2016 IEEE 40th Annual Computer Software and Applications Conference

It Takes Two to Tango: Deleted Stack Overflow Question Prediction with Text and Meta Features

Xin Xia*, David Lo†, Denzil Correa‡, Ashish Sureka§, and Emad Shihab¶
*College of Computer Science and Technology, Zhejiang University, Hangzhou, China
†School of Information Systems, Singapore Management University, Singapore
‡Max Planck Institute for Software Systems (MPI-SWS), Germany
§ABB Corporate Research, India
¶Department of Computer Science and Software Engineering, Concordia University, Montreal, Canada
xxia@zju.edu.cn, davidlo@smu.edu.sg, denzil@mpi-sws.org, ashish.sureka@in.abb.com, eshihab@cse.concordia.ca

***Abstract*—Stack Overflow is a popular community-based Q&A website that caters to technical needs of software developers. As of February 2015 – Stack Overflow has more than 3.9M registered users, 8.8M questions, and 41M comments. Stack Overflow provides explicit and detailed guidelines on how to post questions but, some questions are very poor in quality. Such questions are *deleted* by the experienced community members and moderators. Deleted questions increase maintenance cost and have an adverse impact on the user experience. Therefore, predicting deleted questions is an important task.**

In this study, we propose a two stage hybrid approach – *DelPredictor* – which combines text processing and classification techniques to predict deleted questions. In the first stage, *DelPredictor* converts text in the title, body, and tag fields of questions into numerical textual features via text processing and classification techniques. In the second stage, it extracts meta features that can be categorized into: profile, community, content, and syntactic features. Next, it learns and combines two independent classifiers built on the textual and meta features. We evaluate *DelPredictor* on 5 years (2008–2013) of deleted questions from Stack Overflow. Our experimental results show that *DelPredictor* improves the F1-scores over baseline prediction, a prior approach [12] and a text-based approach by 29.50%, 9.34%, and 28.11%, respectively.

***Keywords*—Deleted Question, Stack Overflow, Text Processing, Classification**

I. Introduction

Software engineers frequently make use of online media, referred to as software information sites [39], to assist them in solving problems in the process of software development and maintenance. Software information sites have changed the way software engineers communicate, collaborate, and share information [8], [31]. Among these software information sites, community-based Q&A websites are very popular. Stack Overflow [2] is one of the most popular and important community-based Q&A websites where software engineers ask and answer technical questions about programming. As of February 2015, Stack Overflow has more than 3.9M registered users, 8.8M questions, and 41M comments, and it would receive more than 3,700 new posted questions per day. The maintenance of Stack Overflow is difficult due to the large number of users and questions [11], [12].

To help software engineers post good questions, Stack Overflow has established explicit and detailed guidelines [1]. However, a number of questions are off topic or of very low quality, hence they get deleted by moderators or reputable users. For example, Figure 1[1] presents a question about whether repetitive strain injury (RSI) affect legs, the question was deemed off-topic since it is not related to programming or software engineering activities.

These deleted questions add extra work for Stack Overflow moderators, and the presence of poor quality questions negatively affects user experience. These deleted question make good quality questions less observable to potential answerers and users searching Stack Overflow, since these questions can be buried among poor quality ones. Prior studies showed that poor quality contents on question and answer (Q&A) websites (e.g., Stack Overflow) drive users away and result in user attrition [12], [20]. Currently, poor quality questions are manually deleted in Stack Overflow, but considering the large number of questions posted daily on Stack Overflow, the manual effort involved is high. A prior work found there are 293,289 deleted questions from the years of 2008–2013 on Stack Overflow [12]. Thus, an automated approach that assists in predicting whether a question would be deleted will greatly help. Deleted question prediction is not only an important and practically useful problem, but also technically challenging, due to the wide variety of user-generated contents on Stack Overflow [11], [12].

In this paper, we propose an approach named *DelPredictor* that employs both text processing and a two-stage ensemble classification for deleted question prediction. In the first stage, we first collect textual contents in the title, body, and tags of questions. Next, we apply text processing and classification techniques to convert these textual contents into numerical textual features. In total, we extract 4 kinds of textual features, namely title, body, mixed, and tags, which correspond to the textual contents extracted from the title, body, both the title and body, and tag fields of the questions. In the second stage, we extract 47 meta-features that were proposed by Correa and Sureka [12] and can be grouped into 4 categories: profile, community, content, and syntactic. For example, considering the user who posts a question, we extract the number of days the user's account exists, the number of questions that the user has posted before, and the average score that the user received

[1]Note, the question is no longer available in the current Stack Overflow website (since it has been deleted). It is available in the Stack Overflow data dump [4].

0730-3157/16 $31.00 © 2016 IEEE
DOI 10.1109/COMPSAC.2016.145

Title: Does RSI affect legs?
Body: Wikipedia says that RSI is also called 'work related upper limb disorder', but I'm getting serious knee pain when I'm sat working for long periods (18+ hours). Has anyone else experienced this, and have you found a solution?
Tags: ergonomics, rsi
User: Peter Coulton

Fig. 1. Question 4452 on Stack Overflow.

for his/her previous answers; considering the textual content of a question, we extract the number of URLs in the question, the length of the question, and the length of code snippets in the question. Next, we build two independent classifiers based on the textual features and the meta features respectively, and combine them by assigning different weights to the two classifiers. The weights are automatically tuned to achieve the best F1-score [16] in a set of training data.

Prior to our work, Correa and Sureka proposed the first approach that can predict deleted questions; their approach extracts 47 features from questions and uses AdaBoost to learn a prediction model [12]. Different from their work, our work considers *thousands* of new features (i.e., the various words in the questions), and we group them into four advanced textual features. Also, we utilize the advantages of text mining and classification techniques to predict deleted questions more effectively.

We evaluate *DelPredictor* on a large dataset which contains a total of 417,685 questions from Stack Overflow. We compare our approach with 3 baseline approaches, i.e., the state-of-the-art approach proposed by Correa and Sureka [12], random prediction that randomly predicts a question to be deleted or not according to the ratio of deleted questions to total questions, and an approach which only uses raw textual contents of questions (denoted as text-only). The experiment results show that DelPredictor achieves a F1-score of 0.597, which improves over the approach proposed by Correa and Sureka, random prediction, and text-only approaches by 9.34%, 29.50%, and 28.11%, respectively.

The main contributions of this paper are:

1) We propose a new deleted question prediction approach *DelPredictor*, which extends the prior work by Correa and Sureka [12] by (1) the consideration of additional textual features (see Section IV-A), (2) the incorporation of an automated tuning step to optimize classifier performance (see Section IV-B2), and (3) the usage of a composition of classifiers (see Section IV-B3).
2) We evaluate our approach with other approaches, such as Correa and Sureka's approach [12], on a large dataset which contains a total of 417,685 questions from Stack Overflow. The experimental results show that our approach can achieve a substantial improvement over the baseline approaches.

The remainder of the paper is organized as follows. Section II contrasts an example deleted question with a frequently asked question and highlights their differences. Section III

Fig. 2. An example question 218384 in Stack Overflow.

Title: Calling RJava function in dbApply?
Body: out $<-$ dbApply (sa_data, INDEX = "len_diff", FUN = J("myclassname") & getNumberOfRevisions ({text_before column}, {text_after column})
Tags: java, mysql, r, rjava, rmysql
User: saigafreak

Fig. 3. Question 6357962 on Stack Overflow.

describes an overview of *DelPredictor*'s architecture. Section IV elaborates on the details of each stage of *DelPredictor*. Section V presents the results of our comparative evaluation of *DelPredictor*. Section VI discusses additional points on the benefits and limitations of our approach. Section VII surveys the related work. Finally, Section VIII concludes the paper.

II. Deleted Questions

A typical question in Stack Overflow contains a number of fields including a title, a body, one or more tags, and a user. The title and body fields provide a brief and detailed description of the question. The tag field contains a set of tags, where a tag is a keyword or label that categorizes a question and links it to other similar questions. The user field indicates the user who posted the question. Figure 2 presents an example question, with identifier 218384, which asked about the root cause of null pointer exception and the way to fix it. The question is one of the frequently asked questions for Java programmers, and it has more than 490,000 views, and received 12 answers. To make a comparison, Figure 3 presents a deleted question, with identifier 6357962. The question asked whether we can call RJava function in dbApply. In the body field, the user only provided a code snippet, and no further description was provided.

A. Observations and Implications

From these 2 questions, and also the previous question shown in Figures 1, we make the following observations:

1) The textual contents in the title and body fields are good indicators to identify whether a question will be deleted. For example, the question in Figure 2 is well written and easy to understand, while the question in Figure 3 is hard to understand. Also, for the question in Figure 1, we can easily decide that it is off-topic after reading it.
2) The tags can help to predict whether a question will be deleted. Questions attached with some particular tags are more likely to be deleted than others. For example, the tag "rmysql" in Figure 3 only appears in 5 questions in our collected data, and 3 out of the 5 questions are deleted questions. Similarly, the

tag "ergonomics" in Figure 1 only appears in 40 questions in our collected data, and 39 out of the 40 questions are deleted questions.

3) Experienced users are likely to post good questions, and new users are likely to post poor quality questions that are eventually deleted. For example, Ziggy that posted the question shown in Figure 2 has a reputation of 5,359, while saigafreak that posted the question in Figure 3 only has a reputation of 94 and 31 respectively.

The above observations tell us that the textual content in the title, body, and tag fields provide some useful and yet hidden information that can help us identify deleted questions. Text mining techniques that mine hidden information from a large text corpus, can potentially be used to solve the problem. Moreover, by considering the importance of users who posted the questions, the performance of a text-mining-based deleted question prediction approach can be further improved. In the following sections, we describe the details of DelPredictor, which is based on the above observations.

B. Relevance to Software Engineering

Notice our proposed DelPredictor not only help moderators to better organize the community based Q&A site by deleting the low quality questions, but also help developers to improve their productivity by increasing their search efficiency. Consider the following scenarios:

Without DelPredictor. Xin is a junior developer in a IT company. One day he wants to solve a programming problem, for example, how to use RJava in the dbApply class. He searches solutions on Stack Overflow which has a number of low quality questions, and he finds a questions shown in Figure 3 which is quite similar to his query. However, it is a low quality question, after reading the question and answer, Xin is still puzzled. Thus, Xin has to spend more time to search for the solution to the programming problem, and due to a number of low quality questions in Stack Overflow, he wastes much time to find the answer.

With DelPredictor. Xin is a junior developer in a IT company. One day he wants to solve a programming problem, for example, how to use RJava in the dbApply class. He searches solutions on Stack Overflow. Stack Overflow moderators have deleted many low quality questions using DelPredictor. Since the quality of the remaining questions in StackOveflow is high, Xin finds the answer very fast and can solve the programming problem well.

III. DelPredictor Architecture

Figure 4 presents the overall architecture of *DelPredictor*, which contains two phases: a model building phase and a prediction phase. In the model building phase, our goal is to build a model from historical questions with known labels (deleted or not). The model building phase consists of two stages: a text mining stage (Stage 1), and a composition stage (Stage 2). In the prediction phase, this model is used to predict if an unknown question will be deleted or not.

In the text mining stage, our goal is to convert textual contents of questions into *numerical textual features*. We first extract textual contents from questions (Step 1). We consider 4 types of textual contents, i.e., textual contents from the title, body, title and body, and tags of questions. For each of the 4 types, we extract term features from it (Step 2)[2], and build a classifier based on the term features (Step 3). Next, we use the classifiers to output the confidence score of each of the questions to be a deleted question (Step 4). These confidence scores are the *numerical textual features* (or textual features for short). In total, we have 4 textual features, $Text^{title}$, $Text^{body}$, $Text^{mixed}$, and $Text^{tags}$, corresponding to the confidence scores computed by the text mining classifiers built on the textual contents of the title, body, title and body, and tags, of questions in our training data respectively[3].

In the composition stage, our goal is to build a composite classifier based on the (numerical) textual features and meta features. We first build a classifier based on the 4 textual features extracted in the text mining stage (Step 5). Next, we extract meta features from the training set of questions (Step 6). We extract the same 47 meta features proposed and tested by Correa and Sureka [12], which are grouped into 4 categories: profile, community, content, and syntactic[4]. Next, we build a meta classifier based on the extracted meta features (Step 7). We then combine the 2 classifiers together to construct a composite classifier (Step 8)[5].

After the composite classifier is constructed, in the prediction phase it is then used to predict whether a question with an unknown label will be deleted or not. For each of such questions, we first extract the values of its textual features and meta features (Step 9). We extract the textual features following the process performed in the model building phase. For each type of textual content, we compute the value of the (numerical) textual feature by leveraging the 4 classifiers that we have built in the first stage of the model building phase. We then input the feature values to the composite classifier (Step 10) that outputs the prediction result, which is one of the following labels: deleted or non-deleted (Step 11).

IV. Our Proposed Approach

In this section, we present the details of DelPredictor. We presents the details of the text mining and composition stage.

A. Text Mining Stage

1) Term Feature Extraction: We process the textual contents in the title and body of questions to extract the term features in 3 steps [5] – tokenization, stop-word removal, and stemming.

Tokenization. In the process of tokenization, a stream of text is split into words, symbols, or other meaningful elements called tokens.

Stop-word Removal. Stop words are words that are used very frequently and thus can not help to distinguish between deleted and non-deleted questions. For example, words such as "the",

[2]For more details of the term features, please refer to Section IV-A1.

[3]For more details of the numerical textual features, please refer to Section IV-A2.

[4]For more details of the meta features, please refer to Section IV-B1.

[5]For more details of the composite classifier, please refer to Section IV-B3.

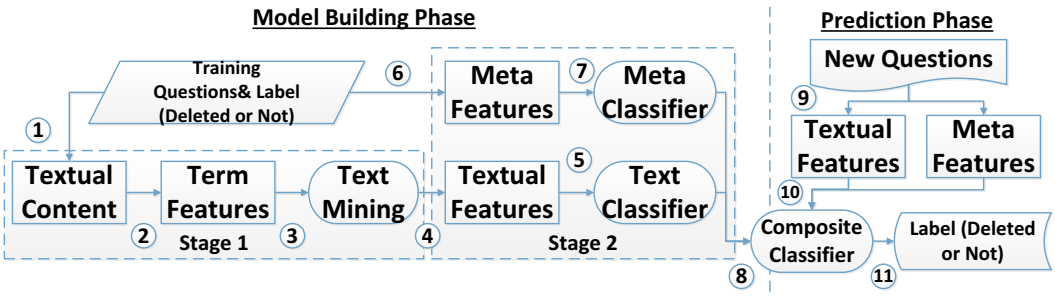

Fig. 4. Overall architecture of DelPredictor.

"a", "she", "I" are stop words. In this paper, we use a set of standard English stop words that come with WVTool[6]. These stop words are removed from the extracted word tokens. We also remove all numbers, punctuation marks, and HTML tags (e.g., <code>, <p>).

Stemming. In this step, the inflected or derived words are reduced to their stem, base or root form. For example, "read" and "reading" are all reduced to "read". In this paper, we use the Porter stemmer[7] to reduce a word token to its representative root form.

We extract 3 types of term features from the title and body of questions. They are processed word tokens taken from the textual contents in the title, body, and both title and body of questions. We denote them as $Term^{title}$, $Term^{body}$, and $Term^{mixed}$ respectively. For each question, the value of each feature is the number of times the corresponding word token appears in it.

For textual contents in the tag field, to keep the original semantic meanings of the tags, we do not perform any of the above text processing steps. For example, for the 3 tags "C#", "C++", and "C", we do not run the text precessing steps, otherwise all of them would become "C". This would change the semantic meanings of 2 out of the 3 tags, and potentially reduce the effectiveness of a prediction technique that learns from the tags. We use each of the tags as a term feature, and denote the set of term features extracted from the tags as $Term^{tags}$. For each question, the value of each feature is one if the corresponding tag appears in the question, and zero otherwise.

2) Textual Feature Extraction: The number of term features are often too many, in this step, we want to reduce them into a few numerical values, which we refer to as (numerical) textual features. In the model building phase, we first divide the training data set into two subsets using stratified random sampling. Using stratified random sampling, the distribution and number of deleted and non-deleted questions in both training subsets are kept the same. We train a text mining classifier from the first training subset, and use it to assign a confidence score (i.e., likelihood of a question to be deleted) to each question in the second training subset. Similarly, we also train a classifier with the second training subset, and use it to obtain the confidence scores of questions in the first training subset. These confidence scores are used as the values of the textual features. By this way, we convert the term features $Term^{title}$, $Term^{body}$, $Term^{mixed}$, and $Term^{tags}$ into 4 simple numerical textual features $Text^{title}$, $Text^{body}$, $Text^{mixed}$, and $Text^{tags}$.

In the prediction phase, for a new question, we leverage the text mining classifiers that are built on all of the training questions to compute the values of the textual features. By default, we build the text mining classifiers using the Naive Bayes multinomial algorithm [21], since it is a fast and effective classification algorithm. We also use complement naive Bayes proposed by Rennie et al. [29] to build the text mining classifiers in research question 3 (RQ3)[8]. Both naive Bayes multinomial and complement naive Bayes are suitable for datasets with a large number of features (in our case, processed words in raw text data). For many other classification algorithms, such as decision tree, random forest, and AdaBoosting, they would take a long time to be run on a raw text dataset.[9]

B. Composition Stage

1) Meta Feature Extraction: We extract the 47 meta features that were proposed by Correa and Sureka [12]. The 47 meta features are grouped into 4 categories: profile, community, content, and syntactic. In the profile category, the features are defined based on the historical statistics of the user who posted the question. In the community category, the features are defined based on the crowdsourced information generated by the Stack Overflow community. In the content category, the features are defined based on the textual content of the question. In the syntactic category, the features are defined based on the writing style of the textual contents. Table I summarizes the meta features used in the paper – for more details, please refer to [12].

Note that the features in the content and syntactic categories are different from our term and textual features. They are counts of various things (e.g., number of words in the title field, number of special characters, etc.) and do not capture individual words used in questions and their semantics. Two documents can have the same values of the meta features but do not share any word in common. Similarly, two documents can be very similar in their word contents, however are very different in their meta feature values.

2) Threshold Learning Classifier: After we extract the textual features and meta features, we build two independent classifiers. By default, we use random forest [9], which

[6]http://sourceforge.net/projects/wvtool/
[7]http://tartarus.org/Xmartin/PorterStemmer/

[8]For more details of RQ3, please refer to Section V-C.
[9]Our preliminary study shows that they can even take days to complete.

TABLE I. META FEATURES PROPOSED BY CORREA AND SUREKA [12]. # DENOTES "NUMBER OF".

Category	Features
Profile	Age of account, # previous questions with negative scores, # previous questions with positive scores, # previous questions with 0 scores, # previous answers with negative scores, # previous answers with positive scores, # previous answers with 0 scores, # previous questions, # previous answers, # previous badges, (# previous questions)/(age of account), (# previous answers)/(age of account)
Community	Average score for previous answers, Average score for previous questions, Average view counts for previous questions, Average number of comments received, Average number of accepted answers, Average favorite votes
Content	# URLs, # tags, length of code snippet, # personal pronouns, # pronouns, # space words, # relativity words, # inclusive words, # cognitive process words, # social words, # first person singular pronouns
Syntactic	# function words, # conjunctions, # prepositions, # characters in body field, # alphabetical characters in body field, # upper case characters in body field, # lower case characters in body field, # digit characters in body field, # white case characters in body field, # special characters in body field, # punctuation marks in body field, # words in body field, # short words in body field, # unique words in body field, Average length of words in body field, # characters in title field, # words in title field, Average length of words in body field

has been widely used in past software engineering studies, e.g., [22], [34], [37], to construct the two classifiers. For this step, we use random forest instead of naive Bayes multinomial since we only have 4 and 47 features to build each of the classifiers, and previous studies have shown that the random forest algorithm can complete training quickly and is more accurate than naive Bayes for datasets with a small number of features (e.g., ≤ 200) [10].

In the model building phase, random forest constructs a number of decision trees [16]. In the prediction phase, random forest processes each new instance (in our case, each new question) into the decision trees, and predicts the label of the instance by considering the outputs of the decision trees. Random forest will output a confidence score based on the number of trees that predict that the new instance will be deleted. If this score is beyond a threshold TH, then the new instance is predicted to be deleted. By default, TH is set to 0.5. However, the optimal threshold value might not be 0.5, and it can differ for different datasets. A suitable setting of this threshold affects the classification performance.

In this paper, we propose an approach that automatically estimates a good threshold in the model building phase. The pseudo-code of our approach is shown in Algorithm 1. It takes as input a training set of questions $Train$. It then randomly divides $Train$ into two subsets, T_1 and T_2 (Line 6). T_1 contains 80% of the questions in $Train$, T_2 contains the remaining 20%. Next, we build a classifier using T_1 (Line 7), and for each question $ques$ in T_2, we also compute its confidence score (i.e., likelihood for it to be a deleted question), denoted as $Score_{del}(ques)$ (Line 8). Finally, to tune

Algorithm 1 $EstimateThreshold$: Estimation of Threshold

1: **EstimateThreshold(**$Train$**)**
2: **Input:**
3: $Train$: Training Historical Questions
4: **Output:** TH
5: **Method:**
6: Randomly divide $Train$ into 2 subsets T_1 and T_2 where T_1 contains 80% of the questions and T_2 contains the remaining 20%;
7: Built a classifier C on T_1;
8: **for all** Question $ques$ in T_2 **do**
9: Compute the confidence score of $ques$ (i.e., its likelihood to be a deleted question) by using C;
10: **end for**
11: **for all** TH from 0 to 1, every time increase TH by 0.01 **do**
12: Predict the labels for questions in T_2 according to Equation (1);
13: Compute the F1-score on T_2;
14: **end for**
15: **Return** TH which maximizes the F1-score for questions in T_2

the threshold value, we gradually increase the threshold from 0 to 1 (we increase the threshold by 0.01 each time), and for each question $ques$ in T_2, we predict its label $Predict(ques)$ using the following equation:

$$Predict(rele) = \begin{cases} \text{Deleted}, & \text{if Score}_{del}(ques) \geq \text{TH} \\ \text{Non-deleted}, & \text{Otherwise} \end{cases} \quad (1)$$

We output the threshold that maximizes the F1-score for the questions in T_2 (Lines 11 - 15).

3) Composite Classifier: We denote the classifiers built on textual features and meta features as C_{text} and C_{meta}. Given a new question, C_{text} and C_{meta} output the following textual score and meta score, respectively:

Definition 1: (**Textual Score.**) Consider a text classifier C_{text} built from the textual features of questions in the training set, and a threshold TH estimated using Algorithm 1. For a new question $ques$, we use C_{text} to get its confidence score (i.e., its likelihood to be a deleted question) – denoted as $\text{Score}_{del}^{text}(ques)$. The textual score of $ques$, denoted as $Text(ques)$, is computed as:

$$Text(ques) = \frac{\text{Score}_{del}^{text}(ques)}{\text{Score}_{del}^{text}(ques) + TH} \quad (2)$$

Definition 2: (**Meta Scores.**) Consider a meta classifier C_{meta} built on the meta features of questions in the training set, and a threshold TH estimated using Algorithm 1. For a new question $ques$, we use C_{meta} to get its confidence score – denoted as $\text{Score}_{del}^{meta}(ques)$. The meta score of $ques$, denoted as $Meta(ques)$, is computed as:

$$Meta(ques) = \frac{\text{Score}_{del}^{meta}(ques)}{\text{Score}_{del}^{meta}(ques) + TH} \quad (3)$$

The composite classifier combines the textual classifier and meta classifier by assigning different weights to them. We

define the composite score that is output by the composite classifier as follows:

Definition 3: (**Composite Score.**) Consider a text classifier C_{text}, and a meta classifier C_{meta} built on the textual features and meta features of questions in the training data. For a new question $ques$, its composite score $Comp(ques)$ is a linear combinations of its textual score $Text(ques)$ and meta score $Meta(ques)$:

$$Comp(ques) = \alpha \times Text(ques) + (1-\alpha) \times Meta(ques) \quad (4)$$

In the above equation, $\alpha \in [0, 1]$. $\alpha = 0$ implies that only the meta classifier is used, while $\alpha = 1$ implies that only the textual classifier is used. If $Comp(ques) \geq 0.5$, we predict that the question is a deleted question, else it is a non-deleted question.

To automatically produce good α value for the composite classifier, we propose a greedy algorithm as shown in Algorithm 2. Similar to the threshold estimation process in Algorithm 1, we input the training data $Train$, and randomly divide $Train$ into two subsets, T_1 and T_2 (Line 6). Next, we build the text classifier C_{text} and meta classifier C_{meta} on T_1 (Line 7), and for each question $ques$ in T_2, we compute its textual score and meta score (Line 8). Finally, to tune α, we gradually increase α from 0 to 1 (increasing α by 0.01 each time). For each α value, we predict the label of each question $ques$ in T_2 (i.e., deleted or not) according to Definition 3. We output the α value that maximizes the F1-score for questions in T_2 (Lines 11 - 15).

Algorithm 2 $Estimate\alpha$: Estimation of α value.

1: $Estimate\alpha(Train)$
2: **Input:**
3: $Train$: Training Historical Questions
4: **Output:** α
5: **Method:**
6: Randomly divide $Train$ into 2 subsets T_1 and T_2 where T_1 contains 80% of the questions and T_2 contains the remaining 20%;
7: Build the text classifier and meta classifier C_{text} and C_{meta} on T_1;
8: **for all** Question $ques$ in T_2 **do**
9: Compute the textual score and meta score of $ques$ by using C_{text} and C_{meta};
10: **end for**
11: **for all** α from 0 to 1, every time increase α by 0.01 **do**
12: Predict the labels of questions in T_2 according to Definition 3;
13: Compute the F1-score on T_2;
14: **end for**
15: **Return** α which maximizes the F1-score for questions in T_2

V. Experiments and Results

In this section, we evaluate the performance of DelPredictor. The experimental environment is a Windows Server 2008, 64-bit, Intel Xeon 2.00GHz server with 80GB RAM.

TABLE II. Statistics of the collected dataset.

Time Period	2008. 08 – 2013. 06
Num. of questions	471,685 questions
Num. of deleted questions	194,130 questions
Num. of non-deleted questions	223,555 questions

A. Experiment Setup

We use the same dataset as Correa and Sureka [12] which contains a total of 470,096 questions. We further pre-process the dataset by removing questions which have only 1 tag and the tag only appear 1 time to reduce noise due to rare tags. Table II presents the statistics of the dataset. In total, we analyze 417,685 questions in Stack Overflow, and among these questions, 194,130 questions are deleted questions, which accounts for 46.5% of the total number of questions. To determine whether a question is a deleted question, all of the 24 database dumps [4] provided by Stack Overflow from August 2008 to June 2013 are downloaded, and if a question appeared in at least one of the previous database dumps, but not in the database dump from June 2013, the question is considered a *deleted* question. We intentionally choose questions that have been published for a long time to ensure that the contents of the questions have stabilized (i.e., no edits are likely to be done any more), the questions are answered and closed, and sufficient time has elapsed to allow more chance for deleted questions to be identified by Stack Overflow users and moderators.

To investigate whether *DelPredictor* can be used to identify deleted questions in the same setting as the one in practice, we performed an experiment using a longitudinal data setup. We sort the questions in the order they are posted (i.e., temporally) and split them into 11 non-overlapping time windows of equal sizes, numbered 0 to 10. The process then proceeds as follows: First, in fold 1, we train using questions in window 0, and test the trained model using questions in window 1. Then, in fold 2, we train using questions in window 1, and test the trained model using questions in window 2. We proceed in a similar manner for the next folds. In the final fold (i.e., fold 10), we train using questions in window 9, and test using questions in window 10. We record the average performance across the 10 folds.

By default, *DelPredictor* uses naive Bayes multinomial to build the 4 classifiers in the text mining stage, and random forest to build the text and meta classifiers in the composition stage. We use the implementation of naive Bayes multinomial and random forest in Weka [15], and set the parameters of these two approaches as the default setting in Weka.

We compare our DelPredictor with the approach proposed by Correa and Sureka [12], random prediction, and variants of an approach that only uses textual contents of questions. Correa and Sureka propose the usage of AdaBoosting [13] to predict deleted questions; we use the same setting of Correa and Sureka, i.e., we use decision tree as the base classifier, and set the number of iterations as 100. For random prediction, a question is randomly predicted to be deleted or not according to the ratio of the number of deleted questions to the total number of questions. The precision of random prediction is the percentage of deleted questions in the data set. Since random prediction is a random classifier with two possible outcomes (e.g., deleted/non-deleted), its recall is 0.5. In the text only approach, we predict whether a question would be deleted or

TABLE III. AVERAGE PRECISION, RECALL AND F1-SCORE OF DELPREDICTOR COMPARED WITH THE BASELINE APPROACHES.

Approaches	Precision	Recall	F1-score
DelPredictor	0.507	0.832	0.597
Correa and Sureka's	0.535	0.604	0.546
Random Prediction	0.464	0.500	0.461
TO^{title}	0.452	0.492	0.458
TO^{body}	0.452	0.502	0.466
TO^{mixed}	0.451	0.500	0.465
TO^{tags}	0.452	0.488	0.454
Text Classifier	0.456	0.451	0.442
Meta Classifier	0.597	0.518	0.526
$Single + TH$	0.521	0.668	0.546
$Single$	0.545	0.493	0.496

not by using only the textual contents in the question. Naive Bayes multinomial is used to build the text only classifier. Considering that we can build the text-only classifier based on the textual content in the title, body, both title and body, and tags, we have four variants of the text-only approach, and we denote them as TO^{title}, TO^{body}, TO^{mixed}, and TO^{tags}.

B. Evaluation Metrics

There are four possible outcomes for a question in the test data: a question can be classified as a deleted question when it truly is a deleted question (true positive, TP); it can be classified as a deleted question when it is actually a non-deleted question (false positive, FP); it can be classified as a non-deleted question when it is actually a deleted question (false negative, FN); or it can be classified as a non-deleted question and it truly is a non-deleted question (true negative, TN). Based on these possible outcomes, precision, recall and F1-score are defined as:

Precision: the proportion of questions that are correctly labeled as deleted questions among those labeled as deleted questions, i.e., $P = \frac{TP}{TP+FP}$.

Recall: the proportion of deleted questions that are correctly labeled, i.e., $R = \frac{TP}{TP+FN}$.

F1-score: a summary measure that combines both precision and recall - it evaluates if an increase in precision (recall) outweighs a reduction in recall (precision), i.e., $F = \frac{2 \times P \times R}{P+R}$.

There is a trade-off between precision and recall. One can increase precision by sacrificing recall (and vice versa). In *DelPredictor*, we can sacrifice precision (recall) to increase recall (precision), by manually lowering (increasing) the value of the threshold TH parameter in Equation (1). This trade-off causes difficulties in comparing the performance of several prediction models by using precision or recall alone [16]. For this reason, we compare the prediction results using the F1-score, which is the harmonic mean of precision and recall. This follows the setting used in numerous other software analytics studies [22], [23], [30], [34], [37], [38], [40]. In general, the higher the F1-score is, the better the performance of an approach is.

C. Research Questions

We are interested in the following three research questions:

RQ1: How effective is DelPredictor at predicting deleted questions? How much improvement can it achieve over other state-of-the-art approaches?

Motivation. Moderators in Stack Overflow can use DelPredictor to identify deleted questions. The more accurate DelPredictor is, the more benefit DelPredictor would give to its users. Hence, we first set out to evaluate the accuracy of DelPredictor with respect to other state-of-the-art approaches.

Approach. To address RQ1, we compare DelPredictor with the approach proposed by Correa and Sureka [12], random prediction, and the text-only approaches, i.e., TO^{title}, TO^{body}, TO^{mixed}, and TO^{tags}. Since *DelPredictor* has two sub-classifiers (text classifier and meta classifier), we also investigate the performance of each of them. We want to see whether the combination of the two classifiers can achieve better result than the individual classifier. Additionally, we build a single classifier trained using all of the 51 features (4 textual features and 47 meta features), and compare the performance of the single classifier with *DelPredictor*. We use random forest to build the single classifier and denote it as $Single$. Furthermore, we also enhance $Single$ with our threshold learning step presented in Section IV-B2. We use this enhanced single classifier, denoted as $Single + TH$, as another baseline. We evaluate them by using the longitudinal data setup, and record the average precision, recall, and F1-scores across the 10 folds.

Results. Tables III presents the average precision, recall, and F1-scores for DelPredictor compared with those of Correa and Sureka's approach, random prediction, TO^{title}, TO^{body}, TO^{mixed}, and TO^{tags}, respectively. On average across the 10 folds, DelPredictor achieves precision and recall values of 0.507 and 0.832, respectively. Precision and recall are both important metrics for deleted question prediction since they measure quality in two aspects. Low precision means a high number of false positives. On the other hand, low recall means that most deleted questions are not identified as such. Since there is a trade off between precision and recall [16], we use the F1-score, which is the harmonic mean of precision and recall, to compare the performance of the different approaches.

From Table III, on average across the 10 folds, DelPredictor can achieve F1-scores of 0.597. The improvement of DelPredictor over the baseline approaches are substantial in terms of F1-score. On average across the 10 folds, DelPredictor improves the F1-scores of Correa and Sureka's approach, random prediction, TO^{title}, TO^{body}, TO^{mixed}, TO^{tags}, text classifier, meta classifier, $Single+TH$, and $Single$ by 9.34%, 29.50%, 30.35%, 28.11%, 28.39%, 31.50%, 35.07%, 13.50%, 9.34%, and 20.36%, respectively.

RQ2: How effective is DelPredictor when different underlying classifiers are used?

Motivation. By default, in the text mining stage, we set the underlying classification algorithm of DelPredictor as naive Bayes multinomial; in the composition stage, we set the underlying classification algorithm as random forest. There are various classifiers which can also be used by the two stages of DelPredictor. We would like to investigate the effectiveness of our approach using different underlying classifiers.

Approach. To address RQ2, in the first stage, we set the underlying classification algorithm as naive Bayes multinomial, and complement naive Bayes proposed by Rennie et al. [29]. In the composition stage, we use random forest, naive Bayes [16], decision tree [16], ADTree [15], and AdaBoosting [13] as

TABLE IV. AVERAGE PRECISION, RECALL, AND F1-SCORES OF DELPREDICTOR WITH DIFFERENT UNDERLYING CLASSIFIERS. NBM = NAIVE BAYES MULTINOMIAL, FOREST = RANDOM FOREST, NAIVE = NAIVE BAYES, TREE = DECISION TREE, ADA = ADABOOSTING, COMPNB = COMPLEMENT NAIVE BAYES.

Approaches	Precision	Recall	F1-score
DelPredictor$^{Forest}_{NBM}$ (Default)	0.507	0.832	0.597
DelPredictor$^{Naive}_{NBM}$	0.463	0.749	0.543
DelPredictor$^{Tree}_{NBM}$	0.491	0.743	0.550
DelPredictor$^{ADTree}_{NBM}$	0.483	0.806	0.575
DelPredictor$^{Ada}_{NBM}$	0.508	0.794	0.580
DelPredictor$^{Forest}_{CompNB}$	0.506	0.786	0.579
DelPredictor$^{Naive}_{CompNB}$	0.464	0.709	0.522
DelPredictor$^{Tree}_{CompNB}$	0.499	0.660	0.528
DelPredictor$^{ADTree}_{CompNB}$	0.509	0.800	0.587
DelPredictor$^{Ada}_{CompNB}$	0.504	0.784	0.572

TABLE V. TOP-20 MOST DISCRIMINATIVE FEATURES.

1	# tags	0.032
2	Length of code snippet	0.029
3	$Text^{tags}$	0.021
4	Average number of accepted answers	0.017
5	# previous badges	0.017
6	Average score for previous answers	0.017
7	previous questions with positive scores	0.015
8	# previous questions	0.014
9	# punctuation marks characters in body field	0.014
10	# special characters in body field	0.014
11	Age of account	0.013
12	$Text^{title}$	0.013
13	Average score for previous questions	0.013
14	$Text^{mixed}$	0.012
15	$Text^{body}$	0.010
16	# previous answers	0.010
17	Average number of comments received	0.010
18	# previous answers with positive scores	0.010
19	# white case characters in body field	0.009
20	(# previous questions)/(age of account)	0.009

the underlying classifiers of DelPredictor. In total we have 10 different compositions of different underlying classifiers in the two stages of DelPredictor. We denote the DelPredictor with the algorithm a in the text mining stage, and the algorithm b in the composition stage as DelPredictor^b_a. For example, if we choose complement naive Bayes and naive Bayes as the two underlying classifiers, we denote our approach as DelPredictor$^{naive}_{compNB}$. We evaluate them using the longitudinal data setup, and record the average precision, recall, and F1-scores across the 10 folds.

Results. Table IV presents the average precision, recall, and F1-scores of DelPredictor with different underlying classifiers. We notice for different underlying classifiers in the two stages, the performance of DelPredictor would be varied. Comparing the two text mining algorithms in the text mining stage, we notice the naive Bayes multinomial achieves a better performance than complement naive Bayes. For example, the F1-score of DelPredictor$^{Forest}_{NBM}$ is 0.597, while the F1-score of DelPredictor$^{Forest}_{CompNB}$ is only 0.579. Also, comparing the five classification algorithms in the composition stage, we notice DelPredictor with ADTree, AdaBoosting and random forest achieve much better performance than those with naive Bayes and decision tree. For example, the F1-scores of DelPredictor$^{Forest}_{NBM}$, DelPredictor$^{ADTree}_{NBM}$, and DelPredictor$^{Ada}_{NBM}$ are 0.597, 0.575, and 0.580 respectively, while the F1-scores of DelPredictor$^{Naive}_{NBM}$ and DelPredictor$^{Tree}_{NBM}$ are 0.543 and 0.550 respectively.

TABLE VI. αS AND THRESHOLDS FOR THE TEXT CLASSIFIER (TH_{TEXT}) AND THE META CLASSIFIER TH_{META} ACROSS THE TEN FOLDS.

Folds	α	TH_{TEXT}	TH_{META}
Fold 1	0.42	0.50	0.37
Fold 2	0.70	0.30	0.39
Fold 3	0.67	0.20	0.37
Fold 4	0.68	0.30	0.40
Fold 5	0.55	0.40	0.42
Fold 6	0.74	0.30	0.42
Fold 7	0.55	0.40	0.42
Fold 8	0.76	0.30	0.46
Fold 9	1.00	0.30	0.25
Fold 10	0.00	0.30	0.35
Avg.	**0.61**	**0.33**	**0.39**

RQ3: What are the best features for discriminating whether a question would be deleted or not?

Motivation. Aside from producing a model that can predict deleted questions, we are also interested in finding discriminative features that could help in distinguishing deleted questions from non-deleted ones. Also, in this paper, we extend the features proposed by Correa and Sureka by adding four numerical textual features, thus we are also interested to investigate whether these newly introduced features are important to predict deleted questions or not.

Approach. For each fold, we compute the information gain scores [16] of all the features in the fold. Information gain is often used as a discriminativeness measure in many past software engineering studies [19], [25], [32], [33], [36], [40]. We compute the average information gain scores for the 51 features across the 10 folds, and rank them.

Results. Table V presents the top-20 most discriminative features based on information gain scores. We notice that the information gain score is low (the highest possible value would be 1), which represents that one feature alone is not sufficient to discriminate deleted questions from non-deleted ones. We notice that among the top-3 most discriminative features, two of them are related to the tags of the questions, i.e., number of tags, and the numerical textual feature $Text^{tags}$. We manually check some questions and we find that many of the deleted questions have less tags than non-deleted questions. Also, all of the four textual features we proposed appear in the top-20 most discriminative features. $Text^{tags}$, $Text^{title}$, $Text^{mixed}$, and $Text^{body}$ are listed in position 3, 12, 14, and 15, respectively. Thus, our proposed numerical textual features can further help to discriminate deleted questions from non-deleted ones.

VI. DISCUSSION

Thresholds Learned: Table VI presents the αs, and thresholds for the text classifier and the meta classifier across the 10 folds. We notice that for different folds, the α and thresholds are different, thus it is not optimal to use the αs and thresholds calculated in the previous folds. This is due to the phenomenon of concept drift, where the conditional distribution of the output (in our case, deleted or non-deleted) is changed as time passes while the distribution of input features may stay unchanged [14].

Recall vs. Precision: Our approach is meant to be a recommendation tool. The goal is to highlight bad cases to improve the quality of StackOverflow. For such setting, recall (its ability to find bad cases) is more important than precision. In terms

TABLE VII. AVERAGE MODEL BUILDING AND PREDICTION TIME OF DELPREDICTOR COMPARED WITH THE BASELINE APPROACHES (IN SECONDS).

Approaches	Model Building Time	Prediction Time
DelPredictor	32.64	0.51
Correa and Sureka's	9.46	0.14
Random Prediction	0.00	0.05
TO^{title}	0.15	0.32
TO^{body}	0.22	0.69
TO^{mixed}	0.22	0.61
TO^{tags}	0.12	0.35
Text Classifier	3.11	0.43
Meta Classifier	9.76	0.38
$Single + TH$	14.74	0.38
$Single$	0.54	0.31

of precision, our DelPredictor does not perform as well as Correa and Sureka's approach, however in terms of recall, DelPredictor shows *much better* performance compared to the baseline approaches. Neither our approach nor Correa et al.'s approach is ready for full automation yet (i.e., automatic deletion of questions without human intervention) since the precisions of these approaches are still relatively low.

On average across the 10 folds, the thresholds for the text classifier and the meta classifier are 0.33 and 0.39, respectively. If we set the thresholds to be less than 0.5, our approach is more likely to predict a question as a deleted question, which would increase recall and decrease precision. In our approach, the target of the threshold learning step is to learn a threshold which maximizes F1-score on the training data. By using the thresholds, DelPredictor sacrifices some precision (from 0.535 to 0.507 as compared with Correa and Sureka's approach) to largely increase recall (from 0.604 to 0.832).

Efficiency: Table VII presents the average model build time and prediction time of DelPredictor compared with the baseline approaches. We notice that the model building and prediction time of *DelPredictor* are reasonable, e.g., on average, we need about 32.64 seconds to build a model, and 0.51 seconds to predict the labels of questions in a test set. Note that the model building phase can be done offline (e.g., overnight). Also, a learned model can be used to predict the labels of many new questions.

Threats to Validity: Threats to internal validity relates to errors in our code and experiment bias. We have double-checked our code, still there could be errors that we did not notice. Also, in this paper, we use a longitudinal data setup to simulate the actual usage of DelPredictor. In practice, we can only use the questions posted before to build a model, and we can not use future questions to build a model. Threats to external validity relate to the generalizability of our results. We have analyzed 417,685 questions in Stack Overflow. In the future, we plan to reduce this threat further by analyzing even more questions from Stack Overflow. Threats to construct validity refer to the suitability of our evaluation measures. We use F1-score which is also used by past studies to evaluate the effectiveness of various automated software engineering techniques [22], [23], [30], [34], [37], [38], [40]. Thus, we believe there is little threat to construct validity.

VII. RELATED WORK

Studies on Stack Overflow: Correa and Sureka perform a large-scale empirical study on Stack Overflow [12]. They find that there is an increasing number of deleted questions in Stack Overflow over the last 2 years, and deleted questions are significantly low in quality than "closed" questions. To help people detect deleted questions, they extract 47 features and propose the usage of AdaBoosting to solve the problem. Our work extend their work by:

1) Considering the textual content in the questions, we leverage text mining techniques to convert the textual content to numerical textual features. By this way, we extend the features proposed by Correa and Sureka.
2) We build a composite classifier which utilizes the advantages of both the textual and meta features to achieve a better performance. The approach proposed by Correa and Sureka only builds a single classifier.

Nasehi et al. study the quality of code examples on Stack Overflow [24]. They find nine attributes of good questions, e.g., concise code, using question context, and step-by-step solutions. Asaduzzaman et al. study the unanswered questions on Stack Overflow, and they find around 7.5% of the questions are unanswered [3]. They further build a prediction model to predict the amount of time that a question will remain unanswered. Correa and Sureka analyze and predict the "closed" questions on Stack Overflow [11]. Different from deleted questions, "closed" questions are questions that are duplicate, off-topic, subjective, not a real question, or too localized, and a question only can be deleted after it is marked as "closed". Ponzanelli et al. propose an automated approach to predict the quality of questions [27], [28]. They categorize a question into one of four classes, i.e., very good, good, bad, and very bad, according to whether the question gets an accepted answer and its score. Our work is related to but different from the above studies: we predict deleted questions on Stack Overflow. Deleted questions are questions with extremely low quality [12].

Barua et al. analyze the topics and trends on Stack Overflow by leveraging latent Dirichlet allocation (LDA) [7]. They find mobile development attract more attention than web development. Xia et al. propose a composite approach which combine multiple components to recommend the tags to the questions [39]. Wang et al. extend their work using two separate models: a Bayesian inference model and a Frequentist inference model [35]. Ponzanelli et al. propose an approach to automatically retrieve related discussions from Stack Overflow given context in the IDE [26]. Bajaj et al. perform an empirical study on the questions posted by web developers, and they find the overall ratio of web development related questions is increasing, and there are more and more discussion on mobile development [6]. Kavaler et al. study the API usage of Andriod platform in Google Play, and the API usage questions in Stack Overflow [17]. Our work is orthogonal to the above studies: we focus on the deleted question prediction, which is a different problem from the above studies.

Text Classification in Software Engineering: Xia et al. propose a fuzzy set based feature selection approach which selects important terms from the natural language description of bug reports to categorize bugs based on their fault triggering conditions [40]. Menzies and Marcus propose SEVERIS which apply a text mining and machine learning technique to predict the severity of bug reports [23]. Menzies and Marcus evaluate their proposed approach on bug reports from NASA and predict fine-grained bug severity levels. Lamkanfi et al. extend

Menzies and Marcus work by proposing another text mining approach for bug severity prediction that can predict coarse-grained bug severity levels with more accuracy [18]. They evaluate their proposed approach on many bug reports from multiple open source projects.

VIII. Conclusion and Future Work

In this paper, we propose a hybrid approach named *DelPredictor* which combines both text processing and classification techniques for deleted question prediction. *DelPredictor* has two stages: a text mining stage, and a composition stage. In the text mining stage, we apply text processing and classification techniques to convert the textual contents of questions into numerical textual features. In the composition stage, we extract meta features, and construct two classifiers, i.e., text classifier and meta classifier, that are trained using numerical textual features and meta features respectively. We perform experiments on a large dataset which contains a total of 417,685 questions from Stack Overflow. The experiment results show that DelPredictor can achieve an F1-score to 0.597, which improves the F1-scores of the approach proposed by Correa and Sureka, random prediction, and an approach that only analyzes textual contents by 9.34%, 29.50%, and 28.11%, respectively. In the future, we plan to evaluate DelPredictor with more questions from Stack Overflow, and develop a better technique that further improves the prediction performance.

Acknowledgment. This research was supported by the NSFC Program (No.61572426), and National Key Technology R&D Program of the Ministry of Science and Technology of China under grant 2015BAH17F01.

References

[1] How to ask a question on stack overflow. *http://stackoverflow.com/help/how-to-ask*.

[2] Stack overflow. *http://stackoverflow.com/*.

[3] M. Asaduzzaman, A. S. Mashiyat, C. K. Roy, and K. A. Schneider. Answering questions about unanswered questions of stack overflow. In *MSR*, 2013.

[4] J. Atwood. Stack overflow creative commons data dump. *http://blog.stackoverflow.com/2009/06/stack-overflow-creative-commons-data-dump/*, 2009.

[5] R. Baeza-Yates, B. Ribeiro-Neto, et al. *Modern information retrieval*. 1999.

[6] K. Bajaj, K. Pattabiraman, and A. Mesbah. Mining questions asked by web developers. In *MSR*, 2014.

[7] A. Barua, S. W. Thomas, and A. E. Hassan. What are developers talking about? an analysis of topics and trends in stack overflow. *EMSE*, 2014.

[8] A. Begel, R. DeLine, and T. Zimmermann. Social media for software engineering. In *Proceedings of the FSE/SDP workshop on Future of software engineering research*, 2010.

[9] L. Breiman. Random forests. *Machine learning*, 2001.

[10] R. Caruana and A. Niculescu-Mizil. An empirical comparison of supervised learning algorithms. In *ICML*, 2006.

[11] D. Correa and A. Sureka. Fit or unfit: analysis and prediction of'closed questions' on stack overflow. In *Proceedings of the first ACM conference on Online social networks*, 2013.

[12] D. Correa and A. Sureka. Chaff from the wheat: characterization and modeling of deleted questions on stack overflow. In *WWW*, 2014.

[13] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 1997.

[14] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia. A survey on concept drift adaptation. *ACM Computing Surveys (CSUR)*, 2014.

[15] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 2009.

[16] J. Han, M. Kamber, and J. Pei. *Data mining, southeast asia edition: Concepts and techniques*. 2006.

[17] D. Kavaler, D. Posnett, C. Gibler, H. Chen, P. Devanbu, and V. Filkov. Using and asking: Apis used in the android market and asked about in stackoverflow. In *Social Informatics*. 2013.

[18] A. Lamkanfi, S. Demeyer, E. Giger, and B. Goethals. Predicting the severity of a reported bug. In *MSR*, 2010.

[19] L. Lucia, D. Lo, L. Jiang, A. Budi, et al. Active refinement of clone anomaly reports. In *ICSE*, 2012.

[20] L. Mamykina, B. Manoim, M. Mittal, G. Hripcsak, and B. Hartmann. Design lessons from the fastest q&a site in the west. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, 2011.

[21] A. McCallum, K. Nigam, et al. A comparison of event models for naive bayes text classification. In *AAAI-98 workshop on learning for text categorization*, 1998.

[22] S. McIntosh, B. Adams, M. Nagappan, and A. E. Hassan. Mining co-change information to understand when build changes are necessary. In *ICSME*, 2014.

[23] T. Menzies and A. Marcus. Automated severity assessment of software defect reports. In *ICSM*, 2008.

[24] S. M. Nasehi, J. Sillito, F. Maurer, and C. Burns. What makes a good code example?: A study of programming q&a in stackoverflow. In *ICSM*, 2012.

[25] M. H. Osman, M. R. Chaudron, and P. Van Der Putten. An analysis of machine learning algorithms for condensing reverse engineered class diagrams. In *ICSM*, 2013.

[26] L. Ponzanelli, G. Bavota, M. Di Penta, R. Oliveto, and M. Lanza. Mining stackoverflow to turn the ide into a self-confident programming prompter. In *MSR*, 2014.

[27] L. Ponzanelli, A. Mocci, A. Bacchelli, and M. Lanza. Understanding and classifying the quality of technical forum questions. In *QSIC*, 2014.

[28] L. Ponzanelli, A. Mocci, A. Bacchelli, M. Lanza, and D. Fullerton. Improving low quality stack overflow post detection. In *ICSME*, 2014.

[29] J. D. Rennie, L. Shih, J. Teevan, D. R. Karger, et al. Tackling the poor assumptions of naive bayes text classifiers. In *ICML*, 2003.

[30] E. Shihab, A. Ihara, Y. Kamei, W. M. Ibrahim, M. Ohira, B. Adams, A. E. Hassan, and K.-i. Matsumoto. Studying re-opened bugs in open source software. *EMSE*, 2013.

[31] M.-A. Storey, C. Treude, A. van Deursen, and L.-T. Cheng. The impact of social media on software engineering practices and tools. In *Proceedings of the FSE/SDP workshop on Future of software engineering research*, 2010.

[32] D. Surian, Y. Tian, D. Lo, H. Cheng, and E.-P. Lim. Predicting project outcome leveraging socio-technical network patterns. In *CSMR*, 2013.

[33] F. Thung, D. Lo, M. H. Osman, and M. R. Chaudron. Condensing class diagrams by analyzing design and network metrics using optimistic classification. In *ICPC*, 2014.

[34] H. Valdivia Garcia and E. Shihab. Characterizing and predicting blocking bugs in open source projects. In *MSR*, 2014.

[35] S. Wang, D. Lo, B. Vasilescu, and A. Serebrenik. Entagrec: An enhanced tag recommendation system for software information sites. In *ICSME*, 2014.

[36] X. Xia, D. Lo, W. Qiu, X. Wang, and B. Zhou. Automated configuration bug report prediction using text mining. In *COMPSAC*, 2014.

[37] X. Xia, D. Lo, E. Shihab, X. Wang, and X. Yang. Elblocker: Predicting blocking bugs with ensemble imbalance learning. *Information and Software Technology*, 2015.

[38] X. Xia, D. Lo, E. Shihab, X. Wang, and B. Zhou. Automatic, high accuracy prediction of reopened bugs. *Automated Software Engineering*, 2015.

[39] X. Xia, D. Lo, X. Wang, and B. Zhou. Tag recommendation in software information sites. In *MSR*, 2013.

[40] X. Xia, D. Lo, X. Wang, and B. Zhou. Automatic defect categorization based on fault triggering conditions. In *ICECCS*, 2014.