A Little Bird Told Me: Mining Tweets for Requirements and Software Evolution

Emitza Guzman
University of Zurich, Switzerland
guzman@ifi.uzh.ch

Mohamed Ibrahim
Technische Universität München, Germany
m.ibrahim@tum.de

Martin Glinz
University of Zurich, Switzerland
glinz@ifi.uzh.ch

***Abstract*—Twitter is one of the most popular social networks. Previous research found that users employ Twitter to communicate about software applications via short messages, commonly referred to as tweets, and that these tweets can be useful for requirements engineering and software evolution. However, due to their large number—in the range of thousands per day for popular applications—a manual analysis is unfeasible.**

In this work we present ALERTme, an approach to automatically classify, group and rank tweets about software applications. We apply machine learning techniques for automatically classifying tweets requesting improvements, topic modeling for grouping semantically related tweets and a weighted function for ranking tweets according to specific attributes, such as content category, sentiment and number of retweets. We ran our approach on 68,108 collected tweets from three software applications and compared its results against software practitioners' judgement. Our results show that ALERTme is an effective approach for filtering, summarizing and ranking tweets about software applications. ALERTme enables the exploitation of Twitter as a feedback channel for information relevant to software evolution, including end-user requirements.

***Keywords*-user feedback; Twitter; software evolution; text mining; requirements elicitation.**

I. Introduction

With over 500 million short messages—tweets—sent per day, Twitter is one of the most popular social networks. Previous work [1] found that users employ Twitter to communicate about software applications and that some of these tweets contain user feedback such as feature requests, feature shortcomings and bug reports. This is a valuable source of information when determining the requirements for future software releases in the course of software evolution.

Given the high number of daily tweets—in the range of 30,000 for popular applications such as Facebook and Snapchat [1], and the large number of tweets not containing any requirements-related information [1], a manual analysis of Twitter streams is not feasible.

Recent research has focused on the mining of user feedback from app stores (e.g., [2], [3], [4]). In this paper, we investigate to which extent the techniques used in that research can be applied to tweets about software applications. While tweets and app store reviews share certain commonalities, such as their high numbers, unstructured nature and informal language, there are also significant differences. In particular, tweets are shorter than average app reviews, and the metadata in both channels is different. Therefore, the suitability and performance of techniques previously applied to app reviews for analyzing tweets need to be assessed.

In this work we present ALERTme (A LittlE biRd Told me), an approach to (1) classify tweets into categories related to software evolution, (2) group tweets according to their content and (3) rank tweets according to their relevance. We use supervised machine learning for classifying tweets, topic modeling for grouping related tweets, and a weighted function for ranking the tweets.

Developers, requirements engineers or product owners can use the results from the three steps of ALERTme to (1) filter tweets that do not contain information relevant for their tasks, (2) further summarize tweet content, thus simplifying the task of identifying requirements or other information relevant for software evolution, and (3) obtain information about the relevance of the tweets which, for example, can be used for prioritizing requirements elicited from the tweets.

We collected 68,108 tweets about three software applications and ran ALERTme on the gathered tweets. In particular, we compared the automatic classification against a manually generated truthset of 1,350 tweets, executed two assessment tasks [5] for systematically determining the quality of the tweet groups according to software practitioners' judgement and evaluated the ranking function against the assessment of software practitioners. Our results are encouraging and show that it is worthwhile to explore tweets as a user feedback channel and potential source for requirements.

The contribution of this work is threefold. First, we present ALERTme as a technical solution for processing user feedback during software evolution. Second, we present empirical evidence that state-of-the-art techniques have a good performance when classifying and grouping tweets. Third we empirically assess that our novel technique for ranking tweets yields promising results.

II. Related Work

We focus the related work discussion in three areas: (1) user feedback and software evolution, (2) mining of user feedback for software evolution—with a special focus on app stores, and (3) Twitter in software evolution.

User Feedback and Software Evolution. Previous research [6] found that user feedback is essential for software quality and for identifying ideas for improvement. Pagano and Maalej [7] and Hoon [8] conducted exploratory studies and

analyzed feedback from app stores. Seyff et al. [9] and Schneider et al. [10] proposed to elicit user requirements with feedback continuously from mobile devices.

Mining User Feedback from App Stores. User feedback mining has received a considerable amount of attention in the recent years. Among the most studied platforms for obtaining user feedback are app stores. Martin et al. [11] survey the most relevant work in the area. We focus on literature performing the same steps as our approach: classification, grouping and ranking. Machine learning approaches have often been applied for automatically classifying user feedback e.g., [12], [13], [14]. For grouping similar user feedback, LDA is one of the most used algorithms [4], [15]. The work most related to ours is that by Chen et al. [2], Villarroel et al. [16] and Di Sorbo et al. [17]. All of them present approaches to classify, group and rank app store user reviews automatically and use similar techniques to the ones presented in this work: supervised machine learning for classifying, topic modeling or clustering for grouping, and a scoring function or machine learning for ranking. Nevertheless, to our best knowledge, the topic modeling algorithm used in ALERTme for grouping related tweets (BTM [18]) has not been applied on software-related artifacts so far. The ranking function of our approach, tailored to specific tweet attributes, is also novel. Another major difference of our work to previous work in this area is our focus on short, informal feedback with social components available on Twitter.

Twitter in Software Evolution. Studies in this area have mainly focused on Twitter messages written by developers, while messages written by software end-users have received little attention so far. Previous work investigated the automatic processing of tweets mentioning programming languages, e.g., [19], [20]. Singer et al. [21] interviewed and surveyed developers on their Twitter use. Their results describe developers' information overload and their difficulties in obtaining technical information. In our previous work [1] we investigated the content of tweets about software applications and applied machine learning techniques for classifying tweets relevant for technical and non-technical stakeholders. The major differences of our current work to [1] are that we now investigate tweets to the support accounts of major applications and present a more complete mining process and evaluation that includes classification, grouping and ranking of tweets.

III. The ALERTme approach

The main goal of ALERTme is to process large streams of Twitter messages and automatically identify, group and rank those tweets that are potentially relevant for software evolution, i.e., report problems, suggest improvements or express user needs. From such tweets, developers and product owners can derive issues and new requirements for evolving their product. Figure 1 shows an overview of our approach. After some *preprocessing*, we *classify* the tweets into *improvement requests* and *other* using supervised machine learning. This allows for the filtering of irrelevant information. Then we *group* the tweets containing improvement requests using a topic modeling algorithm. This step yields *topics*, i.e., groups of tweets with semantically similar content which can be used as summaries of related tweets. Finally, we *rank* the individual tweets and topics by using a weighted function on several tweet attributes. The output of this step is two lists of individual tweets and topics, both ordered by relevance. In the following sub-sections we describe each of the steps in detail.

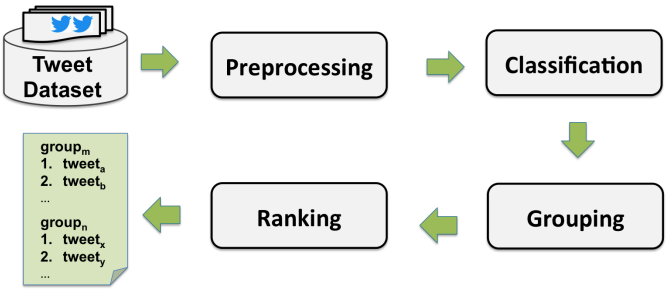

Fig. 1. ALERTme overview.

A. Preprocessing

We prepare the input data by (1) tokenizing all tweet text, (2) converting all text into lower case, (3) extracting n-grams with a one to three word length (these n-grams are only used in the classification step), (4) removing stopwords and (5) stemming the text, thus eliminating inflectional forms of words.

B. Classification

The goal of this step is to classify the tweets automatically into two categories: *improvement request* and *other*. We define *improvement requests* as all tweets reporting *bug reports*, *feature shortcomings* or *feature requests*. This definition is based on the results of an exploratory study [1] that found ten fine-grained categories that can be relevant for technical stakeholders involved in software evolution (e.g., requirements engineers, developers or product owners). We chose the categories that call for explicit enhancements to the software application from these results, as we consider this content the most relevant for software evolution. When using ALERTme in practice, one can decide whether to keep or discard the tweets classified as *other* at the end of this step.

We use supervised machine learning techniques to automatically classify the tweets into the selected categories. In particular, we use Multinomial Naive Bayes (MNB) for the classification task. Our decision is based on the outperformance of Naive Bayes classifiers over other machine learning algorithms when classifying text [22] and its successful application in other software engineering tasks [2], [23], [24].

To train each classifier we apply the following steps on the preprocessed data: (1) convert the preprocessed tweet text into a vector space model using TF-IDF as a weighting scheme, (2) train the classifiers on a set of manually labeled tweets and (3) predict tweet categories using the trained classifier.

C. Grouping

We use a topic modeling algorithm specialized in short text, Biterm Topic Model (BTM) [18], for grouping semantically related tweets. BTM takes as input the preprocessed text of each tweet and outputs (1) the *topics* i.e., groups of words that co-occur in the whole corpus of tweets (for example, the set of words {*crash, update, frustrated, version, bug*} is a topic related to a users' experience when updating a software application) and (2) the *association between the tweets and topics* in the form of a probability matrix. In BTM each tweet is modeled as a mixture of topics i.e., a tweet can be associated to multiple topics. BTM solves the data sparsity problem common in topic modeling when applied on short text [25] by modeling the biterm (unordered word pairs) relationship on the whole tweet corpus. We chose BTM due to its specialization on short text and its outperformance both when grouping short and long text [18] over Latent Dirichlet Allocation (LDA) [26], an algorithm that is commonly applied on longer software engineering artifacts.

D. Ranking

With thousands of tweets to go through, the relevance of each of these tweets needs to be determined automatically so that human analysts can concentrate on the important ones. ALERTme addresses this issue by ranking the individual tweets and topics in *ordered lists*.

Subsequently, we describe the function for ranking individual tweets and topics. Note that we only rank the tweets that are first in the conversation, if several exist. We observed that users report the main topic of the issues they are facing in their first tweet and only follow up with more tweets when the support personnel requires extra information.

Ranking of Tweets. We consider *category, retweets, likes, sentiment, social rank* and *duplicates* to be the *attributes* that influence the relevance of a tweet. We calculate the individual tweet ranking by means of a weighted function, TWR:

$$\text{TWR(tw)} = \sum_{i=1}^{6} w_i * f_i(tw) \quad (1)$$

The f_i are the ranking factors for the considered attributes of a tweet and the w_i are empirically determined weights (see Sect. IV-D). Table I summarizes the factors and their calculation. Subsequently, we present the rationale for the chosen factors.

Category denotes whether the tweet belongs to the *improvement request* or *other* categories defined in Section III-B. We consider only tweets belonging to the *improvement request* category to be relevant for software evolution. Hence, we calculate f_1 with a binary function.

Retweets are the republishing of a tweet. The number of retweets of a given tweet allows for an estimation of its reach. A higher retweet count implies that the tweet will reach more people, which translates to a higher number of users reporting the same improvement request.

Likes are explicit gestures of appreciation towards the concerned tweet. The number of likes provides information about how many people find the tweet interesting. We argue that a high number of likes corresponds to many users having the same issue or request.

TABLE I
FACTORS USED IN THE RANKING FUNCTION.

Attribute	Calculation of ranking factor
Category	f_1(tw) = 1 if category of tw is improvement request, 0 otherwise
Retweets	f_2(tw) = number of times that tw is retweeted
Likes	f_3(tw) = number of likes that tw has received
Sentiment	f_4(tw) = 1 / sentiment score of tw[1]
Social Rank	f_5(tw) = u(tw).followers * u(tw).followers / u(tw).friends[2]
Duplicates	f_6(tw) = number of duplicates of tweet tw

[1]The calculation of the sentiment score is explained in the text
[2]u(tw) is the user who created the tweet tw

Sentiment is the affect or mood expressed in a tweet. We use a lexical sentiment analysis tool specialized in short informal text, SentiStrength [27], which assigns both a positive and a negative score with ranges of [1, 5] and [-1, -5], respectively, to each tweet. 5 denotes an extremely positive and -5 an extremely negative sentiment. 1 and -1 denote the absence of positive and negative sentiment, respectively. We compute a single sentiment score for every tweet in the range of [1, 9] by adding the positive and negative score and then adding 5. As tweets with low sentiment scores typically require more attention than ones containing praise, we use the inverse of the sentiment score to calculate the sentiment factor f_4.

Social Rank is the popularity of Twitter users—characterized by their number of Twitter *followers* and *friends*. We deem tweets by users with a high social rank to be more influential (and thus more important) than those from lower ranked users. Previous research shows that the number of followers is one of the most telling aspects when predicting the influence of a Twitter user [28]. However, exclusively considering the number of followers for measuring influence is problematic, as it does not allow to circumvent bots and users with aggressive following behaviors. To overcome this problem, we define *social rank* as the number of followers multiplied by the ratio of followers to friends.

Duplicates, i.e., lexically similar tweets, are an indicator that several users are discussing the same issue. Thus, tweets with a high number of duplicates should be given some attention. We compare the text similarity of all analyzed tweets using the Jaccard coefficient measure, where $jaccard(tw_i, tw_j) = \frac{|tw_i \cap tw_j|}{|tw_i \cup tw_j|}$, $1 \leq i \leq j \leq n$ and n is the number of tweets. Tweet tw_j is considered a duplicate of tweet tw_i if $jaccard(tw_i, tw_j) \geq \beta$ where β is a predefined threshold. We use the number of duplicates (with $\beta = 0.55$) to calculate the duplicate factor.

The formalization of the TWR function is flexible and new tweet attributes, as well as other forms of computing the already existing attributes can be easily added.

Ranking of Topics. The ranking score given to each topic depends mainly on the *volume* of the tweets relating to the topic and the *average tweet rank* within the topic. In

TABLE II
DATASET.

Application	Domain	#Tweets	Tweets directed to
Spotify	Music	36,386	SpotifyCares account[1]
Slack	Business	29,287	SlackHQ account[2]
Dropbox	Productivity	2,435	DropboxSupport account[3]
Total	–	68,108	–

[1]https://twitter.com/spotifycares [2]https://twitter.com/slackhq
[3]https://twitter.com/dropboxsupport

particular, we consider that the larger the number of highly ranked tweets associated to a topic, the more important the topic. The grouping step of ALERTme produces a probability distribution for each tweet over the k generated topics. This probability distribution can be represented as the matrix $p_{tw,t} = \{p_{tw_1,t_1}, ..., p_{tw_1,t_k}, ..., p_{tw_n,t_1}, ..., p_{tw_n,t_k}\}$ where n is the number of tweets. We use these probabilities along the already calculated tweet ranks, TWR, to calculate the topic rank, TR, as follows:

$$\text{TR(t)} = \sum_{i=1}^{n} p_{tw_i t} * \text{TWR}_i \qquad (2)$$

IV. EVALUATION

The main goal of the evaluation is to assess the result quality of the different steps of ALERTme. The questions that guided our evaluation are:

Q1. Classification: How accurate are machine learning techniques when automatically classifying tweets relevant to software evolution?

Q2. Grouping: How accurate is topic modeling for grouping tweets about software applications according to their content? Are the results from topic modeling coherent according to technical stakeholders (e.g., developers, requirements engineers and product owners)?

Q3. Ranking: How accurate is our ranking approach compared to the assessment of technical stakeholders (e.g., developers, requirements engineers and product owners)?

To answer our questions we collected a large sample of tweets related to three different applications and performed three experiments on this data. We subsequently describe our dataset and the performed experiments.

A. Dataset

Our dataset consists of 68,108 tweets. We collected tweets directed to the support accounts of three popular software applications: Spotify, Dropbox and Slack. We decided to collect data from these three applications due to their popularity, the high frequency of tweets targeted to their support accounts and the domain diversity between the three applications. We decided to collect feedback exclusively directed to support accounts as we assumed that these tweets might have a higher probability of being relevant to software evolution. While the Spotify and Dropbox accounts are exclusively dedicated to giving support to end-users, the Slack account is more generic and besides support is also used for marketing purposes.

We used an open-source library[1] to access the Twitter Search API[2] and import the tweets written in English whose content either mentioned[3] or replied[4] to the accounts of the aforementioned applications. We imported the tweets for a duration of two months, from May 4th until July 5th, 2016. Table II shows the selected software applications, their domain and the number of imported tweets for each one. All three applications have over 2,000 collected tweets.

B. Classification

In this sub-section, we describe the experiment setup and the metrics used to measure the classification performance. Additionally, we present and discuss the results.

1) Setup: During our experiment setup we created a truthset to train the classifier and validate its results.

a) Truthset: We created the truthset by using the content analysis methods described by Neuendorf [29]. Three annotators systematically analyzed the content of a tweet sample according to an annotation guideline containing the category definitions and rules, as well as examples for each category. All annotators were graduate students from the Technical University of Munich with software engineering experience. To assure that the task was clear and avoid major disagreements, two trial runs were executed. During the trials common misunderstandings were discussed. Afterwards, each annotator independently labeled 450 tweets for each software application (same sample for all annotators). These tweets were randomly selected. In total, 81% of the annotations resulted in a complete consensus among the three annotators. Disagreements were handled via a majority voting scheme. For the cases in which the majority scheme did not yield a specific label, two of the authors manually inspected the tweets and came to an agreement about the final label of the tweet.

We consider these 1,350 tweets as our truthset. On average each annotator took 13.6 hours to complete the task, confirming the large amount of time required to manually analyze user feedback [7], [12]. Overall, 42% of the tweets in the classification truthset were labeled as *improvement request*.

b) Training and Comparisons: We trained and tested the classifiers with a 10-fold cross validation on the previously described truthset. Furthermore, we compared the results of the Multinomial Naive Bayes classifier against the results of a Random Forest classifier, which had good results when classifying user feedback from app reviews [16]. The training and evaluation of the classifiers was performed using Weka[5].

2) Metrics: We evaluate the classifier performance using three metrics traditionally used in machine learning: precision, recall and F-Measure. $Precision = \frac{TP}{TP+FP}$ and $Recall = \frac{TP}{TP+FN}$, where TP is the number of tweets correctly classified as belonging to a category, FP is the number of tweets incorrectly classified as belonging to a category and FN is

[1]http://www.tweepy.org
[2]https://dev.twitter.com/rest/public/search
[3]Makes reference to application account anywhere in the tweet.
[4]Direct response to the application account.
[5]http://www.cs.waikato.ac.nz/ml/weka

TABLE III
EXAMPLES OF CORRECTLY PREDICTED RESULTS BY THE MULTINOMIAL NAIVE BAYES CLASSIFIER.

Application	Example	Prediction
Spotify	**(1)** @Spotify @SpotifyCares hey guys when are the lyrics coming back! I need them!!!	*Improvement request*
Spotify	**(2)** @SpotifyCares Hi! paid 99p for 3 months spotify. Payment went through, but didn't give me premium. Now this happens; https://t.co/Lhs5DRQFj4	*Improvement request*
Slack	**(3)** @SlackHQ At my company we share code snippets around a lot. There should be a quick way to copy a raw code snippet to your clipboard.	*Improvement request*
Dropbox	**(4)** @DropboxSupport iOS app takes up gigabytes of space and only thing that helps is reinstall. Why is nobody fixing it? https://t.co/c685U3dGp8	*Improvement request*
Slack	**(5)** I always uwanted t-shirts, but I didn't know socks were an option. I've got the start with my @SlackHQ faves - gotta catch 'em all!	*Other*
Dropbox	**(6)** @DropboxSupport You're great. Thanks.	*Other*

TABLE IV
CLASSIFICATION RESULTS[1].

	Multinomial Naive Bayes			Random Forest		
	P	**R**	**F**	**P**	**R**	**F**
Improvement req.	0.74	**0.78**	**0.76**	**0.79**	0.59	0.68
Other	**0.83**	0.80	**0.82**	0.75	**0.89**	0.81
Weighted Average	0.74	**0.79**	**0.79**	**0.77**	0.76	0.76

[1]We use P for precision, R for recall and F for F-Measure.

the number of tweets that are incorrectly classified as not belonging to a category. The F-Measure is defined as the harmonic mean of the precision and recall.

3) Results: Table III shows examples of tweets and their correctly predicted results by the Multinomial Naive Bayes classifier. Moreover, Table IV shows the results of both evaluated classifiers. Overall, the results from the classification step are encouraging. The Random Forest classifier had a moderately better precision for the *improvement request* category, while the recall was significantly higher for the Multinomial Naive Bayes classifier in this category. The Multinomial classifier outperformed Random Forest on the F-Measure, with 0.76 for the *improvement request* category—an 8% difference.

4) Discussion: These results are comparable to those reported in previous work when classifying app reviews into two categories by using machine learning [2]. A manual inspection of the predicted results revealed that many misclassifications occurred with tweets reporting account issues. Tweets requesting non-technical support from the software company for solving account problems were wrongly predicted as *improvement requests* when there was no obvious malfunctioning in the software and tweets related to account issues were predicted as *other* when there appeared to be an error in the software. The cause of this problem could be the similar semantics between both types of tweets, e.g., the two tweets (1) *"@SpotifyCares @Spotify Phone logged me out and I forgot my password - of course 'forgot password' function doesn't work. Help..Help me!"* which was incorrectly classified as *other* when there appears to be an in the software and (2) *"@SpotifyCares hi there I am having some account issues can you please help"* which was incorrectly classified as *improvement request* when there is no reported malfunctioning of the software.

The predicted *improvement requests* could be used to identify and fix issues in the software (e.g., Example 2 and 4 in Table III could lead to the fixing of a payment and storage issue, respectively), as well as to elicit new requirements (e.g., Example 1 and 3 in Table III could lead to the elicitation of requirements regarding the addition of song lyrics and code snippet support, respectively).

In sum, the application of machine learning techniques for the classification of tweets into the *improvement request* and *other* categories is encouraging—with an average F-Measure of 0.79 for the best performing algorithm, Multinomial Naive Bayes.

C. Grouping

In this sub-section, we describe the setup and used metrics of the experiment conducted to evaluate the quality of the results produced by the grouping step of ALERTme. Additionally, we present and discuss the results.

1) Setup: Topics can be used to group semantically related content. Moreover, the topics—commonly represented as groups of words (see Table V)—can be used to summarize the content of the tweets associated to them. We were interested in measuring the accurateness and coherence of the topics from a software practitioner's perspective. To achieve this goal we measured the coherence of the generated topics and the association quality between the individual tweets and the topics according to the judgement of software practitioners.

We evaluated the quality of the generated topics from the three applications present in our dataset. We selected all tweets classified as *improvement request* by our classifier (cf. Sect. III-B) and fed them into the BTM algorithm (cf. Sect. III-C) separately for each the three applications. We used the BTM implementation provided by Yan et al. [18]. The BTM algorithm requires that we input the number of topics, k, to be generated by the algorithm. We chose the k used by BTM by manually assessing the results of the algorithm for $k = \{10, 15, 20, 25, 30\}$. For each application we chose the k value that we judged to yield the most coherent topics ($k = 10$ for Dropbox, $k = 20$ for Spotify and Slack) and used it for the subsequent experiment.

We systematically measured the coherence of the generated topics and the association accuracy between tweets and topics

with the help of two human assessment tasks [5]: *word intrusion* and *topic intrusion*.

The *word intrusion* task evaluates the semantic coherence of the topics according to human judgement. In the task we show a list of four words belonging to a topic (according to the topic modeling algorithm) and a word with a very low probability[6] of belonging to the topic, a *word intruder*, in random order. The task of the experiment participant is to identify the word intruder. If a topic is coherent it should be easy for participants to find the intruder (e.g., in the topic {*font, color, format, device*} the word *device* can be easily identified as the word intruder). Otherwise, if the topic lacks coherence, it is difficult to identify the intruder (e.g., in the topic {*format, error, crash, sync*} it is unclear which word is the intruder) and participants will usually make a random choice when performing the task.

The *topic intrusion* task tests if the association between topics and tweets is accurate according to human judgement. In the task we present the tweet text, along with at most four topics associated to it with a high probability (threshold of 0.50) by the topic modeling algorithm and an *intruder topic*. The task of the participant is to identify the intruder topic. Similar to the word intrusion task, if the association between the tweets and the topics is accurate and intuitive, participants should not have any trouble identifying the intruder. However, if this is not the case, participants will likely choose randomly.

We evaluated ten randomly chosen topics for each software application. Similarly, we also evaluated ten randomly chosen tweets. Each task was executed twice by two different participants for the selected tweets and topics of each software application. Six people, all software developers working in industry, performed the grouping evaluation. To avoid bias, none of the participants were familiar beforehand with the generated topics and the procedure for its generation. Figure 2 shows an example of a word intrusion and topic intrusion task from the actual experiment.

We compared the BTM results against the results produced by the Latent Dirichlet Allocation (LDA) algorithm [26]. LDA is traditionally used on longer text and has been applied for the grouping and summarization of user feedback provided in different channels, such as app stores [2], [4] and bug reports [30]. We chose the same k values for running LDA as we had used for BTM. For the comparison we used the same word and topic intrusion tasks described above. Similar to the BTM evaluation, ten tasks of each type were executed for each application by the same two participants that evaluated the BTM results. We used the LDA implementation jLDADMM[7].

2) Metrics: For evaluating the topics we use the Model Precision (MP) [5] and Topic Precision (TP) metrics. The metrics are based on the results of the previously described topic and word intrusion tasks. MP defines the proportion of participants concurring with the topic model with respect to which word is considered an intruder. Let w_k^m be the index of the intruding word generated from the k^{th} topic inferred by model m and $i_{k,p}^m$ be the intruder selected by participant p on the set of words of the topic k created by the model m. Additionally, let P be the total number of participants executing the task for the specific m. MP is then defined as follows:

$$\text{MP}_k^m = \sum_P \phi(i_{k,p}^m, w_k^m)/P$$
$$\text{where } \phi(i_{k,p}^m, w_k^m) = \begin{cases} 1 & \text{if } i_{k,p}^m = w_k^m \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

Similarly, TP defines the proportion of participants concurring with the topic model with respect to which topic is considered an intruder.

3) Results: Table VI summarizes the MP and TP results for each software application and Table V shows topics generated by BTM. Note that according to the MP and TP definitions, the closer the MP and TP values to 1, the higher the coherence and better association accuracy between tweets and topics (see Equation 3). Overall, BTM outperformed LDA. The association accuracy within topics and tweets (TP) was strong for all three applications—with BTM having an average TP value of 0.88 and outperforming LDA in two of the three cases. The coherence results (MP) among the different applications varied considerably, with BTM having the best results—an average of 0.52 among all three applications. Slack topics had a very good coherence, with a MP of 0.70 for BTM, while the Spotify had a fair coherence, with a MP of 0.55 for BTM. However, the Dropbox results for coherence were less encouraging with a MP of 0.30.

4) Discussion: The variation among the results of the different applications could be an indicator of a need for

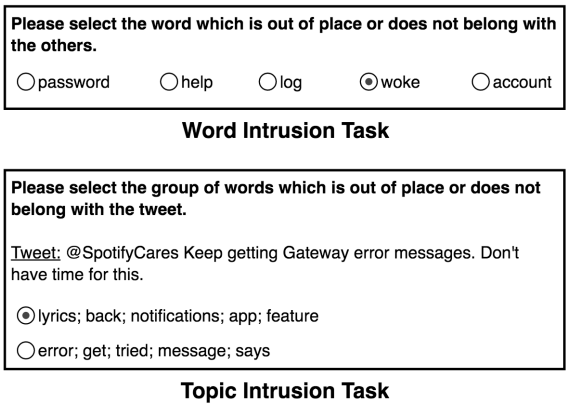

Fig. 2. Examples of word and topic intrusion tasks.

TABLE V
EXAMPLES OF TOPICS GENERATED BY BTM FOR EACH OF THE ANALYZED APPLICATIONS[1].

Application	Topic
Spotify	song, play, playlist, music, stop, listen, skip, everi, track, keep
Slack	messag, file, upload, delet, channel, imag, like, app, link, featur
Dropbox	upload, photo, file, app, folder, camera, save, iphon, featur, io

[1]The content of the topics is stemmed.

[6]We consider a probability as *low* when it is in the bottom quarter of all probabilities related to the topic or tweet in question.

[7]http://jldadmm.sourceforge.net

TABLE VI
GROUPING RESULTS.

	Spotify		Slack		Dropbox		Average	
	BTM	LDA	BTM	LDA	BTM	LDA	BTM	LDA
MP	0.55	0.40	0.70	0.55	0.30	0.30	0.52	0.42
TP	0.95	1	0.80	0.60	0.90	0.75	0.88	0.78

better fine-tuning of the k parameter in our topic models—a challenging task [3]. We manually analyzed the output of both algorithms and found that the generated topics could be relevant for software evolution as they were clearly associated to the functionality of the software. We also found that the topics generated by BTM were more informative and descriptive, compared to those generated by LDA. Both algorithms produced duplicate and mixed topics[8], as reflected in the MP values. From the three analyzed applications, Dropbox had the largest number of duplicate and mixed topics. The low amount of data analyzed for this application and the fact that most of their users reported on very similar issues (e.g., inability to access a file, folder or the application's web site) could be possible explanations for this and thus, for the lower MP values.

Previous research on app store reviews [3],[4] manually analyzed the results of topic modeling algorithms and reached similar conclusions with regard to the presence of mixed and duplicate topics.

Another possible reason for the fair MP results is that interpreting topic models in isolation is a difficult task as its assessors might not have all necessary context to fully understand them. In this respect, an interactive visualization that allows for the navigation between topics and individual tweets might aid in the interpretation of the topics and could be a useful addition for the use of topic models during software evolution.

In sum, our results show that the association accuracy between topics and tweets is strong and that the coherence of the generated topics is reasonable. Overall, BTM outperformed LDA in both evaluated aspects.

D. Ranking

We evaluated the ranking step of ALERTme by comparing its results against the judgement of software practitioners. To assign the weights used in the ranking function (see Equation 1) we used the results of a survey [31] in which we studied the importance of tweet attributes when ranking tweets. In the following sub-sections we describe the experiment setup and used metrics. Also, we present and discuss the results.

1) Setup: We describe the truthset used in this experiment and how we calculated the weights of the ranking function subsequently.

a) Truthset: We compared the results from the ranking step of ALERTme against the judgement of seven software practitioners. They rated the relevance of 110 randomly selected tweets for each software application in our dataset on a four-level scale determining how fast they should react to the specific tweet[9]. During this process we presented the practitioners the tweet text and its corresponding attributes (see Section IV-D).

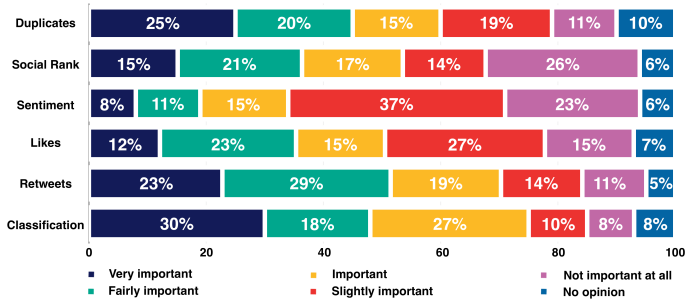

Fig. 3. Tweet attributes that affect tweet ranking according to surveyed participants [31].

The tweets of each application were assessed twice by two software practitioners, with the exception of Spotify—which were assessed three times. In this case, one of the assessments was done by a software developer from the company. All other participants were not involved in the development of the actual applications for which they rated the tweets, but were familiar with the concerned application as end-users. Participants of the ranking experiment did not answer the survey [31] used to determine the tweet attribute weights (see Section IV-D1b).

We handled the disagreements between participants by averaging their relevance ratings. We used these averages to compare against the ranking results of our approach. During this process we did not average the annotation by the Spotify developer, but treated it as a separate comparison.

The topic ranking performance is solely dependent on the accuracy of the ranking of the individual tweets belonging to the topic and of the association quality between topics and tweets. Therefore, a high accuracy in the individual tweet ranking will lead to good results in the topic ranking, provided that the association between tweets and topic is correct. The results from Section IV-C show that the accuracy between tweets and topics is high. Thus, in this section we focus on evaluating the accuracy of ranking individual tweets.

b) Finding the weights of the ranking function: To determine the weights of each tweet attribute we use the survey results of our previous work [31]. In this survey we asked 84 software engineering practitioners and researchers to rate the relevance of tweet attributes on a Likert scale[10]. Figure 3 summarizes its main results.

Let $A = \{a_1, ..., a_6\}$ be the set of tweet attributes described in Section IV-D, $P = \{p_1, ..., p_{84}\}$ the set of survey participants, $r(p_i, a_j)$ the rating given by participant p_i to attribute a_j, and $W = \{w_1, ..., w_6\}$ the set of weights associated to A. The computation of the weight w_k for attribute a_k can be

[8]Topic with more than one main *theme*.

[9]The scale values were defined as follows. 3: tweets that need to be addressed that same day or week, 2: tweets that need attention during the next 2-4 weeks, 1: tweets that need attention sometime afterwards, 0: unimportant tweets that need no attention.

[10]5 denotes *very important*, 1 denotes *not important at all*.

formally described as the following ratio:

$$w_k = \frac{\sum_{i=1}^{84} r(p_i, a_k)}{\sum_{i=1}^{84}\sum_{j=1}^{6} r(p_i, a_j)} \quad (4)$$

We refer to this weighting scheme as $\boldsymbol{W_{survey}}$. Additionally, we compare against three additional weighting schemes: $\boldsymbol{W_{survey\_m}}$, which uses the W_{survey} weighting scheme but has manually assigned labels for the *category* attribute—instead of the ones automatically predicted by the classifier, used by the rest of the weighting schemes in this experiment, $\boldsymbol{W_{no\_social}}$ where all social attributes (i.e., retweets, likes and social rank) are eliminated and emphasis is given on the following attributes: category (w_1=0.50), duplicates (w_6=0.33) and sentiment (w_4=0.16), while $w_2 = w_3 = w_5 = 0$ and $\boldsymbol{W_{no\_mentions}}$ which also uses the W_{survey} scheme but which only ranks tweets that are *not mentions*[11].

We ran the implementation of the ranking function with the four weight variations on all of the tweets in our truthset. We performed three separate runs, one for each of the analyzed applications.

2) Metrics: NDCG [32] is used to measure the quality of ranking algorithm results, usually from search engines. It varies from 0 to 1, with 1 representing the ideal ranking of the documents—tweets, in our case. It is computed as:

$$\text{NDCG} = \frac{\text{DCG}}{\text{IDCG}} \text{ where DCG} = \sum_{i=1}^{n} \frac{rel_{tw_i}}{log_2(i)} \quad (5)$$

rel_{tw_i} is the relevance[12] of the tweet tw in the ranking position i according to the truthset (see Section IV-D1a) and IDCG is the DCG computation of the ideal ranking according to the truthset. We compute NDCG@10, a version of NDCG which considers the top 10 ranked tweets, since we are especially interested in assessing the performance of ALERTme when retrieving the most relevant tweets.

3) Results: Table VII summarizes the results of the ranking evaluation and Figure 4 shows examples of highly ranked tweets. Overall, the results of the ranking step with the different weighting schemes are encouraging, and with the exception of the two worst performing variations for Slack, are better than those reported in the literature when ranking app reviews using weighted functions [2]. The results of W_{survey} and W_{no_social} are comparable with slight improvements when using the latter (with the exception of Slack). Among all evaluated schemes, $W_{no\_mentions}$ has the highest results for Spotify′, Spotify and Slack, and slightly lower for Dropbox.

4) Discussion: The results of the different weighting schemes varied the most among the tweets from Slack, where the removal of noise from the automatic classification (W_{survey_m}) and the removal of mentions ($W_{no\_mentions}$) resulted in a significant improvement. As described in Section IV-A, the Slack account is used for both marketing and support purposes, whereas the Spotify and Dropbox accounts are used purely for support. General software accounts are more likely to contain a larger number of mentions as they contain more promotional and informative content about the software (e.g., blogs, news, additions to the software). While the content of mentions is most likely irrelevant for software evolution, mentions are seen by everyone who follows the user that composed a tweet. In contrast replies are only seen by the composer and receiver of the tweet in question[13]. This leads to more visibility to mentions and translates into more social traction in the shape of retweets and likes, giving potentially irrelevant information a higher rank. Thus, weighting schemes that take these social components into consideration should be further investigated.

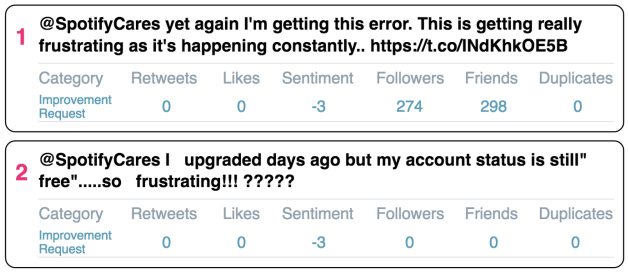

Fig. 4. Example of top two ranked tweets and their attributes for Spotify using W_{survey}.

It is also interesting to note that the ranking function performed best when compared against the results of the Spotify developer. However, we currently cannot make generalizations from this result and further evaluations need to be conducted to draw more concrete conclusions.

The way in which we constructed our truthset only allowed us to evaluate the relevance of tweets in terms of urgency, i.e., how fast practitioners should react to a tweet by, for example, making a fix. However, in the context of requirements engineering, additional aspects should be considered for determining the relevance of user feedback. For example, a report of a bug that is causing the system to crash might have a higher urgency than a feature request that has been retweeted many times and that was originally written by a user with high social rank. Nevertheless, in the context of requirements engineering this request might be more relevant than the bug report originating the crash, despite having less urgency. In our future work, we will assess our ranking function against a truthset in which practitioners take additional aspects into consideration for determining tweet relevance.

In sum, the results from the ranking function are promising. With the exception of one application, the best results were obtained when removing tweets that are mentions—as they generally contain data that is irrelevant for software evolution.

E. Threats to Validity

Despite the encouraging results, this work has three main potential threats to validity.

The evaluation of the three main steps of ALERTme critically depends on the quality of the (manually created)

[11]As explained in Section IV-A, *mentions* are tweets that make reference to an application account anywhere in the tweet (whereas *replies* are direct responses to the application account).

[12]Given in the 4-level scale ([0,3]) described in Section IV-D1a

[13]As well as users who follow both the composer and receiver of the tweet.

TABLE VII
NDCG@10 RESULTS[1].

Variation	Spotify′	Spotify	Slack	Dropbox
W_{survey}	0.91	0.83	0.32	0.48
W_{survey_m}	0.91	0.82	0.71	**0.55**
W_{no_social}	0.94	0.83	0.32	0.52
$W_{no\_mentions}$	**0.95**	**0.84**	**0.79**	0.51

[1]Spotify′ denotes result according to Spotify developer

truthsets and topic evaluation tasks. To address this threat, we created our classification and ranking truthsets based on the judgement of more than one annotator or participant. Additionally, for the creation of the classification truthset we created an annotation guide with definitions and examples, and conducted trial runs to minimize the disagreement among annotators. To avoid strong disagreements on the ranking truthset, we provided the annotators with definitions of the used scale values. However, we did not provide examples of what constitutes a relevant tweet on each scale because we wanted to validate the results of our ranking step against the criteria of software practitioners—and not on a set of predefined rules. We reduced the subjectivity of the word and topic intrusion tasks of the grouping evaluation by providing the participants with examples of the tasks and by alternating the order of the BTM and LDA results.

Most of the software practitioners who participated in the experiments are not directly involved in the evolution of the applications related to the assessed tweets or topics. Nevertheless, they were familiar with the applications as end-users and all were working in the software industry. However, their criteria about what constitutes a relevant tweet or a coherent topic could be different to that of those actively involved in the evolution of the application.

We mitigated external validity threats by considering software applications from three different domains. However, all three are large, popular applications. Hence, we currently cannot generalize our results to small, less popular applications.

V. CONCLUDING REMARKS

A. Summary

We present ALERTme, an approach to automatically *classify*, *group* and *rank* tweets for their use during software evolution. The evaluation results of the three main steps of our approach are encouraging. ALERTme is able to classify tweets automatically into the *improvement request* and *other* categories with a F-Measure of 0.79. The groups of tweets generated by the topic modeling algorithms have a reasonable coherence and the association quality between the generated topics and analyzed tweets is very satisfactory. Additionally, our evaluation shows that the ranking step produces results that strongly agree with practitioners' judgement.

Our results show that applying techniques that are the same or similar to those applied on app store reviews for classification and grouping, as well as our novel ranking function are promising directions for mining user feedback from Twitter data. Thus we are confident that our research will eventually lead to practice-oriented tools for eliciting requirements from large amounts of tweets.

B. Overall Discussion

All individual outcomes of the three main steps of ALERTme can be useful for developers, requirements engineers and product owners.

The ***classification*** helps identify tweets that contain *improvement requests*. This is useful for software evolution as these tweets can contain valuable information for eliciting new requirements as well as identifying problems to be fixed. For example, from the tweet *"At my company we share code snippets around a lot. There should be a quick way to copy a raw code snippet to your clipboard"* (cf. Table III), a requirements engineer or product owner could easily derive a requirement written as a new user story: *"As a Slack user, I want to copy a raw code snippet to the Slack clipboard so that I can share code snippets with co-workers in my company."* On the other hand, the tweet *"OS app takes up gigabytes of space and only thing that helps is reinstall. Why is nobody fixing it?"* gives developers an indication that there is a problem with the storage management of the app.

The ***grouping*** helps sort these *improvement requests* into semantically-related collections, further summarizing the information and potentially reducing the cognitive overload of analyzing individual tweets with the purpose of eliciting requirements or searching for potential software issues during software evolution.

Finally, the ***ranking*** prioritizes the individual tweets and generated summaries (i.e., topics or groups of tweets), giving pointers to those tweets that probably are most worthwhile to analyze and follow-up. For example, if a tweet such as the one on the OS app taking too much space (see above) has a high rank, this is an indicator that this problem should be fixed urgently.

When creating the truthset (see Section IV-B1), we noticed that 42 percent of the manually analyzed tweets contain relevant information for software evolution. In contrast, in our previous work we found that only 2.5 percent of the analyzed tweets are relevant for software evolution [1]. This striking difference is explained by the fact that in [1] we analyzed "generic" tweets i.e., tweets that mention the software, but are not necessarily directed to a support account, while in this paper, we only consider tweets directed to the support accounts of the concerned software companies. The 42 percent found in this research is also higher than the percentage in app reviews, where Pagano and Maalej [7] found that 34 percent of the reviews contained this type of content. Twitter has the advantage of being a widely used tool for communication—not only about software, so that end-users do not need to change their context for providing their feedback. It also allows for bi-directional communication and social incentives. These factors could explain the larger proportion of tweets relevant to software evolution when compared with app store reviews.

One could argue that, due to the limited length of tweets, the information provided over Twitter is insufficient for communicating requirements or reporting bugs. However, during the collection of our dataset and its manual analysis, we noticed that software companies actively engage with users in order to obtain additional information about reported bugs and requested features—and that users follow up on these conversations, enhancing their messages with additional links and screenshots.

From these observations we conclude that in software evolution, it is worthwhile to explore user tweets as an additional information channel, particularly when analyzing tweets directed to the support accounts of software companies. We envisage future tools that link and combine feedback from multiple channels (e.g., Twitter, bug reports, app store reviews, interview or workshop protocols and usage data) for obtaining information and inspiration from users about how to evolve a software product.

We are confident that our ALERTme approach will eventually lead to practice-oriented tools exploiting Twitter as a feedback channel for information relevant to software evolution, including stakeholders' desires and needs, thus complementing the existing set of techniques and tools for requirements elicitation.

ACKNOWLEDGMENTS

We thank Simon Ruesch, Rana Alkadhi, Eya Ben Charrada and Irina Koitz for their insightful feedback on our work. We also thank our evaluation participants for their time and comments. This work was partially supported by the European Commission within the SUPERSEDE project (ID 644018).

REFERENCES

[1] E. Guzman, R. Alkadhi, and N. Seyff, "A needle in a haystack: what do Twitter users say about software?" in *Proc. of the International Requirements Engineering Conference*, 2016, pp. 96–105.

[2] N. Chen, J. Lin, S. C. Hoi, X. Xiao, and B. Zhang, "AR-Miner: Mining informative reviews for developers from mobile app marketplace," in *Proc. of the International Conference on Software Engineering*, 2014, pp. 767–778.

[3] L. V. Galvis Carreño and K. Winbladh, "Analysis of user comments: an approach for software requirements evolution," in *Proc. of the International Conference on Software Engineering*, 2013, pp. 582–591.

[4] E. Guzman and W. Maalej, "How do users like this feature? A fine grained sentiment analysis of app reviews," in *Proc. of the International Requirements Engineering Conference*, 2014, pp. 153–162.

[5] J. Chang, S. Gerrish, C. Wang, and D. M. Blei, "Reading tea leaves: How humans interpret topic models," in *Advances in Neural Information Processing Systems 22*, Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, and A. Culotta, Eds., 2009, pp. 288–296.

[6] D. Pagano and B. Bruegge, "User involvement in software evolution practice: A case study," in *Proc. of the International Conference on Software Engineering*, 2013, pp. 953–962.

[7] D. Pagano and W. Maalej, "User feedback in the appstore: an empirical study," in *Proc. of the International Requirements Engineering Conference*, 2013, pp. 125–134.

[8] L. Hoon, R. Vasa, J.-G. Schneider, J. Grundy, and Others, "An analysis of the mobile app review landscape: trends and implications," *Swinburne University of Technology, Tech. Rep*, 2013.

[9] N. Seyff, G. Ollmann, and M. Bortenschlager, "AppEcho: A user-driven, in situ feedback approach for mobile platforms and applications," in *Proc. of the International Conference on Mobile Software Engineering and Systems*, 2014, pp. 99–108.

[10] K. Schneider, S. Meyer, M. Peters, F. Schliephacke, J. Mörschbach, and L. Aguirre, "Feedback in context: Supporting the evolution of IT-ecosystems," in *Product-Focused Software Process Improvement*. Springer, 2010, pp. 191–205.

[11] W. Martin, F. Sarro, Y. Jia, Y. Zhang, and M. Harman, "A Survey of App Store Analysis for Software Engineering," *IEEE Transactions on Software Engineering*, 2016.

[12] E. Guzman, M. El-Halaby, and B. Bruegge, "Ensemble methods for app review classification: An approach for software evolution," in *Proc. of the Automated Software Enginering Conference*, 2015, pp. 771–776.

[13] W. Maalej and H. Nabil, "Bug report, feature request, or simply praise? On automatically classifying app reviews," in *Proc. of the International Requirements Engineering Conference*, 2015, pp. 116–125.

[14] S. Panichella, A. Di Sorbo, E. Guzman, C. Visaggio, G. Canfora, and H. Gall, "How can I improve my app? Classifying user reviews for software maintenance and evolution," in *Proc. of the International Conference on Software Maintenance and Evolution*, 2015, pp. 281 – 290.

[15] C. Iacob and R. Harrison, "Retrieving and analyzing mobile apps feature requests from online reviews," in *Proc. of the Working Conference on Mining Software Repositories*, 2013, pp. 41–44.

[16] L. Villarroel, G. Bavota, B. Russo, R. Oliveto, and M. Di Penta, "Release planning of mobile apps based on user reviews," in *Proc. of the International Conference on Software Engineering*, 2016, pp. 14–24.

[17] A. Di Sorbo, S. Panichella, C. V. Alexandru, J. Shimagaki, C. A. Visaggio, G. Canfora, and H. C. Gall, "What would users change in my app? summarizing app reviews for recommending software changes," in *Proc. of the International Symposium on Foundations of Software Engineering*, 2016, pp. 499–510.

[18] X. Yan, J. Guo, Y. Lan, and X. Cheng, "A biterm topic model for short texts," in *Proc. of the International Conference on World Wide Web*, 2013, pp. 1445–1456.

[19] P. K. Prasetyo, D. Lo, P. Achananuparp, Y. Tian, and E. P. Lim, "Automatic classification of software related microblogs," in *Proc. of the International Conference on Software Maintenance*, 2012, pp. 596–599.

[20] P. Achananuparp, I. N. Lubis, Y. Tian, D. Lo, and E.-P. Lim, "Observatory of trends in software related microblogs," in *Proc. of the International Conference on Automated Software Engineering*, 2012, pp. 334–337.

[21] L. Singer, F. Figueira Filho, and M.-A. Storey, "Software Engineering at the speed of light: How developers stay current using Twitter," in *Proc. of the International Conference on Software Engineering*, 2014, pp. 211–221.

[22] R. Caruana and A. Niculescu-Mizil, "An empirical comparison of supervised learning algorithms," in *Proc. of the International Conference on Machine Learning*, 2006, pp. 161–168.

[23] A. Bacchelli, T. Dal Sasso, M. D'Ambros, and M. Lanza, "Content classification of development emails," in *Proc. of the International Conference on Software Engineering Pages*, 2012, pp. 375–385.

[24] B. Turhan and A. B. Bener, "Software defect prediction: Heuristics for weighted Naive Bayes." in *Proc. of the International Conference on Software Technologies*, 2007, pp. 244–249.

[25] L. Hong and B. D. Davison, "Empirical study of topic modeling in Twitter," in *Proc. of the First Workshop on Social Media Analytics*, 2010, pp. 80–88.

[26] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent Dirichlet Allocation," *The Journal of Machine Learning Research*, vol. 3, pp. 993–1022, 2003.

[27] M. Thelwall, K. Buckley, G. Paltoglou, D. Cai, and A. Kappas, "Sentiment strength detection in short informal text," *Journal of the American Society for Information Science and Technology*, vol. 61, no. 12, pp. 2544–2558, 2010.

[28] E. Bakshy, J. M. Hofman, W. A. Mason, and D. J. Watts, "Everyone's an influencer: quantifying influence on Twitter," in *Proc. of the International Conference on Web Search and Data Mining*, 2011, pp. 65–74.

[29] K. Neuendorf, *The content analysis guidebook*. Thousand Oaks, CA: Sage Publications, 2002.

[30] K. Somasundaram and G. C. Murphy, "Automatic categorization of bug reports using Latent Dirichlet Allocation," in *Proc. of the India Software Engineering Conference*, 2012, pp. 125–130.

[31] E. Guzman, M. Ibrahim, and M. Glinz, "Prioritizing user feedback from Twitter: A survey report," in *International Workshop on Crowd Sourcing in Software Engineering*, 2017, pp. 21–24.

[32] S. Büttcher, C. L. A. Clarke, and G. V. Cormack, *Information retrieval: Implementing and evaluating search engines*. MIT Press, 2016.