Adapting Frequent Itemsets Mining for Social Media Text by Strengthening the Closed Property

ABSTRACT

Abstract

1. INTRODUCTION

Mining user generated content has been a hot topic since the beginning of Web 2.0. **didn't I say I dislike such inaugrational sentences.. ehem!** Businesses are interested in getting direct insights from consumers without incurring costs for surveys, and so are governments, celebrities, and above all advertisers and market analysts. Researchers are interested because methods developed for other media does not necessarily work for social media. This is particularly correct for Twitter because of its length restriction, high volume and lack of structure. The length restriction has resulted in a very particular language use as detailed in **??**, but it seems to have also played a dual role in encouraging people to keep Tweeting about everything they do, think or feel. This resulted in a very high volume of Tweets[1], most of which is personal updates but some of which represent what is happening now. The speed of news on Twitter has been known to exceed the news on professional news agencies [**?**], and it even travels faster than earthquakes [**?**]. First stories that appear in Twitter rather than official news are covered by Twitter user present at the event, and able to provide content that cannot be acquired after the fact such as the picture of the emergency landing on the Hudson river [**?**]. Such stories are noticed because of the natural collaborative filtering act of retweeting good content, and the use of the community developed convention of Hashtags. The Twitter website makes use of this by following the rise in the volume of keywords, and featuring keywords whose rise in volume accelerates more than any other. However, this does not catch topics with sustained interest. It also fails to catch important topics for which the interest is peaking within a certain niche of users. Further it is almost always polluted by uninteresting topics that are made trending. In this paper we propose...

[1]We receive an average of 100,000 Tweets per hour from Spritzer

2. FREQUENT ITEMSET MINING FOR SOCIAL MEDIA

Short non-canonical text
Works regardless of language... not only English

2.1 Preliminaries

A *frequent itemset* is a set of items that occur together frequently; for example, when mining Tweets posted on November 6th, 2012 using terms as items, the set {of, president, states, the, united} is a frequent itemset (ordered lexicographically). Traditionally, Frequent Itemset Mining is applied to a *database* of *transactions* made at a retail store. This terminology is suitable for market basket data, but as we are mining social media text we use the term *document* instead of transaction. It is noteworthy that this term is used as a matter of convention, yet our methods are tailored for the short "documents" typical of social media as described earlier. Because of the dynamic nature of social media, we call the input to a mining algorithm an *epoch* of data rather than a database. An epoch of data is all documents posted in a certain period of time, the length of this period is the epoch's *span*.

The *support* of an itemset is the number of times it appears, as a ratio of the number of documents in the epoch. The *minimum support* is a threshold separating *frequent* itemsets from *infrequent* ones. Even though selecting itemsets based on their frequency of appearance was criticized in the domain of market basket data because it generates obvious itemsets, it makes sense in the domain of social media because this acts precisely as a collaborative filter.

Following is the notation used in this paper:

- $I = \{i_1, i_2, .., i_n\}$: The set of possible items (vocabulary).
- $d_a = \{i_{a1}, i_{a2}, ..\}$: A document made up of as a set of items, not necessarily terms. Each document has a sequential id denoted by the subscript letter.
- $E^{span} = \langle d_a, d_b, ..\rangle$: An epoch of data of a certain span, such as an hour, made up of a sequence of documents.
- $S \subset I$: An itemset, its support is given by $\|D_S\|$.
- D_S: All documents containing itemset S.
- $\|.\|$: The norm operator; gives the size of the operand.

3. BASIC ALGORITHM

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 corpus of documents. This requires producing all unordered pairs of *1*-itemsets (terms), after pruning out rare ones that appear less than the minimum support. 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.

To overcome the bottleneck of *Candidate Generation*, many algorithms are proposed to take hints from the transaction space rather than operating blindly in the items 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:

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

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 such that any closed itemset is considered an extension of exactly one of its subsets. Thus during candidate generation all its 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 *suffex* 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 documents. We proceed by describing how to implement this algorithm using an inverted index.

3.1 An inverted index based implementation

We show in 1 how to implement LCM and PPO-Extension using an inverted index. 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 tokens list follow the total ordering; all itemsets of length 1 will be considered anyway.

4. MINING TERM N-GRAMS

The use of N-Grams to overcome itemsets from language constructs

5. 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

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 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

```
Input: $\alpha$: Minimum support ratio
Data: E: Epoch of documents
Result: C: Closed itemsets having support $\alpha$ within E
1  C ← $\{\langle\emptyset, \|E\|\rangle\}$ ;            // $\emptyset$ is a closed itemset
2  X ← Inverted index of E;
3  foreach $i \in X.tokens$ do
4      $D_{\{i\}}$ ← X.postingsList[$i$];
5      if $\|D_{\{i\}}\| \geq \alpha\|E\|$ 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 containing S) is
10     frequency[$1 \dots i_n$] ← 0;
11     suffix ← $\{i_{sh}\}$;
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] $\geq \alpha\|E\|$ 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

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 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* → *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 5 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.

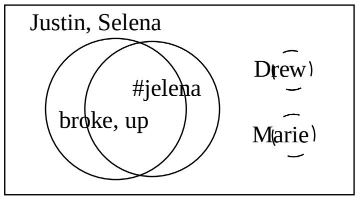

Figure 1: Strongly and not strongly closed itemsets

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.

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

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

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.

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

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.

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.

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?

6. TEMPORAL MINING

KL-Divergence

7. EXPERIMENTAL EVALUATION

Examples from Nov. 6 and 9. Nov. 9 has two events for Bieber.. MTVEMA and break up.

8. CONCLUSION

```
Input: $\theta$: Minimum cosine similarity,
$\kappa$: 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

9. ACKNOWLEDGEMENT