What to Track on the Twitter Streaming API? A Knapsack Bandits Approach to Dynamically Update the Search Terms

Sumeet Kumar, Kathleen M. Carley
School of Computer Science
Carnegie Mellon University 5000 Forbes Ave, Pittsburgh, PA 15213, USA
Email: {sumeetku@cs.cmu.edu, kathleen.carley@cs.cmu.edu}

***Abstract*—We use Twitter streaming API for many purposes like monitoring brands and discovering events. Because Twitter Streaming API only allows tracking words (commonly called 'search-terms'), the data collection goal needs to be formulated in terms of search terms. Twitter limits the number of search terms that can be tracked using the API, and the number of tweets retrieved per search-term depends on the terms being tracked. Therefore it's crucial to use a small set of highly relevant terms for tracking.**

Because social media is very dynamic and conversations evolve fast, the search terms that are relevant now might be less useful in as short of time as an hour. Manual monitoring of such discussions to update the search terms is cumbersome, error-prone and expensive. Can we have an algorithm to update the search terms based on the goals of the dataset collection? Taking inspiration from the knapsack bandits problem that effectively handle exploration (new search terms to explore) and exploitation (keep using useful search terms) when resources (network bandwidth, disk capacity or number of search terms) are constrained, we propose a new approach to dynamically update the search terms based on the goals of the data collection.

***Index Terms*—Twitter Data Collection, Twitter Streaming API, Search Terms**

I. INTRODUCTION

Imagine a disaster scenario. An earthquake hits the city of Anchorage. As people in Anchorage start responding to the event, many people start to tweet about their situation. Many research agencies and disaster response teams monitor Twitter to find such tweets and to respond to them as soon as possible. The standard way to monitor events on Twitter is to use the Twitter streaming API that allows tracking search terms. The streaming API is limited to approximately 1% (or 10% for the paid service) of Twitter data, and the proportion of tweets generated to tweets collected is even less if we track trendy terms [4]. Therefore, despite the best efforts, in events like an earthquake, often a large fraction of useful tweets do not reach the agencies because they do not contain the exact search terms which the agencies are monitoring. For example, if the disaster agency is tracking '#earthquake', the agency will miss tweets that contain '#Alaska' (e.g. Tweet in Fig. 1). Their data collection could be improved by adding new relevant search terms as events unfold, e.g. using '#Anchorage' and '#Alaska' soon after the earthquake in Anchorage.

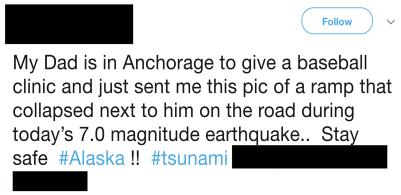

Fig. 1. A sample Tweet sent after an earthquake with some parts blacked out to preserve anonymity. A data collection approach that only tracks '#earthquake' on Twitter streaming API would miss such tweets. In this research, we propose an approach to dynamically update the search terms based on prior collected data. For example, if the goal is to find Tweets relevant to earthquakes then our proposed model will find new relevant search terms (like '#Alaska' in case of Alaska earthquake) to expand the search terms being used.

Twitter has been shown to be useful in disasters [5], [14]. However, that is not its only use case. Twitter remains a popular source of data both for researchers [6], [9] and social-media analytic companies. The common approach to collect tweets is to use a set of words-of-interest as search terms to track on Twitter streaming API. However, as events happen and discussions evolve, the relevant search terms change with time. Thereby, if the search terms are not updated, the old search terms get misaligned with the goals of the data collection. For example, in the earthquake scenario discussed earlier, the new search term '#Alaska' could get dis-aligned with the goal of collecting data on earthquakes in a few days. This begs a question. Is it possible to use the goal of the search in the data collection itself to collect more relevant data over time?

In this research, we suggest a way to dynamically update the terms tracked on Twitter based on the goal of data collection. Rather than formulating the data collection goal as a set of search terms, our approach allows using the goal in a more flexible way (e.g. as a text classifier) and our solution embeds this higher level formulation in the data collection process. We model the search-terms selection problem as a knapsack problem and solve it using standard knapsack and knapsack bandits. The knapsack bandits effectively handle exploration

ASONAM '19, August 27-30, 2019, Vancouver, Canada

ACM ISBN 978-1-4503-6868-1/19/08?/$15.00
http://dx.doi.org/10.1145/3341161.3342919

(new search terms to explore) and exploitation (keep using the most useful search terms) and respect the constraints of data collection such as number of terms that can be used or the amount of data that can be downloaded in a time window. We summarize our main contributions below:

- We suggest ways to collect more relevant Twitter data using the Twitter streaming API. We model the Twitter data collection as a knapsack problem with cost, value and constraints (Section III), and propose two solutions.
- The first solution uses a dynamic programming based knapsack solver that assumes that cost and value of search terms are estimated independently in each time iteration (Section IV).
- The second solution proposes a multi-armed-bandit approach to estimate the cost and value of search terms over multiple time iterations (Section V).
- To the best of our knowledge, this is the first work that suggests a principled approach to dynamically update the search terms while staying relevant to the goal of the data collection. We show the utility of our approach using a real example (Section VI).

Section II provides background on Twitter data collection, highlighting the limitations of the Twitter streaming API. We describe the related prior research that are relevant to this work in Sec. VII. Finally, at the end, we conclude and provide directions for future research.

II. Twitter Data Collection Background

Twitter Streaming API allows three parameters to search the real-time data which are 'follow', 'track' and 'locations' [1]. Here we focus on 'track' as that is commonly used to track a comma-separated list of phrases (e.g. words, mentions, #hashtags). As mentioned earlier, given a topic of interest, the most common approach is to use intuition to come up with a few generic phrases that overlap with the discussions on the topic. For example, if someone is interested in finding information about earthquakes, the phrases to use could be '#earthquake', 'earthquake' or 'earthquake now'. Though this is how Twitter API is commonly used, this approach has a few limitations.

Some phrases are trendy and result in a vast amount of data, much of which is not relevant to the goals of the data collection. Therefore the collected data needs to be filtered later which results in processing and discarding a significant proportion of data. There is one more problem. If multiple search terms are used, the majority of tweets obtained using the API will be from the more commonly used phrases. Searching for more terms results in less number of tweets per search term [4] and, in our experience, the returned data never exceeds around 25GB per day (raw JSON files obtained using Tweepy library [2]) irrespective of the number of search terms used (tested on a computer with 1 Gbps Internet connection speed at the Carnegie Mellon University campus). This limitation of Twitter is not well documented so to better understand it, we used the Twitter Streaming API to track all 195 country names in English.

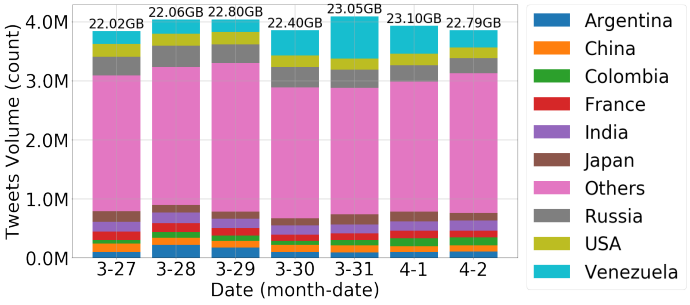

Fig. 2. Tweets volume obtained on different days using all country names as search terms. On y axis, 'M' indicates tweets count in millions. The total data received each day is shown on the top of the day's bar. As we can observe, though there is a wide variation in tweets with certain country-names (e.g. check Venezuela), but still the total volume of tweets has remained between 22 GB and 23.1 GB and the total count is approximately 4 million.

In Fig. 2, we show the volume and the quantity of data obtained by searching 195 countries names for over a week. As we can observe, though there is a wide variation in tweets from specific countries (e.g., check Venezuela), but still the total volume of tweets has remained between 22 GB and 23.1 GB and the total count of tweets is approximately 4 million. This finding confirms the observations in [4] where the authors find that Twitter returns around 2900 tweets per minute when search terms are used with Twitter streaming API. 2900 tweets per minute is 4,176,000 tweets in a day i.e. approximately 4 million. Note that these limits are different while using the Twitter Streaming API without any search terms.

III. Problem Formulation

In collecting Twitter data, there is a cost in collecting data, there is some value of the data collected, and there are some constraints. We describe these next:

1) Cost: Cost in data collection is due to the costs of data streaming (e.g., the internet bandwidth), data storage and/or data processing. Because these costs are proportional to the volume of the data obtained, to keep our model simple, we aggregate the different costs and call the overall cost as w_i, where i is the search term index. Let x_i be the number of tweets retrieved per minute by searching the i^{th} search term. Cost w_i is a function of the amount of data x_i. We expect the cost to be low if the volume of data retrieved is low, but if the data volume x_i is large, the cost should much higher as it reduces the collection of data associated with other search terms (as discussed earlier). Therefore, the cost function should be non-linear and should satisfy the following conditions: a) Cost is proportional to number of tweets collected if the total volume of tweets is low b) If the total volume obtained reaches the maximum limit (2900 per minute),

[1] https://developer.twitter.com/en/docs/tweets/filter-realtime/guides/basic-stream-parameters.html

[2] https://tweepy.readthedocs.io/en/v3.5.0/streaming_how_to.html

then the cost should be very high as we can't get data at any higher rate. Many functions can possibly respect these conditions. We model the cost function as a non-linear logit function (see Fig. 3). As shown in the figure, this function is approximately linear till 2000 tweets per minute and increases rapidly later.

$$w_i(x_i) = \log\left(\frac{(x_i + 2900)/5800}{1 - (x_i + 2900)/5800}\right) \quad (1)$$

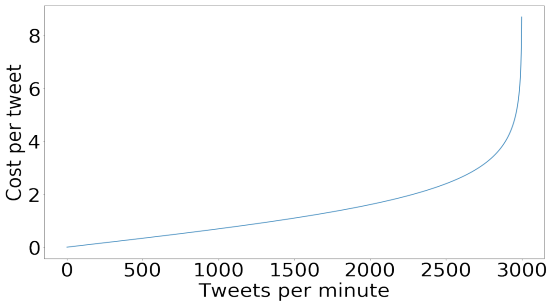

Fig. 3. Cost function plot

The goal of this cost function is to encourage search terms that results in smaller fraction of tweets.

2) Value: This value of the search term i is the mean utility of the data. For example, in the case of an earthquake, the value is a function of the fraction of tweets that are relevant to a real shock. We call this value v_i where again i is the search term index. v_i is estimated based on the goals of the data collection and could be as simple as a text pattern match. For instance, if one is interested in the text of the tweets matching some pattern, we can model the value function as:

$$Value\_Func(t_k) = \begin{cases} 1 & \text{if } t_k \textit{ matches } p \\ 0 & \text{otherwise} \end{cases}$$

and v_i = Mean($Value\_Func(t_k)$) where t_k is a tweet associated with search term with index i.

3) Constraints: While collecting Twitter data, a critical limitation is the total amount of data that we can download. As discussed earlier, it appears that there is a hard limit on the amount of streaming data that can be obtained using a single API connection. We define this constraint as W where $W \approx 2900$ tweets per minute. Thus, $\sum_{i \in I(t)} w_i(x_i(t)) \leq W$ in every iteration t = 0, 1, 2, .., where $I(t)$ is the set of indices of the selected search terms. There are other possible constraints as well e.g. Twitter limits the number of search terms to 400[3]. To satisfy the search term limit, we have: $\sum_{i \in I(t)} 1 \leq 400$.

Using cost, value and constraints, we define our optimization problem as:

[3]https://developer.twitter.com/en/docs/tweets/filter-realtime/overview/statuses-filter.html

$$\max\left(\sum_i \Big[v_i(x_i(t))\Big]\right) \quad (2)$$

subject to:

$$\sum_{i \in I(t)} w_i(x_i(t)) \leq W \quad (3)$$

and

$$\sum_{i \in I(t)} 1 \leq 400 \quad (4)$$

at iteration t = 0, 1, 2, .., where x_i is data collected for the i^{th} search term, and $I(t)$ is the index of all 'search terms' used at time t. The objective is to maximize the expected value at time $t+1$ based on the estimates of v_i and w_i at time t.

Given cost, value and constraints, the standard approach is to model such problems as a knapsack problem (described next).

IV. New Search Terms as a Knapsack Problem

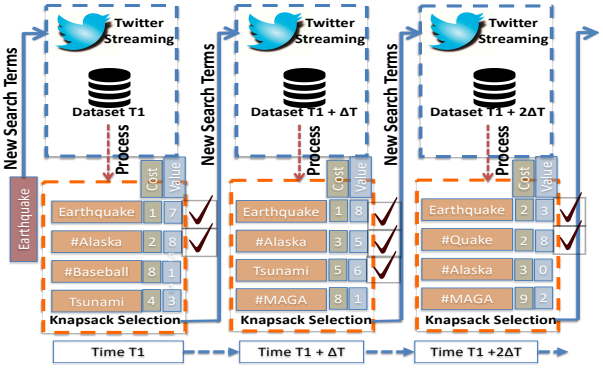

Fig. 4. In every time iteration, a set of search-terms are used to get data from the Twitter Streaming API. The data is then processed to find high frequency terms. For each of these terms, value and cost are estimated (shown with light blue and light green background respectively) based on the tweets associated with each term. Cost and value are then used to find the next set of search-terms using a knapsack solver that also considers the constraints of the data collection. The result of one iteration is used as the search-terms for the next iteration.

We first solve the simpler version of the problem in which cost and value are estimated each iteration. An iteration consists of a small batch of data obtained by connecting to the Twitter streaming API for some time. The time duration of the iteration depends on how often the search terms need to be updated based on the goal of the data collection. For example, in case of an earthquake, since such events are instantaneous, an iteration could be of short time like a few minutes. In contrast, for slow changing goals like political discussions, an iteration could be of larger time duration like 30 minutes. At the end of the iteration, the tweets dataset is processed to find the high-frequency terms which are the potential search-term candidates for the next batch. For each of these terms, value and cost are estimated based on the tweets associated with each term. The search terms are first

filtered to remove any unwanted content (i.e. stop words or pornographic content). Cost and value for the filtered terms are estimated using the cost and value function as described in the last section. Knapsack problems, though NP-hard [10], have many efficient solutions [15]. We use a dynamic programming based knapsack solver to find the next set of search terms [12]. We show the steps in Fig. 4, and summarize the steps as an algorithm which we name as 'Dynamically Update Search Terms' (DUST1) (see Alg. 1).

Algorithm 1 DUST1: Dynamically Update Search Terms

Precondition: $data$ is the tweets collected

```
function DUST1(data)
    terms, terms_tweets_dict ← Process(data)
    utility_scores ← []
    for k ← 0 to len(terms) do
        term ← terms[k]
        term_tweets ← terms_tweets_dict[term]
        value ← Mean(Value_Func((term_tweets))
        cost ← Mean(Cost_Func((term_tweets))
        utility_scores.add((term ,cost, value))
    end for
    search_terms ← KnapsackSolver(utility_scores)
    return search_terms
end function
```

DUST algorithm returns a set of new terms (in addition to seed terms) after each iteration. This helps us with the dynamic update of search terms. However, this approach has two limitations: 1) The approach only considers the current data, thereby ignoring the cost and value estimated in previous iterations. Because the streaming API only consists of a small fraction of the total tweets, it's possible that in a particular iteration, there is no data from a search-term though the search-term is generally useful. 2) The approach does not consider the confidence in estimating the value of the search terms. For example, if a search-term 'X1' had only a single tweet of high value, the mean-value based algorithm is more likely to suggest it when compared to another term 'X2' that has a few hundred tweets, many of which are useful. We improve on these two limitations in the next section.

V. A MAB Approach to Dynamically Update the Search Terms

Here we propose an algorithm that estimates and maintains cost and value of each search term overtime. A common approach to estimate the utility of different options is the *'online controlled testing'*, popularly called A/B testing [8]. Re-looking at our earthquake example, if the options are '#Alaska' and '#Canada', one can wait for certain number of tweets on both the terms to arrive before using A/B test to determine if '#Alaska' is more useful than '#Canada'. However, A/B testing requires large enough sample set to derive the confidence of the benefit of option A over B. The opportunity cost of waiting to get the sample set is high in many cases (like the earthquake example) and we want the more useful options to be picked quickly (i.e. greedily) to get more relevant data. In such situations, the framework of multi-armed bandits is preferred [4]. Therefore, given our goal of greedily exploring more useful search terms, we model the problem as a knapsack-bandit where bandits are used to estimate the value of the search terms and a knapsack solver is used to filter the top search-terms that satisfy the constraints.

In this simple case, the goal in Multi-Armed Bandits (MAB) optimization is to estimate the reward of each arm to find the arm which leads to maximum reward over multiple trials. Such MAB problems can be solved using many different strategies. These strategies attempt to strike a balance between exploration and exploitation in different ways. To suit the MAB paradigm to our problem, we need two changes 1) need to select a set of search-terms that are more useful (in contrast MAB selects the best search-term). 2) selected terms should also satisfy the data collection constraints that we discussed earlier. Therefore, we use MAB only to estimate the value of search-terms over multiple iterations (using two different strategies) and use the knapsack solver to get the final set of search-terms for data collection. The steps of the approach is shown as an algorithm in Alg. 2 where the $MAB\_Strategy$ returns a list of $(term, cost, value)$ tuples.

Algorithm 2 DUST2: Update Search Terms using Bandits

Precondition: $data$ is the tweets collected and $util\_scores\_queue$ is a dictionary of FIFO queues that maintains the costs and values of terms

```
function DUST2(data)
    terms, terms_tweets_dict ← Process(data)
    utility_scores ← []
    for k ← 0 to len(terms) do
        term ← terms[k]
        term_tweets ← terms_tweets_dict[term]
        value ← Mean(Value_Func((term_tweets))
        cost ← Mean(Cost_Func((term_tweets))
        utility_scores_queue[term].enqueue((cost, value))
    end for
    utility_scores = MAB_Strategy(utility_scores_queue)
    search_terms ← KnapsackSolver(utility_scores)
    return search_terms
end function
```

Next, we describe two strategies which we we use for search-terms selection.

1) Mean-k Strategy: Mean-k estimates the mean of a function (value or cost) over last k iterations. Thus, mean-k value of term i at iteration n is be written as:

$$\hat{v}_i(n) = \frac{\sum_{t=n-k}^{n} v_i(x_i(t))}{\sum_{n-k}^{n} 1} \tag{5}$$

[4]https://conversionxl.com/blog/bandit-tests/

where $\hat{v}_i$ is the estimated mean value of search term i over last k iterations.

2) Upper-Confidence-Bound (UCB) Strategy: UCB strategy allows for better exploration by giving higher probability to actions for which reward estimate is not available [1]. Intuitively, UCB uses two criterion 1) try if better candidate i.e. $\hat{v}_i(t)$ is large 2) try if less explored i.e. $N_n(i)$ is small.

$$\tilde{v}_i(n) = \left(\hat{v}_i(n) + c\sqrt{\frac{\ln n}{N_n(i)}}\right) \tag{6}$$

where $\tilde{v}_i(n)$ is the UCB value associated with term i after iteration n, $N_n(i)$ denotes the total number of times i has been selected before iteration n and c is a parameter to control exploration.

DUST2 algorithm is simple but we found some practical limitations while implementing. a) cost and value estimation of some search terms get stale over time. b) large number of search terms needs to be tracked as we get more and more data which slows down the algorithm. To resolve 'a', we use a FIFO queue (data structure) of limited size to store the cost and value of each search term. In addition, if we don't get data for a search term in an iteration, an empty $(cost, value)$ is added for the term to the queue. With this modifications, we are able to get rid of the stale data. But we still keep on aggregating all search terms. To fix 'b', after every iteration, we run a subroutine that removes any search term that contains only empty $(cost, value)$ in the queue.

VI. EXPERIMENT AND RESULTS

We conducted a data-collection experiment for a few weeks to measure the quantity and the relevancy of data collected using our proposed approach. Our goal is to collect tweets that are relevant to earthquake beyond what could be obtained by using the seed terms. To compare different approaches, we collected data in four different ways: 1) Searching with seed terms '#earthquake' and 'earthquake'. 2) Searching using the terms suggested by DUST 1(see Alg. 1) 3) Searching using the terms suggested by DUST2 (see Alg. 2) with mean of cost and value estimation for last 10 iterations. 4) searching using the terms suggested by DUST2 (see Alg. 2) with mean for cost and UCB (strategy) for value estimation using last 10 iterations data. For estimating the value of a tweet, we use a small tweets dataset from prior research [7] that has relevancy labels (relevant vs non-relevant) for a set of tweets related to an earthquake in Nepal. We first removed any words that contain 'Nepal' and use the filtered dataset to build a Support Vector Machine classifier using TF-IDF features that predicts whether a given tweet is relevant or not. Using a separate test set that is around 40% the entire dataset, we found the accuracy of the classifier to be 81%. We use this classifier in our value function as defined below:

$$Value\_Func(t_k) = \begin{cases} 1 & \text{if } EQakeClassifier(t_k) = 1 \\ 0 & \text{otherwise} \end{cases}$$

and v_i = Mean($Value\_Func(t_k)$) where t_k is a tweet associated with search term with index i.

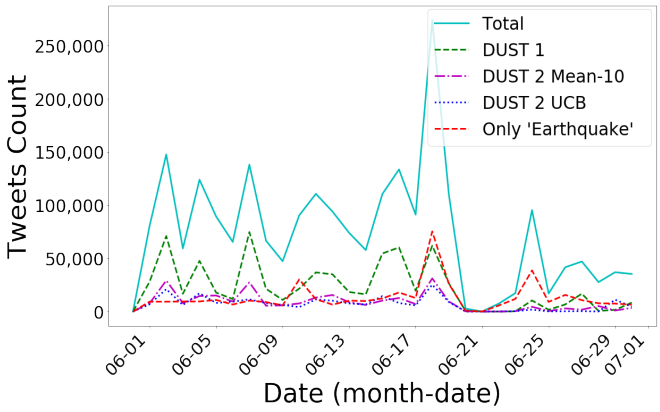

Fig. 5. Results of earthquake data collection using only 'earthquake' like search-terms and our proposed approaches.

Using this value function, we present the amount of tweets obtained by just searching for seed terms (only 'earthquake'), searching for the terms suggest by DUST-1, and by DUST-2 (Mean-10 and UCB strategies) in Fig. 5. As we can see in the plot, on most days, data collected by DUST-1 exceed the data obtained by just searching for seed terms. For the entire time during for the experiment, we estimate that DUST-1 gets 1.71 times the data obtained using the seed terms. For MAB based search, Mean-10 gets 0.65 times and UCB-10 gets 0.53 times additional data. The total data that was found to be relevant using the relevancy classifier is 3.89 times the total data that was collected only using the seed terms (see Fig. 6 for trends).

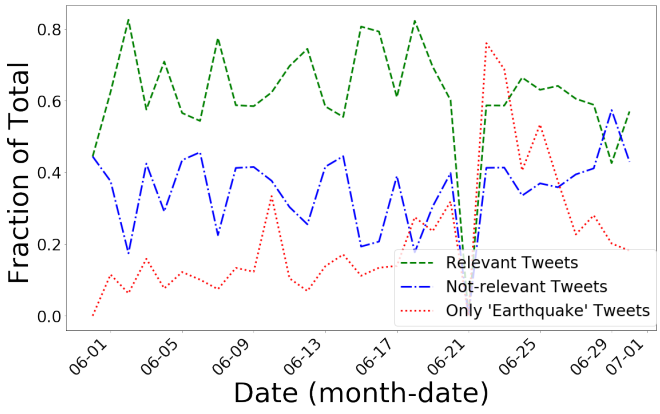

Fig. 6. Relevant vs non-relevant trend of data obtained using our proposed approaches. The relevancy and non-relevancy is determined using a classifier based on a labeled dataset for earthquakes from a prior research.

To summarize, even if we discount the accuracy estimate of the relevancy classifier (which can always be improved with more labeled data), we can still expect the gain of our approach to be over 2x times the data collected using the seed terms.

VII. RELATED WORK

A. Twitter Data Collection and Event Detection

A number of researchers have explored and compared Twitter Streaming APIs ([4], [16]). Campan et al. [4] compares the Twitter Streaming API based on popular and not so popular terms and find that if filtering is used for terms that are not very popular, it's likely that all matching Tweets are provided by Twitter. In contrast, if very popular filtering terms are used, the collected data leads to biased results. Wang et al. [16] used samples obtained from the Spritzer Twitter stream API and Gardenhose Twitter stream API to find that the actual sampling ratios are around 0.95% and 9.6% for Spritzer and Gardenhose respectively. They also suggested that though Spritzer is sufficient when using text terms and URL domains, for hashtags, the small Spritzer sample is not adequate to preserve accurate data statistics. There is also a rich literature on using Twitter for event detection [3]. In most prior work on event detection, the data that was collected apriori. In this work, instead, we suggest a a way to get more data by adding new search terms, and to the best of our knowledge, this is the first work of this type.

B. Multi-Armed Bandit Problems (MAB) and Focused Web Crawling

MABs are commonly used for optimization in noisy environments where there is an exploration (more labels to try) and exploitation (use the best one) trade-off. Many extensions to the standard MAB have been proposed like the contextual-bandit, the collaborative-bandit and the knapsack bandits [2]. In particular, knapsack bandits are useful when both exploration and exploitation incur cost and the total possible cost is constrained [13]. In knapsack bandits, in every iteration, a set of bandit arms are selected that satisfy the knapsack constraints to maximize the value of arms selected in the knapsack. Our problem differs from the previous formulation of knapsack bandits. In the earlier studies, arms are scheduled independently, but in our formulation, a set of arms are scheduled at a time and we use bandits to only estimate the value of arms (search terms) over multiple iterations. In the domain of focused (web) crawling, reinforcement learning has been studied before e.g. to design a web spider [11]. However, to the best of our knowledge, no one has applied our formulation of knapsack-bandits in the context of Twitter data collection earlier.

VIII. CONCLUSION AND FUTURE WORK

In this research, we first show that Twitter limits the amount of data that can be retrieved using their Streaming API to around 4 million tweets in a day. We then propose two novel approaches that respect the constraints on data collection volume and the number of search terms, still get additional data. Given a value function that can quantify the utility of a tweet, our proposed algorithm allows embedding the function in the data collection process itself. Our solution uses the 'search terms' as bandit-arms to find the best set of arms that satisfies the constraints. Using 'earthquake' related data collection as an example, we estimate that our suggested approach could get relevant data that is more than twice the data retrieved by just searching for 'earthquake' and '#earthquake'. In the future, we would like to extend our approach to a distributed system so that multiple machines/processes can coordinate to get more useful data.

ACKNOWLEDGMENTS

This work was supported in part by the ONR Award No. N00014182106, ONR Award No. N0001418SB001 and the Center for Computational Analysis of Social and Organization Systems (CASOS). The views and conclusions contained in this document are those of the authors only.

REFERENCES

[1] P. Auer and R. Ortner. Ucb revisited: Improved regret bounds for the stochastic multi-armed bandit problem. *Periodica Mathematica Hungarica*, 61(1-2):55–65, 2010.

[2] A. Badanidiyuru, R. Kleinberg, and A. Slivkins. Bandits with knapsacks. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 207–216. IEEE, 2013.

[3] H. Becker, M. Naaman, and L. Gravano. Beyond trending topics: Real-world event identification on twitter. In *Fifth international AAAI conference on weblogs and social media*, 2011.

[4] A. Campan, T. Atnafu, T. M. Truta, and J. Nolan. Is data collection through twitter streaming api useful for academic research? In *2018 IEEE International Conference on Big Data (Big Data)*, pages 3638–3643. IEEE, 2018.

[5] A. T. Chatfield and U. Brajawidagda. Twitter early tsunami warning system: A case study in indonesia's natural disaster management. In *2013 46th Hawaii International Conference on System Sciences*, pages 2050–2060. IEEE, 2013.

[6] H. Hirose and L. Wang. Prediction of infectious disease spread using twitter: A case of influenza. In *2012 Fifth International Symposium on Parallel Architectures, Algorithms and Programming*, pages 100–105. IEEE, 2012.

[7] M. Imran, P. Mitra, and C. Castillo. Twitter as a lifeline: Human-annotated twitter corpora for nlp of crisis-related messages. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016)*, Paris, France, may 2016. European Language Resources Association (ELRA).

[8] R. Kohavi, A. Deng, B. Frasca, T. Walker, Y. Xu, and N. Pohlmann. Online controlled experiments at large scale. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1168–1176. ACM, 2013.

[9] S. Kumar and K. M. Carley. Approaches to understanding the motivations behind cyber attacks. In *2016 IEEE Conference on Intelligence and Security Informatics (ISI)*, pages 307–309, Sep. 2016.

[10] M. J. Magazine and M.-S. Chern. A note on approximation schemes for multidimensional knapsack problems. *Mathematics of Operations Research*, 9(2):244–247, 1984.

[11] J. Rennie and A. McCallum. Using reinforcement learning to spider the web efficiently. In *Proceedings of the Sixteenth International Conference on Machine Learning*, ICML '99, pages 335–343, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[12] P. Toth. Dynamic programming algorithms for the zero-one knapsack problem. *Computing*, 25(1):29–45, 1980.

[13] L. Tran-Thanh, A. Chapman, A. Rogers, and N. R. Jennings. Knapsack based optimal policies for budget–limited multi–armed bandits. In *Twenty-Sixth AAAI Conference on Artificial Intelligence*, 2012.

[14] S. Vieweg, A. L. Hughes, K. Starbird, and L. Palen. Microblogging during two natural hazards events: what twitter may contribute to situational awareness. In *Proceedings of the SIGCHI conference on human factors in computing systems*, pages 1079–1088. ACM, 2010.

[15] M. Visée, J. Teghem, M. Pirlot, and E. L. Ulungu. Two-phases method and branch and bound procedures to solve the bi–objective knapsack problem. *Journal of Global Optimization*, 12(2):139–155, 1998.

[16] Y. Wang, J. Callan, and B. Zheng. Should we use the sample? analyzing datasets sampled from twitters stream api. *ACM Transactions on the Web (TWEB)*, 9(3):13, 2015.