Effective Trend Detection within a Dynamic Search Context

Anat Hashavit
IBM Haifa Research Lab
Haifa, Israel
anath@il.ibm.com

Roy Levin
IBM Haifa Research Lab
Haifa, Israel
levin.royl@gmail.com

Ido Guy
Ben Gurion University of the Negev, Israel
Yahoo Research, Israel
idoguy@acm.org

Gilad Kutiel
Department of Computer Science
Technion, Haifa, Israel
gkutiel@cs.technion.ac.il

ABSTRACT

In recent years, studies about trend detection in online social media streams have begun to emerge. Since not all users are likely to always be interested in the same set of trends, some of the research also focused on personalizing the trends by using some predefined personalized context.

In this paper, we take this problem further to a setting in which the user's context is not predefined, but rather determined as the user issues a query. This presents a new challenge since trends cannot be computed ahead of time using high latency algorithms. We present RT-Trend, an online trend detection algorithm that promptly finds relevant in-context trends as users issue search queries over a dataset of documents.

We evaluate our approach using real data from an online social network by assessing its ability to predict actual activity increase of social network entities in the context of a search result. Since we implemented this feature into an existing tool with an active pool of users, we also report click data, which suggests positive feedback.

Keywords: Trends, Analytics, Tag Cloud, Word Cloud,Trend Cloud

1. INTRODUCTION

Nowadays, large social media sites such as Twitter, Facebook and LinkedIn generate huge amounts of information. These social networks expose activity streams which are composed of actions generated by hundreds of millions of users over time. As such, some characteristics are likely to change over time. In recent years, many research papers have focused specifically on the problem of detecting trends over such data [2,3,5]. In addition, since not all trends may warrant the same level of attention from all users. Some research effort was directed at detecting trends within some predefined context such as regional [2] or a predefined user profile [9]. Yet, such approaches require a context to be provided in advance.

SIGIR '16, July 17-21, 2016, Pisa, Italy
© 2016 ACM. ISBN 978-1-4503-4069-4/16/07...$15.00
DOI: http://dx.doi.org/10.1145/2911451.2914705

In contrast to such predefined contexts, users have traditionally relied on keyword search to explore textual content. However getting a list of relevant documents is only a partial solution. As an extension, Faceted search [8] allows users to explore these results by applying multiple filters. For example, in a social network settings such as Twitter, documents can be defined as microblogs and their authors can be added as facets. Using faceted search, the top-k users that appear in the microblogs of the search results can also be displayed. Other examples of facets can also be categories, topics, or when the social network contains structured data [1], entities such as the communities that the search results belong to.

In this paper we present a framework that utilizes faceted search along with a novel realtime algorithm called RT-Trends which calculates trending facets related to the search results. This approach employs a two phase solution. First, during indexing, we associate, as facets, to each document the trending candidate entities along with their corresponding information. Then, when a search query is issued, we collect and aggregate, in real time, trend scores for each candidate trend. We then rank them, and select the top-k to be displayed.

We implemented the feature of detecting trending entities within a search context into the Streamz [4] application. In our experiments we examine how well the RT-Trends algorithm can actually predict which entities exhibit increased activity levels. As further validation we analyze the Streamz query log and report on some encouraging results.

2. TREND SCORING

In this section, we describe the method for calculating trend scores for a stream of events.

2.1 Intuition

We begin by describing the intuition behind our trend scoring algorithm. Consider a community in a social network (a trend candidate in the example), which for the past month has had a relatively stable volume of activity within it. As such, predicting future activity volumes can be based on previous activity levels, even when they are not very recent. Note that the term stability here implies relatively small changes in daily frequencies. However, if a sudden trend appears in the amount of activity then we except our predicted value to significantly deviate from the actual volume of activity. This is due to the fact that our prediction does not give high significance to recent activity volumes. The idea is therefore to use this *predicated error* to derive a trend score.

2.2 Computing an Entity's Trend Score

Let $C = c_i, .., c_n$ denote the time series representing the occurrences of a trending entity candidate e in a sequence of time intervals $T = t_1, .., t_n$. We begin by assuming that the dataset behaves in a predictable manner (as described in the intuition above) and we predict its volume based on an Exponential Moving Average [6] as follows:

$$\chi_{i+1}(e) = \alpha \cdot \chi_i(e) + (1-\alpha) \cdot c_{i+1}(e) \qquad (1)$$

We define the starting condition for the recursion to be $\chi_0(e) = 0$. Note that $\alpha \in (0,1)$ is a smoothing parameter, which controls the weights of older versus newer values. As stated before, when an entity is trending there is a sudden and abrupt change in it's arrival rate (volume per time interval) which can not be predicted based on its previous behavior. In practice we set a high value to α. This captures the underlying assumption that the rate of change is predictable. If there is a trend then this calculation will result in an error. By aggregating these errors we get a quantitative measure of how much the new behavior deviates from the predicted one. The trend score can thus be based on this aggregated error by the recursive formula bellow:

$$TS_{i+1}(e) = TS_i(e) + (c_{i+1}(e) - \chi_i(e)) \qquad (2)$$

The starting condition for the recursive formula is set to be $TS_0(e) = 0$. Note that not every time series that is not predicted well by this Exponential Moving Average will produce a high aggregated error. For example, white noise will constantly exhibit prediction errors, yet when summing these errors, since there are likely to be an equal number of under predictions and over predictions, the aggregated error will be close to zero. This is good since we would not like such cases to be determined as being trendy.

2.2.1 *Optimizing the Recursion*

Since an entity is not necessarily active at each time interval in the series it is possible to optimize Equation (2) such that $TS_{i+1}(e)$ will be dependent solely on the values of $TS_j(e)$ $(j \leq i)$ for which $c_j(e) \neq 0$. This is an important optimization since indeed the volumes in each time interval yield a sparse vector. As a result, the trend score of an entity can be updated only for time slots in which it actually occurs. Let j be an index such that for every index $j < k < i$ it follows that $c_k(e) = 0$ and $c_j(e) \neq 0$. First, note that based on Equation (1) as $c_k(e) = 0$ it follows that $\chi_{k+1} = \alpha \cdot \chi_k(e)$ and therefore:

$$\chi_{i+1}(e) = \alpha^{i-j} \cdot \chi_j(e) + (1-\alpha) \cdot c_j(e) \qquad (3)$$

We can now develop Equation (2) as follows: (Note that we use a shorthand notation here in which $*_i$ denotes $*_i(e)$).

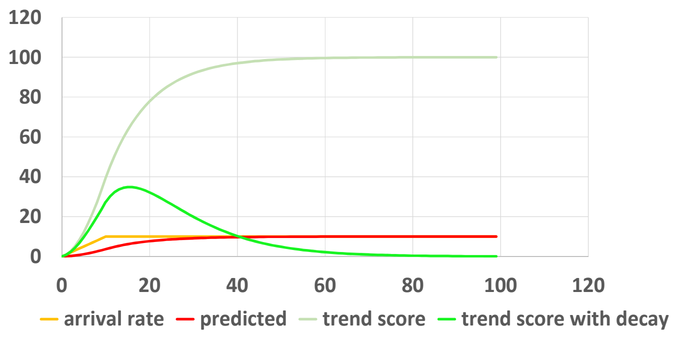

Figure 1: The trend score is based on the difference between the actual and predicted arrival rate. The actual arrival rates in this Figure have been synthetically generated for the purpose of illustration.

$$\begin{array}{lll}
(a) & TS_i = & TS_{i-1} + (c_i - \chi_{i-1}) = \\
(b) & & c_i + TS_{i-2} - \chi_{i-2} - \chi_{i-1} = (\text{since } c_{i-1} = 0) \\
(c) & & c_i + TS_{i-3} - \chi_{i-3} - \chi_{i-2} - \chi_{i-1} = \\
\vdots & \vdots & \vdots \\
(*) & & c_i + TS_j - \chi_j \cdot \sum_{k=0}^{i-j-1} \alpha^k
\end{array}$$

We can now rewrite Equation (2) based on previous values that have been calculated only when the entity e occurs within the corresponding time window.

$$TS_i(e) = c_i(e) + TS_j(e) - \chi_j(e) \cdot \sum_{k=0}^{i-j-1} \alpha^k \qquad (4)$$

2.3 Trend Decay Factor

In Figure 1, we see an example of the gap between the predicted arrival rate, based on the exponential smoothing method proposed above and actual arrival rate of an entity as it becomes a trend. The figure also shows the trend score as calculated by Equation (4). Note that the trend score increases as the difference between the actual and predicted rate grows and levels when the entity's arrival rate convergence to a new constant value. The problem is however, that this new constant value is high and is the result of mispredictions (errors) that have occurred for older values. In fact, note that for values of around $i = 30$ (in the x-axis) the predictions become very accurate, i.e., the arrival rate no longer deviates from its predicted value yet the trend score remains high due to the fact that the entity "was" trending in the past. We therefore need to add a decay factor to "forget" past trends. Therefore, we introduce a slight modification to Equation (4) by multiplying it by a *Decay Factor* $\beta \in (0,1)$.

$$TS_i(e) = \beta \cdot (c_i(e) + TS_j(e) - \chi_j(e) \cdot \sum_{k=0}^{i-j-1} \alpha^k) \qquad (5)$$

The "trends with decay" line in bright green now shows the new trend score with the decay for the example of the trending entity presented in Figure 1. We now see that once the entity's rate is correctly predicted the trends score starts to slowly decrease, as would be expected.

3. ALGORITHM AND EVALUATION

In this section we describe our dataset and the algorithm that calculates trend scores based on the description in Section 2 over our data and within a faceted search context. We also present our experimental evaluation of the results.

3.1 The Dataset and Environment

To construct the dataset for our experiments we used social data from the activity stream of an enterprise social network called IBM-Connections [1]. This data consists of more than 800,000 individuals including employees, contractors and ex-employees. The activities consist of generating and modifying several types of web entities, such as Blogs, Wikis, Files and Discussion Forums. Another web entity form is a Community [7] which groups together Blogs, Wikis, Files and Forums pertaining to a certain topic.

To keep things simple, we defined our documents as Blogs, Wikis, Files and Forums and, since each of them is contained within a community, we define the communities as our trending candidates (which are added as facets).

In addition, we implemented the "in-context trending communities" feature into a web application called Streamz [4] which is deployed in our organization and has more than a thousand internal users. This allowed us to track click rates for each trending community shown based on its rank in the top-5 trends results (1-5) including the overall clicks ratio of the trends feature in general.

3.2 Application in Search Context

We next describe how we implemented our trends score within a search context using the Apache Lucene Open Source search framework. During the indexing phase a document is created for social entities (in our case Blogs, Wikis, Files and Forums) and the unique id of the containing community is associated to that document as a facet [1]. Note that we could similarly add other associations like categories, topics, users and others using the same method. The document along with its facets are then indexed. In addition, during indexing we also calculate our time intervals span denoted $T = \{t_1, t_2, \ldots, t_n\}$.

The next phase occurs during search. Given a search query q we retrieve the matching documents D_q along with their associated facets denoted $F(D_q)$. For a given facet $f \in F(D_q)$, let $VT_f = \{VT_{i_1}, VT_{i_2}, \ldots, VT_{i_{fn}}\}$ denote the set of non-zero volumes per time intervals in T, i.e., VT_j represents the number of times facet f appears in $F(D_q)$ within time interval $T_j \in T$. Note that zeros are not needed due to the trend score calculation optimization described in Section 2. We next compute the facet's trend scores. This can be done efficiently, in memory, by using Lucene's facets mechanism as next described. We start by mapping each facet to its time interval in T and then aggregate the counts to find VT_f for each facet. This represents the volume within the corresponding time interval. Each VT_f is maintained in ascending order and is therefore a time series which we can apply our trend score algorithm over. The trend score algorithm of each VT_f is calculated in parallel. As the trend scores are updated we also maintain a top-k list of the 5 trends with the highest trend scores. Based on trial and error we selected the values for the trend scoring to be α , β which we report in the subsections below.

[1]https://lucene.apache.org/core/4_0_0/facet/org/apache/lucene/facet/doc-files/userguide.html

3.3 Evaluation Based on Prediction

In order to evaluate the RT-Trends algorithm we examined its ability to identify trending communities that contain documents (Blogs, Wikis, Files and Forums) from the search results of a given search query. We evaluate this by checking to see if, in retrospect, these exhibit an actual increase in their activity volume. Specifically, we examined the algorithm's ability to predict a volume increase within a predefined time span Δ, called the *Query Activity Time Span*, from the moment the search query is issued. Given a community c and a search query at time t, we compare the volume of activity in the community c during the time window before the query, $[t - \Delta, t)$, namely the *Prequery Activity* to that after the query, $[t, t + \Delta)$, herein the *Postquery Activity.* If for a community c the postquery activity is greater than the prequery activity we consider c to be an *Accurate Prediction* with respect to the query and the activity time span. To make sure the trends are meaningful, we also report the average factor by which the postquery activity actually increases.

In our experiments, we distinguish between short and long-term trends. The former represent current and immediate topics of interests in the organization. Announcements about new appointments, annual report publishing and internal conferences are just a few examples of events that can trigger short term trends. These events are classified as short term since they need to be quickly detected and their effect usually subsides within a few days. Long term trends, on the other hand, suggest slower but often more significant changes that affect the organization. Examples include adopting new technologies, releasing new products, and starting a new line of business. Oftentimes, such trends tend to exhibit slow and gradual changes but are nevertheless quite meaningful. Taking this into account, we separately examined how well the algorithm can predict both types of trends.

We began by selecting a random sample of user queries from the query log of the Streamz application. Note that these queries are typically more than just a set of keywords. They commonly also contain constraints on employees or departments as well as combinations of all the above. For short term trends, we then picked the date and time d_s of each day within a time period of one month (the month was selected arbitrarily) and set the query activity time span Δ_s to be twelve hours. The queries above were then issued while limiting them to return results before the date d_s. For each retrieved community, the prequery activity was compared to the postquery activity. Similarly, for long term trends we picked a date and time d_l at the beginning of every forth day in a time period of three months. We then applied the RT-Trends algorithm as well as two baseline algorithms to retrieve the trending communities relevant to the given query. The first baseline algorithm, called *Random*, merely selects ten communities relevant to the query by random (in fact this is not really an algorithm, its merely

Trend Type	Random	Volume	RT-Trends
Short Term (Δ_s = day)	17.6%	39.0%	42.4% ($\alpha = 0.999, \beta = 0.999$), |t|=minute
Long Term (Δ_s = month)	22.2%	25.8%	54.2% ($\alpha = 0.7, \beta = 0.765$), |t|=day

Table 1: Average percentage of accurate predictions for each algorithm for short and long term trends.

the ratio of communities whose postquery activity is greater than its prequery activity). The second baseline algorithm called *Volume* selects ten communities from all the relevant communities such that their volume of activity in a given time window is the highest. For short term trends this time window is defined to be the day prior to the query date and for long term trends it is defined as the prior month. We summarize the average percentage of accurate predictions made by each algorithm for short and long term trends in Table 1. The table also includes the activity time span and values of α and β, and the time interval size denoted $|t|$ used for the RT-Trends algorithm.

3.3.1 Results Analysis

We start by looking at the, more important, long term trends. Note that the volume of activity in the previous month is a very weak indicator for long term trends whereas the RT-Trend algorithm, which examines a finer granularity, is able to significantly outperform both Random and Volume by almost a factor of 2.5. Intuitively, the key difference lies in the fact that the RT-Trends algorithm is based on a weighted moving average which gives more weight to recent activities.

For short term trends, the Volume algorithm performs much better than Random. This means that the activity volume of the previous day is a significant indicator for the activity volume of the next day. RT-Trends does not significantly outperform Volume in detecting short term trends since looking at levels of activities of smaller time units than a single day is not that significant. The reason is that in our data the amount of activity in a single community per one day is not high enough to merit such a fine-grained granularity. In addition, short-term trends are somewhat noisier and thus more difficult to predict than long-term trends.

We also report that the average factor by which the activity in communities found trendy by RT-Trends is 2.354 for short term trends and 3.057 for long term trends. That is, the algorithm finds communities that exhibit a significant increase in their activity.

3.4 Analysis of User Log Data from Streamz

When displaying search results Streamz also depicts what we refer to as a **Search Analytics Box**. This box contains information about the search results such as the most active people, topics, and the new addition of trending communities. Since the launch of the trends feature in Streamz we tracked user activity and found that in 19.8% of the searches, the search analytics box was clicked. Out of these clicks, 80% were on the trends feature, indicating that users found this feature to be interesting. As the ranking of the trending communities in the search context is based on the trend score, we expect that the number of user clicks should correlate with the rank. Table 2 depicts the percentage of clicks on the trending communities by users based on their rank in the results. The main point of this is to show that there is no evidence of mis-ranked communities by the RT-Trends algorithm.

Rank	1	2	3	4	5
Clicks	38.5%	25.7%	15.6%	11.0%	9.2%

Table 2: Percentage of user clicks on trending communities based on their rank in the results.

4. CONCLUSION

In this paper, we introduce a new problem, namely finding trending entities, in real time and within a dynamic search context which is not known in advance. We devise a novel framework and an algorithm called RT-Trends, which retrieves scores, ranks, and selects the top-k trending items that are relevant to the search results. We implement the algorithm using Apach Lucene's Faceted Search and deployed it into a web application used within the organization to track its usage. The tool is used by thousands of users in our organization and enables search and analytics capabilities over streaming social media data.

In our experiments, we analyze the algorithm's ability to predict trends. We show that it predicts an actual growth in the level of activity within communities relevant to answers of user queries. Moreover, we show that the average factor by which the activity increases is also significant. Analysis of the Streamz query log shows that a high percentage of users that use the Search Analytics Box click specifically on the trends feature. Finally, we also show that a correlation exists between how we rank trending communities and the number of clicks they receive in the analytics box by users.

5. REFERENCES

[1] IBM-Connections. www-03.ibm.com/software/products/en/conn, 2007.

[2] Z. Al Bawab, G. H. Mills, and J.-F. Crespo. Finding trending local topics in search queries for personalization of a recommendation system. In *SIGKDD 2012*.

[3] N. G. Golbandi, L. K. Katzir, Y. K. Koren, and R. L. Lempel. Expediting search trend detection via prediction of query counts. In *WSDM 2013*.

[4] I. Guy, T. Steier, M. Barnea, I. Ronen, and T. Daniel. Swimming against the streamz: search and analytics over the enterprise activity stream. In *CIKM 2012*.

[5] J. Kleinberg. Bursty and hierarchical structure in streams. In *SIGKDD 2002*.

[6] S. Roberts. Control chart tests based on geometric moving averages. *Technometrics*, 1(3):239–250, 1959.

[7] I. Ronen, I. Guy, E. Kravi, and M. Barnea. Recommending social media content to community owners. Proc. SIGIR '14, pages 243–252. ACM, 2014.

[8] D. Tunkelang. aceted search (synthesis lectures on information concepts, retrieval, and services). *Claypool: Morgan*, 2009.

[9] X. Zhou, S. Wu, C. Chen, G. Chen, and S. Ying. Real-time recommendation for microblogs. *Information Sciences*, 2014.