# Fast algorithms for Mining Emerging Patterns

James Bailey, Thomas Manoukian and Kotagiri Ramamohanarao
Department of Computer Science & Software Engineering
The University of Melbourne, Australia
{jbailey,tcm,rao}@cs.mu.oz.au

**Abstract.** Emerging Patterns are itemsets whose supports change significantly from one dataset to another. They are useful as a means of discovering distinctions inherently present amongst a collection of datasets and have been shown to be a powerful technique for constructing accurate classifiers. The task of finding such patterns is challenging though, and efficient techniques for their mining are needed.
In this paper, we present a new mining method for a particular type of emerging pattern known as a jumping emerging pattern. The basis of our algorithm is the construction of trees, whose structure specifically targets the likely distribution of emerging patterns. The mining performance is typically around 5 times faster than earlier approaches. We then examine the problem of computing a useful subset of the possible emerging patterns. We show that such patterns can be mined even more efficiently (typically around 10 times faster), with little loss of precision.

## 1 Introduction

Discovery of powerful distinguishable features between datasets is an important objective in data mining. Addressing this problem, work presented in [6] introduced the concept of *emerging patterns*. These are itemsets whose support changes significantly from one dataset to another. Because of sharp changes in support, emerging patterns have strong discriminating power and are very useful for describing the contrasts that exist between two classes of data. Work in [11] has shown how to use them as the basis for constructing highly accurate data classifiers. In this paper, we focus on mining of a particular type of emerging pattern called a *jumping emerging pattern* (JEP). A JEP is a special type of emerging pattern, an itemset whose support increases abruptly from zero in one dataset, to non-zero in another dataset. Due to this infinite increase in support, JEPs represent knowledge that discriminates between different classes of data more strongly than any other type of emerging pattern. They have been successfully applied for discovering patterns in gene expression data [12].

Efficient computation of JEPs remains a challenge. The task is difficult for high dimensional datasets, since in the worst case, the number of patterns present in the data may be exponential. Work in [6] introduced the notion of a border for concisely representing JEPs. Yet even using borders, the task still has exponential complexity and methods for improving efficiency are an open issue. With the volume and dimensionality of datasets becoming increasingly larger, development of such techniques is consequently crucial. Indeed for large datasets, approximation methods are also necessary, to ensure tractability.

In this paper, we describe algorithms for computing JEPs that are 2-10 times faster than previous methods. Our approach has two novel features: The first is the use of a new tree-based data structure for storing the raw data. This tree is similar to the so-called frequent pattern tree, used in [9] for calculating frequent itemsets. However, there are significant differences in the kinds of tree shapes that promote efficient mining and interesting new issues and tradeoffs are seen to arise. The second feature is the development of a mining algorithm operating directly on the data contained in the trees. The mining of emerging patterns is unlike (and indeed harder than) that of frequent itemsets. Monotonicity properties relied on by algorithms such as a-priori do not exist for JEPs and thus our algorithm requires greater complexity than the techniques in [9].

We then look at the problem of mining only a subset of the JEPs using approximate thresholding techniques. We outline methods which can achieve further speedups from 2-20 times faster and demonstrate that a small number of patterns can still provide sufficient information for effective classification.

**Related Work**: Emerging patterns first appeared in [6], which also introduced the notion of the border for concisely representing emerging patterns. Unlike this paper, no special data structure was used for mining the JEPs. Techniques for building classifiers using JEPs, whose accuracy is generally better than state-of-the art classifiers such as C4.5 [16] appeared in [11]. Emerging patterns are similar to version spaces [14]. Given a set of positive and a set of negative training instances, a version space is the set of all generalisations that each match (or are contained in) every positive instance and no negative instance in the training set. In contrast, a JEP space is the set of all item patterns that each match (or are contained in) one or more (not necessarily every) positive instance and no negative instance in the set. Therefore, the consistency restrictions with the training data are quite different for JEP spaces.

Work in [9] presented a technique for discovering frequent itemsets (which are useful in tasks such as mining association rules [1]). The primary data structure utilised was the Frequent Pattern Tree (**FP-tree**), for storing the data to be mined. The trees we use in this paper are similar but important new issues arise and there are also some significant differences. Given that we are mining emerging patterns and there are multiple classes of data, tree shape is a crucial factor. Unlike in [9], building trees to allow maximum compression of data is not necessarily desirable for mining emerging patterns and we show that better results are obtained by sacrificing some space during tree construction .

Recent work in [8] also uses trees for calculation of emerging patterns. The focus is different, however, since the algorithm is neither complete nor sound (i.e. it does not discover all JEPs and indeed may output itemsets which aren't actually JEPs). In contrast, work in this paper focuses on both i) Sound and complete mining of JEPs and ii) Sound but not complete JEP mining.

The emerging pattern (EP) mining problem can also be formulated as discovering a theory that requires the solution to a conjunction of constraints. Work in [5, 10] defined three constraint types; i) $f \leq p$, $p \leq f$, $\neg(f \leq p)$ and $\neg(p \leq f)$ ii) $freq(f, D)$ iii) $freq(f, D_1) \leq t$, $freq(f, D_2) \geq t$. Using the first and third, JEP mining for some class $D_i$ with reference $D_j$ can be expressed as; $solution(c_1 \wedge c_3)$ where $f$ is a JEP, $p \in D_i$ and $t = 0$.

Other methods for mining EPs have relied upon the notion of borders, both as inputs to the mining procedure in the form of large borders and as a means of representing the output. [7] employed the Max-Miner [2] algorithm whilst work in [15] is also applicable in generating large borders. JEP mining procedures do not require such sophisticated techniques. Rather than generate large borders, horizontal borders [6] are sufficient. Work in [13] restricts border use to subset closed collections and allows minimal elements that do not appear in the base collections. The borders used in this paper reflect interval closed collections and contain only minimal elements derived from the base collections.

An outline of the remainder of this paper is as follows. In section 2 we give some necessary background and terminology. Section 3 presents the tree data structure we use for mining JEPs and describes several variations. Section 4 gives the algorithm for complete mining of JEPs using this tree and Section 5 gives an experimental comparison with previous techniques. Section 6 then discusses approximate methods for mining a subset of the patterns present. Finally, in section 7 we provide a summary and outline directions for future research.

## 2 Background and Terminology

Assume two data sets $D_1$ and $D_2$, the growth rate of an itemset $i$ in favour of $D_1$ is defined as $\frac{support D_1(i)}{support D_2(i)}$. An Emerging Pattern [6] is an itemset whose support in one set of data differs from its support in another. Thus a $\rho$ Emerging Pattern favouring a class of data $C$, is one in which the growth rate of an itemset (in favour of $C$) is $\geq \rho$. This growth rate could be finite or infinite. Therefore, we define another type of pattern, known as a *jumping emerging pattern* (JEP), whose growth rate must be infinite (i.e. it is present in one and absent in the another). JEPs can be more efficiently mined than general emerging patterns and have been shown to be useful in building powerful classifiers [11]. We will illustrate our algorithms for mining JEPs assuming the existence of two datasets $D_p$ (the positive dataset) and $D_n$ (the negative dataset). The mining process extracts all patterns (i.e. itemsets) which occur in $D_p$ and not in $D_n$.

A *border* [6] is a succinct representation of some collection of sets. [6] also showed that the patterns comprising the left bound of the border representing the JEP collection are the most expressive. Therefore, our procedures will be referring to mining the left bound border of the JEP collection.

In previous work, mining of JEPs used a cross-product based algorithm known as *border-diff* [6]. It takes as input some transaction in $D_p$ from which one wishes to extract JEPs (the positive transaction) and a set of transactions from $D_n$ (negative transactions). It's output is then all JEPs from this positive instance (i.e. all subsets of the positive transaction which do not appear within any negative transaction). We will make use of the *border-diff* algorithm, but use the structure of the tree in order to determine when it should be called and what input should be passed to it. This results in significant performance gains.

Classification using JEPs is described in [11]. Initially all JEPs for each of the classes are computed - *observe this needs to be done once only for the datasets* (the training time). Then, given some test instance, a score is calculated for each class. This score is proportional to the number of JEPs (from the class being examined) contained within the test. Typically the contribution of an individual JEP to the overall score is some function of its support (hence JEPs with high support have a greater influence on classification). The test instance is deemed to match the class with the highest overall score.

## 3 Trees

The tree based data structure we use for mining JEPs is based on the frequent pattern tree [9]. Since we are dealing with several classes of data, each node in the tree must record the frequency of the item for each class. Use of a tree structure provides two possible advantages:

- when multiple transactions share an itemset, they can be merged into individual nodes with increased counts. This results in compression proportional to the number of itemsets which share some prefix of items and the length of the prefix. Such compression can allow the data structure to be kept in memory and thus accessed faster, rather than being stored on disk.
- Different groupings of positive transactions (those from $D_p$) and negative transactions (those from $D_n$) become possible. The efficiency of mining is highly dependent on how this grouping is done. We now examine how transactions are ordered to achieve different groupings.

In choosing an appropriate ordering for the items contained within the itemsets being inserted into the tree, we have the following 2 aims i) To minimise the number of nodes in the tree and ii) To minimise the effort required in traversing the tree to mine JEPs. [9] addressed the first of these in the context of computing frequent itemsets. However, for mining JEPs, we will see that the second is the more important. We have investigated six types of orderings.

*Frequent tree ordering.* Same as in [9]. Take each item and find its probability in the set $(D_p \cup D_n)$. Items are ordered in descending probability. This ordering aims to minimise the number of nodes in the tree.
*Ratio tree ordering* and *inverse ratio tree ordering.* Let the probability of an item in $D_p$ be $p_1$ and its probability in $D_n$ be $p_2$. For $p = p_1/p_2$, order items in descending value of $p$. The intuition here is that we expect JEPs to reside much higher up in the tree than they would under the frequent tree ordering and this

will help limit the depth of branch traversals needed to mine them. The inverse ratio ordering is just the reverse of this ordering.

*Hybrid ordering*. A combination of the ratio tree ordering and the frequent tree ordering. First, calculate both the ratio tree ordering and frequent tree ordering. For a given percentage $\alpha$, the initial $\alpha$ items are extracted from and ordered according to the ratio ordering. All items not yet covered are then ordered according to the frequent ordering. The hybrid ordering thus produces trees which are ordered like a ratio tree in the top $\alpha$ segment and like a frequent tree in the bottom $1-\alpha$ segment. The intuition behind it is that trees are created which possess both good compression characteristics as well as good mining properties.

*Least probable in the negative class ordering* (LPNC). Let the probability of an item in $D_n$ be $p$. Items are ordered in ascending value of $p$. The intuition behind this ordering is similar to that for the ratio ordering, JEPs are likely to occur higher up in the tree, since the quantity of nodes higher up in the tree containing zero counts for the negative classes is greater.

*Most probable in the positive class ordering* (MPPC). Let $p$ be the probability of an item in $D_p$. Items are ordered in descending value of $p$ (first item has highest probability). The intuition here is that by placing nodes higher up in the tree (in accordance with their frequency in the positive class), then if the datasets are inherently dissimilar, we are more likely to find JEPs in the tree's upper regions.

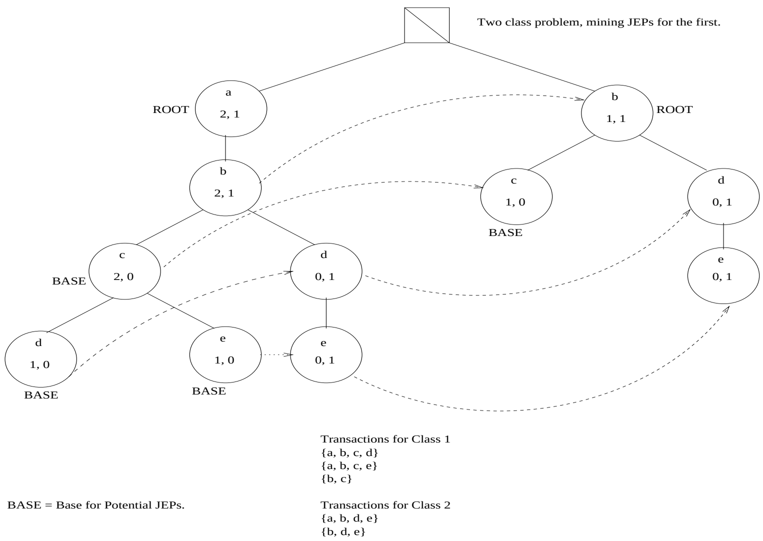
## 4 Tree-Based JEP Mining

We now describe our tree mining procedure. It uses a core function called *border-diff* [6] with format *border-diff(positive_transaction,vector negative_transactions)*. This function returns the set of JEPs present within *positive_transaction* with reference to the list of *negative_transactions*. i.e. All subsets of *positive_transaction* which don't occur within a member of the negative transaction list. The efficiency of border-diff is dependent upon the number of negative transactions which are passed to it, their average dimensionality and the dimensionality of the positive transaction. Our tree mining procedure makes use of this function in a way aimed to reduce all of these parameters.

The initially constructed tree contains a null root node, with each of its children forming the root of a subtree referred to hereafter as a *component tree*. For each component tree, we perform a downwards traversal of every branch. Looking during traversal for nodes which contain a non-zero counter for the class for which we are mining JEPs, and zero counters for every other class ( such nodes are called *base* nodes). The significance of these nodes is that the itemset spanning from the root of the branch to the base node is unique to the class being processed. This itemset is therefore a *potential JEP* and hence any subset of this itemset is also potentially a JEP. The importance of base nodes is not simply that they identify potential JEPs, but they also provide a means of partitioning our problem. By considering all branches which share some root node (i.e. all branches of some component tree) and some additional node, (a base node), we can isolate all other transactions containing these two items. Using this set of transactions as the basis for a sub mining problem provides great flexibility in the inputs which are provided to the border-diff function. We have some control over the number of negative transactions, their cardinality and the cardinality of the positive transaction, all key determinants in performance. After identifying a potential JEP, we gather up all negative transactions that are related to it (i.e. share the root and base node). These negative transactions can be obtained using side links which join all nodes representing the same item. Border-diff is then invoked to identify all actual JEPs contained within the potential JEP. After examining all branches for a particular component tree, we re-insert them back into the remaining component trees, having removed the initial node of each. The following pseudo-code gives a high-level outline of the mining algorithm.

```
Component_Trees CTs = build_tree();
For each component tree, ct of CTs
   For each branch b of ct
      if(b is a potential JEP)
         border_diff(b, negative_transactions);
   relocate_branches(ct, CTs);
```

*Example:* Consider the following tree.




Beginning at the leftmost component tree (with root $a$) we traverse its children looking for base nodes for potential JEPs. In this case there are three, $\{c$, $d$ and $e\}$. On finding $c$ we attempt to gather the associated negative transactions (with reference the root $a$ and base $c$), in this case there exist no such transactions. $\{a, c\}$ is output as a JEP. On encountering base node $d$, we are able to collect the negative transaction $\{a, b, d\}$, border-diff is called with $\{\{a, b, c, d\}, \{a, b, d\}\}$. Finally on discovering $e$ as a base node and collecting the associated negative transactions, we call border-diff with the input $\{\{a, b, c, e\}, \{a, b, d, e\}\}$. The component tree with $a$ as the root has now been processed and its transactions are re-inserted with the $a$ element removed. Mining would then continue on this next component tree.

By examining potential JEPs only, we ensure that the only patterns we mine are JEPs. The fact that we examine every potential JEP with reference to every component tree (and thus every item in the problem), ensures completeness. In all cases the number of component trees is equal to the number of unique items in the database. Component tree traversal in the worst case requires visiting a number nodes equal to the number of attributes in the problem, per branch. Subsequent collation of all negative transactions, in the worst-case, requires gathering $|DB|$-1 transactions.

## 5 Performance of Tree Mining

The following table displays the performance of our JEP mining procedure using various types of tree orderings. The Original column refers to the implementation used to mine JEPs using previous published work. Times recorded are user times and are given in seconds. The column headed hybrid, represents a hybrid tree with the the first thirty percent of items ordered by ratio, the remainder by absolute frequency. The column labelled speedup compares the hybrid tree with the original approach. The data sets used were acquired from the UCI Machine Learning Repository [3]. All experiments were performed on a 500MHz Pentium III PC, with 512 MB of memory, running Linux (RedHat).

| Dataset | MPPC-tree | FP-tree | LPNC-tree | iRatio-tree | Ratio-tree | Hybrid | Original | Speedup |
|---|---|---|---|---|---|---|---|---|
| mushroom | 38.57 | 27.59 | 37.35 | 17.32 | 16.77 | **13.66** | 138.45 | 10.13 |
| census | 497.17 | 459.65 | 385.51 | 221.84 | 214.23 | **182.84** | 1028.00 | 5.62 |
| ionosphere | 83.63 | 69.75 | 59.04 | 21.91 | 21.15 | **19.07** | 86.74 | 4.55 |
| vehicle | 5.86 | 6.01 | 5.82 | 3.92 | 3.85 | **3.20** | 5.33 | 1.67 |
| german | 140.84 | 138.46 | 94.61 | 50.75 | 47.54 | **38.59** | 131.37 | 3.40 |
| segment | 68.16 | 66.86 | 65.65 | 45.52 | 41.42 | **35.60** | 71.99 | 2.02 |
| hypothyroid | 3.65 | 3.03 | 3.84 | 1.99 | 1.89 | **1.77** | 1.79 | 1.01 |
| pendigits | 687.32 | 719.09 | 631.62 | 563.05 | 560.92 | **507.55** | 2951.26 | 5.81 |
| letter-rec | 3632.92 | 3537.37 | 3199.32 | 1815.82 | 1700.91 | **1485.20** | 6896.07 | 4.64 |
| soybean-l | 321.28 | 490.70 | 457.45 | 135.46 | 127.03 | **74.15** | 611.50 | 8.25 |
| waveform | 4382.27 | 4391.85 | 2814.14 | 2794.07 | 2779.66 | **2560.16** | 26180.50 | 10.23 |
| chess | 811.51 | 871.91 | 865.30 | 245.96 | 238.95 | **90.62** | 358.62 | 3.96 |

We see that using our tree mining algorithm, significant savings are achieved over the original method. We now rank the various types of trees:

1. Hybrid tree - always the fastest performer. The parameter $\alpha$ was set to 30% (we conducted other experiments for alternative values, with this choice giving the best overall times). The performance gains are typically around 5 times faster than the original method.

2. Ratio and inverse ratio trees.
3. LPNC tree.
4. Frequent pattern tree and MPPC tree. Each with similar running times.
5. Original method of [6]. The slowest technique serving as a benchmark for the tree-based methods.

We can make a number of observations about these results:

1. The relative memory usage among the various trees will of course vary between the various datasets. However, from additional data not included here due to lack of space, the ranking (from least average tree size to largest average tree size) is i) Frequent pattern tree, ii) MPPC tree, iii) Hybrid tree (when using $\alpha = 30\%$), iv) Ratio and inverse ratio trees, v) LPNC tree. The frequent pattern tree uses the least memory, but takes the longest time to mine. This would indicate that tree size is not a dominant factor in determining the mining effort needed for JEPs. This is in contrast to the work in [9], where the main objective in using frequent pattern trees was to reduce tree size, so that frequent itemset calculation could be carried out entirely within main memory.
2. The LPNC and MPPC trees are consistently worse than the ratio tree variants. The orderings for these trees only consider one of the positive and negative datasets and thus there is less overlap between positive and negative transactions. Consequently more potential JEPs will need testing.
3. Ratio and inverse ratio trees are superior for mining than frequent pattern trees. We believe this is because ratio/inverse ratio tree structure results in fewer potential JEPs needing to be tested. As the component trees are processed according to the ratio order, singleton items which have high support in one class and low support in the other are pruned earlier. Such items are strong differentiators between the classes. Thus the tendency is for positive and negative transactions to have greater overlap as processing proceeds and hence fewer potential JEPs (especially duplicate ones) will be tested by border-diff.
4. The hybrid tree is faster to mine than the pure ratio tree. We conjecture that frequent trees allow items which have high support in both classes to be pruned earlier. This in turn means that there will be fewer and fewer transactions per component tree as processing proceeds, also decreasing the number of required border-diff calls. Combining this property of frequent pattern trees with the properties of ratio trees, results in a tree that is very fast to mine. It is the subject of further research to more deeply analyse the interplay of factors here.

Overall, these tree based methods are significant improvements on previous methods for mining emerging patterns. Nevertheless, for datasets of very high dimensionality, the running time of a complete mine may still be prohibitive. This motivates a supplementary approach which mines only a subset of the complete set of JEPs.

## 6 Mining the Highest Support JEPs Using Thresholds

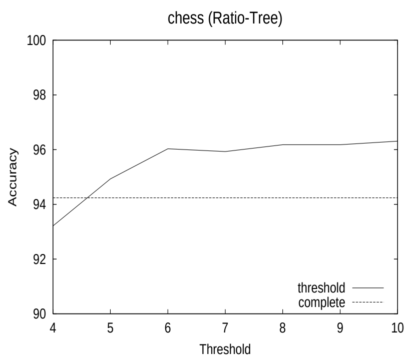
We now examine a method which sacrifices completeness of JEP mining in return for faster computation.

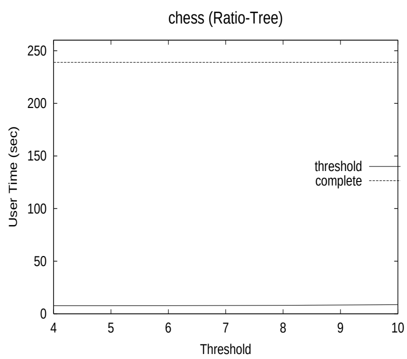
Since completeness will no longer hold, we wish the JEPs we mine to be "important ones", i.e. they should have high support. Examining the characteristics of JEPs of high support, it should be clear that in general shorter JEPs will experience greater support levels than longer JEPs.

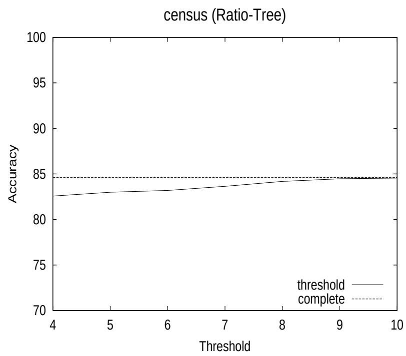
We therefore aim to to mine as many of the short JEPs as possible. Our mining procedure is now modified to only identify potential JEPs whose length is below a specified threshold. Any potential JEPs above this threshold will not be examined by the *border-diff* function to see if actual JEPs are present. Whilst this method will not necessarily ensure mining of only the highest support JEPs, it presents an attractive alternative due to the relatively small computation time and its algorithmic simplicity.

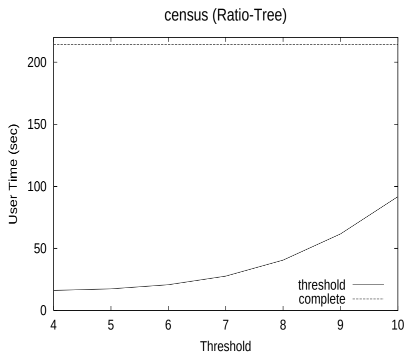
Applying such thresholds means that the number of times the *border-diff* function is called is drastically reduced, as well as ensuring that when used it is not too expensive, since we have complete control over one of the factors, the cardinality of the itemsets passed to it.

The success of such a strategy is dependent upon how many short JEPs actually reside within the threshold one chooses to impose. Sometimes application of the threshold may mean that some short JEPs may be lost. e.g. A potential JEP $J = \{a, b, c, d, e, f, g, h, i, j\}$ in a ten attribute problem (where $a$ is the root item and $j$ is the **base** item) may actually contain the following JEPs; $\{a, b, j\}$, $\{a, c, j\}$ and $\{a, e, j\}$. However, choosing a threshold of four for this example would eliminate the possibility of discovering these JEPs. The following diagrams now illustrate the merit of various threshold values applied to a ratio tree. The four graphs illustrate the variance of accuracy and user time versus threshold for two datasets. The accuracy and time of JEP mining without thresholds (complete mining) is provided as a reference. For these examples we can see that as thresholds increase, accuracy converges relatively quickly and user time increases relatively slowly.






| Dataset | pt=4 | pt=5 | pt=6 | pt=7 | pt=8 | pt=9 | pt=10 | original |
|---|---|---|---|---|---|---|---|---|
| mushroom | 6.03 | 6.11 | 6.28 | 6.48 | 6.82 | 7.38 | 8.19 | 138.45 |
| census | 16.23 | 17.46 | 20.78 | 27.75 | 40.61 | 61.71 | 91.75 | 1028.00 |
| ionosphere | 1.37 | 1.43 | 1.45 | 1.56 | 1.67 | 1.83 | 1.99 | 86.74 |
| vehicle | 0.96 | 0.99 | 1.00 | 1.07 | 1.19 | 1.33 | 1.50 | 5.33 |
| german | 1.53 | 1.64 | 1.86 | 2.28 | 2.93 | 3.86 | 5.19 | 131.37 |
| segment | 8.44 | 8.86 | 9.71 | 11.08 | 12.92 | 15.34 | 18.15 | 71.99 |
| hypothyroid | 1.16 | 1.21 | 1.22 | 1.25 | 1.27 | 1.30 | 1.35 | 1.79 |
| pendigits | 63.25 | 72.53 | 88.32 | 111.37 | 142.30 | 181.77 | 230.28 | 2951.26 |
| letter-rec | 249.14 | 256.49 | 275.44 | 319.41 | 396.38 | 510.47 | 659.98 | 6896.07 |
| soybean-l | 13.67 | 13.68 | 13.69 | 13.75 | 13.83 | 14.09 | 14.24 | 611.50 |
| waveform | 18.09 | 21.31 | 30.53 | 47.13 | 72.71 | 110.47 | 165.44 | 26180.50 |
| chess | 7.64 | 7.66 | 7.75 | 7.83 | 8.02 | 8.33 | 8.68 | 358.62 |

timing-pure thresholding (ratio-tree)

| Dataset | pt=4 | pt=5 | pt=6 | pt=7 | pt=8 | pt=9 | pt=10 | complete |
|---|---|---|---|---|---|---|---|---|
| mushroom | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 |
| census | 82.57 | 82.99 | 83.19 | 83.64 | 84.18 | 84.47 | 84.55 | 84.60 |
| ionosphere | 90.32 | 90.27 | 91.11 | 90.83 | 91.13 | 90.85 | 90.85 | 88.32 |
| vehicle | 46.23 | 48.35 | 49.66 | 52.27 | 53.10 | 55.93 | 58.52 | 65.11 |
| german | 71.50 | 72.10 | 72.90 | 73.70 | 73.90 | 74.40 | 74.20 | 74.70 |
| segment | 94.67 | 94.80 | 94.76 | 94.98 | 94.89 | 94.46 | 94.59 | 93.81 |
| hypothyroid | 97.60 | 97.72 | 97.76 | 97.76 | 97.76 | 97.76 | 97.88 | 98.48 |
| pendigits | 93.17 | 94.72 | 95.30 | 95.57 | 95.72 | 95.92 | 95.95 | 96.16 |
| letter-rec | 63.07 | 76.90 | 82.12 | 84.34 | 85.28 | 86.63 | 87.85 | 92.21 |
| soybean-l | 82.73 | 83.43 | 83.92 | 85.66 | 84.71 | 84.88 | 83.06 | 84.92 |
| waveform | 70.48 | 77.62 | 79.38 | 80.12 | 80.16 | 80.32 | 80.44 | 82.94 |
| chess | 93.21 | 94.93 | 96.03 | 95.93 | 96.18 | 96.18 | 96.31 | 94.24 |

accuracy-pure thresholding (ratio-tree)

The two tables above provide more complete information on mining behaviour using thresholds. We see that mining with a threshold value of 4 is substantially faster than mining the complete set of JEPs using a ratio tree.

Classification accuracy is degraded for three of the datasets (Vehicle, Waveform and Letter-recognition) though. Analysis of the vehicle and chess datasets aid in explaining this outcome (supporting figures have been excluded due to lack of space). It is clear that classification accuracy is dependent upon finding patterns that strongly discriminate and at the same time are strongly representative of the instances of a particular class. The number of patterns one finds can be viewed as an indicator of how well a class' instances can be differentiated from instances of another class. The importance of each pattern, as a representative of the class, can be measured as its support. The discrepancy in classification accuracy of the vehicle dataset, from a threshold of 4 to 10, may be accounted for by a large difference in the number of patterns found for two of its classes (*saab* and *opel*) between threshold 4 and threshold 10. The average support of patterns is roughly the same at each of these threshold values. In contrast, for the chess dataset, we don't experience such marked fluctuation in classification over the thresholds, since the balance between number of JEPs and their average support is kept constant as threshold value increases. A threshold of 4 for chess has fewer number of JEPs, but their average support is greater, while a threshold of 10, has lower average support JEPs, but possesses a greater number of them. Clearly both factors are important and further work is required to determine their precise relationship(s).

Isolating behaviour with a threshold value of 10, we see that the improvement in mining time is not as great, but still substantial (around 2-10 times faster than mining the complete set with a ratio tree, and around 2-158 times faster than the original method). Classification accuracy is also the same (only about 1% difference) as classification using the full set of JEPs.

Adopting a threshold of 10 then, is a useful means of speeding up mining without appreciable loss of precision. For further work, it would be interesting to see whether it is possible to automatically choose different thresholds according to the characteristics of the input datasets or to develop more complex thresholding criteria.

## 7 Summary and Future Work

In this paper we have developed efficient algorithms to mine emerging patterns. We presented a mining algorithm that used tree data structures to explicitly target the likely distribution of JEPs. This achieved considerable performance gains over previous approaches. We also looked at methods for computing a subset of the possible JEPs, corresponding to those with the highest support in the dataset. These approximate methods achieved additional performance gains, while still attaining competitive precision. For future work we intend to:

- Extend our techniques to handle finite growth rate emerging patterns.
- Investigate further ways of ordering trees and investigate whether methods that have been developed in other machine learning contexts (e.g. for ranking

attributes or splitting in decision trees) can help.
- Develop analytical justification for the hybrid tree's performance.
**Acknowledgements**: This work was supported in part by an Expertise Grant from the Victorian Partnership for Advanced Computing.

## References

[1] R. Agrawal and R. Skrikant. Fast algorithms for mining association rules. In Proceedings of the Twentieth International Conference on Very Large Data Bases, Santiago, Chile, 1994. p. 487-499.
[2] Bayardo, R. J. Efficiently Mining Long Patterns from Databases. SIGMOD 1998.
[3] C. L. Blake and P. M. Murphy. UCI Repository of machine learning [www.ics.uci.edu/~mlearn/MLRepository.html]
[4] C. V. Cormack, C. R Palmer and C. L. A. Clarke. Efficient construction of large test collections. In Proceedings of the Twenty-first Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Melbourne, Australia, 1998. p. 282-289
[5] De Raedt, L., Kramer, S. The Level-Wise Version Space Algorithm and its Application to Molecular Fragment Finding. (IJCAI-01), 2001.
[6] G. Dong and J. Li. Efficient mining of emerging patterns: Discovering trends and differences. In Proceedings of the fifth International Conference on Knowledge Discovery and Data Mining, San Diego, USA, ($SIGKDD'99$), 1999, p.43-52.
[7] Dong, G., Li, J. and Zhang, X. Discovering Jumping Emerging Patterns and Experiments on Real Datasets. (IDC99), 1999.
[8] Fan, H. and Ramamohanarao, K. An efficient Single-scan Algorithm for mining Essential Jumping Emerging Patterns for Classification Accepted at PAKDD-2002, Taipei, May6-8, Taiwan.C.
[9] J. Han, J. Pei, Y. Yin. Mining frequent patterns without candidate generation. In Proceedings of the International Conference on Management of Data, Dallas, Texas, USA (*ACM SIGMOD*), 2000. p. 1-12.
[10] Kramer, S., De Raedt, L., Helma, C. Molecular Feature Mining in HIV Data. ACM SIGKDD (KDD-01), 2001.
[11] J. Li, G. Dong and K. Ramamohanarao. Making use of the most expressive jumping emerging patterns for classification. In Proceedings of the Fourth Pacific-Asia Conference on Knowledge Discovery and Data Mining, Kyoto, Japan, 2000. p. 220-232.
[12] J. Li and L. Wong. Emerging patterns and Gene Expression Data. In proceedings of 12th Workshop on Genome Informatics. Japan. December 2001, pages 3–13.
[13] Mannila, H. and Toivonen, H. Levelwise Search and Borders of Theories in Knowledge Discovery. Data Mining and Knowledge Discovery 1(3), 1997.
[14] T.M Mitchell. Generalization as search. Artificial Intelligence, 18, 203-226, 1982.
[15] Pasquier, N., Bastide, R., Taouil, R. and Lakhal, L. Efficient Mining of Association Rules using Closed Itemset Lattices. Information Systems 24(1), 1999.
[16] J.R. Quinlan: C4.5 Programs for Machine Learning. Morgan Kaufmann, 1993.