Efficient Temporal Synopsis of Social Media Streams

ABSTRACT

Search and summarization of 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 straightforward 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. Finally, we perform temporal ranking, based on a simple information gain measure, to identify results that are particularly relevant to the time period of interest. 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. 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.

CIKM '13 Burlingame, California USA

1. INTRODUCTION

The nature of text in social media poses a challenge when applying traditional text mining algorithms. Text in social media is usually short, lacks context and structure, and is created at a very high rate. To tap into the benefit of social media in giving voice to ordinary people, explicit contextual information from the social graph has to be neglected. However, the collective stream from all users is overwhelmed with personal updates and timely finding posts about topics of interest requires an efficient mining algorithm. The frequent itemset mining family of algorithms is very fast and efficient, however it is not readily suited for application on text. First of all, the number of itemsets mined is large and grows with the number of distinct items - which is particularly high in the text domain. Frequent itemset mining was originally proposed as a preliminary stage to association rules mining, which sifts through the numerous itemsets and produces a smaller number of association rules. To reduce the number of itemsets, they could 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. Second, the high frequency of stop words and language constructs is a problem that frequent itemset mining is not equipped for handling; this doesn't happen in the domain of market basket data since the packaging of each item is not enlisted in the receipt. Even if a maximum frequency threshold is set, risking to filter out important itemsets, a lot of non-English language constructs will be mined because the proportion of posts in English is much higher than other languages. Finally, there is a lot of redundancy in frequent itemsets caused by trivial differences in the language used. In this paper we address those problems and adapt frequent itemset mining to be effective on social media text, without degrading its efficiency.

Unlike trending topics[1] [15], the results of frequent itemset mining include itemsets that have high frequency because of sustained, relatively high, interest as well as a spike of interest. The mined itemsets provide the vocabulary associated with events and can be used in various ways for search and summarization. The collection of mining results of different epochs of time can be used for temporal query and document expansion [4, 6]. The mining results of each epoch can be treated as a document, facilitating the creation of the "temporal profile"[9] of a query or a document being expanded. For summarization, frequent itemsets can provide a good foundation for summary creation. We refrain from

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

using the term summary only because we don't take into account the quality measures of summaries such as coherence and cohesion. However, we propose a ranking scheme for itemsets that makes the mining results presentable to human users. The top ranked itemsets cover a variety of open topics, and within one topic different opinions are reported as separate contrastive itemsets.

The first three sections provide the necessary background for the rest of the paper. We start by discussing related work in section 2, then we explain frequent itemset mining and the algorithm on which we build our work in sections 3 and 4 respectively. In sections 5, 6 and 7 we propose solutions to different problems faced when applying frequent itemset mining to social media text. We present the outline of those sections graphically using a frequency ordered prefix tree of the itemsets. Each path from root to a leaf in such a tree represents an itemset, where each itemset is ordered in non-increasing order of the frequency of items. Itemsets having the same prefix share the nodes representing this prefix. This representation is typical in the literature, and it is actually a very good representation of the frequent itemset mining problem. Figure 1 shows conceptually how section 5, 6 and 7 addresses different parts of the problem. In section 8, we conclude and suggest future directions.

2. RELATED WORK

Frequent itemset mining is a large body of work that goes back to the early 90s. 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 [1] and algorithms based on it suffer performance degradation and a big increase in memory requirement when the number of distinct items is high. This is caused by the candidate generation bottleneck as explained later. Another famous class of mining algorithms is the FP-Growth [8] 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 a branch the frequency is non-increasing from the root down to the leaves. The memory requirements of FP-Growth algorithm suffers from the sparsity of the data, since the data structure is succinct only if it can find common prefixes within the constraints of its invariant. A less known algorithm that is robust against data sparsity is Linear time Closed itemset Mining (LCM) [19], which we will describe in detail in section 4. We use the implementation submitted to the workshop for frequent itemset mining Implementations (FIMI) '04 [10], which is the workshop's award winner.

The problem that there are too many itemsets, with a lot of noise and redundancy, is addressed either by clustering [21] the itemsets or by mining only itemsets that satisfy a certain condition. The closure condition [16] is a prominent condition upon which many other conditions are based. Most similar to the strongly closed itemsets we propose are the δ-covered [20] and the δ-free [3] sets. The δ-covered condition is exactly the opposite of the strong closure condition, and it "relaxes the closure condition to further reduce pattern set size" [14]. The δ-free condition is like the strongly closed condition but it is used to eliminate different itemsets, since the motivation is to provide a compressed representation of itemsets by sacrificing support information.

The motivation of most of the work in finding representative itemsets is compressing itemsets mined from a general dataset. This leads to decisions different than what are made in order to find the most informative itemsets mined from text. For example, in [13] itemsets are mined from paragraphs of newswire text, and used to determine term weights for query expansion. Improvements in performance have been achieved by using itemsets known to come from a training set of related documents as well as ones from unrelated documents. In [12], similar methods of term weighting were used in pseudo-relevance feedback for Twitter search, and achieved improvements on a weak baseline. On the other hand, in [22] LCM is also used to mine Twitter posts and then a few itemsets are selected as a "summary" of the data, but they are selected according to their utility in a lossy compression algorithm. The quality of the summary seems to be affected by the choice of utility function, but the only assessment made about this was showing that the actual transactions can be reconstructed from the summary with an accuracy that is expected to be high if the Tweet contains "both recent and frequent keywords". In the experiments, non-negative matrix factorization is used to extract topics from parts of the summary matching a query (world cup) and specific time intervals (before certain matches). It is unclear whether the raw summary would cover a variety of topics, specially non-trending ones, and no ranking scheme was shown to pick interesting topics without specifying a query. Moreover, the data is preprocessed by stemming and stop words removal, which should require a language identification component upstream but it is not discussed.

3. FREQUENT ITEMSET MINING

3.1 Preliminaries

Traditionally, 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 will stick to 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* 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 also adapt the support threshold to the dynamicity of the *volume* at different times of the day. We define the *minimum support threshold* as the threshold at the hour of the least *volume* during the day. The *minimum support* is supplied as an absolute number, a, and then converted to a ratio, $\alpha = \frac{a}{avg(volume_{daily-minimum})}$. The actual support threshold used for mining any given epoch is thus α multiplied by the *epoch*'s *volume*.

We now introduce the notation used in this paper:

- $W = \{w_1, w_2, \ldots, w_n\}$: The set of all items. Can be terms or 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.

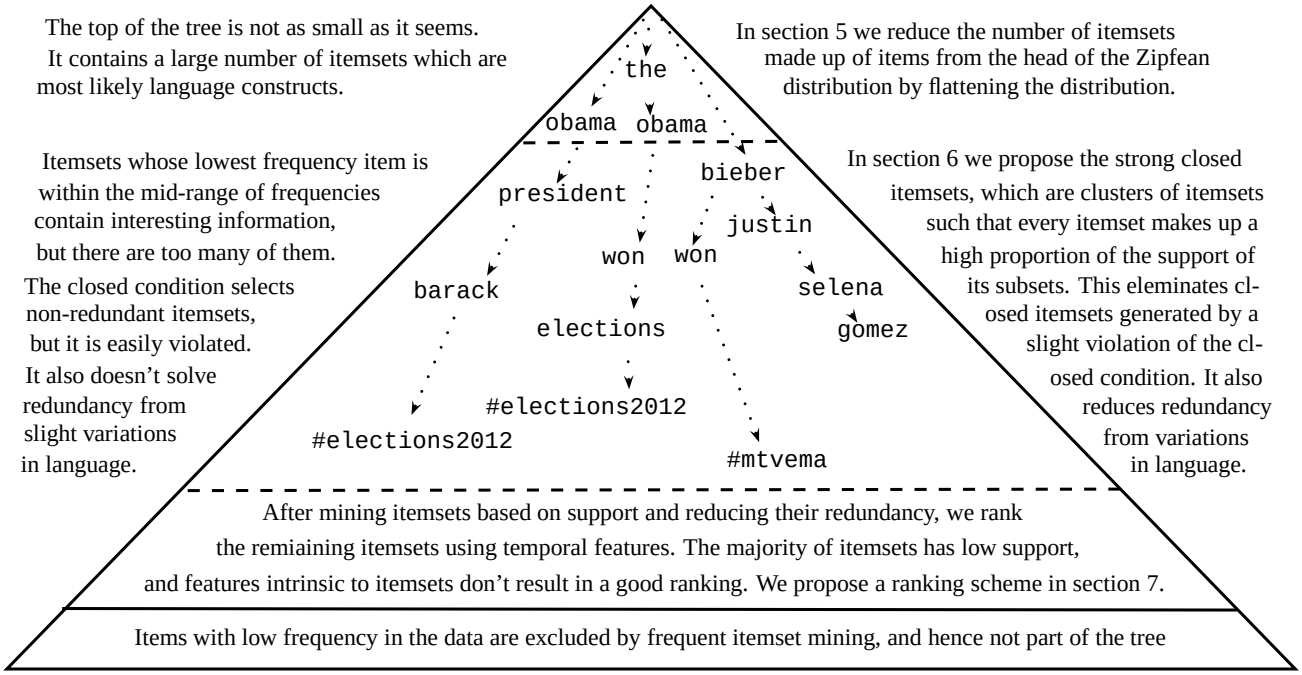


Figure 1: Roadmap of the paper overlaid on an frequency ordered prefix tree

- $E^{span} = \langle t_a, t_b, \ldots, t_v \rangle$: 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.
- $|.|$: Cardinality operator; gives the size of the operand.

3.2 Background

The two basic operations of frequent itemset mining algorithms are *Candidate Generation* and *Solution Pruning*. The original Apriori algorithm by Agrawal et al. [1] 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 frequent (K-1)-itemsets for generating candidate K-itemsets, a lot of possible K-itemsets are implicitly pruned, based on that all subsets of a frequent itemset has to be frequent (the Apriori property). This still generates a very large number of candidates, specially in early iterations of the algorithm. Consider, for example, the generation of candidate 2-itemsets from a database. This 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 algorithms are proposed to take hints from the transaction space rather than operating blindly in the item space. Some algorithms do that by traversing a data structure representing the transactions [8], and others do that by generating itemsets having a certain property that helps pruning out more candidates. In this paper we expand on LCM [19], an algorithm based on a property of a class of itemsets called *Closed Itemsets* [16]. 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 the following:

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*, and they are 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 the other subsets do not need to be extended by items that would lead to the longer closed itemset. This is called *Prefix Preserving Closure Extension (PPC-Extension)* and it is proposed and formally proved by Uno et al. [19]. *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 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 whose removal does not result in an itemset with higher 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 can be extend only by items *larger* than its *suffix head*. Extending by *smaller* items will lead to closed itemsets already generated.

2. After forming an itemset s, add to its *suffix* all items whose frequency within T_s is equal to $|T_s|$.

3. If any item in the *suffix* is *smaller* than the suffix head, 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 $T_{\{barack\}}$. The lower table shows steps of itemsets generation. The current solution along with its frequency is in column 2, solutions marked by an (*) are the closed itemsets emitted. All possible extension items and their frequencies are in column 3 with the one being considered bolded. Column 4 is a comment explaining the step. At each step, a pass is done on $T_{itemset}$ to enumerate and count possible extension items. To enforce a support threshold infrequent extension items are removed, but in this example there isn't such a threshold. Notice that the number of steps is linear in the number of closed itemsets, and the only additional storage required besides the storage of the documents is that of the 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.[19] 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 data set is shown in section 5. We proceed by describing how to implement this algorithm using an inverted index.

4.1 Implementation Details

We show in algorithm 1 how to implement LCM and PPO-Extension using an inverted index. The algorithm takes as input an epoch of data and a support threshold as a ratio α. It outputs the closed itemsets with support more than 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.

The algorithm also lends itself to distributed implementations easily. For example, a Map/Reduce implementation is easy since the only operations are counting (line 14) and projection (line 22). However, the fast execution time and the low memory requirements of the algorithm makes it possible that a distributed implementation will cause an overhead. In the implementation shown, it is not necessary that the index's tokens list follow the total ordering; all itemsets of length 1 will be considered anyway.

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

```
1  C ← {⟨∅, E⟩} ;                    // ∅ is a closed itemset
2  X ← Inverted index of E;
3  foreach w ∈ X.tokens do
4  |  T_{w} ← X.postingsList[w];
5  |  if |T_{w}| ≥ α |E| / E.span then LCM({w}, w, T_{w}) ;
6  end
7  return C;
8  Function LCM(s: Current itemset, w_sh: Suffix head,
9  T_s: Transactions (Tweets) containing s) is
10 |  frequency[1 ... w_n] ← 0;
11 |  suffix ← {w_sh};
12 |  foreach t ∈ T_s do
13 |  |  foreach w ∈ d do
14 |  |  |  frequency[w]++;
15 |  |  |  if frequency[w] = |T_s| then suffix.add(w) ;
16 |  |  end
17 |  end
18 |  if ∃v ∈ suffix : w_sh ≻ v then return;
19 |  C.add(⟨s ∪ suffix, T_s⟩);
20 |  foreach v ≻ w_sh and v ∉ suffix do
21 |  |  if frequency[v] ≥ α |E| / E.span then
22 |  |  |  D ← T_s ∩ v ; // Results of query s AND v
23 |  |  |  LCM(s ∪ suffix ∪ {v}, v, D)
24 |  |  end
25 |  end
26 end
```

Algorithm 1: LCM frequent itemsets mining

5. MINING SOCIAL MEDIA

Throughout this paper we use data we have been collecting from the Twitter public stream[2] as of October 1st, 2012. We use only Tweets written in Latin script to facilitate tokenization using white space and other word boundaries. We collected only the Tweet text to avoid reliance on any features specific to a certain social medium, and make the algorithms applicable to other media where text is short such as comments and Facebook or Google+ status updates. The only preprocessing performed was removing duplicate original Tweets (not retweets) using a Bloom filter. This removes spam Tweets sent by BotNets (such as advertisements about Raspberry Ketone diets).

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.

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

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

Documents (two per row)

Step	Current Solution	Possible Extension Items	Comments
1	{barack} (4)*	**mitt** (2), obama (3), romney (2)	Items are ordered lexicographically
2	{barack,mitt} (2)*	**obama** (1), romney (1)	Extension items reenumerated & counted
3	{barack,mitt,obama} (1)	romney (1)	Rule 2: 'romney' appears in all $D_{itemset}$
4	{barack,mitt,obama,romney}(1)*		Rule 2: 'obama' is the *suffix head*
5	{barack} (4)	mitt (2), **obama** (3), romney (2)	Nothing more to add, back to 'barack'
6	{barack, obama} (3)*	~~mitt~~ (1), **romney** (2)	Rule 1: skipping 'mitt', adding 'romney'
7	{barack, obama, romney} (2)*	~~mitt~~ (1)	Rule 1: Nothing more to add.
8	{barack} (4)	mitt (2), obama (3), **romney** (2)	Back to 'barack', adding 'romney'
9	{barack, romney} (2)	~~mitt~~ (1), obama (2)	Rule 2: add obama to suffix after 'romney'
10	{barack, romney, obama} (2)	~~mitt~~ (1)	Rule 3: suffix isn't ordered, prune solution

Closed itemsets containing 'barack'

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

Figure 2 shows the runtime of LCM on epochs of increasing length, and we will show in section 6.3 that our extensions don't degrade the performance. The times reported in figure 2 are averages across all epochs of the specified length in the months of October, November and December, using a time step that is half the epoch length. The variance is very low and the confidence bands are not shown because they appear as dots on top of the bars.

The support threshold used throughout this paper, unless otherwise specified, is $\alpha = 0.0002$. This is determined as follows: We picked a topical term that is known to steadily appear with a rather high frequency, and is talked about in all languages; 'obama'. The maximum likely hood estimate of the probability of the term 'obama' within the whole collection of Tweets is 0.0001. Since the average number of Tweets per hour is 100000, so the term 'obama' is expected to appear 10 times per hour on average. Thus, we use a minimum support threshold of 10, 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 an 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 because a lot of noise itemsets are mined.

Regardless of the length of the epoch, many itemsets mined are combinations of function words with each other and with other items. In the next section, we propose a novel method for reducing the number of itemsets and eliminating the effect of function words, using a simple but effective technique.

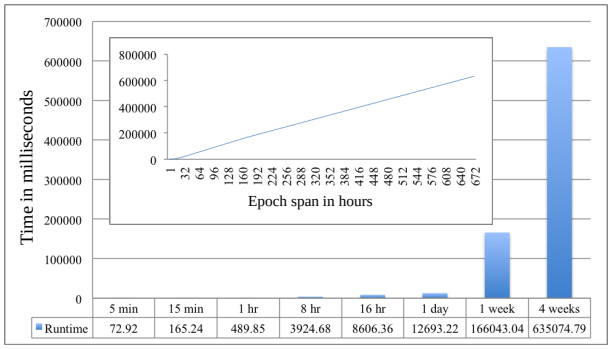


Figure 2: Mean runtime at different epoch spans

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. Actually, we 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 combination 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 and has special properties [17]. We experimented with various measures, and in fact we found out that a very good measure for identifying multiword Named Entities is Yule's Q [18]; a measure of association and disassociation derived from the odds ratio. However, we have finally found that for the purpose of preprocessing before frequent itemset mining what works best is tokenizing the documents into term N-grams with varying N.

We start by tokenizing into unigrams, and counting their frequencies. Then we use the probability of the same word used to determine the support threshold, $P('obama') = 0.0001$, and assume that this is where the head of the Zipfean distribution starts. For each unigram belonging to the head, we create two term bigrams by attaching to it the unigrams before and after it. We repeat this foreach $N \geq 1$ by creating two (N+1)-grams for all N-grams with probabilities above the threshold until there are no more such N-grams. We do not prevent overlap, because there is no guarantee that the N-Gram created makes any sense. 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, since each high frequency N-Gram is replaced by two lower frequency ones. Let the original probability threshold be η, then the adjusted η_N is:

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

Figure 3 shows the effect of increasing the maximum length of N-grams from 1 to 5 on the number of tokens, the number of closed itemsets of length more than 1, and the runtime of mining 1 hour epochs of data. The values shown are averages across all 1 hour epochs in the month of November. The value of η used is 0.0001. Figure 3(a) shows that the number of distinct items increases a lot when N moves from 1 to 2, then keeps increasing slightly until it starts decreasing at N=5. The decrease happens because all 4-grams with probability above the threshold are parts of Tweets from services that use the same text and append a URL, such as Tweets reporting scores from Game Insight[3]. Such Tweets are tokenized into more 4-grams than 5-Gram, and the 4-grams appearing in them don't appear elsewhere; thus each two of them are reduced into one 5-Gram. Figure 3(b) shows that the number of itemsets keeps decreasing as expected. Figure 3(c) shows that runtime also decreases as N goes from 1 to 5, since LCM runtime is proportional to the number of closed itemsets, and is not affected by the sparsity of data. The runtimes in this figure are slightly less from those in figure 2 because they don't include the time taken for writing the posting list of each itemset.

After mining term N-grams we flatten the itemsets to sets of unigrams again. This removes overlap between parts of itemsets making it easier to reason about how they relate to each other. This is also necessary since an itemset will have different N-Gram set representations, and its postings list is the union of those of the different representations.

6. FILTERING ITEMSETS

So far we have successfully overcome the effect of function words, using a simple technique that exploits LCM's tolerance to sparsity. The average number of itemsets mined from an hour of Twitter data dropped from 61,505 to 6,146, however there is still redundancy in the itemsets.

The closed property of an itemset is easily violated by modifying one transaction that contains the itemset and removing one of its items. While an update operation is not supported in the model of frequent itemsets mining, a similar effect happens when people are writing about a certain fine grained topic. For example, figure 4 illustrates itemsets related to Donald Trump's famous Tweets when Obama's victory was clear[4]. Each area in the figure represents the transactions containing the itemset formed by concatenating the items in all intersecting ellipses.

The figure shows the effect of the lack of context and structure in conversations happening on Twitter. Because there was originally no way to refer to a certain Tweet, a Tweet that sparked conversation on Twitter had to be quoted in a retweet along with the retweeter's comment on it. This tradition still continues even though Tweets can now reference each other. Due to the 140 characters length limit of Tweets the quotation usually has to be made as short as possible by selecting only the most discriminative words.

In the figure, the most discriminative words are "sham, and, travesty" which are quoted along with Donal Trump's user name in most of the retweets. Other people choose to also include "not, democracy" and/or "elections", and in most of the cases the retweet indicator "rt" is added. This selection is an act of collaborative filtering, but it results in many trivially different subsets from the original Tweet. The additions of retweeters also form many different supersets of the of the original Tweet, and some additions represent opinions that are supported enough to be mined as itemsets.

We propose two conditions that are not as easily violated as the closed condition for selecting itemsets. The two conditions build on the concept of *association rule confidence*. Confidence is the basic property used for association rules mining, and it is used in the definition of δ-free sets [3]. Mining itemsets based on the confidence of rules they induce has long been recognized as a method for finding "interesting patterns" [5], but since this property is not anti-monotone a variation has to be used (for example, *all confidence* [11]). The confidence of an assocation rule that the presence of an itemset, s_j, implies the presence of another itemset, s_i, is defined as:

$$conf(s_j \rightarrow s_i) = \frac{|T_{s_i} \cap T_{s_j}|}{|T_{s_j}|} \quad (3)$$

6.1 Distinct Itemsets

The *distinct* condition is a strengthening of the closed condition so that it is not violated by trivial differences. 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. This condition chooses closed itemsets which substantially violate the closed condition of their subsets. 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:

$$\mathcal{D} = \{s : s \in \mathcal{C} \, and \, \exists \; s_p \subset s \; where \; \frac{|T_s|}{|T_{s_p}|} \geq \kappa\} \quad (4)$$

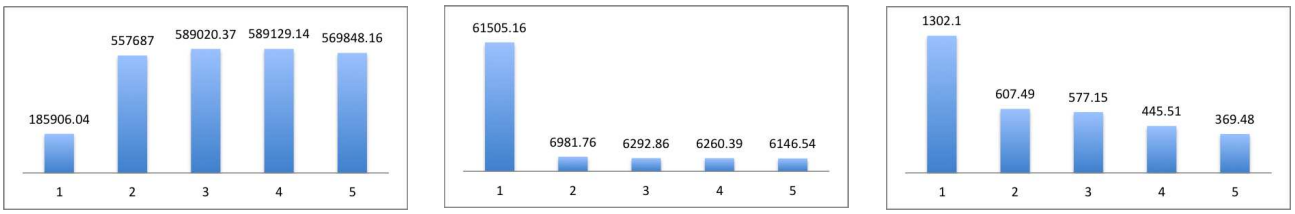
In figure 4, *distinct* itemsets are illustrated in solid lines, and closed itemsets that don't satisfy the distinctiveness condition in dashed lines. It is clear from the figure that there is still a lot of redundancy in *distinct* itemsets.

6.2 Strongly Closed Itemsets

To remove the redundancy in *distinct* itemsets we merge similar ones into *strongly closed* itemset clusters. 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 with the highest overlap ratio. When a *distinct* itemset is clustered with an itemset which 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 support is the size of

[3]http://www.game-insight.com/

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



(a) Mean number of distinct items (b) Mean number of itemsets (c) Mean runtime in milliseconds

Figure 3: Effect of the increasing maximum N-Gram length on results of mining of 1hr epochs of data

the union of their postings lists. We define the desired clustering and the strongly closed itemset represented by each cluster as follows:

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

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

The clustering scheme described selects the cluster that contains the itemset which maximizes the confidence of the rule $conf(s_j \rightarrow s_i)$, with a lower bound on the overlap to maintain distinctiveness. This clustering can be implemented efficiently using techniques similar to the ones proposed by Bayardo et al. [2]. 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 item. 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

We analyze the performance of the filtering conditions proposed by applying them to the mining results of all 1 hour long epochs in the Twitter data. The average number of itemsets mined from an hour long epoch is 2439.17 closed itemsets of length 2 or more; that is, excluding itemsets that are merely a frequent item. The number of length 2 or more maximal itemsets is 1831.92, which we provide just as a reference to the result of a basic way of filtering.

Figure 5 show the effect of varying κ on the mean number of *distinct* and *strong closed* itemsets. The number of *distinct* itemsets keeps dropping as the distinctiveness threshold increases. On the other hand, the number of *strong closed* clusters formed keeps increasing as the similarity (indistinctiveness) threshold decreases. The dashed line shows that the number of unclustered distinct itemsets reaches

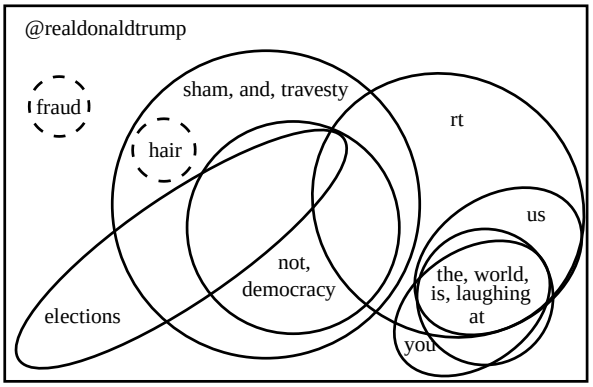


Figure 4: Closed, distinct and strongly closed sets

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

for $i \leftarrow 2$ *to* $|\mathcal{C}|$ **do**
  $C \leftarrow \{s_c : s_c \in \mathcal{C} \; and \; c < i \; and \; |s_c \cap s_i| > 0\}$;
  $P \leftarrow \{s_p : s_p \in \mathcal{C} \; and \; p < i \; and \; s_p \cap s_i = s_i\}$;
  $s_p \leftarrow \mathbf{argmax}_{s_p \in P} \frac{|T_{s_i}|}{|T_{s_p}|}$; `// Direct parent`
  if $\frac{|T_{s_i}|}{|T_{s_p}|} < \kappa$ **then**
    continue; `// Not a distinct itemset`
  $s_m \leftarrow s_i$; `// Cluster centroid, initially self`
  $maxConf \leftarrow 0$; `// Best candidate's score`
  foreach $s_c \in C$ **do**
    $\Delta \leftarrow (1 - (1 - \kappa))|T_{s_c}|$; `// Maximum difference`
    $\delta \leftarrow \text{difference}(T_{s_c}, T_{s_c \cup s_i}, \Delta)$; `// Stops early`
    if $\delta \leq \Delta$ **then**
      $conf \leftarrow \frac{|T_{s_c}| - \delta}{|T_{s_c}|}$;
      if $conf > maxConf$ **then**
        $s_m \leftarrow s_k$; `// Best merge candidate`
        $maxConf \leftarrow conf$;
      end
    end
  end
  $\mathcal{R}[s_i] \leftarrow \mathcal{R}[s_m]$; `// Cluster` s_i `with` s_m
  $\mathcal{R}[s_m].itemset \leftarrow \mathcal{R}[s_m].itemset \cup s_i \cup s_m$;
  $\mathcal{R}[s_m].postingsList \leftarrow \mathcal{R}[s_m].postingsList \cup s_i.postingsList \cup s_m.postingsList$;
end
return $\mathcal{R}$;

Algorithm 2: Forming strong closed itemset clusters

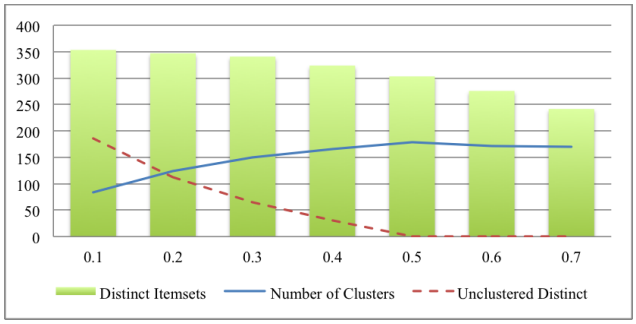


Figure 5: Effect of changing κ on mining results

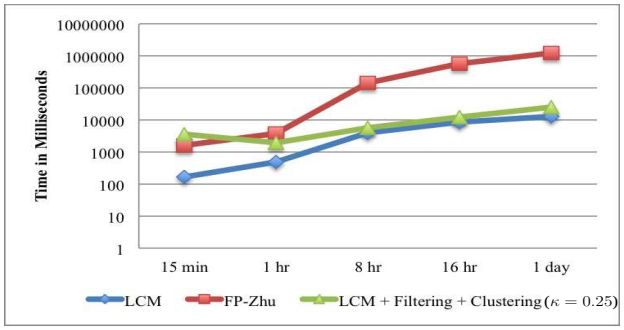


Figure 6: Runtime of itemset mining alone and with filtering and clustering at different epoch spans

zero at $\kappa = 0.5$, which explains why the number of clusters changes very slightly after that. We use $\kappa = 0.25$ for the rest of the paper, which is an arbitrary choice based on the definition not the data. The average number of itemsets (*strongly closed* and unclustered *distinct*) mined from 1 hour epochs at this value of κ is only 224.48 which is about 10% of the number of closed itemsets.

Figure 6 shows the total runtime of the LCM algorithm plus filtering based on the distinct condition and clustering into strong closed itemsets at different epoch spans. The runtime of LCM alone is also plotted for reference. We also plot the performance of another frequent itemset mining algorithm, FP-Zhu [7], which was the runner up at FIMI '04 [10]. 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 minutes epoch and then takes a constant time for epochs of longer span.

The experiments were run using a single threaded Java implementation of algorithm 2. The experiments were run on a 1.4GHz processor with 2MB of cache. Unlike the original LCM algorithm, filtering low confidence itemsets requires keeping mining resluts in memory to calculate the confidence of newly generated ones. The memory requirement is not large because the number of itemsets averages at about 6000. In our experience with the Twitter data it was enough to keep only a buffer of 1000 itemsets, and this is what we use for the runtime performance evaluation and the empirical evaluation of the filtering results in sections 7.1. If all itemsets are kept in memory the runtime increases slightly and averages at 5.2 seconds for filtering an hour epoch.

7. TEMPORAL RANKING

So far we have been reducing the number of itemsets, and we succeeded to reduce it to less than 1% of the original number. We now discuss how to rank the itemset clusters according to their novelty when compared to other time periods, so that the synopsis can be presented to users or the rank can be used for weighting the itemsets.

A good indicator of novelty is the pointwise Kullback-Leibler Divergence (KLD) between an itemset's probability in the current epoch and in a longer past epoch - the background model. The pointwise KLD of the probability of an itemset s_i in the background model Q from its probability in the current epoch P can be considered the information gain, IG:

$$\begin{aligned} IG(P(s_i), Q(s_i)) &= -(H(P(s_i)) - H(P(s_i), Q(s_i))) \\ &= \sum P(s_i) \log P(s_i) - \sum P(s_i) \log Q(s_i) \\ &= KLD(P(s_i)||Q(s_i)) \end{aligned} \quad (6)$$

To generalized this to the case of strong closed itemset clusters, we have to take the overlap between the itemsets of the cluster into account when calculating the information gain of the whole cluster. We note that the joint probability of the itemset, s_i, and its superset, s_j, is equal to the probability of the superset; $P(s_i, s_j) = P(s_j)$ and $Q(s_i, s_j) = Q(s_j)$. Thus, the information gain of the appearance of an itemset and its superset together is essentially the information gain of the superset; $IG(P(s_i, s_j), Q(s_i, s_j)) = IG(P(s_j), Q(s_j))$. Also, their Pointwise Mutual Information (PMI) is the Self Information (SI) of the subset:

$$PMI(s_i, s_j) = \log \frac{P(s_i, s_j)}{P(s_i)P(s_j)} = -\log P(s_i) = SI(s_i) \quad (7)$$

Therefore, the information gain of a superset is different from the information gain of its subset by the information gained (or lost) because of the additional items. Hence, the information gain of a strong closed itemset cluster, $S = \{s_{i1}, .., s_{im}\}$, can be approximated as the information gain of its smallest subset, s_{min}, plus the differences between the IGs of member subsets and the smallest subset. We use the squares of the differences, because the IG value can be positive or negative and we only care about the magnitude of the difference. To give the self information of the smallest subset the same influence, we also square it. Thus, the information gain of a strong closed itemset is given by:

$$\begin{aligned} IG^2(P(s_{i1}, ...s_{im}), Q(s_{i1}, ...s_{im})) = \\ I^2(s_{min}) + \\ \sum_{j=i1..im} (IG(P(s_j)||Q(s_j)) - IG(P(s_{min})||Q(s_{min})))^2 \end{aligned}$$

The formula above can be used directly for ranking clusters, but it will favour larger ones. We normalize by the size of the cluster giving our final ranking formula for strong closed itemsets:

$$\overline{IG}(S) = \frac{IG^2(P(s_{i1}, ...s_{im}), Q(s_{i1}, ...s_{im}))}{m} \quad (8)$$

7.1 Empirical Evaluation

We now show examples of the performance the proposed methods in creating a synopses of the 2012 elections day, November 6^{th}, and another less eventful day, November 9^{th}. The background model we use for each day is the results of mining the 4 weeks before it, using a *minimum support* value of 1 occurrence . Mining the background model at such a low support increases the number of produced itemsets, which is desirable for a background model. All probabilities are smoothed by add-one smoothing.

Tables 2 and 3 show the top 3 itemsets for 1 hour epochs in the days examined. 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. The third column is a commentary to explain the itemsets, but we omit it from table 3 to save space since the itemsets are self explanatory.

In table 2, we can see how the events of the US presidential elections unwind from "get out and vote" to the projections and debates, all the way to the "acceptance speech". Early in the day, itemsets about UEFA Champions football matches and a TV show "geordie shore" appear in the top 3 along with itemsets about the still uneventful elections. Actually, the matches keep occupying top positions and timely updates of their scores appear in the top 30 itemsets, until they end and the elections heats up. Short after the results of the elections became clear, news that "weed is legal in Colorado" occupies the top position. This exemplifies the power of social media as a collaborative filter, picking the important news even if it is unexpected. The user centric definition of importance is eminent in the great attention given to the "lady behind Obama with a flag in her hair" during the acceptance speech.

On November 9^{th}, table 3, the most interesting hour is 15:00. The MTV Europe Music Awards (MTVEMA) was taking place and votes were solicited from audience through Twitter. This is an example of a topic where people have different opinions. The top 3 itemsets of the hour 15:00 are supporting "Katy Perry", "Justin Bieber" and "Lady Gaga" respectively. They are all reported as separate itemsets, showing how clustering using the postings lists avoid forming incohesive clusters. No other major events were happening but many overlapping minor ones happened. The day started by news about the end of two careers; the Laker's "coach Mike Brown" got fired and "CIA director David Petraeus resigns". The relationship of Justin Bieber also ends as he "broke, up, with, Selena, Gomez" at 22:00. This overlaps with his participation in the MTVEMA, and both topics occupied high rankings. By the end of the day many congratulations for the Indonesian Hero's day ("Hari Pahlawan") appear, and the Turkish commemoration day of Ataturk is also mentioned as the 10th of November starts in these countries. These are examples of itemsets from languages with a relatively low number of users, showing how the absolute popularity of a topic doesn't affect its rank. If itemsets from only a specific language is desired, language identification can be applied on the itemsets and Tweets from their postings lists. Moving language identification downstream avoids affecting the results of mining because of error in an upstream component.

8. CONCLUSION AND FUTURE WORK

13:00	0, 1, de, jong	De Jong scores for Ajax
	geordie, shore	Season 5 of the TV series starts
	get, out, and, vote	Still early in the U.S. elections day
17:00	if, obama, wins	Speculations regarding elections
	USERNAME, spots, my, club	Pyramidal marketing scam. Retweeted by people to make money.
	the, polls, close	Polls to start closing at 6 PM
19:00	A partir de que idade você considera alguém velho?	Internet meme from Brazil, discussing when to start considering a person old.
	food, stamps	Discussions pertaining to elections
	linda, mcmahon, senate	Linda McMahon loses CT senate race
20:00	obama, got, this	Announcing states that Obama got
	projected, winner	Early projections about who will win
	moving, to, canada	Reaction to projections
21:00	elizabeth, warren	MA senate elections winner
	popular, vote	Comparing Popular vs Electoral votes
	who, is, the, president?	Anticipation for the elections results
22:00	#forward, #obama2012	Obama won, time to move #forward
	my, president, is, still	Some were saying Black (skin colour), others Blue (party colour)
	back, in, office	Obama is back in office
22:30	once, you, go, black	A popular cultural reference.
	@realdonaldtrump, this, elections, ... [his famous Tweet]	Donald Trump is a Republican and he couldn't accept Obama's victory
	concession, speech, write	The losing candidate has to concede before the winner declares victory
23:00	weed, is, legal, in, colorado	This piece of news got the attention as soon as it was announced
	karl, rove	Fox news challenges results from Ohio
	our, country, is, trouble	Dramatic reactions to elections
23:30	acceptance, speech, wrote	Obama wrote his speech, ...
	give, his, speech	... and is going to deliver it ...
	on, cnn	... on CNN.
00:30	behind, obama, with, flag, in, her [hair]	During the speech, attention on social media goes to a woman appearing behind Obama on TV!
	the, best, is, yet, to, come	Quote from Obama's speech
	flag, that, weaves	Play on words about the "flag in hair"

Table 2: Top 3 itemsets for hours in November 6th

12:00	#iwillneverunderstand, why
	breaking, news, head, coach, mike, brown, have, fired
	USERNAME, spots, available, my, club
14:00	cia, director, david, petraeus, resigns
	você, acha, que
	#tvoh [the voice of Holland], babette
15:00	#emawinkaty, i, think, katy, perry, will, be, the, big, #mtvema, winner, tweet, your, pick, at, URL
	#emawinbieber, i, think, justin, bieber, ... [same as above]
	#emawingaga, ... [same as above]
22:00	justin, bieber, and, selena, gomez, broke, up
	selamat, hari, pahlawan
	aniyoruz, kemal, mustafa [ataturk]

Table 3: Top 3 itemsets for hours in November 9th

We have proposed a method for efficiently creating temporal synposes of social media streams, based on a frequent itemset mining algorithm that is suitable for sparse data, LCM. Our method summarizes an hour long of Twitter data (100,000 Tweets on average) into 224 itemsets in 1945.68 milliseconds on average, and scales well for longer epochs of data. The direct application of LCM on 1 hour long epochs of Twitter data results in 61505.16 closed itemsets and takes 2506.58 milliseconds on average. The improvement is due to using variable length N-grams to mitigate the effect of the skewness of the frequency distribution of unigrams, and strengthening the closure condition such that it selects only itemsets that are distinctively different from its subsets and other itemsets. The distinctiveness between two itemsets is judged based on a parameter κ, which can control the selectivity of the distinctiveness condition.

We also propose a method for ranking itemsets based on their temporal novelty. The top 3 itemsets from hours in the elections day and another less eventful day show that the synopses capture important events, and can actually be directly presented to users. A possible future direction is to use itemsets that appear as a sequence for building extractive coherent summaries of the social media stream at different times.

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. ACKNOWLEDGEMENTS

This paper was funded in part by the Google Focus Award. Right?? What else should we say here? I am writing to fill some space and reserve it for the actual acknowledgements.

10. REFERENCES

[1] R. Agrawal, R. Srikant, et al. Fast algorithms for mining association rules. In *Proc. 20th Int. Conf. Very Large Data Bases, VLDB*, volume 1215, pages 487–499, 1994.

[2] R. J. Bayardo, Y. Ma, and R. Srikant. Scaling up all pairs similarity search. In *Proceedings of the 16th international conference on World Wide Web*, pages 131–140. Citeseer, 2007.

[3] 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.

[4] J. Choi and W. B. Croft. Temporal models for microblogs. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 2491–2494. ACM, 2012.

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

[6] M. Efron, P. Organisciak, and K. Fenlon. Improving retrieval of short texts through document expansion. In *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval*, pages 911–920. ACM, 2012.

[7] G. Grahne and J. Zhu. Reducing the main memory consumptions of fpmax* and fpclose. In *Proc. Workshop Frequent Item Set Mining Implementations (FIMI 2004, Brighton, UK), Aachen, Germany*. Citeseer, 2004.

[8] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *ACM SIGMOD Record*, volume 29, pages 1–12. ACM, 2000.

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

[10] R. J. B. Jr., B. Goethals, and M. J. Zaki, editors. *FIMI '04, Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations, Brighton, UK, November 1, 2004*, volume 126 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2004.

[11] W.-Y. Kim, Y.-K. Lee, and J. Han. Ccmine: Efficient mining of confidence-closed correlated patterns. In *Advances in Knowledge Discovery and Data Mining*, pages 569–579. Springer, 2004.

[12] C. H. Lau, Y. Li, and D. Tjondronegoro. Microblog retrieval using topical features and query expansion. In *Proceedings of the 20th TREC Conference*, Text Retrieval Evaluation Conference (TREC), Gaithersburg, MD, USA, 2011.

[13] Y. Li, A. Algarni, and N. Zhong. Mining positive and negative patterns for relevance feature discovery. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 753–762. ACM, 2010.

[14] G. Liu, H. Zhang, and L. Wong. Finding minimum representative pattern sets. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 51–59. ACM, 2012.

[15] M. Mathioudakis and N. Koudas. Twittermonitor: trend detection over the twitter stream. In *Proceedings of the 2010 international conference on Management of data*, pages 1155–1158. ACM, 2010.

[16] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Discovering frequent closed itemsets for association rules. In *Database Theory—ICDT'99*, pages 398–416. Springer, 1999.

[17] C. Ramisch, V. De Araujo, and A. Villavicencio. A broad evaluation of techniques for automatic acquisition of multiword expressions. In *Proceedings of ACL 2012 Student Research Workshop*, pages 1–6. Association for Computational Linguistics, 2012.

[18] P.-N. Tan, V. Kumar, and J. Srivastava. Selecting the right interestingness measure for association patterns. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 32–41. ACM, 2002.

[19] T. Uno, M. Kiyomi, and H. Arimura. Lcm ver. 2: Efficient mining algorithms for frequent/closed/maximal itemsets. In *Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations (FIMI 04)*, 2004.

[20] D. Xin, J. Han, X. Yan, and H. Cheng. Mining compressed frequent-pattern sets. In *Proceedings of the 31st international conference on Very large data*

bases, pages 709–720. VLDB Endowment, 2005.

[21] X. Yan, H. Cheng, J. Han, and D. Xin. Summarizing itemset patterns: a profile-based approach. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 314–323. ACM, 2005.

[22] X. Yang, A. Ghoting, Y. Ruan, and S. Parthasarathy. A framework for summarizing and analyzing twitter feeds. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 370–378. ACM, 2012.