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] [?], 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 [?]. 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 we describe how using term N-Grams of variable lengths filters out many language constructs. In section 6 we describe the use of differential temporal features to rank interesting itemsets. In section 7 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 9 we discuss related work that applied Frequent Itemsets Mining on text. Finally, we finish by the conclusion and future work in section 10.

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 [?] 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 [?] 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

Items with the highest frequencies within transactions sharing a common prefix are most likely language constructs, but they can also be topical words

In section 3 we reduce the number of itemsets produced through flattening the peakedness of this head.

Itemsets whose lowest frequency item is within the mid-range of frequencies contain interesting information, but there are too many of them. The closed property selects non-redundant itemsets, but it is easily violated. It also doesn't solve redundancy from slight variations in language.

In section 4 we propose the strong closed property. It selects only itemsets comprizing a high proportion of the support of their closed subsets, after combining... For redundancy because of the use of language, clustering of the remaining itemsets solves the problem, as discussed in section 8.

`the`
`obama obama`
`bieber`
`president`
`justin`
`won won`
`barack`
`selena`
`elections`
`gomez`
`#elections2012`
`#elections2012`
`#mtvema`

After mining itemsets based on support and reducing their redundancy, we rank the remiaining itemsets using temporal features. The majority of itemsets has low support, specially in text where frequency has a Zipfean distribution. Section 9 describes our ranking scheme.

Items with low frequency; excluded by Frequent Itemset Mining since they are below the support threshold
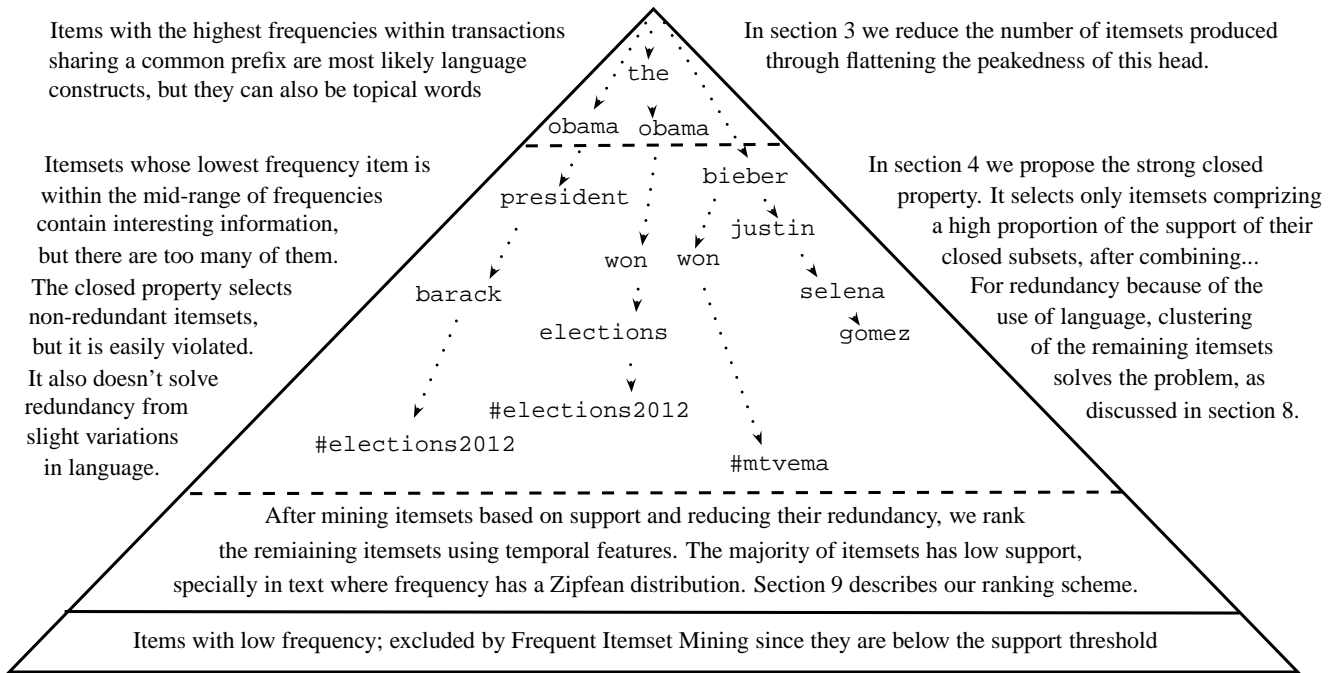
Figure 1: Roadmap of the paper overlaid on an frequency ordered prefix tree

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 [?], which we will describe in detail in section **??**. We use the implementation submitted to the workshop for Frequent Itemset Mining Implementations (FIMI) '04 [?], 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.

The problem that there are too many itemsets, with a lot of noise and redundancy, is addressed either by clustering [?] the itemsets or by mining only itemsets that satisfy a certain condition. The closure condition [?] is a prominent condition upon which many other conditions are based. Most similar to the strongly closed itemsets we propose are the δ-covered [?] and the δ-free [?] 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" [?]. 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. In fact, 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 we made in order to find the most informative itemsets mined from social media text. In [?], LCM is also used to mine Twitter posts and then a few itemsets are selected, but they are selected according to their utility in a lossy compression algorithm, SPUR. SPUR is based on representing transactions as itemsets that cover them, and it might introduce false positives and negatives. The algorithm could meet the

desired constraints on memory consumption (set at 20MB) and on processing data as a stream, but there was no justification why such tight constraints should be set. On the other hand, the quality of the summary seems to be affected by the choice of utility function, but the only assessment made about this was by showing that the actual transactions can be reconstructed from the summary with an accuracy that is expected to be high "for both recent data and frequent keywords". In the experiments, non-negative matrix factorization is used to extract topics from parts of the summary matching the query "world cup" and specific time intervals (before certain matches), but 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. On the other hand, improvements in query expansion performance have been reported in [?], because of using term weights based on itemsets known to come from related documents as well as one from unrelated documents.

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. [**?**] 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. We proceed by describing how to implement this algorithm using an inverted index.

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 *suffex 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

4.1 An inverted index based implementation

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.

The algorithm described works well on the sparse data typical to text mining.

5. MINING TERM N-GRAMS

If we can treat sequential language constructs, and really any other multiword expression, as one item we reduce the number of itemsets produced in these two ways.

There are many measures of association that can be used to detect multiword expressions [?], but each measure is good under certain conditions and has special properties. 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; 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, then we create two term bigrams for each unigram appearing with a probability above a certain threshold, η, by attaching to it the unigrams before and after it. We do this recursively by increasing N and creating two (N+1)-Grams for each N-Gram above

Input: α: Dynamic support ratio
Data: E: Epoch of data
Result: C: Closed itemsets having support α within E
1 C $\leftarrow \{\langle \emptyset, E\rangle\}$;   // $\emptyset$ is a closed itemset!
2 X $\leftarrow$ Inverted index of E;
3 **foreach** $i \in X.tokens$ **do**
4     $D_{\{i\}} \leftarrow$ X.postingsList[i];
5   **if** $\|D_{\{i\}}\| \geq \alpha \frac{\|E\|}{E.span}$ **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[$1 \ldots i_n$] $\leftarrow 0$;
11   suffix $\leftarrow \{i_{sh}\}$;
12   **foreach** $d \in D_S$ **do**
13   **foreach** $i \in d$ **do**
14   frequency[i]++;
15   **if** *frequency[i]* $= \|D_S\|$ **then** suffix.add(i) ;
16   **end**
17   **end**
18   **if** $\exists j : i_{sh} \succ suffix[j]$ **then return**;
19   C.add($\langle S \cup suffix, D_S\rangle$);
20   **foreach** $i \succ i_{sh}$ *and* $i \notin suffix$ **do**
21   **if** *frequency[i]* $\geq \alpha \frac{\|E\|}{E.span}$ **then**
22             $D \leftarrow D_S \cap i$; // Results of query S AND i
23   LCM($S \cup suffix \cup \{i\}, i, D$)
24   **end**
25   **end**
26 **end**

Algorithm 1: LCM Frequent Itemsets Mining

the threshold until there are no more N-Grams above the threshold. Following we present an example of tokenizing "Barack Obama president of the United States":

1. First the tokens will be: `barack`, `obama`, `president`, `of`, `the`, `united`, `states`.

2. Assume that after tokenizing the whole epoch we find that the probability of `obama`, `of` and `the` within the current epoch is more than η. Therefore the tokens are changed to: `barack`, `barack + obama`, `obama + president`, `president`, `president + of`, `of + the`, `the + united`, `united`, `states`.

3. Assume that the bigrams `barack + obama` and `of + the` still appear with a probability higher than η. Therefore trigrams are generated by attaching the unigrams before and after them, and the tokens become: `barack`, `barack + obama + president`, `obama + president`, `president`, `president + of`, `president + of + the`, `of + the + united`, `the + united`, `united`, `states`

4. At N=3, we find that no trigrams still appear with probability greater than η, and thus terminate.

Note that in the above procedure we didn't prevent overlap, because there is no guarantee that the N-Gram created makes any sense. The goal is to create a single unit out of frequent sequences and this is achieved because the N-Gram keeps growing as long as it is frequent, which will stop as soon as a unigram that is not part of the expression is attached to it. We can look at this as flattening the Zipfian distribution by creating more tokens with less frequency each, but the distribution of the input doesn't affect the results of itemsets mining - only its runtime can be affected.

Figure 2 shows the effect of increasing the maximum length of N-Grams from 1 to 5 on the number of tokens, the number of itemsets mined, 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, which is the probability of the term 'obama' within the whole collection of Tweets. The support threshold used for mining in this experiment and all other experiments is also derived from the probability of the term 'obama', since it is a known to be steadily frequent and talked about in all languages. The average number of Tweets per hour is 100000, so the term 'obama' is expected to appear 10 times per hour on average. We use a minimum support threshold of 10, which translates into $\alpha = 0.0002$. Figure 2(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[2]. Such Tweets are tokenized into more 4-Grams than 5-Gram, and the 4-Grams appearing in them don't appear elsewhere and thus each two of them are reduced into one when N=5. Figure 2(b) shows that the number of itemsets keeps decreasing as expected. Figure 2(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 affecting by the sparsity of data.
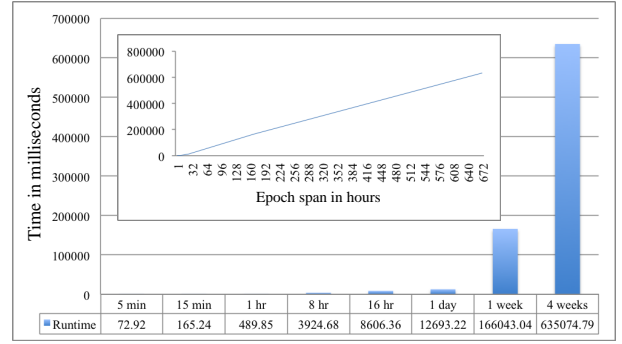
[2]http://www.game-insight.com/

Figure 3: Mean runtime at different epoch spans

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 also avoids the possibility that an itemset might be represented differently at different epochs. Actually, within one epoch an itemset will be represented as different N-Gram sets, and we merge them during flattening by taking the union of the posting lists of all the N-Gram sets flattened to the same itemset.

6. TEMPORAL MINING
KL-Divergence

7. STRONGLY CLOSED ITEMSETS ALLIANCES
The closed property of an itemset is very easily violated by modifying one document 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 9th, 2012 many people where tweeting that "Justin Bieber and Selena Gomez broke up". If all Tweets which use the verb "break up" to report the topic also contain the two names in full, then there will be one closed itemset with all 6 items. However, a Tweet can contain any of 8 other combinations of the four names that fully convey the meaning. We can consider that Tweets with any of these combinations are modifications of the *maximal* closed itemset. Therefore instead of 1 maximal itemset about the topic there will be 9 closed ones. Now consider that it is possible to say that "Justin and Selena broke up" as well as "Justin broke up with Selena", resulting in 2 maximal itemsets. This increase in the number of maximal itemsets because of slight variations in the language is intensified by the length limit on Tweets, specially when people try to make space for their comment about a Tweet they are retweeting. We therefore need a property between the easily violated closed property and the very strict maximal property which also results in a large number of redundant itemsets.

We define a *strongly closed* itemset as a closed itemset whose frequency comprises more than a certain proportion of the frequency of its least frequent subset, or which has at least one strongly closed superset. This property chooses closed itemsets which violate the closed property of their subsets by a significant amount. It also filters out itemsets that are short language constructs, because they appear very
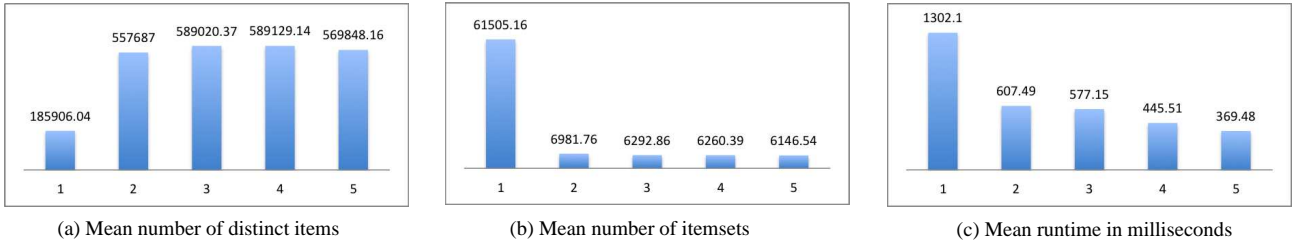
(a) Mean number of distinct items   (b) Mean number of itemsets   (c) Mean runtime in milliseconds

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

frequently and none of their numerous supersets comprise a significant proportion of their high frequency. Notice that within the documents containing a closed itemset, $D_{itemset}$, the probability of a closed superset ($\|D_{superset}\|/\|D_{itemset}\|$) is called the confidence of the rule "$itemset \rightarrow superset$." This property is the basic property used for association rules mining, and some algorithms for mining itemsets based on a variation called *all confidence* have been proposed [?]. The strongly closed property builds on the notion of confidence by choosing itemsets which would result in rules of high confidence. The minimum acceptable confidence of the resulting rule is a parameter κ that can vary between 0 and 1 to increase the strength of the strongly closed property.

Even though choosing strongly closed itemsets filters out many redundant itemsets formed because of slight variations of the same topic, it is still not a strong enough property. The intersection of the sets of documents containing two strongly closed itemsets can be different by only 1 document from either of the sets. For example, the closed itemset {justin, selena} can have three closed supersets: {justin, selena, broke, up, #jelena}, {justin, selena, broke, up} and {justin, selena, #jelena}. Figure 7 illustrates how all supersets can be strongly closed, while they are still redundant itemsets mined from almost the same documents. Each circle in the figure represents the set of documents containing the superset 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 strongly closed property.
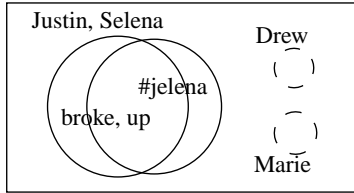

Figure 4: Strongly and not strongly closed itemsets

To remove such redundancy, we merge strongly closed itemsets with high cosine similarity into an itemset *alliance*. The itemset alliance is a bag of items formed by taking the union of all member itemsets. This concentrates the numerous itemsets about a certain topic into one unit. The alliance also include itemsets that are not supersets of one another; for example, itemsets that differ because of items that come earlier in the total ordering and are not pertaining to the topic, such as "with" and "and" in the Justin-Selena example. The use of cosine similarity limits the candidates for merging to itemsets that are very likely about the same

topic, since the Inverse Document Frequency (IDF) value of terms pertaining to a certain topics is typically much higher than the IDF value of other terms. The IDF and cosine similarity formulae used are given in equations 2 and 3 respectively.

$$IDF(i) = \log \frac{\|E\|}{\|D_i\|} \qquad (2)$$

$$cos(S_1, S_2) = \frac{\sqrt{\sum_{i \in S_1 \cap S_2} IDF(i)^2}}{\sqrt{\sum_{i \in S_1} IDF(i)^2} * \sqrt{\sum_{i \in S_2} IDF(i)^2}} \qquad (3)$$

To avoid forming alliances that are not coherent about a certain topic, we calculate the difference between the posting lists of every two itemsets being merged. The merge is permitted only if the difference is below the maximum number of different document allowed for the required closed strength, 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.

$$\Delta(S_1, S_2, \kappa) = (1 - \kappa) * max(\|D_{S_1}\|, \|D_{S_2}\|), \ \kappa \in [0, 1[\qquad (4)$$

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
```
 1  for i ← 2 to ‖S‖ do
 2      C ← ∅ ;          // Candidates for merging with S_i
 3      T ← ∅ ;              // Subsets of current itemset
 4      for j ← 1 to i − 1 do
 5          if ‖S_i ∩ S_j‖ = min(‖S_i‖, ‖S_j‖) then
 6              C.add(S_j);
 7              if ‖S_i‖ > ‖S_j‖ then T.add(S_j);
 8          else if cos(S_i, S_j) ≥ θ then
 9              C.add(S_j);
10          end
11      end
12      M.score ← ∞ ; // Best merge candidate's score
13      foreach S_c ∈ C do
14          if S_c ∈ T then
15              δ ← ‖D_{S_c}‖ − ‖D_{S_i}‖;
16          else
17              δ ← difference(D_{S_i}, D_{S_c}) ;  // Stops early
18          end
19          if δ ≤ Δ(S_i, S_c, κ) and δ < M.score then
20              M ← S_c;
21              M.score ← δ;
22          end
23      end
24      if M.score < ∞ then
25          A[M].itemset ← A[M].itemset ∪ S_i ∪ M;
26      end
27  end
28  return A;
```
Algorithm 2: Merging itemsets into alliances

7.1 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.

7.2 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:

$$
\begin{aligned}
IG(P(S_i), Q(S_i)) &= KLD(P(S_i)\|Q(S_i)) \\
&= -(H(P(S_i)) - H(P(S_i), Q(S_i))) \\
&= \sum P(S_i) \log P(S_i) - \sum P(S_i) \log Q(S_i) \quad (5)
\end{aligned}
$$

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:

$$
PMI(S_i, S_j) = \log \frac{P(S_i, S_j)}{P(S_i)P(S_j)} = -\log P(S_i) = I(S_i) \quad (6)
$$

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:

$$
\overline{IG}_{alliance} = \frac{IG(P(S_{i1}, ...S_{im}), Q(S_{i1}, ...S_{im}))}{m} \quad (7)
$$

The ranking of itemset alliances according to equation 7 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.

8. 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 6th and 9th 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 7.2, for each hour. Times are in the EST zone.

On November 6th, 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 9th, 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 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.

9. RELATED WORK

10. CONCLUSION AND FUTURE WORK

11. ACKNOWLEDGEMENT

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

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