Evolution of Search Methodologies in Information Systems

Ankur Kataria
Department of Computer Science
North Carolina State University
Raleigh, NC, USA
akatari2@ncsu.edu

Shalini Sejwani
Department of Computer Science
North Carolina State University
Raleigh, NC, USA
smsejwan@ncsu.edu

Sneha Shah
Department of Computer Science
North Carolina State University
Raleigh, NC, USA
smshah4@ncsu.edu

ABSTRACT

Searching is indispensable for any information system to retrieve relevant data. Various information retrievals systems like Community Q/A systems, Forums, Bug reporting, etc use some searching techniques. Many search methodologies have been proposed and evolved for these systems over the years. Several mechanisms like tagging , ranking, re-ranking, duplicate avoiding, analysis of language models of the data have been used to improve the search results. In this paper we are reviewing several algorithms and methods developed for improving searching that have evolved over the years.

Keywords

Search; Question-Answer Forums; CQA; Tag; Ranking; re-ranking; Bugs; Duplicates; Keywords;

1. INTRODUCTION

Having an efficient search technique in information systems like Question answering forums , bug tracking systems , software forums is very necessary. These techniques have been developed and researched upon since a long time now. The search techniques are used in various ways like finding duplicates, retrieving similar information, etc. For a given search query, current search engines use traditional information retrieval approach to extract webpages containing relevant keywords. However, in software forums, often there are many threads containing similar keywords where each thread could contain a lot of posts as many as 1,000 or more.

Various techniques have been introduced and evolved in the past decade to make the search efficient. Most used techniques are use of Tagging, Ranking, Re-ranking, Finding Duplicates and understanding language models. In section 2 we have talked about the motivation for the paper . Section 3 talks about the work in several papers and how it is related with each other as well as with the searching in information sites. Section 4 talks about a few important algorithms that have been proposed in the papers we have read . Section 5 briefly talks about the various datasets used in the papers and Section 6 describes the baseline studies. In Section 7 we have given a brief commentary on how these papers are related and how the field of improvement in search has evolved over an entire decade. Section 8, is the conclusion of our learning .

2. MOTIVATION

To become informed of new and interesting technologies and to learn from and help one another software engineers use online software forums. As they provide a huge amount of valuable content. Developers and users often ask questions and receive answers from such forums. The availability of a vast amount of thread discussions in forums provides ample opportunities for knowledge acquisition and summarization. Online media which help software engineers improve their performance in software development, maintenance and test processes.

Manually finding relevant answers from these long threads is a painstaking task to the users. Finding relevant answers is particularly hard in software forums as: complexities of software systems cause a huge variety of issues often expressed in similar technical jargons, and software forum users are often expert internet users who often posts answers in multiple venues creating many duplicate posts, often without satisfying answers, in the world wide web.

To make the search techniques efficient, a lot of research is going on. We intend to review the evolution in the search algorithms in this paper.

3. BACKGROUND AND RELATED WORK

The various techniques used in improving the information retrieval algorithms are explained in this section. Section 3.1 describes review of research done for finding relevant information in Community Question answer forums and other areas. Section 3.2 explains how tags can be used to make the search efficient. It also describes some ways for automatic tag generation. Section 3.3 describes how can we detect duplicates and deduce that the similar information has already been posted. The research concentrated in this section is with respect to bug reporting and not question answer forums. but the algorithms mentioned can be also used to detect duplications in question and answer forums or in general searching.

3.1 Information Retrieval from software forums

3.1.1 Finding Relevant Answers in Software Forums

The first paper[1] provides a semantic search engine framework to process software threads and recover relevant answers according to user queries. Different from standard information retrieval engine, the framework in the paper infers semantic tags of posts in the software forum threads and utilizes these tags to recover relevant answer posts. The authors have analyzed 6,068 posts from three software forums. To empirically study the benefit of their overall framework, they also conducted a user-assisted study which shows that as compared to a standard information retrieval approach, their proposed framework could increase mean average precision.

They created a framework to find relevant answers from software forums. This framework consists of two main steps.

1. First, they propose an engine that automatically classifies and tags posts in software forums as: answers, clarifying answers, clarifying questions, feedbacks - both positive and negative, and junk e.g., "today I am happy". Users could then focus on reading only the questions/answers including those buried deep inside long threads rather than the entire posts. Some questions with correct answers (based on the positive or negative feedbacks) could also be identified.
2. Second, They build a semantic search engine framework that uses the inferred semantic tags to recover the relevant answers from the threads. They also collected a dataset of 4020, 680, and 1368 posts from Oracle, SoftwareTipsandTricks and DZone software forums respectively. Using this dataset, they tested their tag inference engine and show that they could infer tags with 67% precision, 71% recall, and 69% F-measure.

To evaluate their overall search engine framework, based on the same dataset, they conducted experiments where users are tasked to label returned answers to 17 technical software-related queries expressed in natural language returned by a standard information retrieval toolkit[2] and their proposed framework. They have shown that they could increase the mean average precision from 17% to 71%, after the tags were utilized. They show that they could achieve nDCG@1 of 91.2% and nDCG@2 of 71.6% on these queries, with automatically generated inferred tags.

Summarizing the major contributions of this work:

1. They propose an engine that infers a comprehensive set of tags of posts in software forums: questions, answers, clarifying questions, clarifying answers, positive feedback, negative feedback, and junk.
2. They were the first to propose a semantic search engine to find relevant answers to queries by leveraging automatically inferred tags.
3. They have experimented their solution on 3 real software forums analyzing a total of 6068 posts which shows their framework's reasonable overall accuracy in inferring tags.
4. They have run a user-assisted study to evaluate the quality of their proposed framework in returning relevant answers from software forums with encouraging result.

3.1.2. Finding Question Answer Pairs from Online Forums

In this paper[4] the authors propose a sequential patterns based classification method to detect questions in a forum thread, and a graph based propagation method to detect answers for questions in the same thread.

Their technique to finding question answer pairs has 2 components, question detection and answer detection. For Question detection they develop a classification-based technique to detect questions using sequential patterns automatically extracted from both questions and non-question sentences in forums as features. While for answer detection they use the traditional approach of finding answer by casting answer-finding as a traditional document retrieval problem by considering each candidate answer as an isolated document and the question as a query. And then they can employ ranking methods, such as cosine similarity, query likelihood language model and KLdivergence language model.

Contributions of this paper:

1. They develop a classification-based method for question detection by using sequential pattern features automatically extracted from both questions and non-questions in forums.
2. They propose an unsupervised graph-based approach for ranking candidate answers. The results of the graph-based approach can be used as features for supervised method.
3. They conduct extensive experiments on several data. The size of our data is much larger than those used in previous work on forums. The main experimental results include
   - 3.1. Their method outperforms rule-based methods for question detection;
   - 3.2. The unsupervised graph-based method outperforms other methods including the classification methods

3.1.3. Question Retrieval with High Quality Answers in Community Question Answering

The requirement of new approach for question oriented software retrieval is addressed in this paper[7]. The paper proposed a supervised question-answer topic modeling approach. The approach assumed that questions and answers share some common latent topics and are generated in a "question language" and "answer language" respectively following the topics. The topics also determine an answer quality signal. Compared with translation models, their approach models user behaviors on CQA portals, as well as highlights the instinctive heterogeneity of questions and answers. It also takes answer quality into account and performs robustly against noise in answers.

With the topic modeling approach, they propose a topic-based language model, which matches questions not only on a term level but also on a topic level. They conducted experiments on large scale data from Yahoo! Answers and Baidu Knows. Experimental results show that the proposed model could significantly outperform state-ofthe-art retrieval models in CQA.

One big challenge for question retrieval in CQA is that users are used to expressing similar meanings with different words, which creates lexical gaps when matching questions based on common terms. For example, we find that for a user query "how do I get knots out of my cats fur", there are good answers under an existing question "how can I remove a tangle in my cat's fur" in Yahoo! Answers. Although the two questions have very similar meanings, since they share few common words, it is hard for classic retrieval models like BM25 [8] and language models for information retrieval (LMIR) [9] to recognize their similarity. The lexical gap has become a major barricade preventing traditional IR models from retrieving similar questions in CQA.

In this paper, a study on how to leverage answers to retrieve similar questions for queries has been done. Specifically, they head for modeling questions and answers in a more natural way. The model should not only highlight the instinct heterogeneity of questions and answers, but also be flexible enough to take answer quality into account. To this end, they propose a supervised question-answer topic modeling approach for question retrieval in CQA. The underlying assumption is that although questions and answers are heterogeneous in many aspects, they share some common latent factors. The latent factors represent topics in askers' and answerers' minds. Following the common topics, askers ask questions in a "question language", while answerers provide answers in an "answer language". Moreover, there is a signal indicating the quality of an answer. The signal simulates how other users evaluate the answer with respect to the question in a question-answer pair. It is determined by how well askers and answerers follow common topics in the model. With the guide of quality signals, questions and answers are mapped into a common latent space (topic space) and question-question similarity can be measured with the help of information in answers compressed in that space. With the learnt topic space, they further propose a topic-based language model for question retrieval in CQA, which naturally integrates the matches in latent space and traditional retrieval models. They derive a collapsed Gibbs sampling method to estimate parameters in the model. The method is efficient in computation. Compared with existing methods, their learning approach comprehensively models user behaviors in CQA portals and at the same time naturally takes answer quality into account. Another advantage of their method is that it is capable of leveraging rich metadata associated with answers and automatically learning answer quality signals. Specifically, they take "best answer" as the gold standard and extract a variety of features from both answer texts and associated metadata. With the training data and features, they learn a score function to generate answer quality signals and let these signals supervise the learning of the topic space. By this means, their model can not only leverage the power of big data accumulated in CQA sites, but also save on the labor of human annotators. They test the proposed model on large scale Yahoo! Answers data and Baidu Knows data. Yahoo! Answers and Baidu Knows represent the largest and most popular CQA archives in English and Chinese, respectively. They conducted both quantitative and qualitative evaluations. The results show that our model can significantly outperform state-ofthe-art translation-based approaches for question retrieval in CQA.

To summarize the paper:

1. The proposal of considering answer quality when leveraging answers for question retrieval in CQA;
2. The proposal of a supervised question-answer topic model, which not only highlights the heterogeneity of questions and answers but also naturally takes answer quality into account;
3. An empirical verification of the efficacy of the proposed model on large scale English and Chinese CQA data.

3.2 Tagging in forums

3.2.1. Fuzzy Set Approach for Automatic Tagging in Evolving Software Published

The second paper[3] a novel, accurate, automatic tagging recommendation tool that was able to take into account users feedbacks on tags, and was very efficient in coping with software evolution. The core technique was an automatic tagging algorithm that was based on fuzzy set theory. Their empirical evaluation on the real-world IBM Jazz project shows the usefulness and accuracy of their approach and tool.

The key contributions of this paper include

1. Empirical study on the characteristics and relationships of work items and their associated tags,
2. TagRec, an efficient and accurate, automatic tag recommendation algorithm that is based on the fuzzy set theory with aforementioned unique features, and
3. A prototype tool and an empirical evaluation on a real-world software project to show the usefulness and accuracy of TagRec.

3.2.2 Tag recommendation in Software Information Site Proceeding

This paper[10], proposed TagCombine, an automatic tag recommendation method which analyzes objects in software information sites. TagCombine has 3 different components:

1. Multilabel ranking component which considers tag recommendation as a multi-label learning problem;
2. Similarity based ranking component which recommends tags from similar objects;
3. Tagterm based ranking component which considers the relationship between different terms and tags, and recommends tags after analyzing the terms in the objects.

They evaluate TagCombine on 2 software information sites, StackOverflow and Freecode, which contain 47,668 and 39,231 text documents, respectively, and 437 and 243 tags, respectively. Experiment results show that for StackOverflow, our TagCombine achieves recall@5 and recall@10 scores of 0.5964 and 0.7239, respectively; For Freecode, it achieves recall@5 and recall@10 scores of 0.6391 and 0.7773, respectively. Moreover, averaging over StackOverflow and Freecode results, we improve TagRec proposed by Al-Kofahi et al. by 22.65% and 14.95%, and the tag recommendation method proposed by Zangerle et al. by 18.5% and 7.35% for recall@5 and recall@10 scores.

In multi-label ranking component, they consider the tag recommendation problem as a multi-label learning problem [5], where each tag maps to a label. They infer the appropriate label sets (tags) using multi-label learning algorithms, and rank the tags according to their likelihood scores. In similarity based ranking component, they search similar software objects of the untagged objects, and recommend tags from the similar objects. In tag-term based ranking component, they first compute the affinity scores between tags and terms based on the historical tagged software objects. For an untagged object, they compute the ranking scores of various tags using the terms in the object and the pre-computed affinity scores

The main contributions of this paper are as follows:

1. There are limited studies on tag recommendation in the software engineering literature, especially for software information sites. The research fills this gap.
2. They propose TagCombine, an accurate, automatic tag recommendation algorithm which analyzes tag recommendation problem from 3 different views, using 3 different components.
3. They evaluate TagCombine on 2 popular software information sites, StackOverflow and Freecode. The experiment results show that TagCombine achieves the best performance compared with other state-of-the-art methods, i.e., TagRec and Zangerle et al.'s methods.

3.3 Finding Duplicates

3.3.1 Inferring resource specifications from natural language api documentation

The next paper[11] we read explored a unique field which helps in finding bugs occurred in the code if the developers didn't follow the documentation. They propose an approach, called Doc2Spec, that infers resource specifications from API documentation. For their approach, they implemented a tool and conducted an evaluation on Javadocs of five libraries. The results show that their approach infers various specifications with relatively high precisions, recalls, and F-scores. They also evaluated the usefulness of inferred specifications through detecting bugs in open source projects. The results show that specifications inferred by Doc2Spec are useful to detect real bugs in existing projects.

This paper makes the following main contributions: ·

1. They propose a novel approach, called Doc2Spec, that uses a Natural Language Processing (NLP) technique to analyze natural language API documentation and infers resource specifications.
2. They implemented a tool for Doc2Spec and conducted an evaluation on API documentation of five libraries. The results show that their approach infers various specifications with relatively high precisions, recalls, and F-scores.
3. They further conducted an evaluation to detect bugs using inferred specifications. The results show that these specifications are useful to detect previously known or unknown bugs in open source projects

3.3.2 Has This Bug Been Reported?

This paper[12] discussed about bug reporting. This was a new approach which retrieves existing bug reports based on short user queries, before the complete bug report is submitted. They perform evaluations on more than 16,340 Eclipse and Mozilla bug reports. The evaluation results show that the proposed approach can achieve better search results than the existing search functions provided by Bugzilla and Lucene. They believe their work can help users and testers locate potential relevant bug reports more precisely.

They found that using the existing search functions provided by the bug tracking systems, the same bugs could be still repeatedly reported due to the inadequate quality of the search results. Though

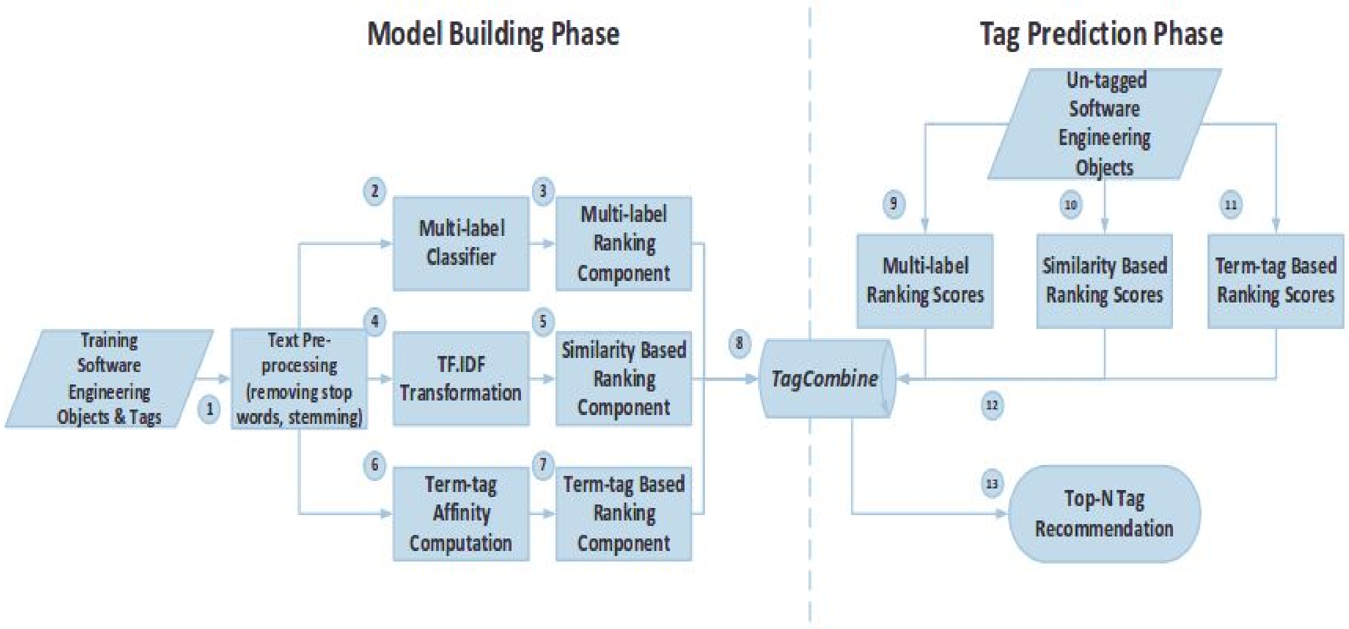

Model Building Phase
Tag Prediction Phase
Training Software Engineering Objects & Tags
1
Text Pre-processing (removing stop words, stemming)
2
Multi-label Classifier
3
Multi-label Ranking Component
4
TF.IDF Transformation
5
Similarity Based Ranking Component
6
Term-tag Affinity Computation
7
Term-tag Based Ranking Component
8
TagCombine
Un-tagged Software Engineering Objects
9
Multi-label Ranking Scores
10
Similarity Based Ranking Scores
11
Term-tag Based Ranking Scores
12
13
Top-N Tag Recommendation

Fig 1

bug reporting doesn't pertain to our theme but the importance of good search results is underlined here.

The increasing number of duplicate bug reports would require more bug management effort and increase maintenance cost. Therefore, it is desirable to provide a more effective search function to a bug tracking system.

They propose a novel approach that applies Ranking SVM, a Learning to Rank (LTR) technique, to search for potential relevant bug reports in a bug tracking system. Their approach enables users and testers to locate potential related bug reports more precisely.

To summarize the contributions of this paper:

1. This paper proposes to use Ranking SVM, a state-of-the-art LTR technique to build a ranking model for effective bug report search. Our approach can help a reporter locate potential related bug reports. Based on this, the reporters may decide whether they need to submit a new bug report or not.

2. They propose a set of features that can represent the specific characteristics of a bug report. Especially, we consider the semantic features obtained from the analysis of Wikipedia and LSI.
3. They conduct experiments on large bug report repositories to evaluate the proposed approach. The results show that our approach outperforms both the conventional search functions and our previous work.

3.3.3 A Benchmark Data Set for Community Question Answering Research

A common issue of community question answering websites is that a novice user may ask questions that have already been asked somewhere on the website. A solution to this problem could be to manually tag these questions by expert users but this paper[5] addresses this issue by trying to automate the process of suppressing such duplicate questions. It presents a benchmark dataset CQADupStack, for use in community question-answering (cQA) research. It contains threads from twelve StackExchange subforums, annotated with duplicate question information. They provide pre-defined training and test splits, both for retrieval and classification experiments, to ensure maximum comparability between different studies using the set.

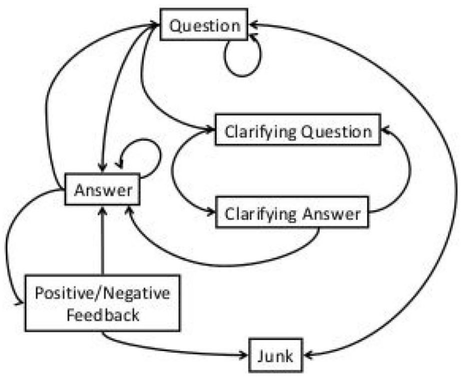
It comes with a script to manipulate the data, evaluate a system outputs, and make predefined splits that take into account the chronology of the questions. They analysed the set and applied several benchmark methods to it. It is a challenging set due to the high imbalance in questions with or without duplicates.

4. Implementation Methods

4.1 Tagging

4.1.1 Tag Inference Engine

Engine is based on a text classification approach. As with any classification problem, they first need to clearly define our class tags. They defined 7 tags including: question, answer, clarifying question, clarifying answer, positive feedback, negative feedback, and junk. They believe these tags well characterize the different types of post in software forums. The relationships among posts



having the seven class tags are illustrated in Figure 2

Fig.2

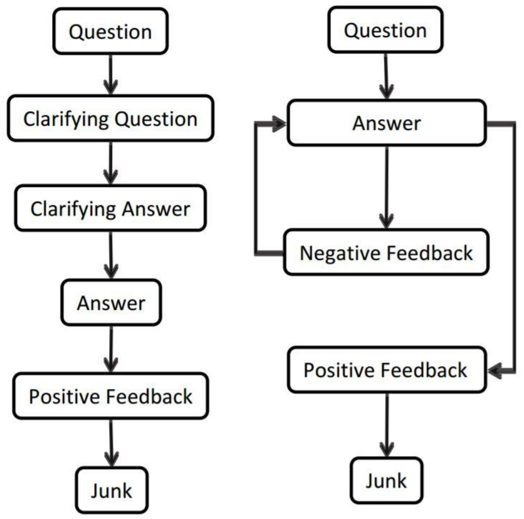


Fig.3

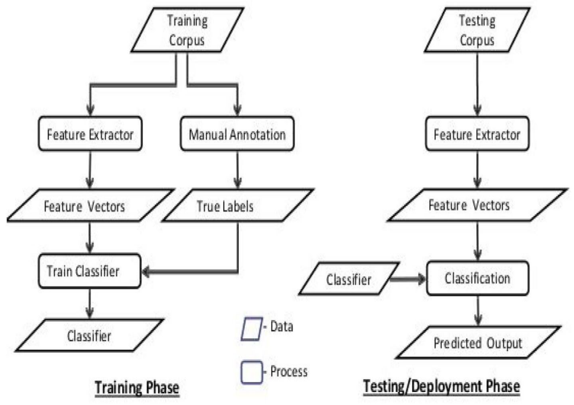


Fig.4

Tag based Search Engine Framework

In this section, they describe how they utilized inferred tags to help the retrieval of relevant answer posts from software forums. They first describe a typical search engine framework. Next, they describe how they could embed their tag inference engine to form a semantic search engine framework. A standard search engine framework, as shown in Figure 5, follows the following steps for processing a natural language query and retrieving relevant documents:

1. **Pre-processing:** The pre-processor takes in a set of raw documents and extracts a set of relevant features from them. The features are in the form of words existing in the document.
2. **Indexing:** The indexing tool builds an index from a collection of documents. This index would be used during the retrieval process to quickly locate relevant documents.
3. **Query processing and retrieval:** The retrieval tool processes user queries and retrieve relevant documents

using the index. Top-k matching documents are returned along with their respective relevance scores.

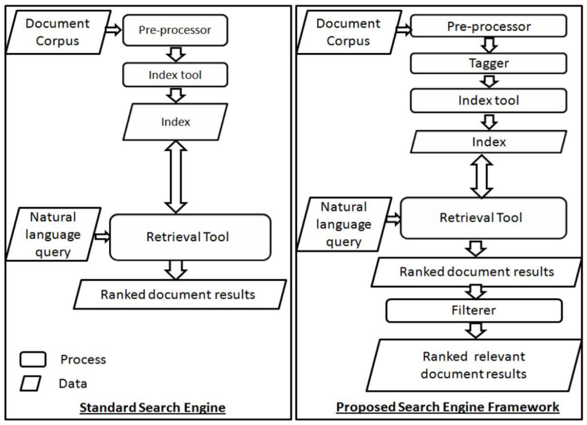


Fig.5

Their proposed search engine model is shown in the Figure 5. This model consists of additional two components that retrieves the relevant answers to the query. 1)

1. **Tagger:** They embed their tag inference engine into the standard search engine framework. As described in the previous sections, our tag inference engine (or tagger) would tag each document with various tags: question, answer, etc.
2. **Filterer:** This additional component filters the top-k matching documents from the retrieval tool based on their inferred tags. With the Filterer, relevant solution posts are kept while irrelevant posts, e.g., junk, are filtered out.

4.1.2 Tag-combine

The overall framework or architecture of TagCombine as shown in Fig 1 is explained below :

The 3 components of TagCombine : multi-label ranking component, similarity based ranking component, and tag-term based ranking component are built as shown in the framework. To construct the multi-label ranking component, a multi-label learning algorithm is used to build a multi-label classifier (Step 2), then modification of the classifier to is done to output ranking scores for the tags given an unlabeled software object (Step 3). To construct the similarity based ranking component, first the "bags of words" are transformed into TF.IDF (term frequency, inverse document frequency) vectors [8] (Step 4). The similarity between 2 software objects is calculated by computing cosine similarity of their TF.IDF vector representations 5 (Step 5) Then the tag-term affinity scores are computed using historical tagged objects (Step 6). After which the affinity scores are used to rank tags for a given unlabeled software object 6 (Step 7). Finally, TagCombine uses these 3 components (Step 8). It ranks tags based on the scores outputted by the 3 components. After TagCombine is constructed, in the prediction phase it is then used to recommend tags for a software object with unknown tags. For each such object, first its multi-label ranking score ,similarity based ranking score, and tag-term based ranking score are computed(Steps 9, 10, and 11). These scores are computed using the 3 trained ranking components constructed at steps 3, 5, and 7. Then these scores are inputted into TagCombine to get the final ranking score for each tag (Step 12). Finally, top-N ranked tags with highest scores are recommended for the object (Step 13).

4.1.3 Tag -Rec

4.1.3.1 MODEL

TagRec is a tagging recommendation tool that automatically assigns tag(s) for any work item. Importantly, it supports the evolutionary process of software work items as well. That is, during software development, it will recommend tags for any new work item while maintaining the existing tags for existing artifacts. It is also useful in the scenario to recommend the tags for current missing-tag work items while keeping the already tagged work items.

Work Item: A work item is modeled as a sequence of terms (i.e. stemmed words that are not grammatical ones or a stopword) in its title, summary, description, keywords, and related software artifacts.

Auto-Tagging: Given a set of work items in which some of them could be already associated with some tags and some are not, when a new set of work items is introduced, the auto-tagging tool analyzes the collection of work items and recommends the tags for all work items that do not have tags yet while maintaining the tags for already tagged items and possibly providing additional tags for them.

4.1.3.2 Fuzzy Set Approach

In TagRec, they model the tagging problem based on the fuzzy set theory [13], [14]. Let us describe the model in details:

After all stopwords and grammatical terms such as "a", "the", "and", etc are filtered, the remaining terms that appear in all work items or are used as keywords and tags should carry some semantic meaning to work items. All of those terms are collected into a set of terms for consideration in the corpus. Each term defines a fuzzy set and each work item has a degree of membership in this set. The key idea is to associate a membership function for each work item with respect to a particular term. The membership function takes values in the interval [0,1] with 0 corresponding to no membership in the class defined by a term, and 1 corresponding to full membership. Membership values between 0 and 1 indicate marginal elements of the class. Thus, membership in a fuzzy set is a notion intrinsically gradual, instead of concrete as in conventional logic [9]. That is, each term has a fuzzy boundary and each work item has a membership value indicating that the content of the work item belongs to that boundary.

Fuzzy Set for a Term: A fuzzy set T in the collection W of all work items is characterized by a membership function $\mu T : W \rightarrow [0, 1]$, which associates with each element w of W a number $\mu T(w)$ in the interval [0,1] in which 0 corresponds to no membership and 1 corresponds to full membership [15].

From the perspective of a work item, each of work items wj has a set of membership values $\mu[i, j]$ between [0,1], signifying the degree of membership it has with respect to each term ki in the corpus. If sorting all of those membership values $\mu[i, j]$ for all terms ki in the corpus, one would have the degrees of relevance of the work item wj with all the terms. In other words, one would have a set of the terms that are most suitable to describe the work item *wj* , and the corresponding ranks of the terms according to their degrees of relevance. The computation of the membership values for all work items with respect to all terms is performed via the computation of term correlation values as follows.

4.1.3.3 Term-Term Correlation

Some information retrieval models consider the terms as independent features (such as in the vector-based model (VSM) [15]). Unlike those models, TagRec examines the relationships among terms via the work items containing the terms. TagRec adopts the concept of keyword connection matrix in [16] to define the term-term correlation values as follows. The term-term correlation matrix is a matrix whose rows and columns are associated to the index terms in the corpus. In this matrix *C*, a normalized correlation factor $C[i, j]$ between two terms k_i and k_j is defined as

$$C[i, j] = |D_{i,j}| / |D_i| + |D_j| - |D_{i,j}| \quad (1)$$

where $|D_i|$ and $|D_j|$ are the numbers of work items containing the term k_i and k_j respectively, and $|D_{i,j}|$ is the number of work items containing both terms. If a work item is tagged with a term, TagRec considers that the term is contained in that work item. This term correlation definition is used for the terms appearing in the texts of work items.

4.1.3.4 Membership Values

Membership Value: A work item w_j has a degree of membership $\mu[i, j]$ with respect to the fuzzy set corresponding to term k_i. The value $\mu[i, j]$ is computed as in [15]:

$$\mu[i, j] = 1 - \prod(1 - C[i, l])$$

The idea is that a work item w_j belongs to the fuzzy set associated to the term *ki*, if its own terms are strongly related to *ki*. If there exists at least one term *kl* occurring within the work item *wj* which is strongly relevant to *ki* (i.e., $C[i, l] \approx 1$), then $\mu[i, j] \approx 1$, and the term *ki* is a good fuzzy index for the work item *wj* . If all terms in *wj* are irrelevant to *ki*, the term *ki* is not a good fuzzy index for *wj* (i.e. $\mu[i, j] \approx 0$).

4.1.3.5 Additional Ranking Scheme

For each work item w_j , the membership function $\mu[i, j]$ of a term k_i shows us how well the term k_i reflects the content of w_j. However, in formula (3), if there exists at least one term k_l in w_j that is strongly related to k_i, then the membership value is 100%. To distinguish between work items that have more than one such terms k_is, TagRec introduces an additional ranking scheme. If a work item *w*1 that has *m* terms that are strongly relevant to *ki* and another work item *w*2 with *n* terms ($m>n$) strongly relevant to *ki*, then the work item *w*1 is considered to be more relevant to the term *ki* than *w*2. Thus, with this ranking scheme, for a given tag *ki*, TagRec could rank and return the list of relevant work items.

4.1.3.6 Tag Recommendation

For each work item w_j, all the membership values corresponding to all the terms will be computed as in the formula (3). The terms that give the higher membership values for *wj* than a chosen threshold will be returned to developers as the recommended tags.

With the mapping scheme between out-texts and in-texts, TagRec is able to recommend the tag(s) that do not need to occur within the texts of that work item.

For the work items that were already tagged, those tags will be maintained the same for such items because the membership values corresponding to those existing tags will be among the highest.

4.1.3.7 Users' Feedbacks Integration

When a new work item is created, TagRec recommends to the developer a set of tags. (S)he is able to accept or refuse any tag and provide new tags for the work item. Let us denote the sets of accepted, rejected, and new tags for a work item w_j by T_a, T_r, and T_{new}, respectively. For the tags in T_a, their membership values with respect to all work items will be kept the same. For the tags in T_r (i.e. being rejected), the membership value μ with respect to the work item w_j will be reduced in half.

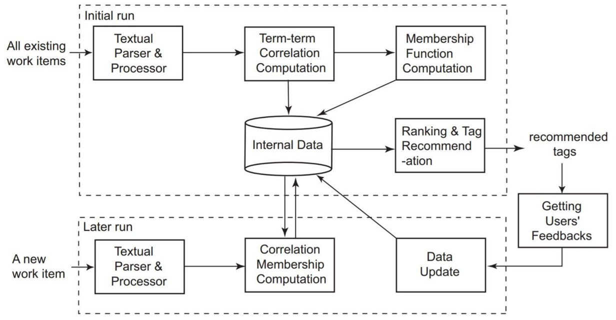

Initial run
All existing work items
Textual Parser & Processor
Term-term Correlation Computation
Membership Function Computation
Internal Data
Ranking & Tag Recommend -ation
recommended tags
Getting Users' Feedbacks
Later run
A new work item
Textual Parser & Processor
Correlation Membership Computation
Data Update

Fig.6

4.2 Bugs

4.2.1 Doc2Spec

Here using a J2EE resource, they illustrate the approach detect bugs.

Inferring specifications. This approach consists of three main steps to infer specifications from API documentation.

The first step is to extract method descriptions and class/interface hierarchies from API documentation. In this example, from J2EE's Javadoc, our approach extracts three method descriptions of interface javax.resource.cci. Connection as follows:

createInteraction():"Creates an interaction associated with this connection."

getMetaData():"Gets the information on the underlying EIS instance represented through an active connection."

close():"Initiates close of the connection handle at the application level."

The second step is to build an action-resource pair from each method description. For a method, its action-resource pair denotes what action the method takes on what resource. In this example, our approach builds the action-resource pairs for the three methods as follows:

createInteraction():⟨*create*,*connection*⟩.

getMetaData():⟨*get*,*connection*⟩.

close():⟨*close*,*connection*⟩.

As method descriptions are written in natural languages, it is difficult to define simple templates to extract action-resource pairs. In particular, the actions of createInteraction() and getMetaData() are predicate verbs, whereas the action of close() is an accusative object. Although the resources of all the three methods are preposition objects, there are multiple preposition objects in one description, and the locations of these resources are different. Here, if we simply pick those common concrete nouns as resources, we may mix specifications of different resources and infer false specifications (see Section VII-A for details). Our approach leverages an NLP technique to extract action-resource pairs accurately.

The final step is to infer automata for resources based on action-resource pairs and class/interface hierarchies. First, for each class/interface, our approach groups methods into categories according to resource names and class/interface hierarchies. In this example, the three methods are grouped into one category since their resources are of the same name and the three methods are declared by the same interface. Second, in each category, our approach maps methods to different types according to their actions. In this example, our approach maps the three methods to their types as follows:

createInteraction()→creationmethod.

getMetaData()→manipulation method.

close()→closure method.

Finally, in each category, our approach builds an automaton based on our predefined specification template shown in Figure 7. Figure 8 shows the inferred specification for the resource of interest. Our approach tailors our specification template to build the automaton shown in Figure 8

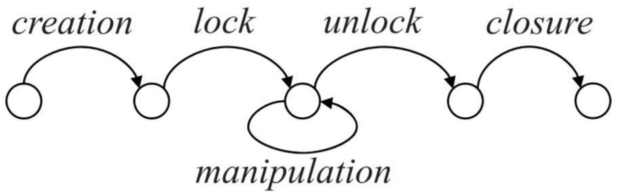


Fig.7

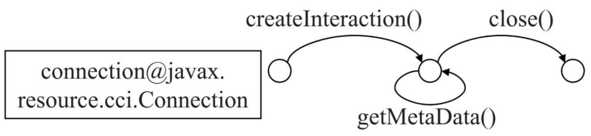


Fig.8

Detecting bugs. To confirm the usefulness of an inferred specification, they need to investigate whether they can detect bugs from violations of the specification. In this example, they can check whether close() is eventually called in all possible execution paths of a code snippet for violations of the specification shown in Figure 8. In fact, they did find such a violation in a code snippet as follows:

```
public float getHomeEquityRate()
    ...
    try{
        javax.resource.cci.Connection myCon
        =connFactory.getConnection();
        javax.resource.cci.Interaction interaction
        = myCon.createInteraction();
        ...
        myCon.close();
        ...
    }catch(Exception e){
        e.printStackTrace();
    }
}
```

In this code snippet, a local variable named *myCon* is not closed in the *exception* clause. The violation is determined as a suspected bug since unclosed connections may cause memory leaks. Such a case shows that the specification shown in Figure 8 is useful to detect bugs. Here, although Java has a garbage-collector-enabled platform, unclosed connections may still cause memory leaks. For example, the Oracle 9i JDBC Developer's Guide and Reference [17] warn "If you do not explicitly close your ResultSet and Statement objects, serious memory leaks could occur." The work proposed by Xu and Rountev [18] also focuses on memory leaks in Java programs.

5. DATASETS

The table below has the information about the various data sets used by all the papers we studied. These Datasets used in these papers for evaluation are mostly from various Community question answers sites like Stackoverflow, StackExchange, Yahoo answers and similar forums.

Paper	Datasets
Has This Bug Been Reported?	16,340 Eclipse and Mozilla bug reports. Eclipse's - 6,340 duplicate groups Mozilla's - 10,000 duplicate groups
Tag recommendation in Software Information Site	evaluations are performed on 2 software information sites, StackOverflow and Freecode, which contain 47,668 and 39,231 text documents, respectively, and 437 and 243 tags, respectively.
Fuzzy Set Approach for Automatic Tagging in Evolving Software	Data from the realworld IBM Jazz repository of 3 years
Interrogative-guided re-ranking for question-oriented software text retrieval	Dataset includes 2,460 answers with positive votes. For the re-ranking evaluation 1,826 questions with positive votes. 7,872 "lucene"-tagged answers are collected.

Finding Question Answer Pairs from Online Forums	1.) 1,212,153 threads from TripAdvisor forum 2.) 86,772 threads from Lonely Planet forum 7 3) 25,298 threads from BootsnAll Network 8 as well as 300,000 question answer pairs from Yahoo! Answers as training set.
Inferring resource specifications from natural language api documentation	687 methods in the J2SE Javadoc were used to train Doc2spec in one day. Then trained Doc2spec performed evaluations on 5 libraries.
Finding Relevant Answers in Software Forums	6068 total posts from 3 software forums , i.e 4020, 680, and 1368 posts from Oracle, SoftwareTipsandTricks and DZone forums were used for testing the tag inference engine.

6. BASELINE RESULT

For these papers , there is no common comparative study possible as some are proposing tag recommendation algorithm while others are detecting duplicates. But we do have certain papers like Paper [10] , TagCombine algorithm uses paper [3] TagRec algorithm results as it baseline for comparison.

So , in paper[10] averaging over StackOverflow and Freecode results, they improve over TagRec[3] proposed by Al-Kofahi et al. by 22.65% and 14.95%

In Paper[3], as per the author there isn't any existing fuzzy set approach for tagging , and thus the results of the algorithm cannot be compared to any base as such.

In Paper[12] , the proposed approach is compared against the built-in search engine of bugzilla as well as the lucene search engine. Fig 9 shows the time taken to train the dataset.

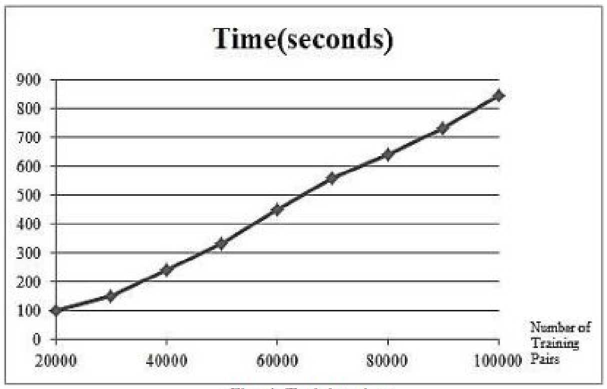


Fig. 4. Training time

Fig 9.

In Paper [1] , as compared to a standard information retrieval approach, the proposed semantic search engine framework could increase mean average precision from baseline of 17% to 71% in retrieving relevant answers to various queries.

7. OUR COMMENTARY

The initial papers have presented some novel solutions which have laid down groundwork for the future studies in the field which have later been tweaked and improved. It is quite safe to say that searching in software forums has become reliable to a certain extent. The following areas are considered to be more promising and are potential topics for future work.

7.1 Information retrieval from forums

Finding relevant question answer pairs was published in 2008, there was no previous research on the similar lines, The approach they use is unsupervised graph method and lexical mapping to find relevant pairs. Using these techniques they are able to effectively extract question-answer pairs. But lexical mapping isn't of much use for them. Finding question answer pairs is a useful technique for generating automatic answers on chatbots or search engines. Further research can also be done on Semantic Parsing of Question-Answer Pairs and Modeling Semantic Relevance in them.

For finding relevant answers there are number of work on extracting software knowledge from the web or via natural language processing techniques. In [1] the authors have used a similar technique. They process textual information in software forums rather than code. There have been a number of recent studies that analyze logged communication among users and developers to aid software engineering activities. They present a new approach to find relevant answers from software forums by leveraging an engine that automatically infers tags of posts in software forum threads. This tag inference engine automatically assigns 7 different tags to posts: question, answer, clarifying question, clarifying answer, positive feedback, negative feedback, and junk. There have been many follow up researches for this particular domain. In this study, they only investigate 3 software forums. there have been studies after this that further investigate more forums to further validate the utility of the approach.

7.2 CQA Research

Most CQA researchers focus on leveraging metadata in CQA to improve the performance of the traditional language models for information retrieval. CQA research is usually divided into two sections, some researches concentrate on question and retrieving them, while others concentrate on answers.

All the research done in this field is more or less about retrieving information in form of answers or clustering the set of similar questions. Learning to Suggest Questions in such forums will be an interesting research deduction made from search for similar question and using some machine learning principles.

7.3 Tagging

A tag is "a freely chosen keyword or term that is associated with or assigned to a piece of information. In the context of software development, tags are used to describe resources such as source files or test cases in order to support the process of finding these resources". But there is a frequent use of informal language in these forums. Informal language is not taken care of properly, so need to incorporate the fact that informal language is used in Software Forums. Despite the importance of tags, there exists limited support for automatic tagging for software artifacts. There is a scope for developing an automated approach that could automatically arrange or cluster the forum posts in a hierarchical fashion to help users in finding the right answer to his/her question thus enriching the user experience.

Furthermore this research on tagging on software forums can be extended to other domains, like use of tags on cloud to make searching of content easier, or while coding for developers.

7.4 Duplicate Removal

Duplication of bugs reporting is the area of focus for paper [12], They propose a novel approach that applies Ranking SVM, a Learning to Rank (LTR) technique. The LTR technique can be also used in software forums to detect duplicates using ranking algorithm they propose.

In paper [11] a novel approach called Doc2Spec is proposed, that uses a Natural Language Processing (NLP) technique to analyze natural language API documentation and infers resource specifications. Their inferred specifications are useful to detect defects since these specifications help detect the inconsistencies between API documentation and source code. As per our view, Doc2spec can be extended further to find duplication of information.

8. CONCLUSION

Our primary learnings from this survey/readings were the main advancements in the field of searching and its automation in Software Engineering. It has been a really good learning as far as knowledge on how search plays an important role in several information sites and how search can be improved upon. We studied several papers related to search including tagging of terms, duplicate detection and information retrieval from software forums which all come under the big umbrella of how searching can be made efficient for the end user. As well as an analysis on how internally automating search can reduce manual effort like in the case of finding duplicate bugs reported or questions asked. Our study of these papers over a span of 8 years (2008 - 2015) has shown a vast improvement in the techniques/tools used for tagging of content to improve search , as well as improvement in the detection of duplicates to enrich the user experience. The information sites, like the QA forums, software forums etc have improved in terms of search but still there is a lot of scope in this field. Finally the evolution and improvement in search has significantly reduced the time and effort spent by users on information sites and has greatly improved the user experience with all these continuously evolving techniques.

9. REFERENCES

[1] Swapna Gottipati, David Lo, and Jing Jiang, Finding Relevant Answers in Software Forums published in ASE '11 Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering

[2] www.lemurproject.org

[3] Jafar M. AlKofahi, Ahmed Tamrawi, Tung Thanh Nguyen, Hoan Anh Nguyen, and Tien N. Nguyen, Fuzzy Set Approach for Automatic Tagging in Evolving Software Published in 2010

[4] Gao Cong, Long Wang Chin Yew Lin, Young In Song, Yueheng Sun, Finding Question Answer Pairs from Online Forums Published in 2008

[5] Doris Hoogeveen, Karin M. Verspoor Timothy Baldwin, CQADupStack: A Benchmark Data Set for Community Question-Answering Research

[6] Ting Ye, Bing Xie* , Yanzhen Zou, Xiuzhao Chen, Interrogative-Guided Re-Ranking for Question-Oriented Software Text Retrieval

[7] Kai Zhang, Wei Wu, Haocheng Wu, Zhoujun Li, Ming Zhou, Question Retrieval with High Quality Answers in Community Question Answering

[8] S. E. Robertson, S. Walker, S. Jones, M. Hancock-Beaulieu, and M. Gatford. Okapi at trec-3. In TREC,

[9] C. Zhai and J. Lafferty. A study of smoothing methods for language models applied to information retrieval. ACM Trans. Inf. Syst., 22(2):179–214, 2004.

[10] Xin Xia, David Lo, Xinyu Wang and, Ha Bo Zhus, Tag recommendation in Software Information Site Proceeding Published in MSR'13 Proceedings of the 10th Working Conference on Mining Software Repositories.

[11] H. Zhong, L. Zhang, T. Xie, and H. Mei. Inferring resource specifications from natural language api documentation. In ASE, pages 307–318, 2009

[12] Kaiping Liu, Hee Beng Kuan Tan, Hee Beng Kuan Tan, Has This Bug Been Reported? Published in 2013 20th Working Conference on Reverse Engineering (WCRE)

[13] "byteMyCode," http://bytemycode.com/.

[14] D. Poshyvanyk and A. Marcus, "Combining formal concept analysis with information retrieval for concept location in source code," in ICPC'07: Int. Conference on Program Comprehension. IEEE CS, 2007

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

[16]] Y. Ogawa, T. Morita, and K. Kobayashi, "A fuzzy document retrieval system using the keyword connection matrix and a learning method," Fuzzy Sets and Systems, vol. 39, pp. 163–179, 1991.

[17] E. Perry, M. Sanko, B. Wright, and T. Pfaeffle. Oracle 9i JDBC developer's guide and reference. Technical report, http://www.oracle.com, Mar 2002

[18] G. Xu and A. Rountev. Precise memory leak detection for Java software using container profiling. In Proc. 30th ICSE, pages 151–160, 2008.

[19] S. Breu, R. Premraj, J. Sillito, and T. Zimmermann. Information needs in bug reports: Improving cooperation between developers and users. In CSCW, 2010.

[20] J. Zhou and H. Zhang, "Learning to rank duplicate bug reports," in Proceedings of the 21st ACM International Conference on Information and knowledge management. ACM, 2012, pp. 852–861.

[21] C. Manning, P. Raghavan, and H. Schutze. Introduction to Information Retrieval. Cambridge University Press, 2008.