Efficient Temporal Synopsis of Social Media Streams

by

Younes Abouelnagah

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

Waterloo, Ontario, Canada, 2013

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

Search and summarization of streaming social media, such as Twitter, requires the ongoing analysis of large volumes of data with dynamically changing characteristics. Tweets are short and repetitious – lacking context and structure – making it difficult to generate a coherent synopsis of events within a given time period. Although some established algorithms for frequent itemset analysis might provide an efficient foundation for synopsis generation, the unmodified application of standard methods produces a complex mass of rules, dominated by common language constructs and many trivial variations on topically related results. Moreover, these results are not necessarily specific to events within the time period of interest. To address these problems, we build upon the Linear time Closed itemset Mining (LCM) algorithm, which is particularly suited to the large and sparse vocabulary of tweets. LCM generates only closed itemsets, providing an immediate reduction in the number of trivial results. To reduce the impact of function words and common language constructs, we apply a filtering step that preserves these terms only when they may form part of a relevant collocation. To further reduce trivial results, we propose a novel strengthening of the closure condition of LCM to retain only those results that exceed a threshold of distinctiveness. Finally, we perform temporal ranking, based on information gain, to identify results that are particularly relevant to the time period of interest. We evaluate our work over a collection of tweets gathered in late 2012, exploring the efficiency and filtering characteristic of each processing step, both individually and collectively. Based on our experience, the resulting synopses from various time periods provide understandable and meaningful pictures of events within those periods, with potential application to tasks such as temporal summarization and query expansion for search.

Acknowledgements

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

Dedication

This is dedicated to the my people struggling for freedom and democracy in the Arab world, while I comfortably study in beautiful Canada.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Social Media Text

Text posted on social media outlets, such as Twitter, is a good and timely source of information about real world events [71]. It also includes posts about topics attracting the collective attention of user communities, such as Internet memes or large scale discussions [59]. Nevertheless, the collective stream from all users is overwhelmed with personal updates and non-informative chatter [38, 35]. Furthermore, the attention span given to the majority of topics is quite short [44]. Timely finding posts about topics of interest requires an efficient mining algorithm.

To realize the benefits of social media in giving a voice to ordinary people, the algorithm should mine the content of posts without explicitly assigning weights based on merits of users. However, the nature of text in social media poses a challenge when applying traditional text mining algorithms. Text in social media is usually short, undermining the effectiveness of within document frequency counting. It lacks context, since posts are standalone and most platforms don't allow explicit linkage. It also lacks structure and other useful formatting cues. One type of mining algorithms that can be applied to such data is frequent itemset mining.

1.2 Frequent Itemset Mining

Frequent itemset mining algorithms count the number of times each combination of "items" appear together. This is called the support of the itemset. An itemset is considered frequent if its support is above a threshold. To mine textual data, an item can be a token and a set of items can be considered as appearing together if they appear within the same document or paragraph. The frequent itemset mining family of algorithms is fast and efficient, however it is not readily suited for application on text. Following are a number of challenges faced when applying frequent itemset mining to textual data. In this thesis we address those problems, adapting frequent itemset mining to social media text without degrading its efficiency:

1. The number of itemsets mined grows with the number of distinct items – which is particularly high in the text domain. In social media, the problem is complicated by the continuous creation of new tokens [52], in the form of hashtags or usernames and due to shortening of words because of the length limit of some platforms.

2. The number of frequent itemsets is generally high. 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 rules associating combinations of items with each other. To reduce the number of itemsets, they may be limited by setting a high support threshold. This is not possible in text mining because frequencies of most items is low, and only a few items have high frequencies; that is, frequencies of items follow a long-tailed Zipfean distribution.

3. Function words are among the few items having high frequencies. This leads to mining itemsets that are uninformative language constructs. 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. Notice that we do not remove function words to allow mining itemsets made up solely of function words, and to make the mining results easier to understand at the user level.

4. There is considerable redundancy in frequent itemsets caused by trivial differences in the language used.

1.3 Motivation and Contributions

Frequent itemset mining is suitable for the dynamic nature of social media streams. It does not require prior knowledge of the distribution of items, nor does it require selecting a few items to monitor or a preset number of topics to mine. It is fast, efficient and scalable. It is also robust against data sparsity, and can actually exploit it for faster computation. Unlike trending topics[1] [57], 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 expansion and document expansion [20, 24]. The results from each epoch can be treated as a document, facilitating the creation of a "temporal profile" [39] of the query or 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.

Previous works have proposed methods tailored for the use of frequent itemsets (patterns) to improve search performance. Itemsets provide better semantics than keywords, and have better statistical properties than phrases [85]. By using carefully selected itemsets and filtering out "meaningless" ones, *pattern-based* approaches achieve better performance than keyword-based approaches [86]. However, the choice of "interesting" or "informative" itemsets and filtering out "meaningless" ones remains an area of active research. In Li et al. [50] itemsets are mined from paragraphs of newswire text, and are used to determine

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

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 a more recent work, an unsupervised version that relies on co-occurrence of itemsets within the same paragraph has been proposed [4].

In this thesis, we propose efficient methods for filtering out non-informative frequent itemsets, reducing redundancy in the mining results, and ranking the selection according to novelty. Our methods exploit the dynamics of social media and make use of the collaborative filtering that users naturally undergo on social media, by sharing interesting posts and participating in conversations. Our main contributions are as follows:

- We propose the use of variable length N-grams as items to avoid mining uninformative language constructs as itemsets.
- We propose conditions for selecting informative itemsets and reducing redundancy in the mining results.
- We propose a formula for ranking itemsets according to novelty.

The effectiveness of our methods is shown quantitatively, in terms of the drop in the number of itemsets at each processing step. The quality of the final outcome is verified by showing mining results from various time periods.

1.4 Thesis outline (SKIP READING.. NOT READY YET

The next three chapters provide necessary background. We start by discussing related work in chapter 2, we then explain frequent itemset mining and the algorithm on which we build our work in chapters **??** and 3.1 respectively. Chapters 3, 4 and 4.4 present our contributions for applying frequent itemset mining to social media text. We present the outline of those chpaters graphically using a frequency ordered prefix tree of the itemsets. Each path from root to a leaf in such a tree represents an itemset, where each itemset is

ordered in non-increasing order of the frequency of items. Itemsets having the same prefix share the nodes representing this prefix. This representation is typical in the literature, and it is actually a very good representation of the frequent itemset mining problem. Figure 1.1 shows conceptually how our contributions in chapters 3, 4 and 4.4 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 chapter 6, we conclude our presentation and suggest future directions.

1.5 Terminology and Notation

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*.

We now introduce the notation used in this paper:

- $W = \{w_1, w_2, \ldots, w_n\}$: The set of all items occurring within the epoch of data being mined. 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, as is common in the Information Retrieval (IR) literature.

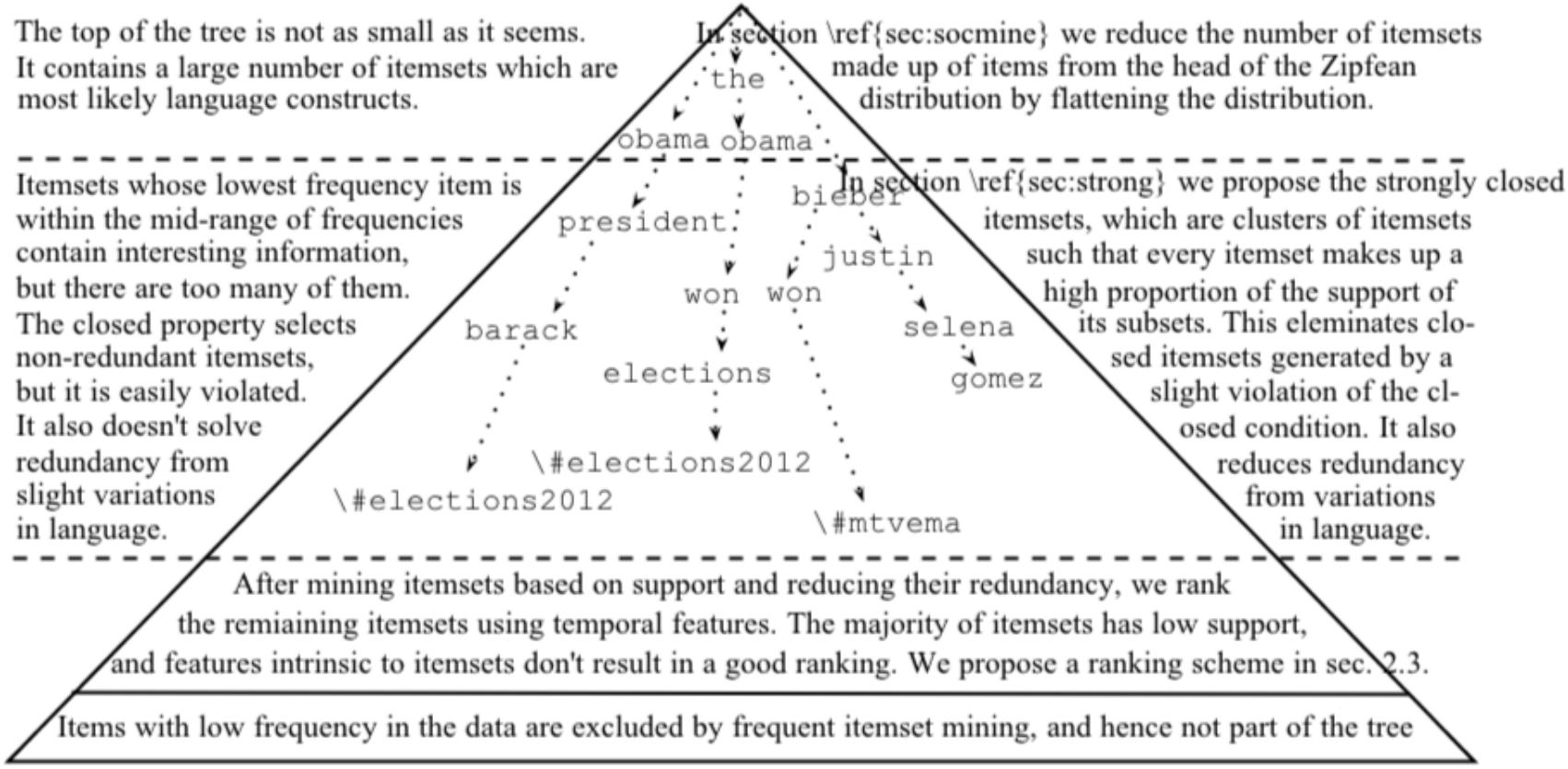


Figure 1.1: Most important contributions overlaid on a frequency ordered prefix tree

Chapter 2

Background and Related Work

2.1 Frequent Itemset Mining

Frequent itemset mining comprises a large body of work that goes back to the early 1990s [2]. We categorize frequent itemset mining algorithms into algorithms that operate in the item space, algorithms that operate in the transaction space, and algorithms that operate in the solution space. While a comprehensive survey is beyond the scope of this thesis, we discuss at least one representative from each category of algorithms.

2.1.1 The Apriori Algorithm

Agrawal et al. [3] proposed the Apriori algorithm as part of the first efficient solution to the problem if association rules mining [2]. In the domain of market basked data, an association rule is an implication that an item is likely to be purchased in the same transaction with a certain set of items. Association rules are mined by finding itemsets that co-occur together frequently, then producing a rule with each individual item as a consequent to the purchase of the rest of the itemset. The Apriori algorithm finds frequent itemsets of increasing lengthes iteratively. It starts by counting frequencies of individual items (itemsets of length 1). Then it iteratively increases the length of itemsets. At each

iteration, it generates candidate itemsets 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 anti-monotonicity between itemsets' support and length: all subsets of a frequent itemset have to be frequent. However, each candidate has to be explicitly checked to verify that it does not have any infrequent subsets.

Apriori 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 the candidate generation bottleneck. A large number of candidates can be generated, 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.

The Apriori based family of algorithms operates in the item space in the sense that an itemset can be expanded by any item regardless of whether there is any support for the combination. Operating in the transaction space avoids this blind enumeration of unsupported candidates, and the subsequent DB scan required to count the support of those candidates. This involves the creation of a succinct representation of the DB that is well suited for itemset mining.

2.1.2 The FP-Growth Algorithm

The first algorithm proposed based on the idea of avoiding candidate generation was FP-Growth [30]. The database is represented as a frequency ordered prefix tree called the FP-tree. Items with low support are removed, so that any node in the FP-tree is necessarily an item with enough support. An FP-tree imposes the invariant that within each branch the frequency is non-increasing from the root down to the leaves. This increases the chance

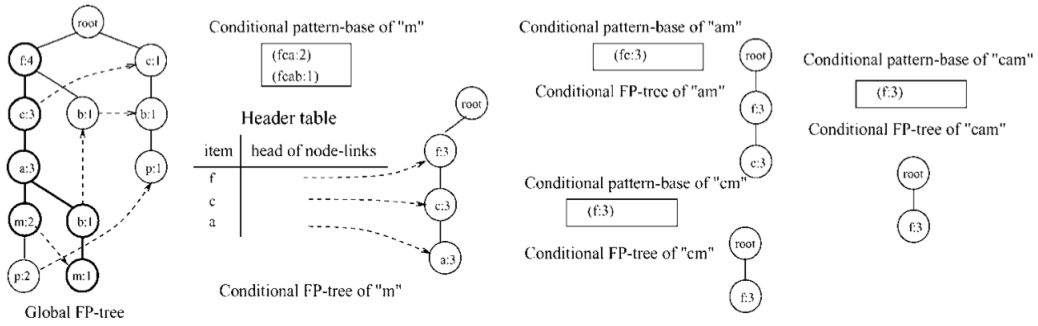


Figure 2.1: A few steps in the FP-Growth algorithm as depicted by Han et al. [31]

of finding common prefixes. An FP-tree also contains links between nodes representing the same item. Those links are used to create the *conditional* FP-tree for an item, which is the FP-tree made only from branches containing this item, excluding any items with lower frequency. The algorithm proceeds by iterating over items in increasing order of frequency. Each item is first output as an itemset, and then its *conditional* FP-Tree is created and recursively mined. At each recursive call, the current item is appended to a growing prefix, which is prepended to all itemsets mined from the conditional FP-tree. Figure 2.1, borrowed from Han et. al [31], shows a few steps in the FP-Growth algorithm.

While the navigation of a representation of the database directly derives itemsets from the transaction space, the creation of such representation is a bottleneck itself. Building conditional pattern bases and sub-FP-trees becomes very time and space consuming as the recursion goes deep and the number of patterns goes large [83].

2.1.3 Solution Space Algorithms

The last category of frequent itemset mining algorithms comprises algorithms that operate in the solution space. It can be argued that algorithms that traverse a representation of the DB are operating in the solution space rather than the transaction space – given that they avoid generating candidates with low support. However, in our categorization we focus on the logic behind how itemsets are generated or pruned rather than the data

structure used to generate them. After all, any algorithm must calculate the support of itemsets, and this needs either maintaining a representation of the DB or scanning the transactions. Algorithms that operate in the solution space prune itemsets that do not possess certain properties. The limited number of itemsets that possess the desired property are sufficient to deduce the rest of the solution (other frequent itemsets and their support). The reduction in the number of itemsets mined improves the performance of such algorithms, even if they rely on candidate generation to enumerate possible itemsets before checking for the desired property. It is not always necessary to produce the full solution since the properties usually filter out only redundant itemsets, and in many cases the elect itemsets can be used directly.

The *closure property* [64, 65, 94] prunes an itemset if it has the same support as its supersets. It is easy to see that all itemsets and their support can be derived from the set of *closed* itemsets. A formal proof is given by Pasquier et al. [64]. The smallest itemset of an equivalence class of itemsets having the same support is called a *generator* [43] or a *free set* [12]. It is also possible to keep only the generator of each equivalence class, but in this case some infrequent itemsets has to be kept in order to be able to calculate the support of all frequent itemsets [43]. A different selection of itemsets can be made according to the Non-Derivable property [17, 18], which is linked to the closed property.

Finding closed itemsets can be done by arranging possible itemsets into a Galois lattice, then traversing the lattice using the Galois connection between itemsets and transactions in which they appear. Figure 2.2 shows an example of a Galois lattice. The Galois connection is a pair of operators, one to map an itemset to transactions in which it appears, and another to map a set of transactions to the largest itemset that appears in all of them. Applying the two operators in cascade grows an itemset directly to its closed superset. More details can be found in Pasquier et al. [65] and Zaki et al. [94].

Some algorithms take advantage of the closure property to reduce the search space more dramatically, such as CLOSET [66] and LCM [79]. Our work is based on LCM (which stands for Linear-time Closed itemset Mining) [79]. 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 [40], which was the workshop's award winner. This algorithm is robust against data sparsity, and has low memory requirements since

TID	Items
1	A B C
2	B C E
3	A B C E
4	B E
5	A B C E

The transaction database

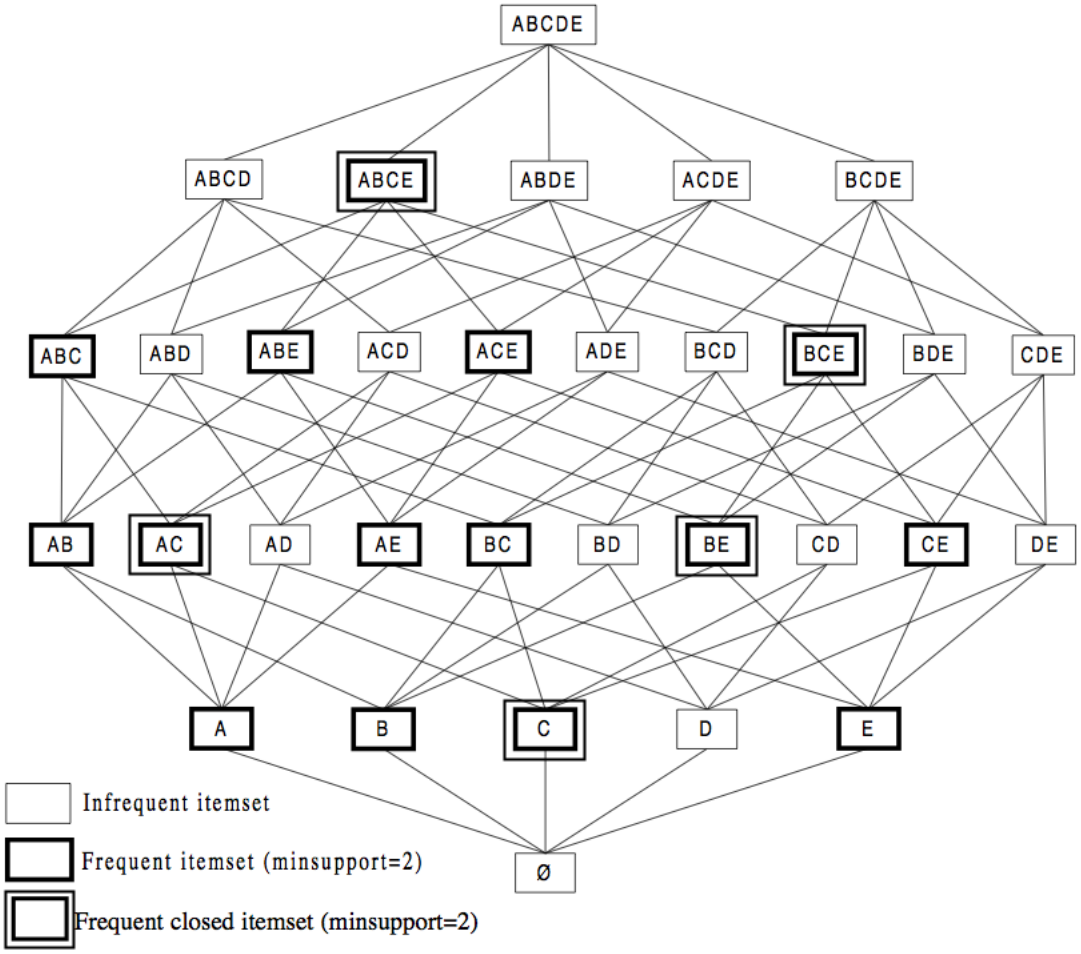


Figure 2.2: Example of a Galois lattice adapted from Pasquier et al. [65]

it does not store intermediate results. We will explain the closure property and describe LCM in detail in section 3.1. We proceed by discussing other areas of related work.

2.2 Election of Representative or Interesting Itemsets

The problem of having many itemsets, with redundancy and noise, can be addressed by picking out representative itemsets that satisfy a certain condition. The representative itemsets are not necessarily sufficient to deduce the rest of the solution (itemsets and their support information). In this thesis we propose two conditions for selecting interesting itemsets; *distinct* itemsets and *strongly closed* itemsets. Related approaches to picking out representative itemsets are choosing a set that will minimize the reconstruction error of the full solution, and limiting the number of itemsets to a user specified number.

The δ-free [13], δ-covered [88], and *master* itemsets [90] are examples of conditions motivated by providing a compressed representation of itemsets through sacrificing support information. The δ-free condition selects an itemsets if its support is different from the support of all its subsets at least by δ. This selection of itemsets can be used to approximate the support of other frequent itemsets with a guarantee on the error, but only if some infrequent itemsets are included in the selection. The δ-free condition implies setting an upper bound on the strength of association rules that can be derived from selected itemsets. It is suitable for compressing mining results from dense datasets, but it is exactly the opposite of *distinct* itemsets we propose in this thesis.

The δ-covered condition is also the opposite of the *distinct* condition, as it "relaxes the closure condition to further reduce pattern set size" [54]. On the other hand, the *distinct* condition strengthens the closure condition. The δ-covered condition sets an upper bound on the confidence of the rule that an itemset implies a superset; if an itemset implies its superset with confidence greater than δ, then it is considered to be covered by the superset and can be pruned. On the other hand, the *distinct* condition sets a lower bound on the confidence of a derived association rule. Nevertheless, the δ-clusters are similar to the *strongly closed itemset* clusters proposed in this thesis. The efficient version of the clustering algorithm proposed by Xin et al. [88], RPLocal, limits the search space for

clustering candidates using a similar reasoning to the reasoning used in our algorithm – exploiting the sequence in which frequent itemsets are found. The difference between δ-clusters and *strongly closed itemset* clusters is that δ-clusters are soft clusters, while *strongly closed itemset* clusters are hard clusters. Also, the distance measure used in δ-clusters is the Jaccard distance, while the *strongly closed itemset* clusters are based on association rule confidence.

Yan et al. [90] proposed a method to choose K *master* itemsets as representatives of the itemsets. However, rather than leaving K as a user specified parameter a method is proposed for choosing K by minimizing the restoration error of support information. 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 all itemsets. On the other hand, *strongly closed itemsets* actually avoid clustering together itemsets representing different topics or different opinions within a topic. Furthermore, Yan et al. improve performance by approximating the calculation of the distance between itemsets, to avoid accessing the data. We can achieve high performance without approximation because our mining results include the postings list of each itemset.

Another approach to choosing itemsets is according to their interestingness. The interestingness of itemsets (patterns) has been an area of active research for a long time. According to Silberschatz and Tuzhilin [73] interestingness of an itemset can be *objective* or *subjective*. Commonly used *objective* measures include (a more extensive coverage is provided by Geng et al. [26] and Tan et al. [75]):

1. *Support*: The number (or fraction) of transactions that contain the itemset

2. *Confidence*: The confidence of a rule derived from the itemset. A rule is derived from the itemset by treating one item, w, as the consequent of the rule while the rest of items, $s \setminus w$, are the antecedent. The confidence of the rule is then defined as:

$$conf(s \setminus w \rightarrow s) = \frac{|T_s|}{|T_{s \setminus w}|} \tag{2.1}$$

   Since different rules with varying confidence can be derived from an itemset, either the minimum (*all-confidence*) or the maximum (*any-confidence*) is used [62].

3. *Lift* [16]: Probability (support) of the itemset over the product of the probabilities of all items in the itemset. This is a measure of dependence and can be linked to Pointwise Mutual Information [14]. Such a measure is suitable for Knowledge Discovery or Exploratory Search tasks, where the goal is to extract correlated item sets or significant phrases.

4. *Bond (or Coherence)* [62]: The support of an itemset over the number of transactions containing *any* of its items.

An important property of a measure, upon which all Apriori based mining algorithm are built, is the downward closure property; the value of a downward closed measure cannot increase as the size of the itemset increases. All the above measures are downward closed [9] with the exception of *lift* which is upward closed (cannot decrease with the increase of the size of the itemset), and *any-confidence* which is neither upward nor downward closed. Algorithms that mine itemsets based on the above measures were proposed [46, 41], but if an application specific measure is to be used then it has to be evaluated in a post mining stage or within the Redundancy-Aware Top-K framework [87].

The Redundancy-Aware Top-K framework [87] selects the top K itemsets according to an interestingness criterion. The major difference between their approach and all of the previous works is that it emphasizes both interestingness and redundancy on the selected top K itemsets, and the itemset interestingness is defined in the context of the application; summarizing the whole collection of itemsets is not the goal. Our work fits in the proposed framework: we reduce redundancy through filtering and clustering, and we propose a ranking that orders itemsets according to some definition of interestingness. If only the top K itemsets are desired, itemsets ranked at positions higher than K can be omitted. An explicit number of itemsets needs not be set, however. This makes our computational model more efficient, since we do not need to solve a constrained optimization problem. In their experiments, Xin et al. [87] used two application specific interestingness measures: they used one that is based on TF-IDF for extracting interesting itemsets from a text corpus, and another that is tailored for predicting block fetches in storage systems [51]. We rank itemsets according to their novelty using a measure based on Information Gain. Preliminary experimentation with various other measures of interestingness ruled them out

as ineffective for our purposes.

Subjective measures of interestingness select itemsets that are actionable or unexpected [74]. Selecting unexpected itemsets entails building a probabilistic model to use it for estimating itemsets probabilities. Jaroszewicz and Simovici [37] use a Baysian Network as a model, which is suitable for cases where items are value levels of a limited number of attributes. A commonly used model is the maximum entropy model [81, 76, 56]. The Maximally informaTiVe itemsets (MTV) algorithm, proposed by Mampaey et al. [56], uses a greedy algorithm to select K itemsets whose probabilities diverge the most from the estimate given by a maximum entropy model. The maximum entropy model is initialized as a uniform distribution. After selecting each of the K itemsets it is updated to fit the selected itemsets actual frequencies using the Iterative Scaling procedure [23]. If K is set to infinity, the algorithm will generate the best model according to the Bayesian Information Criterion (BIC). Since BIC incorporates a penalty term for the number of parameters (itemsets included in the model), this selection will minimize redundancy while maximizing surprise (divergence from model). The model can be initialized using itemset frequencies known from prior knowledge. It is also possible to use the Minimum Description Length (MDL) as a criterion for selecting itemsets. We compare our selection of itemsets to ones obtained from MTV using the implementation provided by the authors[1]. However, the value of K has to be kept low (at most 50) for the algorithm to finish without an error, even though it is running without a limit on its resource usage on a machine with 256 GB of RAM. In our work we focus on efficiency, allowing the summary to include larger numbers of itemsets than the values of K for which MTV is efficient (at most 10). This is crucial for scalability to the volumes of data typical in social media streams.

Another approach to picking out itemsets is choosing ones that can be used to compress the data. The KRIMP algorithm [80] 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.

[1] http://www.adrem.ua.ac.be/implementations

2.3 Frequent Itemsets in Search and Summarization

Different ways to use Frequent Itemset Mining in search has been proposed. Possas et al. [69] proposed using closed itemsets in place of terms, using them in exactly the same way as terms are used. In the tokenization process, they use CHARM [94] to mine closed itemsets from frequent tokens in the document. They then add the document to the postings lists of each closed itemset present in the document. Queries are tokenized similarly and results are ranked using TF-IDF. This simple approach achieves increased precision on various test collections, possibly because frequent itemsets take into account collocation information. The approach is so simplistic, however. The overlap in closed itemsets was not taken into account while tokenizing documents and queries into itemsets. Also, the mining algorithm was not modified to accommodate the high number of terms and the greater length of documents compared to the length of transactions typical to market basket data. Finally, the number of itemsets is higher than the number of terms so the index size will increase.

Wu et al. [86, 85] propose the use of sequential closed patterns instead of terms, in what they call *pattern based approach* to information retrieval. They consider patterns as surrogates to phrases that have better statistical properties than phrases, thus achieving a break through in the performance of *phrase based approaches*. "Although phrases contain less ambiguous and narrower meanings than individual words, the likely reasons for the discouraging performance [49] from the use of phrases are: (1) phrases have inferior statistical properties to words, (2) they have low frequency of occurrence, and (3) there are a large number of redundant and noisy phrases among them." [85] They address the problem of overlap in closed patterns, and they developed an algorithm specifically tailored for mining closed sequential patterns from text. Documents are broken into paragraphs which are the unit of mining and retrieval. This alleviates the problem of having documents that are much longer than traditional market basket transactions. Albathan et al. [4] extend the work of Wu et al. by proposing a method for further reducing the number of closed patterns. Lau et al. [45] used related methods for term weighting in pseudo-relevance feedback for Twitter search, and achieved substantial improvements over a baseline. In an earlier work [1], we have also explored the viability of using frequent itemsets for query

expansion in microblog search. We could achieve improvements over a strong baseline on the TREC 2011 microblog track queries, using frequent itemsets mined by an unmodified mining algorithm that returns the top-k itemsets according to their support. This work can provide such methods by a short ranked list of itemsets.

Our work complements the work done by Yang et al. [91] for using frequent itemsets as a temporal summary. They have proposed a framework for storing mining results of temporally consecutive intervals (batches of data) using a pyramidal time window. Their framework allows for fast execution of temporal queries, and for tracking the evolution of topics. However, the itemsets stored for each interval is not selected for providing the best summary. Itemsets are selected according to their utility in a lossy compression algorithm. The algorithm is based on representing transactions as itemsets that cover them, possibly introducing items that are not originally in the transaction or omitting items from the transaction. The actual transactions can be reconstructed with an accuracy that is expected to be high if the transaction contains "both recent and frequent keywords". The temporal summary is created from the reconstructed transactions in a separate step. First, the pyramidal time window is queried using keywords and a specific time intervals (for example, "world cup" on the day before a certain matche). Then, non-negative matrix factorization is used to extract topics from the returned transactions. The pyramidal time window structure can be used to store itemsets selected by our methods, thus directly providing a temporal summary without the need for the matrix factorization step. Our methods exploit 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 source of real-time information, as we will discuss in the next section.

2.4 Social Media as a Source of Information

The problem of finding information in social media posts is motivated by the uniqueness and variety of information that can be found in these media. Some stories are reported mainly on social media because of censorship on journalism, such as the 2009 - 10 Iranian

election protests (nicknamed "Iran's Twitter Revolution" [2]). Other stories are reported mainly on social media because of the content creator happens to be at the right place at the right time. This could be by chance like in the case of the emergency landing of a US Airways plane on the Hudson river in New York [3]. This could also be an intentional act of citizen journalism. Citizen journalists are "the people formerly known as the audience," who "were on the receiving end of a media system that ran one way, in a broadcasting pattern, with high entry fees and a few firms competing to speak very loudly while the rest of the population listened in isolation from one another – and who today are not in a situation like that at all. " [4].

In a study of the use of Twitter as a news medium during the 2011 Tunisia and Egypt revolutions, Lotan et al. [55] categorized 963 accounts that made influential posts according to the *type of actor*. A post is considered influential if it is among the top 10% of posts in terms of the number of near duplicates made out of it; posts that had at least 16 near duplicates in the Tunisia data, or at least 19 near duplicates in the Egypt data. The types of actors included Mainstream Media organizations and Mainstream New Media organizations, which are news and media organizations with and without offline presence, respectively. It also included Journalists, who are individuals employed by media organizations or who regularly work as freelancers for them.

In both datasets, media organizations make up about 11% only of these accounts (approximately 7% for ones with offline presence and 4% for ones that exist solely online). Journalists make up another 15%, so the total of mainstream media affiliated accounts is approximately 26%. On the other hand, more than half of the accounts from which influential tweets originated belong to individuals not affiliated to the government or mainstream media (55% for Tunisia and 56% for Egypt). Out of those individual, almost half are ordinary people who don't self identify and cannot be identified by assessors as bloggers or activists.

The rate at which an account makes posts is approximately 16 per hour for main-

[2] http://www.theatlantic.com/technology/archive/2010/06/evaluating-irans-twitter-revolution/58337/

[3] http://www.telegraph.co.uk/technology/twitter/4269765/New-York-plane-crash-Twitter-breaks-the-news-again.html

[4] http://archive.pressthink.org/2006/06/27/ppl_frmr.html#more

stream media affiliated accounts, and approximately 10 for individuals accounts (thus a mainstream media account makes 60% more posts). Also the probability that a post will be echoed is 0.88 for mainstream media accounts, 0.55 for journalists, 0.4 for bloggers and activists, and 0.31 for individuals (this is the probability that any follower will echo the post, the average number of followers of all types of actors is much higher than that of ordinary people). However, collectively the fraction of highly echoed threads originating from ordinary people is on a par with the fraction of threads originating from a journalist. Threads originating from ordinary people, bloggers and activists together make up more than half of highly echoed threads, followed by threads originating from journalists (around 20%), then threads originating mainstream media (around 10%, divided equally between media with and without offline presence).

All this emphasizes the influence citizen journalists and ordinary people now have on the information creation and dissemination process. User created content is rich with unique content tackling a variety of topics from different points of view. However, it is dominated by non-informative chatter and personal updates. In a study comparing the use of Twitter's search feature to Web search, Teevan et al. [77] found that Twitter is mostly used for getting timely information: updates about events, news about topics of interest (such as technology news), and realtime local updates (such as traffic jams). It was noted that Twitter search could be the main way of accessing Twitter, since posts from the followed network could to be outside of the user's topical interests. This is supported by findings from an earlier study by Naaman et al. [60] in which 3379 tweets from 350 accounts belonging to individuals were studied. The tweets were hand coded into different categories, then the activity of users in terms of what type of tweets they make was analyzed. It was found that 40% of the studied tweets are personal updates (about "Me now"), and that users can be clustered into a big cluster of "Meformers" (80% of the users, half of their posts are personal updates) and a small cluster of informers (20% of the users, half of their posts are for sharing information). The study also found that most people engage in some form of Meformer activity; on average, users had 41% of their messages in the "Me now" category. Therefore filtering out posts by Meformers, or giving higher weights to posts by Informers will not be enough for finding informative posts made by ordinary people.

To tap into social media as a source of information, social media users collaboratively

select good content. On Twitter, users can *retweet* an interesting post, which means forwarding it to their followers. The dynamics of retweeting was studied by Boyd et al [15]. One of the findings was that posts being retweeted are frequently modified to accommodate additional text, or at least the notation meant to indicate that the message is a retweet. This is because Twitter limits tweets to 140 characters. The selection of what to keep from the post is actually a form of collaborative filtering itself, because the most interesting parts has to be retained. The parts that are selected by many people is likely to be a good summary of what is interesting. The words in this part would make up a frequent itemset. Other methods for finding emerging topics in social media are discussed in the next section.

2.5 Topic Detection and Tracking for Social Media

The problem of Topic Detection and Tracking (TDT) was introduced by Allan et al. in 1998 [6]. Topic Detection is the discovery of previously unknown topics, where a topic was first defined as an "event" then its definition was broadened to include the triggering event as well as other events and activities directly related to it. Our work can be categorized as a method for topic detection from social media, or more specifically "event" detection since we make no effort to group events into topics. Ever since it was introduced, TDT has been an area of active research. Besides the fact that there is always room for improvement, TDT was originally targeting the domain of newswire data, and the solutions developed for this domain are not directly applicable to data from other domains.

In the domain of social media, Petrović et al. [67] used an adaptation for Locality Sensitive Hashing (LSH) [36] to perform first story detection at scale. The computation framework is the same as the one used in traditional systems [92, 7]. For each tweet find the nearest neighbour in the vector space of terms. If the distance is less than a threshold then add the document to the topic represented by the cluster containing the nearest neighbour. Otherwise, create a new cluster representing a new topic. The current document is the first story of the topic. The adaptation to LSH that they propose reduces the number of times a document is considered far from all its neighbours while it actually is not. They

do that by comparing the document to the last 1000 tweets, and update the minimum distance if necessary. Using LSH, with the adaptation, didn't affect the quality of results much as compared to the UMass FSD system [7] on a traditional TDT task (TDT-5). On the other hand, the run time was decreased from 28 hours to 2 hours. The UMass system could not finish processing a dataset of a 160 million tweets collected over a period of 6 months from Twitter, while their proposed system can process each tweet in bounded space and time. One problem remains, however. The precision of determining if a tweet cluster is actually pertaining to an event or just normal chatter or spam is low. Actually their system can detect spam clusters with high precision ($average\,precision = 0.963$), but not events ($average\,precision = 0.34$). The problem of identifying event clusters was further studied by Beker et al. [10], where they used a Support Vector Machine (SVM) classifier and achieved a higher precision. However, their test set was limited to 5 hours of Twitter data and the precision decreases as the number of clusters increases because the classifier is trained on the positive case.

Another method of event detection follows the framework of bursty keywords detection proposed by Kleinberg [42]. In this framework, a set of keywords are tracked – possibly all tokens in the dataset. A state machine is used to indicate the stat of each keyword, where consequitive states correspond to higher levels of *burst* intensity. Each transition between states is associated with a cost, and a keyword moves between the states to minimize the overall cost of its transitions. A simplified version of this framework is use by Parikh and Karlapalem [63], where there are two states for each token. A token is moved to the state of being bursty if the increase in its frequency exceeds a threshold, and it stays in this state for a fixed time interval. The bursty tokens are then clustered into events. Another method that tracks the frequency of all tokens is EDCoW [84], where the wavelet transformation is used to convert counts of occurrences from the time domain to the frequency domain then auto-correlation is used to detect burstiness. However, the use wavelet transformation is not justified specially for the enormous number of tokens in case of social media. Other methods based on burst detection track a small set of keywords. Lehmann et al. [47] track hashtags and study their usage pattern. Chakrabarti and Punera [19] track tokens appearing in tweets tagged by an American Football team name, and use them to create a summary of the game.

Our work can be seen in the light of Meme-tracking [48], where a particularly interesting quotation (meme) is tracked across news and blog posts. The quotation being tracked is not quoted verbatim in all posts, but rather parts of it are selected and it could be slightly paraphrased. This is similar to modifications made during retweeting an interesting tweet. Leskovec et al. [48] tracked quotations made by American politicians during the 2008 U.S. presidential elections. Phrases made up of 4 words or more are tracked if they got repeated 10 times or more, where at most 25% of the repetitions could be on the same domain name. A directed acyclic graph is constructed where phrases are vertices and edges are weighted according to the edit distance between phrases, as well as the number of occurrences of the longer phrase. The graph is then divided into clusters where each cluster contains one *root* that has no outgoing edges. Each vertex is assigned to the cluster with which it has the highest number of edges. Clusters are then ranked according to the number of news and blog posts containing any phrase from the cluster. Leskovec et al. then use the clusters and their associated news and blog posts to perform an analysis of the temporal variation of the volume of posts about each meme. They propose a model that fits the data well, and study the difference between blog posts and news. In our work, we are also using the volume of posts containing phrases and their variations (itemsets). We also do clustering but in a different way, and we work in a totally different scale of time.

O'Connor et al. [61] also start from phrases to summarize tweets about a user specified query. They select *significant* phrases which are statistically unlikely, then cluster them. Frequent itemsets are a better starting point than phrases, since there are much more phrases than frequent itemsets.

Event detection in Twitter can also be done using Natural Language Processing to identify *event* parts of speech [71], or to identify action phrases [68]. Of course, it is also possible to apply topic modelling by Latent Dirichlet Allocation (LDA) [33] or an online varient [32, 93]. Our method tackles the problem in a totally different way.

Chapter 3

Mining Social Media Text

3.1 The Linear-Time Closed Itemset Mining Algorithm

In section 2.1.3 we have discussed frequent itemset mining algorithms that overcome the bottleneck of *candidate generation* by limiting the solution to itemsets with a certain property, reducing the size of the solution and possibly providing a way to quickly navigate the solution space. In this thesis, we expand on the Linear-time Closed itemset Mining (LCM) algorithm [79], which mines only *closed itemsets* [64]. The LCM algorithm is distinctively fast because it also takes hints from the transaction space during candidate generation. This also makes it resilient against data sparsity, since it will not consider a candidate that never appears in the data. The ability to efficiently mine sparse data makes LCM particularly suitable for mining social media text. In this section we explain in detail how LCM works.

Since LCM makes use of the properties of closed itemsets, we begin our presentation by discussing these properties. Informally, 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 3.1:

$$\mathcal{C} = \{s_c : \ s_c \subset W \, and \nexists \, s_d \, where \, s_c \subset s_d \, and \, |T_{s_c}| = |T_{s_d}|\} \tag{3.1}$$

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. [79].

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 which have to be removed to get an itemset with greater support. Notice that they are necessarily the last items added to the itemsets, 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 must be extended by all items that occur in $T_{itemset}$, except items which are *smaller* than its *suffix head*; extending by *smaller* items will lead to closed itemsets already generated in an earlier step. Extension items are added to the itemset

in the order given by the total ordering, recursively applying PPC-Extension to the extended itemset before adding the next extension item.

2. After adding an extension item, w_e, to an itemset, s, we add all other items that appear in all transaction containing $s \cup \{w_e\}$ – that is, items whose frequency within $T_{s \cup \{w_e\}}$ is equal to $|T_{s \cup \{w_e\}}|$. The newly added items become the *suffix*.

3. If all items in the *suffix* are *larger* than the *suffix head* then add the itemset to the solution. Otherwise, prune this solution branch; all closed itemsets within this branch have already been generated, when processing the smallest suffix member.

Table 3.1 is an example of how *PPC-Extentsion* is used to generate closed itemsets starting from the *1*-itemset {'barack'}. The upper table enumerates $T_{\{barack\}}$. The lower table shows steps of itemsets generation. The current itemset along with its frequency is in column 2. Itemsets marked by an (*) are the closed itemsets that are part of the solution. The suffix of the itemset is shown in *italic*. All possible extension items and their frequencies are in column 3. Extension items that are *smaller* than the *suffix head* are shown with a line striked through them. For each itemset, the extension items are kept so that it is known which extension item is next in turn to be added. An item is bolded when it is being added because its turn has come. After adding each item, a pass is done on $T_{itemset}$ to enumerate and count possible extension items. To enforce a support threshold infrequent extension items would be removed after counting, but in this example there is no such threshold. Finally, column 4 is a comment explaining each step.

Table 3.1 shows 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.[79] 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 3. 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 Itemset	Possible Extension Items	Comments
1	{barack} (4)	mitt (2), obama (3), romney (2)	Items in $T_{\{`barack'\}}$ are counted. No items should be added to the itemset.
2	{*barack*} (4)*	mitt (2), obama (3), romney (2)	Rule 3: suffix is ordered, add itemset to solution. This is not shown as a separate step in the rest of the example.
3	{*barack*} (4)	**mitt** (2), obama (3), romney (2)	Items are ordered lexicographically. Adding 'mitt' to itemset.
4	{barack, *mitt*} (2)*	**obama** (1), romney (1)	Extension items reenumerated & counted.
5	{barack, *mitt*, obama} (1)	romney (1)	Rule 2: 'romney' appears in all $T_{itemset}$.
6	{barack, mitt, *obama, romney*}(1)*		Rule 3: 'obama' is the *suffix head*.
7	{*barack*} (4)	mitt (2), **obama** (3), romney (2)	Nothing more to add, back to {'barack'}.
8	{barack, *obama*} (3)*	~~mitt~~ (1), **romney** (2)	Rule 1: skipping 'mitt', adding 'romney'
9	{barack, obama, *romney*} (2)*	~~mitt~~ (1)	Rule 1: Nothing more to add.
10	{*barack*} (4)	mitt (2), obama (3), **romney** (2)	Back to {'barack'}, adding 'romney'.
11	{*barack*, romney} (2)	mitt (1), obama (2)	Rule 2: add 'obama' after 'romney'.
12	{barack, *romney, obama*} (2)	~~mitt~~ (1)	Rule 3: suffix is not ordered, prune.
13	{*barack*} (4)	mitt (2), obama (3), romney (2)	Back to {'barack'}, all possible extension items were added. Done.

Closed itemsets containing 'barack'

Table 3.1: Generation of closed itemsets by Prefix Preserving Closure Extension

3.1.1 Implementation Details

We show in algorithm 1 how to implement LCM and PPO-Extension using an inverted index. The algorithm assumes the presence of a search infrastructure, with the ability of performing boolean queries efficiently. Besides the high likelihood that such an infrastructure already exists, it is also a very good representation of textual data. Other representations that are designed for databases in general might not be well suited for textual data. For example, the memory requirements of the FP-tree 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. The possibility of using an inverted index in the implementation of LCM is yet another reason why it is well suited for textual data.

The algorithm takes as input an epoch of data and a support threshold. 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. In the implementation shown, it is not necessary that the inverted index's tokens list (X.tokens) follow the total ordering; all itemsets of length 1 will be considered anyway. Each 1-itemset is then expanded according to the PPO-Extension rules. Lines 11-18 are the enumeration and occurrence counting of possible extension items, except line 15 which forms the closed itemset according to PPO-Extension rule 2. Lines 19-20 prunes a solution branch according to the rule 3, and line 21 adds the itemset to the solution according to the rule's complement. Lines 22-27 extends the itemset according to rule 1, where line 23 (along with line 5) enforces the support threshold.

The algorithm lends itself to distributed implementations. For example, a map/reduce implementation is straightforward since the only operations are counting (line 15) and projection (line 24). 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. We experimented with Hadoop[1] and the overhead was in the order of minutes. Besides, many problems start to arise when the programming model is complicated, and Hadoop was particularly problematic.

[1] http://hadoop.apache.org

Input: *support*: Support threshold
Data: E: Epoch of data
Result: C: Closed itemsets occurring in at least *support* records

```
C ← {⟨∅, E⟩} ;      // ∅ is a closed itemset.  This is skipped in practice
X ← Inverted index of E;
foreach w ∈ X.tokens do
    T_{w} ← X.postingsList[w];
    if |T_{w}| ≥ support then              // Support threshold for 1-itemsets
        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 ;                    // Prune according to PPO-Extension Rule 3
    C.add(⟨s ∪ suffix, T_s⟩);
    foreach v ≻ w_sh and v ∉ suffix do
        if frequency[v] ≥ support then
            T ← T_s ∩ T_{v} ;                          // Results of query s AND v
            LCM(s ∪ suffix ∪ {v}, v, T)
        end
    end
end
```

Algorithm 1: The LCM frequent itemset mining algorithm

3.2 Using LCM for Mining Social Media Text

3.2.1 Dataset

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. We remove duplicate original tweets (not retweets) using a Bloom filter [58]. This filtering removes spam tweets sent by botnets, averaging at 2.86% of the stream. There were two disruptions during data gathering. One in late October because of Hurricane Sandy which made the Twitter service unaccessible. The other was in late December because of technical problems on the server gathering the data.

The average number of tweets per hour (*volume velocity*) in the data is 119,035.49 tweets (n = 2,546 hours, standard error = 491), and the average number of tweets per day is 2,705,931.83 tweets (n = 112 days, standard error = 55,155). These are simple moving averages calculated by summing the number of tweets in non-overlapping epochs and dividing by the number of epochs. A more accurate estimate of the average number of tweets at different hours of the day can be acquired using a timeseries model of the activity on the social network.

3.2.2 Timeseries model of the dataset

Arrivals entering a system are commonly modelled as a Poisson stochastic process. A Poisson process can be characterized by a random variable $N(t)$ representing the aggregate number of arrivals that has happened up to time t. This random variable has a Poisson probability mass function with a rate λ (as well as two other necessary properties: independent and stationary increment). When the rate is high enough (more than 1000), the Normal distribution can be used as an approximation to the Poisson distribution. Since

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

social media has arrival rates larger than 1000 posts per minute (the time unit is arbitrary but must be long enough to have more then 1000 arrivals), it is therefore possible to assume a normal distribution and use a state space model to describe the activity on the social network.

Values of a variable in time series data is known to be correlated, and specialized models are developed to deal with such explicit correlation. The explicit autocorrelation (correlation between values of the same variable) is usually uninteresting, since it is known and expected. For example, the data can be known to have a trend such as a chronic increase in the volume velocity. Also, a cyclic increases and decreases in the volume velocity according to the time of day can be expected - this is called the seasonal effect in time series analysis. State space methods provide an explicit structural framework for the decomposition of time series [22]. Unlike the more popular Box - Jenkins ARIMA models, trend and seasonal effects are not treated as nuisance parameters, and it is not necessary to remove the trend and seasonal effects from the series before the analysis can begin. Instead, a state space model has an equation to capture each type of effect that is suspected to contribute to the values of the observed variable. Different models can be used to describe the same observed variable, and the best model is selected using the Akaike information criterion which penalizes extra parameters.

After experimentation with different models, we found that the model that best describes the volume velocity of Twitter is the following:

1. A deterministic seasonal cycle that is 1 day long, which captures the effect of the hour of day on the number of posts.

2. A stochastic level component, which represents the volume velocity specific to each hour. The stochasticity is represented as a variance attached to the level component.

There was no need to add a trend component, maybe because the sampling used by the Twitter API public stream endpoint emits a stable volume of Tweets, and maybe because there was actually no significant growth in the use of Twitter during the months of data collection. In our model we use the timestamps of tweet creation, and we did not keep the timestamps of when the API returned each tweet as part of the sample.

Figures 3.1 and 3.2 show different components of the model when it is fitted to the hourly number of tweets in the month of October 2012. Figure 3.1 shows the daily cycle component (green) and the noise component (black), which must be present in all state space models. The cycle is perfectly aligned with the days in the Hawaii Standard Time (HST) time zone, and thus we always use this timezone (GMT-10) to get the date from a timestamp. Figure 3.2 shows the level component (red) along with a scatter plot of the hourly volume velocity. The blue continuous line is the simple moving average and its confidence bands are the blue dashed lines. The simple moving average is clearly not a a good estimate of the hourly volume velocity. A better estimate of the number of tweets in an upcoming hour can be predicted from the model using a Kalman filter [78].

The peaks in the level component coincide with real world events. The peaks on the 3^{rd}, 16^{th} and 22^{nd} coincide with the presidential debates[3], and the peak on the 28^{th} coincide with the emergency declaration for the North Easter states of the USA in anticipation of Hurricane Sandy[4]. The use of such a model to indicate the presence of interesting information can be investigated further, however it will probably require modelling the number of occurrences of many keywords. This is not scalable because handling correlation between different random variables increases the number of parameters of the model quadratically in the number of variables. The correlation between random variables is captured by a variance-covariance matrix, and if two variables are not known to be independent a parameter must be added in the cell of the intersection of their row and column. We will not use timeseries models any further than this exposition.

Nevertheless, it is necessary to adapt the support threshold to the dynamicity of the volume velocity at different times of the day. If a ratio support threshold is used then this would not be needed, however determining a support ratio is difficult and it is more desirable to be able to specify the threshold as an absolute number. We therefore 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 and then converted to a ratio, α, through dividing by the average of the number of tweets in the hour of minimum volume velocity in different days. The actual *support* threshold used for mining any given

[3] http://www.uspresidentialelectionnews.com/2012-debate-schedule/

[4] http://www.fema.gov/hurricane-sandy-timeline#oct28

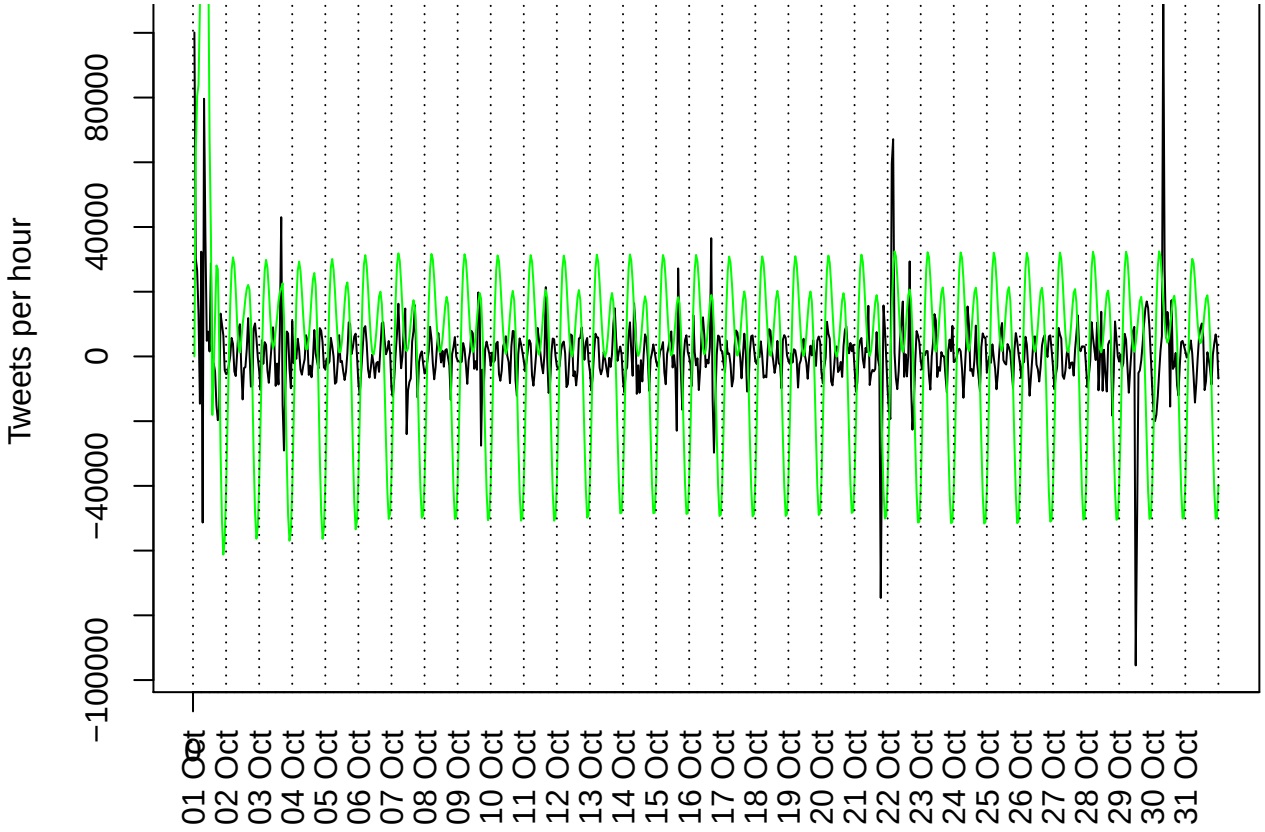


Figure 3.1: Seasonal and noise components of the volume velocity

epoch, E, is thus α multiplied by the number of tweets in the epoch:

$$support = \alpha \times |E| \qquad where\, \alpha : support\, ratio \tag{3.2}$$

$$= \frac{minimum\_support\_threshold}{\mathbf{avg}(\min_{day}(vol.\, vel.))} \times |E| \tag{3.3}$$

Calculating the support as described makes it intuitive to select a support threshold, but it would lead to a high support threshold in epochs including a burst of tweets. The high support might lead to mining only itemsets about the topic that is causing the burst. Mining a sliding window can alleviate this problem.

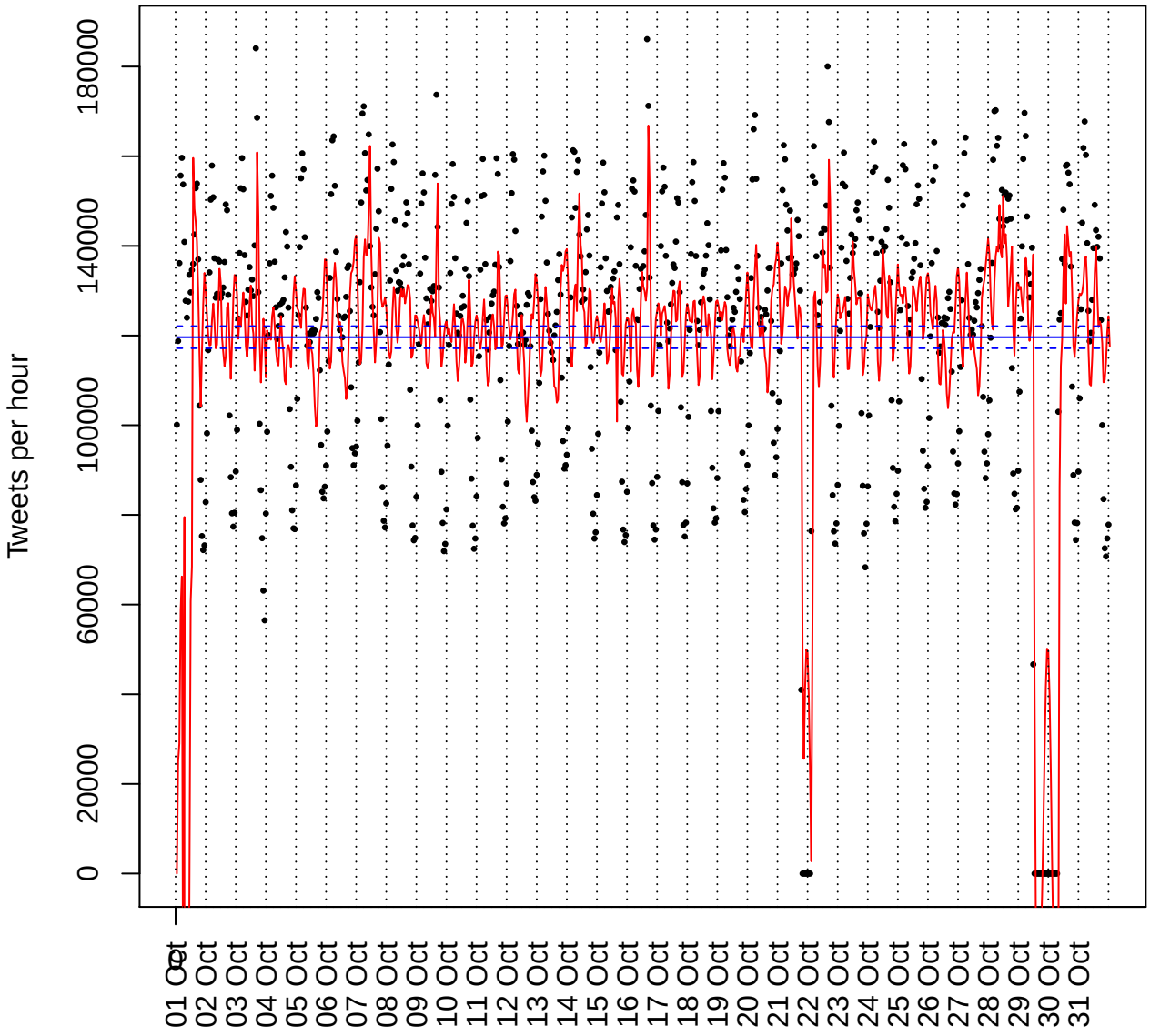


Figure 3.2: Stochastic level of the volume velocity

3.2.3 Sliding window model

One of the prominent stream processing models is the sliding window model, where a window of data *slides* forwards through the stream. The window keeps a fixed amount of history data (fixed in terms of age not size); data with a timestamp older that a certain time is removed from the window as new data is added. The data in the window is processed every time the window slides. Thus, the sliding window model sets a cut-off age after which a datapoint has no effect at all on the results of the stream processing algorithm. This is the simplest way to overcome the problem that an anomaly in the date, such as a burst of tweets about a certain topic, could affect the mining results more than recent data. More elaborate methods include dynamically changing the sliding window size [11], smoothing the data [28], and weighting data according to its age [27].

We apply the algorithms to epochs of data, similar to the *block evolution* stream mining approach [25, 96]. However, our algorithms are not strictly stream processing algorithms. Stream processing algorithms are supposed to process transactions one at a time as they arrive, looking at each transaction only once – in other words, they should make one pass on the data then discard it. This complicates the model of computation and there is no justification to process social media data as a stream. The stream processing model of computation is justified when the data is not supposed to be stored (such as data from sensor networks), or when the processing time has to be as short as possible such that looking at historic data is an overhead (such as in the case of processing stock ticker data, where making a decision a few milliseconds before/after competitors can translate into profits/losses). In the case of social media text, the data has to be stored anyway and the perception of the human users makes processing times in the order of seconds acceptable for a *real time* system.

In essence, processing epochs of data is still 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 as short as a few seconds for epochs up to a day long. Figure 3.3 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 3.3 are averages

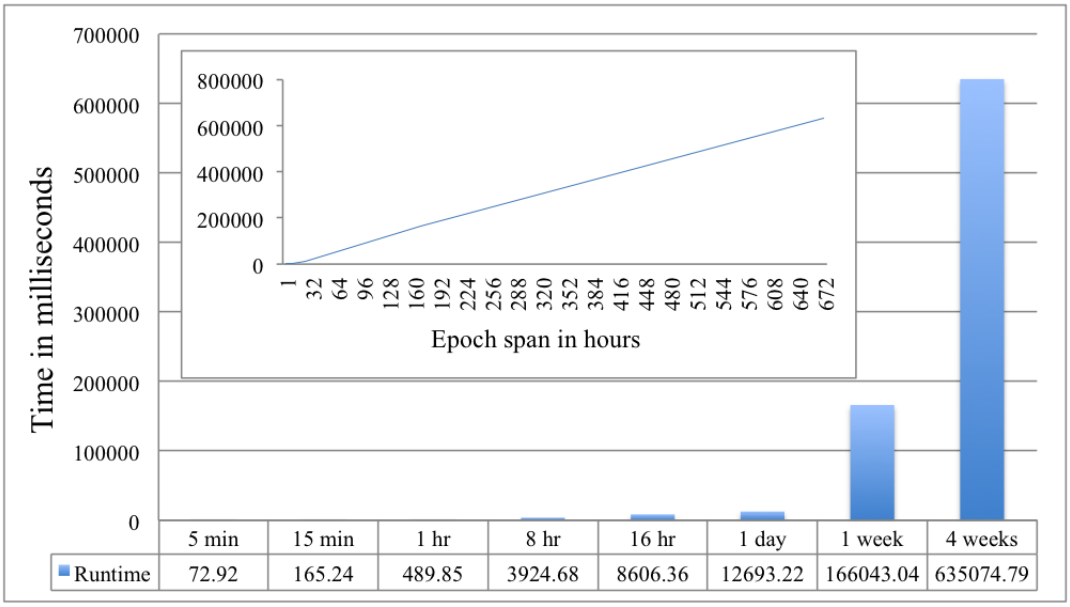


Figure 3.3: Mean runtime of LCM at different epoch spans

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 *minimum support threshold* used throughout this thesis is 10 occurrences in the hour of day in which the volume velocity is the minimum. Figure 3.4 shows the averages of the actual support threshold to which this minimum support threshold translates at different epoch spans.

In the rest of this paper we mine epochs of 1 hour span. The reason behind this choice is an observation that the number of closed itemsets mined from epochs of span 1 hour or more, at the same support threshold, remains the same. This indicates that itemsets mined from shorter epochs 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

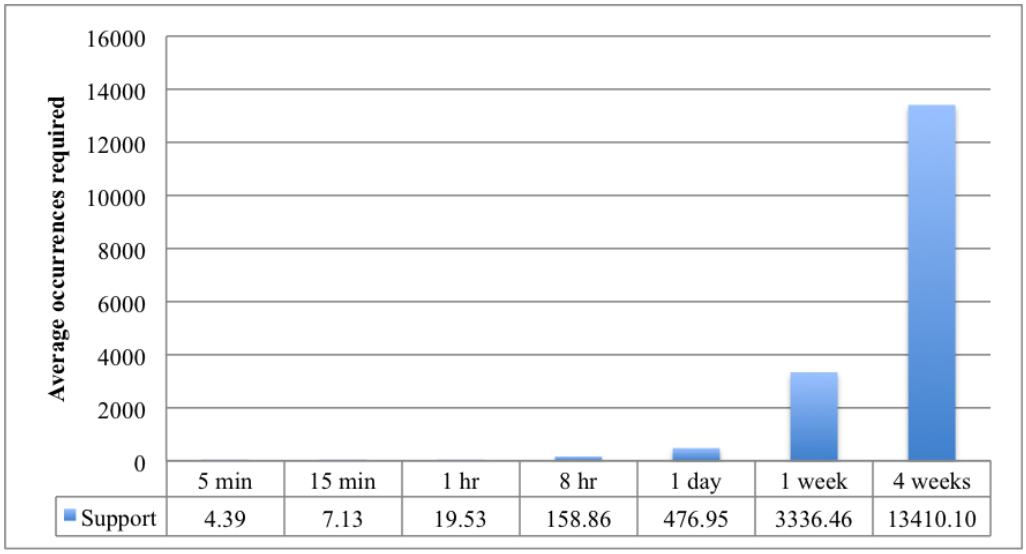


Figure 3.4: Mean support corresponding to minimum support 10, at different epoch spans

the effect of function words by N-gram filtering.

3.3 Filtering Language Constructs

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, the itemset {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 [70]. 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 language model is simply a hash map of the counts of N-grams that appeared within the last month, for N up to 7.

The tokenization of a tweet starts by tokenizing it into unigrams, as described in appendinx A 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 as follows: We picked a topical term that is known to steadily appear with a rather high frequency, and is talked about in all languages; 'obama'. The maximum likely hood estimate of the probability of the term 'obama' within the whole collection of Tweets is 0.0001. 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:

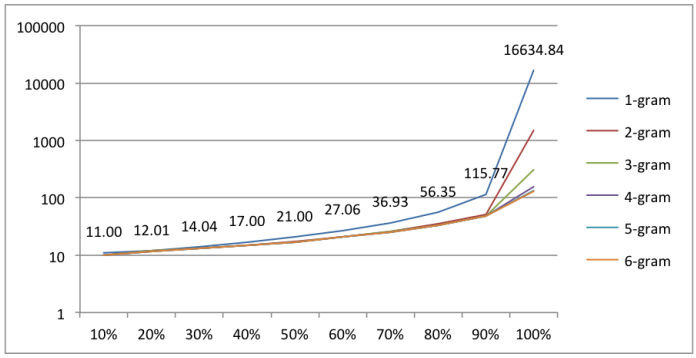
$$\eta_N = \eta_1 \times \frac{\sum_{\{w:\, w \in W \text{ and } w.length \leq N\}} freq(w)}{\sum_{\{v:\, v \in W \text{ and } v.length = 1\}} freq(v)} \tag{3.4}$$

The effect of increasing the maximum length of N-grams is shown in figures 3.5 and 3.6. Figure 3.5 shows the flattening of the distribution by plotting the average decile values at three different times of day. N-grams appearing less than 10 times in an hour (the minimum support threshold we use) are excluded, because they don't contribute to the distribution of items that the algorithm has to mine. The peakedness of the head of the Zipfean distribution is reflected in the peakedness of the 100^{th} percentile. Increasing the maximum N-gram length reduces the peakedness as well as the variance in the peakedness between different hours of day. Figure 3.6 shows how the maximum N-gram length affects the mining algorithm; it shows the number of tokens in the input, 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 3.6(a) shows that the number of distinct items increases substantially as maximum N goes from 1 to 2, then continues increasing slightly until it starts decreasing at $N \leq 5$. The decrease happens because all 4-grams with probability above the threshold are parts

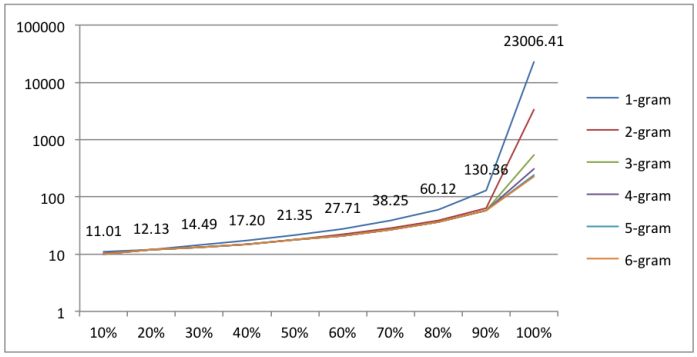
of tweets from services that use the same text and append a URL, such as tweets reporting scores from Game Insight[5]. Such tweets are tokenized into more 4-grams than 5-grams, and the 4-grams appearing in them do not appear elsewhere. Thus, each pair is reduced to one 5-gram. Figure 3.6(b) shows that the number of itemsets continues to decrease as expected. Figure 3.6(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 3.3 because they do not include the time taken for writing the posting list of each itemset.

The number of itemsets in the plots include itemsets of length 1 (frequent N-grams). At $N \leq 5$, the number of closed itemsets of length 2 or more that are mined from an hour epoch averages at 2,439.17. The number of maximal itemsets of length 2 or more averages at 1,831.92, which shows that there aren't many internal nodes in the itemset prefix tree (recall that closed itemsets include maximal itemsets). All of these numbers are counts of distinct unigram sets. After mining itemsets of term N-grams we flatten the itemsets to sets of unigrams again. This is necessary since an itemset will have different N-gram set representations, and its postings list is the union of those of the different representations. This also removes overlap between N-grams of the same itemset, making it easier to reason about how itemsets relate to each other. We will use the relation between an itemset and its subsets to select interesting itemsets in the next chapter.

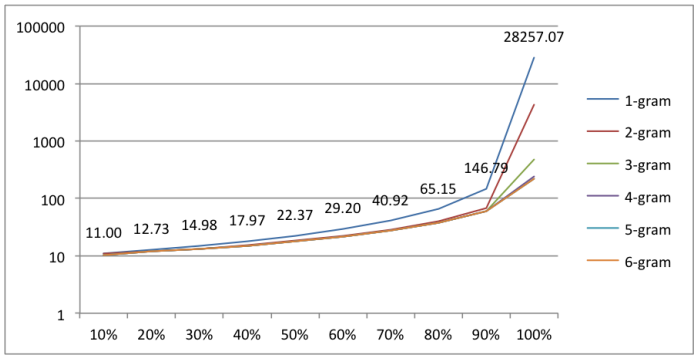
[5]http://www.game-insight.com/



(a) Hour of Day: 00-01 HST (5-6 AM EST)

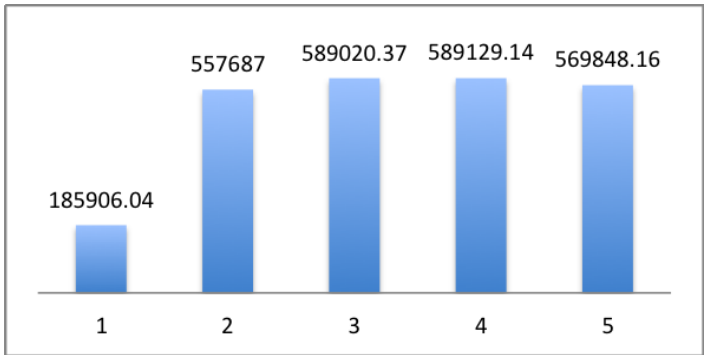


(b) Hour of Day: 08-09 HST (1-2 PM EST)

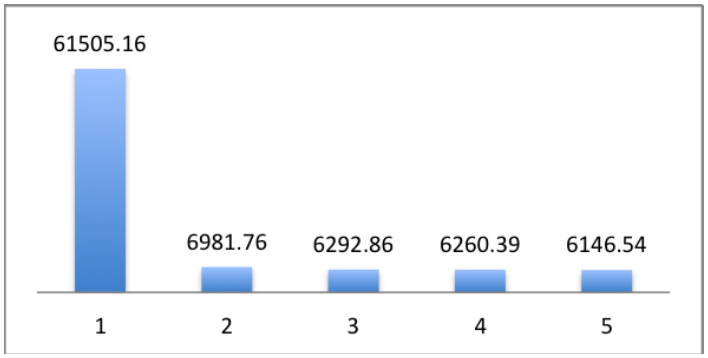


(c) Hour of Day: 16-17 HST (9-10 PM EST)

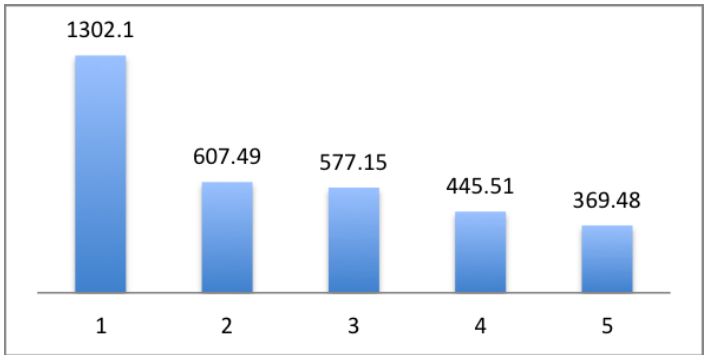
Figure 3.5: Average of token frequency deciles at different times of day and maximum N



(a) Mean number of distinct items at different values of maximum N



(b) Mean number of itemsets at different values of maximum N



(c) Mean runtime in milliseconds at different values of maximum N

Figure 3.6: Effect of changing the maximum N-gram length on mining hour long epoches

Chapter 4

Selecting and Ranking Itemsets

In section 3.3, 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 closed itemsets mined from an hour of twitter data drops from 61,505 to 6,146. In this chapter we propose two conditions that reduces the number of itemsets even further. The first condition filters out some itemsets, and the second condition merges together similar itemsets to reduce redundancy. We also propose a ranking function that scores itemsets according to their novelty. Both conditions and the ranking function exploit the dynamics of social media, therefore we start by discussing it.

4.1 Social Media Dynamics

As discussed earlier, one use case of social media is to share content that a user finds important with her network. The user may share an original status message about the topic of interest, or *re-share* content posted by other users. In twitter, re-sharing is done by retweeting, which can be done in several ways. One way is by using the "retweet" feature of the Twitter client, which simply forwards the tweet to the user's network. The "retweet" feature keeps a reference to the original tweet in the forwarded tweet, but it does not allow editing it - it automatically prepends "RT: @*orignal_poster_username*". Since

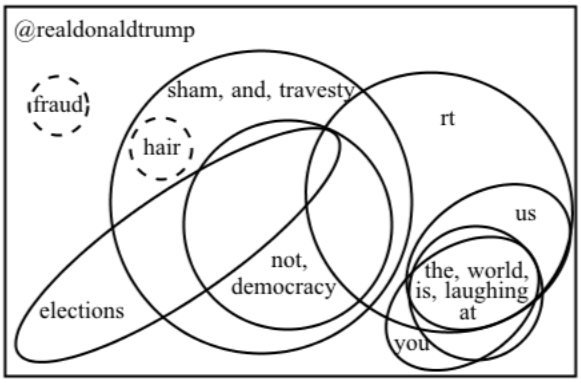


Figure 4.1: Closed, distinct and strongly closed sets

editing is not supported by the "retweet" feature, many users chose to retweet manually to be able to add their own thoughts. Recently this can be done using the "reply" feature, which allows a user to write a tweet that references the original tweet thus maintaining the hierarchy of the conversation. However, manually retweeting is still a prevalent way to re-share a tweet along with the user's comment.

There are several *styles* of manual retweeting [15]. They all basically quote the original tweet along with its poster's username and a retweet indicator, and prepend the user's comment. 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. This selection is an act of collaborative filtering, but it results in many trivially different subsets from the original tweet, and all subsets with enough support will be mined as itemsets. The additions of retweeters also form many different supersets of the of the original tweet, and some additions may represent opinions that are supported enough to be mined as itemsets.

Figure 4.1 illustrates closed itemsets related to Donald Trump's famous tweets in reaction to Obama's victory in the 2012 US elections[1]. Each area in the figure represents the transactions containing the itemset formed by concatenating the items in all intersecting ellipses. 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

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

to also include "not, democracy" and/or "elections", and in most of the cases the retweet indicator "rt" is added. A few people also added something about Donald Trump's "hair" while retweeting, and some people talked about "fraud".

The figure illustrates how 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 itemset mining, a similar effect happens when people retweet, or even when they are writing original posts about a certain fine grained topic. In case of retweeting, people actually start from the original tweet and then remove parts of it to make space for their content. We can imagine that for each fine grained topic there is also a base post that represent the information or ideas within the topic. People make posts about the fine grained topic by selecting parts of the base post and adding their own thoughts. This results in many closed itemsets about the topic that are trivially different from each other.

4.2 Distinct Itemsets

We have seen that the dynamics of posting on social media result in redundant closed itemsets. To reduce redundancy, we propose the *distinct* condition which is not as easily violated as the closed condition and use it for selecting itemsets. The *distinct* condition builds 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 [13]. Mining itemsets based on the confidence of rules they induce has long been recognized as a method for finding "interesting patterns" [21], but since this property is not anti-monotone it cannot be directly used to mine itemsets. It has to be used in a post-processing phase. 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:

$$conf(s_j \rightarrow s_i) = \frac{|T_{s_i} \cap T_{s_j}|}{|T_{s_j}|} \tag{4.1}$$

The *distinct* condition is a novel strengthening of the closed condition so that it is not easily satisfied by any superset of a closed itemset. A closed itemset of length k, call it s_p,

can have as many frequent supersets of length $k+1$ as its support, where s_p will be the intersection of the supersets. From the properties of closed itemsets, each superset that has enough support will result in one or more closed itemsets; if a superset is a maximal itemset then it will result in only one closed itemset, and if it can be extended further it will result in more closed itemsets. Now consider that s_p has a particularly high support, as is the case for itemsets made up of the most discriminative words of a certain topic. In this case, we can filter out closed supersets that have support high enough to satisfy the support threshold but much lower than s_p's support. These itemsets have enough support to be mined as frequent itemsets and they are closed because they have a closed subset, but they do not represent a *distinctive* piece of information or opinion within the topic that s_p represents. This is similar in essence to mining itemsets at multiple minimum supports [53, 72, 82, 89], where a different support threshold is used for each itemset according to the frequency of its components, or its length, or some other function that calculates the *suitable* support threshold. However, we filter or keep itemsets based on a very simple condition that is easy to implement efficiently and intuitive to reason about.

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 condition, or the *distinctiveness* of itemsets selected by the 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:

$$\mathcal{D} = \{s : s \in \mathcal{C} \; and \; \exists \; s_p \subset s \; where \; \frac{|T_s|}{|T_{s_p}|} \geq \kappa\} \tag{4.2}$$

In figure 4.1, *distinct* itemsets are illustrated in solid lines, and closed itemsets that do not satisfy the distinctiveness condition are illustrated in dashed lines. This is not an accurate illustration that is only meant to give an intuition of how *distinct* itemset are different from closed itemsets that are not *distinct*.

Figure 4.2 is a more realistic example. It shows some itemsets (and their support, next to them) from the hour when the US presidential election results were becoming clear.

The itemsets shown are *distinct* at $\kappa = 0.25$. The figure is produced by the graph drawing algorithm proposed by Yifan Hu [34], using the implementation provided in the graph visualization platform Gephi[2]. We run the algorithm on a graph with a node for each itemset and an edge between similar itemsets. Similarity between itemsets is determined based on association rule confidence, as explained in the next section. The similarity is used as the edge weight, thus it determines the distance between itemsets. The figure shows a lot of interesting itemsets. Itemsets pertaining to Donald Trump's tweets appear on the right side of the page, where the itemsets from the "sham and travesty" tweet are in the middle. Close to the bottom left of the page there are itemsets showing the number of electoral votes each candidate got so far ({191, 238} and {191, 249}). It is clear from the figure that considerable redundancy remains in *distinct* itemsets. In the next section we propose overcoming this redundancy by clustering similar itemsets together.

To assess the filtering power of the *distinct* condition we compare the sizes of the set *distinct* itemsets ($\kappa = 0.25$) with the set of closed itemsets of length 2 or more. In the mining results of hour long epochs from the first two weeks of November 2012, the number of *distinct* itemsets averages at 911.71 itemsets as compared to 2428.3 closed itemsets (37.55%). To make sure that no important information is filtered out, we look at itemsets that are filtered out. To facilitate the investigation, we look at itemsets containing known topical terms. In the mining results of hour long epochs from the first two weeks of November 2012, the number of closed itemsets containing at least one of the unigrams 'obama' or 'romney' but are not *distinct* averages at 7.2 itemsets. (an epoch is included in the average only if the keywords occur in it). The hour of the 2012 US elections victory announcement happens to be the hour with the most number of such itemsets, because of the large number of posts about 'obama' and 'romney'. Figure 4.3 shows the filtered itemsets from this hour. Most of the itemsets in the figure are not interesting. Furthermore, interesting ones have surrogates in the set of *distinct* itemsets, either a subset or a superset or an alternative with similar (sometimes more accurate) information. The *distinct* surrogate for each filtered itemset that we find interesting is shown below it.

[2]https://gephi.org/

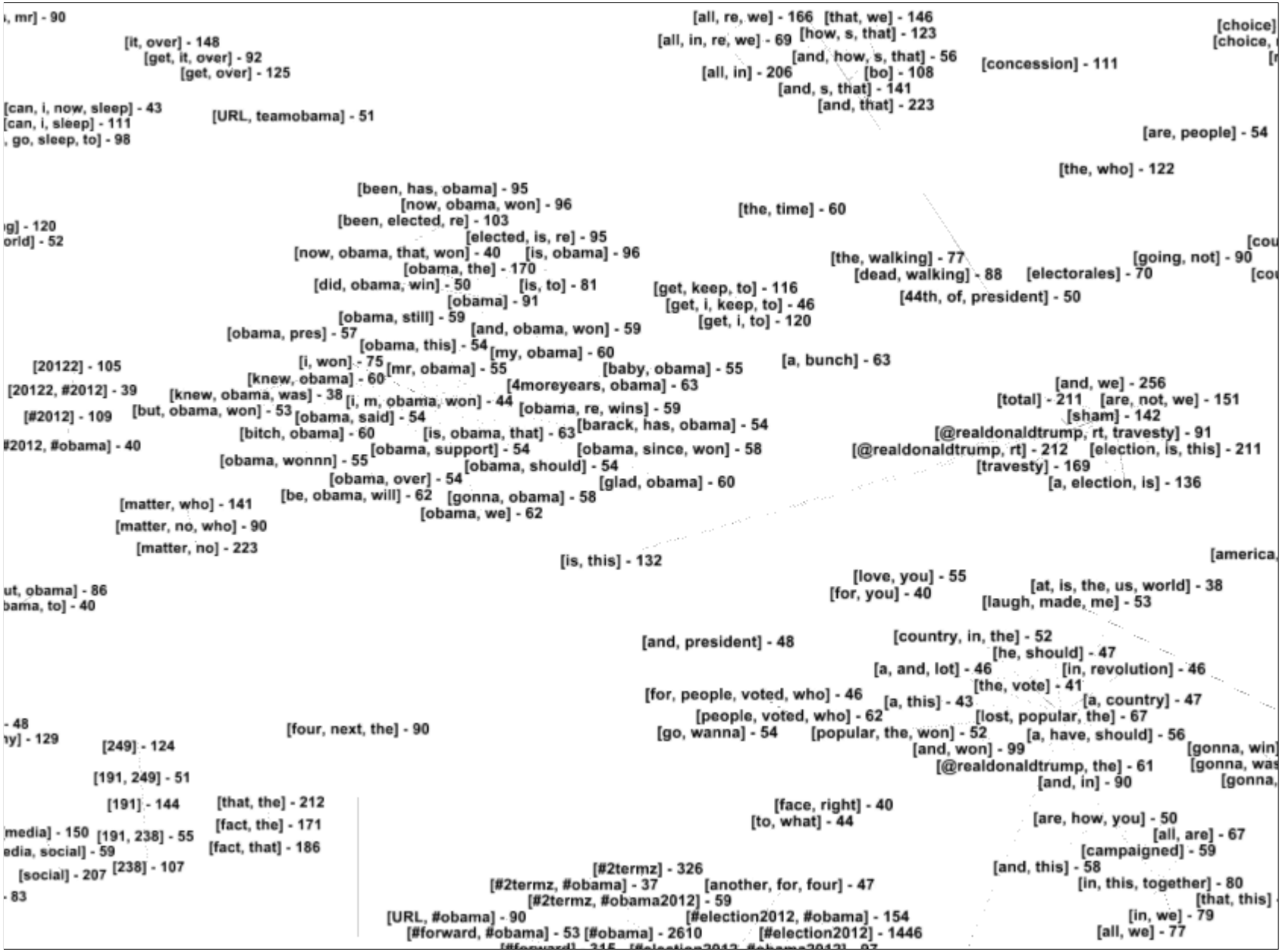


Figure 4.2: Scatter plot of *distinct* itemsets from the hour starting at 10 PM EST on 6 Nov. 2012

1 [romney, that]
2 [still, obama, is]
↪ [still, obama]
3 [romney, what]
4 [romney, with]
5 [will, romney]
6 [romney, if, wins]
7 [romney, for, mitt]
8 [romney, lost]
9 [romney, like]
10 [romney, campaign]
11 [romney, is, mitt]
12 [romney, still]
13 [romney, supporters]
14 [romney, a]
15 [romney, i]
16 [s, romney]
17 [romney, of]
18 [romney, on]
19 [romney, or]
20 [romney, in]
21 [romney, is]
22 [romney, had]
23 [romney, at]
24 [romney, 244]
↪ [obama, 244]
25 [romney, has]
26 [romney, got]
27 [romney, can]
28 [romney, but]
29 [romney, was]
30 [romney, the]
31 [voted, for, obama, who]
32 [2012, obama]
33 [romney, aint]
34 [obama, is, barack]
35 [gana, obama]
36 [obama, gano]
37 [michelle, obama]
39 [years, more, obama, 4]
↪ [#4moreyears, obama]
40 [romney, for, vote]
41 [de, obama]
42 [obama, an]
43 [on, obama]
44 [obama, it]
45 [so, obama]
46 [obama, d]
47 [did, obama, it]
48 [for, obama, it]
49 [que, obama]
50 [URL, obama]
51 [can, obama]
52 [obama, all]
53 [so, obama, won]
↪ [wonnn, obama]
54 [yes, obama]
55 [obama, is, the]
56 [supporters, obama]
57 [obama, when]
58 [with, obama]
59 [obama, team]
60 [obama, has, won]
↪ [obama, we, won]
61 [this, got, obama]
62 [yeah, obama]
63 [romney, about]
64 [obama, happy, won]
↪ [glad, obama, won]
65 [yesss, obama]
66 [president, obama, barack]

Figure 4.3: Itemsets containing 'barack' or 'romeny' that are closed but not *distinct*

4.3 Strongly Closed Itemsets

To remove the redundancy in *distinct* itemsets we merge similar ones into *strongly closed* itemset clusters. This will also filter out any itemsets that do not belong to any particular topic since they will not be part of a cluster. 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. According to equation 4.1 this is the confidence of the association rule that $s_c \rightarrow s_d$. Confidence is not a symmetric measure,

but our proposed clustering makes use of the order in which PPC-extension generates itemsets (as discussed in section 3.1) to determine which itemset should be the antecedent and which should be the consequent.

If PPC-extension uses a total ordering of items that is non-increasing in item frequency, then an itemset generated earlier should be the antecedent, and an itemset generated later should be the consequent. This builds upon the use of variable length N-grams as items, as described in section 3.3, so that the most frequent items are not function words. In that case, following the non-increasing order of item frequency leads to the following:

- Itemsets related to a certain topic are mined close to each other. Recall that PPC-Extension is equivalent to a depth first traversal of a hypothetical prefix tree of closed itemsets. That is, closed supersets of each itemset are generated before moving to an itemset that is not a superset.
- The most discriminative itemset from each topic (according to users' selection) is mined before other itemsets from the same topic, because it is composed of items having the highest support within its topic.
- If an itemset branches out into supersets related to two topics or two opinions within one topic, this itemset is mined before itemsets from either of the two branches. It should not be clustered with either of the branches.
- For an itemset to belong to a topic, it must contain a set of topical words. Since we are processing items in descending order of frequency, then itemsets containing multiple topical words are generated when mining the supersets of the itemset containing only the most frequent of them. Thus, itemsets belonging to a topic will be mined after the topic's most discriminative itemset, even if they are not its superset.

According to the observations above, the *topical* similarity between itemsets can be indicated by the confidence of the association rule that an itemset generated earlier implies an itemset generated later. Using this similarity measure, we cluster itemsets as follows:

1. Itemsets are processed one by one as they are generated by PPC-extension (using a total ordering that is non-decreasing in frequency).

2. If the itemset is not *distinct* it is discarded.

3. A *distinct* itemset is clustered with a previously generated 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).

4. If more than one satisfies this condition, the *distinct* itemset is clustered with the one having the highest overlap ratio.

5. 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.

Each cluster is aggregated into a *strongly closed* itemset, which is the union of all cluster members, and its support is the size of the union of their postings lists.

This clustering scheme adds an itemset, s_i, to the cluster containing the itemset, s_j, which maximizes the confidence of the rule $conf(s_j \rightarrow s_i)$, with a lower bound on the overlap to maintain distinctiveness. Using $\mathcal{D}$ to denote the set of *distinct* itemsets, we define the desired clustering and the strongly closed itemset represented by each cluster as follows:

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

$$S_l = \{w : \; w \in \bigcup_{(s_i,s_j) \in r_l} s_i \; where \; r_l \in \mathcal{R}\} \tag{4.3}$$

A different clustering scheme could be used; the goal is to reduce redundancy, and we devised this scheme because it has several merits that makes it suitable. First, the similarity is measured in the transaction space, so that itemsets are clustered according to *topical* similarity and regardless of the lexically similarity. Jaccard similarity is another possible similarity measure, but it cannot be interpreted as the strength of an implication.

Also, calculating confidence requires calculating the intersection only, so it is easier to terminate the calculation early if the confidence would not be more than the required threshold. Second, this clustering can be implemented efficiently using techniques similar to the ones proposed by Bayardo et al. [8].

The main ideas for an efficient implementation 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. We calculate the similarity between the postings lists of two itemsets, only if they intersect in one item at least. 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 4.1. Algorithm 2 shows a possible implementation.

To illustrate how the algorithm works, in figure 4.4 we show the clustering hierarchy of itemsets pertaining to the "sham and travesty" tweet. The figure shows a tree structure of how itemsets were added to the cluster. An itemset's parent in the tree is the itemset that resulted in the rule with the maximum confidence; we call it the best clustering candidate. An itemset is added to a cluster through its best candidate, and if the best candidate does not belong to a cluster a new one is created with the best merge candidate as the centroid. The best merge candidate always has a smaller id, since ids are given sequentially to itemsets and we cluster each itemset with ones that were generated before it. The support of the best candidate is not necessarily higher than the support of the itemset being clustered, but all itemsets within the topic has a support lower than that of the topic's most discriminative itemset, n810={sham}. For example, the hierarchy $n1018 \rightarrow n1019 \rightarrow n1075$ has support values 120, 124, 126 respectively, which are all less than 142 (the support of n810). Finally, we note that this single *strongly closed itemset* cluster replaces 26 *distinct* itemsets. The overall reduction in the number of itemsets will be discussed in section 5.1.

Another example is given in figure 4.5 to illustrate an itemset that branches into different opinions within one topic. These itemsets are mined from tweets of fans voting for different artists to win the MTV Europe Music Awards (MTVEMA). There are 4 distinct clusters in the figure; one for each artist, and one for itemsets not mentioning any artist.

Input: κ: Minimum distinctiveness threshold

Data: $\mathcal{C}$: Closed itemsets produced by LCM, in the order they were generated

Result: $\mathcal{R}$: Strong closed itemset clusters

for $i \leftarrow 2$ *to* $|\mathcal{C}|$ **do**
  $C \leftarrow \{s_c : s_c \in \mathcal{C}\, and\, c < i\, and\, |s_c \cap s_i| > 0\}$; // Candidates for clustering
  $P \leftarrow \{s_p : s_p \in \mathcal{C}\, and\, p < i\, and\, s_p \cap s_i = s_p\}$; // s_i's subsets (ancestors)
  $s_p \leftarrow \mathbf{argmax}_{s_p \in P} \frac{|T_{s_i}|}{|T_{s_p}|}$; // Direct parent
  if $s_p = null$ ***OR*** $\frac{|T_{s_i}|}{|T_{s_p}|} < \kappa$ **then** // Not a distinct itemset
    $\mathcal{C} \leftarrow \mathcal{C} \setminus s_i$; // Should not be considered in later iterations
    continue ; // Should not be added to a cluster
  end
  $s_m \leftarrow null$, $maxConf \leftarrow 0$; // Best clustering candidate and its score
  foreach $s_c \in C$ **do**
    $\Delta \leftarrow (1 - (1 - \kappa))|T_{s_c}|$; // Maximum difference
    $\delta \leftarrow \text{difference}(T_{s_c}, T_{s_c \cup s_i}, \Delta)$; // Terminates early if $\delta > \Delta$
    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 clustering candidate
        $maxConf \leftarrow conf$;
      end
    end
  end
  if $s_m \neq null$ **then**
    if $\mathcal{R}[s_m] = null$ **then** $\mathcal{R}[s_m] \leftarrow s_m$; // New cluster with centroid s_m
    $\mathcal{R}[s_i] \leftarrow \mathcal{R}[s_m]$; // Add s_i to the cluster containing 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
end
return $\mathcal{R}$;

Algorithm 2: Forming strongly closed itemset clusters

Id	Itemset	Support
n810	[sham]	142
n906	[and, sham]	135
n907	[and, sham, travesty]	134
n908	[and, sham, total, travesty]	132
n944	[a, and, is, sham, total, travesty]	131
n945	[a, and, election, is, sham, this, total, travesty]	130
n1016	[@realdonaldtrump, this]	129
n1018	[@realdonaldtrump, a, democracy, not, this]	120
n1019	[@realdonaldtrump, a, and, is, sham, this, total, travesty]	124
n1075	[@realdonaldtrump, election, this]	126
n1076	[@realdonaldtrump, a, and, election, is, sham, this, total, travesty]	125
n1055	[a, and, are, democracy, is, not, sham, total, travesty, we]	125
n1056	[a, and, are, democracy, election, is, not, sham, this, total, travesty, we]	124
n1057	[@realdonaldtrump, a, and, are, is, not, sham, this, total, travesty, we]	120
n1079	[@realdonaldtrump, a, and, are, election, is, not, sham, this, total, travesty, we]	121
n1059	[@realdonaldtrump, a, and, are, democracy, is, not, sham, this, total, travesty, we]	119
n1080	[@realdonaldtrump, a, and, are, democracy, election, is, not, sham, this, total, travesty, we]	120
n963	[a, are, not]	132
n964	[a, are, not, we]	128
n1053	[a, and, are, is, not, sham, total, travesty, we]	127
n1054	[a, and, are, election, is, not, sham, this, total, travesty, we]	126
n965	[a, are, democracy, not]	128
n966	[a, are, democracy, not, we]	127
n1033	[travesty, we]	128
n925	[a, is, total]	134
n943	[a, and, sham, travesty]	133

Figure 4.4: The hierarchy of itemsets in the cluster pertaining to the “sham and travesty” tweet

Id	Itemset	Support
n0	#mtvema	624
n2	#mtvema,be,big,the,will	608
n3	#mtvema,be,big,the,will,winner	607
n4	#mtvema,pick,tweet,winner,your	606
n5	#mtvema,at,pick,tweet,winner,your	605
n6	#mtvema,big,tweet,winner,your	605
n11	URL,at,pick,tweet,your	604
n25	#mtvema,be,bieber,big,justin,the,will,winner	405
n26	#mtvema,at,be,bieber,big,justin,pick,the,tweet,will,winner,your	403
n27	#mtvema,be,bieber,justin,think,will	402
n28	#mtvema,URL,at,be,bieber,big,justin,pick,the,tweet,will,winner,your	401
n91	#mtvema,be,big,katy,perry,the,think,will	166
n119	#mtvema,#emawinkaty,be,big,katy,perry,the,think,will	159
n126	#mtvema,#emawinkaty,be,big,i,katy,perry,the,think,will	157
n122	#mtvema,#emawinkaty,be,big,katy,perry,the,think,will,winner	158
n99	#mtvema,be,big,i,katy,perry,the,think,will	164
n104	#mtvema,be,big,i,katy,perry,the,think,will,winner	163
n95	#mtvema,be,big,katy,perry,the,think,will,winner	165
n97	#mtvema,be,big,katy,perry,the,think,tweet,will,winner,your	164
n5207	be,big,gaga,the,will	25
n5275	#mtvema,be,big,gaga,the,tweet,will,winner	24
n5569	#mtvema,at,be,big,gaga,pick,the,tweet,will,winner,your	23
n5562	#mtvema,#emawingaga,at,pick,tweet,winner,your	23
n5572	#mtvema,be,big,gaga,lady,the,tweet,will,winner	23

Figure 4.5: Several clusters for itemsets pertaining to the MTV Europe Music Awards votes

4.3.1 Bounding Space and Time Requirements

Algorithm 2 requires keeping *distinct* itemsets and their postings lists in memory, so that the similarity between a *distinct* itemset and all the previously generated ones can be efficiently calculated. This is an overhead that the original LCM does not incur, but it is not a large overhead; due to the limited number of *distinct* itemsets, the memory requirements does not exceed 6 GB for mining hour long epochs using a Java implementation that uses a lot of objects. However, it is possible to use less memory and also bound the memory requirement by keeping only a limited number of itemsets in memory.

The number of positions between an itemset and a clustering candidate is 1973.11 on average, with a standard deviation of 1443.64. The large standard deviation indicates that candidates are either found close to the itemset (less than 500 positions behind), or in a position that is several thousands of itemsets behind. According to our observation that itemsets from one topic are generated close to each other, when the total ordering is descending in items frequency, we limit the mining results kept in memory to the last 1000 *distinct* itemsets and their postings lists. This sets a bound on the memory requirement, as well as the runtime which is dominated by the similarity calculations. Without limiting the number of itemsets kept in memory, the runtime for filtering and clustering hour long epochs at $\kappa = 0.25$ is 5.2 seconds on average. This is reduced to less than 1 second if only the last 1000 results from LCM are kept in memory. More detailed discussion of the runtime performance of the algorithm is given in section 5.1.

4.4 Temporal Ranking

A General Measure of Rule Interestingness Szymon Jaroszewicz and Dan A. Simovici University of Massachusetts at Boston

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:

$$\begin{aligned} IG(P(s_i), Q(s_i)) &= -(H(P(s_i)) - H(P(s_i), Q(s_i))) \\ &= \sum P(s_i) \log P(s_i) - \sum P(s_i) \log Q(s_i) \\ &= KLD(P(s_i)||Q(s_i)) \end{aligned} \tag{4.4}$$

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:

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

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

closed itemsets:

$$\overline{IG}(S) = \frac{IG^2(P(s_{i1}, ...s_{im}), Q(s_{i1}, ...s_{im}))}{m} \tag{4.5}$$

Chapter 5

Evaluation

5.1 Efficiency

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 5.1 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 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 5.2 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

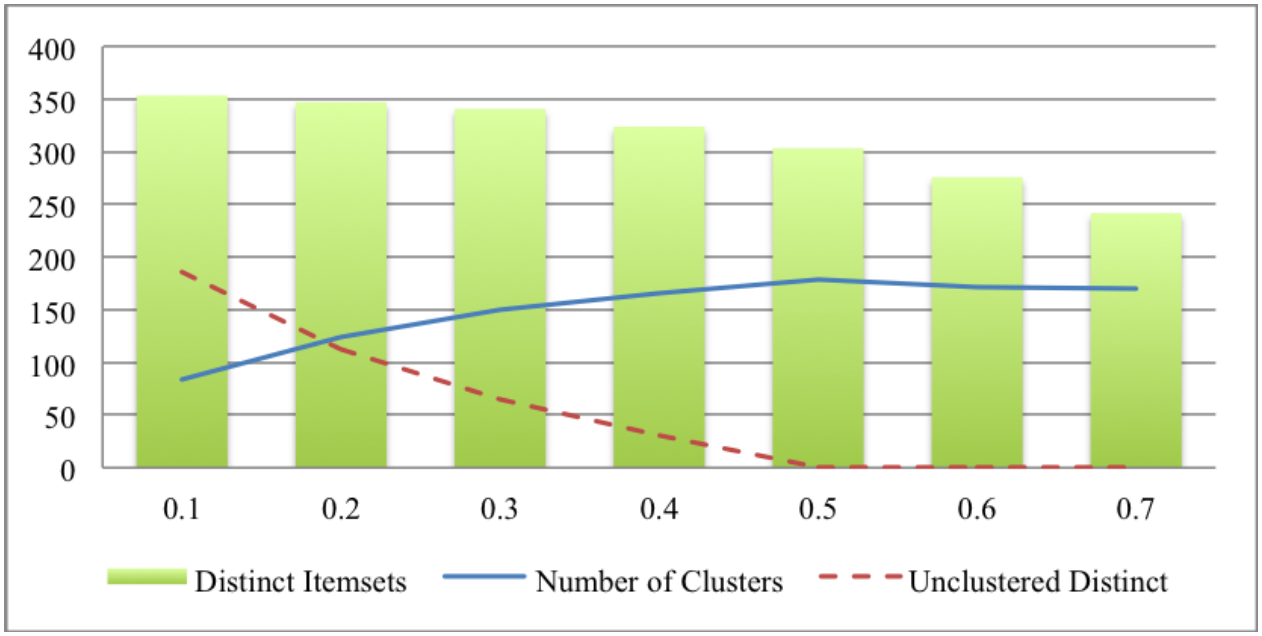


Figure 5.1: Effect of changing κ on mining results

another frequent itemset mining algorithm, FP-Zhu [29], which was the runner up at FIMI 2004 [40]. 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 2. 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 5. If all itemsets are kept in memory the runtime increases slightly and

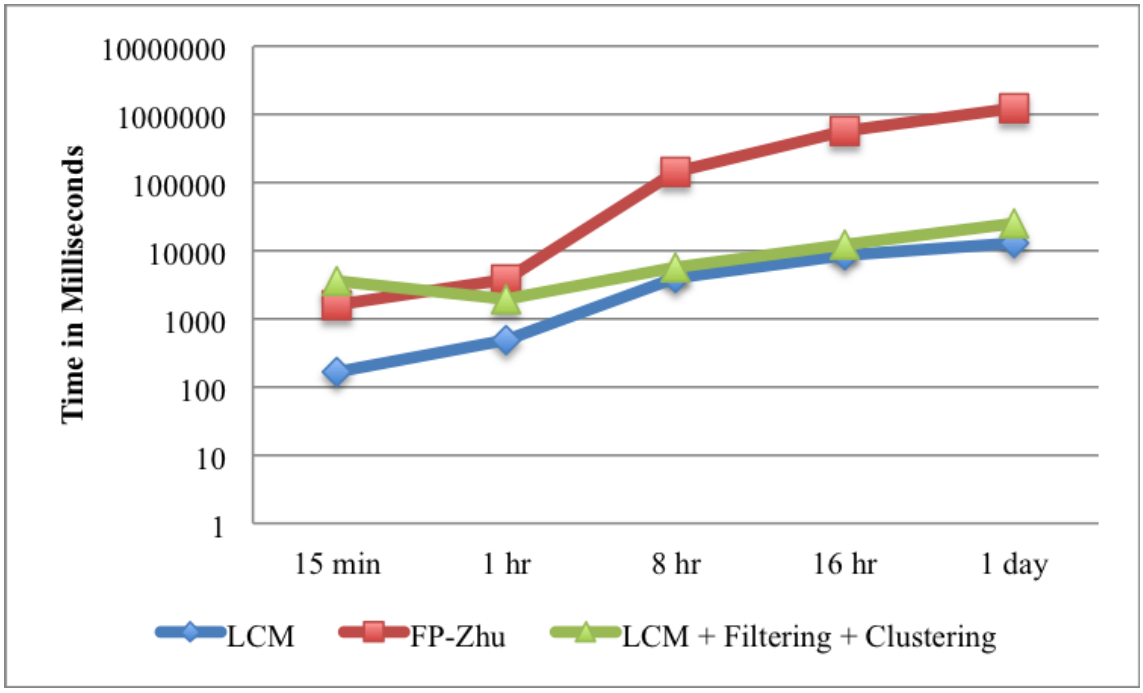


Figure 5.2: Runtime of itemset mining alone and with filtering and clustering at different epoch spans

averages at 5.2 seconds for filtering a one-hour epoch.

5.2 Effectiveness

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 5.1 and 5.2 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 5.2 to save space since the itemsets are self explanatory.

In table 5.1, 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 5.2, 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 5.3 we show the top 3 itemsets picked by the MTV algorithm [56] 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*.

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
	once, you, go,	A popular cultural reference.

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 5.2: Top 3 itemsets for hours in November 9th

Nov	@realdonaldtrump,this,elections,...[266035509162303492]
6th,	we, re, all, in, this, together, ..., bo [266031109979131904]
22:00	@realdonaldtrump,our,country,is,...[266037143628038144]
Nov	URL,and,autotweet,checked,followed,me,today,unfollowed
6th,	house,of,representatives,shouldnt,...[266040877552656385]
23:00	URL,#android,#androidgames,#gameinsight
Nov	URL,and,autotweet,checked,followed,me,today,unfollowed
6th,	URL,#android,#androidgames,#gameinsight
23:30	@barackobama,@ryanseacrest, what, was, your, first, words, in, reaction, to, re, election, 2
Nov	URL,and,autotweet,checked,followed,me,today,unfollowed
6th,	the, best, is, yet, to, come
00:00	URL,#android,#androidgames,#gameinsight
Nov	#emawinbieber, i, think, justin, bieber, ... [see table 5.2]
9th,	@boyquotations,do,everyone,follow,followers,gain,more
15:00	#emawinkaty, i, think, katy, perry, ... [see table 5.2]

Table 5.3: Top 3 picked by the MTV algorithm

Chapter 6

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.

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.

APPENDICES

Appendix A

Unigram Tokenization

The data is tokenized into unigrams using white space and punctuation marks as separators. The characters '@', '#' and '_' are allowed within a unigram, while the rest of the punctuation marks are treated as delimiters. Only Latin characters (Unicode code points less than 'u024F') and numbers are allowed within a unigram. Any other characters that are not delimiters are skipped. If a unigram is a number then the dot and the comma characters are allowed within it. The tokenizer also does the following:

- All URLs (from 'http[s]:' to the next whitespace) are replaced by the unigram "URL".
- Runs of the same character is reduced to only 3 repetitions (for example, "coooool" is replaced by "coool").
- Hashtags are stored twice, with and without the the '#' character. The tag without the '#' character is left at tag's positions, while the other is appended at the end of the tweet. This handles cases where the hashtag is used in place of a word, such as "president #obama...".
- The apostrophe used in contractions (can't, don't, ..etc) is removed from the unigram, but does not act as a delimiter.

References

[1] Younos Aboulnaga and Charles L. A. Clarke. Frequent itemset mining for query expansion in microblog ad-hoc search. In *Proceedings of the 21st TREC Conference*, Text Retrieval Evaluation Conference (TREC), Gaithersburg, MD, USA, 2012.

[2] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between sets of items in large databases. In *ACM SIGMOD Record*, volume 22, pages 207–216. ACM, 1993.

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

[4] Mubarak Albathan, Yuefeng Li, and Abdulmohsen Algarni. Using patterns co-occurrence matrix for cleaning closed sequential patterns for text mining. In *2012 IEEE/WIC/ACM International Conference on Web Intelligence*, pages 201–205. IEEE, 2012.

[5] James Allan. *Topic detection and tracking: event-based information organization*, volume 12. Kluwer Academic Pub, 2002.

[6] James Allan, Jaime G Carbonell, George Doddington, Jonathan Yamron, and Yiming Yang. Topic detection and tracking pilot study final report. 1998.

[7] James Allan, Victor Lavrenko, Daniella Malin, and Russell Swan. Detections, bounds, and timelines: Umass and tdt-3. In *Proceedings of Topic Detection and Tracking Workshop (TDT-3)*, pages 167–174. Vienna, VA, 2000.

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

[9] Roberto J Bayardo Jr and Rakesh Agrawal. Mining the most interesting rules. In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 145–154. ACM, 1999.

[10] Hila Becker, Mor Naaman, and Luis Gravano. Beyond trending topics: Real-world event identification on twitter. In *Proceedings of the Fifth International AAAI Conference on Weblogs and Social Media (ICWSM'11)*, 2011.

[11] Albert Bifet and Ricard Gavalda. Learning from time-changing data with adaptive windowing. In *SIAM International Conference on Data Mining*, pages 443–448, 2007.

[12] Jean-Francois Boulicaut, Artur Bykowski, and Christophe Rigotti. Approximation of frequency queries by means of free-sets. In *Principles of Data Mining and Knowledge Discovery*, pages 75–85. Springer, 2000.

[13] Jean-François Boulicaut, Artur Bykowski, and Christophe Rigotti. Free-sets: a condensed representation of boolean data for the approximation of frequency queries. *Data Mining and Knowledge Discovery*, 7(1):5–22, 2003.

[14] Gerlof Bouma. Normalized (pointwise) mutual information in collocation extraction. *Proceedings of GSCL*, pages 31–40, 2009.

[15] Danah Boyd, Scott Golder, and Gilad Lotan. Tweet, tweet, retweet: Conversational aspects of retweeting on twitter. In *System Sciences (HICSS), 2010 43rd Hawaii International Conference on*, pages 1–10. IEEE, 2010.

[16] Sergey Brin, Rajeev Motwani, and Craig Silverstein. Beyond market baskets: generalizing association rules to correlations. In *ACM SIGMOD Record*, volume 26, pages 265–276. ACM, 1997.

[17] Toon Calders and Bart Goethals. Mining all non-derivable frequent itemsets. In *Principles of Data Mining and Knowledge Discovery*, pages 74–86. Springer, 2002.

[18] Toon Calders and Bart Goethals. Non-derivable itemset mining. *Data Mining and Knowledge Discovery*, 14(1):171–206, 2007.

[19] Deepayan Chakrabarti and Kunal Punera. Event summarization using tweets. In *Proceedings of the Fifth International AAAI Conference on Weblogs and Social Media*, pages 66–73, 2011.

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

[21] Edith Cohen, Mayur Datar, Shinji Fujiwara, Aristides Gionis, Piotr Indyk, Rajeev Motwani, Jeffrey D. Ullman, and Cheng Yang. Finding interesting associations without support pruning. *Knowledge and Data Engineering, IEEE Transactions on*, 13(1):64–78, 2001.

[22] Jacques JF Commandeur and Siem Jan Koopman. *An introduction to state space time series analysis*. OUP Oxford, 2007.

[23] John N Darroch and Douglas Ratcliff. Generalized iterative scaling for log-linear models. *The annals of mathematical statistics*, 43(5):1470–1480, 1972.

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

[25] Venkatesh Ganti, Johannes Gehrke, and Raghu Ramakrishnan. Mining data streams under block evolution. *ACM SIGKDD Explorations Newsletter*, 3(2):1–10, 2002.

[26] Liqiang Geng and Howard J Hamilton. Interestingness measures for data mining: A survey. *ACM Computing Surveys (CSUR)*, 38(3):9, 2006.

[27] Chris Giannella, Jiawei Han, Jian Pei, Xifeng Yan, and Philip S Yu. Mining frequent patterns in data streams at multiple time granularities. *Next generation data mining*, 212:191–212, 2003.

[28] Anna C Gilbert, Yannis Kotidis, S Muthukrishnan, and Martin J Strauss. Surfing wavelets on streams: One-pass summaries for approximate aggregate queries. In *Proceedings of the International Conference on Very Large Data Bases*, pages 79–88, 2001.

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

[30] Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. In *ACM SIGMOD Record*, volume 29, pages 1–12. ACM, 2000.

[31] Jiawei Han, Jian Pei, Yiwen Yin, and Runying Mao. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data mining and knowledge discovery*, 8(1):53–87, 2004.

[32] Matthew Hoffman, David M Blei, and Francis Bach. Online learning for latent dirichlet allocation. *Advances in Neural Information Processing Systems*, 23:856–864, 2010.

[33] Liangjie Hong and Brian D Davison. Empirical study of topic modeling in twitter. In *Proceedings of the First Workshop on Social Media Analytics*, pages 80–88. ACM, 2010.

[34] Yifan Hu. Efficient, high-quality force-directed graph drawing. *Mathematica Journal*, 10(1):37–71, 2005.

[35] Jonathan Hurlock and Max L Wilson. Searching twitter: Separating the tweet from the chaff. *Proc. ICWSM 2011*, 2011.

[36] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 604–613. ACM, 1998.

[37] Szymon Jaroszewicz and Dan A Simovici. Interestingness of frequent itemsets using bayesian networks as background knowledge. In *Proceedings of the tenth ACM*

SIGKDD international conference on Knowledge discovery and data mining, pages 178–186. ACM, 2004.

[38] Akshay Java, Xiaodan Song, Tim Finin, and Belle Tseng. Why we twitter: understanding microblogging usage and communities. In *Proceedings of the 9th WebKDD and 1st SNA-KDD 2007 workshop on Web mining and social network analysis*, pages 56–65. ACM, 2007.

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

[40] Roberto J. Bayardo Jr., Bart Goethals, and Mohammed Javeed Zaki, editors. *FIMI '04, Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations, Brighton, UK, November 1, 2004*, volume 126 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2004.

[41] Won-Young Kim, Young-Koo Lee, and Jiawei Han. Ccmine: Efficient mining of confidence-closed correlated patterns. In *Advances in Knowledge Discovery and Data Mining*, pages 569–579. Springer, 2004.

[42] Jon Kleinberg. Bursty and hierarchical structure in streams. *Data Mining and Knowledge Discovery*, 7(4):373–397, 2003.

[43] Marzena Kryszkiewicz. Concise representation of frequent patterns based on disjunction-free generators. In *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*, pages 305–312. IEEE, 2001.

[44] Haewoon Kwak, Changhyun Lee, Hosung Park, and Sue Moon. What is twitter, a social network or a news media? In *Proceedings of the 19th international conference on World wide web*, pages 591–600. ACM, 2010.

[45] Cher Han Lau, YueFeng Li, and Dian Tjondronegoro. Microblog retrieval using topical features and query expansion. In *Proceedings of the 20th TREC Conference*, Text Retrieval Evaluation Conference (TREC), Gaithersburg, MD, USA, 2011.

[46] Young-Koo Lee, Won-Young Kim, Y Dora Cai, and Jiawei Han. Comine: Efficient mining of correlated patterns. In *Proceedings of the Third IEEE International Conference on Data Mining*, page 581, 2003.

[47] Janette Lehmann, Bruno Gonçalves, José J Ramasco, and Ciro Cattuto. Dynamical classes of collective attention in twitter. In *Proceedings of the 21st international conference on World Wide Web*, pages 251–260. ACM, 2012.

[48] Jure Leskovec, Lars Backstrom, and Jon Kleinberg. Meme-tracking and the dynamics of the news cycle. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 497–506. ACM, 2009.

[49] David D Lewis. An evaluation of phrasal and clustered representations on a text categorization task. In *Proceedings of the 15th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 37–50. ACM, 1992.

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

[51] Zhenmin Li, Zhifeng Chen, Sudarshan M Srinivasan, and Yuanyuan Zhou. C-miner: Mining block correlations in storage systems. In *Proceedings of the 3rd USENIX Conference on File and Storage Technologies*, volume 186. USENIX Association, 2004.

[52] Jimmy Lin and Gilad Mishne. A study of "churn" in tweets and real-time search queries. In *Sixth International AAAI Conference on Weblogs and Social Media*, 2012.

[53] Bing Liu, Wynne Hsu, and Yiming Ma. Mining association rules with multiple minimum supports. In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 337–341. ACM, 1999.

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

[55] GILAD LOTAN, ERHARDT GRAEFF, MIKE ANANNY, DEVIN GAFFNEY, IAN PEARCE, and DANAH BOYD. The revolutions were tweeted: Information flows during the 2011 tunisian and egyptian revolutions. 2011.

[56] Michael Mampaey, Nikolaj Tatti, and Jilles Vreeken. Tell me what i need to know: succinctly summarizing data with itemsets. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 573–581. ACM, 2011.

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

[58] Ahmed Metwally, Divyakant Agrawal, and Amr El Abbadi. Duplicate detection in click streams. In *Proceedings of the 14th international conference on World Wide Web*, pages 12–21. ACM, 2005.

[59] Mor Naaman, Hila Becker, and Luis Gravano. Hip and trendy: Characterizing emerging trends on twitter. *Journal of the American Society for Information Science and Technology*, 62(5):902–918, 2011.

[60] Mor Naaman, Jeffrey Boase, and Chih-Hui Lai. Is it really about me?: message content in social awareness streams. In *Proceedings of the 2010 ACM conference on Computer supported cooperative work*, pages 189–192. ACM, 2010.

[61] Brendan O'Connor, Michel Krieger, and David Ahn. Tweetmotif: Exploratory search and topic summarization for twitter. *Proceedings of ICWSM*, pages 2–3, 2010.

[62] Edward R Omiecinski. Alternative interest measures for mining associations in databases. *Knowledge and Data Engineering, IEEE Transactions on*, 15(1):57–69, 2003.

[63] Ruchi Parikh and Kamalakar Karlapalem. Et: events from tweets. In *Proceedings of the 22nd international conference on World Wide Web companion*, pages 613–620. International World Wide Web Conferences Steering Committee, 2013.

[64] Nicolas Pasquier, Yves Bastide, Rafik Taouil, and Lotfi Lakhal. Discovering frequent closed itemsets for association rules. In *Database Theory—ICDT'99*, pages 398–416. Springer, 1999.

[65] Nicolas Pasquier, Yves Bastide, Rafik Taouil, and Lotfi Lakhal. Efficient mining of association rules using closed itemset lattices. *Information systems*, 24(1):25–46, 1999.

[66] Jian Pei, Jiawei Han, Runying Mao, et al. Closet: An efficient algorithm for mining frequent closed itemsets. In *ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, volume 4, pages 21–30, 2000.

[67] Saša Petrović, Miles Osborne, and Victor Lavrenko. Streaming first story detection with application to twitter. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 181–189. Association for Computational Linguistics, 2010.

[68] Ana-Maria Popescu, Marco Pennacchiotti, and Deepa Paranjpe. Extracting events and event descriptions from twitter. In *Proceedings of the 20th international conference companion on World wide web*, pages 105–106. ACM, 2011.

[69] Bruno Pôssas, Nivio Ziviani, Wagner Meira Jr, and Berthier Ribeiro-Neto. Set-based model: a new approach for information retrieval. In *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 230–237. ACM, 2002.

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

[71] Alan Ritter, Oren Etzioni, Sam Clark, et al. Open domain event extraction from twitter. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1104–1112. ACM, 2012.

[72] Masakazu Seno and George Karypis. Finding frequent patterns using length-decreasing support constraints. *Data Mining and Knowledge Discovery*, 10(3):197–228, 2005.

[73] Abraham Silberschatz and Alexander Tuzhilin. What makes patterns interesting in knowledge discovery systems. *Knowledge and Data Engineering, IEEE Transactions on*, 8(6):970–974, 1996.

[74] Avi Silberschatz and Alexander Tuzhilin. On subjective measures of interestingness in knowledge discovery. In *Proceedings of KDD-95: First International Conference on Knowledge Discovery and Data Mining*, pages 275–281, 1995.

[75] Pang-Ning Tan, Vipin Kumar, and Jaideep Srivastava. Selecting the right interestingness measure for association patterns. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 32–41. ACM, 2002.

[76] Nikolaj Tatti. Maximum entropy based significance of itemsets. *Knowledge and Information Systems*, 17(1):57–77, 2008.

[77] Jaime Teevan, Daniel Ramage, and Merredith Ringel Morris. # twittersearch: a comparison of microblog search and web search. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 35–44. ACM, 2011.

[78] Fernando Tusell. Kalman filtering in r. *Journal of Statistical Software*, 39(2):1–27, 2011.

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

[80] Jilles Vreeken, Matthijs Van Leeuwen, and Arno Siebes. Krimp: mining itemsets that compress. *Data Mining and Knowledge Discovery*, 23(1):169–214, 2011.

[81] Chao Wang and Srinivasan Parthasarathy. Summarizing itemset patterns using probabilistic models. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 730–735. ACM, 2006.

[82] Ke Wang, Yu He, and Jiawei Han. Mining frequent itemsets using support constraints. In *Proceedings of the 26th International Conference on Very Large Data Bases*, pages 43–52, 2000.

[83] Ke Wang, Liu Tang, Jiawei Han, and Junqiang Liu. *Top down fp-growth for association rule mining.* Springer, 2002.

[84] Jianshu Weng and Bu-Sung Lee. Event detection in twitter. In *Proceedings of the 5th International AAAI Conference on Weblogs and Social Media*, volume 3, 2011.

[85] Sheng-Tang Wu, Yuefeng Li, and Yue Xu. Deploying approaches for pattern refinement in text mining. In *Data Mining, 2006. ICDM'06. Sixth International Conference on*, pages 1157–1161. IEEE, 2006.

[86] Sheng-Tang Wu, Yuefeng Li, Yue Xu, Binh Pham, and Phoebe Chen. Automatic pattern-taxonomy extraction for web mining. In *Web Intelligence, 2004. WI 2004. Proceedings. IEEE/WIC/ACM International Conference on*, pages 242–248. IEEE, 2004.

[87] Dong Xin, Hong Cheng, Xifeng Yan, and Jiawei Han. Extracting redundancy-aware top-k patterns. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 444–453. ACM, 2006.

[88] Dong Xin, Jiawei Han, Xifeng Yan, and Hong Cheng. Mining compressed frequent-pattern sets. In *Proceedings of the 31st international conference on Very large data bases*, pages 709–720. VLDB Endowment, 2005.

[89] Hui Xiong, P-N Tan, and Vipin Kumar. Mining strong affinity association patterns in data sets with skewed support distribution. In *Data Mining, 2003. ICDM 2003. Third IEEE International Conference on*, pages 387–394. IEEE, 2003.

[90] Xifeng Yan, Hong Cheng, Jiawei Han, and Dong Xin. Summarizing itemset patterns: a profile-based approach. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 314–323. ACM, 2005.

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

[92] Yiming Yang, Tom Pierce, and Jaime Carbonell. A study of retrospective and on-line event detection. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 28–36. ACM, 1998.

[93] Limin Yao, David Mimno, and Andrew McCallum. Efficient methods for topic model inference on streaming document collections. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 937–946. ACM, 2009.

[94] Mohammed J Zaki and Ching-Jui Hsiao. Charm: An efficient algorithm for closed itemset mining. In *2nd SIAM international conference on data mining*, volume 15, pages 457–73, 2002.

[95] Mohammed J Zaki, Srinivasan Parthasarathy, Mitsunori Ogihara, Wei Li, et al. New algorithms for fast discovery of association rules. In *3rd Intl. Conf. on Knowledge Discovery and Data Mining*, volume 20, pages 283–286, 1997.

[96] Yunyue Zhu and Dennis Shasha. Efficient elastic burst detection in data streams. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 336–345. ACM, 2003.