Applying Text Classification for Post-Release Issue Triage

***Abstract*—Experience Paper — Software organizations spend a lot of effort discovering, triaging, and fixing customer-reported issues with their software. Their goal is to discover and fix as many issues as possible with minimal cost, while avoiding the introduction of new problems in the process. Despite first-round defenses, such as diagnostic data, extensive testing, and staggered releases, sometimes new issues pop up. Users are quick to discuss issues in their software, especially on social media. While social media platforms such as Twitter offer new valuable data sources for companies to learn about issues with their products, manually triaging tweeted complaints requires substantial effort, almost as difficult as finding needles in a haystack. To use this feedback channel effectively, companies must turn to automated methods to cheaply discover impactful issues. In this paper, we describe how a large US-based software company applied machine learning to filter and classify Twitter posts related to issues that customers noticed after installing service releases for one of its products. We found that it is possible to effectively and efficiently filter irrelevant tweets with high precision, and thereby reduce the daily workload of the triage engineers tasked with classifying them. Our work has enabled the company to identify and fix issues as early as possible, minimizing their impact to all of its customers.**

***Index Terms*—triage, classification, Twitter**

I. Introduction

Bugs — defined in the software industry as defects, errors, or conflicts within a code base or the intersection of operations between software —- are an inevitable reality of technology. Developers must constantly stay ahead of new hardware updates, feature changes, and security holes. To this end, some software companies release periodic, automated updates to keep their software products functioning for all its customers [1]. These updates are released in stages, meaning that groups of customers receive updates gradually over time. The stages generally start with internal users, move to enthusiasts and other volunteers, and end with home and enterprise users. This process allows companies to manage risk; although update releases are intended to fix a set of problems, they may unintentionally cause new ones. We call these kinds of issues *regressions*.

Regressions risk the health of software companies' ecosystems. If updates break other features and programs, users may be incentivized to block them, preventing the deployment of critical security patches. To catch flaws as quickly as possible, one large US-based software company with whom we worked (Company A) employs a team whose primary mission is to monitor the update release process. The team searches for regressions, quickly assembles reports on the most severe problems, and alerts relevant product teams to fix the issues. If an update proves to be excessively problematic, the team halts its distribution, preventing further adoption until the issues with the update are solved.

In the era before social media and pervasive Internet access, regressions could be discovered through explicit manual and automated testing by internal teams and external early adopters [1], [2]. If a regression was not identified in this test phase, it could cause a high volume of technical support calls when the release was sent out. The long latency between the release, the support calls, and the company's response potentially exposed many customers to the problematic update.

From the rise of the Internet to the broad adoption of social media, software companies have innovated in their quality feedback programs, supporting automated diagnostic data, very low-latency, deep product instrumentation, customer-centric validation programs, and both internal and external flighting rings. However, though each technique covers a different portion of the product and customer space, gaps remained to be filled.

In the era of social media, new channels have emerged as low-latency methods of communication from its users, allowing software companies to build on their prior software quality processes. Rather than submitting support tickets, users have now become accustomed to broadcasting their complaints on Twitter, enabling software issues to be "discovered" almost immediately. In addition, a greater population of users can voice their feedback, enriching the diversity of responses and discovered issues.

Company A funnels many tweeted messages per day about its software releases to skilled developers who read through them to identify regressions and submit bug reports. However, while thorough, manually combing through so many tweets is slow, tedious, and requires highly-skilled personnel to really understand whether a tweet is describing a problem with a released update.

Our goal was to help their triage team perform the triage task more effectively and efficiently. If we succeeded, they would either spend less time reading through irrelevant tweets, enabling them to focus on their other software development tasks, or they could spend the same amount of time triaging tweets, but find more of the issues distributed within. Recent work suggests this is doable. In 2017, Guzman *et al.* mined Twitter to look for software issues and found more than 10 million tweets about software applications [3]. While there was a lot of noise in the Twitter stream, they could find a lot of relevant and useful information using machine learning. This past year, Nayebi *et al.* compared tweets against mobile

app store reviews, and found that after eliminating duplicate, irrelevant, and spam tweets (42% of the total number), people tweeted 22% more feature requests and almost 13% more issue reports than they wrote about in their app reviews [4].

In this study, we explored the possibility of applying machine learning to reduce the cost and effort of triaging tweets about Company A's software updates. Here, we report on our investigation into the following questions:

- RQ_1: How can we identify tweets that are specifically about problems with released updates?
- RQ_2: How can we reduce effort by effectively filtering irrelevant tweets?

We found that it was possible to build a classifier to categorize tweets into four kinds of problems relevant to the triage team, but only 20% of problems could be identified with acceptable precision. In other words, the false positive rate was too high. However, we enjoyed much greater success in improving productivity by filtering irrelevant tweets. Our classifier achieved 98% precision in identifying these irrelevant tweets, which enables Company A to filter 70% of the tweets that the triagers needed to read through. In both cases, we found that logistic regression produced better classifiers than those using support vector machines (SVM). Finally, we integrated the tweet filtering classifier into the web-based classification system used by the triage team.

In this paper, we describe the following contributions:

1) A machine learning classifier that can effectively filter irrelevant tweets, lessening the workload on the triage team.
2) An enhanced web-based system for improving manual triage of tweets, shrinking the team's workload by 70%.

II. Prior Work

Many organizations can take advantage of social media to better engage with their customers. Users have many options to communicate their feelings and assessment of products they use, for example in online forums, product review sites, blogs, and Twitter. In order to make it simpler for organizations to identify important feedback, engineers make use of natural language processing (NLP) algorithms such as topic extraction, text summarization, and sentiment analysis to identify issues and summarize user feedback.

One avenue of literature focuses on the mechanics of extracting user feedback from online sources. For example, Dave *et al.* [5] applied sentiment analysis to semantically classify product reviews. Turney applied an unsupervised algorithm that applied a part-of-speech tagger to identify phrases in reviews as positive or negative [6]. Hu and Wu built on the field's body of work to create a summary of all customer reviews of a product into a list of pros and cons [7]. Zhan *et al.* investigated techniques to extract user complaints about products by summarizing online product reviews [8]. They apply topic analyses to extract salient subjects from reviews, rank them, and then use these as seeds for summarization. For those interested in more details of mining user attitudes, feedback, and sentiment, see Pang and Lee's survey paper and Liu's more recent book on opinion mining and sentiment analysis [9], [10]. We apply many of these standard techniques in our own extraction of salient tweets reporting user complaints about recently released software updates.

Another fork of the literature focuses on extracting specific subsets of user feedback that are relevant for software engineers and product designers. Hedegaard and Simonsen mine online product reviews of software for specific content that related to usability information, in order to help designers learn which aspects of the product's UI are most impactful to its use [11]. Anam and Yeasin looked at app reviews to identify accessibility issues [12], while Zhu and Fang used a lexical analysis approach to identify the positive and negative attributes users ascribed to video games that they reviewed [13]. Iacob and Harrison mined feature requests from product reviews of mobile software applications using pattern matching with pre-defined linguistic rules [14]. More recently, Jin *et al.* were able to connect their summaries of user dissatisfaction in product reviews directly to product features to help designers improve the product [15].

In our work, rather than look at product reviews, which may not appear for weeks or months after a release, we look at tweets, which often appear almost instantly. Though there are bots that monitor software update sites for activity and tweet about new releases, other bots retweet the information to their followers. However, these "informational" tweets are not useful to identify complaints. The most useful tweets arrive only after early adopters have had a chance to try out the software and write about any ill effects. In our data, we find that regressions are often described by users with words such as "crash," "brick," "doesn't work," etc. These descriptions are similar to problems found in app reviews.

While tweets offer the opportunity for rapid feedback, they also come with a low signal to noise ratio. Since this was Company A's first mining effort, we looked only at tweets tagged with multi-digit numbers from their public customer support document database to make it easier to filter out noise. When He *et al.* mined tweets related to disaster responses, they found that many sub-topics all using the same hashtag. However, very few were relevant in providing logistical assistance to victims [16]. Their strong need to assess relevance is similar to ours, as there was a lot of noise in their data. He *et al.* used the number of retweets as a relevance signal to more precisely identify the signal in this noise. When Anchiêta and Moura summarized product reviews [17] from the Google and Apple app stores, they had to first filter the reviews with an unsupervised learning approach because very few contained useful information. Chen *et al.* discovered the same low information problem in app review mining, requiring that they filter noisy and irrelevant reviews prior to clustering reviews by topic [18]. Fortunately after filtering, they found that as many as 35% of reviews were relevant. After applying our own filtering techniques (described in the next Section), we found a similar fraction of our Twitter dataset to be relevant.

Once filtered, relevant tweets have to be classified by topic

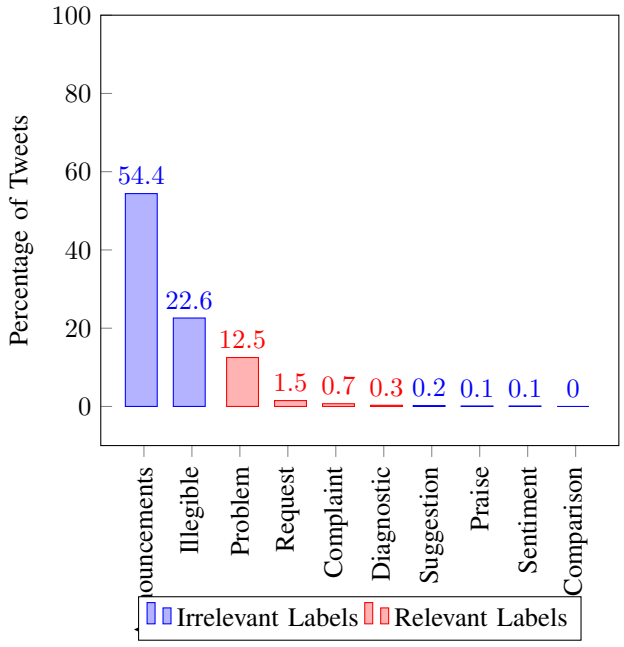

Fig. 1. The distribution of categories in our dataset.

into the type of problem the users describe. Maalej and Nabil classified app store reviews into four categories: feature request, bug report, user experience, and other [19]. They found that the use of multiple binary classifiers performed better than a single multiclass classifier. Our raters classify tweets into nine categories, shown in Figure 1. Meng *et al.* mined and grouped tweets by topic using hashtags [20]. They then summarized the tweets into "insights" using a paraphrase-based pattern-matching algorithm with a sentiment polarity analysis. We found it difficult to create pre-defined linguistic rules to support pattern matching techniques because we do not know what kinds of specific problems users will report. Often, their complaints are often about computing in general.

Classification is actually easier with tweets than with product reviews because they are much shorter and less grammatical. In fact, the lexically-based bag-of-words and n-gram approaches work well enough to extract the problems reported by users. But their short length can often make them difficult to understand in isolation. Othman *et al.* mined tweets for opinion summarization using threaded conversations of tweets rather than individual ones [21]. Once tweeted conversations were aggregated, topic analysis and sentiment analyses were performed, demonstrating improved identification of relevant tweets that were difficult to understand in isolation, but which made more sense in context. Unfortunately, most of our tweets do not have any conversation threads.

Simply filtering and classifying tweets is only part of an end-to-end solution to help organizations improve their products. Di Sorbo *et al.* mined product reviews of mobile applications to look for feedback on feature requests, problems, information seeking, and information giving. However, not only do they classify the reviews into categories, they also summarize them through topic analysis and clustering into user intentions in an interactive tool used by application developers [22]. They evaluated their approach in a user study with 23 app developers who judged the summaries on their own apps' reviews. We integrated our filtering and classification algorithms into Company A's internal regression triage tool, enabling any engineer to more quickly focus in on relevant tweets that signal possible regressions in their software updates.

III. Business Case

At Company A, the team responsible for discovering regressions has two goals:

1) Find as many issues as they can.
2) Find issues as efficiently as possible, i.e. minimize the effort and cost of finding issues.

In this section, we examine how these goals were accomplished in the past, and how social media has changed Company A's approach to regression discovery.

A. Pre-Social Media

In the past, Company A applied a manual approach to discovering regressions in software updates. Prior to release, test engineers would apply a variety of manual and automated techniques to ensure that the functionality of the software update worked as intended. Though fairly effective at finding issues, in-house testing cannot possibly find every regression, especially given the enormous number of possible novel combinations of hardware and software configurations at customer sites. Thus, as a second line of defense, before any customer received it, Company A employees would try it out on their own computers and report issues directly to support personnel on the development team, in a common industry practice known as "dogfooding."In a third line of defense, Company A asks a selection of its larger customers to test out upcoming security patches prior to general release, to ensure no compatibility is broken with critical software.

Once the update was released to the world, external users could report problems by calling or emailing customer support, or by engaging with account managers assigned to provide a first line of contact with the company. If customer support thought the problem was caused by a bug in the software code, they would collect as much evidence as they could gather into an actionable report for the developers and forward it to the appropriate software team. These reports could be very detailed and provided evidence of real customer impact, two attributes of bugs known to be highly prioritized by software teams [23].

Relying so heavily on customer support also has a downside. Company A will not learn about some issues right away because customers experiencing the problem might spend their efforts investigating workarounds. Finding your own workaround addresses the issue immediately, whereas waiting for Company A to identify the underlying issue, fix it, and release an update might take a lot longer. Company A addressed this shortcoming by integrating diagnostic data

gathering software deeply into their software, enabling crash-reporting software to respond to detected crashes and error conditions and send debug information back to the developer.

To anticipate customer execution conditions more accurately, Company A shifted much of its testing strategy away from automated synthetic testing to diagnostic data-based testing. Tests are based on customer conditions as reported by diagnostic data and help developers "see" more clearly the true state of customer computers rather than relying on the simplified description taken during bug reporting through customer support.

B. Post-Social Media

Social media has altered the way that Company A searches for and patches regressions in software updates. Using social media, user feedback can be directly integrated into the issue-finding process. Company A monitors a vast array of customer discussion media, including Twitter, various blogs, and user-initiated feedback using a tool integrated into their software. Of the sources of customer feedback, Twitter is particularly important because it is high volume, widely available (especially to home customers), and is a low-latency feedback channel, making software companies aware of problems more quickly [24], [25]. In addition, it is a vital source of early warning information for updates installed by customers running older versions of the software.

As an example, one user posted *"So my tool randomly started activating over and over... Company A's software also updated?"* [1] This post is an excellent early warning that a recent update could have caused a problem with the user's tool.

Twitter's ecosystem is vast (over 500 million tweets per day,[2]), making it impractical to examine every tweet to extract regressions. In addition, Twitter is noisy. Not only do Twitter users provide relevant technical feedback, but they also make announcements (i.e., simply stating that Company A has done something noteworthy like release a new version of software), and create and spread spam. Announcements are particularly problematic as they often mimic meaningful feedback without providing information about issues. For example, tweets often make statements such as *"Company A releases new update to fix new tool bug"*. Such a tweet looks at first glance to be a bug report do to the presence of the phrase "new tool bug." However, it actually conveys no useful information, as the reported issue has already been fixed.

In order to improve their signal-to-noise ratio, Company A uses automated text-based, regular expression filtering techniques to extract tweets that are most likely to be useful. Specifically, we look for a Company A customer support-specific signal: a number having the form *9999999*. Each number uniquely identifies a service release, indicating that a given tweet is highly likely to discuss an update. The presence of a number indicates a strong likelihood that a user is discussing an update, and therefore may be reporting a regression. The resulting set of filtered tweets is very large[3]. Each must be read by hand to ascertain what it is saying in order to create actionable bug reports for the software developers. Our work to apply machine learning is motivated by the sheer amount of manual effort required to discover the useful tweets.

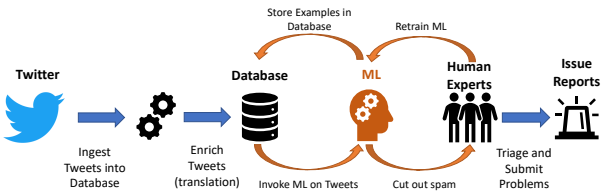

Fig. 2. An illustration of the triage process, including the ML service. Tweets are ingested, where they are translated and enriched with custom metadata. Then, the enriched tweets are stored in a database where they are labeled by the ML system and presented to triagers. Triagers mark confirm or deny these labels, allowing the ML system to be retrained. Finally, bug reports are assembled from relevant tweets.

C. The Triage Task

Company A wants to identify problems with updates as quickly as possible. To that end, an internal team is employed to continuously monitor regressions. Every week, one team member is assigned to comb through social media from a variety of sources and manually identify problems with new releases, a task that takes on average, three to four hours per day. These triagers are skilled software developers and project managers with high domain knowledge. Therefore, the effort spent reading social posts is fairly expensive.

Triagers utilize a web-based tool which present them with a list of relevant posts from social media threads. This tool displays the tweet text, along with various enrichments such as translation into English, sentiment analysis, and the source of the data (Twitter, a web site, etc.). The user is allowed to sort and filter posts by source, as well as organize similar posts under a common *parent*. The parent is a label which is used to group posts together, associating them with a specific issue. The triagers carefully consider the content and context of posts, manually searching through the list to identify relevant posts and hide irrelevant ones. The triage process is illustrated in Figure 2.

Since tweets typically contain very short messages, they often lack sufficient detail to construct a bug report. Fortunately however, serious issues will often be discussed in multiple tweets. To capture this information, triagers identify some important aspect of the issue, such as a diagnostic message or important keyword. These aspects are then used to group relevant, related tweets together. For example, several customer complaints about poorly performing graphics or drawing might be grouped under a phrase like *"graphics"*

[1]All tweets have been minimally altered to preserve anonymity and conform to double-blind submission.

[2]https://blog.twitter.com/engineering/en_us/a/2013/new-tweets-per-second-record-and-how.html

[3]scale anonymized for double-blind submission

TABLE I
OUR TWITTER CATEGORY LABELS. TWEETS HAVE BEEN MINIMALLY ALTERED TO PRESERVE ANONYMITY.

Category	Description	Example
Problem	Posts expressing a problem, particularly one occurring as a result of a software update.	*Feature failure after updating software? Blame it on the 9999999 softzone patch.*
Complaint	Generic complaints which do not discuss specific problems or contain reproducible issues.	*Oh, I've been yelling since the day before: the software updater update module 9999999 is bad*
Help Request	Posts asking for help or information without expressing that the request is a result of a problem.	*9999999 how do you install this*
Diagnostic	Posts including error codes, error message text, or other diagnostic information obtained after a fault, but not explaining or giving reproduction steps.	*Update 9999999 giving error code 0x56fe.*
Suggestion	Posts requesting a new feature or change to functionality.	*Integrations should use local authentication to escape delays by the password validation: #9999999*
Praise	Posts expressing positive sentiment towards a software product.	*9999999 I am pleased!*
Comparison	Posts in which a customer compares products or releases.	*Slower as compared not only to pre-9999999, but also to pre-AAA.*
Announcements	Posts which do not match any of the labels above, typically purely informational in content (e.g., news items about new releases)	*Users please note: To avoid vulnerabilities, Company A is releasing security updates 9999999. Please install ASAP!*
Illegible	Tweets which appear to lack any informational content.	*Posts 9999999 12-22:5 (Out RT: 1)*

After spending time triangulating the problem with other social media sources, crash logs, and Company A's customer support database, the triager builds up enough information to submit an official bug report, with specific details included from the relevant posts.

Given the size of the workload, noise can often be overwhelming. Engineers assigned to triage duty informed us that at first glance, informational posts can appear to be useful. For example, a typical news tweet might state: *"Fix for graphics, fixes flickering, in new patch."* Words such as "fix", "flickering", and "patch" make the post seem like a bug report. It is not until the triager carefully reads the post that it becomes clear that it is merely giving information about a prior update. The triagers made it clear to us to make a substantial impact in streamlining their process, they wanted better tooling to identify relevant tweets and filter out noise.

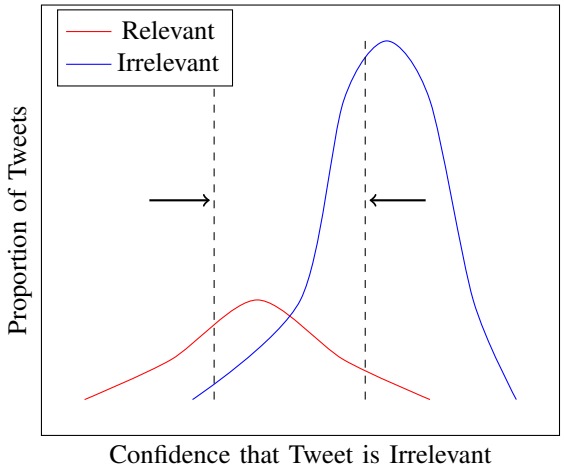

Fig. 3. An illustration of the two studies' approaches to classifying the same dataset. As our confidence that a tweet is irrelevant increases, we see that there are a number of tweets which can be confused. Approaching from the left means extracting as many relevant posts as possible before too many false positives are obtained. Approaching from the right means detecting high confidence irrelevant tweets, and removing them before triagers see them.

IV. STUDIES

In this section, we present two studies that demonstrate the feasibility of solving the goals introduced in Section III. We also introduce the algorithms, dataset, and labels that are common to both.

We represent both business cases as supervised classification tasks:

1) **Business Case 1**: Classify *problem* tweets in a set of tweets, with the goal of quickly identifying as many problem regressions as possible.
2) **Business Case 2**: Classify and cull irrelevant tweets to reduce triage effort.

A. The Dataset

Supervised classification algorithms rely on having pre-labeled data to build a statistical model. The team responsible for machine learning in the organization compiled a dataset using a crowd-sourcing service , in which people are asked to solve human intelligence tasks (HITs) such as simple classification and extraction problems. Workers on the platform were asked to read a thousand tweets of user feedback and classify it under one or more categories. In an effort to make use of a single process to categorize several sources using the same taxonomy, the team defined a set of labels which we list in Table I. These labels are used in common across many different forms of media, and were designed originally for user initiated feedback (internal feedback application native to Company A's software). *Problem*, *Complaint*, *Help Request*, and *Diagnostic* are mutually exclusive: only the most likely

category is selected. *Suggestion* and *Praise* are orthogonal; it is possible for a post to express both a suggestion and praise. Announcements are essentially spam or other irrelevant tweets, mutually exclusive with other categories. *Illegible* is also mutually exclusive with every other category: it is only selected if no other label applies. Finally, *Comparisons* between Company A's products are almost non-existent in our dataset: the judges found only one instance. Tweets can be written in any language, and are translated to English by another team before being sent for judgment. Given the importance of this project, the workers on the crowd-sourcing platform received coaching and additional oversight as they learned the categories over the course of their first week.

As we worried, announcements made up the majority of the tweets in the crowd-source-labeled training set (see Figure 1). These noisy tweets increase the difficulty of discovering useful technical feedback from Twitter. We then examined the other non-announcement labeled instances using the taxonomy. We determined that *problems*, *requests*, *complaints*, and *diagnostics* were all of interest to discovering regressions, and thereby highly likely to be of interest in accomplishing Business Case 1. Tweets from the remaining categories were irrelevant, thus, their removal would accomplish Business Case 2. Figure 3 illustrates the difference between the two approaches. For Business Case 1, we attempt to extract as many relevant tweets as possible before false positives become too problematic (the left dotted line). Both approaches are complimentary, but it is not the case that feasibility of one implies feasibility of the other. For example, the dataset could contain large numbers of irrelevant data with common text patterns, which would be easy to remove. However, the highly relevant posts contained in the same dataset could be similar to irrelevant posts, making their extraction more difficult.

In an effort to avoid judgment bias in the dataset, majority voting was used to determine the relevance of a given tweet. Specifically, if at least three of the labels assigned to a given tweet were relevant, the tweet was considered relevant as a whole. Through this synthetic labeling, we could make progress towards Business Cases 1 and 2 through *binary classification*.

B. Algorithms and Tools

To investigate the research questions in line with the business cases above, we evaluated machine learning algorithms for their ability to classify relevant and irrelevant tweets. In the literature, two algorithms are commonly found to yield effective machine learning classifiers for the related problem of issue detection in tweets:

- **Support Vector Machines (SVMs)**: Binary classifiers based on treating instances as members of a vector space. The SVM model is essentially a hyperplane that splits the set of instances into two classes. This hyperplane is determined by an quadratic programming process which determines the plane with the greatest distance between differently classed vectors. In maximizing this distance, SVMs become resistant to overfitting, the undesirable property of overemphasizing non-predictive differences in the data. Classification of a particular instance is performed by determining which side of the hyperplane it falls on. SVMs can be linear or non-linear. We applied a linear model, commonly used in text classification, which benefits from having few hyper-parameters.
- **Logistic Regression Classifier (LR)**: is an algorithm which attempts to determine the parameters of a logistic model used to yield the log-odds of an instance belonging to either class. The parameters are initialized randomly and learned through expectation maximization, an iterative process of evaluation and retraining. Like SVMs, LR classifiers are highly effective in classifying tweets.

These algorithms operate on vector data, thereby requiring us to encode tweets into a numerical format. We do this using the bag-of-skip-grams representation. Essentially, each tweet is expressed as a vector, and each word is represented as a separate vector attribute. A value of 1 for an attribute indicates that a particular word is present, while a 0 indicates absence. For example, if the total vocabulary of the dataset is $< software, update, successful, unsuccessful >$, the tweet *"The update was unsuccessful"* would be encoded as $< 0, 1, 0, 1 >$, due to the terms "update" and "unsuccessful" being present. N-Grams extend this approach by allowing common pairs (bi-grams), and triples (tri-grams) of adjacent words to be considered as a single term. For example, "successful update" would be one of the vocabulary features if we included bi-grams. Skip-grams further augment N-Grams by allowing non-adjacent words that occur in the same instance to be represented, such was "update successful" in the sentence "the update was successful". We apply all of the above by using a 3-skip-3-gram representation of tweets, thereby encoding all common words, pairs and triples of words, and as pairs and triples of non-adjacent words with at most three terms in between. This representation is sufficiently rich to capture many useful idioms such as "crashes after update" and "update fails", which are highly informative for identifying regressions. Vocabulary items that only occur a small number of times have a very small effect on the overall performance of the classifier, therefore we eliminate all skip-grams with fewer than 5 occurrences.

Numerous machine learning frameworks provide implementations for these algorithms and data encodings, as well as providing a means of deployment. The algorithms themselves are well known, and a particular algorithm will produce virtually identical results regardless of which platform it runs on. However, deployment mechanisms and ergonomics vary greatly across frameworks. In our project, we wanted a framework which was easy to deploy and had a low barrier-to-entry, in order to minimize the maintenance burden imposed by the system. We used Microsoft's AzureML Studio to implement our algorithms and deploy the completed models into the cloud.

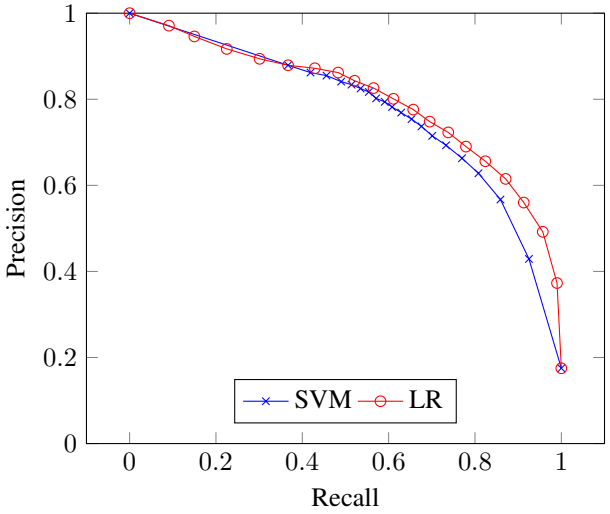

Fig. 4. The precision-recall curve for LR and SVM classifiers for classifying *relevant* tweets.

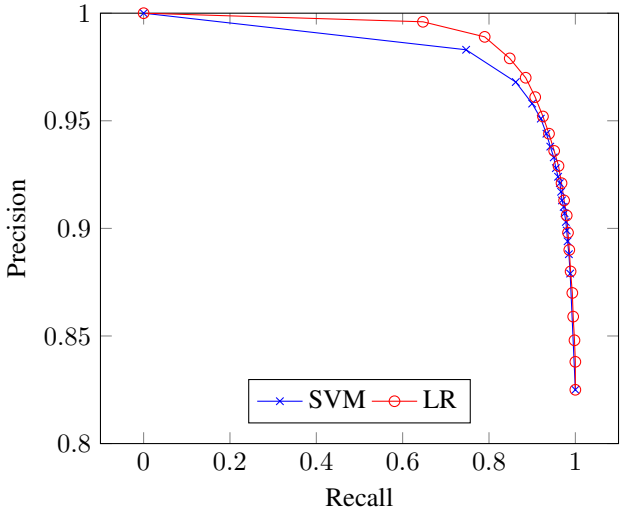

Fig. 5. The precision-recall curve for LR and SVM classifiers when classifying *irrelevant* tweets.

C. Evaluation Method

We evaluate potential classifiers using the precision and recall metrics. Precision is defined as $P = TP/(TP + FP)$, where TP refers to true positives, and FP refers to false positives. Precision measures how often a classifier is correct when it asserts a particular label—in our case, that a tweet is relevant. Recall, defined as $R = TP/(TP + FN)$, where FN refers to false negatives, is a measure of coverage. In our case, recall refers to the proportion of relevant tweets that were correctly labeled by the classifier. Precision and recall are evaluated at a particular threshold, which is a confidence level over which a particular tweet is taken to be relevant. For example, a classifier may evaluate a particular tweet as having a 55% probability of being relevant. If our threshold is taken to be 50%, the tweet is assigned the label of relevant. On the other hand, if 60% is chosen, the tweet is considered irrelevant. By varying the threshold in small increments from 0 to 100%, we can generate a curve representing the overall effectiveness of the classifier.

Evaluation of machine learning classifiers is performed by holding a proportion of judged instances in a training set to build the model, and using the remaining instances to evaluate the model. Using only a single training set may cause excessive sampling variance. To reduce variance, we partition the total data set into N distinct subsets (folds), with one fold being used for evaluation and the remainder for training. Then, each set of N partitions can be evaluated, with the average precision and recall being used to appraise the overall performance of the model. In our analysis, we take $N = 10$, which is common in NLP applications.

D. Results: Classifying Relevant Tweets

In our first business case, we classify tweets into one of ten categories, only some of which are relevant for the team. Relevance was defined as categorizing a tweet into one of four categories: *problems*, *requests*, *complaints*, or *diagnostics*. Figure 4 portrays the 10-fold cross-validated precision-recall curves of the SVM and LR classifiers we trained to do this task. First, observe that the overall performance of the two classifiers are similar, with LR achieving higher precision at very low and high recalls, and SVM performing better over most of the interval. Second, it would be ideal if it were possible to classify a large majority of problems correctly. However, we see that neither classifier achieves sufficient precision ($\geq 90\%$) to adequately automate the triage expert's job. The preliminary results indicate that only 20% of problems could be correctly identified with acceptable precision. In order to extract a majority of problems (i.e. $\geq 50\%$), a false positive rate of 15% is expected. Such a false positive rate is unacceptable: misidentifying a relevant tweet as irrelevant is more harmful than the converse. In the former scenario, the triager may miss critical regressions, while in the latter, they must simply spend more time to ignore noise. Therefore, Business Case 1, the ability to directly extract useful tweets, turns out to be infeasible.

E. Results: Filtering Irrelevant Tweets

In our second business case, we wish to filter irrelevant tweets (i.e. noise) from the dataset. Figure 5 expresses the 10-fold cross-validated precision-recall curve for the two classifiers on the task of identifying noise. As with our first classification task, the LR classifier slightly outperformed the SVM, although the performance overall was similar. One important requirement for Business Case 2 was that precision should be no worse than 98%. A 98% precision on irrelevant tweets implies a 2% false positive ratio, meaning meaningful tweets wrongfully classified as noisy. Given the imbalanced ratio of relevant to irrelevant tweets in the dataset, this means 10% of relevant tweets would be misclassified. After interviews with triagers from the team and the team manager (the fourth

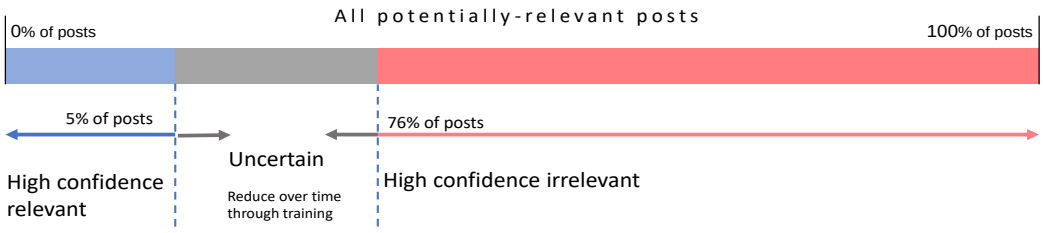

Fig. 6. A representation of the two-ended approach to tweet classification. We attempt to classify both high-likelihood relevant and irrelevant posts. Posts with uncertain relevance require human analysis.

author), we discovered that it was rare to find an issue reported by only a single user. In fact, virtually every regression observed up to this point had more than five different users describe the issue in their own words. Therefore, we felt a 10% cutoff rate (i.e., 98% precision) would be an acceptable limit to achieve Business Case 2.

F. Deployment

As mentioned in Section IV-B, our machine learning classifier was intended to be deployed via Microsoft's AzureML Studio platform. However, this was only part of the deployment story. The system would have to be introduced to the triage team for evaluation.

Our evaluation social triage tool pulls posts from a central database and acts as a staging area for all social media content. An internal loader runs every hour to pull new content, process its metadata, and insert the information into the database. We modified this loader to invoke our machine learning system, using the Azure ML Studio RESTful API. Each tweet is first labeled as relevant/irrelevant using the ML classifier and then stored back in the database. The modified triaging tool presents this information in a new column next to the text of the tweet. It displays the tweet's irrelevance likelihood, using the 90% confidence level discussed in Business Case 2. This new column is filterable, allowing triage experts to hide noisy posts. Despite our finding that it is infeasible to capture enough relevant tweets in Business Case 1, we allow the column to also display a "likely problem" category label (confidence greater than 50%), in order to allow triagers to quickly establish the most common issues in the dataset. Unlike the noise value, triagers are not expected to use this category to filter tweets, as too many useful problems would be filtered out. Posts in between the confidence interval of "likely problem" and "irrelevant" are marked as "unclassified," to be determined by human experts. As expressed in Figure 6, 5% of posts are expected to be identified as a "likely problem", while 76% of posts are expected to be identified as "irrelevant".

One of the most important features of any ML system is the ability to be retrained. The modified social triage tool allows the user to change the relevant/irrelevant/unclassified judgment using a dropdown list. When the user indicates that they have finished triaging a particular post, the application saves the final label to a training database, to be used to later to periodically retrain the classifier. Under this process, it is possible for the machine learning classifier to more closely reflect the knowledge of the triagers over time.

G. Outcome and Impact

During the development and deployment process, we asked triagers to describe the expected impact of our ML system on their workflow. One triager told us that, on average, it took one minute to triage each tweet. This figure included time spent reading the tweet, determining whether it described a recent regression or was merely an announcement, and correcting misclassifications. The triagers told us that the changes would substantially reduce the number of person-hours required to classify tweets. In addition, triagers indicated that culling irrelevant tweets would allow them to focus more on those with value, preventing posts from getting "lost in the noise."

While the ML system will save effort, individual triagers still want more help to identify regressions. In particular, once a social post has been identified as relevant, they still have to generate a bug report and route it to the correct team. They would appreciate additional automation to help improve the workflow for these tasks. Currently, we are investigating techniques to automatically cluster and label relevant posts (especially across different social media channels), to automatically associate posts with non-social data (such as internal bug reports), and to bring in additional data sources into the machine learning-based approach.

V. Summary and Conclusion

In this paper, we have presented an approach to leverage machine learning to partially automate the work of triaging customer complaints about software regressions found on social media. Though we found it is not yet feasible to classify tweets into appropriate complaint categories for triage, it *was* possible to train a logistic regression classifier to filter a large fraction of irrelevant tweets. We have evaluated our classifier by integrating it into the tools used by triage experts, receiving favorable feedback, and enabling triage experts to more efficiently monitor and respond to the incoming firehose of tweets. Utilizing machine learning to categorize user feedback has the potential to help all kinds of software engineers in their tasks. We also encourage other organizations to look for opportunities to improve their engineers' efficiency and effectiveness with machine learning.

REFERENCES

[1] J. Czerwonka, R. Das, N. Nagappan, A. Tarvo, and A. Teterev, "Crane: Failure prediction, change analysis and test prioritization in practice – experiences from windows," in *Proceedings of the 2011 Fourth IEEE International Conference on Software Testing, Verification and Validation*, Washington, DC, USA, 2011, pp. 357–366.

[2] K. Herzig, M. Greiler, J. Czerwonka, and B. Murphy, "The art of testing less without sacrificing quality," in *Proceedings of the 37th International Conference on Software Engineering - Volume 1*, Piscataway, NJ, USA, 2015, pp. 483–493.

[3] E. Guzman, R. Alkadhi, and N. Seyff, "An exploratory study of twitter messages about software applications," *Requirements Engineering*, vol. 22, no. 3, pp. 387–412, Sep. 2017.

[4] M. Nayebi, H. Cho, and G. Ruhe, "App store mining is not enough for app improvement," *Empirical Software Engineering*, vol. 23, no. 5, February 2018.

[5] K. Dave, S. Lawrence, and D. M. Pennock, "Mining the peanut gallery: Opinion extraction and semantic classification of product reviews," in *Proceedings of the 12th International Conference on World Wide Web*, New York, NY, USA, 2003, pp. 519–528.

[6] P. D. Turney, "Thumbs up or thumbs down? semantic orientation applied to unsupervised classification of reviews," *Computing Research Repository*, vol. cs.LG/0212032, pp. 417–424, 12 2002. [Online]. Available: http://arxiv.org/abs/cs.LG/0212032

[7] X. Hu and B. Wu, "Classification and summarization of pros and cons for customer reviews," in *Proceedings of the 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology - Volume 03*, 2009, pp. 73–76.

[8] J. Zhan, H. T. Loh, and Y. Liu, "Gather customer concerns from online product reviews – a text summarization approach," *Expert Systems with Applications*, vol. 36, no. 2, Part 1, pp. 2107 – 2115, 2009. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S095741740700663X

[9] B. Pang and L. Lee, "Opinion mining and sentiment analysis," *Found. Trends Inf. Retr.*, vol. 2, no. 1-2, pp. 1–135, Jan. 2008. [Online]. Available: http://dx.doi.org/10.1561/1500000011

[10] B. Liu, *Sentiment Analysis and Opinion Mining*. Morgan & Claypool Publishers, 2012.

[11] S. Hedegaard and J. G. Simonsen, "Extracting usability and user experience information from online user reviews," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. New York, NY, USA: ACM, 2013, pp. 2089–2098.

[12] A. I. Anam and M. Yeasin, "Accessibility in smartphone applications: What do we learn from reviews?" in *Proceedings of the 15th International ACM SIGACCESS Conference on Computers and Accessibility*. New York, NY, USA: ACM, 2013, pp. 35:1–35:2.

[13] M. Zhu and X. Fang, "What nouns and adjectives in online game reviews can tell us about player experience?" in *CHI '14 Extended Abstracts on Human Factors in Computing Systems*. New York, NY, USA: ACM, 2014, pp. 1471–1476.

[14] C. Iacob and R. Harrison, "Retrieving and analyzing mobile apps feature requests from online reviews," in *Proceedings of the 10th Working Conference on Mining Software Repositories*. Piscataway, NJ, USA: IEEE Press, 2013, pp. 41–44.

[15] J. Jin, P. Ji, and C. Kwong, "What makes consumers unsatisfied with your products?" *Eng. Appl. Artif. Intell.*, vol. 47, no. C, pp. 38–48, Jan. 2016.

[16] X. He, D. Lu, D. Margolin, M. Wang, S. E. Idrissi, and Y.-R. Lin, "The signals and noise: Actionable information in improvised social media channels during a disaster," in *Proceedings of the 2017 ACM on Web Science Conference*. New York, NY, USA: ACM, 2017, pp. 33–42.

[17] R. T. Anchiêta and R. S. Moura, "Exploring unsupervised learning towards extractive summarization of user reviews," in *Proceedings of the 23rd Brazillian Symposium on Multimedia and the Web*. New York, NY, USA: ACM, 2017, pp. 217–220.

[18] N. Chen, J. Lin, S. C. H. Hoi, X. Xiao, and B. Zhang, "Ar-miner: Mining informative reviews for developers from mobile app marketplace," in *Proceedings of the 36th International Conference on Software Engineering*. New York, NY, USA: ACM, 2014, pp. 767–778.

[19] W. Maalej and H. Nabil, "Bug report, feature request, or simply praise? on automatically classifying app reviews," in *2015 IEEE 23rd International Requirements Engineering Conference (RE)*, vol. 00, Aug. 2015, pp. 116–125. [Online]. Available: doi.ieeecomputersociety.org/10.1109/RE.2015.7320414

[20] X. Meng, F. Wei, X. Liu, M. Zhou, S. Li, and H. Wang, "Entity-centric topic-oriented opinion summarization in twitter," in *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA: ACM, 2012, pp. 379–387.

[21] R. Othman, R. Belkaroui, and R. Faiz, "Customer opinion summarization based on twitter conversations," in *Proceedings of the 6th International Conference on Web Intelligence, Mining and Semantics*. New York, NY, USA: ACM, 2016, pp. 4:1–4:10.

[22] 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 *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. New York, NY, USA: ACM, 2016, pp. 499–510.

[23] T. Zimmermann, R. Premraj, N. Bettenburg, S. Just, A. Schröter, and C. Weiss, "What makes a good bug report?" *IEEE Transactions on Software Engineering*, vol. 36, no. 5, pp. 618–643, September 2010.

[24] G. Williams and A. Mahmoud, "Mining twitter feeds for software user requirements," in *2017 IEEE 25th International Requirements Engineering Conference (RE)*. IEEE, 2017, pp. 1–10.

[25] E. Guzman, R. Alkadhi, and N. Seyff, "A needle in a haystack: What do twitter users say about software?" in *Requirements Engineering Conference (RE), 2016 IEEE 24th International*. IEEE, 2016, pp. 96–105.