Efficient Temporal Synopsis of Social Media Streams

ABSTRACT

Mining streaming social media, such as Twitter, requires the ongoing analysis of large volumes of data with dynamically changing characteristics. Tweets are short and repetitious – lacking context and structure – making it difficult to generate a coherent synopsis of events within a given time period. Although some established algorithms for frequent itemset analysis might provide an efficient foundation for synopsis generation, the unmodified application of standard methods produces a complex mass of rules, dominated by common language constructs and many trivial variations on topically related results. Moreover, these results are not necessarily specific to events within the time period of interest. To address these problems, we build upon the Linear time Closed itemset Mining (LCM) algorithm, which is particularly suited to the large and sparse vocabulary of tweets. LCM generates only closed itemsets, providing an immediate reduction in the number of trivial results. To reduce the impact of function words and common language constructs, we apply a filtering step that preserves these terms only when they may form part of a relevant collocation. To further reduce trivial results, we propose a novel strengthening of the closure condition of LCM to retain only those results that exceed a threshold of distinctiveness. We also propose a clustering scheme that removes redundancy from the mining results. We evaluate our work over a collection of tweets gathered in late 2012, exploring the efficiency and filtering characteristic of each processing step, both individually and collectively. By performing temporal ranking, based on information gain, we identify results that are particularly relevant to the time period of interest. Based on our experience, the resulting synopses from various time periods provide understandable and meaningful pictures of events within those periods, with potential application to tasks such as temporal summarization and query expansion for search.

WSDM '14 New York City, NY, USA

1. INTRODUCTION

Mining social media text poses unique challenges because of the text's content and form. The user generated content includes a substantial amount of social chatter, which ought to be filtered out by mining algorithms. In terms of form, social media posts are short thus undermining the effectiveness of within document term frequency counting. They also lack structure and other formatting cues, thus important terms cannot be pinpointed easily. Moreover, the rate at which users generate and consume social media posts necessitates the use of efficient algorithms to provide users by real-time mining results. To realize the benefits of social media in giving a voice to ordinary people, it is desirable to devise techniques that focuses on the content of posts.

In this work we study the use of frequent itemset mining to create a synopsis of social media streams. Frequent itemset mining is suitable to the dynamic nature of social media streams. It does not require prior knowledge of the distribution of items, nor does it require selecting a few items to monitor or a number of topics to mine. It is fast, efficient and scalable. However, frequent itemset mining is not readily suited for application on text without preprocessing steps that are difficult to apply to social media text; for example, stop word removal which requires language identification to be done before it, and will thus suffer from the poor performance of language identification on social media [4]. In our experience, directly applying frequent itemset mining algorithms to social media text results in myriads of itemsets that are mostly meaningless, repetitive or bear no information. In this work we address the shortcomings of frequent itemset mining which lead to such poor performance. As a result, the frequent itemset mining results become suitable for use as a synopsis representing what is happening on social media at different points of time. The synopsis is particularly rich because it includes the itemsets as well as the ids of posts in which they occur.

Frequent itemset mining was originally proposed as a preliminary stage for association rules mining. It produces a large number of itemsets, which has to be further processed to produce a smaller number of association rules. Furthermore, the number of itemsets grows with the number of distinct items, which is particularly high in the text domain. To reduce the number of itemsets, they may be limited by setting a high frequency threshold, but this is not possible in text mining because frequencies of items follow a long-tailed Zipfean distribution. We propose a condition for selecting informative itemsets, and a clustering scheme for reducing redundancy in the mining results. Our methods exploit the

dynamics of social media and make use of the collaborative filtering that users naturally undergo on social media, by sharing interesting posts and participating in conversations.

We process the social media stream in batches of data from consecutive epochs of time. We propose a formula for ranking the itemsets in the synopsis of each epoch according to temporal novelty. An empirical evaluation of the effectiveness of our proposed methods shows that the hour by hour synopses at different times actually reflects what was happening at those times. It is easy to see that we achieve improvements over a baseline of applying unmodified frequent itemsets mining, in terms of the reduction in the number of itemsets and the quality of individual itemsets. We also make an effort to compare our synopses with those generated by a state of the art algorithm for summarization based on itemset mining, MTV [17]. However, the number of topically relevant itemsets our algorithm mines is much larger than what MTV can mine in reasonable time, inhibiting the use of standard comparison measures. The large number of topics in the mining results also impede the use of quality measures which require the creation of a gold standard, such as those proposed for evaluating key-phrase extraction algorithms [30].

For the purpose of evaluation, we show the synopses of periods of time when it is known what social media users are expected to be talking about. Specifically, the day of the U.S. presidential elections, and periods in which Google Top Charts[1] show high volume of queries about certain celebrities. The synopses of all hours in the last 3 months of 2012 are available for download[2].

As a merit of using frequent itemset mining, the synopses include itemsets about topics with sustained user interest as well as a spike of interest – unlike trending topics[3] [18] which are limited to topics showing spikes of interest. The mined itemsets provide the vocabulary associated with events and can be used as a preliminary step for search and summarization. For example, the collection of mining results from different epochs of time can be used for temporal query and document expansion [7, 9]. The results from each epoch can be treated as a document, facilitating the creation of a "temporal profile" [12] of the query or a document being expanded. For summarization, frequent itemsets can provide a good foundation for summary creation. While the frequent itemsets themselves are not summaries, since they lack qualitative properties such as coherence and cohesion, the results are understandable at the user level.

The rest of the paper is organized as follows. The next three sections provide necessary background. We start by discussing related work in section 2, we then explain frequent itemset mining and the algorithm on which we build our work in sections 3 and 4 respectively. In sections 5 and 6 we propose solutions to the problems faced when applying frequent itemset mining to social media text. Section 5 deals with the large number of language constructs and reduces the number of itemsets. In section 6 we propose a condition for selecting itemsets and filtering out ones that are not *distinct* from their subsets, and we propose a clustering that merges together similar itemsets into *strongly closed itemsets*. In section 7 we introduce a ranking formula specially tailored for ranking *strongly closed itemsets*. We use this formula to rank itemsets and construct the examples used for evaluation. Finally, section 8 concludes our presentation and suggests future directions.

[1] http://www.google.ca/trends/topcharts#vm=chart&cid=people&geo=US&date=201211

[2] http://blinded.blinded.com/blinded/thesis_results/twitter_synopses/1hr+30min_ngram5-relsupp10_oct-nov-dec/

[3] http://blog.twitter.com/2010/12/to-trend-or-not-to-trend.html

2. RELATED WORK

Frequent itemset mining comprises a large body of work that goes back to the early 1990s. We cover the topic only briefly, as the focus of this paper is not frequent itemset mining but rather its adaptation to social media text. The original Apriori algorithm [2] and algorithms based on it suffer performance degradation and large increases in memory requirement when the number of distinct items is high. These limitations are caused by a candidate generation bottleneck, as explained later. Another well-known class of mining algorithms are the FP-Growth [11] based algorithms. FP-Growth skips the candidate generation step, and instead creates a succinct representation of the data as a frequency ordered prefix tree called the FP-tree. An FP-tree imposes the invariant that within each branch the frequency is non-increasing from the root down to the leaves. The memory requirements of the FP-Growth algorithm suffers from the sparsity of data, since the data structure is succinct only if it can find common prefixes within the constraints of its invariant. Another algorithm, not as widely used but robust against data sparsity, is Linear-time Closed itemset Mining (LCM) [25], which we will describe in detail in section 4. As a starting point for our work, we use the implementation of LCM submitted to the workshop for Frequent Itemset Mining Implementations (FIMI) in 2004 [13], which was the workshop's award winner.

The problem of having many itemsets, with redundancy and noise, can be addressed by picking out representative itemsets that satisfy a certain condition that reduces redundancy in the mining results (itemsets and their support information). The closure condition [20] is a prominent condition upon which many other conditions are based. Most similar to the *distinct* itemsets we propose in this paper are the δ-covered [27] and the δ-free [5] sets. The δ-covered condition is exactly the opposite of the *distinct* condition, and it "relaxes the closure condition to further reduce pattern set size" [16]. The δ-free condition is similar to the *distinct* condition, but it is used to eliminate different itemsets, since its motivation is to provide a compressed representation of itemsets by sacrificing support information.

Another approach to picking out itemsets is choosing ones that can be used to compress the data. The KRIMP algorithm [26] is a good example of methods that follow this approach. Our goal is different because we aim to filter out itemsets pertaining to personal updates, which make up a large portion of social media data. A similar goal is sought by the Maximally informaTiVe itemsets (MTV) algorithm [17]. MTV uses a maximum entropy model to judge if an itemset is redundant, finally choosing the top K itemsets according to the KL-Diverge between the itemset's frequency and the model's estimate. In this work we focus on efficiency, allowing the summary to include larger numbers of itemsets than the values of K for which MTV is efficient. This is crucial for scalability to the volumes of data typical of social media streams. However, we compare our results to the ones obtained from MTV.

Likewise, Yan et al. [28] choose K *master* patterns as rep-

resentatives of the data. A *master* pattern is the union of all itemsets in a cluster, similar to our proposed *strongly closed itemset*. While the use of clustering is similarly motivated by the trivial difference between itemsets, the K *master* itemsets have to cover the whole data, unlike the *strongly closed itemsets* which actually avoid clustering together different topics or different opinions within a topic.

Our work complements the work done by Yang et al. [29] for using frequent itemsets to build a temporal index on social media posts. They use frequent itemsets to decrease the space requirements of the index, but they choose itemsets to include in the index based on their utility in compressing the data. We propose methods to choose itemsets that are topically relevant, making use of the nature of social media.

The use of frequent itemsets for improving search performance has been considered before. For example, in Li et al. [15] itemsets are mined from paragraphs of newswire text, and are used to determine term weights for query expansion. Improvements in performance have been achieved by using itemsets taken from a training set of related documents, as well as ones from unrelated documents. Related methods for term weighting were used in pseudo-relevance feedback for Twitter search [1, 14], and achieved substantial improvements over a baseline. Our work can provide such methods by a short ranked list of itemsets.

Existing techniques for summarization and topic extraction were adapted for social media. Zhao et al. [30] build summaries based on an adaptation of Latent Dirichlet Allocation (LDA) for Twitter. Parikh and Karlapalem [19] extract topics from Tweets using a simplified state machine for burst detection. The short length of social media text is overcome by applying the adapted techniques on large batches of posts made in a long period of time. Our method processes data in batches as small as an hour worth of posts, resulting in temporally relevant synopses. Petrović et al. [21] used an adaptation of Locality Sensitive Hashing (LSH) to perform first story detection at scale. Their system clusters tweets efficiently, but it does not achieve high precision in determining if a tweet cluster pertains to an event or not.

The use of Twitter to provide a summary of a specific event has also been studied. Chakrabarti and Punera [6] track tokens appearing in tweets tagged by the name of an American football team, and use them to create a summary of a game using statistically improbable phrases. Sharifi et al. [23] also use phrases from tweets about a user specified event. Their method ends up to be very similar to frequent itemset mining, but uses a graph of terms.

3. FREQUENT ITEMSET MINING

3.1 Preliminaries

Classically, frequent itemset mining is applied to a *database* of *transactions* made at a retail store. This terminology is suitable for market basket data and we retain it out of convention, even though we are mining text where the terms "corpus" and "document" are normally used. Because of the dynamic nature of social media, rather than giving the whole database as input to mining algorithms, the input is an *epoch* of data; data with timestamps within a certain period of time. The epoch's *span* is the length of this period in hours, and the *volume velocity* at this epoch is the number of transactions in the epoch divided by its *span*.

A *frequent itemset* is a set of items that occur together a number of times higher than a given threshold, called the *support* threshold. We adapt the support threshold to the dynamicity of the *volume velocity* at different times of the day. We define the *minimum support threshold* as the threshold at the hour of the least *volume velocity* during the day. The *minimum support* is supplied as an absolute number, a, and then converted to a ratio, $\alpha = \frac{a}{\mathbf{avg}(\min_{day}(vol.\,vel.))}$. The actual support threshold used for mining any given epoch is thus α multiplied by the number of transactions. We now introduce the notation used in this paper:

- $W = \{w_1, w_2, \ldots, w_n\}$: The set of all items appearing in an *epoch*. We use term N-grams in this paper.
- $t_a = \{w_{a1}, \ldots, w_{am}\}$: A transaction made up of a set of items. Each transaction has a sequential id, denoted by the subscript letter, derived from its timestamp.
- $E^{span} = \langle t_a, t_b, \ldots, t_v \rangle$: An epoch of data of a certain span, such as an hour, made up of the sequence of the transactions created within this hour.
- $s \subset W$: An itemset; any possible combination of items.
- $T_s = \{t : t \in E \, and \, s \subseteq t\}$: All transactions containing itemset s. We refer to it as the itemset's postings list.

3.2 Fundamentals

The two basic operations of frequent itemset mining algorithms are *candidate generation* and *solution pruning*. The original Apriori algorithm by Agrawal et al. [2] generates candidates of length K (K-itemsets) by merging frequent itemsets of length (K-1) ((K-1)-itemsets) that differ in only 1 item, usually the last item given a certain total ordering of items. By using only the frequent (K-1)-itemsets for generating candidate K-itemsets, many possible K-itemsets are implicitly pruned, based on the downward-closure property: all subsets of a frequent itemset must be frequent. This approach still can generate a large number of candidates, especially in early iterations of the algorithm. Consider, for example, the generation of candidate 2-itemsets from a database. This generation requires producing all unordered pairs of 1-itemsets (terms), after pruning out rare ones with frequency less than the support threshold. In many domains, including text mining, the number of frequent 1-itemsets is large enough to prohibit generating a number of candidates in the order of this number squared. In text mining, a rather low support threshold has to be used, because the frequency of terms follow a long-tailed Zipfean distribution.

4. BASIC ALGORITHM

To overcome the bottleneck of *candidate generation*, many proposed algorithms take hints from the transaction space rather than operating blindly in the item space. Some of these algorithms operate by traversing a data structure representing the transactions [11]; others generate itemsets having specific properties that help in pruning out more candidates. In this paper, we expand on LCM [25], an algorithm based on a property of a class of itemsets called *closed itemsets* [20]. A closed itemset contains any item that is present in all the transactions containing this itemset. A formal definition of closed itemsets is given in equation 1:

$$\mathcal{C} = \{s_c : \, s_c \subset W \, and \nexists \, s_d \, where \, s_c \subset s_d \, and \, |T_{s_c}| = |T_{s_d}|\} \quad (1)$$

The properties of closed itemsets are as follows:

1. Adding an item to a closed itemset reduces its support.
2. A subset of a closed itemset is not necessarily closed, but one or more closed subset must exist for any itemset (formally this could be the empty set, given that any item that appears in all transactions is removed in a preprocessing step).
3. If a closed K-itemset can be extended any further then one of its supersets will be closed, however not necessarily a (K+1) superset. Itemsets that cannot be extended any further are called *maximal itemsets*, which form a subclass of closed itemsets.

Besides being much smaller than the solution space of frequent itemsets, the solution space of closed itemsets can be navigated efficiently. By using an arbitrary total ordering of items, any closed itemset can be considered an extension of exactly one of its subsets. Thus, only this subset is extended during candidate generation. All other subsets do not need to be extended by items that would lead to the longer closed itemset. This property is called *prefix preserving closure extension (PPC-Extension)* and it was proposed and formally proved by Uno et al. [25]. *PPC-Extension* is achieved by following three rules, which we state after a few definitions to facilitate their statement. First, an item is *larger/smaller* than another item if it comes later/earlier in the total ordering. This terminology comes from the fact that LCM is most efficient if the items are ordered in ascending order of their frequency. Second, the *suffix* of an itemset is one or more items which have to be removed to get an itemset with greater support. Notice that they will necessarily be at the end of the itemset, regardless of the total ordering. Finally, we call the first item added to the suffix of the itemset its *suffix head*. With this terminology, the rules for *PPC-Extentsion* are:

1. An itemset must be extended by every item that occurs in $T_{itemset}$, except items which are *smaller* than its *suffix head*; extending by *smaller* items will lead to closed itemsets already generated in an earlier step.
2. After adding an extension item, w_e, to an itemset, s, we add all other items that appear in all transaction containing $s \cup \{w_e\}$. The added items are the *suffix*.
3. If all items in the *suffix* are *larger* than the *suffix head* then add the itemset to the solution. Otherwise, prune this solution branch; all closed itemsets within this branch have already been generated.

Table 1 is an example of how *PPC-Extentsion* is used to generate closed itemsets starting from the *1*-itemset {'barack'}. The upper table enumerates $T_{\{\text{'barack'}\}}$. The lower table shows steps of itemsets generation. The current itemset along with its frequency is in column 2. Itemsets marked by an (*) are the closed itemsets that are part of the solution. The suffix of the itemset is shown in *italic*. All possible extension items and their frequencies are in column 3. Extension items that are *smaller* than the *suffix head* are shown with a line striked through them. For each itemset, the extension items are kept so that it is known which extension item is next in turn to be added. An item is bolded when its turn to be added has come. After adding each item, a pass is done on $T_{itemset}$ to enumerate and count possible extension items. To enforce a support threshold, infrequent extension items would be removed after counting, but in this example there is no such threshold. Finally, column 4 is a comment explaining each step.

Table 1 shows that the number of steps is linear in the number of closed itemsets, and the only additional storage required, besides storage for the documents, is that required for possible extension items. Of course, this is a simplified example, but it shows in essence how LCM achieves its low run time and memory requirements. We refer the interested reader to Uno et al. [25] for a theoretical proof that the algorithm runs in linear time in the number of closed itemsets, and that this number is quadratic in the number of transactions. Performance on a real dataset is shown in section 5. We proceed by describing how to implement this algorithm.

4.1 Implementation Details

We show in algorithm 1 how to implement LCM and PPC-Extension using an inverted index. The inverted index is part of the standard text search infrastructure, and it is also a good representation of textual data. Other representations that are designed for databases in general might not be well suited for such sparse data with a large number of items.

The algorithm takes as input an epoch of data and a support threshold as a ratio α. It outputs the closed itemsets with support above the threshold. Along with each itemset in the solution, it also outputs the transactions in which it occurs – which is represented as $\langle items, T_{itemset} \rangle$. The symbol $\succ$ denotes that the lefthand side succeeds the righthand side in the total ordering of terms.

Input: α: Dynamic support ratio
Data: E: Epoch of data
Result: C: Closed itemsets having support α within E

```
C ← {⟨∅, E⟩} ;                    // ∅ is a closed itemset
X ← Inverted index of E;
foreach w ∈ X.tokens do
    T_{w} ← X.postingsList[w];
    if |T_{w}| ≥ α |E|/E.span then LCM({w}, w, T_{w}) ;
end
return C;
Function LCM(s: Current itemset, w_sh: Suffix head,
T_s: Transactions (tweets) containing s) is
    freq[1 ... w_n] ← 0;   // Initialize freq. counts
    suffix ← {w_sh};
    foreach t ∈ T_s do
        foreach w ∈ t do
            freq[w]++;
            if freq[w] = |T_s| then suffix ← suffix ∪ {w} ;
        end
    end
    if ∃v ∈ suffix: w_sh ≻ v then return;
    C ← C ∪ {⟨s ∪ suffix, T_s⟩};
    foreach v ≻ w_sh and v ∉ suffix do
        if freq[v] ≥ α × |E| then
            T ← T_s ∩ v ; // Results of query s AND v
            LCM(s ∪ suffix ∪ {v}, v, T)
        end
    end
end
```

Algorithm 1: LCM frequent itemsets mining

Doc. Id	Document	Doc. Id	Document
a	barack & mitt	b	barack obama & mitt romney
c	barack obama & romney	d	barack obama

Documents (two per row)

Step	Current Itemset	Possible Extension Items	Comments
1	{barack} (4)	mitt (2), obama (3), romney (2)	Items in $T_{\{`barack'\}}$ are counted.
2	{*barack*} (4)*	mitt (2), obama (3), romney (2)	Rule 3: suffix is ordered, emit itemset.
3	{*barack*} (4)	**mitt** (2), obama (3), romney (2)	Items are ordered lexicographically.
4	{barack, *mitt*} (2)*	**obama** (1), romney (1)	Extension items reenumerated & counted.
5	{barack, *mitt*, obama} (1)	romney (1)	Rule 2: 'romney' appears in all $T_{itemset}$.
6	{barack, mitt, *obama, romney*}(1)*		Rule 3: 'obama' is the *suffex head*.
7	{*barack*} (4)	mitt (2), **obama** (3), romney (2)	Nothing more to add, back to {'barack'}.
8	{barack, *obama*} (3)*	~~mitt~~ (1), **romney** (2)	Rule 1: skipping 'mitt', adding 'romney'
9	{barack, obama, *romney*} (2)*	~~mitt~~ (1)	Rule 1: Nothing more to add.
10	{*barack*} (4)	mitt (2), obama (3), **romney** (2)	Back to {'barack'}, adding 'romney'.
11	{*barack*, romney} (2)	mitt (1), obama (2)	Rule 2: add 'obama' after 'romney'.
12	{barack, *romney, obama*} (2)	~~mitt~~ (1)	Rule 3: suffix is not ordered, prune.
13	{*barack*} (4)	mitt (2), obama (3), romney (2)	All possible extension items were added.

Closed itemsets containing 'barack'

Table 1: Generation of closed itemsets by Prefix Preserving Closure Extension

5. MINING SOCIAL MEDIA

Throughout this paper we use data collected from the Twitter public stream[4] since October 1st, 2012. We collect only tweets written in the Latin script to facilitate tokenization using white space and other word boundaries. The average number of tweets collected per day is 2,779,091.82 tweets. We collect only the tweet text to avoid reliance on any features specific to a certain social medium, and to make the algorithms applicable to other media where text is short such as YouTube comments. The only preprocessing we performed was to remove duplicate original tweets (not retweets) using a Bloom filter. This filtering removes spam tweets sent by botnets, averaging at 2.86% of the stream.

We apply the algorithms to epochs of data, so they are not strictly stream processing algorithms. However, we regard the process as mining a sliding window that is moved forward by time steps of short span. The time step must be longer than the time needed to mine an epoch of data, and the performance of our algorithms makes it possible to use a time step of a few seconds for epochs up to a day long. Figure 1 shows the runtime of LCM on epochs of increasing length, and we will show in section 6.3 that our extensions do not degrade performance. The times reported in figure 1 are averages across all epochs of the specified length in the last 3 months of 2012, using a time step that is half the epoch length. The *minimum support* threshold we use throughout this paper is 10 occurrences in the hour of least *volume velocity*, which translates into $\alpha = 0.0002$.

In the rest of this paper we mine epochs of 1 hour span. The reason behind this choice is our observation that the number of closed itemsets mined from epochs of span 1 hour or more, at the same support threshold, remains the same. This indicates that itemsets mined from shorter epochs of social media text are not included in the results of mining longer epochs. Therefore, the epoch span should be minimized. However, when the epoch span is shorter than an hour the frequency required to surpass the support threshold becomes very low, and number of mined itemsets increases, with many noise itemsets appearing in the results.

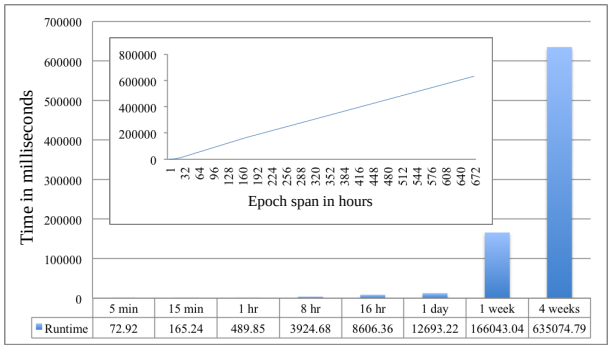


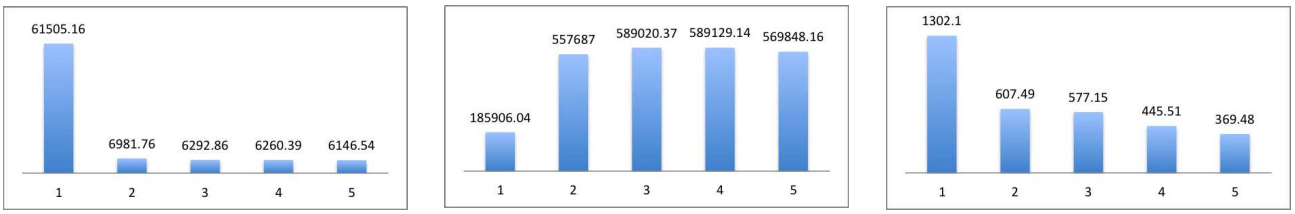
Figure 1: Mean runtime at different epoch spans

Regardless of the length of the epoch, many mined itemsets are combinations of function words. In the next section, we outline how we reduce the number of itemsets and eliminate the effect of function words by N-gram filtering.

5.1 Mining Term N-grams

A large number of itemsets are language constructs that bear no information, such as "such as". By treating sequential language constructs, and any other multiword expression, as one item we eliminate a large number of such itemsets. We can also eliminate itemsets that are made up of all the different fragments of the language construct along with other items; for example, {we, did, it, #teamobama} can produce 10 other combinations of length 2 or more. There are many measures of association that can be used to detect multiword expressions, but each measure is good only under certain conditions [22, 24]. After preliminary experiments with various measures, we determined that the best performance could be obtained by tokenizing the documents into

[4] https://dev.twitter.com/docs/streaming-apis/streams/public



(a) Mean number of closed itemsets

(b) Mean number of distinct items

(c) Mean runtime in milliseconds

Figure 2: Effect of the increasing maximum N-Gram length on results of mining of 1hr epochs of data

term N-grams with varying N.

We use term N-gram tokens such that N-grams of high probability are replaced by (N+1)-grams, resulting in a distribution with no high peaks. An N-gram is considered to have a high probability if its maximum likelihood estimate from a background model is higher than a threshold η_N. A background language model built from a long epoch of data from the same stream is used for probability estimation. The tokenization of a tweet starts by tokenizing it into unigrams, then each unigram of high probability is replaced by two term bigrams – by attaching to it the unigrams before and after it. We keep replacing N-grams of high probability by two (N+1)-grams until there are no more such N-grams.

The threshold of high probability, η, is different for each value of N. The threshold for unigrams is determined as follows: We pick a topical term that is known to steadily appear with a rather high frequency, and is talked about in all languages; for example, 'obama'. The maximum likelihood estimate of the probability of the term 'obama' is P("obama") = 0.0001, which we use as η_1. At each N, the probability threshold is adjusted to account for the increase in the number of tokens and the overall increase in the grand sum of counts (caused by overlap). The adjusted η_N is:

$$\eta_N = \eta_1 \times \frac{\sum_{\{w:\, w \in W \text{ and } w.length \leq N\}} freq(w)}{\sum_{\{v:\, v \in W \text{ and } v.length = 1\}} freq(v)} \quad (2)$$

Figure 2 shows the effect of increasing the maximum length of N-grams from 1 to 5 on the number of closed itemsets, the number of tokens, and the runtime of mining one hour of data. The values shown are averages across all one-hour epochs in the month of November 2012. Figure 2(a) shows that the number of itemsets drops by about 90% by using variable length N-grams, which overcomes the problem of mining language constructs while limiting the increase in the sparsity of the data. Figure 2(b) shows that the number of distinct items increases substantially as N goes from 1 to 2, then continues increasing slightly until it starts decreasing at N $\leq$ 5. The decrease happens because some tweets are tokenized into less 5-grams than 4-grams, eliminating 4-grams not appearing elsewhere. Figure 2(c) shows that runtime also decreases as N goes from 1 to 5, since the runtime of LCM is proportional to the number of closed itemsets, and is not affected by the increasing sparsity of data.

6. SELECTING ITEMSETS

In the previous section, we discussed our handling of function words, using a technique that exploits LCM's tolerance to sparsity. After applying this technique, the average number of itemsets mined from an hour of twitter data drops from 61,505 to 6,146. However, there is still redundancy in the itemsets. For example, figure 3 illustrates itemsets related to Donald Trump's famous tweets in reaction to Obama's victory in 2012[5]. Each area in the figure represents the transactions containing the itemset formed by concatenating the items in all intersecting ellipses.

The closed property of an itemset is easily satisfied by an itemset created through the modification of transactions that contain another closed itemset. For example, if a transaction containing a closed itemset is modified by removing one of its items, another closed itemset with enough support is immediately formed. While an update operation is not supported in the model of frequent itemset mining, a similar effect happens when people retweet, or even when they are writing original posts about a certain fine grained topic. In case of retweeting, people actually start from the original tweet and then remove parts of it, to make space for their content or to keep only the part they want to emphasize. For example, in figure 3 the most discriminative words are "sham, and, travesty" which are quoted in most of the retweets. We can imagine that for each fine grained topic there is also a base post that represent the information or ideas within the topic. People make posts about the fine grained topic by selecting parts of the base post and adding their own thoughts. This results in many closed itemsets about the topic that are trivially different from each other.

The fact that any *maximal* itemset is a closed itemset is another *weakness*. If a closed itemset is expanded by certain items a number of times over the support threshold, this results in a maximal itemset which is closed by definition. Now consider a tweet that is retweeted hundreds of times, and a small group of people appends the same words to it. Since a low support threshold has to be used when mining text, this will result in a maximal itemset that might not be adding any information to the closed itemset. For example, in figure 3 a number of people talked about "hair" along with "sham, and, travesty". The number is higher than the support threshold, but it is relatively small within the topic.

We propose two conditions that are not as easily satisfied as the closed condition for selecting itemsets. The two conditions build on the concept of *association rule confidence*. Mining itemsets based on the confidence of rules they induce has long been recognized as a method for finding "interesting patterns" [8], but since this property is not anti-monotone it cannot be directly used. The confidence of an association rule that the presence of an itemset, $s_{anteced.}$, implies the presence of another itemset, $s_{conseq.}$, is defined as:

$$conf(s_{anteced.} \rightarrow s_{conseq.}) = \frac{|T_{s_{conseq.}} \cap T_{s_{anteced.}}|}{|T_{s_{anteced.}}|} \quad (3)$$

[5] http://www.huffingtonpost.com/2012/11/07/donald-trump-election-revolution_n_2085864.html

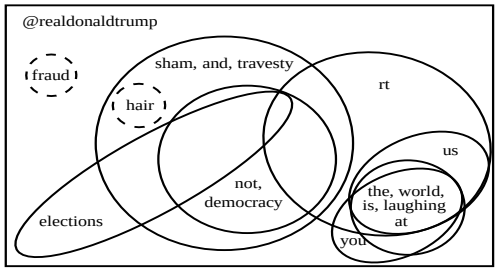


Figure 3: Closed, distinct and strongly closed sets

6.1 Distinct Itemsets

The *distinct* condition is a novel strengthening of the closed condition so that it is not as easily satisfied. We define a *distinct* itemset as a closed itemset whose frequency comprises more than a certain proportion of the frequency of its least frequent subset – we call this its *parent* itemset. The proportion is a parameter, κ, that controls the selectivity of the *distinctiveness* condition. This can be interpreted as selecting itemsets which are implied by a subset with confidence greater than κ. Formally, the set of *distinct* itemsets, $\mathcal{D}$, is defined as follows:

$$\mathcal{D} = \{s : s \in \mathcal{C} \, and \, \exists \; s_{parent} \subset s \; where \; \frac{|T_s|}{|T_{s_{parent}}|} \geq \kappa\} \tag{4}$$

In figure 3, *distinct* itemsets are illustrated in solid lines, and closed itemsets that do not satisfy the distinctiveness condition in dashed lines. It is clear from the figure that considerable redundancy remains in *distinct* itemsets.

6.2 Strongly Closed Itemsets

To overcome the redundancy in *distinct* itemsets we merge similar ones into *strongly closed* itemset clusters. This will also filter out any itemset that does not belong to a particular topic since it will not be part of a cluster. The similarity of a *distinct* itemset, s_d, and another distinct itemset, s_c, is measured as the overlap of the *transactions* containing both of them with the transactions containing s_c. A *distinct* itemset is clustered with another itemset if one exists such that the overlap exceeds a similarity threshold, which we take to be $1 - \kappa$ (the indistinctiveness of s_d from s_c). If more than one satisfies this condition, the *distinct* itemset is clustered with the one having the highest overlap ratio. When a *distinct* itemset is clustered with an itemset that is already part of a cluster, the *distinct* itemset is added to the existing cluster. Finally, the *strongly closed* itemset is the union of all cluster members, and its postings lists is the union of their postings lists. We define the desired clustering and the strongly closed itemset represented by each cluster as follows:

$$\begin{aligned}\mathcal{R} = \{r : r = \bigcup_i (s_i, s_j) \; where \; s_i \in \mathcal{D} \, and \, s_j \in \mathcal{D} \\ and \; \forall_{(s_i,s_j)} \; s_j = \mathbf{argmax}_{s_k} conf(s_k \rightarrow s_i) \\ and \; conf(s_j \rightarrow s_i) \geq (1 - \kappa) \\ and \; (s_j = r.centroid \; or \; (s_j, r.centroid) \in r)\}\end{aligned}$$

$$S_l = \{w : \; w \in \bigcup_{(s_i,s_j) \in r_l} s_i \; where \; r_l \in \mathcal{R}\} \tag{5}$$

Confidence is not a symmetric measure, but our proposed clustering makes use of the order in which PPC-Extension generates itemsets to determine which itemset should be the antecedent and which should be the consequent. If PPC-Extension uses a total ordering of items that is non-increasing in item frequency, then an itemset generated earlier should be the antecedent, and an itemset generated later should be the consequent. This builds upon the use of variable length N-grams as items, so that the most frequent items are not function words. In that case, following the non-increasing order of item frequency leads to mining itemsets related to a certain topic close to each other. Notice that PPC-Extension is equivalent to a depth first traversal of a hypothetical prefix tree of closed itemsets. That is, closed supersets of each itemset are generated before moving to an itemset that is not a superset.

The most discriminative itemset from each topic is mined before other itemsets from the same topic. If an itemset branches out into supersets related to two topics or two opinions within one topic, this itemset is mined before itemsets from either of the two branches. It should not be clustered with either of the branches. For an itemset to belong to a topic, it must contain a set of topical words. Since we are processing items in descending order of frequency, then itemsets containing multiple topical words are generated when mining the supersets of the itemset containing only the most frequent of them. According to these observations, the *topical* similarity between itemsets can be indicated by the confidence of the association rule that an itemset generated earlier implies an itemset generated later.

This clustering can be implemented efficiently using techniques similar to the ones proposed by Bayardo et al. [3]. The main ideas are to limit the comparisons to a few candidates, and to terminate the comparison early if the similarity threshold will not be met. In our case, the postings lists are longer than the itemsets, so we generate candidates for comparison by calculating similarity between itemsets. When calculating the similarity between two postings lists, we can terminate early if the difference exceeds the maximum difference permissible to achieve a similarity of $1 - \kappa$, which can be derived from equation 3.

Algorithm 2 shows a possible implementation. For each itemset, s_i, we find the itemsets produced before it and overlapping with it in one or more items. Then we find the candidate, s_c, that maximizes $conf(s_c \rightarrow s_i)$ such that the confidence exceeds $1 - \kappa$. Notice that confidence is not a symmetric measure, and we only check the confidence of the rule that the clustering candidate implies the itemset.

6.3 Performance Analysis

Unlike the original LCM algorithm, filtering low confidence itemsets requires us to keep mined results in memory to calculate the confidence of newly generated ones. The memory requirement is not large because the number of itemsets averages at about 6,000. In our experience with the Twitter data, it was enough to keep only a buffer of 1,000 itemsets when mining an hour long epoch. This is based on the observation that the number of positions between an itemset and a clustering candidate is 1,973.11 on average, with a standard deviation of 1,443.64. The large standard deviation indicates that candidates are either found close to the itemset (less than 500 positions behind), or in a position that is several thousands of itemsets behind.

Input: κ: Minimum distinctiveness threshold
Data: $\mathcal{C}$: Closed itemsets produced by LCM
Result: $\mathcal{R}$: Strong closed itemset clusters

```
for i ← 2 to |C| do
    C ← {s_c : s_c ∈ C and c < i and |s_c ∩ s_i| > 0};
    P ← {s_p : s_p ∈ C and p < i and s_p ∩ s_i = s_p};
    s_p ← argmax_{s_p ∈ P} |T_{s_i}| / |T_{s_p}| ;        // Direct parent
    if s_p = null OR |T_{s_i}| / |T_{s_p}| < κ then
        continue;                                          // Not a distinct itemset
    s_m ← null, maxConf ← 0 ;                              // Best clustering
    foreach s_c ∈ C do
        Δ ← (1 − (1 − κ))|T_{s_c}| ; // Maximum difference
        δ ← difference(T_{s_c}, T_{s_c ∪ s_i}, Δ) ;  // Stops early
        if δ ≤ Δ then                                      // Confidence ≥ threshold
            conf ← (|T_{s_c}| − δ) / |T_{s_c}|;
            if conf > maxConf then
                s_m ← s_k ;                                // Best merge candidate
                maxConf ← conf ;
            end
        end
    end
    if s_m ≠ null then
        if R[s_m] = null then R[s_m] ← s_m;                // New cluster
        R[s_i] ← R[s_m] ;                                  // Add s_i to the cluster
        R[s_m].itemset ← R[s_m].itemset ∪ s_i ∪ s_m;
        R[s_m].postingsList ← R[s_m].postingsList ∪
        s_i.postingsList ∪ s_m.postingsList;
    end
end
return R;
```

Algorithm 2: Forming strongly closed itemset clusters

Following is an analysis of the performance of the filtering conditions proposed, in terms of the number of itemsets after their application, and the runtime overhead they add on top of the runtime of the LCM algorithm. The experiments were run using a single threaded Java implementation of algorithm 2, running on a 1.4GHz processor with 2MB of cache. We apply the algorithm to the mining results of all one-hour long epochs in the Twitter data. Before filtering, the average number of itemsets mined from an hour-long epoch is 2,439.17 closed itemsets of length 2 or more; that is, excluding itemsets that are merely a frequent item.

Figure 4 show the effect of varying κ on the mean number of *distinct* and *strongly closed* itemsets. The number of *distinct* itemsets drops as the distinctiveness threshold increases. On the other hand, the number of *strongly closed* clusters increases as the similarity (indistinctiveness) threshold decreases. The dashed line shows that the number of unclustered distinct itemsets reaches zero at $\kappa = 0.5$, explaining why the number of clusters changes very slightly after that. We use $\kappa = 0.25$ in the remainder of the paper, which is an arbitrary choice based on the definition not the data. The average size of an hourly synopsis at this value of κ averages at 139.1 *strongly closed* itemsets.

Figure 5 shows the total runtime of the LCM algorithm plus filtering based on the *distinct* condition and clustering into *strongly closed* itemsets at different epoch spans. The runtime of LCM alone is also plotted for reference. We also

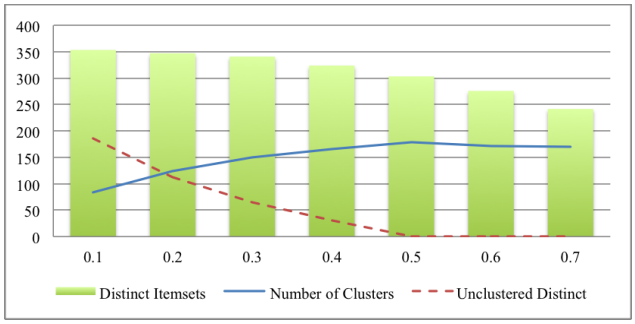


Figure 4: Effect of changing κ on mining results

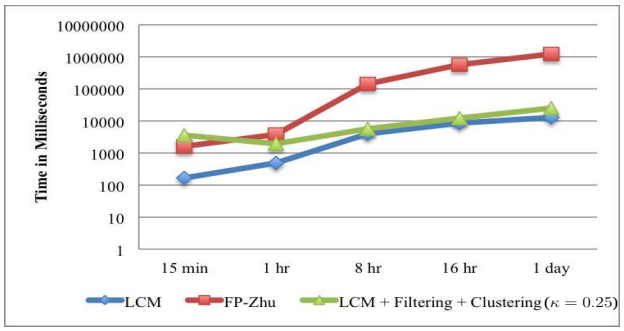


Figure 5: Runtime of itemset mining alone and with filtering and clustering at different epoch spans

plot the performance of another frequent itemset mining algorithm, FP-Zhu [10], which was the runner up at FIMI 2004 [13]. We include it to show that our extensions do not degrade the performance of LCM even in the context of competitions. The y-Axis is in logarithmic scale to keep the scale of the plot suitable for seeing slight differences. The output of LCM is the input to the filtering and clustering step, so it is affected by the number of closed itemsets produced. This explains why it takes slightly longer time for clustering results from the 15-minute epoch and then takes a constant time for epochs of a longer span.

These runtimes are achieved by using a buffer of 1,000 itemsets, which is the same setting we use when evaluating the quality of the synopsis in the next section. If all itemsets are kept in memory the runtime increases slightly and averages at 5.2 seconds for filtering a one-hour epoch.

7. TEMPORAL RANKING

The previous filtering steps reduce the number of itemsets to under 10% of their original number. In this section, we present a method for ranking itemset clusters such that they can be presented to users in rank order as a summary of events in the epoch. The ranking can also be used as input to additional search or summarization steps.

We rank *strongly closed* itemsets according to their temporal novelty, compared to a longer period of time leading up to the mined epoch – a background model. A good indicator of novelty is the point-wise Kullback-Leibler Divergence (KLD) between an itemset's probability in the current epoch and in the background model. The KLD of the probability of an itemset, s_i, in the background model, Q, from the current epoch's model, P, can be considered as the Information

Gain, IG, in moving from a prior to a posterior distrubtion:

$$IG(P(s_i), Q(s_i)) = -(H(P(s_i)) - H(P(s_i), Q(s_i))) = \sum P(s_i) \log P(s_i) - \sum P(s_i) \log Q(s_i) = KLD(P(s_i)||Q(s_i)) \quad (6)$$

To calculate the collective IG of a strongly closed itemset cluster, we have to take into account that the itemsets of the cluster are not independent. For simplicity we will consider only the pairwise dependence between every itemset and the smallest common subset, s_{min}. We also normalize by the size of the cluster, m, to avoid favouring large clusers. Hence, our ranking formula for *strongly closed* itemsets:

$$score(r) = IG(P(s_{min}), Q(s_{min})) + \frac{\sum_{j=i_1..i_m} IG(P(s_j|s_{min}), Q(s_j|s_{min}))}{m} \quad (7)$$

7.1 Empirical Evaluation

We now show examples of the synopses produced by ranking the *strongly closed* itemsets. The examples will serve as our assessment of the quality of the synopses. We make an effort to choose epochs from time periods when it is known what people should be talking about. We show how the 2012 U.S. presidential elections day is summarized by our methods. We compare our summary to the one produced by a state of the art algorithm, MTV, which generates better quality summaries than other summarization algorithms based on itemsets [17]. We also investigate the synopses of hours selected by the aid of Google Trends, assessing which popular queries were or were not reflected in our synopses.

7.1.1 U.S. Presidential Elections Day

Table 2 shows the top 3 *strongly closed* itemsets from one-hour epochs on the elections day. The itemsets shown are the first appearances of the most interesting itemsets; that is, an hour is shown only if its top 3 feature novel interesting itemsets. The first column is the beginning of the hour, EST time. The second column is the top itemsets, reordered and punctuated to be meaningful phrases by looking at a few tweets from the postings list of each itemset. The third column is a commentary to explain the itemsets, also composed according to tweets in the postings lists. The fourth column is the rank of an equivalent itemset in the top 30 itemsets mined by the MTV algorithm, if one exists.

Table 3 shows the top 3 itemsets picked by MTV, for hours when they include interesting itemsets. Each interesting itemset has an equivalent *strongly closed* itemset in the top 50 (actually less)[6]. Its rank is in the second column.

7.1.2 Google Trends

We use the Top Charts of November 2012 for the people category[7] as an aid for finding out what social media users are expected to be talking about. We observed that named entities that make it to the top 10 chart in the people category are divided into finer groups. The queries about people in each group have similar volumes over time, with peaks at the times of real world events in which the people in the

[6] http://blinded.blinded.com/blinded/thesis_results/twitter_synopses/elections-day_top50.txt

[7] http://www.google.ca/trends/explore#q=Justin%20Bieber%2C%20Taylor%20Swift%2C%20Rihanna%2C%20Nicki%20Minaj%2C%20Selena%20Gomez&geo=US&date=11%2F2012%201m&cmpt=q

Time	Itemset	Explanation	M
08:30	*get out and vote*	Encouraging participation	17
	happy elections day	Congratulations	27
	it's elections day	Excitement	-
09:30	*if romney wins*	Reflecting about possiblities	-
	of your ballot	Telling people to stop sending pictures of their ballots	-
	in line to vote	Tweeting while in line	28
19:30	*Linda McMahon*	She loses CT senate race	-
	Florida is so ...	tight or close; the vote count	-
	Romney is winning	Speculations about results	-
20:00	*New Hampshire*	Obama won in NH	9
	Obama to win / is winning	Two itemsets speculating	-
	too close to call	The results are still not clear	22
21:00	*Ohio and Florida*	Obama won in 2 more states	-
	Sarah Palin	Sarah Palin on Fox news	-
	concession speech	Todd Akin's speech	-
22:00	*The electoral college*	Republicans complaining about the voting system	9
	President of the United States	Barack Obama is the 44th president of the USA	8
	who voted for	Taking things personally	-
	once you go black you never go	A popular culture reference, missing "back"	15
22:30	*@realdonaldtrump this elections ... [complete tweet]*	Trump's "sham and travesty" tweets. This cluster merges 26 *distinct* itemsets.	1
	concede to	The loser has to concede	-
	Obama won. I ...	Reactions to the result	-
23:00	*same sex marriage*	Some states legalized same sex marriage	27
	for recreational use	CO and WA legalized weed for recreational use	21
	acceptance speech	Anticipation for the speech	26
00:30	*Delivered, sealed, signed*	Stevie Wonder's song used in Obama's campaigns	10
	The best is yet to come	Quote from Obama's acceptance speech	2
	with the flag in her [hair]	During the speech, attention goes to a woman appearing behind Obama on TV!	14

Table 2: Top 3 *strongly closed* itemsets for hours in US presidential elections day (November 6, 2012)

Itemset (with [tweet id] if the itemset is a whole tweet)	**S**
@realdonaldtrump,this,elections,...[266035509162303492]	1
we, re, all, in, this, together, ..., bo [266031109979131904]	22
@realdonaldtrump,our,country,is,...[266037143628038144]	24
URL,and,autotweet,checked,followed,me,today,unfollowed	N/A
house,of,representatives,shouldnt,...[266040877552656385]	15
URL,#android,#androidgames,#gameinsight	N/A
URL,and,autotweet,checked,followed,me,today,unfollowed	N/A
URL,#android,#androidgames,#gameinsight	N/A
@barackobama,@ryanseacrest, what, was, your, first, words, in, reaction, to, re, election, 2	36
URL,and,autotweet,checked,followed,me,today,unfollowed	N/A
the, best, is, yet, to, come	2
URL,#android,#androidgames,#gameinsight	N/A

Table 3: Top 3 picked by the MTV algorithm

group took part. Five out of the top 10 are pop artists, with high query volume at times of music awards events.

The top 30 *strongly closed* itemsets mined from hours leading to the MTV Europe Music Awards contain many itemsets about the events and the artists, especially that this event used social media to collect audience votes. Voting for the award winner is a good example of a topic where people have strongly different opinions. Such different opinions are all reported as separate *strongly closed* itemsets, because clustering in the transaction space avoids forming incohesive clusters. Moreover, different *strongly closed* itemsets from one such topic can all occupy high ranks. For example, the top 3 itemsets for 3 PM on November 9 are all variations of the itemset: {i, think, ARTIST, NAME, will, be, the, big, winner, tweet, your, pick, at, URL, #mtvema}; replacing ARTIST NAME by Justin Bieber, Katy Perry, or Lady Gaga and adding a hashtag for each artist.

One of the 5 artist does not appear in the itemset selection. The name of Rihanna never appeared in any *strongly closed* itemset. Actually it rarely appeared in any itemset of length 2 or more. This indicates that there was not enough support for any itemset with her name (of length 1) and other information. It is impossible for an itemset of length 1 to be selected, since our proposed conditions select itemsets implied by a subset with high confidence.

8. CONCLUSION AND FUTURE WORK

We have proposed adaptations for frequent itemset mining such that it can be efficiently and effectively applied to social media text. We based our work on a frequent itemset mining algorithm that is suitable for sparse data, LCM [25]. The direct application of the LCM algorithm resulted in a large number of closed itemsets (on average 61,505.16 itemsets per hour), which are dominated by language constructs and contain considerable redundancy and noise. The result of the modified algorithm is a concise selection of high quality itemsets (on average 139.1 itemsets per hour), which was shown to capture important events. Moreover, the improvement in effectiveness does not tax the efficiency of the algorithm. Thus we provided a recipe for mining frequent itemsets from social media text, starting from recommending a suitable algorithm. Our main contributions are: (1) a novel method to avoid mining itemsets that are merely language constructs, (2) a novel condition for selecting itemsets that are distinctively different, and (3) an efficient clustering scheme to reduce redundancy.

We aim to exploit the synopses for temporal query expansion in our future work. Terms from itemsets relevant to a query (or occurring in a relevant document) can be used for query expansion, thus acting as precomputed results of pseudo-revelance feedback. We also wish to explore ways to make use of the temporal signal during mining, such as when calculating similarity during clustering.

9. REFERENCES

[1] Y. Aboulnaga and C. L. A. Clarke. Frequent itemset mining for query expansion in microblog ad-hoc search. In *TREC*, 2012.

[2] R. Agrawal, R. Srikant, et al. Fast algorithms for mining association rules. In *VLDB*, 1994.

[3] R. J. Bayardo, Y. Ma, and R. Srikant. Scaling up all pairs similarity search. In *WWW*, 2007.

[4] S. Bergsma, P. McNamee, M. Bagdouri, C. Fink, and T. Wilson. Language identification for creating language-specific twitter collections. In *LSM*. ACL, 2012.

[5] J.-F. Boulicaut, A. Bykowski, and C. Rigotti. Free-sets: a condensed representation of boolean data for the approximation of frequency queries. *Data Mining and Knowledge Discovery*, 7(1):5–22, 2003.

[6] D. Chakrabarti and K. Punera. Event summarization using tweets. In *AAAI*, 2011.

[7] J. Choi and W. B. Croft. Temporal models for microblogs. In *CIKM*, 2012.

[8] E. Cohen, M. Datar, S. Fujiwara, A. Gionis, P. Indyk, R. Motwani, J. D. Ullman, and C. Yang. Finding interesting associations without support pruning. *IEEE Trans. on Knowledge and Data Engineering*, 13(1):64–78, 2001.

[9] M. Efron, P. Organisciak, and K. Fenlon. Improving retrieval of short texts through document expansion. In *SIGIR*, 2012.

[10] G. Grahne and J. Zhu. Reducing the main memory consumptions of fpmax* and fpclose. In *FIMI*, 2004.

[11] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *SIGMOD*, 2000.

[12] R. Jones and F. Diaz. Temporal profiles of queries. *ACM Trans. on Information Systems*, 25(3):14, 2007.

[13] R. J. B. Jr., B. Goethals, and M. J. Zaki, editors. *Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations*, 2004.

[14] C. H. Lau, Y. Li, and D. Tjondronegoro. Microblog retrieval using topical features and query expansion. In *TREC*, 2011.

[15] Y. Li, A. Algarni, and N. Zhong. Mining positive and negative patterns for relevance feature discovery. In *SIGKDD*, 2010.

[16] G. Liu, H. Zhang, and L. Wong. Finding minimum representative pattern sets. In *SIGKDD*, 2012.

[17] M. Mampaey, N. Tatti, and J. Vreeken. Tell me what i need to know: succinctly summarizing data with itemsets. In *SIGKDD*, 2011.

[18] M. Mathioudakis and N. Koudas. Twittermonitor: trend detection over the twitter stream. In *CIKM*, 2010.

[19] R. Parikh and K. Karlapalem. Et: events from tweets. In *WWW*, 2013.

[20] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Discovering frequent closed itemsets for association rules. In *ICDT*, pages 398–416. Springer, 1999.

[21] S. Petrović, M. Osborne, and V. Lavrenko. Streaming first story detection with application to twitter. In *NAACL-HLT*, 2010.

[22] C. Ramisch, V. De Araujo, and A. Villavicencio. A broad evaluation of techniques for automatic acquisition of multiword expressions. In *ACL Student Research Workshop*, 2012.

[23] B. Sharifi, M.-A. Hutton, and J. Kalita. Summarizing microblogs automatically. In *NAACL-HLT*, 2010.

[24] P.-N. Tan, V. Kumar, and J. Srivastava. Selecting the right interestingness measure for association patterns. In *SIGKDD*, 2002.

[25] T. Uno, M. Kiyomi, and H. Arimura. Lcm ver. 2: Efficient mining algorithms for frequent/closed/maximal itemsets. In *FIMI*, 2004.

[26] J. Vreeken, M. Van Leeuwen, and A. Siebes. Krimp: mining itemsets that compress. *Data Mining and Knowledge Discovery*, 23(1):169–214, 2011.

[27] D. Xin, J. Han, X. Yan, and H. Cheng. Mining compressed frequent-pattern sets. In *VLDB*, 2005.

[28] X. Yan, H. Cheng, J. Han, and D. Xin. Summarizing itemset patterns: a profile-based approach. In *SIGKDD*, 2005.

[29] X. Yang, A. Ghoting, Y. Ruan, and S. Parthasarathy. A framework for summarizing and analyzing twitter feeds. In *SIGKDD*, 2012.

[30] W. X. Zhao, J. Jiang, J. He, Y. Song, P. Achananuparp, E. Lim, and X. Li. Topical keyphrase extraction from twitter. In *ACL-HLT*, 2011.