Support over time

Younos Aboulnaga[1]

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

1 Introduction

Minimum support is a good constraint for Frequent Itemsets Mining (FIM) because it is anti-monotonic; if an itemset has support lower than the *frequent* itemsets support threshold then no superset of it can be *frequent*. Pruning based on anti-monotone constraints works well in batch mode, but in case of online mining of a timeseries this could cause pruning itemsets that are growing to become frequent when they are still infrequent. This is inevitable if the mining algorithm is run on non-overlapping windows, because regardless of the window size it can always happen that the window borders separate the occurrences of an itemset into two sets of occurrences each of size less than the minimum support. Therefore, the mining algorithm has to be run on overlapping windows, which means that it will repeat some work whose overhead, in terms of processing power and storage of the results, must be minimized.

The FP-Growth family of algorithms for FIM works by mining frequent itemsets from *conditional pattern bases*, which are transactions including a certain frequent item. The choice of items for which to create the conditional pattern base and proceed with the mining can be done using a metric other than support. To minimize the processing power needed for repeated work it is possible to mine only the conditional pattern bases of items that occurred in the last time step. The support can't be limited to the occurrences in transactions within the time step, and can't be extended to go all the way back to the beginning of the time series. In the next section we propose different measures of support that are related to time. However, it is note worthy that all items occurring within the time step have to be processed, even if they occur only once.

Since the real time constraint is not very strict for our purposes, we adopt the block update model. The time series is divided into blocks of similar length, and the window size must be a multiple of this block length. Counts of items are kept for each block, and thus counts of any window can be very quickly calculated by summing the counts of the blocks forming it. Block length should not be confused with the *time step*, which is the amount of time that the window is advanced every time it is moved. Naturally, the time step should also be a multiple of the block size. The Nyquist-Shannon sampling theorem states that frequency of sampling must be at least double the highest frequency component in the signal, to avoid loss of information. On the contrary, selecting a block of length B entails that the lowest frequency itemset that would be mined from one block is one that arrives at a rate of $1/2B$, using the lowest possible support of 2.

2 Support over time

The most basic way to relate support and time is to divide the support by the interval of time over which the occurrences happened; which we call *velocity*. This spreads the occurrences uniformly over time thus increasing the effect of spikes of interest, such as a spike of interest about a celebrity doing something mundane like appearing in a TV show wearing something strange. The change in support slope from one time step to the next indicates if interest is increasing or decreasing; thus, its *acceleration*. Actually, the current Twitter trending topics are elected using a variation of this measure along with some novelty measure. This choice to favour topics with high increase in the volume of conversations causes topics of sustained interest to be buried under topics of momentary interest. We will use the sign of the acceleration as a binary feature to indicate if interest is rising or falling, disregarding volume which will be captures by other features.

To decrease the effect of high intensity spikes a curve fitting technique can be used, smoothing the values between discrete points in time instead of using the support directly. Examples of curve fitting techniques are Haar Wavelets, Discrete Time Fourier Transform (DTFT), Singular Value Decomposition (SVD), Piecewise Constant Approximation (PCA) and dynamic histograms. Each of these methods has its assumptions and should be used to achieve a certain optimization criterion. The most intuitive criterion is that of dynamic histograms which divides the time series into bins to minimize variance, but unfortunately calculating this requires a scan on the whole timeseries with every update. Other methods can be calculated online and incrementally, but results in a compressed version of the data that is hard to work with. For example calculating the correlation in DTFT space is possible, but arriving at the way to do it was a contribution in itself [**?**]. Generally working with such representations would require decompressing it through calculating the value at the time point needed, and thus requires more processing power. However, storage space is cheap and we aim to reduce the processing power needed for mining the stream. Also, the shape of the occurrences plot with time is not important to us - as it is in cases where curves are matched to answer queries. Therefore, we leave using curve fitting techniques as a last resort, and explore alternatives.

One alternative is to measure the length of time to go backwards until support is accumulated; *support lag*. Using this metric is similar to going back in time to check if items that are infrequent in the latest time step would actually be frequent if the time step were longer; thus it is unaffected by the choice of step size. This metric is also anti-monotonic, and independent of the window size and model (sliding or other). The conditional pattern base of each item should go back in time the same number of time steps as its support lag, and we set an arbitrary upper bound on the number of steps it is permitted to go back to avoid loading the whole time series. Since different pattern bases go back in time different lengths, the timestamp has to be kept with the input after loading it from disk. Keeping the time stamp of each transaction in memory is a very large overhead that will greatly affect the scalability of the algorithm. Another complication, specific to the FP-Growth algorithm, arises from the use of prefix trees to compactly represent the conditional pattern base. This allows loading large databases into memory and alleviates the effect of the high memory requirements of FP-Growth. However, since prefix trees cannot be updated once balanced, it is not possible to reuse of the input already loaded in memory by appending to it as input from time steps further back is loaded. Therefore the input has to be read from disk for each lag length, and thus if a lag length is specific to only a few items they should use the input loaded for a longer lag length.

Support lag is still sensitive to the effect of spikes of high intensity. The effect of the spike decreases as it becomes older but its intensity still affects the length of time it will remain of high importance. We believe we should give more importance to more recent itemsets even if their intensity is much lower than the spiky itemset that their support lag is still longer than the old spike. This increases diversity and prevents the mining results from being stagnant for long periods because of high spikes, which would cause revisiting users to get bored and abandon the service. We achieve this by counting the *boolean arrival rate*; that is, disregarding the amount of support at each time step and just counting the number of time steps when the itemset occurred at all. This measure is also anti-monotonic, but it depends on the window size and assumes a block update model for counts. Dependence on window size is alleviated by using this metric along with the support lag, where the window size will be determined by the support lag. This measure can be considered a smoothed version of the time since last occurrence, which would jump to 0 as soon as an occurrence happen and gradually increase.

Once the window size is determined for a specific item, the average, variance, skewness, and any other statistical measures can be calculated over this window. We defer the use of such measures because we first need to show that they are anti-monotonic.

3 History of counts

It was mentioned that there will be an upper bound on the length of the window. Block that are too old to be included in a window can be merged into a history block. The counts in the history can also be decayed with every time step, for example using exponential moving average: $H_t = \alpha * C_t + (1 - \alpha) * H_{t-1}$ where H is the history, C is the count of the latest block and α is a constant smoothing factor between 0 and 1.

Counts from the history block can be used to smooth counts of items in each window before they are used in the metrics above. This gives more support to items that are always of high support when they appear, such as the names of celebrities or political parties. The smoothing can be done using Jelinek and Mercer smoothing: $C = \lambda * C + (1 - \lambda) * H$ where H is the history count, C is the count of the latest block and λ is a constant smoothing factor between 0 and 1.

4 Measures not based on support

There are other anti-monotonic measures which can be used, but don't provide an intuitive way to define a threshold. For example, Normalized Mutual Information (NMI) is anti-monotonic because the Point-wise Mutual Information of extra items added to the itemset with each of the existing items is positive. Also, NMI has values between 0 and 1, so it is actually a very good measure. However, it is difficult to decide whether an itemset of NMI 0.7 is better than itemset of NMI 0.5 and thus decide on a threshold. Actually, maybe an itemset with a slightly lower NMI would be better than that of a high NMI because the later is just predictable. For example, (Justin, Bieber, Dead) would have a lower NMI than (Justin, Bieber, Sing) even though it is more interesting. Other examples are the measures from the Association Rules Mining literature such as lift, all-confidence and coherence. Those metrics are specifically tailored to favour itemsets with high confidence (*stronger*), which could also mean itemset with lower surprise.

We prefer to use support based itemsets because they leverage the wisdom of the crowd. As for other measures based on inter-relation between items (such as coherence or NMI), we could use them to detect Multi Word Expressions (MWE) or Named Entities (NE), and thus merge them into one item or at least avoid reporting them as an itemset in their own. Actually, as these measures would not be used for pruning but only for filtering results, then we can use measures that are not anti-monotonic such as Pearson's Correlation Coefficient and other contingency table metrics. However, this should be done as a preprocessing phase rather than after mining the itemsets. This is easy to be done on pairs of items, and should be able to provide language independent stop word filtering besides detecting MWEs.

5 Conclusion

We showed that for mining frequent items from a stream there must be an overlap of the processed windows, and introduced the block update framework to avoid repeating the counting step. We introduced metrics based on support that are related to time; namely, *acceleration*, *support lag*, and *boolean arrival rate*. We have also argued that the proposed measures are more suitable for our purposed than other famous measures or the use of curve fitting techniques. We also showed the importance of keeping history and proposed how to keep it.