Mining Social Streams: Models and Applications

Karthik Subbian, Charu C. Aggarwal
University of Minnesota, IBM T. J. Watson Research Center
karthik@cs.umn.edu, charu@us.ibm.com

Abstract

Online social networks are rich in terms of structural connections between entities and the content propagated between them. When this data is available as a continuous stream of interactions on the social graph, it is referred to as a social stream*. Social streams have several challenges: (1) size of the underlying graph, (2) volume of the interactions, and (3) heterogeneity of the content and type of interactions between the entities. Mining social streams incorporates discovering patterns and trends using* both structure and interaction data*.*

In this work, we discuss two important social stream applications: (1) event detection and (2) influence analysis. Event detection is seen as a social stream clustering problem that can be either supervised and unsupervised, depending on the availability of labeled data. While influence analysis is an unsupervised problem modeled in a query-based framework. We discuss the key characteristics of these two problems, their computational difficulties and efficient modeling techniques to address them. Finally, we highlight several challenges and future opportunities for research in this area.

1 Introduction

The prominence of social network and the rise of cloud- and web-based applications, in the last decade, has enabled greater availability of streaming social data. The data available in the stream could be a continuous stream of edges as in the case of new friendships in Facebook or streaming interactions between users such as stream of tweets in Twitter. There are several challenges in mining and discovering patterns in streaming social data. Some of them are (a) processing the data in single pass (i.e., one-pass constraint), (b) continuous maintenance of the model, and (c) its efficient representation in memory through hashing, sampling, or latent factors. Depending on the availability of supervised knowledge, the learned model can be either supervised or unsupervised. characteristics of such social streams and their applications.

Social streams consist of *content-based interactions between structurally connected entities* in the data. Let us assume that the structure of the social network is denoted by the graph $G = (N, A)$. The node set is denoted by N and edge set is denoted by A. The concept of a social stream is overlaid on this *social network structure* G.

Definition 1 (Social Stream): A social stream is a continuous and temporal sequence of objects $S_1 \ldots S_r \ldots$, such that each object S_i is a tuple of form (q_i, R_i, T_i) where,

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

- $q_i \in N$ is the sender of the message T_i to other nodes.
- T_i corresponds to the content of the interaction.
- $R_i \subseteq N$ is a *set of one or more* receiver nodes, which correspond to all recipients of the message T_i from node q_i. It is assumed that each edge (q_i, r) belongs to the set A, for all $r \in R_i$.

One can also view a social stream as an enriched version of a graph stream, which does not have associated content [1].

1.1 Social Stream Applications

There are numerous applications that are popular in the context of social streams. We discuss some of them below and with their key characteristics.

- **Event Detection** The problem of event detection [3, 22] is to identify the most important set of keywords that describe a trending event. Such trending events often occur in temporal bursts in specific network localities. This problem is typically unsupervised where the new events are detected with no prior knowledge. In some cyber-security applications, such as detecting terrorist attacks or civilian unrest, previous occurrences of such instance can be used to supervise and detect their repeating occurrence.
- **Influence Analysis** The diffusion of information in a social network via re-sharing is referred to as cascades. Identifying influential users [24] in these information cascades in social streams is a challenging problem. The streaming nature of the data makes it harder to track the diffusion of information across longer paths in the network when there are numerous messages propagated. Moreover, to be able to query and understand the influential users in a time-sensitive manner requires efficient data structures and their maintenance.
- **Link Prediction** Discovering links in streaming graph data can be important in a variety of applications where the graph is changing rapidly. For instance camera-first applications, like Snapchat or Instagram, where an user visits the application several times a day, recommending the right group or user to follow is critical to maintain a low churn rate. In such scenarios computing centrality measures, such as Jaccard coefficient, common neighbors, and Adar-Adamic in streaming graphs can be computationally hard. There are some recent approaches [30] that tackle this problem using cost-effective graph sketches based on hashing and sampling techniques.

In this paper, we will discuss the first two applications: (1) Event detection and (2) Influence Analysis. For event detection [3], we discuss an online clustering algorithm which tracks both content and network structure efficiently using hashing schemes. In the case of influence analysis [26], we consider a more flexible querying model to query influence scores either using the set of influencers, set of users influenced, the context or time. Event detection is discussed in both supervised and unsupervised settings. While influence analysis is an unsupervised problem modeled in a query-based framework.

2 Event Detection

The problem of event detection is closely related to that of *topic detection and tracking* [5, 4, 7, 29]. This problem is also closely related to *stream clustering*, and attempts to determine new *topical trends in the text stream* and their significant evolution. The idea is that important and newsworthy events in real life are often captured in the form of *temporal bursts of closely related messages in a network locality*. Clearly, messages which are sent between a tightly knit group of actors may be more indicative of a particular event of social interest, than a set of messages which are more diffusely related from a structural point of view. At the same time, the *content and topics* of the documents should also play a strong role in the event detection process. Thus, both network locality and content of interaction need to be leveraged in a *dynamic streaming scenario* for the event detection process [3].

2.1 Social Stream Clustering

We begin with describing an unsupervised technique for event detection. This approach continuously characterizes the incoming interactions in the form of clusters, and leverages them in order to report events in the social stream. Formally, the problem of social stream clustering can be defined as follows:

Definition 2 (Social Stream Clustering): A social stream $S_1 \dots S_r \dots$ is continuously partitioned into k *current* clusters $\mathcal{C}_1 \dots \mathcal{C}_k$, such that:

- Each object S_i belongs to at most one of the current clusters $\mathcal{C}_r$.
- The objects are assigned to the different clusters with the use of a similarity function which captures both the content of the interchanged messages, and the dynamic social network structure implied by the different messages.

As the clusters are created dynamically, they may change considerably over time with the addition of new points from an evolving stream. Furthermore, in some cases, an incoming object may be significantly different from the current clusters. In that case, it may be put into a cluster of its own, and one of the current clusters may be removed from the set $\mathcal{C}_1 \dots \mathcal{C}_k$. Such an event may be an interesting one, especially if the newly created cluster starts a new pattern of activity in which more stream objects are subsequently added. Therefore, there are two kinds of events *novel* and *evolutionary events*.

The arrival of a data point S_i is said to be a *novel event* if it is placed as a single point within a newly created cluster $\mathcal{C}_i$. It is often possible for previously existing clusters to match closely with a sudden burst of objects related to a particular topic. This sudden burst is characterized by a change in fractional presence of data points in clusters. Let $F(t_1, t_2, \mathcal{C}_i)$ be the fractional cluster presence of objects arrived during time period (t_1, t_2) which belong to cluster $\mathcal{C}_i$ normalized by the total number of objects in cluster $\mathcal{C}_i$.

In order to determine *evolutionary events*, we determine the higher rate at which data points points have arrived to this cluster in the previous time window of length H, as compared to the event before it. A parameter α is used in order to measure this evolution rate and $t(\mathcal{C}_i)$ is the creation time for cluster $\mathcal{C}_i$. This condition of evolution rate is formally defined in Eqn. 1. Here it is assumes that the value of $t_c - 2 \cdot H$ is larger than the cluster creation time $t(\mathcal{C}_i)$ in order to define the evolution rate in a stable way.

$$\frac{F(t_c - H, t_c, \mathcal{C}_i)}{F(t(\mathcal{C}_i), t_c - H, \mathcal{C}_i)} \geq \alpha \tag{1}$$

2.2 Online Model Maintenance

The design of an effective *online clustering* algorithm is the key to event detection. There are two main components that decide the efficiency of online clustering in social streams: (a) representation of clusters and (b) efficient similarity computation using text and structural content.

In order to detect a novel or an evolutionary event it is critical to decide which cluster to assign the incoming social stream object. Hence, the similarity score computed between the incoming object and the clusters, using structure and content information, plays a crucial role. The structure and content of the *incoming object* is usually smaller and easier to represent in-memory. However, the cluster information is extremely large to fit in-memory, as the cluster may represent all the incoming social stream objects until that time. Hence, we need an efficient summarization of clusters. A typical way to summarize the clusters is using bit vectors or normalized frequency counters [3, 2]. However, when the size of the vectors become extremely large it needs to be compressed to fit in-memory for tracking the social stream clusters. In the following we discuss one such efficient summarization and hash-based similarity computation approach for online model maintenance

2.2.1 Cluster Summarization

Each of the social stream clusters for event detection must maintain both text and network information in their summaries. The cluster-summary $\psi_i(\mathcal{C}_i)$ of cluster $\mathcal{C}_i$ is defined as follows:

- It contains the node-summary, which is a set of nodes $V_i = \{j_{i1}, j_{i2} \dots j_{is_i}\}$ together with their frequencies $\eta_i = \nu_{i1} \dots \nu_{is_i}$. The node set V_i is assumed to contain s_i nodes.
- It contains the content-summary, which is a set of word identifiers $W_i = \{l_{i1}, l_{i2}, \dots l_{iu_i}\}$ together with their corresponding word frequencies $\Phi_i = \phi_{i1}, \phi_{i2} \dots \phi_{iu_i}$. The content-summary W_i is assumed to contain u_i words.

The overall summary is $\psi_i(\mathcal{C}_i) = (V_i, \eta_i, W_i, \Phi_i)$.

2.2.2 Efficient Similarity Computation

The size of word frequencies $|\Phi_i|$ is generally smaller compared to $|\eta_i|$. This is because the size of vocabulary (in 100 thousands) is often much smaller than the number of users in the social network (usually in billions). Hence, it requires efficient computation of similarity for the structural part. Note that the content-based similarity computation $SimC(S_i, C_r)$ is straightforward, and is simply the tf-idf based [21] similarity between the content T_i and W_r.

The structural similarity between the nodes V_r and the nodes $R_i \cup \{q_i\}$ in the social stream is computed as follows. Let $B(S_i) = (b_1, b_2, \dots b_{s_r})$ be the bit-vector representation of $R_i \cup \{q_i\}$, which has a bit for each node in V_r, and in the same order as the frequency vector $\eta = (\nu_{r1}, \nu_{r2}, \dots \nu_{rs_r})$ of V_r. The bit value is 1, if the corresponding node is included in $R_i \cup \{q_i\}$ and otherwise it is 0. The structural similarity between the object S_i and the frequency-weighted node set of cluster $\mathcal{C}_r$ is defined as follows:

$$SimS(S_i, \mathcal{C}_r) = \frac{\sum_{t=1}^{s_r} b_t \cdot \nu_{rt}}{\sqrt{||R_i \cup \{q_i\}||} \cdot (\sum_{t=1}^{s_r} \nu_{rt})} \tag{2}$$

Note the use of L_1-norm for the node-frequency vector (as opposed to L_2-norm) in the denominator, in order to to penalize the creation of clusters which are too large. This will result in more balanced clusters. Note that the incoming node set contains both sender and the receivers.

The overall similarity $Sim(S_i, \mathcal{C}_r)$ can be computed as a linear combination of the structural and content-based similarity values.

$$Sim(S_i, \mathcal{C}_r) = \lambda \cdot SimS(S_i, \mathcal{C}_r) + (1 - \lambda) \cdot SimC(S_i, \mathcal{C}_r) \tag{3}$$

The parameter λ is the balancing parameter, and lies in the range $(0, 1)$. This parameter is usually specified by the user.

The numerator of Eqn. 2 is harder to compute as maintaining the vector η_r is expensive. One approach is to compress the size of η_r and estimate the numerator using count min-hash technique [10]. Consider w pairwise independent hash function, each of which resulting in a hash table of size 0 to $h - 1$. Whenever a node is encountered in $q_i \cup R_i$ it is hashed using all w hash functions and the corresponding counts in each of the hash tables are incremented. Since there are collisions in hash table there may be an over-estimate of the exact count of nodes in each hashtable. To upper-bound this over-estimate one can use the minimum value across w hash tables. The numerator of Eqn. 2 can be thus obtained using the sum of minimum counts of hashed incoming nodes in the social stream object. Let this estimated structural similarity be denoted as $EstSimS(S_i, \mathcal{C}_r)$. In Lemma 3 the upper bound of this estimated similarity is shown. We ask the readers to refer to [3] for details of this upper bound proof and [10] for min-hash count technique.

Lemma 3: If a sketch-table with length h and width w is used, then for some small value $\epsilon > \sqrt{|R_i| + 1}/h$, the estimated value of the similarity $EstSimS(S_i, \mathcal{C}_r)$ is bounded to the following range with probability at least $1 - \left(\frac{\sqrt{|R_i|+1}}{h \cdot \epsilon}\right)^w$:

$$SimS(S_i, \mathcal{C}_r) \leq EstSimS(S_i, \mathcal{C}_r) \leq SimS(S_i, \mathcal{C}_r) + \epsilon. \tag{4}$$

The similarity of the social stream object to the assigned cluster is often maintained as a distribution. If the similarity of the newly arrived object is β standard deviations away from the mean, then the object is assigned to its own cluster, resulting in a novel event (See Section 2.1). When β is too small it results in highly unstable clusters and when too large it leads to stale clusters. Note that one must maintain the zeroth, first and second order moments M_0, M_1 and M_2 of the closest similarity values continuously to compute the mean (μ) and standard deviation (σ). These values can be easily maintained in the streaming scenario, because they can be *additively* maintained over the social stream. The mean and standard deviation can be expressed in terms of these moments as follows:

$$\mu = M_1/M_0, \quad \sigma = \sqrt{M_2/M_0 - (M_1/M_0)^2}.$$

2.3 Supervised Event Detection

In several cyber-security applications users look for suspicious events, based on historical occurrences of these instances. This is the case of *supervised event detection*. Here, we assume that we have access to the past history of the stream in which the event $\mathcal{E}$ has been known to have occurred. The *event signature* of a social stream is a k-dimensional vector $V(\mathcal{E})$ containing the (average) relative distribution of event-specific stream objects to clusters. Here k is the number of clusters in the model. Clearly, the event signature provides a useful characterization of the relative topical distribution during an event of significance.

During a period of mideast unrest (the event $\mathcal{E}$), some clusters are likely to be much more active than others, and this can be captured in the vector $V(\mathcal{E})$, as long as ground truth is available to do so. The event signatures can be compared to *horizon signatures*, which are essentially defined in the same way as the event signatures, except that they are defined over the more recent time horizon $(t_c - H, t_c)$ of length H. One can compute the dot product similarity of the horizon signature to the event signature and raise an alarm if its value is above a certain threshold. The tradeoff between false positives and false negatives is determined by the threshold.

3 Influence Analysis

The problem of finding influential actors is important in various domains such as viral marketing [6, 27] and political campaigns. The problem was formally defined by Kempe et al. [16] as an optimization problem over all possible subsets of nodes with cardinality k. Subsequently, a significant amount of work [16, 18, 9, 8, 14, 13] has been done on this area. All these approaches are static in the sense that they work with a fixed model of network structure and edge probabilities. In practice, however, the influence of actors are defined by how their messages are propagated in the social network over time. Such propagation can only be observed from the underlying *social stream*, such as a Twitter feed or the sequence of Facebook activities. Since the problem is dynamic in nature, using the actual flow of information is becoming more popular [25].

A major disadvantage of existing influence analysis methods is that they are not able to *query* the influencers in a *context-specific* fashion. Ideally, one would like to be able to use search terms to determine influencers that are specific to a given context. For example, the top influencers for the search term "*Egypt Unrest*" would be very different from that of the search term "*PGA Golf Tour*". The inflexibility of existing methods is, in part, because the existing methods [16, 18, 9, 8, 14] decouple the problem of influence analysis from learning content-centric influence probabilities [12].

The influencers in a social stream are *time-sensitive* and may rapidly evolve [1], as different external events may lead to changes in the influence patterns over time. For instance, a query such as "*winter boots*" may generally have prominent entities associated with shoe stores as the most influential entities, but an (advertisement) statement from a popular figure in the entertainment industry, such as Justin Bieber, on a specific boot style may change this ordering. Important events can often dynamically change the influencers, and this can only be tracked in a *time-sensitive* and *online fashion* from the underlying activities in the social stream.

3.1 Influence Querying in Social Streams

Influence function is usually composed of four important components: (a) set of influencers, (b) set of users being influenced, (c) context of influence (or keywords), and (d) time of influence. One can compute the influence score as a function of these parameters and use it to query influencers in context-sensitive and time-sensitive manner using social streams.

Let the influence function $\mathcal{I}(S_1, S_2, \mathcal{Q}, t)$ represents the aggregate influence score of actor set S_1 on actor set S_2 with respect to content $\mathcal{Q}$ at time t. Most of the queries are resolved by evaluating this score and then ranking it over various possibilities in the argument. One or more of the arguments in $\mathcal{I}(S_1, S_2, \mathcal{Q}, t)$ can be instantiated to a "*" (don't care) in order to enable more general queries in which all possibilities for the matching are considered. For instance, the queries $\mathcal{I}(\text{"David"}, *, \text{"Egypt unrest"}, t)$ and $\mathcal{I}(\text{"John"}, *, \text{"Egypt unrest"}, t)$ can be used to compare the total influence of David and John on the topic "*Egypt unrest*" at time t. Some examples of useful queries that can be resolved with this approach are as follows:

- For a given query context $\mathcal{Q}$, determining the top-k *influencers* at time t can be formulated as:

$$\max_{X:|X|=k} \mathcal{I}(X, *, \mathcal{Q}, t).$$

  It is also possible to constrain the query to consider a specific subset Y in the second argument, corresponding to the influenced actors. For example, a winter clothing merchant might decide to consider only those actors whose location profiles correspond to colder regions.

- Determining the top-k *influenced* actors at time t, for a given query context $\mathcal{Q}$, can be extremely useful in context-specific subscriptions and in recommending interesting public content to influenced users.

- Influence queries can also be useful in context-sensitive link recommendation, such as finding the top-k influencer-influenced pairs, for a given query context $\mathcal{Q}$.

3.2 Information Flow Based Querying

The context information used in the influence function for a query can be a set of hashtags, tokens or keywords. Such keywords propagated via nodes (or actors) in a social network is considered as an *information flow path*. A flow path must also satisfy a valid path in the network structure. For instance, if there is information reshared from a_1 to a_2 to a_3 and there is no network edge between a_2 to a_3. Then, the flow is only valid until a_1 to a_2. Hence, the valid flow path would be $\mathcal{P} = \langle a_1, a_2 \rangle$ (not $\langle a_1, a_2, a3 \rangle$).

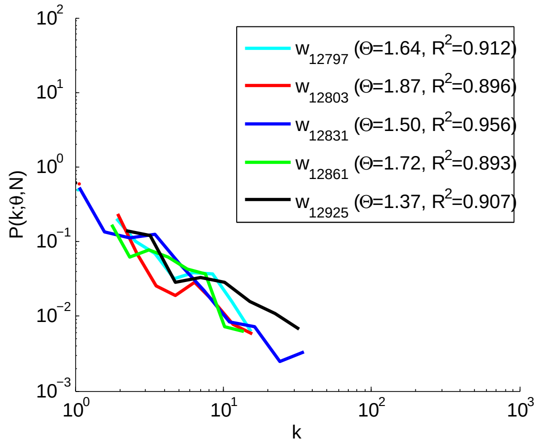

Figure 1: The decayed flow weights in the flow-path trees follow a Zipf distribution. The best-fit estimate of the Zipf parameter θ and corresponding R^2 for few flow-path trees are shown in the legend.

Time-sensitive influence can be computed using an half-life function or exponentially decaying function $exp(-\lambda\delta t)$. Note that the decay rate (λ) is application specific as some social networks may have faster turn around time and hence information propagates quickly. In such cases λ can be set to a higher value to reduce the amount of influence effect. The time difference between the sender (t_o) and the final receiver (t_c) of the message in the flow path $\mathcal{P}$ is denoted by δt. Then the influence score for one flow path $\mathcal{P}$ for carrying a single keyword K_i at time t_c is given by $exp(-\lambda * (t_c - t_o))$ where $\delta t = t_c - t_o$.

There can be many paths through which many keywords may propagate between a pair of actors. Hence, one can accumulate all of that influence between actors a_j and a_k for keywords K_i at time t_c as $\mathcal{V}(a_j, a_k, K_i, t_c)$. When there are several keywords in the query Q, one can compute the atomic influence as the aggregate pairwise influence across all keywords in the query Q. Formally, it is defined as follows:

Definition 4 (Influence Function): The atomic influence function $\mathcal{I}(a_j, a_k, \mathcal{Q}, t)$ for a node a_j to influence a_k, is defined as the sum of the aggregate pairwise flows over all keywords $K_i \in \mathcal{Q}$ in the data: $\mathcal{I}(a_j, a_k, \mathcal{Q}, t) = \sum_{K_i \in \mathcal{Q}} \mathcal{V}(a_j, a_k, K_i, t)$.

3.3 Efficient Tracking of Influencers

In order for one to compute the influence function $I(a_i, a_j, K_i, t_c)$ we need to know all the information flow paths between a_i and a_j, for single keyword K_i, until time t_c. A simple tree based data structure is proposed in [26] called *Flow Path Tree*. The notion of this tree data structure is to have one tree for each keyword and as the social stream objects are encountered the paths are back tracked and the tree data structure is updated. The back tracking may seem exponential in nature, particularly due to the power law degree distribution of the social graphs. However, the keyword K_i is not propagated by all nodes and hence back tracking in practice is much cheaper from a computational perspective.

The main disadvantage of tracking the flow paths using the tree is the size of the tree. The tree grows at an exponentially faster rate as the volume of the stream increases. So, we need an efficient way to maintain the in-memory representation of the flow path tree. The influence weights computed in a single flow path tree for a single keyword K_i using exponentially decaying functions generally follows a skewed Zipf distribution [26]. This is shown in Fig. 1 using a log-log plot. This observation implies that most of the tree weight is generated from a very few important flow paths in the tree. Using this observation, one can trim down the tree by a fraction of $1-\alpha$, where α is the fraction of nodes retained in the tree after trimming. Also, only leaves of the tree are trimmed when the number of nodes in the tree reaches a maximum threshold, say N. Then the fraction of weights that remain in the tree can be lower-bounded using Theorem 5. We request the readers to refer to [26] for the proof of Theorem 5.

Theorem 5: Let the flow weights on the N nodes in the tree be distributed according to the Zipf distribution $1/i^\theta$ for $\theta \geq 1$. Then, the total fraction $F(N, \alpha)$ of the flow in the top $n = \alpha \cdot N$ nodes of the tree is at least equal to:

$$F(N, \alpha) \geq \log(n)/\log(N) = 1 - \log(1/\alpha)/\log(N). \tag{5}$$

The skew helps significantly in retaining the vast majority of the heavy flows. For example, in a flow path tree with $100,000$ total flow weight, discarding half the nodes ($\alpha = 0.5$) would result in total flow weight reduction of only $1 - \log(50000)/\log(100000) = 0.06$. Thus, the vast majority of the heavy flows (i.e. 94%) are retained, which are the ones most relevant for the influence analysis process. This suggests that the pruning process can be used to significantly reduce the complexity of the flow-path tree, while retaining most of the information needed for the influence analysis task.

3.4 Relationship with Katz Measure

The atomic influence function is closely related to the Katz measure [20]. The Katz measure is defined in terms of the weighted sum of the number of all possible paths between a pair of nodes and the weight decays exponentially with the length of the underlying path. Specifically, if $\mathcal{P}_{ij}$ be the set of paths between nodes a_i and a_j, then the Katz measure $K(a_i, a_j)$ is defined as follows:

$$K(a_i, a_j) = \sum_{P \in \mathcal{P}_{ij}} \gamma^{|P|} \tag{6}$$

Here γ is the discount factor on the path length, which is analogous to the flow-based temporal decay factor. Thus, flow-based approach computes exponentially decayed flow weights across different paths, as a more dynamic, time- and content-sensitive way of measuring the importance of nodes. In an indirect sense, this way of computing node importance can be considered a flow-based analogue to the Katz measure in static networks. Because the Katz measure has been shown to be effective for link recommendation in static networks [20], it lends greater credence to its use in the flow-based streaming scenario. Of course, the Katz measure is used rarely in static networks because of the complexity of enumerating over a large number of possible paths. The important point to understand is that the flow-based measure *significantly* changes in the importance of different paths in the update process, and can also be more efficiently computed in the streaming scenario, such as using the pruning technique discussed in Section 3.3.

4 Future Work and Conclusions

The area of social stream research lies in the intersection of big data, graph mining, and social network analysis. One of the important characteristics of social streams is the availability of high velocity streaming data that includes both structural and content information. Moreover, the sheer volume and the heterogeneity of the underlying social interactions makes the problem much more challenging. For example, a social network could have billions of nodes, trillions of edges, trillion interactions per day and a wide variety of such interactions (e.g. like, share, and comment).

There are numerous social network algorithms that are yet to be developed for various social streaming applications. Community detection falls in the realm of social stream clustering. However, incorporating structure, content and time aspects simultaneously and being able to query the nearest neighbors within a cluster or obtaining cluster assignment probabilities in near real-time is a challenging problem. This is different from traditional clustering in graph streams as it encompasses content and meta information about nodes and edges. Link prediction is similarly another interesting problem, which has been solved in static and dynamic scenarios. However, when given content information propagated by nodes and their meta data, finding relevant links for recommendation based on recent content interactions is quite challenging. Again this problem is very different from link prediction in heterogeneous graphs [11] and streaming link prediction [30], as it does not take into account the content and time of propagation simultaneously.

In several social and online applications users create several implicit signals for analysis, based on their online interaction. For example, listening habits in Last.Fm or Pandora.com, along with the social relationships, can be used to understand an users musical interests and make right recommendations. There are several papers [28, 15, 23, 19, 17] that discuss about making recommendations incorporating the temporal dynamics. However, their models cannot be updated in real-time particularly for such high volume and velocity social data.

The problem of mining social streams is very important to discover real-time trends, using both content and structural information with a wide variety of practical applications. We discussed two important applications: event detection and influence analysis. There are other interesting social network problems that are less studied in the context of social streams, such as link prediction and recommendation. These areas will grow significantly in the next few years with the advent of scalable and distributed infrastructure, availability of multiple social streams, and need for real-time answers.

References

[1] C. Aggarwal and K. Subbian. Evolutionary network analysis: A survey. *ACM Computing Surveys (CSUR)*, 47(1):10, 2014.

[2] C. C. Aggarwal. An introduction to social network data analytics. *Social network data analytics*, pages 1–15, 2011.

[3] C. C. Aggarwal and K. Subbian. Event detection in social streams. In *SDM*, pages 624–635. SIAM, 2012.

[4] J. Allan, V. Lavrenko, and H. Jin. First story detection in tdt is hard. In *CIKM*, pages 374–381. ACM, 2000.

[5] J. Allan, R. Papka, and V. Lavrenko. On-line new event detection and tracking. In *SIGIR*, pages 37–45. ACM, 1998.

[6] S. Bhagat, A. Goyal, and L. V. Lakshmanan. Maximizing product adoption in social networks. In *WSDM*, pages 603–612, 2012.

[7] T. Brants, F. Chen, and A. Farahat. A system for new event detection. In *SIGIR*, pages 330–337. ACM, 2003.

[8] W. Chen, C. Wang, and Y. Wang. Scalable influence maximization for prevalent viral marketing in large-scale social networks. In *KDD*, pages 1029–1038. ACM, 2010.

[9] W. Chen, Y. Wang, and S. Yang. Efficient influence maximization in social networks. In *KDD*, pages 199–208. ACM, 2009.

[10] G. Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

[11] Y. Dong, J. Tang, S. Wu, J. Tian, N. V. Chawla, J. Rao, and H. Cao. Link prediction and recommendation across heterogeneous social networks. In *ICDM*, pages 181–190. IEEE, 2012.

[12] A. Goyal, F. Bonchi, and L. V. Lakshmanan. Learning influence probabilities in social networks. In *WSDM*, pages 241–250. ACM, 2010.

[13] A. Goyal, F. Bonchi, and L. V. Lakshmanan. A data-based approach to social influence maximization. In *VLDB*, pages 73–84, 2011.

[14] A. Goyal, W. Lu, and L. V. Lakshmanan. Simpath: An efficient algorithm for influence maximization under the linear threshold model. In *ICDM*, pages 211–220, 2011.

[15] K. Kapoor, K. Subbian, J. Srivastava, and P. Schrater. Just in time recommendations: Modeling the dynamics of boredom in activity streams. In *WSDM*, pages 233–242. ACM, 2015.

[16] D. Kempe, J. Kleinberg, and É. Tardos. Maximizing the spread of influence through a social network. In *KDD*, pages 137–146. ACM, 2003.

[17] Y. Koren. Collaborative filtering with temporal dynamics. In *Communications of the ACM*, 53(4):89–97. ACM, 2010.

[18] J. Leskovec, A. Krause, C. Guestrin, C. Faloutsos, J. VanBriesen, and N. Glance. Cost-effective outbreak detection in networks. In *KDD*, pages 420–429. ACM, 2007.

[19] X. Li, J. M. Barajas, and Y. Ding. Collaborative filtering on streaming data with interest-drifting. *Intelligent Data Analysis*, 11(1):75–87, 2007.

[20] D. Liben-Nowell and J. Kleinberg. The link-prediction problem for social networks. *Journal of the Association for Information Science and Technology*, 58(7):1019–1031, 2007.

[21] G. Salton and M. J. McGill. Introduction to modern information retrieval. 1986.

[22] H. Sayyadi, M. Hurst, and A. Maykov. Event detection and tracking in social streams. In *Icwsm*, 2009.

[23] K. Subbian, C. Aggarwal, and K. Hegde. Recommendations for streaming data. In *CIKM*, pages 2185–2190. ACM, 2016.

[24] K. Subbian, C. Aggarwal, and J. Srivastava. Mining influencers using information flows in social streams. *TKDD*, 10(3):26, ACM, 2016.

[25] K. Subbian, C. Aggarwal, and J. Srivastava. Content-centric flow mining for influence analysis in social streams. In *CIKM*, pages 841–846. ACM, 2013.

[26] K. Subbian, C. C. Aggarwal, and J. Srivastava. Querying and tracking influencers in social streams. In *WSDM*, pages 493–502. ACM, 2016.

[27] K. Subbian and P. Melville. Supervised rank aggregation for predicting influencers in twitter. In *SocialCom*, pages 661–665. IEEE, 2011.

[28] J. Z. Sun, K. R. Varshney, and K. Subbian. Dynamic matrix factorization: A state space approach. In *ICASSP*, pages 1897–1900. IEEE, 2012.

[29] Y. Yang, J. Zhang, J. Carbonell, and C. Jin. Topic-conditioned novelty detection. In *KDD*, pages 688–693. ACM, 2002.

[30] P. Zhao, C. Aggarwal, and G. He. Link prediction in graph streams. In *ICDE*, pages 553–564. IEEE, 2016.