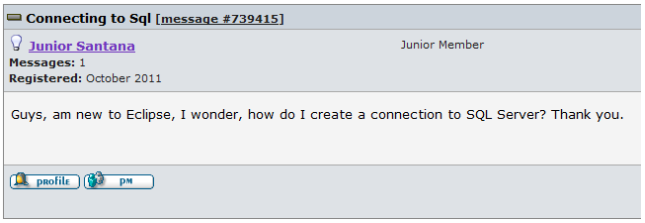


Figure 5: Example question

4.1 Message preprocessing

Message preprocessing includes code example removal, exception log removal and the invalid sentences removal.

In forum users can put code examples into special [code] blocks. In case code examples are placed in those blocks, they are detected and removed by the system.

For removing exception logs from the user post regular expressions are used. The applied set of regular exceptions does not guarantee detection of all exception logs, but majority of them can be detected and removed.

The last part of the preprocessing is elimination of invalid sentences. Invalid will be called the sentences which contain less

than three words, such as "Thank you.", "Please help…" etc. For that, message is split into sentences using OpenNLP[3] library, invalid sentences are identified and removed.

In our example there are no code examples and exception logs, but the last sentence "Thank you" will be removed as invalid sentence.

4.2 Matching Question keywords

Assuming that number of different ways of asking for knowledge is limited to some small number, popular structures for posing a question are manually identified. For that, words frequencies of the set of questions and processed it was calculated. In the resulting list of words frequencies, among the most popular words, besides stop words and eclipse specific terminology, are "how", "want", "need", "possible", "know", "can" and "tried". In considered 10257 questions the number of occurrences of the word "how" is 5014, "want" is 2503, "need" is 2138, "know" is 2837 and "possible" is 1206. This means that vast amount of questions in forums can be covered by structures using these words which will be called question "keywords". These keywords can help to find what the exact knowledge the user asks for is.

According to that finding the following list of keywords are

- How… (ex. "**How** to cast/change IVariableBinding to ILocalVariable?")
- want (but not "don't want") … (ex. "I **want** to create a runnable JAR file")
- need (but not "need help.") … (ex. "I **need** to write an eclipse plugin.")
- know (but not "I know")… (ex. "I don't **know** whether there is any different settings for fedora for BIRT viewer?")
- can (ex. "where I **can** find a jar or zip for glassfish")
- possible (ex. "Is it **possible** to search the output generated in the console?")

Note that though the word "tried" has high frequency as well, it is not considered as a question keyword, because mainly it expresses the unsuccessful attempts for solving the problem. That does not necessarily mean that it would include the formulation of the main problem user is trying to address.

Also the sentences can contain more than one of the keywords. Therefore, from keywords is considered the one which does not contain any other keyword from the fixed list. For example, for sentence like "does anyone **know** where I **can** find a jar or zip for glassfish" only last part is considered.

For finding keywords the fixed set of regular expressions are applied after which the keywords and the following part of the sentences are detected.

In our example, from the sentence "Guys, am new to Eclipse, I wonder, how do I create a connection to SQL Server?" the keyword "how" will be matched and the remaining of the sentence "how do I create a connection to SQL Server?" will be separated.

[3] http://incubator.apache.org/opennlp/index.html

4.3 Query formulation

After finding the question keywords we can search for main ideas of the user nearby in the same sentence. The part of the system for generating queries out of the found parts of the sentences is step by step discussed below. Explanations are followed by the example for each of them based on the same forum question example.

First step is tagging each word in the sentence by its parts of the speech using Stanford POS tagger[4]. For our example, we will have

[how/WRB, do/VBP, I/PRP, create/VB, a/DT, connection/NN, to/TO, SQL/NNP, Server/NN, ?/.]

The second step is multi-word term detection such as "Visual C++" or "data structures". The tokens are combined to form possible terms in Software Engineering term vocabulary[5]. The detected multi-word terms will be tagged with label "SET"(Software Engineering term) and will be added to already tagged set of words. In our example, "SQL Server" will be such multi-word term and tagged word set will be

[how/WRB, do/VBP, I/PRP, create/VB, a/DT, connection/NN, to/TO, SQL/NNP, Server/NN, ?/. SQL Server/SET]

Next, stop words are removed. The list of stop words is the standard list of stop words in English language with addition of the Top100 most popular words in the corpus. Corpus here is the collection of javadocs and user guides used in evolution, which will be discussed more in detail later in this paper. So, after stop words removal, in our example, we will have the following set of words.

[how/WRB, create/VB, connection/NN, SQL/NNP, Server/NN, ?/. SQL Server/SET]

Next, from the list of the tagged words [verb]+[noun] pairs are detected. From the [verb]+[noun] pair 4 combinations of query phrases are generated.

- [verb]+[noun]
- [noun]+[verb]
- [verb synonym]+[noun]
- [noun]+[verb synonym]

For synonyms the WordNet library[6] is used with combination of manually defined SE specific synonym sets. For our example, following phrases will be formed

[create] +[connection] , [connection]+ [create],
[open] +[connection], [connection] + [open],
[get] +[connection], [connection] +[get],
[make] +[connection], [connection] + [make]

From the remaining set of the words nouns (tagged as NN, NNS), proper nouns (tagged as NNP) and Software Engineering terms (tagged as SET) are selected to be joined to the query. For our example, those will be

[4] http://nlp.stanford.edu/software/tagger.shtml

[5] http://www.javvin.com/softwareglossary/index.html

[6] http://wordnet.princeton.edu/

[SQL], [Server], [SQL Server]

In addition all tokens are stemmed using Snowball stemming package[7] included in the Lucene library.

The last detail left is prioritization of the terms or boost of the query phrases. For all the terms a boosting factor is added equal to their priority. Priorities are assigned as follows

- Nouns (NN or NNS) – priority 1
- Proper nouns – priority 2
- Software Engineering term and Verb and noun phrases –priority 3

Eventually two queries are formed. One query is for searching the content of the documents and one for searching the names of the documents. Query for content includes all so far derived phrases and terms. The query for the names of the documents includes only NN, NNS, NNP, SET.

4.4 Search and highlighting with Lucene

For searching, first of all indexing of all available documents is necessary. In this case documents are javadocs and Java development Tools (JDT) guide. These documents are indexed by functionality provided by Lucene search engine. It is important to mention that documents are processed the same way as queries are. That is, the same stopwords removal and stemming as discussed in previous section is applied. For each document path, document name and content fields are saved in index. After search index is created it is saved and accessed for all queries during the search.

Set of query phrases discussed in previous section are passed to Lucene search engine. For searching documents content the combination of the Lucene *PhraseQuery* are used. The only customization is that search will allow having up to 2 additional words in between phrase words. So, for our example the content query will look like

> [creat connect]~2^3 [connect creat]~2^3, [open connect]~2^3, [connect open] ~2^3, [get connec] ~2^3, [connect get] ~2^3, [mak connect] ~2^3, [connect mak] ~2^3 [SQL]^2, [Server]^1, [SQL Server]^3

Where "~" followed by number is a number of words allowed to be in between the words and the "^" followed by a number is the boosting factor.

For document name search the combination of Lucene *WildcardQuery* are selected, which allows to have prefixes and suffixes for the phrases. For our example document name query will look like

> [*SQL*]^2, [*Server*]^1, [*SQL Server*]^3

Though these two queries search two different fields of the index, Lucene search engine allows to easily combine queries and to perform search only one time.

After performing search Lucene returns matched documents with corresponding score. To show the results to the user the additional step needs to be done. The returned documents and the search query are used to highlight the matched terms. First the listing of matched documents is shown to the user with highest scored 3 fragments of the documents. From that list users can navigate to the document which will be shown with appropriate highlighting as well.

For our example the first three of the results will be as in Figure 6. From this page users can navigate to the corresponding documents. For example, java.sql class javadocs can be opened by clicking on the title.

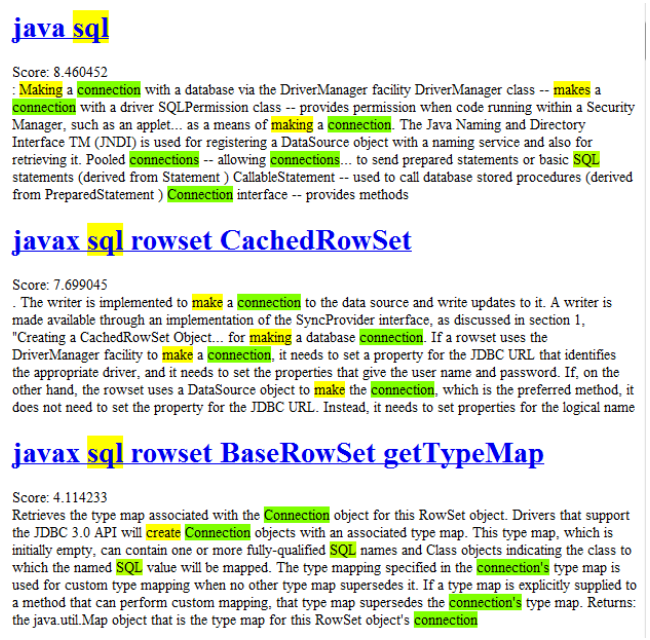


Figure 6: Recommended documents

Note that though the listed documents are not particularly for creating a connection, but they still discuss creating a connection, that is why they are returned by the search query.

5. Tools and resources used

As the first step, for acquiring database of the questions a web scraper tool has been used. By using Helium scraper[8] questions have been downloaded from Eclipse forum. Helium Scraper is an easy to use Web Scraper tool. Helium Scraper allows to extract text, images and files. Using Helium Scraper it can be easily defined the rules for extracting questions from forums, with separated title, message and code blocks. The rules are used in Action Trees to extract as many questions as needed. Afterwards, all the extracted information can be exported to different data structures. For current project .CSV was selected as the export format.

For the question processing step four tools have been used. OpenNLP is used for sentence detection and tokenization, Snowball is used for stemming, Stanford parser for POS tagging

[7] http://snowball.tartarus.org/

[8] http://www.heliumscraper.com/en/index.php?p=home

and the java wrapper for WordNet library JAWS[9]. OpenNLP is an open source natural language processing library written in Java. Though OpenNLP library includes POS tagger, a bit of practical use showed that Stanford parser in practice works better. Therefore, Stanford POS tagger is used as well.

For the next step of the system, searching technical resources for formed queries, Lucene[10] searching software is used. Both document indexing and searching is carried out by functionality provided by Lucene search engine. More detailed description of the Lucene search engine is provided in following Section 5.1.

5.1 Apache Lucene search engine

Lucene is a searching engine which provides a simple API for building-in search functionality into the application. Lucene provides ranked searching with, many powerful query types: phrase queries, wildcard queries, proximity queries, range queries and more, fielded searching (e.g., title, author, contents), date-range searching, sorting by any field, multiple-index searching with merged results. However, the key of the Lucene engine is the scoring system. Lucene combines Boolean model (BM) of Information Retrieval with Vector Space Model (VSM) of Information Retrieval. For query clauses with "must occur", "should occur", "must not occur" Boolean model is used to filter out the documents matching those restrictions. Afterwards, documents "approved" by Boolean Model are ranked by Vector Space Model. Documents and queries are represented as weighted vectors in a multi-dimensional space, where vectors have the length of the vocabulary and weights are Tf-idf values.

VSM score of document d for query q is the Cosine Similarity of the weighted query vectors V(q) and V(d). In standard formula vectors are normalized, but normalizing V(d) to the unit vector is known to be problematic in that it removes all document length information. Lucene introduces different document length normalization factor is used, which normalizes to a vector equal to or larger than the unit vector: doc-len-norm(d).

At indexing, users can specify that certain documents are more important than others, by assigning a document boost. For this, the score of each document is also multiplied by its boost value doc-boost(d).

Lucene is field based, hence each query term applies to a single field, document length normalization is by the length of the certain field, and in addition to document boost there are also document fields boosts.

The same field can be added to a document during indexing several times, and so the boost of that field is the multiplication of the boosts of the separate additions (or parts) of that field within the document.

At search time users can specify boosts to each query, sub-query, and each query term, hence the contribution of a query term to the score of a document is multiplied by the boost of that query term query-boost(q).

A document may match a multi term query without containing all the terms of that query (this is correct for some of the queries), and users can further reward documents matching more query terms through a coordination factor, which is usually larger when more terms are matched: coord-factor(q,d).

Multiplication of all these properties makes the score(q,d).

6. Data used

The corpus of the project, that is the documents to be searched, is Java™ Platform, Standard Edition 6 API Specification[11] and Java Development User Guide[12]. Those documentations are split into logical small documents for more effective search. Javadocs are split by the class and function descriptions and the name of the document is considered to be class name or function name accordingly. The user guide is split into documents according the browsing tree. The names of the documents assigned in this case are the names of the nodes of the browsing tree.

For the examination of word frequencies, for all the provided examples, and for the evaluation one category of the Eclipse forum was used. All questions are taken from "Language IDEs -> Java Development Tools (JDT)" section provided by http://www.eclipse.org/forums. Java Development Tools (JDT) category was selected taking into account that it will contain more simple questions, answer for which might be found in the technical resources. Besides, javadocs could be used as an additional resources covering Java Development Tools (JDT) category of the forum.

7. Evaluation

Evaluation of the system mainly addresses the coverage of the problem. For evaluation 1829 latest questions (on Feb 2012) are considered from Java development Tools(JDT) forum.

The questions discusses are how many of user post will hve question keywords, how many of the user post will have any answers, whether only searching by title is enough and etc.

First question evaluated was the coverage provided by the set of question keywords selected. Out of 1829 selected questions only 560 didn't have any question keyword. However, 488 of those still had some answer according to title.

The next question was how many of the user post will have some answer. For these and for identifying the relative roles of two sources of the queries (title and summary of the question) 3 different runs of the program were done.

First, the program was run for 1829 user post without considering the titles. In this case only 1054 of the question found any answer. Second, for the same 1829 user post program was run by taking into account only title. In this case 1593 questions found some answers. The last third run combined both title and the body of the message. For this last run 1705 questions found answers. Not

[9] http://lyle.smu.edu/~tspell/jaws/index.html

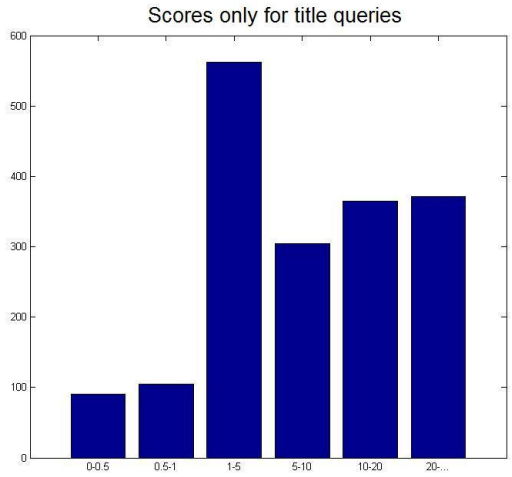
[10] http://lucene.apache.org/

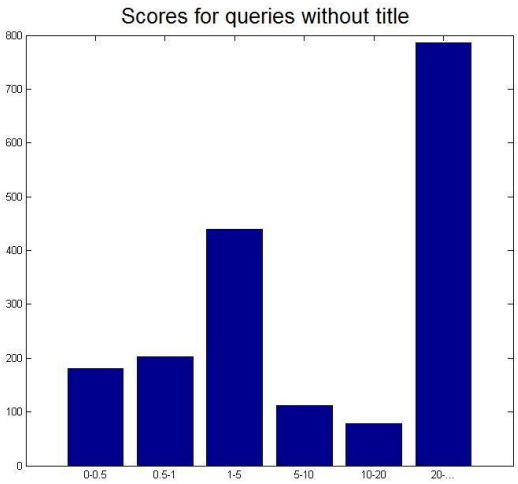
[11] http://docs.oracle.com/javase/6/docs/api/

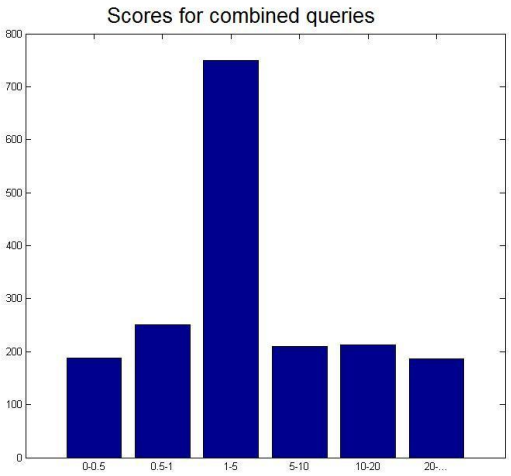
[12] http://www.eclipse.org/documentation/

surprisingly, in case when title is not considered the number of user post with any suggestion returned is the lowest of the three, because this run gives no chances for 560 user posts which do not include any question keywords. The largest number of user post which has any suggestion has the third run. This suggests that actually title and the summary of the questions contain information which completes each other. As titles are some kind of summary of the question it might be the case that title and identified summary of the question would repeat each other. However, the numbers show that it is not the case.

Further, the scores for each three run are divided into following bins: (0, 0.5], (0.5, 1], (1, 5], (5, 10], (10, 20], (20, …]







As can be seen in the figures above, scores for runs with only title and without title are higher than in combined case. This suggests one more time that those queries supplement each other. Queries with title and without title have higher scores than in combined case because in those cases the queries are shorter. Therefore, it is easier to match those queries. However, it is important to note that in all cases, even in combined case the majority of the answers have score, that is relevance, higher than 1. Thought correctness of the system is not evaluated, however, having relevance score higher than 1 gives hope.

8. Discussion

A lot of things can be done to improve the behavior of the describe system.

First, more resources (user guides, articles, papers, etc.) would enrich the index and the results of the search will be more relevant.

Second, the question keywords are identified manually and might not be static over time. It is possible that question patterns will be changed or maybe there are some not that popular question patterns which remain unnoticed. Therefore, detection of more question keywords might increase coverage and hence increase in relevance of the search results.

The SE terms, synonyms and stopwords seem to be an implementation detail. However they decide a lot in above search process. However, at this point no vocabulary can be found specific for programming lexicon. Therefore, better SE terms, more suitable synonyms, better stopwords list, would definitely improve the behavior of the system.

The other factor which might improve the system is the use of code elements. Among the user post which don't have any question keywords discussed above are many which includes mostly code example and a simple question like "The following code is not working. What can be done?". In this kind of the questions all the knowledge is covered by code example.

Therefore, the understanding of the code and transforming it to search query would cover at least this cases.

Some users of the system might come across to unexpected behavior of the system and post that scenario as a question. However, that behavior can be a known issue for the system. If the system could distinguish those cases then useful recommendations can be provided to the user. Recommendation system can look through database of known bugs and give back the available workarounds to the user.

9. REFERENCES

[1] Gideon Dror, Yehuda Koren, Yoelle Maarek, and Idan Szpektor. 2011. I want to answer; who has a question?: Yahoo! answers recommender system. In Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD '11). ACM, New York, NY, USA, 1109-1117. DOI=10.1145/2020408.2020582 http://doi.acm.org/10.1145/2020408.2020582

[2] Jiang Bian, Yandong Liu, Ding Zhou, Eugene Agichtein, and Hongyuan Zha. 2009. Learning to recognize reliable users and content in social media with coupled mutual reinforcement. In*Proceedings of the 18th international conference on World wide web* (WWW '09). ACM, New York, NY, USA, 51-60. DOI=10.1145/1526709.1526717 http://doi.acm.org/10.1145/1526709.1526717

[3] Eugene Agichtein, Carlos Castillo, Debora Donato, Aristides Gionis, and Gilad Mishne. 2008. Finding high-quality content in social media. In *Proceedings of the international conference on Web search and web data mining* (WSDM '08). ACM, New York, NY, USA, 183-194. DOI=10.1145/1341531.1341557 http://doi.acm.org/10.1145/1341531.1341557

[4] Kyumin Lee, James Caverlee, and Steve Webb. 2010. Uncovering social spammers: social honeypots + machine learning. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval* (SIGIR '10). ACM, New York, NY, USA, 435-442. DOI=10.1145/1835449.1835522 http://doi.acm.org/10.1145/1835449.1835522

[5] Paul Heymann, Georgia Koutrika, and Hector Garcia-Molina. 2007. Fighting Spam on Social Web Sites: A Survey of Approaches and Future Challenges. *IEEE Internet Computing* 11, 6 (November 2007), 36-45. DOI=10.1109/MIC.2007.125 http://dx.doi.org/10.1109/MIC.2007.125

[6] Marek Lipczak and Evangelos Milios. 2010. Learning in efficient tag recommendation. In*Proceedings of the fourth ACM conference on Recommender systems* (RecSys '10). ACM, New York, NY, USA, 167-174. DOI=10.1145/1864708.1864741 http://doi.acm.org/10.1145/1864708.1864741

[7] Sushil K. Bajracharya, Joel Ossher, and Cristina V. Lopes. 2010. Leveraging usage similarity for effective retrieval of examples in code repositories. In *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering* (FSE '10). ACM, New York, NY, USA, 157-166. DOI=10.1145/1882291.1882316 http://doi.acm.org/10.1145/1882291.1882316

[8] Hao Zhong, Lu Zhang, Tao Xie, and Hong Mei. 2009. Inferring Resource Specifications from Natural Language API Documentation. In *Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering* (ASE '09). IEEE Computer Society, Washington, DC, USA, 307-318. DOI=10.1109/ASE.2009.94 http://dx.doi.org/10.1109/ASE.2009.94

[9] Horatiu Dumitru, Marek Gibiec, Negar Hariri, Jane Cleland-Huang, Bamshad Mobasher, Carlos Castro-Herrera, and Mehdi Mirakhorli. 2011. On-demand feature recommendations derived from mining public product descriptions. In *Proceedings of the 33rd International Conference on Software Engineering* (ICSE '11). ACM, New York, NY, USA, 181-190. DOI=10.1145/1985793.1985819 http://doi.acm.org/10.1145/1985793.1985819