Strong Closure and Novelty Ranking for Efficiently Mining a Temporal Synopsis of Social Media

ABSTRACT

Abstract

1. INTRODUCTION

The nature of text in social media poses a challenge when applying traditional search and text mining algorithms. Text in social media is usually short, lacks context and structure, and is created at a very high rate. The sheer volume of social media streams mandates the use of efficient algorithms, and the short length of individual documents makes it possible to apply Frequent Itemsets Mining. This family of algorithms is very fast and efficient, however it is not readily suited for application on text. The problem with Frequent Itemsets Mining is the large number of itemsets it produces. Actually it 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. The number of itemsets produced can be reduced by setting a higher threshold on the minimum frequency of an itemset. However, in text this threshold has to be kept low because frequencies follow a long tailed Zipfean distribution. Also, the head of the distribution is made up mostly of stop words; a situation that 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. In this paper we propose methods for adapting Frequent Itemset Mining to be effective on social media text, without degrading its efficiency.

Unlike trending topics[1] [10], 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. For example, the collection of mining results of different epochs of time can be used for temporal query expansion [3]. The mining results of each epoch can be treated as a document, so that the probability of an epoch conditioned on a query can be estimated as the probability of this document on the query using any pseudo-revelance feedback method. Further, terms from individual itemsets relevant to a query (using any relevance ranking) can be used for query expansion, thus acting as precomputed results of a simple version of pseudo-revelance feedback. We also propose a quality ranking scheme that makes the mining results presentable to human users. We don't call this a summary because we don't take into account the quality measures of summaries such as coherence and cohesion. However, the top ranked itemsets cover a variety of open topics, and within one topic different opinions are reported as separate contrastive itemsets.

The rest of the paper is organized as follows: We start by an overview of properties of text in social media and describe the dataset we use in section **??**. Then we explain the Frequent Itemset Mining algorithm which we build upon in section 3. In the next 3 sections we describe the adaptations we propose for making the algorithm suitable for mining social media text. In section 5.1 we describe how using term N-Grams of variable lengths filters out many language constructs. In section **??** we describe the use of differential temporal features to rank interesting itemsets. In section 6 we propose the strong closed property and show how to reduce the number of itemsets without loosing important ones. After that we show an example of the results of applying our methods in section **??**. In section 8 we discuss related work that applied Frequent Itemsets Mining on text. Finally, we finish by the conclusion and future work in section 9.

2. RELATED WORK

Frequent Itemset Mining is a very 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 Itemsets 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 [5] 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

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

CIKM '13

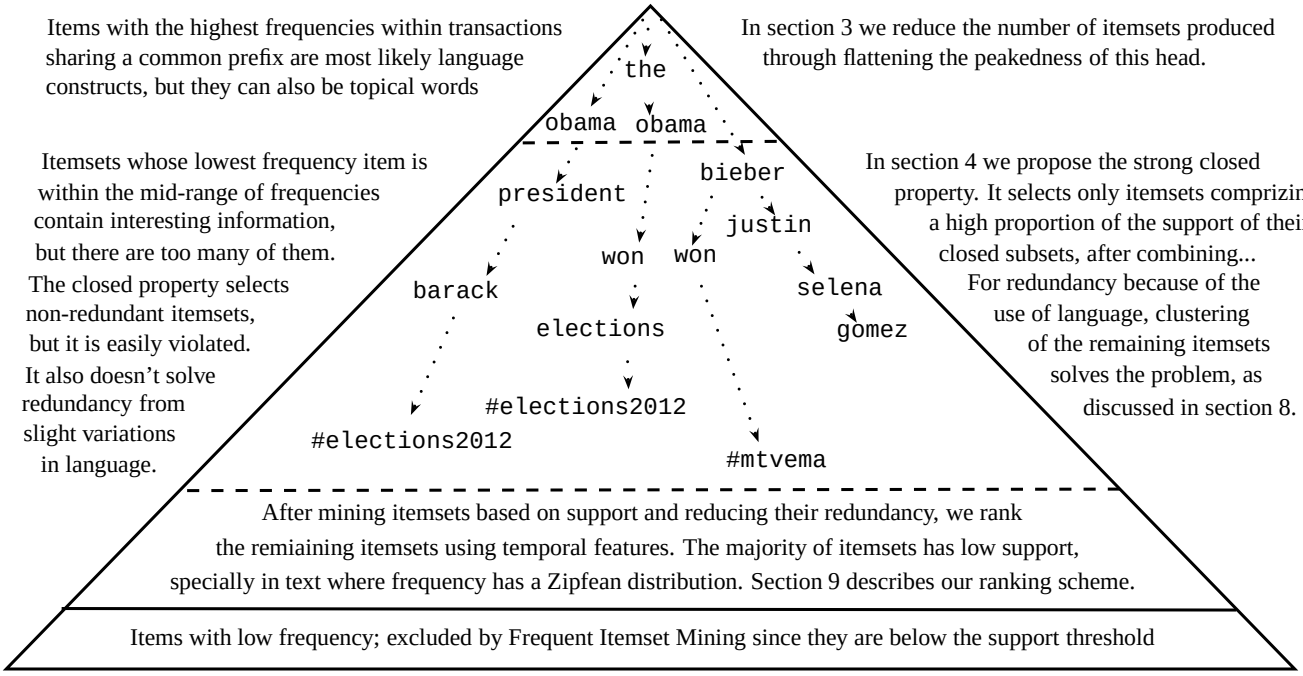


Figure 1: Roadmap of the paper overlaid on an frequency ordered prefix tree

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 LCM [12], which we will describe in detail in section **??**. We use the implementation submitted to the workshop for Frequent Itemset Mining Implementations (FIMI) '04 [6], which is the workshop's award winner. In section **??** we show that our extensions do not degrade the performance of LCM by comparing the extended LCM to the runner up at FIMI '04 [4].

The problem that there are too many itemsets, with a lot of noise and redundancy, is addressed either by clustering [14] the itemsets or by mining only itemsets that satisfy a certain condition. The closure condition [11] is a prominent condition upon which many other conditions are based. Most similar to the strongly closed itemsets we propose are the δ-covered [13] and the δ-free [2] 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" [9]. 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 [8] 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 [7], similar methods of term weighting were used in pseudo-relevance feedback for Twitter search, and achieved improvements on a weak baseline. **should I include the afore mentioned attempt?** On the other hand, in [15] 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:

- $I = \{i_1, i_2, .., i_n\}$: The set of all items (vocabulary).
- $t_a = \{i_{a1}, i_{a2}, ..\}$: A transaction made up of as a set of items, not necessarily terms. Each transaction has a sequential id denoted by the subscript letter.
- $E^{span} = \langle t_a, t_b, ..\rangle$: An epoch of data of a certain span, such as an hour, made up of a sequence of transactions.
- $s \subset I$: An itemset, its support is given by $|T_s|$.
- $T_s = \{t : s \subseteq t\}$: All transactions containing itemset s.
- $|.|$: 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 fequent *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, each based on a certain property that helps pruning out more candidates. In this paper we expand on LCM [**?**], an algorithm based on a property of a certain class of itemsets called *Closed Itemsets*. A formal definition of closed itemsets is given in equation 1:

$$\mathcal{C} = \{S_c : \nexists\, S_d \, where \, S_c \subset S_d \, and \, \|D_{S_c}\| = \|D_{S_d}\|\} \quad (1)$$

The properties of Closed Itemsets are the following:

1. An itemset is closed if adding any item to it will reduce 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 in [**?**]. This 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 D_S is equal to $\|D_S\|$.
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 D_{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 $D_{itemset}$ 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 [**?**] 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 $\langle items, \|D_{itemset}\| \rangle$. 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 i ∈ X.tokens do
4  |  D_{i} ← X.postingsList[i];
5  |  if ‖D_{i}‖ ≥ α ‖E‖/E.span then LCM({i}, i, D_{i}) ;
6  end
7  return C;
8  Function LCM(S: Current itemset, i_sh: Suffix head,
9  D_S: Documents (transactions) containing S) is
10 |  frequency[1 ... i_n] ← 0;
11 |  suffix ← {i_sh};
12 |  foreach d ∈ D_S do
13 |  |  foreach i ∈ d do
14 |  |  |  frequency[i]++;
15 |  |  |  if frequency[i] = ‖D_S‖ then suffix.add(i) ;
16 |  |  end
17 |  end
18 |  if ∃j : i_sh ≻ suffix[j] then return;
19 |  C.add(⟨S ∪ suffix, D_S⟩);
20 |  foreach i ≻ i_sh and i ∉ suffix do
21 |  |  if frequency[i] ≥ α ‖E‖/E.span then
22 |  |  |  D ← D_S ∩ i ; // Results of query S AND i
23 |  |  |  LCM(S ∪ suffix ∪ {i}, i, 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. Figure 2 shows the runtime of LCM on epochs of increasing length, and we will show in section **??** 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 that 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.

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 [**?**]. We experi-

[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

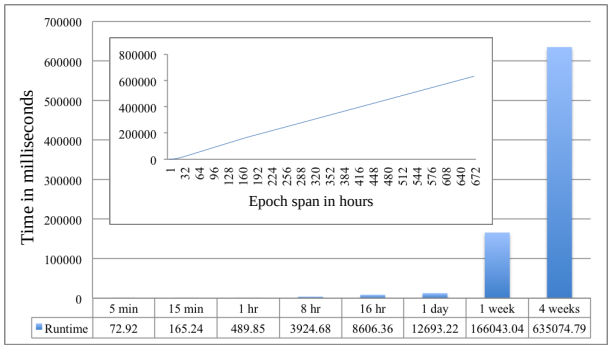


Figure 2: Mean runtime at different epoch spans

mented with various measures, and in fact we found out that a very good measure for identifying multiword Named Entities is Yule's Q; a measure of association and disassociation derived from the odds ratio. However, we have finally found that for the purpose of preprocessing before Frequent Itemsets 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, $P(\text{'}obama\text{'}) = 0.0001$, 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 ≥ 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:

$$\eta_N = \eta * \frac{\sum_{\{i:\, i \in I \text{ and } i.length \leq N\}} freq(i)}{\sum_{\{i:\, i \in I \text{ and } i.length = 1\}} freq(i)} \qquad (2)$$

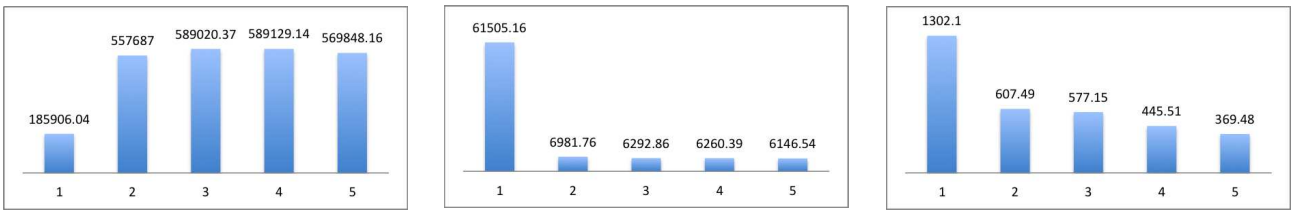
Figure 3 shows the effect of increasing the maximum length of N-Grams from 1 to 5 on the number of tokens, the number of closed itemsets of length more than 1, and the runtime of mining 1 hour epochs of data. The values shown are averages across all 1 hour epochs in the month of November. The value of 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 lists.

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. STRONGLY CLOSED ITEMSETS

The closed property of an itemset is very 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, on November 6th, 2012 many people where tweeting that "(Barack Obama) was elected (as president of the United States of America)". If all the Tweets which use the verb phrase "was elected" to report

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



(a) Mean number of distinct items

(b) Mean number of itemsets

(c) Mean runtime in milliseconds

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

the event also contain the two names (in parenthesis) in full, then there will be one closed itemset with all 12 items. However, either or both the names can be shortened by omitting an underlined part and still convey the meaning. We can consider Tweets with any combination of shortened names to be modifications of the *maximal* closed itemset. Therefore instead of 1 maximal itemset about the event there will be 8 closed ones. Now consider that it is possible to say that "Obama was elected" as well as "Obama got elected", resulting in 2 maximal itemsets because of a slight variation in the language. Thus, maximal itemsets also has redundancy. We therefore need a condition that is not as easily violated as the closed and maximal conditions for selecting itemsets.

We define a *high confidence* itemset as a closed itemset whose frequency comprises more than a certain proportion of the frequency of its least frequent subset. This property chooses closed itemsets which violate the closed property of their subsets by a significant amount. We state this condition formally as follows. Let κ be a parameter that can vary between 0 and 1 to increase the selectivity of the condition. Then the set of *high confidence* itemsets is:

$$\mathcal{K} = \{s : \frac{|T_s|}{|T_{s_p}|} \geq \kappa \; where \; s_p \subset s \; and \; |T_{s_p}| < |T_{s_a}| \, \forall \, s_a \subset s\}$$

Notice that $\frac{|T_s|}{|T_{s_p}|}$ is the called confidence of the rule "$s_p \rightarrow s$." This property is the basic property used for association rules mining, and it is used in the definition of δ-free sets. Mining itemsets based on the confidence of rules they induce has long been recognized as a method for finding "interesting patterns" [**?**], but since this property is not anti-monotone a variation has to be used (for example, *all confidence* [**?**]).

Even though choosing *high confidence* itemsets filters out many redundant itemsets formed because of slight variations of the same topic, it is still does not completely solve the problem of redundancy. The intersection of the sets of transactions containing two *high confidence* itemsets can be different by only 1 transction from either of the sets. For example, the closed itemset {Obama, Romney} can have three closed supersets: {Obama, Romney, debate, #elections2012}, {Obama, Romney, debate} and {Obama, Romney, #elections2012}. Figure 6 illustrates how all supersets can be *high confidence*, while they are still redundant itemsets mined from almost the same Tweets. Each circle in the figure represents the set of Tweets containing the itemset formed by concatenating the words in the circle with the words in its container. The figure also shows, as dashed circles, examples of itemsets that would be filtered out by the*high confidence* property.

To remove such redundancy, we merge similar itemsets into *strongly closed* itemset clusters. The *strongly closed*

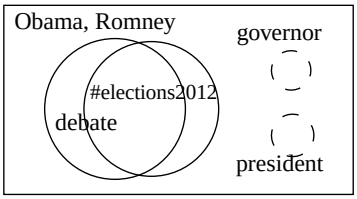


Figure 4: High and low confidence itemsets

itemset is a bag of items formed by taking the union of all cluster members. This concentrates the numerous itemsets about a certain topic into a few units. We do the clustering in the postings list space to avoid merging itemsets including different aspects or opinions about a topic. The clustering scheme we use is similar to [**?**], but it is much simpler. First, we do an all pairs similarity search on the itemsets, using IDF weighted cosine similarity. Similar itemsets are then clustered such that *high confidence* is achieved between cluster members. The notion of *confidence* still applies even though clusters include itemsets that are not supersets of one another, but its accurate definition has to be used:

$$c(s_i \rightarrow s_j) = \frac{|T_{s_i} \cap T_{s_j}|}{|T_{s_i}|} \tag{3}$$

To see if the *high confidence* condition will be violated, it is enough to check if the difference between the posting lists of two itemsets is above a certain threshold. The maximum number of different documents allowed for two itemsets being merged, so that they satisfy the *high confidence* condition, is given by equation 4. In case of merging an itemset with its superset, this difference can be calculated directly from the sizes of the posting lists. Otherwise, the difference can be efficiently calculated from the postings lists since they are sorted, and in fact it is enough to check if the difference exceeds the maximum number allowed. Equation 5 gives a formal definition of *strong closed* itemset cluster.

$$\Delta(S_1, S_2, \kappa) = (1 - \kappa) * max(|T_{s_1}|, |T_{s_2}|) \tag{4}$$

$$S = \{i : i \in \bigcup s \; where \; \forall_{j,k} \, c(s_j \rightarrow s_k) \leq \kappa\}$$
$$\mathcal{S} = \{S : s_j \; where \; \mathbf{argmax}_{j,k} c(s_j \rightarrow s_k) \; \forall \; s_j \in \mathcal{C}, s_k \in S\} \tag{5}$$

After joining an alliance an itemset seizes to exist outside of the alliance, so that an itemset can be part of only one alliance. However, when checking which itemsets to consider for merging with a still unallied itemset, its similarity is calculated with the individual itemsets rather than the

alliance. Otherwise, an itemset could fail to join an existing alliance because the similarity between the larger bag of items and the itemset is likely to be lower than the similarity between individual member itemsets and the itemset. This can cause cascading many alliances that should have been separate into one large alliance. Such an oversized alliance could also be about different topics. Empirically, we observe that one and only one oversized alliance about different topics is formed. This alliance catches many itemsets made up of low IDF terms, thus not interesting. Topics with high IDF terms in them cannot have high cosine similarity with topics of only low IDF terms. The size of the bag of items in this alliance is significantly larger than other alliances, making it easy to distinguish and discard it.

Algorithm 2 shows the described algorithm for merging itemsets alliances. Table **??** shows the number of closed itemsets of length at least 2 without filtering any of them out, as well as after applying the KLD filter and the strongly closed filter, and the number of itemsets alliances. The table also shows the average quality of the itemsets after each stage of reduction. The quality is calculated as Basic Elements? PERPLEXITY? CASCADE MEASURE?

Input: θ: Minimum cosine similarity,
κ: Minimum closed strength
Data: S: Frequent Itemsets produced by LCM
Result: A: Frequent Itemsets alliances

for $i \leftarrow 2$ *to* $\|S\|$ **do**
  $C \leftarrow \emptyset$; `// Candidates for merging with` S_i
  $T \leftarrow \emptyset$; `// Subsets of current itemset`
  for $j \leftarrow 1$ *to* $i-1$ **do**
    if $\|S_i \cap S_j\| = min(\|S_i\|, \|S_j\|)$ **then**
      C.add(S_j);
      if $\|S_i\| > \|S_j\|$ **then** T.add(S_j);
    else if $cos(S_i, S_j) \geq \theta$ **then**
      C.add(S_j);
    end
  end
  $M.score \leftarrow \infty$; `// Best merge candidate's score`
  foreach $S_c \in C$ **do**
    if $S_c \in T$ **then**
      $\delta \leftarrow \|D_{S_c}\| - \|D_{S_i}\|$;
    else
      $\delta \leftarrow$ difference(D_{S_i}, D_{S_c}) ; `// Stops early`
    end
    if $\delta \leq \Delta(S_i, S_c, \kappa)$ *and* $\delta < M.score$ **then**
      $M \leftarrow S_c$;
      $M.score \leftarrow \delta$;
    end
  end
  if $M.score < \infty$ **then**
    A[M].$itemset \leftarrow$A[M].$itemset \cup S_i \cup M$;
  end
end
return A;

Algorithm 2: Merging itemsets into alliances

6.1 Bounding runtime and memory

Expanding on the previous observation that adding a high IDF term to an itemset with low IDF terms only prevents achieving a high enough cosine similarity, we introduce an optimization that improves both runtime, memory requirements and filtering power. Unlike the original LCM algorithm, our extension requires keeping previous itemsets in memory so that newly generated ones are compared to them to find candidates for alliance. Instead of keeping all previous itemsets in memory we keep only a limited number, b. This obviously improves runtime and memory requirements, and it can also improve the quality of itemsets chosen for alliance. If the total ordering follows the descending order of items' frequencies, then, because of the way PPC-Extension produces itemsets, for an itemset S_x and any candidate subset $S_{(x-b)}$ that was produced b itemsets earlier $\|D_{S_{(x-b)}}\| - \|D_{S_x}\| \geq b$. This lower bound is achieved when the intersection of the documents containing the current itemset S_x and all the $b-1$ supersets before it is exactly the subset $S_{(x-b)}$, and there are no more itemsets with the same intersection. In this case each of the b supersets must have support at most $\|D_{S_{(x-b)}}\| - 1$ to be considered closed, but because there are b of them then actually their support is $\|D_{S_{(x-b)}}\| - b$. Since the minimum support difference is b, then the maximum confidence of S_x is $\frac{\|D_{S_{(x-b)}}\|-b}{\|D_{S_{(x-b)}}\|}$. Therefore for a given κ we can determine the buffer size b as the frequency of itemsets keeps decreasing. In our implementation we use a fixed b of 1000.

6.2 Ranking alliances

So far we have been reducing the number of itemsets, and we succeeded to reduce it to about 1% of the original number (in alliances). We now discuss how to rank the itemset alliances, making use of the structure of the alliance. Again, we exploit the pointwise KL Divergence from a 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:

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

We note that the joint probability of the itemset and its superset is equal to the probability of the superset; $P(S_i, S_j) = P(S_j)$. Thus, the information gain of the appearance of an itemset and its superset is essentially the information gain of the superset. Also, their Pointwise Mutual Information is the Self Information of the subset:

$$PMI(S_i, S_j) = \log \frac{P(S_i, S_j)}{P(S_i)P(S_j)} = -\log P(S_i) = I(S_i) \quad (7)$$

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. Thus, the information gain of the appearance of all itemsets in an alliance of m itemets, $S_{i1}, S_{i2}, ...S_{im}$, can be calculated as the information gain of its smallest subset, S_{min}, plus the differences between the KLDs of member subsets and the smallest subset. We use the squares of the differences, because the pointwise KLD value might be positive or negative, depending on whether the subset appears with lower or higher probability than in the background. Thus the information

gain of an alliance is given by:

$$IG(P(S_{i1},...S_{im}),Q(S_{i1},...S_{im})) = I(S_{min}) + \sum_{j=i1..im} (KLD(P(S_j)||Q(S_j)) - KLD(P(S_{min})||Q(S_{min})))^2$$

Finally, the average information gain of an itemset in an alliance is give by:

$$\overline{IG}_{alliance} = \frac{IG(P(S_{i1},...S_{im}),Q(S_{i1},...S_{im}))}{m} \quad (8)$$

The ranking of itemset alliances according to equation 8 gives superior results to many other ranking schemes we tried, including ones based on scores derived from the itemset content directly. An intuitive explanation is that this ranking scheme gives high scores to itemsets alliances made up of a minimum set with small KLD and supersets with large KLD. According to the structure of the alliance produced by PPC-Extension if the total ordering follows the descending order of items' frequencies, the small KLD minimum subset will be an entity with sustained interest, and the high KLD supersets will be new appearing with probabilities very different from their background probabilities. In the next section we show the results of applying this ranking scheme along with the methods described in previous sections.

7. EMPIRICAL EVALUATION

We have shown in the last 3 sections how to improve the efficiency of Frequent Itemsets Mining on social media data, by reducing the number of itemsets produced. To show the performance of the proposed methods in choosing important itemsets, we apply them on Tweets posted on the 6^{th} and 9^{th} of November 2012. In tables 2 and 3 we show the results of mining overlapping 1 hour epochs. Because of space limitation we show only the top 3 itemsets, ranked as described in section 6.2, for each hour. Times are in the EST zone.

On November 6^{th}, the US presidential elections took place and we can see how its events unwind from "get out and vote" to the projection of "obama to win" and the votes counts (only "163, 172" made it to the top 3 at 20:30), all the way to the "acceptance speech" and the social media attention given to the "lady behind obama with a flag in her hair". Early in the day, the "goal [de] Pepe" scored for "Real Madrid" and other itemsets about UEFA Champions football games appear in the top 3. At 22:30 the news that "weed is legal in Colorado" breaks into the top 3 because of its novelty, even though this is short after Obama's victory was declared. Throughout the day itemsets about the elections as well as about other topics are ranked high, even if they are topics in a language other than English and thus with a smaller supporting user base (such as the topic in Portuguese about when to consider a person old [velho]).

On November 9^{th}, no major events were happening but many overlapping minor ones happened. The day starts 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" at 22:00 (the name of Bieber is included in other itemsets not in the top 3). The MTV Europe Music Awards (MTVEMA) was also taking place and votes were solicited from audience through Twitter. This is an example of a topic where people have different opinions, as the hour

12:00	and, get, out, vote - @laliminati, lali - geordie, shore
12:30	0, 1, de, jong - geordie, shore - if, obama, wins
13:00	de, gol, pepe - hala, madrid - geordie, shore
13:30	de, gol, pepe - geordie, shore - for, i, voted
14:00	for, i, voted - for, obama, vote - fuera, juego
14:30	and, basketball, love - for, i, voted - el, madrid, real
15:00	#countkun, 11, 6 - for, i, voted - eu, te, vivo
15:30	#geordieshore, @charlottegshore - @sophiaabrahao, URL, live, on - academy, tool
16:00	if, romney, wins - geordie, shore - if, obama, wins
16:30	geordie,shore - if,obama,wins - @noemiking20,club,spots
17:00	a, alguém, considera, de, idade, partir, que, velho, você - for, i, voted - if, obama, wins
17:30	virginia, west - election, is, the, this - food, stamps
18:00	a, alguém, considera, de, idade, partir, que, velho, você - election, is, this - virginia, west
18:30	food, stamps - a, alguém, considera, de, idade, partir, que, velho, você - linda, mcmahon, senate
19:00	#stayinline, in, line - got, obama, this - a, alguém, considera, de, idade, partir, que, velho, você
19:30	got, obama, this - projected, winner - canada, move, moving, to
20:00	obama,to,win - election,the,watching - projected, winner
20:30	elizabeth, warren - popular, vote - 163, 172
21:00	#forward, #obama2012 - election, is, this - is, my, president, still
21:30	#forward,#obama2012 - of,president,the - back,in,office
22:00	#forward, #obama2012 - colorado, in, legalized - black, go, never, once, you
22:30	colorado,in,legal - food,stamps - colorado,in,is,legal,weed
23:00	acceptance, speech, wrote - in, is, legal, weed - cnn, on
23:30	come, to, yet - est, mais - colorado, move, to
00:00	come, to, yet - behind, flag, hair, her, in, obama - flag, that, weave

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

15:00 shows. The top 3 itemsets of this hour are supporting "Katy Perry", "Lady Gaga" and "Justin Bieber" respectively, and they are all reported as separate itemsets. The neutral itemset only soliciting votes is also mined and appears in the top 3 of other hours. 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 starts in these countries.

8. RELATED WORK

9. CONCLUSION AND FUTURE WORK

10. ACKNOWLEDGEMENT

11. 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] 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.

[3] J. Choi and W. B. Croft. Temporal models for microblogs. In *Proceedings of the 21st ACM*

12:00	#iwillneverunderstand, why - brian, shaw - breaking, brown, coach, head, mike
12:30	@boyquotations, do, everyone, follow, followers, gain, more, rt, s, want, who, you *(written as '@boyquotations,..' later)* - brian, shaw - jackson, jerry, phil, sloan
13:00	'@boyquotations,..' - ao, lado, sempre, seu - brian, shaw
13:30	09, 11 - #tvoh, babette - #tvoh, ivar
14:00	futuro,será - acha,que,você - cia,david, director,petraeus, resigns
14:30	#emawinbieber, #mtvema, URL, at, be, bieber, big, i, justin, pick, the, think, tweet, will, winner, your *(written as '#emawinbieber,..')* - give, love, me - #tvoh, johannes
15:00	#emawinkaty, #mtvema, URL, at, be, big, i, katy, perry, pick, the, think, tweet, will, winner, your - #emawingaga, #mtvema, at, big, pick, the, tweet, winner, your - '#emawinbieber,..'
15:30	'#emawinbieber,..' - #atatürkenot, atam, atatürkenot - qui, veut
16:00	brown, mike - '#emawinbieber,..' - qui, veut
16:30	brown, mike - #qvemf, lilou - '#emawinbieber,..'
17:00	brown, mike - bulan, lahir - @venomextreme, venom
17:30	'@boyquotations,..' - '#emawinbieber,..'
18:00	#ullychat, qual - #ullychat, @ullylages, qual - brown, mike
18:30	o, pensado, que, sobre, tem, ultimamente, você - URL, business, i, online - coffee, green
19:00	'@boyquotations,..' - o, que, tem, você - o, pensado, que, sobre, tem, ultimamente, você
19:30	#iwillneverunderstand, why - o, pensado, que, sobre, tem, ultimamente, você - #mtvema, URL, at, be, big, pick, the, tweet, will, winner, your
20:00	#iwillneverunderstand,why - o, pensado, que, sobre, tem, ultimamente, você - o,pensado,que,tem,ultimamente,você
20:30	hari, pahlawan, selamat - #mtvema, URL, at, be, big, pick, the, tweet, will, winner, your - #mtvema, URL, at, be, big, the, tweet, will, winner
21:00	hari, pahlawan, selamat - #mtvema, URL, at, be, big, pick, the, tweet, will, winner, your - #iwillneverunderstand, why
21:30	hari, pahlawan, selamat - #iwillneverunderstand, why - pick, your
22:00	hari, pahlawan, selamat - and, broke, selena, up - #iwillneverunderstand, why
22:30	#asdtanya, #hmmomspik, @ahspeakdoang - #iwillneverunderstand, why - hari, pahlawan, selamat
23:00	#iwillneverunderstand, why - brown, mike - hari, pahlawan, selamat
23:30	10, nov - brown, mike - 10, 11, 2012
00:00	ataturk, aniyoruz, kemal, mustafa - hari, pahlawan, selamat - #iwillneverunderstand, why

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

international conference on Information and knowledge management, pages 2491–2494. ACM, 2012.

[4] 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.

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

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.