# Online Retweet Recommendation with Item Count Limits

Xiaoqi Zhao
Graduate School of Informatics
Kyoto University
Yoshida-Honmachi, Sakyo, Kyoto 606-8501 Japan
Email: x.zhao@dl.kuis.kyoto-u.ac.jp

Keishi Tajima
Graduate School of Informatics
Kyoto University
Yoshida-Honmachi, Sakyo, Kyoto 606-8501 Japan
Email: tajima@i.kyoto-u.ac.jp

***Abstract*—Some Twitter accounts provide information to the followers not by publishing their own tweets but by forwarding (i.e., retweeting) useful information from their friends. These accounts need to select an appropriate number of tweets that match the followers' interests. If they retweet too many or too few tweets, it annoys the followers or degrade the value of the accounts. They also need to retweet them in a timely manner. If they retweet a tweet long after they receive it, the informational value of the tweet may diminish before the followers read it. There is, however, a trade-off between these two requirements. If they select tweets after seeing all the candidates, they can select the best given number of tweets, but in order to provide timely information, they have to decide to (or not to) retweet each tweet before seeing all the following candidates. In order to help the management of such Twitter accounts, we developed a system that reads a sequence of tweets from the friends one by one, and select a given number of (or less) tweets in an online (or near-online) fashion. In this paper, we propose four algorithms for it. Two of them give priority to the timeliness, and make a decision immediately after reading a new tweet by comparing its score with a threshold. The other two give priority to the selection quality, and make a decision after seeing some following tweets: after seeing the following tweets belonging to the same time subinterval or after seeing a fixed number of following tweets. The former two are truly online algorithms and the latter two are near-online algorithms. Our experiment shows that the near-online algorithms achieve high selection quality only with acceptable time delays.**

## I. Introduction

Recently, microblogging services are rapidly growing. Twitter, which is the most popular microblogging service, has over 115 million monthly active registered users, sending 58 million messages (called *tweets*) per day [1]. In Twitter, a user *follows* other users so that all the tweets by them are shown on the user's Twitter interface, called *timeline*. Those who follow a user are called *followers* of the user, and those whom a user follows are called *friends* of the user.

Twitter is not only playing a role of communication media, but also used as a medium for collecting useful real-time information. About 40% of active Twitter users do not submit tweets but just watch other people's tweets [2]. For those users, Twitter is a tool for *real-time journalism* [3].

For those users, however, it is not easy to find useful information buried in the huge number of tweets. To help those users to collect useful information on Twitter, accounts that play the role of "portal sites" on Twitter have appeared. These accounts provide their followers with useful information mainly by *retweeting* (i.e., forwarding) tweets from their friends, rather than tweeting their own original tweets. In this paper, we call those accounts *portal accounts*.

The daily task of the administrators of such portal accounts is to read all tweets from the friends, and select tweets to retweet that best match the followers' interests. They need to select an appropriate number of tweets in each time interval, e.g., in every 24 hours. If they retweet too many tweets, it annoys the followers and make them stop following the accounts. If they retweet too few tweets, it also degrades the value of the accounts as a portal, and they lose the followers.

In addition, they need to retweet tweets in a timely manner. According to [4], the life cycle of a tweet is only 48 hours. If they retweet a tweet long after receiving it, the informational value of the tweet may diminish before the followers read it. Even when the value of a tweet lasts longer, if the followers receive the same tweet from other sources before receiving from the portal account, it degrades the value of the portal. Even when these are not the case, the followers want to receive fresh information. Making the followers feel they are "first to know" the information will add value to the portal account [5]. For these reasons, it is important to reduce the time lag between arrival and retweeting of tweets.

As explained above, portal accounts on Twitter need to satisfy two requirements: selection of an appropriate number of good tweets and prompt forwarding of them. There is, however, a trade-off between these two requirements. If they select tweets after seeing all the candidates in some time interval, e.g. at the end of a day, they can select the best given number of tweets. In order to provide timely information, however, they have to make a decision for each tweet before seeing all the following candidates. Once we retweet a tweet, we cannot withdraw it later even if we find many following tweets are much better than the previous one. We can delete it from Twitter, but some followers may have already read it. These mean that this is a typical environment where we need some online algorithm.

In addition, manually selecting all retweets is a hard task for the administrators of portal accounts. Portal accounts usually have many friends and many followers. The administrators have to monitor a huge number of tweets from the friends for 24 hours a day. They also have to analyze the interests of many followers in order to select tweets best matching their interests. It would be better for the administrators if these procedures are automatated.

In this paper, we propose a system that automates these tweet evaluation and selection processes. Our system reads a sequence of tweets from the friends one by one, evaluate how much each tweet match the followers' interests, and select a given number of tweets (or less if there are not enough tweets). We evaluate a tweet by comparing its feature vector with the feature vector produced from the profiles and tweets of the followers. The selection is done in an online or near-online fashion. When a new tweet arrives, the system decides to retweet it or not immediately, or after seeing some (but not all) following tweets. This system automatically retweets an appropriate number of good tweets within short delays.

For the selection process, we propose and compare four algorithms in this paper. The basic strategies of the four algorithms are summarized as below:

- timeliness-oriented approaches (online)
    1) history-based threshold estimation
    2) stochastic threshold estimation
- selection-quality-oriented approaches (near-online)
    3) selection at every time interval
    4) selection at every $n$ items

The former two algorithms give priority to the timeliness, and make a decision immediately after reading a new tweet. These two algorithms use threshold to make a decision. The first algorithm computes the threshold by taking the average of offline-computed best thresholds for the past time intervals. The second algorithm computes the threshold based on the estimated probability distribution of the values of tweets. The other two give priority to the selection quality, and permits some delay of retweeting. They make a decision after seeing some (but not all) following tweets. In the third algorithm, the system divides time into subintervals, and periodically select tweets at the end of each subinterval from the tweets arrived within that subinterval. In the fourth algorithm, the system periodically selects tweets every time it receives $n$ tweets, where $n$ is computed based on the number of recently received tweets. The former two are truly online algorithms and the latter two are near-online algorithms.

In order to create a dataset for the evaluation of these four algorithms, we selected some real portal accounts on Twitter, and collected the tweets by their friends. In order to infer the interests of the followers of the portal accounts, we collected the profiles of the followers, and also collected tweets by the followers. Our experiments on this data set shows that the two near-online algorithms achieve high selection quality with only short acceptable time delays in retweeting.

The remainder of the paper is organized as follows. In Section II, we review some related research. In Section III, we formally define our problem. In Section IV, we explain our method of evaluating how much a tweet match followers' interests. In Section V, we describe our four algorithms for tweet selection. In Section VI, we evaluate our four algorithms by experiments. Finally, we conclude the paper and address future research directions in Section VII.

## II. Related Work

In this section, we review related work on retweet recommendation, and also some theoretical work on problems related to our problem.

### *A. Retweet Recommendation*

There have been much research on Twitter, such as research on user classification, hot topic detection, recommendation of tweets to read, and friend recommendation. On the other hand, there have been only a little research on retweet recommendation, i.e., recommendation of tweets that the target user may want to share with the followers. The survey paper by Kywe et al. [6] explained that there was no paper about personalized retweet recommendation when the paper was written. The authors, however, also pointed out that the work by Uysal et al. [7] and the work by Nasirifard et al. [8] can be applied to retweet recommendation as explained below.

Uysal et al. [7] developed a classifier to predict the probability that a given user retweets a given tweet. They model the user's past retweets by using four types of features: author-based, tweet-based, content-based and user-based features. This classifier can be used for retweet recommendation.

Nasirifard et al. [8] developed a system that helps various activities on Twitter, and it can determine whether the hashtags appearing in a tweet are relevant to the user's followers. This function can also be used for retweet recommendation.

To the best of our knowledge, the only existing study on personalized retweet recommendation is the study by Wang et al. [9]. Their method ranks tweets by using the information on the past retweeting activities of the target user. The ranking score of a tweet is computed from three kinds of information: (1) how often the target user has retweeted tweets from the sender of the tweet, (2) which tweets the user retweeted or not retweeted in the past, and (3) which retweets by the user in the past had a big impact on the followers.

There have been also studies on the analysis of users' retweeting behavior and prediction of future retweets [10], [11], [12], [13], [14], [15], [16]. Their primary purpose is not recommendation of tweets to retweet, but these results can also be used for retweet recommendation.

For example, Macskassy et al. [12] proposed four models of users' retweeting behavior. The first model simply assumes that a user retweets recent tweets with higher probability than older ones. The other models add some factor to this base model. The second model adds a factor that gives higher probability to tweets from users with whom the target user recently interacted through retweeting or direct messaging. The third model adds a factor that gives higher probability to tweets whose topic is similar to the target user's topic of interest. The fourth model adds a factor that gives higher probability to tweets from users whose interests are similar to that of the target user. The result of their experiment with real Twitter users shows that a combination of more than one model can better explain the behavior of any user, but it also shows that the third model based on the similarity between tweets and user interests works best in average.

On the other hand, Peng et al. [13] used CRFs (Conditional Random Fields) to model the behavior of retweeting users. The model includes three types of features: features defined for each tweet, features defined for each combination of a tweet and a user, and also features defined for each combination of a tweet and two users (e.g., the author of the tweet and a user retweeting it). Some features are related to temporal factors. Their experimental results show that the features defined for a tweet and a user are the most predictive indicators.

Both [12] and [13] conclude that the topic similarity between tweets and user interests is the most important. Following their results, we also use similarity between tweets and user interests for tweet evaluation. [12] and [13], however, construct a model based on users' retweets in the past. It means that their models mainly predict which tweets the users want to retweet. On the other hand, our purpose is to help management of portal accounts by recommending tweets that the followers wants to read. Because of this, we use similarity between tweets and the interests of the followers, instead of similarity between tweets and the interests of the target users.

Vu et al. has proposed a system that aggregates information produced by group members on various online services into one social stream. The system filters out irrelevant or private information from the stream, and provide only relevant information to all the group members for information sharing [17]. This idea is similar to the idea of portal accounts on Twitter.

All the studies above are related to retweet recommendation, but none of them discussed the problem of selecting appropriate number of tweets in an online or near-online fashion, which is the main concern of this paper.

There have been also much research on recommendation of tweets not for retweeting but for the target users' own reading. When we recommend tweets for the target user her/himself, we do not need to care about the number of tweets we recommend. We simply show the ranked list of recommended tweets to the user. Then the user reads the list starting from the top-ranked tweet, and stop when the user runs out of time. Therefore, there have been no research on tweet recommendation discussing the problem of the number of tweets to recommend.

### B. Secretary Problems and Online Knapsack Problems

As we explained in I, the recommendation for a portal account should satisfy the following requirements.

1) Maximize the match between selected tweets and users' interests.
2) Recommend at most a given number of tweets in each time period.
3) Decide to recommend a tweet or not as early as possible.

Such a problem is known as a multiple-choice secretary problem [18]. In a multiple-choice secretary problem, you want to hire a given number of secretaries, you interview candidates one by one, and after each interview, you have to decide to hire the candidate or not immediately. The goal is to maximize the total score of the hired candidates.

Another related problem is an online knapsack problem [19], [20]. In an online knapsack problem, items with a value and a weight are shown to you one by one, and after seeing each item, you have to decide whether you keep the item in your knapsack or not immediately. You need to maximize the total values of the selected items while keeping the total weights within the given capacity. The multiple-choice secretary problem is a special case of the online knapsack problem where the weights of all items are equal.

There are variations of the online knapsack problem. For example, the total number of items may or may not be given in advance, and the distribution of item values may or may not be given. In our case, both are not given. Therefore, in order to apply some existing algorithm that requires either of them, we first need to estimate it.

## III. Problem Definition

In this section, we give the formal definition of our problem.

Let $u$ be the current target portal account. $V(u) = \{v_1^u, \ldots, v_m^u\}$ denotes the followers of $u$, and $D(u,T) = \langle d_1^u, \ldots, d_n^u \rangle$ denotes the sequence of tweets posted by the friends of $u$ during a time interval $T = [t_s, t_e]$, where $d_1^u$ is the oldest tweet. $t(d_i^u)$ denotes the posting time of $d_i^u$ and $s(d_i^u, u)$ denotes the score of $d_i^u$ for $u$. Note that $t(d_1^u) < t(d_2^u) < \cdots < t(d_n^u)$. The details of how to compute $s(d_i^u, u)$ is explained in the next section.

Our recommendation system chooses a sub-sequence of $D(u,T)$ including at most $c$ tweets. Let $D'(u,T) = \langle d_{c_1}^u, d_{c_2}^u, \ldots \rangle$ denotes the selected sub-sequence. Our goal is to maximize $\sum_{d \in D'(u,T)} s(d,u)$ while keeping $|D'(u,T)| \leq c$.

## IV. Tweet Evaluation Process

Our recommendation process consists of two sub-processes: whenever a new tweet arrives, we first compute the score of the tweet (evaluation process), and we decide to or not to select it (selection process). In this paper, however, we focus on the latter, i.e., the selection process, for the sake of space limitation. In this section, we explain how we compute the score of tweets only briefly. As explained in Section II, we compute it based on the topic similarity between the tweet and the followers' interests.

### A. Follower Filtering

Followers of a target user, however, may include many inactive users. In retweet recommendation, we should eliminate inactive users, and focus only on the interests of active followers. Login timestamps are the best clue for distinguishing active users from inactive users, but it is not publicly accessible through Twitter API. In this paper, we regard an account is inactive if it satisfies both of the following conditions:

- it posts less than one tweet per week, and
- the number of its friends has not changed for 60 days.

## B. Estimation of Followers' Interests

Next, we estimate the interests of the active followers. Some social networks services, e.g., Facebook, have metadata explicitly describing categories each user is interested in (although some users may not provide information). Twitter, however, only has a free-format profile data, and many users do not provide detailed information in their profiles. In order to obtain as much information on the followers' interests as possible, we use text in their profiles, and also use text in tweets posted by them.

There are various ways to compute a vector representing the topic of the tweets or the profiles. In this paper, however, our main concern is not the evaluation process but the comparison of four algorithms for the selection process. We, therefore, use the most basic tf-idf method instead of other approaches, such as LDA (Latent Dirichlet Allocation).

Before any processing, we preprocess all text contents in our dataset by word stemming, stop word elimination, and unicode normalization. We also eliminate all non-ascii characters, such as Cyrill characters and Greek characters. After that, for each follower $v$ of $u$, we first produce a standard tf-idf vector from the tweets by $v$ in the following way.

For each combination of a word $w$ and a follower $v$, we calculate its tf value $tf(w, v)$, and for each word $w$, we also calculate $idf(w)$, idf of the word $w$ in the Twitter world, by the formula below:

$$\begin{aligned} tf(w,v) &= \frac{occur(w,v)}{\sum_{w'} occur(w',v)} \\ idf(w,u) &= \log \frac{|\{v_i | v_i \in V(u), tf(w,v_i) > 0\}|/|V(u)|}{|\{v_i | v_i \in U, tf(w,v_i) > 0\}|/|U|} \end{aligned}$$

where $occur(w, v)$ is the total number of occurrences of the word $w$ in all tweets by $v$, and $U$ is a set of randomly sampled Twitter users. By using $tf(w, v) * idf(w, u)$, we create a tf-idf vector for each follower $v$.

In our implementation, however, for efficiency, we do not use all words appearing in tweets by the followers of $u$. For each $u$, we select 5,000 words that have larger values for $(\sum_{v \in V(u)} tf(w, v)) * idf(w, u)$, i.e., the sum of the tf-idf values of $w$ over all the followers of $u$.

Next, we augment the vector above by using the information from the profile of the followers. We extract nouns from the profiles of all the followers of $u$ by using morphological analyzer Mecab[1]. We add dimensions for these words to the vector above (except when a word is already included in the 5,000 words above). For each of those nouns, if it appears in the profile of a follower $v$, we set the coordinate for the noun in the tf-idf vector for $v$ to the average of tf-idf values within that vector.

Finally, we compute a vector $\vec{v}(u)$ representing the interests of all the followers of $u$ by taking the sum of the vectors of all the followers of $u$, and normalizing it so that its norm is 1.

[1]Mecab, https://code.google.com/p/mecab/

## C. Evaluation of Tweets

In the same way as we produced the vector $\vec{v}(u)$ for the followers, we produce a vector $\vec{d_i^u}$ for each incoming tweet $d_i^u$ in $D(u, T)$. We apply the same preprocessing to the text contents of $d_i^u$, extract words used in the vector $\vec{v}(u)$ from $d_i^u$, create a tf vector (not tf-idf vector), and normalize it so that its norm is 1. More formally, $\vec{d_i^u}$ is defined as follows:

$$\vec{d_i^u} = \frac{(occur(w_1, d_i^u), occur(w_2, d_i^u), \ldots)}{\sqrt{occur(w_1, d_i^u)^2 + occur(w_2, d_i^u)^2 + \cdots}}$$

where $w_1, w_2, \ldots$ are the words used when producing $\vec{v}(u)$, and $occur(w_j, d_i^u)$ is the number of occurrences of the word $w_j$ in $d_i^u$.

We compute the score of $d_i^u$ for $u$, i.e., $s(d_i^u, u)$, by taking cosine similarity between $\vec{v}(u)$ and $\vec{d_i^u}$ as defined below:

$$s(d_i^u, u) = \vec{v}(u) \cdot \vec{d_i^u}.$$

## V. Selection Process

In this section, we explain our four algorithms for selecting tweets. As explained in Section I, two algorithms are online algorithms that uses thresholds, and the other two algorithms are near-online algorithms that make decisions periodically.

Our two online algorithms have the same basic structure. We compute a threshold, and whenever a tweet arrives, if its value exceeds the threshold, we retweet it. The two algorithms differ only in the method of deciding the threshold.

The input of the algorithms is $c$, the upper limit of the number of tweets, and $\langle d_1^u, \ldots, d_n^u \rangle$, the sequence of tweets posted by the friends during some time interval. The output is a sequence of selected tweets.

### A. History-Based Threshold Algorithm

In the first method, we use the results of the past time intervals to define the threshold. Let $T_z$ be the current time interval. We record the scores of tweets in the past $k$ time intervals, i.e., $T_{z-k}, \ldots, T_{z-1}$, and for each $T_i (z - k \leq i \leq z - 1)$, we compute the $c$-th largest score among those of the tweets in $T_i$. This value corresponds to the best threshold value for the time interval $T_i$, computed offline after we obtain all tweets in $T_i$. Let $\hat{\theta}_i$ denote such an "best offline threshold" for $T_i$. We compute $\hat{\theta}_{z-k}, \ldots, \hat{\theta}_{z-1}$, i.e., the best offline thresholds for the past $k$ intervals, and use their average as the threshold for the next time interval $T_z$. Algorithm 1 shows this algorithm, which we call history-based threshold algorithm.

This algorithm is based on the assumption that the optimal thresholds for consecutive time intervals do not largely change.

### B. Stochastic Threshold Algorithm

In our second algorithm, we estimate the threshold in a probabilistic framework.

In our definition, $s(d_i^u, u)$ is the similarity between the vector for a tweet and the vector for the followers. The latter, i.e., the vector for the followers, is fixed throughout the time interval. We also assume that the former, i.e., the vectors for tweets, are independent from each other. Therefore, we

**Algorithm 1** History-Based Threshold Algorithm

```
$i := 1$
$O := \emptyset$
$\theta := \sum_{i=z-k}^{z-1} \hat{\theta}_i / k$
while $i \leq n$ and $|O| < c$ do
    if $s(d_i^u, u) \geq \theta$ then
        retweet $d_i^u$
        $O := O \cup \{d_i^u\}$
    end if
    $i := i + 1$
end while
```

assume $s(d_i^u, u)$ of tweets in $D(u, T)$ are i.i.d. (independent and identically distributed).

To model $s(d_i^u, u)$, we use the beta distribution for the following reasons.

1) The score $s(d_i^u, u)$ is in the range $[0, 1)$. Although $s(d_i^u, u)$ is defined as cosine similarity which is in the range $[0, 1]$, $s(d_i^u, u)$ can never be 1 because the number of words included in a single tweet is less than 140, while the dimension of our vectors is larger than 5,000. Therefore, we should use some univariate probability distribution whose domain is $[0, 1)$.
2) According to our survey, scores of tweets from the friends of a portal account $u$ are not usually uniformly distributed, so we should not use the uniform distribution.

The probability density function of the beta distribution is defined as follows:

$$f(x; \alpha, \beta) = B(\alpha, \beta)^{-1} x^{\alpha-1} (1-x)^{\beta-1}$$

where $B(\alpha, \beta)$ is a Beta function, which is defined by using Gamma function as follows:

$$B(\alpha, \beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)}.$$

We estimate parameters $\alpha, \beta$ from the scores of the tweets that has arrived so far by using maximum likelihood estimation.

Here we make the following assumptions:

1) the beta distribution can fit the scores of the tweets in the past,
2) scores of tweets in the current time interval are randomly sampled from the estimated distribution,
3) the upper bound of the scores of the tweets in the current interval $T$, denoted by $s^{max}(T)$, is known, and
4) the number of incoming tweets in the current interval, i.e., $|D(u, T)|$ can be estimated.

Then we obtain:

$$\frac{\int_{\theta}^{s^{max}(T)} f(x; \alpha, \beta) dx}{\mu} = \frac{c}{|D(u, T)|} \quad (1)$$

where $\theta$ is the threshold we want to compute, and $\mu$ is the mean value of $f(x; \alpha, \beta)$.

Let $I_t(\alpha, \beta)$ be the cumulative distribution function of the beta distribution. Then (1) can be written as:

$$\frac{I_{s^{max}(T)}(\alpha, \beta) - I_{\theta}(\alpha, \beta)}{\mu} = \frac{c}{|D(u, T)|}$$

and then we have

$$\theta = I^{-1}(I_{s^{max}(T)}(\alpha, \beta) - \frac{\mu c}{|D(u, T)|}). \quad (2)$$

We can estimate the threshold $\theta$ by (2) only if we can estimate $s^{max}(T)$ and $|D(u, T)|$. We estimate them in the following ways:

1) we estimate $s^{max}(T)$ by the average of maximum scores in the last $k$ time intervals, and
2) we estimate $|D(u, T)|$ by the average of the number of incoming tweets in the last $k$ time intervals.

However, the assumptions we explained before do not necessarily hold, and the threshold computed by (2) includes errors for the following reasons.

1) Scores during the time interval $T$ are biased sampling from the distribution $f(x; \alpha, \beta)$ which does not exactly fit the score distribution.
2) $s^{max}(T)$ varies from time to time.
3) $|D(u, T)|$ also varies from time to time.

In order to reduce the error between $\theta$ computed by (2) for the current interval $T_z$ and the real best threshold for $T_z$, we use the errors between them in the past intervals. We compute $\theta_{z-k}, \ldots, \theta_{z-1}$, i.e., thresholds estimated by (2) for the past $k$ intervals $T_{z-k}, \ldots, T_{z-1}$, and also compute the best offline thresholds $\hat{\theta}_{z-k}, \ldots, \hat{\theta}_{z-1}$ defined in the subsection V-A. Then we compute the average error in the past $k$ time intervals by the formula below:

$$\epsilon = \sum_{i=z-k}^{z-1} (\theta_i - \hat{\theta}_i)/k.$$

By using this $\epsilon$, we define a revised threshold $\theta'$ as below:

$$\theta' = \theta - \epsilon.$$

Our second algorithm use this $\theta'$ as the threshold. We omit the formal definition of the algorithm because it is the same as the first algorithm except that we use $\theta'$ as the threshold.

## C. Time-Interval Algorithm

The remaining two algorithms select tweets periodically. The third algorithm does it in a fixed time interval.

We divide the current time interval $T$ into $c$ sub-intervals. Notice that $c$ is the number of tweets we select. In each sub-interval, we select one tweet, and thus we select $c$ tweets in total.

At the beginning of each sub-interval, we create an empty buffer for storing tweets. During a sub-interval, we store all incoming tweets in that buffer, and at the end of the sub-interval, we select a tweet that has the highest score among the tweets in the buffer.

When $c$ is large compared with the number of incoming tweets, or when the arrival of tweets are not uniformly

**Algorithm 2** Time-Interval Algorithm

```
$i := 1$
$O := \emptyset$
$B := \emptyset$
$subinterval := 1$
**while** $i \leq n$ **and** $|O| < c$ **do**
  **if** just passed the end of a sub-interval **then**
    $k := subinterval - |O|$
    retweet top-$k$ tweets in $B$
    $O := O \cup \{\text{top } k \text{ tweets in } B\}$
    $B := \emptyset$
    $subinterval := subinterval + 1$
  **end if**
  $B := B \cup \{d_i^u\}$
  $i := i + 1$
**end while**
```

distributed but are skewed to specific time period, we may have sub-intervals during which we have no incoming tweet. In such a case, we "carry over" the slot to the next sub-interval, and we select two tweets in the next sub-interval. If we have less than two tweets in the next sub-interval, we "carry over" the unused slots to the following sub-interval again.

In the implementation, we do not need to store all incoming tweets, but need to store only $n$ tweets that have the highest scores among the tweets that have arrived so far in the current sub-interval, where $n$ is the number of "carry-over" plus 1. In the definition below, however, we simply store all the incoming tweets for simplicity of the definition.

Formal definition of the third algorithm, which we call time-interval algorithm, is shown in Algorithm 2.

### D. Every n-Tweets Algorithm

The last algorithm also periodically makes a decision, but not at every fixed time interval. Instead, the fourth algorithm makes a decision at every fixed number of tweet arrivals. First, we estimate the number of incoming tweets in the current time interval $|D(u,T)|$ by using the average number of tweets in the last $k$ time intervals. Then we calculate $n = \lfloor |D(u,T)|/c \rfloor$, and every time $n$ tweets arrive, we select a tweet with the highest score among them.

Similarly to the time-interval algorithm, we need a buffer to store tweets. In the fourth algorithm, however, we do not have "carry over". Therefore, we only need to store at most one tweet. However, in the definition of the algorithm in Algorithm 3, we store all $n$ tweets for simplicity of the definition.

**Algorithm 3** Every n-Tweets Algorithm

```
$i := 1$
$O := \emptyset$
$B := \emptyset$
**while** $i \leq n$ **and** $|O| < c$ **do**
  $B := B \cup \{d_i^u\}$
  **if** $|B| \geq n$ **then**
    retweet top tweet in $B$
    $O := O \cup \{\text{top tweet in } B\}$
    $B := \emptyset$
  **end if**
  $i := i + 1$
**end while**
```

## VI. Experiment

### A. Dataset Creation

In this research, we focus on the recommendation of tweets for a portal account to retweet. In order to evaluate our four algorithms, we chose real portal accounts on Twitter, and created a dataset for simulating each of them. We first selected 20 portal accounts satisfying the following conditions:

1) It has more than 300 followers.
2) The account has been used for more than 2 years.
3) At least tweet and retweet 20 tweets a week.
4) More than 50% of its tweets are retweeted tweets.

Then we selected 7 accounts from them. In order to preserve diversity, we classify these 20 portal accounts into 7 categories, and selected the most typical portal accounts from each of the 7 categories. The names of the selected 7 accounts are shown in Table I.

For each of these portal accounts, we collected the tweets by their friends for more than two months (from Oct. 1, 2013 to Dec. 24, 2013) in order to simulate the activity of the portal accounts. In order to estimate the interests of their followers, we also collected the profiles of their followers, and collected tweets by the followers for more than two months (from Oct. 1, 2013 to Dec. 24, 2013). Because of the rate limit of Twitter REST API[2], we could not collect all the tweets by the followers for portal sites with a large number of followers, e.g., Microsoft Japan PR and Tokyo Government PR, which have 19776 and 65768 followers, respectively. In such cases, we randomly select at least 1,500 followers, and collect tweets by them. On the other hand, we did not apply sampling for the friends, even when a portal account has many friends.

The number of the followers and the friends of each portal account are also summarized in Table I. Table I also shows the number of followers for which we actually collected tweets. Although we chose 1,500 users from the followers of each portal account, there was a large number of followers that are shared by more than one portal account, and as the result, the number of monitored followers is far larger than 1,500 for most accounts.

Our dataset include 5,649 users in total and more than 180 million tweets. 5,649 is far smaller than the sum of the monitored followers of the seven portal accounts. It is because there were large duplication among them as explained above.

### B. Experimental Results

By using the dataset explained above, we compared the performance of proposed four algorithms. A standard metric

[2]REST API Rate Limiting in v1.1 https://dev.twitter.com/docs/rate-limiting/1.1

TABLE I
STATISTICS OF 7 PORTAL SITES USED IN OUR EXPERIMENTS

| No. | User Name (English Translation) | Screen Name | # of Followers | # of Monitored Followers | # of Friends |
|---|---|---|---|---|---|
| 1 | Microsoft Japan PR | mskkpr | 19776 | 3091 | 57 |
| 2 | U. Tokyo Navi | Todai_Navi | 4743 | 2415 | 29 |
| 3 | Tokyo Government PR | tocho_koho | 65768 | 2497 | 153 |
| 4 | Nara-city Official Twitter | naracity_tweets | 1718 | 1718 | 28 |
| 5 | Kagawa Prefecture PR | PrefKagawa | 5854 | 1749 | 30 |
| 6 | Kyoto U. COOP Travel Bureau | travel_id | 330 | 330 | 145 |
| 7 | Tokyo Parc Association PR | TokyoParks | 1667 | 1667 | 57 |

TABLE II
THE AVERAGE COMPETITIVE RATIO $r(u, T)$ OF FOUR ALGORITHMS WITH $c = 6, 12, 24$

| Average of $r(u,T)$ | $c = 6$ | $c = 12$ | $c = 24$ |
|---|---|---|---|
| History-Based Threshold | 8.54% | 19.35% | 31.69% |
| Stochastic Threshold | 4.75% | 8.79% | 13.34% |
| Time-Interval | 68.47% | 65.22% | 63.00% |
| Every n-Tweets | 70.56% | 69.94% | 71.76% |

for the evaluation of online algorithms is *competitive ratio*, which is the ratio between the performance of an online algorithm and the performance of the optimal offline algorithm. We compute competitive ratio for a portal $u$ at a time interval $T$ by the formula below:

$$r(u,T) = \frac{\sum_{d \in D'(u,T)} s(d,u)}{\sum_{d \in \hat{D}'(u,T)} s(d,u)}$$

where $D'(u,T)$ is the the sub-sequence of tweets selected by some online algorithm, while $\hat{D}'(u,T)$ is the best answer computed by an offline algorithm. In other words, $\hat{D}'(u,T)$ is the top-$c$ tweets posted by the friends of $u$ during $T$. $r(u,T)$ represents the ratio of total value of tweets selected by an online algorithm and by the optimal offline algorithm.

We run our four algorithms for the 7 portal sites for 75 days with $c = 6, 12, 24$ and the length of the interval $T$ is 24 hours. In other words, we need to recommend 6, 12, or 24 tweets every 24 hours.

The average values of $r(u,T)$ over 75 days for each algorithm and for each of $c = 6, 12, 24$ are summarized in Table II and Figure 1. As shown in the table and the graph, the performance of two online algorithms are low while the performance of two near-online algorithms are high. The performance of Stochastic Threshold algorithm is always the lowest, and the performance of Every n-Tweets algorithm is always the highest. We omit the detailed results in each of $7 * 3 = 21$ cases for 7 sites and $c = 6, 12, 24$ for the sake of space limitation, but n-Tweets algorithm shows the highest performance for 15 cases among the 21 cases, and for the other 6 cases, Time-Interval algorithm shows the highest performance. Their differences are, however, not large in all these 21 cases.

Time-Interval algorithm and Every n-Tweets algorithm, however, have one serious disadvantage: time delay of retweeting. We calculated the average delay of retweeting for each algorithm with $c = 6, 12, 24$. The results are summarized in

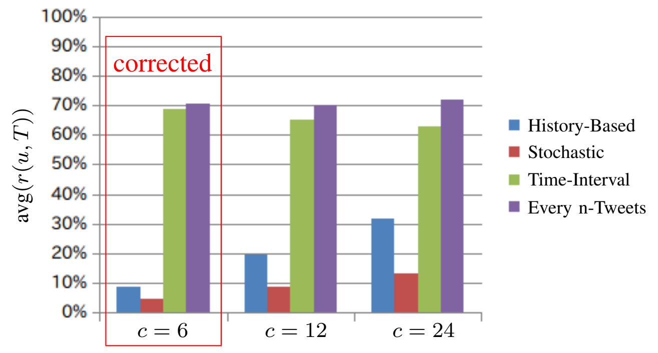

Fig. 1. Average competitive ratio $r(u, T)$ of the algorithms with $c = 6, 12, 24$

TABLE III
AVERAGE DELAY OF FOUR ALGORITHMS WITH $c = 6, 12, 24$

| Delay (sec) | $c = 6$ | $c = 12$ | $c = 24$ |
|---|---|---|---|
| History-Based Threshold | 0 | 0 | 0 |
| Stochastic Threshold | 0 | 0 | 0 |
| Time-Interval | 7051 | 3609 | 1946 |
| Every n-Tweets | 6084 | 2716 | 1198 |

Table III and Figure 2.

The delay is almost 0 for two online algorithms. The delay for Time-Interval algorithm for $c = 6$ is about two hours. This is because we retweet one tweet for every $24/6 = 4$ hours, and the selected tweet is two hours old in average. When $c = 12$ and $c = 24$, the delay for Time-Interval algorithm is about one hour and 30 minutes for the same reason.

On the other hand, Every n-Tweets algorithm has the average delay ~~longer than the Time-Interval algorithm for $c = 6$, but~~ slightly shorter ~~for $c = 12$ and $c = 24$~~. Unlike the Time-Interval algorithm, the Every n-Tweets algorithm does not have the upper bound of the delay (except for the end of the current interval $T$). Suppose $|D(u,T)| = 96$ and $c = 24$. Then we retweet every time we receive 4 tweets. In this case, even if we have three tweets in the buffer, if the fourth tweet do not arrive for long time, no tweet is retweeted for a long time. This does not happen in Time-Interval algorithm as long as we have at least one tweet in the buffer. However, these experimental results show that the n-Tweets algorithm does not cause long delays even when $c$ is small, and produce even shorter delay than Time-Interval algorithm ~~when $c$ is large~~.

In order to determine how long delays can be acceptable for ordinary followers of the portals, we calculated time lags between the arrival and retweeting for all tweets retweeted by

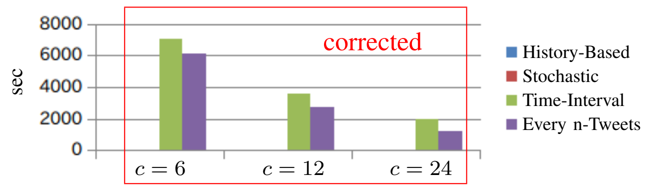

Fig. 2. Average delay of the algorithms with $c = 6, 12, 24$

TABLE IV
DELAY IN 7 REAL PORTAL ACCOUNTS ON TWITTER

| No. | average (sec) | max (sec) | min (sec) | median (sec) |
|---|---|---|---|---|
| 1 | 8982 | 1121476 | 22 | 1560 |
| 2 | 39966 | 1491408 | 26 | 5950 |
| 3 | 13180 | 74051 | 258 | 4775 |
| 4 | 33069 | 611051 | 88 | 10747 |
| 5 | 10524 | 243247 | 33 | 2625 |
| 6 | 63665 | 578893 | 117 | 60075 |
| 7 | 16458 | 265742 | 98 | 3098 |

our 7 portal accounts. The result is summarized in Table IV. Compared with the average delays in these manually managed portal accounts, the delays in our automated system seems acceptable even with near-online algorithms.

Based on our experimental results and the discussion above, our conclusion is that Every n-Tweets algorithm is the best choice for our purpose. Its advantages are as follows:

1) It shows the highest selection quality which is slightly better than that of Time-Interval algorithm.
2) ~~It causes only slightly longer delays than Time-Interval algorithm even in the worst case, i.e., even when $c$ is small.~~

On the other hand, the two real-time algorithms did not work well. The reason why Stochastic Threshold algorithm did not work well could be the lack of enough information on the distribution of tweet scores.

## VII. CONCLUSION

In this paper, we proposed four tweet selection algorithms for retweet recommendation systems. Our experimental results with real Twitter data shows that Every n-Tweets algorithm is the best because it achieves the highest selection quality only with acceptable delays. Our algorithms are also applicable to other stream media, such as RSS. There are several directions for future research. Instead of specifying a rigid upper limit, we can give some penalty to retweets exceeding the limit. In such a setting, even when a portal has retweeted the given number of tweets, it can retweet additional tweets if some of the following tweets are really important. Another interesting direction is dynamically changing the limit depending on the time in a day. For example, we can set the limst to be a small value during the daytime when the users are at work, while we can set it to be a larger value durint the evening.

## REFERENCES

[1] "Twitter statistics," http://www.statisticbrain.com/twitter-statistics/, Jan. 2014.
[2] M. Rosoff, "Twitter has 100 million active users – and 40% are just watching," http://www.businessinsider.com/twitter-ceo-dick-costolo-2011-9, Sept. 2011.
[3] R. Maestre and F. Tapia, "Information propagation in twitters network," http://labs.paradigmatecnologico.com/2011/07/06/information-propagation-in-twitters-network/, Jul. 2011.
[4] J. Fritz, "The life cycle of a tweet and facebook post - still worth it?" http://nonprofit.about.com/b/2011/08/31/the-life-cycle-of-a-tweet-and-facebook-post-still-worth-it.htm, 2011.
[5] J. Nielsen, "Writing for social media: Usability of corporate content distributed through facebook, twitter & linkedin," http://www.nngroup.com/articles/writing-social-media-facebook-twitter/, Oct. 2009.
[6] S. M. Kywe, E.-P. Lim, and F. Zhu, "A survey of recommender systems in Twitter," in *Proc. of SocInfo*, 2012, pp. 420–433.
[7] I. Uysal and W. B. Croft, "User oriented tweet ranking: A filtering approach to microblogs," in *Proc of CIKM*, 2011, pp. 2261–2264.
[8] P. Nasirifard and C. Hayes, "Tadvise: A Twitter assistant based on twitter lists," in *Prof. of SocInfo*, 2011, pp. 153–160.
[9] S. Wang, X. Zhou, Z. Wang, and M. Zhang, "Please spread: Recommending tweets for retweeting with implicit feedback," in *Proc. of DUBMMSM*, 2012, pp. 19–22.
[10] B. Suh, L. Hong, P. Pirolli, and E. H. Chi, "Want to be retweeted? large scale analytics on factors impacting retweet in Twitter network," in *Proc. of SocialCom/PASSAT*. IEEE, 2010, pp. 177–184.
[11] Z. Yang, J. Guo, K. Cai, J. Tang, J. Li, L. Zhang, and Z. Su, "Understanding retweeting behaviors in social networks," in *Proc. of CIKM*. ACM, 2010, pp. 1633–1636.
[12] S. A. Macskassy and M. Michelson, "Why do people retweet? Anti-homophily wins the day!" in *Proc. of ICWSM*, 2011, pp. 209–216.
[13] H.-K. Peng, J. Zhu, D. Piao, R. Yan, and Y. Zhang, "Retweet modeling using conditional random fields," in *Proc. of ICDMW*, 2011, pp. 336–343.
[14] Z. Xu and Q. Yang, "Analyzing user retweet behavior on twitter," in *Proc. of ASONAM 2012*. IEEE, 2012, pp. 46–50.
[15] R. Hochreiter and C. Waldhauser, "A stochastic simulation of the decision to retweet," in *Proc. of ADT*, 2013, pp. 221–229.
[16] H. He, Z. Yu, B. Guo, X. Lu, and J. Tian, "Tree-based mining for discovering patterns of reposting behavior in microblog," in *Proc. of ADMA (1)*, 2013, pp. 372–384.
[17] X.-T. Vu, P. Morizet-Mahoudeaux, and M.-H. Abel, "Empowering collaborative intelligence by the use of user-centered social network aggregation," in *Proc. of Web Intelligence*, 2013, pp. 425–430.
[18] R. D. Kleinberg, "A multiple-choice secretary algorithm with applications to online auctions," in *Proc. of SODA*, 2005, pp. 630–631.
[19] G. S. Lueker, "Average-case analysis of off-line and on-line knapsack problems," in *Proc. of SODA*, 1995, pp. 179–188.
[20] A. Marchetti-Spaccamela and C. Vercellis, "Stochastic on-line knapsack problems," *Mathematical Programming*, vol. 68, no. 1-3, pp. 73–104, 1995.
[21] Y. Song, H. Wang, Z. Wang, H. Li, and W. Chen, "Short text conceptualization using a probabilistic knowledgebase," in *Proc. of IJCAI Vol.3*, 2011, pp. 2330–2336.
[22] D. Kuseta, "The average tweet length is 28 characters long, and other interesting facts," http://www.smk.net.au/article/the-average-tweet-length-is-28-characters-long-and-other-interesting-facts, Nov. 2012.
[23] H. Kellerer, et al. *Knapsack Problems*. Springer, 2004.
[24] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.