KeySee: Supporting Keyword Search on Evolving Events in Social Streams

Pei Lee  
University of British Columbia  
Vancouver, BC, Canada  
peil@cs.ubc.ca

Laks V.S. Lakshmanan  
University of British Columbia  
Vancouver, BC, Canada  
laks@cs.ubc.ca

Evangelos Milios  
Dalhousie University  
Halifax, NS, Canada  
eem@cs.dal.ca

ABSTRACT
Online social streams such as Twitter/Facebook timelines and forum discussions have emerged as prevalent channels for information dissemination. As these social streams surge quickly, information overload has become a huge problem. Existing keyword search engines on social streams like Twitter Search are not successful in overcoming the problem, because they merely return an overwhelming list of posts, with little aggregation or semantics. In this demo, we provide a new solution called KeySee by grouping posts into events, and track the evolution patterns of events as new posts stream in and old posts fade out. Noise and redundancy problems are effectively addressed in our system. Our demo supports refined keyword query on evolving events by allowing users to specify the time span and designated evolution pattern. For each event result, we provide various analytic views such as frequency curves, word clouds and GPS distributions. We deploy KeySee on real Twitter streams and the results show that our demo outperforms existing keyword search engines on both quality and usability.

Categories and Subject Descriptors
H.2.8 [Database Management]: Database Applications—Data Mining; H.3.3 [Information Storage and Retrieval]: Information Search and Retrieval—Search process

General Terms
Measurement, Experimentation

Keywords
KeySee, Keyword Search, Event Evolution, Social Stream

1. INTRODUCTION
Online social streams such as Twitter/Facebook timelines and forum discussions have emerged as prevalent channels for information dissemination. As these social streams surge quickly, information overload (a.k.a. infobesity) has become a spectacular problem, which leads to "information anxiety", the gap between the information we are able to access and the information we are able to perceive [3]. Current social stream search engines like Twitter Search merely return a huge list of posts to a given keyword query, with little aggregation or semantics, and leave it to the users to sift through the large collection of results to figure out the very small portion of useful information. The noisy and redundant nature of social streams degrades user's experience further.

On the other hand, since a post like tweet only conveys a very small piece of information, it would be ideal if we can group the posts talking about the same event together. Here we define an "event" as a transient group of posts sharing the same topic words within a short time span. For example, supposing the query is "iphone 6", instead of showing thousands of posts containing "iphone 6", users expect to see a small number of events, e.g., "iphone 6 release rumor". Users can click on each event to dig into details. Moreover, as time rolls on, new posts stream in and old posts fade out quickly, making the events evolve over time. Hence, user's experience on searching social streams will be improved a lot if the keyword search on evolving events is supported.

Searching evolving events in social streams is a challenging problem. First, the effective organization of meaningful information from noisy unstructured social streams is not easy. Second, the tracking of evolving events and their evolution patterns in a streaming environment is a challenging problem from an effectiveness and efficiency standpoint. Third, supporting efficient keyword search on evolving events poses new challenges for query optimization.

In this demo, we propose a system called KeySee to support Keyword Search on evolving events tracked from social streams. In related work, an efficient index structure for tweets is proposed in [1] and an NLP-based tweet search platform is introduced in [4]. Both [1] and [4] propose results similar to Twitter Search, as shown in Figure 1(a). TweeQL [5] is a query language that allows tweets to be queried by exposing fields like location and text, and offering predicates on them. TwitInfo [6] detects peaks in tweets containing keywords specified by users. Note that a peak detected by TwitInfo is a group of tweets containing designated keywords, but not necessarily an event telling the same story. There are a handful of related works [6, 8, 7] that focus on detecting new emerging events from social streams. However, events may evolve in different ways, and various evolution patterns can be defined, e.g.,

1https://twitter.com/search-home
Users may be interested in monitoring the whole life cycle of events, which is not addressed by existing work on event detection from social streams.

In this demo, KeySee supports a complete set of event evolution patterns as shown in Figure 2(b), and emerge is just one pattern in the set. KeySee provides users the ability to search events that are evolving with a designated pattern in a given time span. The form of query input KeySee supports is \((K, T, P)\), where \(K\) is a set of keywords, \(T\) is a time span and \(P\) is an event evolution pattern. Any member in a KeySee query can be empty but not all of them, i.e., \((\emptyset, \emptyset, \emptyset)\) is not allowed. If the keyword list is empty, the KeySee query means finding all events satisfying designated pattern in a specific time span. For example, \((\emptyset, 3/1/2013-3/7/2013, \text{disappear})\) means finding the disappeared events in the first week of March 2013. The flexibility of KeySee query makes it more powerful than traditional keyword search systems.

The techniques we use in this demo are summarized as follows. We first extract entities (keywords) from a social stream and transform it into an evolving post network by measuring pairwise post similarity. Then, density-based clustering is applied to identify events, defined as dense subgraphs, from the post network. As the time window moves, evolution patterns of events are tracked incrementally. To support keyword query on evolving events, a event evolution pattern is chosen by heuristically selecting the dimension with the observation that the distribution of events on keywords, and the evolution trajectory between snapshots as paths. An event will be a chain in the evolution graph. Based on the observation that the distribution of events on keywords, moments and patterns are very skewed, an optimal query plan is chosen by heuristically selecting the dimension with the lowest probability (i.e., highest selectivity) first. Finally, event results will be ranked and presented to users.

Figure 1 shows the results of searching “Google Nexus” in Twitter Search and KeySee respectively. While Twitter Search returns an overwhelming list of tweets with noise and redundancy, KeySee returns a small number of aggregated events. Each event is annotated with a short text. Users can click buttons below each event to view the event from specific perspectives. KeySee provides four analytic views: volume curves, word clouds, GPS distributions and timelines.

**2. TRACKING EVOLVING EVENTS FROM SOCIAL STREAMS**

**2.1 Social Stream Preprocessing**

Social posts such as tweets are usually written in an informal way with lots of grammatical errors, and even worse, a correctly written post may have no significance and be just noise. Our target is to design a processing strategy that can quickly judge what a post talks about and is robust enough to the informal writing style. In particular, we treat the entities words in a post as keywords, since entities depict the topic. For example, given a tweet “iPad 3 battery pointing to thinner, lighter tablet?”, the entities are “iPad”, “battery” and “tablet”. Traditional Named Entity Recognition tools only support a narrow range of entities like locations or organizations. We broaden the applicability by treating nouns as candidate entities. Finally, a post \(p\) is described as a triple \((p^e, p^e, p^i)\), where \(p^e\) is the list of keywords, \(p^e\) is the time stamp and \(p^i\) is the author. The similarity between posts is measured by a combination of content similarity and temporal proximity, written as

\[
S(p_i, p_j) = \frac{|p^e_i \cap p^e_j|}{|p^e_i \cap p^e_j| \cdot |p^i_i \cap p^i_j|}
\]

We monitor social streams using a sliding time window with length \(Len\), as illustrated in Figure 2(a). At moment \(t\), all posts in the time window \([t - Len, t]\) constitute the snapshot of current social streams. A post network at moment \(t\) can be defined as a graph \(G_t(V_t, E_t)\), where each node \(p \in V_t\) is a post, and an edge \((p_i, p_j) \in E_t\) is constructed if \(S(p_i, p_j) \geq \varepsilon_0\), where \(0 < \varepsilon_0 < 1\). As the time window moves forward, new posts flow in and old posts fade out and \(G_t(V_t, E_t)\) is dynamically updated at each moment, with new nodes/edges added and old nodes/edges removed. We illustrate this process in Figure 2(a). \(Len\) and \(\varepsilon_0\) will be empirically set.
2.2 Event Identification in Post Network

An event snapshot e can be defined as a dense subgraph in post network. The intuition is that, since post network is constructed by pairwise post similarity, a dense subgraph with high internal connectivity shares the same topical words. Various clustering approaches can be applied on a post network to extract dense subgraphs. In this demo, we choose density-based clustering [2] and distinguish nodes in $\mathcal{G}(V, E)$ into three types: core posts, border posts and noise posts, by tuning density parameters. The reason we opt for density-based clustering is that, compared with partitioning-based approaches (e.g., K-Means) and hierarchical approaches (e.g., BIRCH) [2], density-based clustering (e.g., DBSCAN) defines clusters as areas of higher density than the remainder of the data set, which is robust to noise in social streams.

2.3 Tracking Event Evolution

We use $E$ to denote an event and $e$ is a snapshot of $E$ at a specific moment. For simplicity, if we talk about event $e$, it actually means event $E$ at moment $t$. Let $S_t$ denote the set of events at moment $t$. We analyze the evolutionary process of events at each moment and abstract them into four primitive patterns and two composite patterns. The four primitive patterns are emerge, disappear, grow and decay. The two composite patterns are merge and split, which can be decomposed into a series of emerge and disappear patterns. From moment $t$ to $t+1$, they are defined below:

- **emerge**: add event $e$ to the event set $S_t$;
- **disappear**: remove event $e$ from the event set $S_t$;
- **grow**: increase the size of $e$ by adding new posts;
- **decay**: decrease the size of $e$ by removing old posts;
- **merge**: remove a list of events $\{e_1, e_2, \cdots, e_n\}$ from $S_t$ and add a new event $e$, where $e = e_1 + e_2 + \cdots + e_n$;
- **split**: remove an old event $e$ from $S_t$ and add a list of events $\{e_1, e_2, \cdots, e_n\}$, where $e = e_1 + e_2 + \cdots + e_n$.

Compared with primitive patterns, merge and split are not very common in event evolution. To a specific event, emerge/disappear or merge/split can only happen once, but grow/decay may happen at each moment. Thus, if the evolution pattern $P$ in the query $(K, T, P)$ is grow or decay, it is natural to find events evolving with pattern $P$ at every moment in $T$; otherwise, it is natural to find events with pattern $P$ at at least one moment in $T$.

In the following, we explain how to track event evolution incrementally as the post network gets updated. Suppose a post $p$ is added into the post network. If $p$ is a noise post, we simply ignore it. If $p$ is a border post to the neighboring event $e$, grow $e$. If $p$ is a core post without neighboring event, a new event emerges. If $p$ is a core post that is a neighbor of exactly one event $e$, grow $e$. If $p$ is a core post that is a neighbor of multiple events $\{e_1, e_2, \cdots, e_n\}$, merge them into a new event. The analysis of event evolution patterns for removing a post $p$ from post network is very similar. We show evolution patterns of various cases in Table 1.

3. SUPPORTING KEYWORD SEARCH

We built a social stream by aggregating all the timelines of Twitter users in the Technology category of “Who to follow” and their followers. This dataset has 5,196,086 tweets, created by 224,242 users, with a time span from Jan. 1 to Jan 16, 2012.

#### Table 1: Event evolution patterns

<table>
<thead>
<tr>
<th>Cases</th>
<th>Add</th>
<th>Remove</th>
</tr>
</thead>
<tbody>
<tr>
<td>If $p$ is a noise post</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>If $p$ is a border post to the neighboring event $e$</td>
<td>grow</td>
<td>decay</td>
</tr>
<tr>
<td>If $p$ is a core post without neighboring event</td>
<td>emerge</td>
<td>disappear</td>
</tr>
<tr>
<td>If $p$ is a core post to exactly one neighboring event $e$</td>
<td>grow</td>
<td>decay</td>
</tr>
<tr>
<td>If $p$ is a core post to multiple neighboring events ${e_1, e_2, \cdots, e_n}$</td>
<td>merge</td>
<td>split</td>
</tr>
</tbody>
</table>

![Figure 3](image)

**Figure 3**: (a) The evolution of events related to “Stop Online Piracy Act (SOPA)”. We illustrate the snapshots of events by word clouds in this example. Notice that an event can be described in various ways, e.g., frequency curves, Map view, timelines and word clouds.

Feb. 1, 2012. We deploy KeySee on top of this social stream, by setting $Len=1$ day and $\epsilon_0 = 0.3$.

3.1 Event Evolution Graph

Each event has its life cycle. An event can be born by emerging, merging or splitting. As an example, we show the evolution of events related to “Stop Online Piracy Act (SOPA)” in Figure 3, by illustrating each event snapshot by a word cloud. As we can see, the merging of “sopa pipa protest” and “Apple products” makes the birth of a new event called “sopa protest and Apple” on Jan 19, which evolves on Jan 20 and splits into two events on Jan 21.

The evolution of events in a social stream can be captured by a directed graph, where each node $e_n$ is a snapshot of an event $E_n$, which is basically a dense subgraph in the post network, and an edge between node $e_n$ and $e_0$ means “$e_n$ evolves into $e_0$”, which is annotated by an evolution pattern. We call this graph event evolution graph. Figure 4(b) shows the evolution graph regarding events illustrated in Figure 3. If an event spans several time moments, the snapshots of this event should be connected by grow or decay only. For example, the chain $(e_1, e_2, e_3)$ is an event. Merge is a composite pattern, which terminates multiple events and produces a new event in the same time. Split works in an opposite way. In a nutshell, event evolution graph preserves the whole history of event evolution patterns, and serves as basis for keyword search on evolving events. Given a query

2http://twitter.com/who_to_follow/interests
(K,T,P), KeySee will find the events with evolution pattern P in the time span T that hit keywords K.

3.2 Query Plan Selection

The queries we support in this demo has three dimensions: keywords, time span and evolution pattern. We observe that the distributions of social streams on these three dimensions are very skewed. Figure 4(b) shows the frequency of tweets on each day of a week in 5 samples and it indicates that the frequency of tweets on weekdays are obviously higher than the frequency on the weekend. Figure 4(c) shows the count of tweets for top 1000 keywords ranked in descending order. The power law distribution suggests that a very few keywords have very high occurrence. Empirical study also shows merge and split patterns occur much less frequently than other patterns. All this skewness in social streams indicates the scope for query optimization.

Theoretically, query optimization relies on the cost estimation of alternative query plans, which are in turn decided by the selectivities on each dimension. In practice, we maintain the probability mass functions (PMF) of keywords, moments and patterns and estimate the selectivity on each dimension. A heuristic rule is applied by always selecting the dimension with the lowest probability (i.e., highest selectivity), making it likely that the intermediate results fit in memory, facilitating pipeline processing. For example, if \( PMF_P < PMF_K < PMF_T \), the query plan chosen by our system will find all events in time span T first, retain the events with keywords K, and only keep those with evolution pattern P. The percentage of events in the result set is estimated by

\[
P(K,T,P) = PMF_T \cdot PMF_K \cdot PMF_P
\]

3.3 Rank Event Results

Event results will be ranked by the score function and presented to users. Given event \( E \) and a KeySee query \( Q = (K,T,P) \), we define the score function as a combination of normalized keyword frequency and event volume changing function, that is

\[
s(E, Q) = \frac{\min\{f(k,E) : k \in K\}}{\max\{f(x,E) : x \in E_K\}} \cdot w(E, Q)
\]

\[
w(E, Q) = \frac{\Delta|E|}{|E|} \quad \text{if } P \in \{\text{grow, decay}\}
\]

\[
1 \quad \text{otherwise}
\]

where \( f(x,E) \) is the frequency of keyword x in event E and \( E_K \) is the set of all keywords in E. \( \Delta|E| \) is the volume change of the event at the beginning and end of the time span T, which is applicable only when \( P \in \{\text{grow, decay}\} \).

4. CONCLUSION

The information overload problem in current social web age can be frustrating to the user. Existing keyword search engines on social streams are not successful in conquering the problem, because they merely return a huge list of posts, with little aggregation or semantics. To overcome the information overload problem and filter out the noise and redundancy in social streams, we propose KeySee, which transforms a social stream into an evolving post network and uses density-based clustering to identify the events. As time rolls on, event evolution patterns are tracked and stored in an evolution graph. KeySee supports refined keyword query on evolving events by allowing users to specify the time span and the evolution pattern. Empirical study on real Twitter streams shows that KeySee outperforms existing keyword search engines on both quality and usability. In the future work, we look forward to running KeySee on very high throughput Twitter streams and performing advanced analytics on events, such as sentiment analysis, link prediction, recommendation, etc.

5. REFERENCES