An Question Recommendation System for Question Answer Communities

Haoran Hou
Department of Computer Science
University of Virginia
Charlottesville, VA 22903
hh6wh@virginia.edu

Haoyu Chen
Department of Computer Science
University of Virginia
Charlottesville, VA 22903
hc6as@virginia.edu

ABSTRACT

In this project, we propose a system called Question Recommendation System (QRS) with which users could input a new question, they would get a list of existing questions that may already have good answers. The way to get those questions is to mine the existing similar questions and related answers, from an online community environment. These are the environments such that users involved in both asking questions and giving answers. In our system, we offer users a short-cut of a faster and more straightforward way of getting the information they desire, without necessarily digging into large data set. The common community Q&A sites would simply match the newly submitted query with existing questions. In our project, we have developed a user-friendly question recommendation system based on Apache Solr, in which the search results consider both question titles and tags. Our evaluation shows our system outperform the existing question recommendation system on Stack Overflow in terms of user experience.

Categories and Subject Descriptors

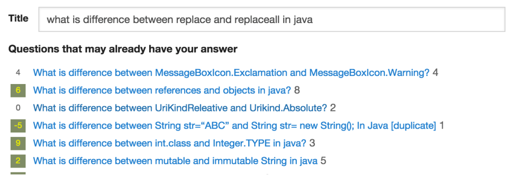
[**Text Mining**]: Community Question AnsweringApache Solr Google APP Engine

Keywords

text mining, community question answering, Apache Solr, question recommendation system

1. INTRODUCTION

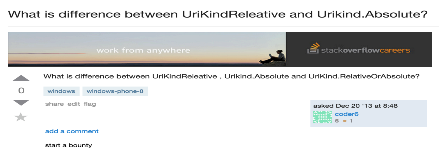
In todays Internet environment theres an increasing reliance on question answering communities for problem solving. For example, a programmer may need to search or ask on websites like *StackOverflow* when coming across confusing bugs. However, in some cases users would just post a new question without searching for existing solutions, or they do have but still couldnt get what they want and they turn to start up a new question. Chances are that similar questions


Title: what is difference between replace and replaceall in java

Questions that may already have your answer

4 What is difference between MessageBoxIcon.Exclamation and MessageBoxIcon.Warning? 4
6 What is difference between references and objects in java? 8
0 What is difference between UriKindReleative and Urikind.Absolute? 2
-5 What is difference between String str="ABC" and String str= new String(); In Java [duplicate] 1
9 What is difference between int.class and Integer.TYPE in java? 3
2 What is difference between mutable and immutable String in java 5

Figure 1: Questions suggested by stackoverflow.


What is difference between UriKindReleative and Urikind.Absolute?

What is difference between UriKindReleative , Urikind.Absolute and UriKind.RelativeOrAbsolute?

windows windows-phone-8

asked Dec 20 '13 at 8:48 coder6 6 ● 1

Figure 2: The content of the third suggested question.

have already been asked in the forum, and answering the new one would both waste the time of users and increase redundancy of the forum. Therefore, its very vital to design a system which is able to match existing similar questions and answers right after the user submit the question.

Quite a few Question Answer Communities (QAC) such as *stackexchange* and *Quora* have already implemented this kind of Question Recommendation System (QRS). However, these systems have drawbacks. For example, in StackExchanges QRS, theres no tag matching when it comes to question suggestion. A tag of a question indicates the subject involved in the post, which is highly useful for question matching. As shown in **Figure 1.**, the third question is totaly unrelated to the query asking about Java, whose tags are actually 'windows' and 'windows-8-phone'(shown in **Figure 2.**). So in our implementation, we have incorporated enterprise search engine *Apache Solr*. Our search will consider both tags and question title which result in more reasonable results.

Beside that, although the existing tools, such as Stack Overflow, do provide similar existing questions, they have lots of flaws in their user interface so that it takes lots of extra effort of users to find what they look for. The unpleasant and confusing user experience also make our users neglect the existing functions. Therefore, during our design process, we also pay special attentions to improve current user interface in order to help users find the answer easily and effectively.

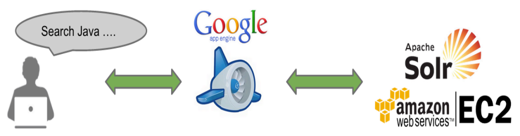


Figure 3: A general structure of our system.

For our project we will use data from *stackoverflow*, in that it's open source, it's more Q/A oriented than general community forums. **Figure 3** shows the high-level system schematic for our Answer Extraction System. Generally, the input question titles from users will be the only input parameter in our system. It is firstly processed by the search interface website hosted on Google APP Engine (GAE). And then our code on GAE will compose and visit a query URL for our Solr on Amazon EC2. Then the query will return the Top-5 most similar questions' ids to search interface website. With question id, we are able to display every thing about that questions with the help of Stack Exchange API.

In the following chapters, we will firstly introduce the existing work in Question Recommendation System. And then methodology will be presented which is mainly based the algorithm on Solr. Then there will be a chapter to offer a detailed explanation about the implementation of our demo system. After that, we will evaluate our system from the perspective of both user experiment and search results. At last, we will summarize what we do and put forward some possible future works.

2. RELATED WORK

In the meantime, there are many other methods for improving the performance of related area, including using regression-based gradient to leverage the weight between relevance and quality of an answer[bian et al.][3], using votes of an answer from a post, as well as the textual relevance of it to leverage the rank of an answer[Agichtein et al][1]. [Jurczyk et al] use the link structure of a general-purpose question answering community to discover users whose answers are more promising[4], and [Li et al] build category-sensitive language models to cluster questions[5].

3. METHODOLOGY

The core of our searching algorithm can be broken down to three main components. The first is to determine the best algorithm to use during the searching of matching corpus. The second is to determine the best features used to match the queries of the posts in database best, and the third one is to determine the best answer once the post of question is found, which will then be the answers we provide to the users.

3.1 Algorithm for Text Matching

Since there are already many existing platforms that provide the functionality of corpus searching, such as Apache Lucene, MeTA, Sphinx, and so on, that can satisfy our needs. We finally decided to use Apache Solr, which uses Apache Lucene as core engine to operate our search needs, and we also used the searching algorithm thats provided by solr. Generally Solr works on large scale of data for many companies, and the feature it had, reliability, scalability and fault tolerance are the reason we choose it. As mentioned before, Solr uses an algorithm in Lucene to calculate the similarity score and return the matched documents based on the score in descending order. The algorithm is based on TF-IDF model. It uses a combination of the Vector Space Model and the Boolean model to determine how relevant a given Document is to a User's query. The VSM is used to determine how much similar one document is to the target query. It also uses boolean model to first narrow down the documents that need to be scored based on the use of boolean logic in the Query specification. Lucene also adds some capabilities and refinements onto this model to support boolean and fuzzy searching, but it essentially remains a VSM based system at the heart, as shown as equation 1. The specifics of the algorithm used in practice are shown as equation 2:

$$\begin{aligned} score(q,d) = coordfactor(q,d) * queryboost(q) * \\ \frac{V(q)V(d)}{|V(q)|} * doclennorm(d) * docboost(d) \end{aligned} \tag{1}$$

$$\begin{aligned} score(q,d) = coord(q,d) * queryNorm(q) * \\ \sum_{q}^{t} (tf(\text{t in d}) idf(t)^2 t.getBoost() norm(t,d)) \end{aligned} \tag{2}$$

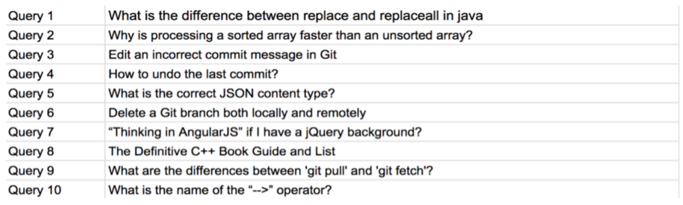
Query Euclidean norm |V(q)| can be computed when search starts, as it is independent of the document being scored. Query-boost for the query (actually for each query term), Document length norm doc-len-norm(d) and document boost doc-boost(d) are known at indexing time. They are used for normalizing the boosted terms, queries or fields. Since we did not use any boost during our implementation, there parts can be ignored.

Tf(t in d) correlates to the term's frequency, defined as the number of times term t appears in the currently scored document d. Idf(t) stands for Inverse Document Frequency.

Coord(q,d) is a score factor based on how many of the query terms are found in the specified document. Typically, a document that contains more of the query's terms will receive a higher score than another document with fewer query terms. This is a search time factor computed in coord(q,d) by the Similarity in effect at search time.[org.apache.lucene.search][2]

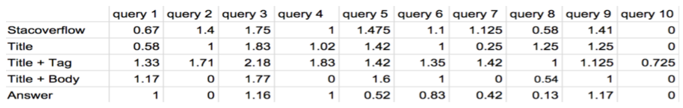
3.2 Component of Query Matching

Since for this project, we are only dealing with users initial input, the amount of information is very limited. Even though users would try to provide as much information as possible, many of their questions involve description of the specific situation, explanation or even lines of code, and while it is for the users to decide whether should they giving more information towards their questions, we should try to find as much as useful information from our database as possible. Every Stack Overflow question consists of two main parts: the question, and the answers. The question contains title, tag and detailed description, and comments, and the answer part would commonly have answers from different users, votes for each answer and comments as well. For this project we took title of each question, description, tags and most voted answer of the post into consideration, and see which feature can be used for query matching process. The



Query 1	What is the difference between replace and replaceall in java
Query 2	Why is processing a sorted array faster than an unsorted array?
Query 3	Edit an incorrect commit message in Git
Query 4	How to undo the last commit?
Query 5	What is the correct JSON content type?
Query 6	Delete a Git branch both locally and remotely
Query 7	"Thinking in AngularJS" if I have a jQuery background?
Query 8	The Definitive C++ Book Guide and List
Query 9	What are the differences between 'git pull' and 'git fetch'?
Query 10	What is the name of the "-->" operator?

Figure 4: 10 test queries



	query 1	query 2	query 3	query 4	query 5	query 6	query 7	query 8	query 9	query 10
Stacoverflow	0.67	1.4	1.75	1	1.475	1.1	1.125	0.58	1.41	0
Title	0.58	1	1.83	1.02	1.42	1	0.25	1.25	1.25	0
Title + Tag	1.33	1.71	2.18	1.83	1.42	1.35	1.42	1	1.125	0.725
Title + Body	1.17	0	1.77	0	1.6	1	0	0.54	1	0
Answer	1	0	1.16	1	0.52	0.83	0.42	0.13	1.17	0

Figure 5: Detailed System Flow Chart

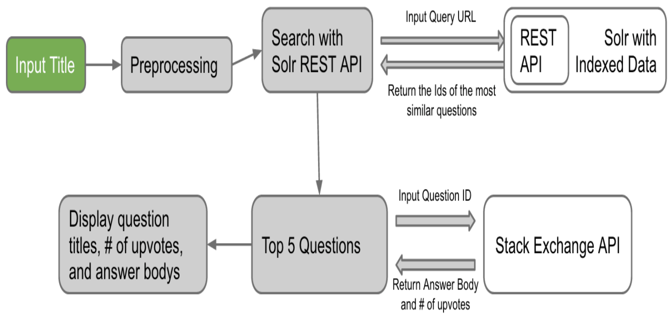

Input Title
Preprocessing
Search with Solr REST API
Input Query URL
Return the Ids of the most similar questions
REST API
Solr with Indexed Data
Display question titles, # of upvotes, and answer bodys
Top 5 Questions
Input Question ID
Return Answer Body and # of upvotes
Stack Exchange API

Figure 7: Detailed System Flow Chart

determine this, we picked 10 questions with top voted answers from Stack Overflow as our test queries(shown as 3.2), and take top 5 returned query for feature matching scenario. To further quantify the result, we hand-tagged the relevance R of each result as 3, and compute as 4:

$$R_r = \begin{cases} 0 & \text{irrelevant result} \\ 0.5 & \text{probably relevant result} \\ 1.0 & \text{positive result} \end{cases} \tag{3}$$

$$score(method) = \sum_n \frac{1}{n} * R_r(n) \tag{4}$$

The result of query matching of the 10 test queries using different methods are shown as table 3.2. We also used search results from stackoverflow as baseline comparison:

And its line chart is shown as figure 3.2

From figure we can see that using title and tag to match has generally the highest score and the most stable performance. Simply using title for matching would get the result slightly worse than the results yielded by Stackoverflow itself. Matching title and body would get quite unstable result, and we think its because some users like to use daily language, or they need to further explain the question, which would bring large amount of extra words to the scenario and thus bring down the relevance between the input query and the document to be matched, and the same reason goes for matching with answers of the post.

We would also like to point out that, even though the title-tag matching method has the best performance, it tends to find better results when it comes to technical questions instead of ask-for-opinion questions, and all of them works very bad when there are non-word characters in the query.

From the analysis above, we decided to use title and tag for matching criterion.

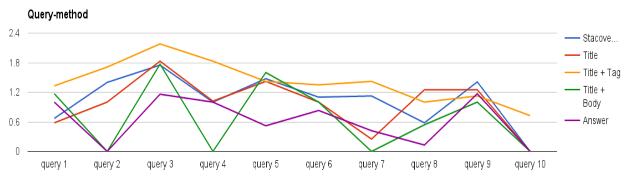


Figure 6: Query method line chart

3.3 Retrieve answer

We took a step further than solely provide the matching queries for this project, in that most of the times we looks things up in stackoverflow we are only looking for syntax or parse of error message without providing very detailed information. This need can be satisfied by giving the best answer of the post matching the user input query.

Instead of expert-trusting mechanism for most question answering community, Stack Overflow let the user who proposes the question as well as other users who received help from the post decide which answer is better. Generally, when an answer got most votes it tends to be the answer accepted by the question poser, which will be the ideal return answer for our project. However, there are times when the user accepts the answer that doesnt have the highest votes. In cases like this we would favor votes over acceptance in that the more votes one answer gets, the more generally helpful it is towards other users.

4. IMPLEMENTATION

Based on the methodology we talked about before, a StackOverflow question searching system has been implemented. It aims at finding and displaying the most similar questions as the user inputs the new question title. A detailed system flow chart is shown in **Figure 7.**. The following subsection about GAE and Amazon EC2 will be introduced based on that figure. And also if you want to get real experience of using our system. You can visit our demo website (https://stackoverflow-search.appspot.com/). And also the source code is also available on GitHub

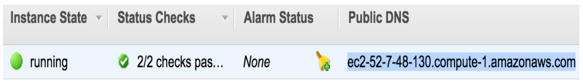
$$(https://github.com /haoyuchen1992/GAE{-}StackOverFlow{-}QuestionSearch). \tag{5}$$

4.1 Data Preprocessing

Stack Exchange network public their data under CC BY-SA 3.0. Developer could not only downloaded Stack Exchange data dump (http://data.stackexchange.com/help) which is an anonymized dump of all user-contributed content on the Stack Exchange network, but also could use the Stack-Exchange API to access data by customized query. Obviously, Stack Overflow is under the Stack Exchange network. So we are able to download all the question posts in Stack

```
<add>
  <doc>
    <field name="employeeId">05991</field>
    <field name="office">Bridgewater</field>
    <field name="skills">Perl</field>
    <field name="skills">Java</field>
  </doc>
  [<doc> ... </doc>[<doc> ... </doc>]]
</add>
```

Figure 8: The Indexable Format for Apache Solr



Instance State	Status Checks	Alarm Status	Public DNS
running	2/2 checks pas...	None	ec2-52-7-48-130.compute-1.amazonaws.com

Figure 9: Amazon EC2 Instance For Hosting Solr

Overflow. The data we used in our demo website is the question posts dataset updated on April 27th, 2015. It contains nearly 8 million question posts and the size is around 28 GBits. However, based on our algorithm, we only care about the question ID, question title and question tags. Besides filtering other unnecessary fields, in order to index data in Solr, we also need to convert the file into Solr index format (See **Figure 8.**) .

However, 28 GBits is stored in one XML file. It is unfeasible to process it by directly opening such a big file. So in this case, we use SAX parser in Java which will not load file in memory at a time and insteadly read the one element per time in XML file. the data size has been decreased to 1.6 GBits.

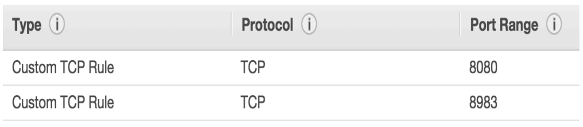
4.2 Solr Hosted on Amazon EC2

With the indexed data, we are able to index our data in Solr. In our implementation, we host Solr in Amazon EC2(See **Figure 9.**). In order to do that, we firstly create Amazon EC2 Instance and then download both solr 5.1. After that, we upload our preprocessed indexable data.

One thing should be paid special attention is to remember to add a rule in the appropriate security group to enable inbound traffic on port 8983 (See **Figure 10.**), which is the default port for running Solr. Then everyone accessible to internet is able to use the query in that Solr. In another word, now the Solr in our Amazon EC2 could offer REST API service for other applications.

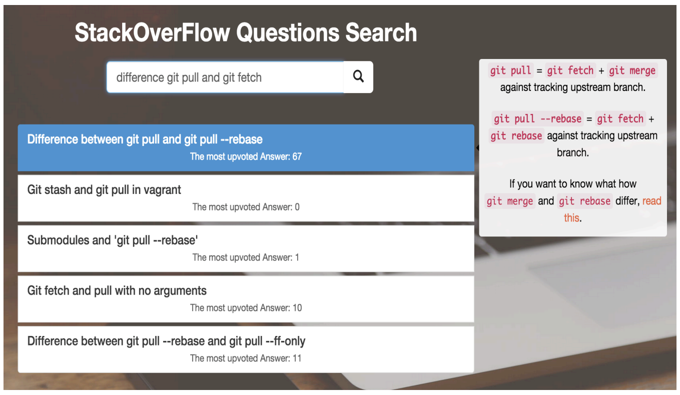
4.3 Search Interface Website Hosted on GAE

In interface website, it directly gets the input title from users and do a simple processing towards the input, for example, delete the unnecessary space. The result of preprocessing will be treated as the input of query. So we need take advantage of REST API from Solr. We customize query URL for our search. At the beginning, we need add our host for Solr –*http://ec2-52-7-48-130.compute-1.amazonaws.com:8983/solr/test/select.* And then the fields of search should be designated which is *fl=Id+Title+Tags* and then it will search the query within id, tags and title.



Type	Protocol	Port Range
Custom TCP Rule	TCP	8080
Custom TCP Rule	TCP	8983

Figure 10: Customize TCP Port 8983 for Solr



StackOverFlow Questions Search

difference git pull and git fetch

Difference between git pull and git pull --rebase
The most upvoted Answer: 67

Git stash and git pull in vagrant
The most upvoted Answer: 0

Submodules and 'git pull --rebase'
The most upvoted Answer: 1

Git fetch and pull with no arguments
The most upvoted Answer: 10

Difference between git pull --rebase and git pull --ff-only
The most upvoted Answer: 11

git pull = git fetch + git merge against tracking upstream branch.

git pull --rebase = git fetch + git rebase against tracking upstream branch.

If you want to know what how git merge and git rebase differ, read this.

Figure 11: Display Example in Our implementation of Question Search for Stack Overflow

As for *wt=json*, it denotes that the type of return file will be json. Finally "row=5" means that the five most similar questions' id will be returned.

After we know about question IDs, Stack Exchange API can be applied to get the answer Id of the most upvoted answer for each question. Firstly, The API head will be noted as *https://api.stackexchange.com/2.2/questions/* and then follow with question id. Finally with the query setting–*/answers?order=desc &sort=votes&site=stackoverflow*, it will return the answer id with the largest number of upvotes.

Then we apply Stack Exchange API again to get the answer body of the most upvoted answer. Now the query setting is

order=desc&sort=activity&site=stackoverflow&filter=withbody

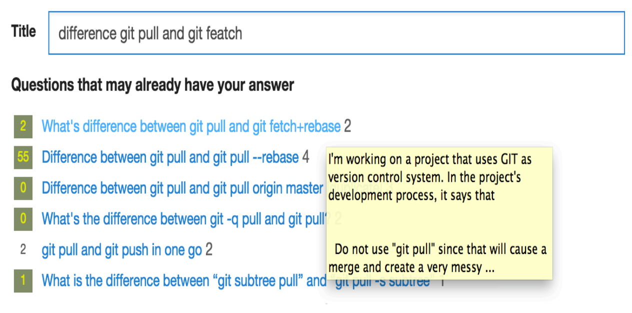
After that, we have got both the number of votes and the answer body from the most upvoted answer. The only thing left is to display it.

5. EVALUATION

Since Stack Overflow is not open source project, it is really difficult for us to compare our algorithm in details. However, it is easy to do a usability comparison between our implementation and the existing version of Stack Overflow. So from the perspective of usability, what makes our implementation different from the existing version in StackOverflow? There are two major differences(improvements) in our implementation.

The first improvement is about what information should be displayed. In existing version of Stack Overflow, they show the number of votes for the questions and the number of answers under those questions. Besides that, they also use green color box to represent the questions that have accepted answer.It is shown in **Figure 1.**.

However, we dont think those information offer the most important information for user. On the contrary, there is too much information that distracts users attention. If we think from the perspective of users, it is easy to found that what users really care is about whether the existing answers could solve their problems. So now the problem is which answer is most likely to achieve that goal. In our system, we assume it is the most upvoted answer from the existing questions. For an existing question, our system will only show the number of upvotes from the most upvoted answer. Another improvement is about what information should be displayed when


Title: difference git pull and git featch

Questions that may already have your answer

2 What's difference between git pull and git fetch+rebase 2
55 Difference between git pull and git pull --rebase 4
0 Difference between git pull and git pull origin master
0 What's the difference between git -q pull and git pull
2 git pull and git push in one go 2
1 What is the difference between "git subtree pull" and "git pull -s subtree" 1

I'm working on a project that uses GIT as version control system. In the project's development process, it says that

Do not use "git pull" since that will cause a merge and create a very messy ...

Figure 12: Display Example in The Current Version of Stack Overflow

users hover their mouse on an existing questions. Similar as the logic in first improvement, we think the best practice is to suggest the best answer to users(shown in **Figure 11.**). In the current version of Stack Overflow(shown in **Figure 12.**), when mouse hovers on a questions, it takes around 2 seconds to show the question body with plain text. It is so slow that almost nobody realize Stack Overflow has such a functionality. On the contrary, in our system, with Stack-Exchange API, the mouse hover on a question will enable to display an overview of the best answer. It not only offers more useful and informative overview, but also react with a faster speed(less than 0.25 second).

6. FUTURE WORK

There are still many improvement that we can make based on what we've done for now. As explained previously, we only used simple question title + tag as target of our matching. Since the frequency of appearance of title and tag are different, a set of different weights should be applied to both, to make the return result more accurate. During our experiment, we also found that the similarity between answers and the questions are higher than we expected, yet we didn't find out if the answers could also be used for a part of query matching, and what's more, could be used for returning the answer the user is looking for. We tried to use a feature method 'interestingTerms' provided by Solr.MoreLikeThis, but we didn't make very far. The method would find out the terms with highest TF-IDF value of a matched document and return to user. This could be used as tag recommendation right after user finish an input. There's another import part of Stack OverFlow community, which is comments within each post. There are many times when the comments would judging if the answer is correct and maybe provide modification of it. Making use of the information provided by comments would definitely increase the recommendation performance.

7. CONCLUSION

We have implemented a complete question recommend system for Stack OverFlow. Comparing with existing tool, it improves both user experience and recommended results. To be more specific, our system display the best answer and its number of upvotes associated with the recommended question. Our searching result also consider tags in existing questions. We do realize that this is only the very first step of the whole process of information interaction, and we would love to explore more possibilities if possible.

8. REFERENCES

[1] E. Agichtein, C. Castillo, D. Donato, A. Gionis, and G. Mishne. Finding high-quality content in social media. In *Proceedings of the 2008 International Conference on Web Search and Data Mining*, pages 183–194. ACM, 2008.

[2] Apache. org.apache.lucene.searchclass similarity, 2000.

[3] J. Bian, Y. Liu, E. Agichtein, and H. Zha. Finding the right facts in the crowd: factoid question answering over social media. In *Proceedings of the 17th international conference on World Wide Web*, pages 467–476. ACM, 2008.

[4] P. Jurczyk and E. Agichtein. Discovering authorities in question answer communities by using link analysis. In *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, pages 919–922. ACM, 2007.

[5] B. Li, I. King, and M. R. Lyu. Question routing in community question answering: putting category in its place. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 2041–2044. ACM, 2011.