University of Waterloo E-Thesis Template for $\LaTeX$

by

Pat Neugraad

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Science
in
Zoology

Waterloo, Ontario, Canada, 2007

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

This is the abstract.

Vulputate minim vel consequat praesent at vel iusto et, ex delenit, esse euismod luptatum augue ut sit et eu vel augue autem feugiat, quis ad dolore. Nulla vel, laoreet lobortis te commodo elit qui aliquam enim ex iriure ea ullamcorper nostrud lorem, lorem laoreet eu ex ut vel in zzril wisi quis. Nisl in autem praesent dignissim, sit vel aliquam at te, vero dolor molestie consequat.

Tation iriure sed wisi feugait odio dolore illum duis in accumsan velit illum consequat consequat ipsum molestie duis duis ut ullamcorper. Duis exerci odio blandit vero dolore eros odio amet et nisl in nostrud consequat iusto eum suscipit autem vero. Iusto dolore exerci, ut erat ex, magna in facilisis duis amet feugait augue accumsan zzril delenit aliquip dignissim at. Nisl molestie nibh, vulputate feugait nibh luptatum ea delenit nostrud dolore minim veniam odio volutpat delenit nulla accumsan eum vero ullamcorper eum. Augue velit veniam, dolor, exerci ea feugiat nulla molestie, veniam nonummy nulla dolore tincidunt, consectetuer dolore nulla ipsum commodo.

At nostrud lorem, lorem laoreet eu ex ut vel in zzril wisi. Suscipit consequat in autem praesent dignissim, sit vel aliquam at te, vero dolor molestie consequat eros tation facilisi diam dolor. Odio luptatum dolor in facilisis et facilisi et adipiscing suscipit eu iusto praesent enim, euismod consectetuer feugait duis. Odio veniam et iriure ad qui nonummy aliquip at qui augue quis vel diam, nulla. Autem exerci tation iusto, hendrerit et, tation esse consequat ut velit te dignissim eu esse eros facilisis lobortis, lobortis hendrerit esse dignissim nisl. Nibh nulla minim vel consequat praesent at vel iusto et, ex delenit, esse euismod luptatum.

Ut eum vero ullamcorper eum ad velit veniam, dolor, exerci ea feugiat nulla molestie, veniam nonummy nulla. Elit tincidunt, consectetuer dolore nulla ipsum commodo, ut, at qui blandit suscipit accumsan feugiat vel praesent. In dolor, ea elit suscipit nisl blandit hendrerit zzril. Sit enim, et dolore blandit illum enim duis feugiat velit consequat iriure sed wisi feugait odio dolore illum duis. Et accumsan velit illum consequat consequat ipsum molestie duis duis ut ullamcorper nulla exerci odio blandit vero dolore eros odio amet et.

In augue quis vel diam, nulla dolore exerci tation iusto, hendrerit et, tation esse consequat ut velit. Duis dignissim eu esse eros facilisis lobortis, lobortis hendrerit esse dignissim nisl illum nulla minim vel consequat praesent at vel iusto et, ex delenit, esse euismod. Nulla augue ut sit et eu vel augue autem feugiat, quis ad dolore te vel, laoreet lobortis te commodo elit qui aliquam enim ex iriure. Ut ullamcorper nostrud lorem, lorem laoreet eu ex ut vel in zzril wisi quis consequat in autem praesent dignissim, sit vel. Dolore at te, vero

dolor molestie consequat eros tation facilisi diam. Feugait augue luptatum dolor in facilisis et facilisi et adipiscing suscipit eu iusto praesent enim, euismod consectetuer feugait duis vulputate veniam et.

Ad eros odio amet et nisl in nostrud consequat iusto eum suscipit autem vero enim dolore exerci, ut. Esse ex, magna in facilisis duis amet feugait augue accumsan zzril. Lobortis aliquip dignissim at, in molestie nibh, vulputate feugait nibh luptatum ea delenit nostrud dolore minim veniam odio. Euismod delenit nulla accumsan eum vero ullamcorper eum ad velit veniam. Quis, exerci ea feugiat nulla molestie, veniam nonummy nulla. Elit tincidunt, consectetuer dolore nulla ipsum commodo, ut, at qui blandit suscipit accumsan feugiat vel praesent.

Dolor zzril wisi quis consequat in autem praesent dignissim, sit vel aliquam at te, vero. Duis molestie consequat eros tation facilisi diam dolor augue. Dolore dolor in facilisis et facilisi et adipiscing suscipit eu iusto praesent enim, euismod consectetuer feugait duis vulputate.

Acknowledgements

I would like to thank all the little people who made this possible.

Dedication

This is dedicated to the one I love.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Introduction

The nature of text in social media poses a challenge when applying traditional text mining algorithms. Text in social media is usually short, lacks context and structure, and is created at a high rate. To realize the benefits of social media in giving a voice to ordinary people, this work focuses on the content of posts. However, the collective stream from all users is overwhelmed with personal updates, and timely finding posts about topics of interest requires an efficient mining algorithm. The frequent itemset mining family of algorithms is fast and efficient, however it is not readily suited for application on text. First, the number of itemsets mined is large and grows with the number of distinct items – which is particularly high in the text domain. Frequent itemset mining was originally proposed as a preliminary stage for association rules mining, which sifts through the numerous itemsets and produces a smaller number of association rules. To reduce the number of itemsets, they may be limited by setting a high frequency threshold, but this is not possible in text mining because frequencies of items follow a long-tailed Zipfean distribution. Second, the high frequency of stop words and language constructs is a problem that frequent itemset mining is not equipped to handle. Even if a maximum frequency threshold is set, incurring the risk that we will filter out important itemsets, many non-English constructs will be mined because the proportion of posts in English is much higher than other languages. Finally, there is considerable redundancy in frequent itemsets caused by trivial differences in the language used. In this paper we address those problems, adapting frequent itemset mining to social media text without degrading its efficiency.

Figure 1.1: Most important contributions overlaid on a frequency ordered prefix tree

Unlike trending topics[1] [?], the results of frequent itemset mining include itemsets that have high frequency because of sustained interest, as well as a spike of interest. The mined itemsets provide the vocabulary associated with events and can be used as a preliminary step for search and summarization. For example, the collection of mining results from different epochs of time can be used for temporal query and document expansion [?, ?]. The results from each epoch can be treated as a document, facilitating the creation of a "temporal profile" [?] of the query or a document being expanded. For summarization, frequent itemsets can provide a good foundation for summary creation. While the frequent itemsets themselves are not summaries, since they lack qualitative properties such as coherence and cohesion, the results are understandable at the user level. As we shall see in later examples, the top ranked itemsets cover a variety of open topics, and within one topic different opinions are reported as separate contrastive itemsets.

The next three sections provide necessary background. We start by discussing related work in section ??, we then explain frequent itemset mining and the algorithm on which we build our work in sections ?? and ?? respectively. In sections ??, ?? and ?? we propose solutions to different problems faced when applying frequent itemset mining to social media text. We present the outline of those sections graphically using a frequency ordered prefix tree of the itemsets. Each path from root to a leaf in such a tree represents an itemset, where each itemset is ordered in non-increasing order of the frequency of items. Itemsets having the same prefix share the nodes representing this prefix. This representation is typical in the literature, and it is actually a very good representation of the frequent itemset mining problem. Figure ?? shows conceptually how our contributions in sections ??, ?? and ?? affect different parts of the problem. We overlay the contributions on such a figure to make it clear how each one addresses a particular challenge of applying frequent itemset mining to text data from social media. In section ??, we conclude our presentation and suggest future directions.

1.2 Related Work

Frequent itemset mining comprises a large body of work that goes back to the early 1990s. We cover the topic only briefly, as the focus of this paper is not frequent itemset mining but

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

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

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

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

Likewise, Yan et al. [?] choose K *master* patterns as representatives of the data. A *master* pattern is the union of all itemsets in a cluster, similar to our proposed *strongly closed itemset*. While the use of clustering is similarly motivated by the trivial difference between itemsets, the K *master* itemsets have to cover the whole data, unlike the *strongly closed itemsets* which actually avoid clustering together different topics or different opinions within a topic.

Our work complements the work done by Yang et al. [**?**] for using frequent itemsets as a temporal summary. They have proposed a framework for storing mining results of temporally consecutive batches of data using a pyramidal time window. Their framework allows for fast execution of temporal queries, and for tracking the evolution of topics. The choice of itemsets from each batch is based on their utility in compressing the data, and thus they use non-negative matrix factorization to extract topics related to a query. We propose methods to choose itemsets that are topically relevant from each batch, making use of the nature of social media to choose topically relevant itemsets without the need for a query. Social media has been shown to be a good and timely source for discovering real-world events [**?**, **?**].

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

1.3 Frequent itemset mining

1.3.1 Preliminaries

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

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

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

1.3.2 Fundamentals

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

1.4 Basic Algorithm

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

$$\mathcal{C} = \{s_c :\ s_c \subset W\, and \nexists\, s_d\, where\, s_c \subset s_d\, and\, |T_{s_c}| = |T_{s_d}|\} \tag{1.1}$$

The properties of closed itemsets are as follows:

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

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

1. An itemset can be extended 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, we add to its *suffix* all items whose frequency within T_s is equal to $|T_s|$.

3. If any item in the *suffix* is *smaller* than the suffix head, prune this solution branch. All closed itemsets within this branch have already been generated.

Table **??** is an example of how *PPC-Extentsion* is used to generate closed itemsets starting from the *1*-itemset 'barack'. The upper table enumerates $T_{\{barack\}}$. The lower table shows steps of itemsets generation. The current solution along with its frequency is in column 2, solutions marked by an (*) are the closed itemsets emitted. All possible extension items and their frequencies are in column 3 with the one being considered bolded. Column 4 is a comment explaining the step. At each step, a pass is done on $T_{itemset}$ to enumerate and count possible extension items. To enforce a support threshold infrequent extension items are removed, but in this example there is no such threshold. Notice that the number of steps is linear in the number of closed itemsets, and the only additional storage required, besides storage for the documents, is that required for possible extension items. Of course, this is a simplified example, but it shows in essence how LCM achieves its low run time and memory requirements. We refer the interested reader to Uno et al.[**?**] 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 **??**. We proceed by describing how to implement this algorithm using an inverted index.

1.4.1 Implementation Details

We show in algorithm **??** 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 above the threshold. Along with each itemset in the solution, it also outputs the transactions in which it occurs – which is represented as $\langle items, T_{itemset} \rangle$. The symbol $\succ$ denotes that the lefthand side succeeds the righthand side in the total ordering.

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 $T_{itemset}$
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 is not ordered, prune solution

Closed itemsets containing 'barack'

Table 1.1: Generation of closed itemsets by Prefix Preserving Closure Extension

The algorithm also lends itself to distributed implementations. For example, a map/reduce implementation is straightforward 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 unnecessary overhead for all but the largest data sets. 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.

1.5 Mining social media

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

We apply the algorithms to epochs of data, so they are not strictly stream processing algorithms. However, we regard the process as mining a sliding window that is moved forward by time steps of short span. The time step must be longer than the time needed to mine an epoch of data, and the performance of our algorithms makes it possible to use a time step of a few seconds for epochs up to a day long. Figure **??** shows the runtime of LCM on epochs of increasing length, and we will show in section **??** that our extensions do not degrade performance. The times reported in figure **??** are averages across all epochs of the specified length in the last 3 months of 2012, using a time step that is half the epoch length. The variance is very low and the confidence bands are not shown because they appear as dots.

The support threshold used throughout this paper 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; i.e., 'obama'. The maximum likelihood estimate of the probability of the term 'obama' within the whole collection of tweets is 0.0001. The average number of tweets per hour is 116920.21, so the term 'obama' is

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

```
Input: α: Dynamic support ratio
Data: E: Epoch of data
Result: C: Closed itemsets having support α within E
C ← {⟨∅, E⟩} ;                                        // ∅ is a closed itemset
X ← Inverted index of E;
foreach w ∈ X.tokens do
    T_{w} ← X.postingsList[w];
    if |T_{w}| ≥ α |E|/E.span then LCM({w}, w, T_{w}) ;
end
return C;
Function LCM(s: Current itemset, w_sh: Suffix head,
T_s: Transactions (tweets) containing s) is
    frequency[1 ... w_n] ← 0;
    suffix ← {w_sh};
    foreach t ∈ T_s do
        foreach w ∈ t do
            frequency[w]++;
            if frequency[w] = |T_s| then suffix.add(w) ;
        end
    end
    if ∃v ∈ suffix : w_sh ≻ v then return;
    C.add(⟨s ∪ suffix, T_s⟩);
    foreach v ≻ w_sh and v ∉ suffix do
        if frequency[v] ≥ α |E|/E.span then
            T ← T_s ∩ v ;                              // Results of query s AND v
            LCM(s ∪ suffix ∪ {v}, v, T)
        end
    end
end
```

Algorithm 1: LCM frequent itemsets mining

Figure 1.2: Mean runtime at different epoch spans

expected to appear 12 times per hour on average. Thus, we use a minimum support threshold of 12, which translates into $\alpha = 0.0002$.

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

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

1.5.1 Mining Term N-grams

Figure 1.3: Effect of the increasing maximum N-Gram length on results of mining of 1hr epochs of data

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

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

The threshold of high probability is different for each value of N. The threshold for unigrams is determined in a similar fashion to how we determined the support threshold. We use $\eta_1 = P(\text{“}obama\text{”}) = 0.0001$. At each N, the probability threshold is adjusted to account for the increase in the number of tokens and the overall increase in the grand sum of counts (caused by overlap). The adjusted η_N is:

$$\eta_N = \eta_1 \times \frac{\sum_{\{w:\, w \in W \, and \, w.length \leq N\}} freq(w)}{\sum_{\{v:\, v \in W \, and \, v.length = 1\}} freq(v)} \tag{1.2}$$

Figure **??** 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, and the runtime of mining one hour of data. The values shown are averages across all one-hour epochs in the month of November 2012. The value of η used is 0.0001. Figure **??**(a) shows that the number of distinct items increases substantially as N goes from 1 to 2, then continues increasing slightly until it starts decreasing at N=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-grams, and the 4-grams appearing in them do not appear elsewhere. Thus, each pair is reduced to one 5-gram. Figure **??**(b) shows that the number of itemsets continues to decrease as expected. Figure **??**(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 than those in figure **??** because they do not include the time taken for writing the posting list of each itemset.

1.6 Filtering itemsets

In the previous section, we discussed our handling of function words, using a technique that exploits LCM's tolerance to sparsity. After applying this technique, the average number of

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

itemsets mined from an hour of twitter data drops from 61,505 to 6,146. However, there is still redundancy in the itemsets.

The closed property of an itemset is easily violated by modifying one transaction that contains the itemset and removing one of its items. While an update operation is not supported in the model of frequent itemsets mining, a similar effect happens when people are writing about a certain fine grained topic. For example, figure **??** illustrates itemsets related to Donald Trump's famous tweets in reaction to Obama's victory in 2012[4]. Each area in the figure represents the transactions containing the itemset formed by concatenating the items in all intersecting ellipses.

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

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

We propose two conditions that are not as easily violated as the closed condition for selecting itemsets. The two conditions build on the concept of *association rule confidence*. Confidence is the basic property used for association rules mining, and it is used in the definition of δ-free sets [**?**]. 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 it cannot be directly used. The confidence of an assocation rule that the presence of an itemset, s_j, implies the presence of another itemset, s_i, is defined as:

$$conf(s_j \rightarrow s_i) = \frac{|T_{s_i} \cap T_{s_j}|}{|T_{s_j}|} \tag{1.3}$$

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

Figure 1.4: Closed, distinct and strongly closed sets

1.6.1 Distinct Itemsets

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

$$\mathcal{D} = \{s : s \in \mathcal{C}\, and\, \exists\, s_p \subset s \; where \; \frac{|T_s|}{|T_{s_p}|} \geq \kappa\} \tag{1.4}$$

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

1.6.2 Strongly Closed Itemsets

To remove the redundancy in *distinct* itemsets we merge similar ones into *strongly closed* itemset clusters. The similarity of a *distinct* itemset, s_d, and another distinct itemset, s_c, is measured as the overlap of the *transactions* containing both of them with the transactions containing s_c. A *distinct* itemset is clustered with another itemset if one exists such that the overlap exceeds a similarity threshold, which we take to be $1 - \kappa$ (the indistinctiveness of s_d from s_c). If more than one satisfies this condition, the *distinct* itemset is clustered with the one having the highest overlap ratio. When a *distinct* itemset is clustered with an itemset that is already part of a cluster, the *distinct* itemset is added to the existing cluster. Finally, the *strongly closed* itemset is the union of all cluster members, and its support is the size of the union of their postings lists. We define the desired clustering and the strongly closed itemset represented by each cluster as follows:

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

$$S_l = \{w : \; w \in \bigcup_{(s_i,s_j) \in r_l} s_i \; where \; r_l \in \mathcal{R}\} \tag{1.5}$$

This clustering scheme selects the cluster that contains the itemset which maximizes the confidence of the rule $conf(s_j \to s_i)$, with a lower bound on the overlap to maintain distinctiveness. This clustering can be implemented efficiently using techniques similar to the ones proposed by Bayardo et al. [**?**]. The main ideas are to limit the comparisons to a few candidates, and to terminate the comparison early if the similarity threshold will not be met. In our case, the postings lists are longer than the itemsets, so we generate candidates for comparison by calculating similarity between itemsets. When calculating the similarity between two postings lists, we can terminate early if the difference exceeds the maximum difference permissible to achieve a similarity of $1 - \kappa$, which can be derived from equation **??**.

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

1.6.3 Performance Analysis

We analyze the performance of the filtering conditions proposed by applying them to the mining results of all one-hour long epochs in the Twitter data. The average number of itemsets mined from an hour-long epoch is 2439.17 closed itemsets of length 2 or more; that is, excluding itemsets that are merely a frequent item.

Figure **??** show the effect of varying κ on the mean number of *distinct* and *strong closed* itemsets. The number of *distinct* itemsets drops as the distinctiveness threshold increases. On the other hand, the number of *strong closed* clusters formed increases as the

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

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

Algorithm 2: Forming strongly closed itemset clusters

Figure 1.5: Effect of changing κ on mining results

Figure 1.6: Runtime of itemset mining alone and with filtering and clustering at different epoch spans

similarity (indistinctiveness) threshold decreases. The dashed line shows that the number of unclustered distinct itemsets reaches zero at $\kappa = 0.5$, explaining why the number of clusters changes very slightly after that. We use $\kappa = 0.25$ in the remainder of the paper, which is an arbitrary choice based on the definition not the data. The average number of itemsets (*strongly closed* and unclustered *distinct*) mined from one-hour epochs at this value of κ is only 224.48, which is about 10% of the number of closed itemsets.

Figure **??** shows the total runtime of the LCM algorithm plus filtering based on the distinct condition and clustering into strong closed itemsets at different epoch spans. The runtime of LCM alone is also plotted for reference. We also plot the performance of another frequent itemset mining algorithm, FP-Zhu [**?**], which was the runner up at FIMI 2004 [**?**]. We include it to show that our extensions do not degrade the performance of LCM even in the context of competitions. The y-Axis is in logarithmic scale to keep the scale of the plot suitable for seeing slight differences. The output of LCM is the input to the filtering and clustering step, so it is affected by the number of closed itemsets produced. This explains why it takes slightly longer time for clustering results from the 15-minute epoch and then takes a constant time for epochs of a longer span.

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

1.7 Temporal Ranking

The previous filtering steps reduce the number of itemsets to under 1% of their original number. In this section, we present a method for ranking itemset clusters according to their novelty when compared to other time periods. These itemsets can then be presented to users in rank order, providing a synopis of events in the epoch, or used as input to additional search or summarization steps.

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

$$\begin{aligned} IG(P(s_i), Q(s_i)) &= -(H(P(s_i)) - H(P(s_i), Q(s_i))) \\ &= \sum P(s_i) \log P(s_i) - \sum P(s_i) \log Q(s_i) \\ &= KLD(P(s_i) || Q(s_i)) \end{aligned} \tag{1.6}$$

To calculate the collective IG of a strongly closed itemset cluster, we have to take into account that the itemsets of the cluster are not independent. For simplicity we will consider only the pairwise dependence between every itemset and the smallest common subset. The joint probability of an itemset, s_i, and its superset, s_j, is equal to the probability of the superset. Thus, the IG of the appearance of an itemset and its superset together is essentially the IG of the superset. Also, their pointwise mutual information (PMI) is the self information (SI) of the subset. Therefore, the IG of a superset is different from the IG of its subset by the information gained because of the additional items.

Hence, the IG of a strongly closed itemset cluster, $S = \{s_{i1}, .., s_{im}\}$, can be approximated as the IG of its smallest subset, s_{min}, plus the differences between the IGs of member itemsets and the smallest subset. We use the squares of the differences, because it can be between two negative number, and we only care about the magnitude of the difference. To give the self information of the smallest subset the same influence, we also square it. Thus, the information gain of a strong closed itemset is given by:

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

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

$$\overline{IG}(S) = \frac{IG^2(P(s_{i1}, ...s_{im}), Q(s_{i1}, ...s_{im}))}{m} \tag{1.7}$$

1.7.1 Empirical Evaluation

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

Tables **??** and **??** show the top 3 itemsets for one-hour epochs in the days examined. The itemsets shown are the first appearances of the most interesting itemsets; that is, an hour is shown only if its top 3 feature novel interesting itemsets. The first column is the beginning of the hour, EST time. The second column is the top itemsets. The third column is a commentary to explain the itemsets, but we omit it from table **??** to save space since the itemsets are self explanatory.

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

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

As a form of comparison, in table **??** we show the top 3 itemsets picked by the MTV algorithm [**?**] for the same epochs and support. We show hours in which the top 3 itemsets included interesting itemsets. For brevity, we truncate itemsets that are complete tweets and append the tweet id for reference. The input to the algorithm was transactions made up of N-grams up to 5 terms long, which helped the algorithm converge faster since the distribution is flatter. The use of N-grams also overcomes the dominance of language constructs, which are otherwise ranked high in all hours. All the topically relevant itemsets chosen by MTV are present in the top 50 *strongly closed itemsets*.

1.8 Conclusion and future work

We have proposed a method for efficiently creating temporal synposes of social media streams, based on a frequent itemset mining algorithm that is suitable for sparse data, LCM. Our method summarizes an hour of Twitter data (116920 tweets on average) into 224 itemsets in 1945.68 milliseconds on average, and scales for longer epochs of data. The direct application of LCM on one-hour epochs of Twitter data results in an average of 61505.16 closed itemsets and takes 2506.58 milliseconds on average. The improvement is due to the following contributions: (1) strengthening the closure condition such that it selects an itemset only if it is distinctively different from its subsets and other itemsets, and (2) using variable length N-grams to mitigate the effect of the skewness of the frequency distribution of unigrams. The distinctiveness between two itemsets is based on a parameter, κ, which controls tolerance to redundancy.

13:00	0, 1, de, jong	De Jong scores for Ajax
	geordie, shore	Season 5 of the TV series starts
	get, out, and, vote	Still early in the U.S. elections day
17:00	if, obama, wins	Speculations regarding elections
	USERNAME, spots, my, club	Pyramidal marketing scam. Retweeted by people to make money.
	the, polls, close	Polls to start closing at 6 PM
19:00	A partir de que idade você considera alguém velho?	Internet meme from Brazil, discussing when to start considering a person old.
	food, stamps	Discussions pertaining to elections
	linda, mcmahon, senate	Linda McMahon loses CT senate race
20:00	obama, got, this	Announcing states that Obama got
	projected, winner	Early projections about who will win
	moving, to, canada	Reaction to projections
21:00	elizabeth, warren	MA senate elections winner
	popular, vote	Comparing Popular vs Electoral votes
	who, is, the, president?	Anticipation for the elections results
22:00	#forward, #obama2012	Obama won, time to move #forward
	my, president, is, still	Some were saying Black (skin colour), others Blue (party colour)
	back, in, office	Obama is back in office
22:30	once, you, go, black	A popular cultural reference.
	@realdonaldtrump, this, elections, ... [his famous tweet]	Donald Trump is a Republican and he did not accept Obama's victory. This cluster merges 15 *distinct* itemsets.
	concession, speech, write	The losing candidate has to concede before the winner declares victory

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

Table 1.3: Top 3 itemsets for hours in November 9th

Another important contribution is a method for ranking itemsets based on their temporal novelty. The top 3 itemsets from the hours of election day and another less eventful day shows that the synopses captures important events, and might reasonably be directly presented to users as a summary. A possible future direction is to use itemsets that appear as a sequence for building extractive coherent summaries of the social media stream at different times.

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

Nov 6th, 22:00	@realdonaldtrump,this,elections,...[266035509162303492]
	we, re, all, in, this, together, ..., bo [26603109979131904]
	@realdonaldtrump,our,country,is,...[266037143628038144]
Nov 6th, 23:00	URL,and,autotweet,checked,followed,me,today,unfollowed
	house,of,representatives,shouldnt,...[266040877552656385]
	URL,#android,#androidgames,#gameinsight
Nov 6th, 23:30	URL,and,autotweet,checked,followed,me,today,unfollowed
	URL,#android,#androidgames,#gameinsight
	@barackobama,@ryanseacrest, what, was, your, first, words, in, reaction, to, re, election, 2
Nov 6th, 00:00	URL,and,autotweet,checked,followed,me,today,unfollowed
	the, best, is, yet, to, come
	URL,#android,#androidgames,#gameinsight
Nov 9th, 15:00	#emawinbieber, i, think, justin, bieber, ... [see table ??]
	@boyquotations,do,everyone,follow,followers,gain,more
	#emawinkaty, i, think, katy, perry, ... [see table ??]

Table 1.4: Top 3 picked by the MTV algorithm