

Cover Letter for Revised Manuscript

THE ACTUAL PAPER FOLLOWS THIS COVER LETTER.

PaperID : 654 (March 2013 track) or 915 (July 2013 revision track)

Title : Towards Social Data Platform:
Automatic Topic-focused Monitor for Twitter Stream

Venue : VLDB 2013

We thank the reviewers for insightful comments and meaningful suggestions. We have given careful consideration to the suggestions and improved the paper by addressing all the comments. Below we first list issues raised by each reviewer and corresponding solutions, and then summarize our major revisions.

Legend: We number the reviewer with the prefix “R”, the weak points with the prefix “W” and the detailed comments with “D”.

Reviewer 1

Comment W1 & D2: *The influence of the length of iteration is not investigated (also missing in evaluation). I would like to see some formal analysis since such a length evidently impacts not only the overall performance but also computational complexity.*

Response: Great suggestion. While ATM works for iterations with any lengths, the length of an iteration, denoted as l , is an *important* parameter. Following the suggestion, we add discussion to analyze the impacts of l and experiments to validate our analysis.

First, in Section 3 (after describing our framework), we analyze the impacts of l in terms of the *performance* of ATM (*i.e.*, the coverage of target tweets).

- On one hand, l should be long enough to collect sufficient samples, since we need sufficient samples to accurately estimate the usefulness of keywords and we collect more samples in a longer iteration. If l is too short, we may select useless keywords due to inaccurate estimation. Formally, since Theorem 4.1 shows how the estimation accuracy is related to the sample size, and we can obtain the number of samples collected in an iteration, we may measure the estimation accuracy for a given l or the minimum l for achieving a certain accuracy.

- On the other hand, l should be short enough to capture the dynamics of the Twitter stream, since we assume that tweets are similar within an iteration but the content of target tweets may change over time. If l is too long, we may use outdated keywords and miss many new target tweets. However, it is difficult to formally estimate how short l should be, because it requires accurately modeling the dynamics of the Twitter stream, which is an open research problem.

In addition, we explain that l should be different for different topics, since different topics require different numbers of samples for accurate estimation (*e.g.*, a sparse topic like crime requires a lot of samples), and their target tweets change at different rates (*e.g.*, tweets about olympics news develop very fast).

Thus, we treat l as an important parameter to tune for a given topic. As ATM works for any given l , we can find a reasonable l for a topic easily. Specifically, we can first test the performance of ATM with different l and then find the best one to use. Here, we clarify that in this paper we focus on the two essential problems (*i.e.*, tweet sampling and keyword selection) within each iteration, and we plan to study how to automatically set l for a topic as our future work. As we now discuss in Section 7, we aim to model the dynamics of the Twitter stream and automatically adjust l to update keywords.

Second, we explain the impacts of l on the computation complexity of our two algorithms. For the keyword selection algorithm, in Section 5 (complexity analysis), we now show that its complexity is linearly related to the size of a corpus. The longer l is, the more samples ATM collects. Thus, its complexity increases as l increases. However, as our existing experiments in Section 6.2.1 (efficiency) show, the selection algorithm is efficient to handle a very large corpus. For the sampling algorithm, in Section 4.2 (after our algorithm), we now explain that its efficiency only depends on how quickly we get samples with Twitter APIs. Thus, its efficiency is independent of l .

Third, we add experiments in Section 6.2.1 (iteration lengths) to evaluate the performance of our framework with different l . The results validate our analysis. If l is very short (*e.g.*, 30 mins), the performance is low, since we cannot get sufficient samples in short iterations. If l is very long (*e.g.*, 8 hours), the performance is low as well, because long iterations cannot capture the dynamics of the Twitter stream well.

Comment W2 & D1: 1) Authors described how filter API and sample API are selected with different probabilities, but how such API selection influences the overall performance and its complexity are not clearly explained in the evaluation. 2) In the evaluation section, no experiments are conducted to validate the tweet graph, *e.g.*, its impact to the overall performance and the incurred overheads, etc. (not just coverage). 3) Furthermore, simply deriving a probability to decide which Twitter API is used seems very straightforward and coarse-grained: will the effects of the two API overlap (you dont know how Twitter

answers an query, the tweets are returned randomly or follow certain distribution?)?

Response: Thanks for the meaningful comments. We provide additional explanation and evaluation for our algorithm in Section 4 and 6. Here, we first clarify that our algorithm is to provide additional random samples besides the samples returned by the *sample* API instead of replacing them. Thus, we interpret the “performance” as whether our algorithm collects *random* samples and the “complexity” (or efficiency) as how many *additional* samples our algorithm collects per hour besides what the *sample* API returns.

For the first comment, we add analysis and evaluation about the impacts of the selection of the two APIs (*i.e.*, the *sample* and *filter* APIs). Recall that, in our algorithm (Algorithm 2), the selection of the APIs is controlled by the parameter λ . *E.g.*, a larger λ means using the *sample* API more.

- In Section 4.2 (before Theorem 4.2), we add discussion about the impacts of λ . Here, λ is introduced to *theoretically* guarantee that our graph is ergodic and random walk converges. The λ value plays the same role as the teleport weight used in pagerank. Pagerank converges for any non-zero teleport weight. Similarly, our algorithm collects unbiased samples with different λ , since, as we show in Theorem 4.2, our random walk converges to known distributions with different λ , all of which can be adjusted to the uniform distribution. However, a large λ will lead our algorithm to use the *sample* API a lot and collect only a small set of *additional* samples with the *filter* API, which is not efficient in our setting. Thus, we should set λ to a small value.
- In Section 6.2.1 (sampling algorithm), we add experiments to evaluate 1) whether our algorithm collects random samples with different λ and 2) how many additional samples it collects with different λ . The results validate our analysis. Our algorithm collects random samples with different λ and collects less additional samples with a larger λ .

Following the second comment, we discuss and evaluate the performance and the overheads of our design. Here, we clarify that our sampling algorithm does not need to construct a complete tweet graph explicitly, and it only needs to “walk” from a tweet (sample) to another according to their edge weight in the graph with the APIs.

- For the “performance”, we clarify two points here. First, in Section 4.2, we formally prove that our algorithm collects unbiased samples. Second, in Section 6.2.1 (sampling algorithm), we empirically validate that our algorithm performs 1) similarly to a uniform sampler and 2) much better than a biased sampler.
- We then focus on the “overheads”. First, in Section 4.2 (after Algorithm 2), we add discussion to explain that our algorithm 1) costs insignificant CPU resources, as it only needs some easy-to-compute variables (*e.g.*, the set of keywords in a tweet), but 2) requires time to get samples with the APIs. Thus, we interpret the “overheads” as how

much time our algorithm takes to get an additional sample (*i.e.*, efficiency). Second, in Section 6.2.2, we add experiments to evaluate our algorithm’s efficiency on the Twitter stream. We compare our algorithm with the only available random sampling method on the Twitter stream (*i.e.*, the *sample* API), since our goal is to enable sampling additional tweets with a classical sampling framework instead of comparing different sampling frameworks. The results show that 1) running our algorithm in a single machine collects a large number (*i.e.*, 30K) of additional samples per hour, which speeds up the *sample* API by 1.4 times, and 2) while calling the *sample* API from different machines gives the same samples, running our algorithm in parallel scales up the efficiency (*i.e.*, two instances collect 1.96 times as many additional samples as one instance), as different instances choose different keywords randomly and collect different samples.

Next, we answer the third comment. Here, we clarify that, while it is intuitive to combine the two APIs to collect additional tweets, how to use them intelligently to collect additional *random* samples is *non-trivial*. Specifically, to get a random sample, we need to carefully design 1) how to decide which API to use, 2) how to choose a keyword for the *filter* API to get a sample, and 3) how to adjust the sample for uniform sampling. Our design combines the two APIs based on a probabilistic sampling framework (*i.e.*, random walk sampling and rejection sampling). Our design is *theoretically* guaranteed to collect samples uniformly. Our experiment (sampling algorithm) empirically validates this point.

Further, we clarify the concern about the overlap of the two APIs. Our algorithm assumes that the APIs work as what Twitter API documents describe. The *filter* API returns the tweets matching a keyword, and the *sample* API returns 1% *random* samples. It is possible that the two APIs overlap (*i.e.*, return the same sample by chance). However, our algorithm is guaranteed to collect random samples, as our probabilistic sampling framework allows getting the same sample with probabilities. Further, we add new experiments in Section 6.2.2 to show that our algorithm indeed collects many additional samples with the two APIs from the Twitter stream.

Comment D3 & W3: *Keywords selection problem is somehow similar to topic modeling problem. Such related work should be at least mentioned.*

Response: Following this great suggestion, we add discussion about topic models in our related work (Section 2). We explain that we cannot use standard topic models (*e.g.*, PLSA [9], LDA [5]) in our problem for two reasons. First, they are unsupervised probabilistic models for discovering hidden topics (a topic is a multinomial distribution of keywords) from a text corpus, so the discovered topics may not correspond to the specific topic of our interest (*e.g.*, crime). Second, they do not have the concept of “coverage” and “cost,” so they cannot find the optimal keywords that have the maximum coverage of target tweets for a topic under cost constraints. We note that some supervised probabilistic models can

find relevant keywords for a topic but cannot find the optimal keywords under cost constraints either due to the second reason. In our experiments, we compare our approach with one of them (*i.e.*, BH-2, which finds relevant keywords based on probabilistic language models) and demonstrate the advantages of our optimization based approach (*i.e.*, it has the best coverage among all the baselines and considers different cost constraints).

Reviewer 2

Comment W1, D1, & W3 *More technical details can be included for the tweet stream. The tweet stream is processed for each time interval, however evaluation or discussion on the effect of different lengths are missing. More evaluation is preferred.*

Response: Insightful comment. Following the comment, we add discussion and evaluation about the iteration length. As Reviewer 1 also mentioned this point, please see our response to Reviewer 1, W1 & D1 for details.

Comment W2 & D2: *A good running example and better case study are preferred. e.g, how are keywords evolving for time intervals?*

Response: Thanks for this great suggestion. We now add running examples when we explain our framework in Section 3. We also give a new case study to show how keywords are updated iteratively in Section 6.2.3.

D3: *My last concern is about the generation of K . Tweets stream come in continuously, K must be dynamic as well. Is the fixed K sufficient in the current implementation? How to update the K accordingly?*

Response: Thanks for this meaningful suggestion. We clarify three points about candidate keywords K here. First, our framework works for any given K (*i.e.*, it can take different K in each iteration). Second, in practice, our framework prefers a complete K (*i.e.*, it covers all the keywords in tweets), and thus K should be updated iteratively. Otherwise, it may miss useful keywords. Third, K can be updated. We can start from a general vocabulary, and in every iteration we add all new unigrams in the samples of that iteration. To make these points clear, 1) in Section 3 we add explanation when we define K , and 2) in Section 6.1 we discuss how we construct and update K in our experiments. We clarify that we update K every iteration in the Twitter stream setting.

Reviewer 3

Comment W(a) & D(a): *The motivation of designing new sampling algorithms needs more support? One intuitive strategy would be to spend more/sufficient time to collect tweets from this sample APIs. The authors need to provide some explanation or argument against the aforementioned method*

Response: Great comment. We provide additional arguments to motivate our algorithm.

In Section 4.1, we explain that our sampling algorithm is useful for ATM, because ATM works in short iterations and needs sufficient samples for each iteration. Specifically, we add discussion and evaluation in Section 3 and 6 to show that ATM works iteratively and prefers a short iteration to capture the dynamics of the Twitter stream (please refer our response to Reviewer 1, W1 & D2 for the details). Further, as Theorem 4.1 shows, to accurately estimate the usefulness of keywords, ATM needs sufficient samples for every iteration, which might be more than 1% of the tweets in an iteration. Thus, ATM needs our sampling algorithm, which provides additional samples besides the *sample* API, to collect sufficient samples in a short iteration or make an iteration shorter to better capture the dynamics of the stream. We clarify that, while we can take a long iteration to collect many tweets with the *sample* API, we may fail to update keywords timely and miss many target tweets.

In Section 3 and Section 7, we further emphasize that our sampling algorithm is useful for many other settings. While we can collect a lot of samples with the *sample* API in a long time period, the samples are still 1% of the tweets in that period. Many applications (*e.g.*, estimating some sparse prosperities of the Twitter stream in a given day) require collecting more than 1% of tweets and need our sampling algorithm to get additional samples.

Comment W(b) & D(b): *The assumption of given classifier seems to be too strong. 1) Firstly, it is a rather strict requirement. It is hard to train a good classifier. 2) Secondly, the performance of ATM may largely depend on the performance of such a classifier. A better user interface could allow user to specify keywords and enrich such keywords. 3) Finally, suppose a good classifier is indeed available. it remains a question whether such complicated framework is needed. For example, an intuitive idea would be to adopt a heuristic to obtain keywords with the classifier.*

Response: Thanks for the meaningful comments. To better motivate the use of a classifier in our framework, we add the following explanations in Section 3 (ATM overview).

For the first comment, we explain that our focus is to design a general platform to collect (potentially) relevant tweets for various social media based applications, which have

already applied classifiers to *automatically* determine whether a tweet is relevant to their applications. Thus, our framework only leverages the existing classifiers in those applications and does not add any extra burden. To provide additional supports, we give additional references to show that many classifiers have been developed for classifying tweets in those applications. Further, one of our existing experiments (*i.e.*, classification function) shows that our framework works for any given classifiers.

For the second comment, we agree that, as the reviewer suggested, users may prefer inputting keywords instead of a classifier in some scenarios. However, we clarify that 1) those scenarios are not our focus, since they are different from the social media based applications mentioned above, and 2) our framework still works well for such scenarios, because, when we use seed keywords to find target tweets, the keywords can be viewed as a simple rule-based classifier, which is a special case in our setting. Further, we argue that, for those scenarios, training a strong classifier using advanced models and novel features (*e.g.*, social signals) might be a better choice than using only keywords, since a strong classifier can predict target tweets more accurately than keywords. We can better support users to train classifiers in our framework as well. *E.g.*, we can assist users to define keywords and other novel features, select different models, and label tweets as training data.

For the third comment, we clarify that we need our ATM framework, as it can meet the two crucial requirements (*i.e.*, “optimal” and “continues”) motivated in Section 1. First, ATM takes an optimization approach, so it can select near-optimal keywords, which collect more target tweets than the keywords selected by heuristics. Second, ATM takes an iterative approach to update keywords, so it can monitor the dynamic Twitter stream continuously, which collects more target tweets than a static approach (*i.e.*, using a same set of manually or heuristically selected keywords). We also validate ATM’s advantages in our experiments. First, our existing experiments (ATM vs. baselines) compare our optimization approach with three heuristics, which all heuristically select keywords with a classifier or from the target tweets determined by the classifier (*e.g.*, baseline BH-1 uses a heuristic similar to how a Naive Bayes classifier determines its feature weights), and show that ATM selects better keywords than the heuristics. Further, we add a new experiment in Section 6.2.1 (iteration lengths) to compare ATM with static approaches, and show that it is beneficial to update keywords in iterations.

Comment W(c) & D(c): *The description of random walk sampling needs elaboration. The random walk sampling starts with a “burning”. The authors need to clarify it.*

Response: It is a great suggestion. We add explanation about “burning” in Section 4.2 to better describe our algorithm.

Comment W(d) & D(d): *More metrics could be used in the evaluation. In experiment the recall is used to evaluate the performance of monitoring. It is natural for the readers to ask what the "precision" is.*

Response: Thanks for the suggestion. We add explanation in Section 6.1 (measures) to clarify that, since our problem aims to select the keywords that have the maximum coverage of target tweets under two cost constraints (*i.e.*, cardinality and budget constraints), we are indeed interested in the "coverage" (the recall mentioned in the comments) and thus "coverage" is the most meaningful measure for evaluation.

Here, we clarify that the "precision" (*i.e.*, the percentage of target tweets in collected tweets) might not be suitable in our setting. Recall that, we use the cardinality constraint to limit that at most M keywords can be selected, and the budget constraint to limit that at most B tweets can be collected. If algorithms can fully utilize the budget, which means that they collect B tweets with M keywords, a high "coverage" under B budgets means a high "precision" at B tweets. However, algorithms may not fully utilize the budget, which means that they collect only B' tweets with M keywords, where B' is smaller than B . Thus, a high "precision" at B' documents does not mean a high "coverage" under B budgets. Algorithms with high precisions may not be desirable for our task. *E.g.*, baseline BH-3, which cannot fully utilize a large budget with M keywords, selects M specific keywords and collects only a small number of target tweets.

Thus, in our experiments, we choose to use the coverage as our main measure. In addition, when we compare ATM with the baselines in the two setting, we report the number of collected tweets (p-costs) to further illustrate the performance of each method. For example, we show that ATM achieves the maximum coverage among all the baselines, and costs much less than the best baseline method.

Comment W(e) & D(e): *The dynamic change of selected keyword sets could be shown in the work. It would be better if the authors display the dynamic change of keyword set for a particular topic when the ATM framework is monitoring target tweets on Twitter stream.*

Response: Great suggestion. We add experiments in Section 6.2.1 to quantitatively show that ATM collects more tweets than a static approach does, and present a new case study in Section 6.2.3 to illustrate how keywords are updated in every iteration.

Summary

Finally, we summarize our major revisions in Table 1. We note that, 1) besides these major revisions, we revised many details throughout the paper to better explain our ideas,

and 2) with the addition of the material in revision, we moved a previous experiment (*i.e.*, the synthetic setting), which compares our keyword selection algorithm with a basic exponential algorithm in a synthetic setting, to our technical report [16]. Reviewers can refer our report for this experiment.

Table 1: Revision Summary

No.	Place (Sec.)	Revision Description	Corresponding Comments
#1	1	We revised the presentation of our introduction.	
#2	2	We added discussion about topic models.	R1: W3 & D3
#3	3	We added running examples to illustrate ATM framework.	R2: W2 & D2
#4	3	We explained about taking classifiers as input in ATM.	R3: W(b) & D(b)
#5	3	We analyzed the impacts of the iteration length in ATM.	R1: W1 & D2 R2: W1, W3 & D1
#6	3	We explained how to construct candidate keywords K .	R2: D3
#7	4.1	We added motivations for the sampling algorithm.	R3: W(a) & D(a)
#8	4.2	We discussed the impacts of the selection of the two APIs.	R1: W2 & D1
#9	4.2	We revised the description of the sampling algorithm.	R3: W(c) & D(c)
#10	4.2	We analyzed the efficiency of the sampling algorithm.	R1: W2 & D1
#11	5	We provided additional complexity analysis for the selection algorithm in terms of the corpus size.	R1: W1 & D2
#12	6.1	We explained how to construct and update candidate keywords K in our experiments.	R2: D3
#13	6.1	We clarified the use of our metrics in experiments.	R3: W(d) & D(d)
#14	6.2.1	We reported the number of tweets collected by a method (p-costs) as an additional metric for comparing ATM with the baselines.	R3: W(d) & D(d)
#15	6.2.1	We added experiments to investigate the impacts of the selection of the two APIs in our sampling algorithm.	R1: W2 & D1
#16	6.2.2	We added experiments to evaluate the efficiency of our sampling algorithm in the Twitter stream.	R1: W2 & D1
#17	6.2.1	We added experiments to 1) compare our iterative framework with static approaches and 2) investigate the impacts of different iteration lengths.	R1: W1 & D2 R2: W1, W3 & D1
#18	6.2.3	We provided a new case study to illustrate how keywords are updated in iterations.	R2: W2 & D2 R3: W(e) & D(e)

Towards Social Data Platform: Automatic Topic-focused Monitor for Twitter Stream

Rui Li, Shengjie Wang Kevin Chen-Chuan Chang

{ruili1, wang260, kcchang}@illinois.edu

Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL, USA

ABSTRACT

Many novel applications (*e.g.*, crime detection) have been built based on analyzing tweets about a specific topic (*e.g.*, crime). While these applications provide different kinds of analysis, they share a common task of monitoring “target” tweets from the Twitter stream for a given topic. The current solution for this task tracks a set of manually selected keywords (*e.g.*, {“police”, “kill”}) with Twitter APIs. Obviously, this manual approach has many limitations. *E.g.*, it is laborious and performs without optimality guarantees.

In this paper, we propose a social data platform to automatically monitor target tweets from the Twitter stream for any given topic. In order to monitor target tweets in an optimal and continuous way, we design Automatic Topic-focused Monitor (ATM), which iteratively 1) “samples” tweets from the stream and 2) “optimizes” keywords to track based on the samples. To realize ATM, we solve two challenging problems. We develop a *tweet sampling algorithm* to enable sampling sufficient unbiased tweets with a biased Twitter API, and a *keyword selection algorithm* to efficiently select keywords that have a near-optimal coverage of target tweets under two cost constraints. We conduct extensive experiments to show the effectiveness of ATM. *E.g.*, ATM covers 87% of target tweets for a topic and improves the manual approach by 46%.

1. INTRODUCTION

Recently, various social media services, such as Twitter and Weibo, have emerged to support users to publish and share information online. In particular, Twitter, a pioneer of such social media services which we will focus on, is the most popular “micro-blog” for users to publish and share tweets. It now has nearly 140 million active users, who generate 340 million tweets everyday.

Compared to traditional media (*e.g.*, web pages), Twitter and other social media have several unique advantages: 1) *broad coverage*, tweets cover every aspect of our life, from national news (*e.g.*, president election), local events (*e.g.*, car theft at 2nd st.), to personal expressions (*e.g.*, “I like iPad”); 2) *fresh content*, with the brevity of tweets (140 characters) and the wide use of mobile devices, tweets are generated timely; 3) *rich attributes*, tweets not only contain text but also are associated with many attributes (*e.g.*, authors, time stamps and locations).

The unique advantages of Twitter and other social media make them information treasures for many novel applications.

Emergency Management (EM) Monitoring tweets about emergencies (*e.g.*, crimes and disasters) helps first aid responders to detect and handle crises timely. *E.g.*, Sakaki *et al.* [25] track earthquakes from related tweets in real time, and we [15] detect crime and disaster events instantly from related tweets.

Business Intelligence (BI) Monitoring tweets about products (*e.g.*, “iPad”) or brands (*e.g.*, “Apple”) helps business owners to know customers’ opinions [30] and address them accordingly. Moreover, it even provides accurate indicators for market analysts to predict stock prices [26].

Political Analysis (PA) Monitoring tweets about politics helps politicians to find concerns of voters in particular demographic groups (*e.g.*, females in California), and supports political analysts to predict election results [28].

As these and many other scenarios indicate, social media based applications [25, 15, 28, 26] usually share the same workflow. They start with collecting potentially relevant tweets for a topic (*e.g.*, crime), apply a classifier f to automatically determine whether a collected tweet is indeed relevant to the topic or not, and process the tweets that pass f for application-specific analysis.

Thus, while these applications conduct different kinds of analysis, they face the same problem of *monitoring target tweets from the Twitter stream with respect to a given classifier f* . Everyday, while hundreds of millions of tweets are generated, only a small percentage of them may pass the classifier f as target tweets for the application. It is difficult to collect most of them effectively.

Our Task To effectively support monitoring target tweets for any applications (*e.g.*, EM, BI, PA), we propose a *social data platform*, which allows users to plug in any classifier f as input and automatically collects target tweets for f from the Twitter stream as output. Ideally, the platform should meet the following requirements.

- *Optimal*: it should collect, with *optimal* or *near-optimal* guarantees, as many target tweets as possible under given computation resources, since many applications need a comprehensive coverage of target tweets to perform accurate analysis (*e.g.*, reporting a car theft at 2nd street). As target tweets are sparsely scattered, they are hard to catch comprehensively.
- *Continuous*: it should collect target tweets from the Twitter stream *continuously*, since new tweets are being created all the time. As target tweets with new content (*e.g.*, “Boston bomb”) may arise, it is challenging to capture the dynamics of the Twitter stream.

While the two requirements are crucial for all applications, current solutions of monitoring target tweets for a given classifier f (topic) cannot satisfy them.

Twitter provides a set of APIs [29], which represent standard programmatic ways of monitoring a document stream, but none of

them can directly monitor target tweets for a topic. Particularly, the *filter* API, which returns all the tweets containing a given keyword, will miss many target tweets that do not contain the keyword. The *sample* API, which returns 1% of all tweets (and thus 1% of target tweets), is insufficient for many applications mentioned above. The *firehose* API returns all tweets but requires a specific permission to use. Even if the *firehose* API is open to access, as it requires prohibitive processing costs (e.g., classifying all tweets), it is inefficient to use. Thus, for Twitter, monitoring target tweets for a topic is an unsolved problem.

Given Twitter APIs, since target tweets for a topic (e.g., crime) may share some topic-relevant keywords (e.g., “shoot”), many existing applications [25, 30, 26] use the *filter* API with a set of manually selected keywords (e.g., {“shoot”, “kill”}). However, this manual approach has severe disadvantages. First, it is *laborious*, as it requires extensive human efforts to select keywords for each topic. Second, the selected keywords are *not guaranteed* to be optimal or near-optimal. People might miss useful keywords (e.g., “police”) and thus many target tweets. Third, the keywords may quickly become *outdated* as time goes by, since new target tweets, which have different contents from previous ones (e.g., “Boston bomb”), are emerging and will be missed by them.

ATM Framework Thus, to monitor target tweets for a given classifier (topic), we have to address how to enable *automatically* selecting “optimal” and “continuous” keywords.

As our first contribution, in Sec. 3, we propose the Automatic Topic-focused Monitor (ATM) framework. Our basic intuition is that the “current” can predict the “near future”. Specifically, to select optimal and continuous keywords, we can estimate the “usefulness” of any set of keywords based on recent samples from the Twitter stream and select the most “useful” set to use. Thus, ATM takes a sampling, optimizing and tracking approach to monitor target tweets *iteratively*. In an iteration, ATM 1) samples tweets from the stream to enable estimation, 2) optimizes keywords to use based on their estimated coverage of target tweets, and 3) tracks target tweets with the selected keywords. To monitor target tweets continuously, ATM repeats the procedure in iterations (i.e., it tracks a new set of keywords every iteration). Since, within a short iteration, contents of tweets are similar and ATM selects keywords based on their coverage, the keywords are *optimal*. Further, in every iteration, ATM updates keywords according to recent samples from the Twitter stream, so the keywords are *continuous*.

Tweet Sampling Problem To realize ATM, we need to sample a *sufficient* number of *random* tweets from the Twitter stream in each iteration to enable accurate estimation. It is challenging because the available APIs are *limited* or *biased*. As we will prove in Sec. 4, the “accuracy” of the estimated coverage of keywords is inversely related to the square root of the sample size. Thus, the *sample* API, which only returns 1% of tweets in an iteration, may not provide enough samples. Further, directly using the *filter* API with keywords is biased to the tweets containing those keywords.

Thus, as our second contribution, in Sec. 4, we develop a random sampling algorithm to enable collecting a sufficient number of random tweets with the limited and biased APIs. To sample additional tweets, we effectively combine the available APIs to conduct *random walk sampling* on a “tweet graph”. As random walk sampling samples tweets according to a trial distribution defined by the graph, we further utilize *reject sampling* to adjust the tweets to the uniform distribution. As the main merit of our algorithm, we carefully design a *tweet graph*, which connects tweets with appropriate weights, to make sure that random walk sampling on the graph 1) can be easily realized with the available APIs and 2) theoretically converges to a known trial distribution over tweets.

Keyword Selection Problem To realize ATM, we further need to optimize keywords to use for a given classifier based on samples. When optimizing keywords, we have to consider “filtering costs” at Twitter and “post-processing costs” in ATM, since both computation resources are limited. Thus, we formally model our problem as selecting a set of keywords that have the maximum coverage of target tweets under two cost constraints. 1) Cardinality constraint: at most M keywords can be selected, as each keyword takes filtering costs. 2) Budget constraint: the total number of collected tweets is under a budget B , as each tweet brings post-processing costs. It is challenging because we need to efficiently select keywords in every iteration. As we will prove in Sec. 5, the problem is *NP-hard*, which prohibits finding the optimal solution in real time.

Thus, as our third contribution, in Sec. 5, we develop a keyword selection algorithm, which finds a near-optimal solution in polynomial time. Towards developing such an algorithm, we observe and prove that our problem possesses a desirable “submodular” property. While optimizing a submodular function with one constraint is well known, our problem has more than one constraints and needs a new solution. Based on the fact that maximizing a submodular function with one constraint has a greedy approximation algorithm, we develop a new greedy algorithm for our problem. The algorithm first relaxes our problem to a simple problem with one constraint (i.e., budget) and then handles the other constraint (i.e., cardinality). We also show the approximation rate of our algorithm.

Experiments With the two algorithms, we implement our ATM framework and evaluate it with extensive experiments in Sec. 6. Our experiments demonstrate the following results. First, our sampling algorithm collects a large number of additional random tweets. Second, our selection algorithm 1) is effective, which collects 83% of target tweets for a topic with only 20 keywords and improves the best baseline by 18%, 2) is efficient, and 3) works for various topics and constraints. Third, as an integrated framework, ATM greatly improves the manual (and static) approach by 46%.

2. RELATED WORK

In this section, we discuss our related work. Our work is related to crawling web pages, monitoring social media, retrieving relevant documents with keywords, and modeling topics in a text corpus.

Web page crawling has been a fundamental task since the beginning of the Web. Many studies, which focus on different issues, have been done. A good survey of them can be found in [20]. Among them, Chakrabarti *et al.* [7] propose the concept of focused crawling, which crawls pages relevant to a predefined topic. Specifically, before crawling a URL, a topic-focused crawler analyzes the URL’s context and its link structure to determine whether it is relevant. Our task is different, as we monitor tweets for a topic with keywords instead of crawling pages via hyperlinks.

Social media monitoring becomes an important task due to the emergence of social media based applications. While most applications [25, 26, 28] monitor their data based on a manual approach, some automatic monitors have been proposed. Hurst and Maykov [10] propose an architecture for monitoring general blogs, but they select blog feeds using simple rules (e.g., how often a new blog is posted) without considering topics. Our problem is different, as we design a topic-focused monitor. Boanjak *et al.* [6] propose a focused crawler, which crawls topic related tweets from heuristically selected users (i.e., the user who has the most connections to the existing ones). We [15] also present a heuristic rule to select keywords for monitoring crime-related tweets. However, these methods have two limitations: 1) they select users/keywords heuristically without any performance guarantees, and 2) they do not consider cost constraints (e.g., budget constraint).

Selecting keywords to retrieve relevant documents have been studied [24, 11, 1, 8] in other settings (instead of monitoring social media). For example, Robertson and Jones [24] design a weighting function to find keywords to retrieve additional relevant documents for a query. Agichtein and Gravano [1] utilize the function in [24] and other rules to find the keyword queries that retrieve *only* the relevant documents for information extraction. As these methods [24, 11, 1, 8] focus on different settings, they are not to find the keywords that have the *maximum coverage* of target tweets for a topic under *cost constraints*. Thus, they 1) neither have guaranteed performance for our problem (e.g., keywords selected in [1, 24] are too specific to cover many target tweets), 2) nor consider cost constraints (e.g., they [8] do not limit the total number of collected documents). Further, they do not address how to sample sufficient tweets from the Twitter stream to update keywords iteratively.

Topic models [5, 9] have been proposed to discover topics for a text corpus, where each topic is a multinomial distribution of keywords. We cannot apply them in our problem for two reasons. First, standard topic models [5, 9] are unsupervised methods for discovering hidden topics in a corpus instead of selecting keywords for a given topic. The discovered topics may not correspond to the specific topic of our interest. Second, although some extended topic models [4] can incorporate supervised information, as topic models do not model “coverage” and “cost”, their discovered topics (*i.e.*, distributions of keywords) neither have a near-optimal coverage of target tweets with guarantees nor consider cost constraints.

Our ATM framework advances the above methods [15, 6, 24, 11, 8] from two aspects. First, ATM selects keywords in a constrained optimization approach, which 1) finds near-optimal keywords with guarantees and 2) considers two types of costs. Second, ATM updates keywords in iterations, which monitors the dynamic Twitter stream continuously. To enable updating keywords, we design a sampling algorithm to sample random tweets from the stream.

3. OVERVIEW

In this section, we propose our ATM framework and abstract two challenging problems in realizing ATM.

Twitter APIs To begin with, we first introduce three APIs provided by Twitter for monitoring public tweets from the Twitter stream. They represent three standard programmatic ways of accessing a corpus. Details of them can be found in [29].

- **Sample** returns a set of random samples (approximately 1%) of all public tweets.
- **Filter** returns the public tweets that match given filter predicates (e.g., a keyword “police”).
- **Firehose** returns all public tweets.

Given the three APIs, we choose the *filter* API to monitor target tweets for a topic for two reasons. First, we cannot use the *sample* API or the *firehose* API. The *sample* API only gives 1% of target tweets, which are not enough for many applications (e.g., detecting local crimes [15] or predicting stock prices based on redundancies [26]). We note that the *sample* API returns the same samples even if we call it from different machines. The *firehose* API requires a specific permission to use and is not available to general users (e.g., data mining researchers). Even if it is available, we should not use it, as it requires prohibitive processing costs (e.g., processing all the tweets). Second, if keywords are well selected, it is possible to collect most target tweets for a topic (e.g., crime) using the *filter* API with well selected keywords, since, intuitively, different target tweets may share similar keywords (e.g., “shoot”). We note that Twitter has other APIs, but they are not for monitoring public tweets. E.g., the *user* API requires a user’s authentication, and returns the tweets from his friends.

ATM Overview Based on Twitter APIs, we propose ATM to effectively monitor the Twitter stream for any given classifier (topic). As Fig. 1 illustrates, ATM generally takes any classifier f as input and outputs target tweets for f under certain cost constraints.

Here, we emphasize that it is reasonable to take a classifier f as input. As we motivated in Sec.1, our goal is to generally support monitoring target tweets for various social media based applications (e.g., EM, BI, PA), which have already used classifiers [15, 26, 30] to *automatically* determine their target tweets. Thus, our framework only leverages the existing classifiers in those applications and does not add any extra burden. Further, while our focus is not the scenarios where classifiers do not already exist, it is possible to train classifiers and use ATM for these scenarios, since many classifiers have been studied in Twitter [30, 23] and in other settings, and they can accurately predict target tweets for a topic with advanced models [30, 23] and novel features [15].

Specifically, given a classifier f (e.g., crime), to collect its targets tweets from the Twitter stream in an optimal and continuous way, ATM iteratively selects optimal keywords to track.

At the i^{th} iteration, to monitor target tweets with “optimal” keywords, ATM works in three steps. First, to enable estimating the usefulness of any set of keywords, a *sampler* collects a large amount of tweets S_i (e.g., t_1 : “police detained 17 people...”, t_2 : “car theft...”, t_3 : “enjoy my tea...” from the stream. Then, given the samples S_i and the classifier f , a *selector* selects a set of keywords K_i^* (e.g., {“police”, “theft”}) that have the maximum coverage of target tweets in S_i (e.g., t_1, t_2) under some cost constraints. Finally, a *tracker* calls the *filter* API with K_i^* to collect target tweets (e.g., t : “police arrested ...”) from the stream for this iteration.

Since target tweets with new content are emerging, ATM updates keywords iteratively to monitor target tweets “continuously”. While the tracker monitors target tweets with keywords K_i^* for the i^{th} iteration, the *sampler* collects new samples S_{i+1} (e.g., t_1 : “bomb in Marathon...”, t_2 : “FBI came...”, t_3 : “good food...” for the $(i+1)^{th}$ iteration. When the i^{th} iteration finishes, the *selector* selects a new set of keywords K_{i+1}^* ({“bomb”, “FBI”}) based on S_{i+1} , and the *tracker* uses K_{i+1}^* to collect target tweets for this new iteration.

The iteration length l should be carefully set in ATM. For any topic, l should be short enough (e.g., an hour) to capture the dynamics of the Twitter stream, since target tweets with new content may emerge and need to be captured with new keywords. However, l cannot be too short (e.g., 5 mins), since we may not collect enough tweets in a short iteration to accurately estimate of the usefulness of keywords. Further, since different topics require different numbers of samples for accurate estimation (e.g., a sparse topic like crime needs many samples) and their target tweets change at different rates (e.g., tweets about olympics news develop very fast), l should be different for different topics. Thus, for a given topic, we treat l as an important parameter to tune. As ATM works for any given l , we can find a reasonable l for a topic easily. We can first test the performance of ATM with different l and then select the best one to use. We note that in this paper we focus on the two essential problems in each iteration, and we plan to study how to automatically set l for a given topic as our future work.

ATM meets our requirements in Sec. 1. First, ATM is guaranteed to use *optimal* or *near-optimal* keywords, since we can intuitively assume that, within a short iteration, target tweets are similar to those in samples, and ATM selects keywords based on their usefulness on the samples. Second, ATM can *continuously* monitor target tweets, since ATM selects new keywords based on the most recent samples in every iteration.

To realize ATM, in each iteration (*i.e.*, a short time period), we have to 1) sample a sufficient amount of random tweets, which

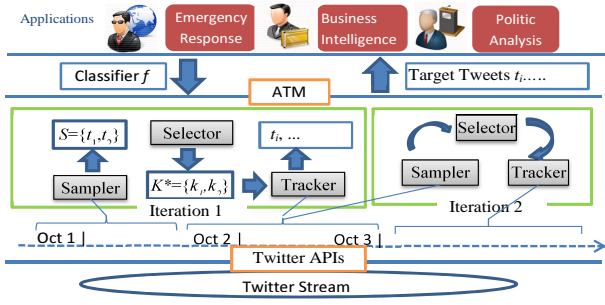


Figure 1: The Overview of ATM

may be more than the samples returned by the *sample* API (i.e., 1% of tweets in an iteration) to enable accurate estimation, and 2) efficiently find the optimal keywords to use under cost constraints based on their estimated usefulness.

Here, we treat them as two independent problems, *tweet sampling* and *keyword selection*, for two reasons. First, a separate sampler is “topic-independent” and can collect samples for serving different topics (e.g., crime or politic). Second, each problem is meaningful by itself with many applications. The solution for tweet sampling can be used as a general tweet sampler to collect sufficient random tweets for different applications, as many applications (e.g., estimating sparse prosperities of the Twitter stream in a given day) require collecting more than 1% of tweets. The solution for keyword selection can also be applied to other scenarios (e.g., selecting experts for a community). We note that, for few systems (e.g., Twitter company), which have access to all the tweets, the first problem might be easy, but how to solve the second problem is unclear. Further, as most social media based applications [25, 30, 26, 15] only have access to the *filter* and *sample* APIs, both problems are challenging. Next, we define them formally.

Problem Abstraction To begin with, we introduce some notations. We use 1) t as a tweet, 2) k as a keyword, 3) T as the set of all the tweets in an iteration, and 4) K as the set of all the keywords that can be used as filters. A keyword $k \in K$ can be any single term (e.g., “police”). To cover all useful keywords, K should be complete (i.e., it covers all the keywords in T). We can construct K via enumerating all the unigrams in T . We use K' to denote a subset of K . We define the *match* of a tweet t , denoted as $M(t)$, as the set of keywords that t contains, and the *volume* of a keyword k , denoted as $V(k)$, as the set of the tweets containing k .

First, we abstract the *tweet sampling* problem. We use S to represent a set of samples of T . To make unbiased estimation, S should be *uniformly* sampled from T , which means that $\forall t_i, t_j \in T$, the probability of t_i in S , denoted as $P(t_i \in S)$, is the same as $P(t_j \in S)$. To make accurate estimation, S should contain a *sufficient* number of samples. Thus, $|S|$ should be larger than a threshold γ . Now, we formally state the tweet sampling problem.

Tweet Sampling Problem Let T be all the tweets in an iteration. Given a threshold γ , which is smaller than $|T|$, the *filter* API, and the *sample* API, output a set of samples $S \subset T$, s.t. $\forall t_i, t_j \in T$, $P(t_i \in S) = P(t_j \in S)$ and $|S| > \gamma$.

Next, we abstract the *keyword selection* problem. We denote the given classifier for a topic as a binary function f . Given a tweet t , f outputs 1 if t is relevant to the topic, and 0 otherwise. We call t a *target* tweet if $f(t) = 1$, and use R to represent all target tweets, $R = \{t | f(t) = 1, t \in T\}$. Based on f , we can quantitatively define the “usefulness” of keywords. In particular, we define the *cover* of a keyword k , denoted as $C(k)$, as the set of the target tweets containing k , $C(k) = \{t_j | t_j \in V(k), f(t_j) = 1\}$, and measure the *usefulness* of a set of keywords $K' \subset K$, denoted as $U(K')$, as the

number of the target tweets covered by K' , $|\cup_{k_i \in K'} C(k_i)|$. In this paper, we use “usefulness” and “coverage” interchangeably. Then, we define two constraints to limit “costs” of selected keywords.

Cardinality Constraint limits “filtering” costs of a solution K' . It takes costs to filter incoming tweets for each keyword, but such computation resources are limited. E.g., the *filter* API only accepts up to 400 keywords as filters. Thus, we use K' ’s cardinality $|K'|$ to model its filtering costs, and limit $|K'|$ below a threshold M .

Budget Constraint limits “post-processing” costs of a solution K' . Each collected tweet needs to be processed (e.g., classified by f), but such computation resources are also limited. We use the number of the tweets collected by K' to model its post-processing costs, denoted as $P(K')$, where $P(K') = \sum_{k_i \in K'} |V(k_i)|$, and limit $P(K')$ below a budget B .

Here, we measure $P(K')$ as the sum of the volumes of the keywords in K' without considering that a tweet could be covered by multiple keywords, because each keyword filter is applied individually, and we suffer from processing such redundancies. We are aware that other measures can also model post-processing costs. We use the current measure to illustrate how to consider the budget constraint. Our solution is general to deal with different measures. Now, we formally abstract the keyword selection problem.

Keyword Selection Problem Given a classifier f , a set of tweets T , a set of candidate keywords K , a threshold M and a budget B , output $K' \subset K$, s.t. $U(K')$ is maximized subject to $|K'| \leq M$ and $P(K') \leq B$.

4. TWEET SAMPLING PROBLEM

We now focus on the *tweet sampling* problem. Specifically, we aim to collect a *sufficient* number of *random* samples S from all the tweets T in an iteration with the available Twitter APIs (i.e., the *filter* API and the *sample* API) to estimate the usefulness $U(K')$ and the post-processing cost $P(K')$ for any set of keywords K' .

4.1 Motivation

First, we motivate the need for a tweet sampling algorithm besides the *sample* API, which returns 1% of tweets. As we discussed in Sec. 3, to capture the dynamics of the Twitter stream, especially for fast developing topics (e.g., olympic news), ATM prefers a short iteration. Further, as we will show below, to accurately estimate the usefulness for any set of keywords, ATM needs a sufficient number of samples, which may be more than 1% of tweets in an iteration. Thus, it is desirable to have a sampling algorithm, which provides additional samples besides the *sample* API, to enable collecting sufficient samples in a short iteration or speed up the *sample* API to make an iteration shorter and capture the dynamics of the stream better. We clarify that, while we can take a long iteration to collect many tweets with the *sample* API, this strategy might not be good since a long iteration may fail to capture the dynamics of the stream and will miss many target tweets.

Then, we develop a theorem to formally relate the estimation accuracy and the sample size. Here, we focus on estimating $U(K')$ for a set of keywords K' . However, our discussion can be applied to $P(K')$ as well. We denote the estimated value in S as $\tilde{U}(K')$ to differentiate it from the true value $U(K')$ in T .

Assumption We first give a simple but realistic assumption. Although the *sample* API samples tweets *without replacement*, we assume that it samples tweets *with replacement*, as T is very large and the chance of getting the same sample is negligible.

Intuition We then give our intuition. $U(K')$ measures the number of the tweets that 1) are target tweets and 2) match any keyword $k \in K'$ in T . If we randomly draw a tweet from T , it has a

probability $U(K')/|T|$ to meet the two requirements. A set of random samples S can be viewed as drawing tweets repeatedly for $|S|$ times. We can model S as a *Bernoulli Process* with a success probability $U(K')/|T|$, and $\tilde{U}(K')$ as the number of successes in $|S|$ independent Bernoulli trials. Thus, $\tilde{U}(K')$ follows the *binomial distribution* with a success probability $U(K')/|T|$.

Our problem becomes how accurately we can estimate the parameter $|U(K')|/|T|$ of the binomial distribution with $\tilde{U}(K')$ successes observed from $|S|$ samples. We apply existing results about the parameter estimation for the binomial distribution in statistics [31], and obtain the following theorem (the proof is in [31]).

THEOREM 4.1. *Given a set of random samples S from the entire set T and an error percentile α , with $1 - \alpha$ confidence, $\frac{|U(K')|}{|T|}$ is within $\frac{\tilde{U}(K')}{|S|} \pm z_{1-\alpha/2} \sqrt{\frac{\tilde{U}(K')/|S| - (\tilde{U}(K')/|S|)^2}{|S|}}$, where $z_{1-\alpha/2}$ is the $1 - \alpha/2$ percentile of the standard normal distribution.*

The theorem is useful from several aspects. 1) It shows that, given a confidence level (e.g., 95%), we should increase the sample size $|S|$ to make our estimation $\tilde{U}(K')/|S|$ close to the true value $U(K')/|T|$. 2) It gives a formula to calculate the necessary number of samples for achieving a certain accuracy. 3) It shows that the required numbers of samples are different for different topics since $U(K')$ are different.

4.2 Tweet Sampling Algorithm

Now, we develop our sampling algorithm with the available Twitter APIs. Since the *sample* API may not provide enough samples, we need to use the *filter* API. However, directly using it with a set of keywords is *biased* to the tweets containing the given keywords. Thus, it is challenging to utilize the *filter* API to collect additional *unbiased* (or uniform) samples. Here, we clarify that we aim to provide *additional* samples besides the samples returned by the *sample* API instead of replacing them.

We develop our sampling algorithm based on a widely used sampling framework, which *uniformly* samples nodes from a graph via integrating two sampling methods, *random walk* sampling and *rejection* sampling. In the literature, specific algorithms have been developed based on the framework to sample pages from the Web graph [3] or users from a social network graph [12]. In this paper, we adopt the framework to develop a new algorithm for sampling tweets with the available Twitter APIs. It is possible, because we can connect tweets as a “tweet graph” through the APIs. However, we cannot apply the existing algorithms, because they sample from different graphs (e.g., a social network graph) with different access functions (e.g., getting friends of a user). We must design our own “tweet graph” and sample with the available APIs.

Preliminary To begin with, we briefly describe *random walk* sampling and *rejection* sampling methods.

Random Walk on a graph $G(N, E)$, where N denotes a set of nodes and E denotes a set of weighed edges, is a markov chain on a finite state space N [17]. It can be described as a surfer randomly walking among G . At a node n_i , the surfer visits a neighbor node n_j randomly according to the weight of their edge e_{ij} . After several steps, the surfer reaches different nodes with different probabilities. In theory, if G is “ergodic”, the probabilities of visiting different nodes are guaranteed to converge to a distribution ϕ over nodes N . G is *ergodic*, if 1) G is strongly connected, and 2) the greatest common denominator of all cycle lengths is 1. Thus, random walk sampling works in two steps. 1) It randomly walks for several steps, which are called as the *burning period*. 2) It generates the next visited node as a sample. If G is ergodic, the samples are generated according to the distribution ϕ defined by G , which we call a *trial distribution*.

Rejection Sampling is a simulation method for generating samples according to a target distribution π with samples generated from a trial distribution ϕ . Intuitively, it uses “acceptance probabilities” to bridge the gap between ϕ and π . E.g., when π is the uniform distribution and ϕ is another distribution, it assigns high acceptance probabilities to instances that have low probabilities in ϕ . As rejection sampling only cares the relativity of ϕ and π , it is defined based on their *un-normalized forms*. We define an *un-normalized form* of a distribution π , denoted as $\hat{\pi}$, if $\exists Z_\pi$, s.t. $\forall n \in N$, $\hat{\pi}(n) = \pi(n) \times Z_\pi$. Given an un-normalized trial distribution $\hat{\phi}$ and an un-normalized target distribution $\hat{\pi}$ over the space N , we define the *acceptance probability* of an instance n as $\hat{\pi}(n)/(C\hat{\phi}(n))$, where C is a constant that satisfies $C \geq \max_{n \in N} \hat{\pi}(n)/\hat{\phi}(n)$.

Algorithm 1 TweetSample()

```

while true do
   $t = \text{RandomWalk}_\phi()$ ;
  toss a coin with head probability  $\frac{\hat{\pi}(t)}{C\hat{\phi}(t)}$ ;
  if head then return  $t$ 
end if
end while

```

Sampling Algorithm Then, based on the framework, we develop our sampling algorithm, which is shown in Alg. 1. The algorithm first calls RandomWalk_ϕ to get a sample t . This function utilizes the available APIs to conduct random walk sampling on a “tweet graph”. We denote the *tweet graph* as $G(T, E)$, where the nodes are tweets T and they are connected by weighed edges E . We will define G and describe RandomWalk_ϕ later. As t follows the trial distribution ϕ defined by G instead of a uniform distribution π , Alg. 1 then applies rejection sampling to decide whether t is accepted with the acceptance probability $\frac{\hat{\pi}(t)}{C\hat{\phi}(t)}$. As π is the uniform distribution, $\hat{\pi}(t) = 1$. We show $\hat{\phi}(t)$ and C after we define G .

Challenges To complete Alg. 1, we need to design $G(T, E)$, which connects tweets T with weighed edges E . It is not easy, as G has to meet two requirements.

- **Feasible:** We can realize random walk from t_i to t_j according to the weight of their edge e_{ij} with the available APIs.
- **Ergodic:** G must be ergodic so that random walk on G converges to a unique probability distribution ϕ .

Tweet Graph As the key merit of our algorithm, we design a tweet graph G , which meets the two requirements. Here, we clarify that our algorithm only needs to conduct random walk from a tweet t_i to another tweet t_j with the APIs according to their weight in G and does not need to build a complete G explicitly. We use $P(t_i \rightarrow t_j)$ to denote the probability of walking from t_i to t_j . According to the random walk theory, $P(t_i \rightarrow t_j) = \frac{e_{ij}}{D(t_i)}$, where $D(t_i)$ is the degree of t_i , and $D(t_i) = \sum_{t_j \in T} e_{ij}$.

Feasible. To make G feasible, we need use the *filter* API to randomly “walk” from a tweet t_i to a tweet t_j . As the *filter* API uses a keyword to retrieve tweets, we can implement walking in two steps. First, we randomly pick a keyword k from $M(t_i)$, which is the set of keywords in t_i . Second, we use the *filter* API with k to get a random tweet t_j from $V(k)$, which is the set of the tweets containing k . In this way, the probability of walking from t_i to t_j through a keyword k is $\frac{1}{|M(t_i)| \cdot |V(k)|}$. Since t_i and t_j may share multiple keywords $M(t_i) \cap M(t_j)$, we have $P(t_i \rightarrow t_j) = \frac{1}{|M(t_i)|} \sum_{k \in M(t_i) \cap M(t_j)} \frac{1}{|V(k)|}$. For different t_j , $P(t_i \rightarrow t_j)$ is proportional to $\sum_{k \in M(t_i) \cap M(t_j)} \frac{1}{|V(k)|}$, as $|M(t_i)|$ is a constant at a specific t_i . According to the definition, $P(t_i \rightarrow t_j)$ is proportional to e_{ij} , so we directly set e_{ij} as $\sum_{k \in M(t_i) \cap M(t_j)} \frac{1}{|V(k)|}$.

However, G with e_{ij} defined above may not be ergodic, as G may not be strongly connected.

Ergodic. To make G ergodic, we add a small teleport weight to the edge of any pair of tweets. Thus, at a tweet t_i , we can “jump” to any tweet t_j with a *small* probability. As any pair of tweets is connected, G is ergodic. Specifically, we add a total weight λ for jumping from t_i to all the tweets in T , and a weight $\frac{\lambda}{|T|}$ for jumping from t_i to t_j . Thus, we adjust e_{ij} as $(\sum_{k \in M(t_i) \cap M(t_j)} \frac{1}{|V(k)|}) + \frac{\lambda}{|T|}$. To implement jumping from t_i to t_j , we use a sample returned by the *sample* API, as we can view the *sample* API as a uniform sampler, which returns a tweet t_j with $\frac{1}{|T|}$ but can only be used for a limited number of times (i.e., $|T|/100$). Thus, G is feasible.

According to the new weight, we need to determine how likely we do “walking” and “jumping” at a tweet t_i . We first calculate the new $D(t_i)$ based on the new weight e_{ij} , and then derive $P(t_i \rightarrow t_j)$ based on $\frac{e_{ij}}{D(t_i)}$.

$$D(t_i) = \sum_{t_j \in T} ((\sum_{k \in M(t_i) \cap M(t_j)} \frac{1}{|V(k)|}) + \frac{\lambda}{|T|}) = |M(t_i)| + \lambda \quad (1)$$

$$P(t_i \rightarrow t_j) = \frac{\frac{|M(t_i)|}{|M(t_i)| + \lambda} (\sum_{k \in M(t_i) \cap M(t_j)} \frac{1}{|V(k)|} \frac{1}{|M(t_i)|})}{\frac{|M(t_i)|}{|M(t_i)| + \lambda} + \frac{\lambda}{|T|}} \quad (2)$$

Based on Eq. 2, we can interpret $P(t_i \rightarrow t_j)$ as a combination of “walking” ($\sum_{k \in M(t_i) \cap M(t_j)} \frac{1}{|V(k)|} \frac{1}{|M(t_i)|}$) based on the *filter* API with a probability $\frac{|M(t_i)|}{|M(t_i)| + \lambda}$ and “jumping” ($\frac{1}{|T|}$) based on the *sample* API with a probability $\frac{\lambda}{|M(t_i)| + \lambda}$. λ works as a parameter for choosing “jumping” or “walking”.

Here, we discuss how to set λ . We introduce λ (i.e., a non-zero value) to theoretically guarantee that our graph is ergodic and our random walk converges. As we will prove below, our random walk converges to different known distributions with different λ , and all of them can be adjusted to the uniform distribution. Here, the λ value plays the same role as the teleport weight used in pagerank [21]. Pagerank converges with any non-zero teleport weight. However, a large λ will lead to use the *sample* API a lot and thus collects only a small percentage of *additional* samples with the *filter* API, which is not useful. Thus, we set λ to a small value. As we will show in experiments, with $\lambda = 0.1$, our algorithm performs like a uniform sampler and collects a large percentage of additional samples.

Convergence Distribution As G is ergodic, random walk on G converges to a unique distribution ϕ over T . Intuitively, according to $P(t_i \rightarrow t_j)$, a tweet t_j is more likely to be sampled, if it contains more keywords. We formally give the un-normalized distribution $\hat{\phi}$ over T with Theorem 4.2.

THEOREM 4.2. *The random walk on $G(T, E)$ converges to an un-normalized distribution $\hat{\phi}$ over T , where $\hat{\phi}(t) = |M(t)| + \lambda, \forall t \in T$.*

PROOF. According to our definition, we have $e_{ij} = e_{ji}$. Thus, G can be viewed as an undirected graph. According to [17], the stationary distribution of an undirected and complete graph is proportional to the degree distribution. As Eq. 1 shows, $D(t) = |M(t)| + \lambda$. Thus, $\hat{\phi}(t)$ is $|M(t)| + \lambda$. \square

We emphasize that the theorem formally shows that we can sample tweets according to a known distribution for any λ . We can further use rejection sampling to adjust them according to the uniform distribution. As $\hat{\phi}(t)$ is at least one and $\hat{\pi}(t) = 1$, $C = 1$ is sufficient for the acceptance probability.

Random Walk Algorithm Now, we present RandomWalk $_{\phi}$ in Alg. 2. “Burning” is a general term used in Markov chain Monte Carlo methods (e.g., random walk sampling) to describe getting a

“good” starting point t_0 . Usually, we can start from the previous sample collected the algorithm and may throw away some iterations at the very beginning. After burning, the algorithm generates a sample based on t_0 . It decides whether “walking” or “jumping” with probability $\frac{|M(t_0)|}{|M(t_0)| + \lambda}$. If yes, it samples a keyword k from $M(t_0)$, calls the *filter* API with k , and outputs a random tweet matching k as the sample. Otherwise, it uses a sample from the samples returned by the *sample* API.

Algorithm 2 RandomWalk $_{\phi}$ ()

```

 $t_0$  = do Burning;
toss a coin with head probability  $\frac{|M(t_0)|}{|M(t_0)| + \lambda}$ ;
if !head then
     $k$  = randomly sample a keyword from  $M(t_0)$ ;
     $t$  = a random tweet returned by the filter API with  $k$ ;
else
     $t$  = a random tweet returned by the sample API;
end if
return  $t$ ;

```

Efficiency We now discuss the efficiency of our algorithm. Our algorithm costs insignificant CPU resources, as it only requires to compute a few easy-to-compute variables (e.g., $|M(t_0)|$) to collect a sample. Its efficiency mainly depends on how quickly we get a sample with the Twitter APIs (e.g., it takes time to connect Twitter and get a sample). Due to space limits, we only show the major steps of our algorithm (in Alg.2) to illustrate the main design. We leave details of how to efficiently implement our algorithm with Twitter APIs in our technical report [16] (e.g., we start multiple random walkers together and merge their API requests; and when calling the *filter* API with a keyword, we cache several samples for future reuse). As our experiments will show, our algorithm runs efficiently in the Twitter stream. E.g., it collects 30K *additional* samples per hour from the Twitter stream, which helps to speed up the *sample* API by 1.4 times.

Further, we explain that our algorithm enables collecting a desired percentage of random samples from the Twitter stream, which exceeds the limit of the *sample* API (i.e., 1%). While the number of additional samples collected by a single instance of our algorithm is limited (i.e., 30K), running our algorithm in parallel can scale up the efficiency, since different instances randomly choose different keywords and collect different samples. Recall that calling the *sample* API from different machines gives the same samples. As our experiments will show, two instances of our algorithm collect 1.96 times as many additional samples as a single instance. Thus, as our main contribution to the tweet sampling problem, our algorithm enables sampling a large percentage of additional random tweets from the Twitter stream with the available Twitter APIs. We note that it is reasonable to collect random samples with multiple instances for ATM, since, as we mentioned in Sec 3, the samples are “topic-independent” and can serve many topics (e.g., crime, politic).

5. KEYWORD SELECTION PROBLEM

In this section, we develop an efficient algorithm for the keyword selection problem. As we need to select keywords in each iteration (e.g., every hour) timely, the algorithm has to be efficient. Here, we view the collected samples S as the set of all the tweets T .

NP-hard Problem To formally argue that our problem is difficult, we prove the following theorem.

THEOREM 5.1. *Keyword selection problem is NP-hard.*

To prove the theorem, we reduce the *set cover* problem, to our problem. Due to space limits, we leave the proof in our report [16].

The theorem suggests that there is no polynomial time algorithm for the optimal solution. A basic exponential algorithm works

as follows. It enumerates all the subsets that contain at most M keywords, evaluates their usefulness and post-processing costs, and outputs the most useful set whose costs are under the budget. It is inefficient, as it enumerates $|K|^M$ subsets, where $|K|$ is usually larger than thousands and M is larger than 10.

While we cannot find the optimal solution efficiently, we design an efficient algorithm to find a near-optimal solution. Towards developing such an algorithm, we make two contributions.

Nontrivial Submodular Maximization Problem As our first contribution, we formally prove a desirable property (submodular) of the usefulness measure $U(K')$ in our problem, and model our problem as a non-trivial submodular function maximization problem.

In combinatorial optimization problems, the *submodular* property of a target function F is a desirable property for deriving efficient approximation algorithms. Specifically, a function $F : 2^S \rightarrow \mathbb{R}$, which returns a real value of any subset $S' \subset S$, is a submodular function if $F(B \cup \{e\}) - F(B) \leq F(A \cup \{e\}) - F(A)$ for any $A \subset B \subset S$ and $e \in S \setminus B$. Maximizing such a function with some types of constraints (e.g., the cardinality constraint or the budget constraint) can be solved near-optimally with simple greedy algorithms. In the literature, the submodular property has been studied for many NP-hard problems, such as the *set cover* problem and the *knapsack* problem, and leads to efficient approximation algorithms for those problems. Recently, it is explored to solve many data mining [13] and machine learning [2] problems. Here, we explore the submodular property for a new problem of monitoring social media, and present the following theorem as our finding.

THEOREM 5.2. *The function $U(K')$ in the keyword selection problem is a monotonic submodular function.*

PROOF. First, we show $U(K')$ is a monotonic function. Specifically, $U(K_1) \leq U(K_2)$ for all $K_1 \subset K_2 \subset K$, since adding any keyword k to K_1 can only increase its coverage.

Second, we show that $U(K_1 \cup \{k\}) - U(K_1) \geq U(K_2 \cup \{k\}) - U(K_2)$ for all $K_1 \subset K_2 \subset K$.

$$\begin{aligned} U(K_1 \cup \{k\}) - U(K_1) &= |(\cup_{k_i \in K_1} C(k_i) \cup C(k)) - \cup_{k_i \in K_1} C(k_i)| \\ &= 2|(T - \cup_{k_i \in K_1} C(k_i)) \cap C(k)| \geq 3|(T - \cup_{k_i \in K_2} C(k_i)) \cap C(k)| \\ &= 4|(\cup_{k_i \in K_2} C(k_i) \cup C(k)) - \cup_{k_i \in K_2} C(k_i)| = 5U(K_2 \cup \{k\}) - U(K_2) \end{aligned}$$

At step 2, we apply $(A \cup B) - A = (U - A) \cap B$, where U is the universe, $A \subset U$, and $B \subset U$. At step 3, since $K_1 \subset K_2$, $\cup_{k_i \in K_1} C(k_i) \subset \cup_{k_i \in K_2} C(k_i)$ and $T - \cup_{k_i \in K_2} C(k_i) \subset T - \cup_{k_i \in K_1} C(k_i)$. \square

Thus, we model our problem as maximizing a monotonic submodular function $U(K')$ under two constraints, 1) the cardinality constraint $|K'| \leq M$, and 2) the budget constraint $P(K') = \sum_{k_i \in K'} |V(k_i)| \leq B$.

This is a non-trivial problem because greedy algorithms are only proved to work for maximizing a submodular function under either the cardinality constraint [19] or the budget constraint [27]. Alg. 3 and Alg. 4 are the corresponding greedy algorithms. The combination of two constraints makes both algorithms fail, since the result of the algorithm for one constraint may violate the other constraint. We note that the problems [13, 2] explored in data mining or machine learning are all associated with one constraint.

Only until recently, theoretical computer scientists develop a randomized approximation algorithm *MLC* [14] for maximizing a submodular function with multiple constraints with a $(1 - \epsilon)(1 - e^{-1})$ approximation by expectation for a given constant ϵ . However, *MLC* is hardly applied to our setting, as it has a high order in its polynomial complexity (e.g., it has to solve several linear programming problems), and the result is non-deterministic.

Greedy Algorithm As our second contribution, we develop an efficient algorithm and show its approximation rate. Our intuition is that we can relax our problem to the problem with the budget constraint first, which we solve with a greedy algorithm (Alg. 4), and then handle the cardinality constraint only if the returned solution of the relaxed problem violates it.

Alg. 5 shows our algorithm. It considers the budget constraint first and calls Alg. 4, which iteratively selects useful keywords based on the *marginal usefulness ratio* in a greedy way. If the returned solution K' of Alg. 4 satisfies the cardinality constraint, our algorithm returns K' as the solution. Otherwise, it handles the cardinality constraint via selecting the M keywords that have the maximum usefulness from K' . This can be viewed as a problem of maximizing the submodular function under the cardinality constraint. Thus, it calls Alg. 3, which selects keywords based on its *marginal usefulness*, and returns its result as the solution.

Algorithm 3 CardinalityConstraint(M,T,K)

```

 $K' = \{\};$ 
for  $a = 1 \rightarrow M$  do
   $\text{let } k = \text{argmax}_{k_i \in K - K'} U(\{k_i\} \cup K') - U(K');$ 
   $K' = K' \cup \{k\};$ 
end for
return  $K'$ ;

```

Algorithm 4 BudgetConstraint(B,T,K)

```

 $K' = \{\};$ 
 $\text{let } \text{best} = \text{argmax}_{k_i \in K} U(k_i) \text{ subject to } P(\{k_i\}) \leq B;$ 
while true do
   $\text{let } k = \text{argmax}_{k_i \in K - K'} \frac{U(\{k_i\} \cup K') - U(K')}{P(\{k_i\} \cup K') - P(K')} \text{ subject to } P(\{k_i\}) + P(K') \leq B;$ 
  if  $k$  does not exist then
    break;
  end if
   $K' = K' \cup \{k\};$ 
end while
return  $\text{argmax}_{K'_{\text{best}} \in \{\{\text{best}\}, K'\}} U(K'_{\text{best}})$ 

```

Algorithm 5 GreedyApproximation(B,T,M,K)

```

 $K' = \text{BudgetConstraint}(B, T, K);$ 
if  $|K'| \leq M$  then
  return  $K'$ ;
else
  return  $\text{CardinalityConstraint}(M, T, K');$ 
end if

```

Complexity Now we analyze the complexity of Alg. 5. Both routines (Alg. 3 and Alg. 4) greedily select keywords one by one. At most B and M keywords are selected in Alg. 4 and Alg. 3. To select a keyword, it needs to measure the weights for at most $|K|$ keywords, and the weight of a keyword requires at most $O(|T|)$ comparisons, where $|T|$ is the size of the corpus. Thus, its complexity is $O((M + B)|K||T|)$, which is much more efficient than the exponential algorithm, whose complexity is $O(|T|K^M)$.

Approximation Rate Further, we analyze the approximation rate of our algorithm with the following theorem.

THEOREM 5.3. *Alg. 5 achieves an approximation rate at least $\frac{M}{|\mathcal{O}_B|}(1 - e^{-1})^2$, where $|\mathcal{O}_B|$ is the number of keywords returned by Alg. 4.*

PROOF. Our proof is based on the intuition of our algorithm. First, we denote the optimal solutions for maximizing the function U under both constraints and only the budget constraint as \mathcal{O} and \mathcal{O}_B , respectively. As \mathcal{O} is the solution with an additional constraint, we have $U(\mathcal{O}_B) \geq U(\mathcal{O})$. Second, we denote the

solution returned by Alg 4 as \mathcal{O}'_B . As shown in [27], $U(\mathcal{O}'_B) \geq (1 - e^{-1})U(\mathcal{O}_B)$. Thus, if \mathcal{O}'_B satisfies the cardinality constraint, we have $U(\mathcal{O}'_B) \geq (1 - e^{-1})U(\mathcal{O}_B) \geq (1 - e^{-1})U(\mathcal{O})$. Otherwise, we run Alg 3. We denote the result of the optimal M keywords in \mathcal{O}'_B as \mathcal{O}_M . As \mathcal{O}_M is the optimal set of M keywords in \mathcal{O}'_B , we have $U(\mathcal{O}_M) \geq \frac{M}{|\mathcal{O}'_B|}U(\mathcal{O}_B)$. As shown in [19], Alg 3 returns a $(1 - e^{-1})$ approximation to $U(\mathcal{O}_M)$, and thus a $\frac{M}{|\mathcal{O}'_B|}(1 - e^{-1})^2$ approximation to $U(\mathcal{O})$. \square

We note that, although the approximation rate is lower than *MLC* in theory, as our experiments will show, our algorithm is very accurate in practice. Intuitively, if the budget is small, since keywords usually have large volumes and the budget constraint is easily to be violated, our algorithm rarely goes to the second routine. Even if it goes to the second routine, $|\mathcal{O}'_B|$ is not much larger than M . If the budget is large, our algorithm first finds a large set of useful keywords from candidates and then selects M -best keywords from those useful ones, which will perform similarly as the M -best keywords selected from all candidates without the budget constraint. We can also improve the approximation rate. Since we can estimate it with Theorem 5.3, for the rare cases that have rates lower than a threshold $C(1 - e^{-1})$, where C is a constant, we can call *MLC* [14] as backup to return accurate results. Thus, our algorithm can have an approximation rate of $C(1 - e^{-1})$.

Approximation Rate with Samples We now discuss the approximation rate for the case that a set of samples S instead of the entire set of tweets T is used. Our above discussion assumes that S is equal to T , but S is usually a subset of T in practice. We derive the following theorem to estimate how well a solution derived from a set of samples S performs on the entire set of tweets T .

THEOREM 5.4. *Let K_a be the set of keywords derived by Alg. 5 with a set of samples S , and K_b the optimal solution for all the tweets T . For an error percentile α , with $1 - \alpha$ confidence level, K_a achieves a guaranteed approximate rate of K_b . It is $\gamma \frac{\tilde{U}(K_a) - z_{1-\alpha/2} \sqrt{\tilde{U}(K_a)(|S| - \tilde{U}(K_a))}}{\tilde{U}(K_a) + z_{1-\alpha/2} \sqrt{\tilde{U}(K_a)(\gamma|S| - \tilde{U}(K_a))}}$, when $\tilde{U}(K_a)/\gamma < |S|$, and $\tilde{U}(K_a)/|S| - z_{1-\alpha/2} \sqrt{(\tilde{U}(K_a)/|S| - (\tilde{U}(K_a)/|S|)^2)/|S|}$ otherwise, where γ is the approximation rate of our algorithm, and $\tilde{U}(K_a)$ is the usefulness of K_a in S .*

We prove the theorem by deriving the lower bound of $U(K_a)$ and the upper bound of $U(K_b)$ on T based on Theorem 4.1 and 5.3. Due to space limits, we leave the proof in our report [16].

This theorem is useful, as it 1) provides a theoretical guarantee for applying the approximated solution derived from samples to the entire set and 2) gives a formula for estimating the lower bound of the performance of our solution, which can help us to understand the performance and adjust the number of samples accordingly.

6. EXPERIMENTS

6.1 Experiment Setup

Experiment Settings To fully evaluate ATM, we conduct experiments in the following settings.

Fixed Corpus We first conduct experiments on a pre-crawled Twitter corpus T to fully evaluate ATM (and other baselines). The corpus contains 5 million English tweets with their meta data (e.g., time stamp). We collect them with the *sample* API in Aug 2012. In this setting, we assume that T is all the tweets, and select keywords to cover target tweets in T . We use a fixed corpus instead of the Twitter stream for two reasons. First, with a fixed corpus, to which

we have complete access, we can evaluate ATM with different configurations (e.g., different sets of samples). Second, with a fixed corpus, we can isolate the dynamics of the Twitter stream and thus compare experiments executed at different time.

We construct candidate keywords K based on all unigrams in T . To get meaningful keywords, we remove stop words (e.g., “the”), common Twitter words (e.g., “rt”, which means retweet), and infrequent words (e.g., misspelled words) that appear less than 10 times. We note that this step is optional. In our experiments, while ATM performs similarly without this step, this step greatly improves baseline methods. Finally, K contains 23859 keywords.

Twitter Stream We then conduct experiments on the Twitter stream. Although we cannot fully evaluate ATM on the stream due to our limited access (e.g., we cannot compare many algorithms simultaneously, as Twitter limits the number of simultaneous connections for a user), the experiments are important to show ATM’s performance in practice. In this setting, we monitor target tweets iteratively. In each iteration, we sample tweets from the stream, select keywords based on the samples, and track target tweets with the keywords. In every iteration, we update candidate keywords K via adding all non-stop terms in the samples of the iteration. We tune the iteration length l from 30 mins to 4 hours and use the best one.

Classifiers To show that ATM works for any classifier, we evaluate it with classifiers of two topics, 1) *crime/disaster* [15, 25] and 2) *sport*. We obtain a classifier f of a topic in the following steps. First, we define different types of features (refer [15] for detailed features), including 1) words and 2) social signals (e.g., whether a tweet is from a news agent). Then, we label 3000 tweets for training, and train classifiers with different models (i.e., Naive Bayes (NB) and SVM). Finally, we select the most accurate one to use. The accuracy is 83% for *crime/disaster* and 85% for *sport*.

To further show that ATM is effective for any classifier, we also evaluate ATM (and other baselines) using different classifiers for *crime/disaster*. Here, we emphasize that our focus is not designing classifiers. Instead, we aim to show that ATM can take any classifier as input and monitor target tweets for it.

Baseline Methods To show that ATM advances existing methods, we compare it with three kinds of baseline methods.

- **BaseS** monitors target tweets for a topic using the *sample* API without any keyword. It is used in many existing social media based systems [18, 22]. However, as it samples 1% of tweets, it only retrieves about 1% of target tweets. We use it as a baseline to motivate the need for topic-focused monitoring.
- **BaseM** monitors target tweets for a topic using the *filter* API with a set of *manually* selected keywords. It is the most commonly used approach for focused monitoring [25, 30]. However, as we have discussed in Sec. 1, it has many limitations. We evaluate it to show its limitations and motivate our *automatic* approach. In our experiments, we obtain the keywords by asking 10 cs students to work together and select a ranked list of 20 keywords for each topic. We show them in our case studies.
- **BaseH** monitors target tweets for a topic using the *filter* API with a set of *heuristically* selected keywords. Here, we compare three heuristic methods proposed in the literature. We use *BaseH* to refer all the three methods.
 - **BH-1** is proposed to select keywords for monitoring target tweets for a classifier [15]. It weighs a keyword k according to $\frac{|C(k)| + \alpha}{|V(k)| - |C(k)| + \beta}$, where α and β are priors to penalize rare keywords, and selects M keywords according to their weights. We tune α and β from 1 to 200, and use the best values.

- *BH-2* is a probabilistic method [8] for finding relevant hashtags from relevant tweets of a topic. We use it to select keywords from target tweets of a classifier. Specifically, it estimates a language model θ_R (i.e., a multinomial distribution over keywords K) for target tweets R , and a language model θ_k for each keyword k based on the tweets containing k , and ranks k according to the negative KL divergence between θ_R and θ_k , denoted as $-D_{KL}(\theta_R||\theta_k)$.
- *BH-3*, called Robertson-Sparck-Jones weight, is proposed to select keywords for finding relevant documents for a query [24], and has been widely used for finding relevant keywords in other settings [1]. Specifically, it weighs a keyword k according to $\log \frac{(|C(k)|+0.5)/(|R|-|C(k)|+0.5)}{(|V(k)|-|R|+|C(k)|+0.5)}$.

Our Configurations To fully evaluate ATM, we test ATM with different configurations. First, to show that ATM works for different cardinality (M) and budget (B) constraints, we evaluate ATM on different M and B . Second, to validate that ATM collects unbiased samples, we evaluate ATM on the samples collected by three sampling methods, 1) standard *uniform* sampling (ATMu), 2) *biased* sampling (ATMb), and 3) our *random walk* based sampling (ATMr). ATMb uses the *filter* API with a set of randomly selected keywords to get samples, so the samples are biased to the tweets containing those keywords. Third, to show the advantages of our *iterative* framework, we compare ATM with a static approach (ATMs) and evaluate ATM with different iteration lengths (l).

Measure To measure the effectiveness of a method, we report the “coverage” of its selected keywords K' . In the fixed corpus setting, since the total number of target tweets is known, we report the percentage of target tweets covered by K' , named as *c-rate*. In the Twitter stream setting, since we do not have the total number of target tweets to normalize to, we report the number of target tweets covered by K' , named as *c-size*. We also report the number of tweets collected by K' , named as *p-cost*, to measure its post-processing costs $P(K')$. Here, we clarify that since our goal is to maximize the number of target tweets covered by K' under two cost constraints (i.e., cardinality M and budget B), *c-rate* (or *c-size*), which represents the “recall” in IR, is the the most meaningful measure in our setting. Other measures like “precision” (i.e., the percentage of target tweets in the collected tweets) are not suitable, because algorithms with high precisions may not fully utilize B budgets with M keywords and thus collect only few target tweets, which are not desirable for our problem. In addition, to evaluate the efficiency of a method, we report its running time. Each experiment is repeated for 5 times and the average time is reported.

6.2 Experiment Results

In this section, we first fully evaluate ATM and the baselines on the fixed corpus setting. Then, we show their performances on the Twitter stream setting. Finally, we give some case studies.

6.2.1 Fixed Corpus Setting

In this setting, we conduct the following experiments to fully evaluate ATM. First, we compare ATM with the baselines to show that 1) ATM *outperforms* all the baselines. Second, we evaluate ATM with *additional* configurations, including different constraints, sampling methods and iteration lengths, to show that ATM can 2) handle *various constraints*, 3) collect *unbiased* samples, and 4) take advantages of *iterations*. Third, we report the efficiency of ATM to show that 5) ATM is *efficient*.

To rule out the impacts of different samples or iteration lengths in selecting keywords K' , we estimate $U(K')$ and $P(K')$ based on the entire T for most of the experiments, except those experiments that compare sampling methods or iteration lengths.

Method	<i>BaseS</i>	<i>BaseM</i>	<i>BH-1</i>	<i>BH-2</i>	<i>BH-3</i>	ATM
C-Rate	0.01	0.41	0.58	0.65	0.33	0.83
P-Cost	50000	187102	31417	259288	16289	47727

Figure 2: ATM vs. Baselines for Crime/Disaster

ATM vs. Baselines First, we show the performances of ATM and the baselines for *crime/disaster* in Fig. 2. Here, we set M to 20, as *BaseM* only selects 20 keywords. Further, since all the baselines do not consider the budget constraint, we set B to a large value (i.e., 5 million, which is the corpus size) to reduce the impacts of the budget constraint for ATM. This configuration represents a very useful setting, which selects M keywords with a large budget B .

We have the following observations. First, *BaseS* performs the worst, as it randomly samples only 1% of all tweets. It clearly shows that monitoring target tweets based on random samples is far from enough and we should use a topic-focused approach. Second, *BaseM* greatly improves *BaseS*. It demonstrates the advantage of monitoring target tweets with well selected keywords. Third, *BH-1* and *BH-2* further improve *BaseM*. It suggests that it is possible to select good keywords automatically. Here, *BH-3* performs worse than *BaseM*, because its heuristic is biased to specific keywords that appear and only appear in target tweets for a few times. *BH-1* uses α and β to punish those keywords and improves *BH-3*. *BH-2* further improves *BH-1*, as it uses the similarity between the language models of the tweets associated a keyword and target tweets to select general and useful keywords. Fourth, ATM is the best, as it is designed to find the *optimal set* of keywords that together have the maximum coverage of target tweets.

We note that, while this configuration has a large budget and *p-cost* is not our primary concern, ATM actually achieves the best performance with reasonable costs and costs much less than the best baseline *BH-2*.

Classification Function Next, we validate that ATM outperforms the baselines for any given classifier f . Here, we use the same M and B as the previous experiment.

We first show the performances of ATM and the baselines for *sport* in Fig. 3. The results confirm the above findings. E.g., ATM significantly outperforms all the baselines.

Further, we compare their performances for four different classifiers of *crime/disaster* in Fig. 4. The classifiers are trained with two models (NB and SVM) and two sets of features (word (W) and social (S) signals). Their accuracies are also presented. The results show that ATM performs the *best* for all the classifiers. Specifically, for *NB-W* and *SVM-W*, which only use word signals, ATM covers most target tweets with only 20 keywords, as it successfully reveals the important keyword signals used by the classifiers. For *NB-W+S* and *SVM-W+S*, which use additional social signals to accurately determine target tweets, ATM might not cover all target tweets with only 20 keywords but still performs much better than the baselines. Note that, as different classifiers predict target tweets differently, it is meaningless to compare performances across them.

Thus, we can safely conclude that ATM performs the best for any given f . In the rest of experiments, we use the most accurate classifier for *crime/disaster* as the given f .

Cardinality Constraints Next, we demonstrate that ATM outperforms the baselines for different constraints. Since the baselines cannot model the budget constraint, we evaluate them with different cardinality constraints (M) in Fig. 5. ATM outperforms the baselines for any M . Specifically, 1) ATM selects keywords of any large size, while *BaseM* uses a limited number of keywords, as it is difficult for human to select many keywords. 2) ATM is better than *BaseH* for any M , because *BaseH* selects keywords individually while ATM optimizes a set of keywords. Here, ATM has less

Method	BaseS	BaseM	BH-1	BH-2	BH-3	ATM
C-Rate	0.01	0.52	0.64	0.69	0.19	0.81

Figure 3: ATM vs. Baselines for Sport

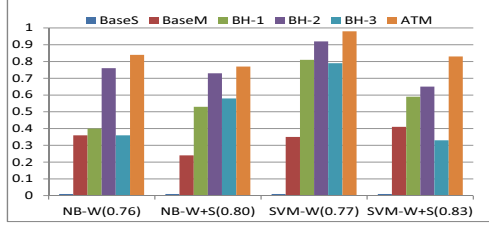


Figure 4: ATM vs. Baselines for Different f .

improvement at $M = 400$ than it does at $M = 20$, because, in this specific experiment, most target tweets can be covered by 50 keywords (ATM has 98% c -rate at $M = 50$), and it is possible for *BaseH* to find these 50 keywords at top 400. We emphasize that, at $M = 400$, ATM is still better than *BaseH*, and is much better than *BaseH* at $M = 50$.

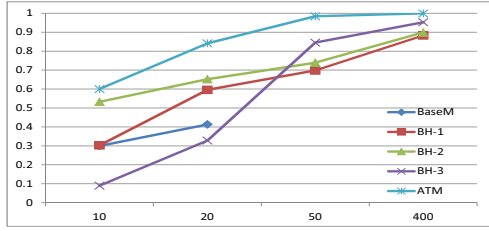


Figure 5: ATM vs. Baselines for Different M

Different Configurations Then, we fully evaluate ATM with different configurations.

Cardinality Constraints First, we evaluate ATM with different cardinality constraints M (from 10 to 400) and a small budget constraint B (30K) in Fig. 6. This experiment is different from the previous one as it uses a small B to represent the case that we have to limit post-processing costs. The results demonstrate that ATM handles cardinality constraints well with a small budget B . ATM’s c -rate increases as M increases, since ATM can take advantage of additional keywords, but it does not change much from $M = 50$ to $M = 400$, as the selected keywords reach the bottleneck of B .

Budget Constraints Second, we show the results of ATM with different budget constraints B (from 20K to 50K) and a moderate M (20) in Fig. 7. The results show that ATM can handle budget constraints well. Specifically, its p -costs are all under the given B and its c -rate increases as B increases.

Sampling Algorithm Third, we evaluate our sampling algorithm. To enable evaluation, we simulate the *sample* and *filter* APIs in the fixed corpus according to their specifications described in Sec 3.

In the first experiment, we use three sampling algorithms, uniform sampling (ATMu), biased sampling (ATMb), and our random walk based sampling (ATMr) to collect different numbers of samples from T , and report the performances of the keywords selected based on them in Fig. 8. We set $M=20$, and B to a large value (*i.e.*, the corpus size). As we will discuss below, we set the parameter λ of ATMr to 0.1. From the results, we obtain three observations. 1) the performance of ATMu increases as the number of samples increases, which validates that we need *sufficient* samples for accurate estimation. 2) ATMu outperforms ATMb significantly on different numbers of samples, which validates that we need *unbiased* samples for estimation. 3) ATMr performs similarly to ATMu, which suggests that ATMr is a *uniform* sampler like ATMu.

In the second experiment, we focus on ATMr. We evaluate the impacts of the parameter λ , which controls how likely ATMr calls

M	10	20	50	400
C-Rate	0.37	0.48	0.57	0.58
P-Cost	14429	23650	28343	29999

Figure 6: ATM with Different M and $B = 30000$

B	20000	30000	40000	50000
C-Rate	0.39	0.48	0.62	0.75
P-Cost	17983	23650	32923	41856

Figure 7: ATM with Different B and $M = 20$

the *sample* and *filter* APIs. We use ATMr to collect 50K samples with different λ . First, we report the c -rates of the keywords selected based on different samples in Fig 9 (the first row). The results show that, with different λ , ATMr achieves similar c -rates and performs similarly to a uniform sampler ATMu, whose c -rate is 0.748. Then, we report the numbers of *additional* samples (among the 50K samples collected by ATMr with different λ) in the second row of Fig 9. Here, the additional samples are those that are not collected by the *sample* API. The results show that the number of additional samples decreases as λ increases. Thus, we should set λ to a small number (*i.e.*, 0.1) to collect many additional samples with ATMr.

Iteration Lengths Fourth, we evaluate ATM with different iteration lengths l and compare ATM with static approaches. To enable evaluation, we partition our corpus into iterations according to tweets’ time stamps. Our corpus contains about 80 consecutive hours. Like in the Twitter stream, we select keywords based on the tweets in the i^{th} iteration and use the keywords to monitor tweets in the $i + 1^{th}$ iteration. We set M to 20 and B to the corpus size of an iteration.

Fig. 10 shows the overall c -rates of ATM with different l . The results validate our analysis in Sec. 3. When l is small (*e.g.*, 0.1 hour), the c -rate of ATM is low, because there are not sufficient samples for accurate estimation in short iterations. When l becomes very large (*e.g.*, 8 hours), the performance decreases, because long iterations cannot capture the dynamics of the Twitter stream well. ATM performs the best when l is 2 hours.

We also compare ATM with two static approaches, *BaseM*, which keeps using the manually selected keywords, and *ATMs*, which keeps using the keywords selected by our selection algorithm based on the tweets of the first iteration. We set l as 2 hours. The c -rates of *BaseM* and *ATMs* are 0.41 and 0.76. ATM greatly improves them by 46% and 11% respectively. The results demonstrate that we should update keywords iteratively to capture the dynamics of the Twitter stream. We clarify that the performance of *ATMs* is different from ATM in Fig. 2, since it does not use the entire corpus, which is not available in practice, to select keywords.

Efficiency Finally, we show the efficiency of ATM on a moderate computer (4GB Memory and Intel i7-2640M 2.8Ghz CPU). Since the efficiency of our sampling algorithm depends on Twitter APIs, we evaluate it in the Twitter stream setting and focus on the selection algorithm here.

First, we report the efficiency of ATM and the baselines in Fig. 11. Since *BaseS* does not select keywords and *BaseM* selects keywords manually, we compare ATM with *BaseH*. Here, we set M and B

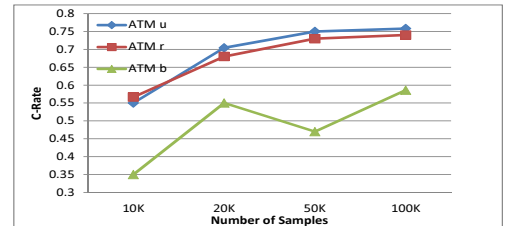


Figure 8: Different Samples with $M = 20, B = 50K$

λ	0.1	0.5	1.0	2.0	3.0	4.0
C-Rate	0.743	0.747	0.745	0.756	0.746	0.742
Samples	48106	42963	40180	37365	33534	31254

Figure 9: ATM with Different λ

Length (hours)	0.1	0.5	1	2	4	8	12
C-Rate	0.58	0.76	0.79	0.87	0.84	0.82	0.78

Figure 10: ATM with Different Iteration Lengths

as the first experiment. The results show that 1) ATM is efficient, which takes only 23 seconds to process a large corpus with 5M tweets and 26K candidate terms, and 2) while ATM is less efficient than *BH-1* and *BH-3*, it is much more efficient than *BH-2*, which is the best baseline in Fig. 2. *BH-1* and *BH-3* are more efficient than ATM because they measure keywords’ weights only once but ATM updates the weights iteratively. ATM is more efficient than *BH-2*, because ATM weighs keywords with an easy-to-compute measure but *BH-2* uses a complex formula (*i.e.*, the KL divergence).

Then, we show the efficiency of ATM with different constraints in Fig. 12. We do not evaluate *BaseH*, as they cannot model budget constraints and have similar efficiency for different cardinality constraints (*i.e.*, *BaseH* computes the weights of keywords only once). First, we analyze the efficiency with different B and a fixed M . The results show that 1) the running time increases as B increases, since ATM uses additional loops in its first step (Alg. 4) to select keywords when B increases, and 2) such increases are sub-linear, because each selected keyword can take many budgets instead of one. Thus, ATM runs efficiently when B is small, and is scalable for a large B . Second, we analyze the efficiency with different M and a fixed B . The running time increases insignificantly as M increases, because, after running the first step (Alg. 4), only a limited number of keywords are selected, and the second step (Alg. 3) of ATM takes a small amount of time to select M keywords from them. Thus, ATM works efficiently for different M .

Further, we show ATM’s efficiency on corpora of different sizes in Fig. 13. We set B to the corpus size and $M = 20$. The results show that the running time increases linearly with the size and ATM only takes seconds for processing 400K tweets. Thus, ATM is efficient and scalable for a big corpus like the Twitter stream.

6.2.2 Twitter Stream Setting

We first report ATM’s effectiveness on the Twitter stream for *crime/disaster* to demonstrate that ATM is effective in practice. Here, we tune the iteration length l from 30 mins to 4 hours and select the best l (*i.e.*, 2 hours) to use. We set M to 20 and B to 140K (the number of the tweets collected by *BaseS* (*i.e.*, the *sample* API) in an iteration). We note that, since we have evaluated the impacts of different l in the previous setting (Fig. 10), we only report the performance of the best l here. Fig. 14 shows their average *c-sizes* and *p-costs* per hour. The results confirm our findings from the previous setting. *E.g.*, ATM has a large improvement over all the baselines and it costs even less than *BaseM* and *BH-2*. *BH-3* is low, because it only selects specific keywords.

We then evaluate ATM’s efficiency on the Twitter stream. Since we have evaluated the selection algorithm in the previous setting (Fig. 11) and it works the same way, we focus on the sampling algorithm ATM. We compare ATM with the only available random sampling method for the Twitter stream (*i.e.*, the *sample* API). Fig. 15 shows how many *additional* samples (besides what returned by calling the *sample* API from a single machine) each method col-

Method	<i>BH-1</i>	<i>BH-2</i>	<i>BH-3</i>	ATM
Time (sec)	1.67	247.12	1.68	23.68

Figure 11: Efficiency of ATM and Baselines.

$M = 20$	$B = 20K$	$B = 40K$	$B = 80K$	$B = 120K$
Time (sec)	1.65	1.94	2.97	3.32
$B = 40K$	$M = 10$	$M = 20$	$M = 50$	$M = 400$
Time (sec)	1.90	1.94	1.96	2.04

Figure 12: Efficiency of ATM with Different M and B

Size	100k	200k	300k	400k
Time (sec)	4.66	5.42	6.19	6.69

Figure 13: Efficiency of ATM on Different Sizes of Samples

lects using different hours. The results show that 1) running ATM on a single machine collects about 30K *additional* samples per hour, which speeds up the *sample* API by 1.4 times (the *sample* API returns about 70K samples per hour), 2) calling the *sample* API from different machines (*i.e.*, 2* *sample*) does not provide any additional sample, and 3) running ATM in parallel can scale up the efficiency (*e.g.*, 2*ATM collect 1.96 times as many additional samples as ATM does), since multiple ATM will choose keywords randomly and collect different samples. The results demonstrate that our algorithm is beyond the limit of the *sample* API. It can help to collect sufficient samples in a short iteration or make an iteration shorter to better capture the dynamics of the stream. We note that our implementation follows all Twitter APIs service’s rules [29]. (*e.g.*, an instance sends an API request about every 25 seconds).

6.2.3 Case Study

We first give the top five keywords selected by each method in the fixed corpus setting in Fig. 16. We can see that all the methods choose topic-related keywords (*e.g.*, “traffic”, “kill”). As all keywords look meaningful, it is difficult for human to select the optimal set, which motivates our optimization based approach. We can also find why *BH-2* is better than *BH-3*. *BH-2* selects general and useful keywords (*e.g.*, “kill”), while *BH-3* selects specific keywords (*e.g.*, “shoplifter”). In addition, the results illustrate that ATM indeed performs the best. *E.g.*, it ranks “traffic”, which is the most useful keyword in the corpus, at the top.

Fig. 16 also gives the keywords selected by each method based on the Twitter stream setting (on 10/30/2012). First, we can see that ATM keeps keywords *up-to-date* via selecting keywords based on recent tweets. It rates “hurricane” as the top one, since hurricane Sandy hit New York in late Oct, and rates “earthquake” in top 5, as an earthquake struck Canada on Oct 28. All such keywords can hardly be predicated by experts or discovered by a static approach. Second, ATM selects more meaningful words than the baselines. *E.g.*, the words selected by *BaseH* are less informative (*e.g.*, *BH-1* uses “flood” instead of “hurricane”) and less complete (*e.g.*, *BH-2* misses “earthquake”) than those selected by ATM.

Further, we show how ATM updates keywords iteratively during a one-day period (*i.e.*, 05/09/2013). We set the iteration length to 2 hours and select 20 keywords every iteration. Here, we use iterations 8-11 as examples. In Fig. 17, the second row shows the percentages of *new* keywords in each iteration, and the third row gives examples of newly added (+) and retired (-) keywords in each iteration. We can clearly see that more than 20% keywords are updated to capture new content. *E.g.*, as users frequently discuss “heroin” related news (*e.g.*, “cops look to link heroin busts”) initially, “heroin” is used. After four hours, when users talk more about “tsunami” (*e.g.*, “tsunami hit Malaysia”), “tsunami” is picked.

7. CONCLUSION

Method	<i>BaseS</i>	<i>BaseM</i>	<i>BH-1</i>	<i>BH-2</i>	<i>BH-3</i>	ATM
C-Size	309	11962	4292	12373	1564	17760
P-Cost	70291	33750	10804	38349	3628	33186

Figure 14: ATM vs. Baselines on Twitter Stream

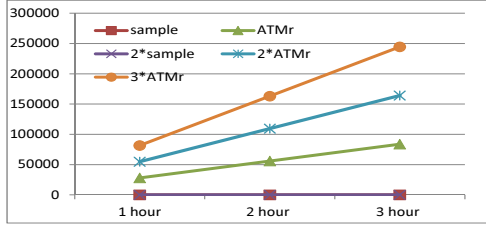


Figure 15: Efficiency of Sampling Algorithms

BaseM	BH-1	BH-2	BH-3	ATM
Fixed Corpus				
kill	burglary	fire	publicity	traffic
shoot	suspect	traffic	shoplifter	kill
fire	hazard	kill	warning	rob
traffic	warning	police	robbery	suspect
police	traffic	warning	traffic	firefighter
Twitter Stream				
kill	warning	hurricane	severe	hurricane
shoot	flood	warning	hazard	kill
fire	murder	flood	warning	fire
traffic	rob	sandy	assault	earthquake
police	cocaine	robbery	firefighter	traffic

Figure 16: Examples of ATM and Baselines

In this paper, we study the task of monitoring target tweets for a topic with Twitter APIs, which is important to many social media based applications. Specifically, we make the following contributions to the task. First, we propose ATM framework, which enables monitoring target tweets in an optimal and continuous way, and abstract two interesting problems in ATM. Second, for the tweet sampling problem, we develop a sampling algorithm, which enables collecting additional random tweets from the Twitter stream with the limited and biased APIs. The algorithm is useful for many settings that need more than 1% tweets (e.g., estimating some sparse properties of the Twitter stream). Third, for the keyword selection problem, we develop an efficient algorithm, which finds a set of keywords that have a near-optimal coverage under two constraints in polynomial time. Forth, we conduct extensive experiments to evaluate ATM and demonstrate that ATM covers most target tweets for a topic and greatly improves all the baseline methods.

As our further work, we plan to study how to automatically set and adjust the iteration length of ATM. Specifically, we would like to develop a model to capture and infer the dynamic changes of target tweets and adjust the iteration length accordingly.

8. REFERENCES

- [1] E. Agichtein and L. Gravano. Querying text databases for efficient information extraction. In *ICDE*, pages 113–124, 2003.
- [2] F. R. Bach. Structured sparsity-inducing norms through submodular functions. In *NIPS*, pages 118–126, 2010.
- [3] Z. Bar-Yossef and M. Gurevich. Random sampling from a search engine’s index. *J. ACM*, 55(5):24:1–24:74, Nov. 2008.
- [4] D. M. Blei and J. D. McAuliffe. Supervised topic models. In *NIPS*, pages 121–128, 2007.
- [5] D. M. Blei, A. Ng, and M. Jordan. Latent dirichlet allocation. *JMLR*, 3:993–1022, 2003.
- [6] M. Boanjak, E. Oliveira, J. Martins, E. Mendes Rodrigues, and L. Sarmiento. Twitterrecho: a distributed focused crawler to support open research with twitter data. In *WWW Companion*, pages 1233–1240, 2012.
- [7] S. Chakrabarti, M. van den Berg, and B. Dom. Focused crawling: A new approach to topic-specific web resource discovery. *Computer Networks*, 31(11-16):1623–1640, 1999.
- [8] M. Efron. Hashtag retrieval in a microblogging environment. In *SIGIR*, pages 787–788, 2010.
- [9] T. Hofmann. Probabilistic latent semantic analysis. In *UAI*, 1999.
- [10] M. Hurst and A. Maykov. Social streams blog crawler. In *ICDE*, pages 1615–1618, 2009.
- [11] P. G. Ipeirotis, L. Gravano, and M. Sahami. Qprober: A system for automatic classification of hidden-web databases. *ACM TOIS*, 21:2003, 2003.
- [12] L. Katzir, E. Liberty, and O. Somekh. Estimating sizes of social networks via biased sampling. In *WWW*, pages 597–606, 2011.
- [13] D. Kempe, J. Kleinberg, and E. Tardos. Maximizing the spread of influence through a social network. In *KDD*, pages 137–146, 2003.
- [14] A. Kulik, H. Shachnai, and T. Tamir. Maximizing submodular set functions subject to multiple linear constraints. In *SODA*, pages 545–554, 2009.
- [15] R. Li, K. H. Lei, R. Khadiwala, and K. C.-C. Chang. Tetas: A twitter-based event detection and analysis system. In *ICDE*, pages 1273–1276, 2012.
- [16] R. Li, S. Wang, and K. C.-C. Chang. An automatic topic-focused monitor for twitter stream. <https://dl.dropboxusercontent.com/u/49890843/VLDBReport.pdf>, June 2013.
- [17] L. Lovsz. Random Walks on Graphs: A Survey, 1993.
- [18] M. Mathioudakis and N. Koudas. Twittermonitor: trend detection over the twitter stream. In *SIGMOD*, pages 1155–1158, 2010.
- [19] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functions. *Mathematical Programming*, 14:265–294, 1978.
- [20] C. Olston and M. Najork. Web crawling. *Foundations and Trends? in Information Retrieval*, 4(3):175–246, 2010.
- [21] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab, November 1999.
- [22] S. Petrović, M. Osborne, and V. Lavrenko. The edinburgh twitter corpus. In *Workshop on Computational Linguistics in a World of Social Media*, pages 25–26, 2010.
- [23] X.-H. Phan, L.-M. Nguyen, and S. Horiguchi. Learning to classify short and sparse text & web with hidden topics from large-scale data collections. In *WWW*, pages 91–100, 2008.
- [24] S. E. Robertson and S. K. Jones. Relevance weighting of search terms. *Journal of the American Society for Information Science*, 27(3):129–146, 1976.
- [25] T. Sakaki, M. Okazaki, and Y. Matsuo. Earthquake shakes twitter users: real-time event detection by social sensors. In *WWW*, pages 851–860, 2010.
- [26] T. Sprenger. Tweettrader.net: Leveraging crowd wisdom in a stock microblogging forum. In *ICWSM*, 2011.
- [27] M. Sviridenko. A note on maximizing a submodular set function subject to a knapsack constraint. *Oper. Res. Lett.*, 32(1):41–43, 2004.
- [28] A. Tumasjan, T. Sprenger, P. Sandner, and I. Welp. Predicting elections with twitter: What 140 characters reveal about political sentiment. In *AAAI conference on weblogs and social media*, pages 178–185, 2010.
- [29] Twitter. Streaming apis documentation. <https://dev.twitter.com/docs/streaming-apis>, 2012.
- [30] X. Wang, F. Wei, X. Liu, M. Zhou, and M. Zhang. Topic sentiment analysis in twitter: a graph-based hashtag sentiment classification approach. In *CIKM*, pages 1031–1040, 2011.
- [31] Wikipedia. Binomial proportion confidence interval. http://en.wikipedia.org/wiki/Binomial_proportion_confidence_interval, Oct 2012.

Iteration 8 (5:00pm)	Iteration 9 (7:00pm)	Iteration 10 (9:00pm)	Iteration 11 (11:00pm)
25%	20%	25%	20%
flood (+) heroin(+) assault(+) hurricane (-) severe(-) injuries(-)	stabbed (+) earthquake(+) injuries (+) robbed (-) assault(-) police (-)	shot (+) tsunami (+) investigate (+) stabbed(-) heroin(-) injuries(-)	fatal(+) death (+1) injured (+) earthquake (-) brush (-) investigate(-)

Figure 17: Keyword Changes in Each Iteration