Very informal literature review

Younos Aboulnaga[1]

David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, ON, Canada
yaboulna@uwaterloo.ca

1 Introduction

There are three distinct areas of related work; frequent itemsets mining on streams, stream mining, and itemset interestingness. The first might seem to be a subset of the second, but it is not. Works in frequent itemsets mining on streams delve into the algorithms directly without any formalism, like I used to do. Their focus is on data structures, and they validate their work mainly by empirical results... and they do funny stuff! Works in stream mining are more rigorous but they focus on numeric variables and thus many of the works can't be directly applied to text mining where the variables are nominal. Also, only a few works can be applied to multivariate cases, and even fewer can be applied to cases when there are millions of variables. For example, the Adaptive Window framework proposed by Bifet (the book you saw with me) solves the problem of defining a different window size for each variable by doing something similar to dynamic histogram creation (a new window is defined every time a change happens; to decrease variance in the window). He solves this in better time than the existing algorithms (the formula is not handy right now but it's $t * log(something)$), and that's great... but can we do that for millions of items (and even more itemsets) with every time step? Finally, itemset interestingness is a form of ranking, after the fact that itemsets were mined.. and as far as I know it was never applied on streams where there are features derived from the evolution over time... I found very little in this area anyway!

2 Highlights

2.1 Frequent itemsets mining on streams

Most of the works assume a landmark window model; time 0 is when the mining started and with every time step new transactions are added, and some works use a fading/decay function to reduce the effect of earlier transactions. In most cases the main challenge is to approximate the counts of single items, then itemsets are created by candidate generation (Apriori style as opposed to FP-Growth style that avoids candidate generation by using an FP-Tree). Actually the generation of candidate itemsets by taking the power set of all items with enough support in each transaction is the bottle neck of frequent itemset mining, but I wasn't looking into it because I was focusing on the Parallel FP-Tree based algorithm. FP-Tree based algorithms struggle in stream environments because FP-Trees are not updatable (specially deletion which is needed for sliding windows to

work). However, I didn't care because I imagined that the mining itself is not important and assumed I would use the Map/Reduce based PFP-Growth algorithm to solve the problem by parallelization.. we just mine the whole window with every time step without caring that that is too much work.. I was just trying to know what would be a good length of the window to give good mining results. Actually I had modified the PFP-Growth algorithm to work on a landmark window, applying a fading function, or on a sliding window. However, I was trying to reduce the extra work by selecting the "head" terms that made sense (not just avoiding stop words, but also avoiding mining both Justin and Beiber or mining mundane words such as table). We can abandon all this and start doing the candidate generation, applying the heuristics that we will pick to reduce the number.

2.2 Stream Mining

I have looked mainly in burst detection, and in the general frameworks for stream mining. The burst detection problem seemed to have good solutions, either based on the state space model (Kleinberg 2002), or probability of occurrence (the stochastic process model we were discussing). However, I liked the block update model and I thought that the wavelet tree would be useful, but I couldn't find answers for how to select a window size or Clustering streams is also very related since it can achieve a very similar result. However, it mainly depends on maintaining sufficient statistics much in th

2.3 Sample papers in frequent itemsets mining on streams (cited by a survey in 2007)

Manku & Motwani (2002). Approximate frequency count over data streams: This is a lossy counting algorithm for single items. When it starts generalizing to itemsets it does funny stuff. First, it divides the stream into an arbitrary number of buckets so that β buckets can fit in memory, and it keeps only item(set)s which appear β times. This β is not related to support, and they just recommend that β should be large because otherwise *spurious* itemsets will make their way into D. Second, it generates all candidate itemsets and counts them separately in each batch, then adds the total support for all candidates from each batch. I don't see how they are calling this a one pass algorithm with efficient memory use.

Jin R, Agrawal G (2005). An algorithm for in-core frequent itemset mining on streaming data: That's all right, but it isn't really something amazing. It just counts 2-itemsets using a special counting technique that is efficient in memory usage, then uses the frequent 2-itemsets as seeds to larger ones. This increases the resource requirements of the counting steps, but reduces the candidates generated.

Lin C, Chiu D, Wu Y, Chen A (2005). Mining frequent itemsets from data streams with a time-sensitive sliding window: This is really disappointing! The "time-sensitive" window they *invented* is just a sliding window that has a start and end time, instead of being defined using a number of transactions. The most disappointing part is how they mine frequent itemsets on the sliding window. They don't mine the whole window, but rather smaller "blocks" and stitch the results together! Blocks are actually time steps,

so they should be small to reduce lag in the mining results, but they are forced to be large enough to make mining them useful.

Chi Y, Wang H, Yu PS, Muntz R (2004). Moment: maintaining closed frequent itemsets over a stream sliding window: This is a good piece of work. It uses a trie to store the transactions based on their lexicographic order, so it doesn't try to save memory, but this allows both addition and deletion. Of course, they prune branches based on the Apriori property. It is an also exact algorithm, so there is no need for correctness guarantees. However, their runtime does funny things and goes up and down as they decrease support; this makes me doubt the correctness of their implementation specially that they don't discuss the funny stuff. Most importantly, for our purposes, they assume a stationary stream where the number of nodes that change their type in the trie is low. The cost of re-*exploring* the branches that has become (in)frequent is not clear but it is the main cost of their algorithm. Also, the candidate generation algorithm is the good old power set method.

Efficient Computation of Frequent and Top-k Elements in Data Streams (2005): Another incarnation of the idea of using a relaxed support (a lower threshold to keep itemsets that almost made it), and some theorems and lemmas to update the itemsets of the previous window to become recent. The big problem is that the itemsets are generated using a batch algorithm at each time t, not even a landmark window algorithm.. and they use a relaxed support for that!

2.4 Sample papers in burst detection

Parameter Free Bursty Events Detection in Text Streams (2005): Models the arrival of each word (called feature) as a binomial distribution, with probability equals to the mean of averages across consecutive batches. Filters stop words by normalizing the number of documents in a the batch interval, and using a heuristic. words whose