NIRMAL: Automatic Identification of Software Relevant Tweets Leveraging Language Model

Abhishek Sharma, Yuan Tian and David Lo
School of Information Systems
Singapore Management University
Email: {abhisheksh.2014,yuan.tian.2012,davidlo}@smu.edu.sg

Abstract—**Twitter is one of the most widely used social media platforms today. It enables users to share and view short 140-character messages called "tweets". About 284 million active users generate close to 500 million tweets per day. Such rapid generation of user generated content in large magnitudes results in the problem of information overload. Users who are interested in information related to a particular domain have limited means to filter out irrelevant tweets and tend to get lost in the huge amount of data they encounter. A recent study by Singer et al. found that software developers use Twitter to stay aware of industry trends, to learn from others, and to network with other developers. However, Singer et al. also reported that developers often find Twitter streams to contain too much noise which is a barrier to the adoption of Twitter. In this paper, to help developers cope with noise, we propose a novel approach named NIRMAL, which automatically identifies software relevant tweets from a collection or stream of tweets. Our approach is based on language modeling which learns a statistical model based on a training corpus (i.e., set of documents). We make use of a subset of posts from StackOverflow, a programming question and answer site, as a training corpus to learn a language model. A corpus of tweets was then used to test the effectiveness of the trained language model. The tweets were sorted based on the rank the model assigned to each of the individual tweets. The top 200 tweets were then manually analyzed to verify whether they are software related or not, and then an accuracy score was calculated. The results show that decent accuracy scores can be achieved by various variants of NIRMAL, which indicates that NIRMAL can effectively identify software related tweets from a huge corpus of tweets.**

I. Introduction

Twitter, as one of the largest and popular on-line social network sites, provides a platform to let people share news, disseminate opinions, and connect with one another. It is growing fast in the recent years and is reported to have more than 600 million registered users. Along with the rapid increase in the number of users, there is also a dramatic increase in the number of microblogs (i.e., tweets) posted every day. Many Twitter users who *follow* a large number of other users[1] receive thousands of tweets daily. This causes an information overload problem which makes it hard for users to find relevant and interesting tweets among the mass of tweets that they receive.

A large number of software developers also use Twitter quite frequently, even for their professional activities, e.g., to share and obtain latest technical news, to support project and community management, etc. Unfortunately, as is the case with other normal Twitter users, they find hard to extract useful information from tweets, especially those that can help them in their professional activities. Singer et al. surveyed 271 and interviewed 27 active developers and found that although developers are using Twitter for their professional activities and development, they often find it a challenge to deal with the many irrelevant tweets (i.e., noise) in their Twitter streams [16]. Bougie et al. found that many of the tweets that are generated even by software developers are related to "daily chatter" [5]. Indeed, it is common for people to spread personal yet inconsequential information in Twitter, e.g., "Yay! Today is Friday", "It's cloudy today", etc. To make Twitter a better tool for software engineers, there is a need for a technique that can help developers identify software related tweets from the mass of other tweets. This automated approach should be able to prioritize or rank tweets for software developers' professional use (e.g., getting latest technical news) so that more relevant tweets (i.e., software related tweets) can be ranked higher than irrelevant tweets (e.g., daily chatter). Additional benefits of identifying software related tweets are elaborated in Section II-A.

Prior studies on Twitter have proposed two basic approaches to identify software related tweets. One is a support vector machine (SVM) based approach proposed by Prasetyo et al. that requires a training set of tweets labeled as software related and non software related [15]. Unfortunately, building a representative set of tweets for training an SVM classifier requires much manual effort and to the best of our knowledge no such representative training data is available till date. Another approach is a keyword based approach proposed by Achananuparp et al. and Tian et al. that takes as input a list of software related keywords and identifies a tweet as software related if it contains at least one of the keywords [1], [20]. Both Achananuparp et al. [1] and Tian et al. [20] make use of a list containing 100 software related words. However, this list is not comprehensive and many software related tweets do not contain any of the 100 words. Furthermore, software related contents on Twitter might change over time. Unfortunately, it is hard for both the SVM and keyword based approaches, which rely on a static set of training data or a list of keywords, to keep with this change without much effort (e.g., continuous labeling effort). Additional limitations of the existing approaches are elaborated in Section II-B.

To deal with the limitations of existing approaches, in this work we propose NIRMAL, which is a language model based approach to identify software related tweets. A language model can be regarded as a statistical tool to capture regulations in a text corpus. It is widely used in natural language processing

[1] In Twitter, a user will receive tweets generated by users that they *follow*.

(NLP) area to help machine translation, speech recognition, etc. [10], [12], [13], [30]. Recently, it was also adopted by researchers in software engineering to capture the naturalness of source code and support code suggestion and completion tasks [2], [6], [14], [23]. With a language model, we can take a corpus (i.e., a set of documents), model its regularities and use the resultant model to predict if another document is related to the documents in the corpus. To build a language model, our approach namely NIRMAL takes a large number of contents from StackOverflow, the largest site for developers to post questions and get answers. By doing this, we can capture the regularities among documents that are software related. Given a new tweet, we calculate the probability of the tweet to be software related, using the model learned from StackOverflow content. The probability calculated corresponds to the computation of the similarity between the given tweet and the software related contents on StackOverflow. If a tweet receives a higher probability using the model, it means that it has a higher chance of being a software related tweet. To improve the performance further, we have also extended a standard language model by considering the repetitiveness of contents in a tweet that can often differentiate between informative tweets and meaningless tweets.

Note that compared to the previous two approaches (i.e., SVM based [15] and keyword based [1], [20]), our approach is more useful as it does not require a representative set of manually labeled tweets which takes much manual effort to create, and it does not suffer the same limitations as the keyword based approach. To capture new trends and developments in software development, the language model can also be incrementally updated with new contents from StackOverflow easily ,which requires no/little manual effort (i.e., no manual labeling effort is needed).

To evaluate the effectiveness of our approach, we have trained NIRMAL on a large set of contents from the StackOverflow data dump. We then used NIRMAL to rank 6,294,015 tweets posted by 90,883 micro bloggers that we had collected in April 2013. We manually evaluated the top-ranked tweets and determined whether they are software related or not. The results were evaluated using a measure accuracy@K, which is defined as the proportion of tweets in the top-K positions that are software related. The metric accuracy@K has also been used to evaluate past studies such as [8], [27], [29]. We found that NIRMAL can achieve an accuracy@10, accuracy@50, accuracy@100, accuracy@150, and accuracy@200 of 0.900, 0.820, 0.720, 0.707 and 0.695, respectively. On the other hand, a random model can only achieve an accuracy@10, accuracy@50, accuracy@100, accuracy@150, and accuracy@200 of 0.400, 0.280, 0.280, 0.220 and 0.240, respectively. Thus, NIRMAL can improve the random model by 125%, 192.86%, 157.14%, 221.21% and 189.58%, in terms of accuracy@10, accuracy@50, accuracy@100, accuracy@150, and accuracy@200 respectively. We have also integrated our approach with the keyword based approach by Achananuparp et al. and Tian et al. and found that we can improve the accuracy@10, accuracy@50, accuracy@100, accuracy@150, and accuracy@200 of the keyword based approach by 11.11%, 31.43%, 28.38%, 28.32% and 29.14%, respectively.

The contributions of this paper are as follows:

1) We propose a new approach, named NIRMAL, that can automatically identify software related tweets. Our approach makes use of a language model to capture the regularities of software related documents by leveraging the mass of data available in StackOverflow. Our approach also measures the repetitiveness of contents to differentiate between meaningful tweets and meaningless ones. Different from the existing approaches, NIRMAL does not require a representative training set of labeled tweets or a long list of representative keywords.
2) We have used NIRMAL to rank 6,290,415 tweets from 90,883 microbloggers that were collected in April 2013. The experiment results show that NIRMAL can achieve a high accuracy@K scores (i.e., up to 0.900) and also improve the keyword based approach by up to 31%. We have also investigated the impact of different settings to determine the effectiveness of NIRMAL.

The structure of the remainder of this paper is as follows. In Section II, we elaborate the motivation of our work further. In Section III, we present the background information about Twitter and language model. In Section IV, we describe our proposed language-based approach that can automatically identify software related tweets. In Section V, we present our experiment settings and the results of our experiment. Threats to validity have been discussed in Section VI. Related work is presented in Section VII. We finally conclude and mention future work in Section VIII.

II. Motivation

In this section, we first describe the benefits of identifying software related microblogs in more detail. We then elaborate limitations of the two basic approaches that have been used to extract software related tweets and how these limitations are addressed by NIRMAL.

A. *Why identify software related tweets?*

As microblogging services have become very popular in recent years, more and more developers are using microblogs to share news and connect with one another. Software engineering researchers also noticed this trend among developers, and have started to analyze how do microbloggging sites, e.g., Twitter, help developers in their professional activities. Several studies have analyzed the contents of microblogs that developers post on Twitter [5], investigated behaviors of software microbloggers on Twitter [20], [26], and surveyed developers on how they use Twitter [16].

Researchers who conducted the above studies found that developers indeed use Twitter a lot to support their professional activities by sharing and discovering various information from microblogs, e.g., new features of a library, new methodologies to develop a software system, opinions about a new technology or tools, etc. They also find that developers use Twitter to post contents to support project management and coordinate activities inside a community. However, software microbloggers also post a lot of microblogs that are not relevant to software development, e.g., microblogs about non-technical news, personal events, jokes, etc. Since software microbloggers are creating some amount of software related knowledge together with a larger amount of non software related contents, it becomes a challenge to discover interesting software related information from microblogs that a developer receives. In fact, this is reported as one of the major challenges faced by software microbloggers who are using Twitter and is one of the barriers to the adoption of Twitter [16]. Therefore, an automated approach that can identify interesting microblogs, e.g., software related tweets, is needed.

Besides the above-mentioned practical need, automatic identification of software related tweets, can also open up a new avenue of research: it can be used to extract a new type of software repository that can be mined to support various software development and evolution tasks. Some potential tools that can be built from the identified software related tweets include:

1) Tools that discover and visualize trends of software related contents on Twitter.
2) Tools that recommend contents on Twitter that are specific to a developer's specific needs and interests.
3) Tools that mine opinions about APIs, IDEs, programming languages, technical solutions, etc., from contents on Twitter.

B. Why a new language model based approach?

In the literature, researchers have proposed two basic approaches to identify software related microblogs, i.e., Support Vector Machine (SVM) based approach and keyword based approach. We elaborate the details of these two existing approaches as well as their limitations below:

SVM based approach. Prasetyo et al. proposed to use SVM to predict if a tweet is software related or not [15]. They manually labeled 300 tweets as either software related or not and used a part of the labeled tweets as a training data to learn a classifier using SVM, and applied the classifier on another set of tweets to predict whether they are software related or not. However, their approach only considered 300 tweets from the millions of tweets. To generalize the SVM based approach, researchers need to label a large and representative sample of tweets on Twitter, which takes a lot of time. The 300 tweets are selected by checking the presence of nine software related hashtags, and therefore they have a nearly balanced data set: 47% of the tweets are software related while the other 53% are non software related. In reality, the majority of tweets do not have hashtags and there are much more non software related tweets than software related tweets. The extremely unbalanced data is likely to impact the effectiveness of the SVM classifier. In addition, contents on Twitter are evolving as new technologies and tools are introduced to the market; this means that the model might need to be updated based on new labeled tweets. Unfortunately, this would require a continuous effort to label new tweets as either software related or not which would be costly.

Keyword based approach. Achananuparp et al. and Tian et al. used a list of keywords to identify software related tweets [1], [20]. Different from the SVM based approach, the keyword based approach does not require labeled tweets. It simply takes a set of software related words as input and identifies a given tweet as software related if it contains any of the words in the provided set of words. Unfortunately, there are a number of limitations with this approach. First, it is hard to construct a comprehensive list of software related words that could identify whether a tweet is software related or not. For instance, tweet *How To: Use the Entity Framework Designer http://t.co/SteQkWAKfN* is talking about a new resource that a developer wants to share with other developers, however it does not contain any of the keywords considered by Achananuparp et al. and Tian et al. Second, some words can have multiple meanings and not all of the meanings will be software related, e.g., Java, eclipse, etc. The problem is aggravated with the fact that tweets can be written in multiple languages. In English, a word might correspond solely to a software related concept; however, it can correspond to a completely unrelated concept in another language (e.g., Ada is a programming language and the same word means "there is (are)" in Indonesian). Third, similar to the SVM based approach, the keyword based approach cannot automatically update itself when new technologies or tools are introduced. Someone needs to manually update the list of keywords to make the approach adapts to new technological updates.

To address the challenges faced by the existing two approaches, we propose a new approach namely NIRMAL to identify software related tweets leveraging language model learned from StackOverflow. The benefits of NIRMAL include: 1) it does not require labeled software related and non software related tweets, 2) it does not require manually defined keyword list, 3) it takes the context of a word into consideration. We describe our approach in detail in Section IV.

III. Background

In this section, we first describe the two platforms that we consider in this study, namely Twitter and StackOverflow. We then provide a brief introduction of language model.

A. Twitter

Twitter is the most popular and largest microblogging site worldwide. It already has more than 600 million registered users generating over 500 million tweets daily [2]. The number of active Twitter users is growing rapidly, it increased from around 167 millions in the 3rd quarter of 2012 to 284 millions in the 3rd quarter of 2014. [3]

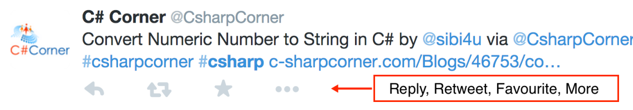
Twitter allows a user to post text messages, referred to as "tweets", with a maximum length of 140 characters. To address the limitation on the length of tweets, many Twitter users include url links in their tweets pointing to webpages such as blogs, news, etc. that contains more information. Twitter users can also include hashtags, which typically start with a "#" symbol. If a user clicks on a hashtag, Twitter will show other tweets with the same hashtag. In Twitter, one can *follow* another user; a user (follower) who follows another user (followee) will subscribe to all tweets that are posted by the followee. Besides composing new tweets, Twitter allows users to perform other activities. This includes *retweeting* an existing tweet that are posted by other users. A user that retweets a tweet will broadcast the tweet to all of his/her followers.

[2]https://about.twitter.com/company

[3]http://www.statista.com/statistics/282087/number-of-monthly-active-twitter-users/

Retweets typically start with the keyword “RT”. Users can also reply to an existing tweets or tweeting directly to a user. A reply or direct tweet contains “@Username” to identify the user the tweet is intended for. Twitter also supports users to favorite a tweet to show their interest in the content of a tweet.

Figure 1 shows a sample tweet posted by “C# Corner”. The sample tweet contains two hashtags, i.e., csharp and csharpcorner. This tweet specifies another Twitter user using the “@” symbol. It also contains a url link that points to a blog that talks about how a number can be converted to a string in C#.



C# Corner @CsharpCorner
Convert Numeric Number to String in C# by @sibi4u via @CsharpCorner
#csharpcorner #**csharp** c-sharpcorner.com/Blogs/46753/co…

Fig. 1: A sample microblog (i.e., tweet) on Twitter.

B. *StackOverflow*

StackOverflow, created in 2008, is the most popular question answering sites specially designed for developers. StackOverflow provides a platform for developers to help one another by asking and answering software related questions. It has become a large knowledge source with more than 2 million registered users contributing over 7 millions questions.[4] The large amount of software related question-and-answer threads in StackOverflow is a good source of information to mine and study. Past research works have used StackOverflow data to: discover development topics and trends [3], build software-specific word similarity database [21], automatically generate code comments [28], etc.

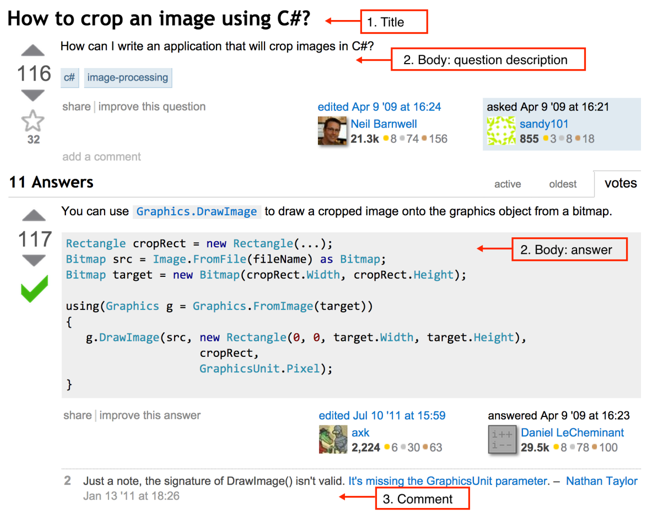
Figure 2 shows a sample question-and-answer thread extracted from StackOverflow. Each question-and-answer thread in StackOverflow contains three types of information: title, body, and comments. The title of a thread is a short summary of the question. The body of a thread contains the description of the question and one or more answers if the question has been answered. The comments of a thread could be comments to the question or comments to any of the answers. These three different types of contents have different properties: title and comments contain more natural language text, while body is usually a mixture of text and pieces of code. Furthermore, comparing title and comments, title usually contains more technical words while comments might contains some non technical sentences or phrases, such as “thank you”, etc.

C. *Language Model*

A statistical language model is a probability distribution over word sequences. It assigns a probability to any sequence of words to present the likelihood of the sequence occurring in the language it models. For example, a good language model learned from English corpus will assign higher probability scores to sentences in English than sentences in other languages.

More formally, given a word sequence $S = t_1 t_2 t_3 \dots t_n$, a language model estimates the probability of this sequence to be represented by the model as:

[4]http://en.wikipedia.org/wiki/Stack_Overflow#cite_note-soUSERS-17



How to crop an image using C#?

How can I write an application that will crop images in C#?

c# image-processing

116 — 32

share | improve this question — edited Apr 9 '09 at 16:24 Neil Barnwell 21.3k ●8 ●74 ●156 — asked Apr 9 '09 at 16:21 sandy101 855 ●3 ●8 ●18

add a comment

11 Answers — active oldest votes

117

You can use `Graphics.DrawImage` to draw a cropped image onto the graphics object from a bitmap.

```
Rectangle cropRect = new Rectangle(...);
Bitmap src = Image.FromFile(fileName) as Bitmap;
Bitmap target = new Bitmap(cropRect.Width, cropRect.Height);

using(Graphics g = Graphics.FromImage(target))
{
   g.DrawImage(src, new Rectangle(0, 0, target.Width, target.Height),
                    cropRect,
                    GraphicsUnit.Pixel);
}
```

share | improve this answer — edited Jul 10 '11 at 15:59 axk 2,224 ●6 ●30 ●63 — answered Apr 9 '09 at 16:23 Daniel LeCheminant 29.5k ●8 ●78 ●100

2 Just a note, the signature of DrawImage() isn't valid. It's missing the GraphicsUnit parameter. – Nathan Taylor Jan 13 '11 at 18:26

Fig. 2: A sample question-and-answer thread on StackOverflow.

$$P(S) = P(t_1) \prod_{i=2}^{n} P(t_i | t_1, \dots, t_{i-1}) \tag{1}$$

In the Equation 1, the probability for a sequence S is defined as a product of a series of conditional probabilities. Conditional probability $P(t_i|t_1, \dots, t_{i-1})$ represents the likelihood that word t_i follows the words that appear before it (i.e., $t_1, \dots, t_{i-1}$).

In practice, it is not practical to store all $P(t_i|t_1, \dots, t_{i-1})$ since there is a huge number of possible prefixes. Therefore, researchers have proposed methods to simplify this probability by including some assumptions. N-gram language model is one of such simplifications, which has been proven to be effective in practice. The N-gram language model assumes that the probability of a word t_i to appear after a series of words $t_1, \dots, t_{i-1}$ could be estimated by considering only the previous $N - 1$ words rather than all previous words. More formally, the following equality is assumed.

$$P(t_i|t_1, \dots, t_{i-1}) = P(t_i|t_{i-N+1}, \dots, t_{i-1}) \tag{2}$$

The probability on the right hand side of Equation 2 can be estimated from a training corpus (i.e., a set of textual documents) by computing the ratio of the number of times word t_i follows the prefix sequence $t_{i-N+1}, \dots, t_{i-1}$ and the number of times the prefix sequence $t_{i-N+1}, \dots, t_{i-1}$ appears in the training corpus. More formally, we can compute the probability as follows:

$$P(t_i|t_{i-N+1}, \dots, t_{i-1}) = \frac{count(t_{i-N+1}, \dots, t_{i-1}, t_i)}{count(t_{i-N+1}, \dots, t_{i-1})} \tag{3}$$

In the above equation, $count(S)$ corresponds to the number of times sequence S appears in the training corpus.

Consider the sample tweet shown in Figure 1 which is a sequence of words. If we use a bigram language model (N=2), the probability of "convert numeric number to string in C#" could be calculated as

$$\begin{aligned}P(convert, numeric, number, to, string, in, C\#) = \\ P(convert|\langle s\rangle)P(numeric|convert)P(number|numeric) \\ P(to|number)P(string|to)P(in|string) \\ P(C\#|in)P(\langle /s\rangle|C\#)\end{aligned}$$

In the above equation, $\langle s\rangle$ denotes the start-of-sentence marker and $\langle /s\rangle$ denotes the end-of-sentence marker.

Since the probability of each word in a sentence is often small, and the multiplication of many small numbers can cause underflow problem, rather than computing the probability of a sentence, the logarithm of this probability is often computed. The negation of this logarithm normalized by the number of words is often referred to as the *perplexity* of the sentence. In this paper, we denote the perplexity of a sentence S as $PP(S)$. It is defined more formally below:

$$PP(S) = \frac{-1}{n} log(P(t_1 t_2 t_3 \ldots t_n)) \tag{4}$$

Note that a low perplexity score corresponds to a high probability score. Thus, the lower the perplexity score of a sentence is, the more closely the language model captures the sentence.

One problem of applying N-gram model in real tasks is that it assigns a zero probability to a sentence if an N-gram in the sentence does not appear in the training corpus. To deal with this problem, many *smoothing* techniques have been proposed in the literature. A smoothing technique assigns a small but non-zero probability to an N-gram that does not appear in the training corpus. One of the well known smoothing technique is the Katz backoff model [9]. It replaces the probability a word w considering the prior $N-1$ words, with the probability of w considering the prior $M-1$ words (where $M < N$), if the earlier probability is zero. In effect, it reduces a N-gram model to a M-gram model, where M is less than N, if an N-gram does not exist in the training corpus.

IV. Approach

Figure 3 shows the overall framework of NIRMAL. The approach includes three major phases: the model creation phase, tweet ranking phase, and evaluation phase. In the model creation phase, NIRMAL learns a language model from StackOverflow data. In the tweet ranking phase, NIRMAL first uses the learned model to compute the perplexity score of each tweet. The lower the perplexity score, the more likely the tweet is software related. NIRMAL then ranks the tweets in ascending order of their perplexity scores and returns this ranked list. The three phases are described in more detail in the following subsections.

A. StackOverflow Data Acquisition & Preprocessing

We used the StackOverflow data dump that is provided in the following website: archive.org/download/stackexchange. In the website, there are many files corresponding to contents from various StackExchange websites (including StackOverflow). We use the following two files: Posts.7z[5] and Comments.7z[6]. Posts.7z contains the title and body (i.e., question and answers) of posts that appear in StackOverflow. Comments.7z contains comments that people give to the questions and answers in StackOverflow. These files contain contents posted in StackOverflow from July/September 2008 to September 2014. There are a total of 7,990,787 titles, 21,736,594 bodies (i.e., questions + answers), and 32,506,636 comments.

Since there are too many bodies (i.e., questions + answers) and comments, to reduce the time it takes to learn a language model, we only used 8,000,000 of them. We randomly selected 8,000,000 bodies and comments from the data dump. We also performed simple text pre-processing. We removed all punctuation marks and URLs from the sentences in the titles, bodies, and comments. We also changed all words to their lower case.

B. Language Model Building

We used SRILM [17], a popular language modelling toolkit, to create an N-gram language model. SRILM takes as input a set of documents, a parameter N, and outputs an N-gram language model that characterizes the regularities of text in the input set of documents. SRILM performs smoothing following the Katz backoff model [9]. Thus, it reduces a N-gram model to a M-gram model, where M is less than N, if an N-gram does not exist in the training corpus.

C. Twitter Data Acquisition & Preprocessing

To collect tweets, we first obtained a set of microbloggers that are more likely to generate software related contents. We started with a collection of 100 seed microbloggers who are well known-software developers[7]. Among these seed microbloggers we have `codinghorror` which is the Twitter alias of Jeff Atwood, the founder of StackOverflow. Next, we analyzed the *follow* links of these microbloggers on March 1, 2013, to identify other microbloggers that follow or are followed by at least 5 seed microbloggers. We added these other microbloggers to the set of seed microbloggers to get a set of 90,883 microbloggers. After, we had identified the target microbloggers, we downloaded tweets that are generated by these microbloggers from April 1 to April 31, 2013. We downloaded these tweets using the Twitter REST API. We have a Twitter whitelist account that allows us to make 20,000 API calls every hour. In total, we collected 6,294,015 tweets.

We performed simple pre-processing on the collected tweets. We removed punctuation marks and URLs, and also changed all words into their lowercase.

D. Ranking of Tweets

To rank tweets, we made use of two sources of information: first, we used the perplexity score that is output by the language model; second, we computed a scaling factor based on the repetitiveness of words in a tweet. We found that many tweets

[5]https://archive.org/download/stackexchange/stackoverflow.com-Posts.7z

[6]https://archive.org/download/stackexchange/stackoverflow.com-Comments.7z

[7]http://www.noop.nl/2009/02/twitter-top-100-for-softwaredevelopers.html

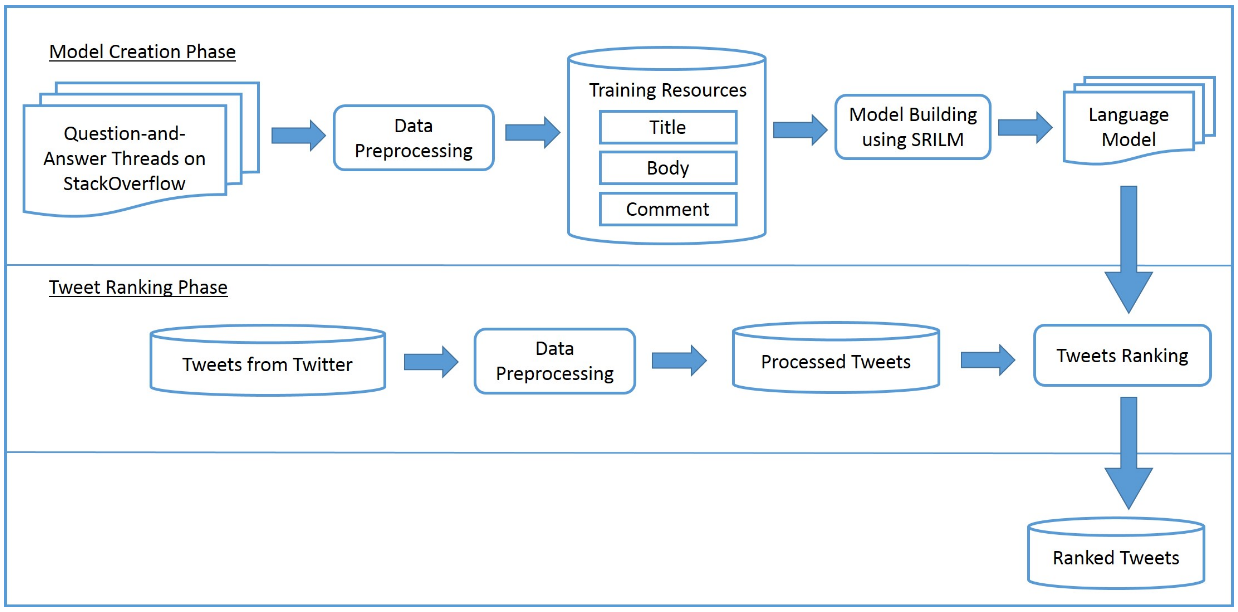


Fig. 3: Framework of the proposed approach - NIRMAL

with repetitive contents, e.g., “1 1 1 1 1 1 1”, “a a a a a a”, are rather meaningless. Most meaningful tweets do not have a high number of repetition. We computed the scaling factor of a sentence S using the following equation:

$$S_t(S) = \frac{wc_t(S)}{wc_u(S)} \tag{5}$$

In the above equation, $wc_t(S)$ is the number of words in the sentence S and $wc_u(S)$ is the number of unique words in the sentence S. Note that the lower the scaling factor the less repetitive a tweet is and the more likely it is meaningful.

Given a sentence S after we have computed its perplexity score (i.e., $PP(S)$) and its scaling factor (i.e., $S_t(S)$), we can compute its revised perplexity score, denoted by $PP^R(S)$, as follows:

$$PP^R(S) = PP(S) \times S_t(S) \tag{6}$$

The lower the ranking score of a tweet the higher is the likelihood of it to be software related and not meaningless. Note that due to smoothing using Katz backoff model, although both “1 1 1 1 1 1 1”, “a a a a a a” do not appear in the StackOverflow data, SRILM will assign a relatively low perplexity score to both sentences since “1” and “a” appears often in the StackOverflow data. Thus, we need to leverage the repetitiveness of contents in a tweet to increase the score of these meaningless tweets.

V. Experiments & Results

In this section, we describe our dataset and experimental settings, followed by our evaluation metric. We then present our four research questions and the results of our experiments.

A. Dataset and Settings

The detailed statistics of the Twitter and StackOverflow datasets that we use in this experiment are shown in Table I. The number of words in the tweets that we collected amounts to more than 77 million. The number of words in the titles, bodies (i.e., questions + answers), and comments that we collected amounts to, slightly more than 39 thousand, 725 million and 200 million, respectively.

TABLE I: Statistics of Twitter Data and StackOverflow Data.

Corpus	#Documents	#Words
Twitter	6,294,015	77,491,505
StackOverflow (Title)	7,990,787	39,786
StackOverflow (Body)	8,000,000	725,449,601
StackOverflow (Comment)	8,000,000	200,584,369

NIRMAL accepts two inputs: the dataset used to learn a language model and the parameter N of N-gram. By default, unless otherwise stated, we use the StackOverflow (Title) corpus (i.e., the titles of the StackOverflow posts) to learn a language model, and set the value of N to 4. We run the experiment using the following machines: Preprocessing and tweet ranking steps are run on Intel Core i5-4570 3.2 GHz CPU, 8 GB RAM desktop running Windows 7 64 bit. All SRILM related steps are performed on a 7 core Intel(R) Xeon(R) CPU E5-2667 0 @ 2.90GHz, 64 GB RAM server running CentOS release 6.5.

B. Evaluation Metrics

We use NIRMAL to sort the 6.2 million tweets and we manually inspect the top-K tweets that are returned by NIRMAL. We evaluate the effectiveness of our approach using accuracy@K. accuracy@K is defined as the proportion of tweets in the top-K positions that are software related. accuracy@K has also been used to evaluate other past studies [8], [27], [29].

C. Research Questions

Our experiments aim to answer following four research questions that assess the strengths and limitations of NIRMAL and several baseline approaches.

RQ1: How effective is our approach in identifying software related tweets?

In this question, we want to evaluate how effective is NIRMAL in ranking tweets such that the software related ones are ranked higher than the non software related ones. To answer this research question we simply run NIRMAL with the default setting on the 6.2 million tweets and manually evaluate the top-K tweets that are returned by NIRMAL. We report the accuracy@K scores that are achieved by NIRMAL. We compare the performance of NIRMAL with the performance of a random model that randomly labels K tweets as software related.

RQ2: What are the effects of varying NIRMAL inputs on its effectiveness?

NIRMAL accepts two kinds of inputs: the value N for the N-gram model, and the dataset used to train the N-gram model. In this research question, we want to investigate the impact of using different values of N and different datasets on the overall effectiveness (i.e., accuracy@K scores) of NIRMAL. We investigate four different N values, i.e., 1,2,3, and 4, and five different datasets: StackOverflow (Title), StackOverflow (Body), StackOverflow (Comment), StackOverflow (Title) $\bigcup$ StackOverflow (Body), StackOverflow (Title) $\bigcup$ StackOverflow (Body) $\bigcup$ StackOverflow (Comment).

RQ3: How efficient is our approach?

Many new tweets are continuously generated every second. For our approach to work in practice, it needs to be able to process new tweets efficiently. In this research question, we investigated the time it takes for NIRMAL to learn a language model and the time it takes to compute the revised perplexity score of a tweet. Since a trained language model can be used to label many tweets before it needs to be retrained, the time NIRMAL takes to learn a language model can be long (e.g., a few hours) but cannot be excessively long (e.g., a few months). On the other hand, the time NIRMAL takes to compute the revised perplexity score of a tweet needs to be very short (i.e., less than a second).

RQ4: Could our approach improve the effectiveness of the keyword based approach?

Achananuparp et al. and Tian et al. have used a set of keywords to detect if a tweet is software related or not. However, many tweets that contain one or more of the keywords are not software related. In this research question, we investigated whether we can use NIRMAL to effectively sort tweets that have been filtered such that the software related ones appear in the top of the list. To answer this research question, we first filtered the 6.2 million tweets using the 100 keywords that Achananuparp et al. and Tian et al. used. In total, among the 6.2 million tweets, we have 227,225 tweets that contain at least one of the keywords.We then selected a random sample of 200 tweets and calculated the accuracy@K scores for *keyword only* approach. We then applied NIRMAL to sort all the 227,225 keyword containing tweets and manually evaluated the top-K tweets to compute the accuracy@K score to check if NIRMAL is able to improve the accuracy of keyword based approach.

D. Research Results

In this section, we present our experiment results that answer each of the research questions raised in the previous section.

1) RQ1: Effectiveness of Our Approach: The results of our experiment are shown in Table II. From the results we can note that the accuracy@K of NIRMAL ranges from 0.695 to 0.900 using the default setting. When we investigated the top-10 tweets, we found that 90% of them are software related. When we investigated the top-200 tweets, we found that 69.5% of them are software related when NIRMAL is used.On the other hand for a random model only 24% of the top-200 tweets were software related. This shows that NIRMAL is accurate, and also that the tweets ranked higher in the list are more likely to be software related than those ranked lower in the list.

TABLE II: acc@K (i.e., accuracy@K) of NIRMAL for Various K

Approach	acc@10	acc@50	acc@100	acc@150	acc@200
NIRMAL	0.900	0.820	0.720	0.707	0.695
Random	0.400	0.280	0.280	0.220	0.240

2) RQ2: Effectiveness of Various Parameter Settings and Learning Resources: Varying the parameter N of the N-gram model. The results of our experiment are shown in Table III. From the results we note that if we increase N for N-gram language model the accuracy@K increases for all values of K, e.g., The accuracy@200 increases from 0.120 to 0.695 as we move from 1-gram to 4-gram model.

TABLE III: Effect of Varying N on the Performance of NIRMAL

N	acc@10	acc@50	acc@100	acc@150	acc@200
1	0.000	0.140	0.140	0.127	0.120
2	0.500	0.460	0.460	0.473	0.485
3	0.600	0.640	0.680	0.660	0.630
4	0.900	0.820	0.720	0.707	0.695

Varying the training corpus. The results of our experiment are shown in Table IV. From the results we note that for any N-gram language model the highest values of accuracy@K were achieved when the training corpus containing only Titles was used. This can be explained as the Titles will generally contain less noise, i.e., natural language text not related to software. However the Body and Comments contain a lot of normal language text, as well as code samples and fragments, which should explain the relatively lower accuracy scores attained by language models created using their corpus.

3) RQ3: Efficiency of Our Approach: The results of our experiment is shown in Table V. We show the time NIRMAL takes to create a language model from the StackOverflow title data (i.e., Model Creation Time), and the average time

TABLE IV: Effect of Using Different Training Corpus on the Performance of NIRMAL. T = Title, B = Body, C = Comment, TB = Title + Body, TBC = Title + Body + Comment.

Corpus	acc@10	acc@50	acc@100	acc@150	acc@200
T	0.900	0.820	0.720	0.707	0.695
B	0.300	0.560	0.530	0.500	0.475
C	0.500	0.380	0.280	0.227	0.200
TB	0.400	0.640	0.560	0.540	0.500
TBC	0.400	0.600	0.540	0.447	0.435

NIRMAL takes to compute the revised perplexity score of a tweet (i.e., Model Compu. Time), for various values of the N parameter. All the times are shown in seconds. We can observe that as N increases the time to create a model also increases. This is pretty evident because the model will need to consider a higher number of N-grams (word pairs) when N increases. However change in N seems to have a negligible effect on the time required to calculate revised perplexity score for a new tweet. Please note that perplexity score calculation time has been averaged over score calculation time for all 6,294,015 tweets.

TABLE V: Efficiency of NIRMAL

N	Model Creation Time (in Sec.)	Score Compu. Time (in Sec.)
1-gram	14	0.000268827
2-gram	46	0.000261518
3-gram	101	0.000261359
4-gram	175	0.000278042

4) RQ4: Integration with Keyword Based Method: The experiment results are shown in Table VI. We can clearly observe that applying the NIRMAL to the keyword approach improves the accuracy@K for all values of K. Our results show that NIRMAL can be used to improve the accuracy score up to 31%. The lowest observed increase of about 11.11% for accuracy@10 value. But this should be seen as a positive result as it was the maximum increase possible at K=10. The new value achieved i.e., 1 (obtained after applying NIRMAL) is the highest value possible value for the parameter accuracy@K. Thus, we can deduce that applying NIRMAL to a keyword approach seems to result in an improved performance.

TABLE VI: Keyword VS. NIRMAL + Keyword

Approach	acc@10	acc@50	acc@100	acc@150	acc@200
Key.	0.900	0.700	0.740	0.753	0.755
NIRMAL + Key.	1.000	0.920	0.950	0.967	0.975

VI. Threats to Validity

In this section, we discuss threats to three types of validity, i.e., internal, external, and construct validity.

Threats to internal validity. Threats to internal validity relate to errors in our experiments and our labelling. Most of our experimental process is based on SRILM, a commonly used language model learning and application tool. We believe the code of SRILM is stable and reliable. To label the tweets as software related or not, we asked one PhD student with more than 5 years of experience in software industry and more than 10 years of experience in programming to manually label the tweets. We believe the PhD student has enough expertise to decide if a tweet is software related or not. When labeling the tweets, the PhD student not only reads the tweets but also opens the URLs contained in the tweets (if needed). The labeling process might be subjective, however, since one person labels all tweets the judging criteria used remains consistent.

Threats to external validity. Threats to external validity relates to the generalizability of our approach and evaluation. In this work, to reduce threats brought by using a small training corpus, we have downloaded and used millions of titles, questions, answers, and comments from the official StackOverflow dump which contains contents posted in StackOverflow from 2008 to 2014. We have used NIRMAL to rank more than 6.2 million tweets that are generated by more than 90 thousand microbloggers over a two month period. We have manually labeled the top-200 tweets generated by NIRMAL. In the future, to reduce the threats to external validity, we plan to use NIRMAL to rank a larger number of tweets generated by more microbloggers. We also plan to manually label a larger number of tweets.

Threats to construct validity. Threats to construct validity relates to the suitability of our evaluation metric. In this work, we use accuracy@K to measure the effectiveness of our approach. This metric is intuitive and it has been used in many previous studies, e.g., [8], [27], [29]. Thus, we believe there is little threat to construct validity.

VII. Related Work

In this section, we first present existing software engineering studies that analyze Twitter data. We then present studies that implement or customize language models for analyzing various software engineering data. Finally, we describe some software engineering studies that employ other text analysis techniques to analyze various software artifacts.

A. Twitter and Software Engineering

Singer et al. surveyed 271 and interviewed 27 active developers on Github [16]. They found that many developers are using Twitter to “keep up with the fast-paced development landscape”. Specifically, developers use Twitter to get awareness of people and trends, extend their technical knowledge, and build connections with other developers. They also found that information overload, i.e., few useful information hidden in thousands of useless tweets, is one of the biggest challenges faced by developers in using Twitter. Their finding shows the need to help developers pull out software related tweets from the massive amount of non software related tweets, which motivates our study.

A number of researchers have analyzed microblogs posted in Twitter (aka tweets) or built tools to support developers to better use Twitter for their day-to-day work. Bougie et al. analyzed 11,679 tweets posted by 68 developers from three open source projects [5]. They observed that software engineers leverage Twitter to communicate and share information. They also conducted a qualitative study on 600 tweets and group them into four categories: software related, gadgets and technological topics, non-technical topics, and daily chatter. Wang et al. analyzed 568 tweets posted by developers from the Drupal open source project [26]. They found that Drupal developers use Twitter to coordinate efforts, share knowledge, encourage potential contributors to join, etc. Tian et al. analyzed behaviours of software microbloggers on a large dataset that contains more than 13 million tweets posted by 42 thousand microbloggers [20]. They used 100 software related words to identify software related tweets and found that software related tweets often contain more URLs and hashtags, but less mentions than non-software related tweets. They also find that some of the microbloggers are very active on Twitter and have contributed many software related tweets. In another work, Tian et al. manually categorized 300 tweets that contain software related hashtags, e.g., "java", "csharp" into ten groups [19]. These ten groups include commercial, news, tools and code, question and answer, events, personal, opinion, tips, job, and miscellaneous. Achananuparp et al. build a tool that can visualize trends from a large number of software related tweets [1]. To collect software related tweets, they first selected a set of famous software engineers on Twitter as seed users, and then they expanded this set to include other potential software microbloggers by including those that *follow* or are *followed* by many of the seed users. They then collected tweets from this set of microbloggers and identified software related tweets by checking the presence of 100 software related words.

Our work is closest to the work of Prasetyo et al. [15]. They manually analyzed a sample of 300 tweets and labeled each of them as a software related tweet or non-software related tweet. They then used Support Vector Machine (SVM) to train a model from a set of labeled tweets and applied the model on another set of tweets to predict if each of them is software related or not. For their approach to work well on a large set of tweets, there is a need to manually label a large set of tweets. This would require much manual work. Due to the unavailability of a large set of labeled tweets, their approach has only been tested on a small set of tweets (i.e., 300 tweets). Different from Prasetyo et al.'s approach, our approach does not require any labeled tweet. We learn a model based on the publicly available StackOverflow data and use it to compute a likelihood score of a tweet being software related or not.

B. Language Models and Software Engineering

In recent years, many researchers have applied and customized techniques from Natural Language Processing (NLP) domain to analyze software data [2], [4], [6], [11], [14], [18], [23]. Among the NLP techniques, language models are getting more popular in software engineering research, and have shown their power in supporting code recommendation [2], [6], [14], [23]. We highlight a number of studies that make use of language models for code recommendation below.

Hindle et al. investigated whether a piece of code can be modelled by a language model, and how the language model can be used to support software engineering tasks [6]. They find that code is even more repetitive in nature than natural language, and therefore it can also be modelled by a language model. They further build a code completion tool for Java and show that it can improve Eclipse's code completion capability. Nguyen et al. propose a semantic language model that combines an N-gram model with a topic model to represent code [14]. They show that the resultant semantic language model named SLAMC can improve a standard N-gram model in representing code and it can be used to build an improved code completion tool which achieves a higher accuracy as compared to the tool built by Hindle et al. Tu et al. extended an N-gram model by incorporating a special characteristic of code namely its localness [23]. Due to module specialization, code in a particular module typically has different N-gram distributions as compared to code in other modules. A standard N-gram model only captures global regularities but not local regularities exhibited in a particular module. To take the localness of code into consideration, Tu et al. use a cache to store N-gram code patterns in each locality. Their experiment shows that their approach can be used to build a better code completion tool that can achieve a better accuracy than Hindle et al.'s approach and Nguyen et al.'s approach. Allamanis and Sutton build a language model from 352 million lines of Java code [2]. They show that by using this large code base, a better language model can be learned. They applied the learned language model to build a code completion tool and show that it is better than the tool proposed by Hindle et al. which is only trained on a smaller code base. In the same paper, Allamanis and Sutton also propose a new code complexity metric that can distinguish reusable classes from other classes. Different from the above studies, we use language model for a different purpose, namely to predict the likelihood of a tweet to be software related or not.

C. Text Analysis and Software Engineering

Besides language models, many other text analysis techniques, e.g., information retrieval, topic modeling, etc., have been used by researchers to analyze software and its related textual artifacts [3], [7], [22], [24], [25]. We highlight a few software engineering studies that make use of text analysis below.

Wang et al. conducted a comparative study on the effectiveness of 10 different information retrieval (IR) techniques in locating code units that implement a feature [25]. They tested the performance of the IR techniques on a large Linux code corpus, that contains 1,561 features and their linked 85,466 functions. Their experiments show that simple IR techniques, i.e., vector space model (VSM) and smoothed unigram model (SUM) perform better than more complicate IR techniques, e.g., probabilistic latent semantic indexing (pLSA), non-negative matrix factorization (NMP), etc. Tian et al. applied IR techniques on another problem, i.e., predicting the severity level of a new bug report based on previous recorded bug reports [22]. Given a new bug report, their approach returns a list of k historical bug reports that are most similar to the given bug report, and then use the severity levels of similar bug reports to predict the severity level of the new bug report. They used an extension of a popular information retrieval similarity measurement, i.e., BM25F, to compute the similarity between two bug reports.

Hindle et al. applied latent Dirichlet allocation (LDA), a statistical topic modeling technique, to commit logs to discover what topics are being worked on by developers at any given time, and to see how development trends are changing [7]. Barua et al. applied LDA to discover topics and trends present in questions and answers on StackOverflow [3]. They found that: the topics range widely from jobs to version control systems to C# syntax; questions in some topics lead to discussions in other topics; and the topics gaining the most popularity over time are web development, mobile applications, Git, and MySQL.

VIII. Conclusion and Future Work

Twitter has become a popular means to share and disseminate information. To date, there are hundreds of millions of Twitter users generating billions of microblogs (aka tweets). Software developers are also using Twitter, even for their professional activities. Singer et al. found that software developers use Twitter to get awareness of people and trends, extend their technical knowledge, and build connections with other developers [16]. Unfortunately, developers often find it a challenge to deal with the many irrelevant tweets (i.e., noises) in their Twitter streams. Many developers follow many people that generate many tweets (many of which are irrelevant) that get broadcasted to them every day.

To make Twitter a better tool for developers in their professional activities, we propose a new approach that can help developers identify software related tweets from the mass of other irrelevant tweets. Our approach, named NIRMAL, trains a language model from a corpus of software related contents on StackOverflow. The trained language model infers the regularities of software related contents and use these regularities to compute the likelihood of a tweet to be software related. To improve the performance further, NIRMAL also considers the repetitiveness of words in a tweet that can be used to differentiate between informative and meaningless tweets. In our experiment, we have used NIRMAL to rank a set of 6.2 million tweets generated by more than 90 thousands microbloggers. Most of the tweets are not software related while only a minority of them are software related. The experiment results show that NIRMAL can achieve an accuracy@200 score of up to 0.695 which is greater than the accuracy@200 score of a random model by up to 192%. Furthermore, NIRMAL can be used to improve the accuracy score of a keyword based approach by up to 31%.

As a future work, we plan to build N-grams with larger N and evaluate how they perform w.r.t parameters of accuracy and computational performance. We plan to investigate the effect on performance of current models by adding more preprocessing steps such as stemming and stop word removal. We also plan to propose an approach that can summarize the identified software related tweets to help developers better manage the large number of tweets that they receive daily.

Acknowledgement

This research is supported by the Singapore National Research Foundation under its International Research Centre @ Singapore Funding Initiative and administered by the IDM Programme Office, Media Development Authority (MDA). We would also like to thank Maksim Tkachenko for a good technical discussion on language model.

References

[1] P. Achananuparp, I. N. Lubis, Y. Tian, D. Lo, and E.-P. Lim, "Observatory of trends in software related microblogs," in *ASE*, 2012, pp. 334–337.

[2] M. Allamanis and C. Sutton, "Mining source code repositories at massive scale using language modeling," in *MSR*, 2013, pp. 207–216.

[3] A. Barua, S. W. Thomas, and A. E. Hassan, "What are developers talking about? an analysis of topics and trends in stack overflow," *EMSE*, vol. 19, no. 3, pp. 619–654, 2014.

[4] L. R. Biggers, C. Bocovich, R. Capshaw, B. P. Eddy, L. H. Etzkorn, and N. A. Kraft, "Configuring latent dirichlet allocation based feature location," *EMSE*, vol. 19, no. 3, pp. 465–500, 2014.

[5] G. Bougie, J. Starke, M.-A. Storey, and D. M. German, "Towards understanding twitter use in software engineering: preliminary findings, ongoing challenges and future questions," in *Proceedings of the 2nd international workshop on Web 2.0 for software engineering*, 2011, pp. 31–36.

[6] A. Hindle, E. T. Barr, Z. Su, M. Gabel, and P. Devanbu, "On the naturalness of software," in *ICSE*, 2012, pp. 837–847.

[7] A. Hindle, M. W. Godfrey, and R. C. Holt, "What's hot and what's not: Windowed developer topic analysis," in *ICSM*, 2009, pp. 339–348.

[8] Y. Inagaki, J. Bian, and Y. Chang, "An effective general framework for localized content optimization," in *WWW*, 2013, pp. 65–66.

[9] S. Katz, "Estimation of probabilities from sparse data for the language model component of a speech recognizer," *Acoustics, Speech and Signal Processing, IEEE Transactions on*, vol. 35, no. 3, pp. 400–401, 1987.

[10] R. Kuhn and R. De Mori, "A cache-based natural language model for speech recognition," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 12, no. 6, pp. 570–583, 1990.

[11] S. K. Lukins, N. A. Kraft, and L. H. Etzkorn, "Bug localization using latent dirichlet allocation," *Information and Software Technology*, vol. 52, no. 9, pp. 972–990, 2010.

[12] C. Manning and H. Schutze, *Foundations of Statistical Natural Language Processing*. MIT Press, 1999.

[13] J. B. Marino, R. E. Banchs, J. M. Crego, A. de Gispert, P. Lambert, J. A. Fonollosa, and M. R. Costa-Jussà, "N-gram-based machine translation," *Computational Linguistics*, vol. 32, no. 4, pp. 527–549, 2006.

[14] T. T. Nguyen, A. T. Nguyen, H. A. Nguyen, and T. N. Nguyen, "A statistical semantic language model for source code," in *FSE*, 2013, pp. 532–542.

[15] P. K. Prasetyo, D. Lo, P. Achananuparp, Y. Tian, and E.-P. Lim, "Automatic classification of software related microblogs," in *ICSM*, 2012, pp. 596–599.

[16] L. Singer, F. M. Figueira Filho, and M.-A. D. Storey, "Software engineering at the speed of light: how developers stay current using twitter." in *ICSE*, 2014, pp. 211–221.

[17] A. Stolcke *et al.*, "Srilm-an extensible language modeling toolkit." in *INTERSPEECH*, 2002.

[18] S. W. Thomas, B. Adams, A. E. Hassan, and D. Blostein, "Modeling the evolution of topics in source code histories," in *MSR*, 2011, pp. 173–182.

[19] Y. Tian, P. Achananuparp, I. N. Lubis, D. Lo, and E.-P. Lim, "What does software engineering community microblog about?" in *MSR*, 2012, pp. 247–250.

[20] Y. Tian and D. Lo, "An exploratory study on software microblogger behaviors," in *MUD*, 2014.

[21] Y. Tian, D. Lo, and J. Lawall, "Automated construction of a software-specific word similarity database," in *CSMR-WCRE*, 2014, pp. 44–53.

[22] Y. Tian, D. Lo, and C. Sun, "Information retrieval based nearest neighbor classification for fine-grained bug severity prediction," in *WCRE*, 2012, pp. 215–224.

[23] Z. Tu, Z. Su, and P. Devanbu, "On the localness of software," in *FSE*, 2014.

[24] S. Wang, D. Lo, and J. Lawall, "Compositional vector space models for improved bug localization," in *ICSME*, 2014, pp. 171–180.

[25] S. Wang, D. Lo, Z. Xing, and L. Jiang, "Concern localization using information retrieval: An empirical study on linux kernel," in *WCRE*, 2011, pp. 92–96.

[26] X. Wang, I. Kuzmickaja, K.-J. Stol, P. Abrahamsson, and B. Fitzgerald, "Microblogging in open source software development: The case of drupal and twitter," *Software, IEEE*, 2013.

[27] X. Wang, D. Lo, J. Jiang, L. Zhang, and H. Mei, "Extracting paraphrases of technical terms from noisy parallel software corpora," in *ACL-IJCNLP*. Association for Computational Linguistics, 2009, pp. 197–200.

[28] E. Wong, J. Yang, and L. Tan, "AutoComment: Mining question and answer sites for automatic comment generation," in *ASE*, 2013.

[29] X. Xia, D. Lo, X. Wang, and B. Zhou, "Accurate developer recommendation for bug resolution," in *WCRE*, 2013, pp. 72–81.

[30] C. Zhai, "Statistical language models for information retrieval," *Synthesis Lectures on Human Language Technologies*, pp. 1–141, 2008.