Visually Mining and Monitoring Massive Time Series

Jessica Lin[a] Eamonn Keogh[a] Stefano Lonardi[a] Jeffrey P. Lankford[b] Donna M. Nystrom[b]

[a]Computer Science & Engineering Department
University of California, Riverside
Riverside, CA 92521
{jessica, eamonn, stelo}@cs.ucr.edu

[b]The Aerospace Corporation
El Segundo, CA 90245-4691
{Jeffrey.P.Lankford, Donna.M.Nystrom}@aero.org

ABSTRACT

Moments before the launch of every space vehicle, engineering discipline specialists must make a critical *go/no-go* decision. The cost of a false positive, allowing a launch in spite of a fault, or a false negative, stopping a potentially successful launch, can be measured in the tens of millions of dollars, not including the cost in morale and other more intangible detriments. The Aerospace Corporation is responsible for providing engineering assessments critical to the *go/no-go* decision for every Department of Defense space vehicle. These assessments are made by constantly monitoring streaming telemetry data in the hours before launch. We will introduce VizTree, a novel time-series visualization tool to aid the Aerospace analysts who must make these engineering assessments. VizTree was developed at the University of California, Riverside and is unique in that the same tool is used for mining archival data and monitoring incoming live telemetry. The use of a single tool for both aspects of the task allows a natural and intuitive transfer of mined knowledge to the monitoring task. Our visualization approach works by transforming the time series into a symbolic representation, and encoding the data in a modified suffix tree in which the frequency and other properties of patterns are mapped onto colors and other visual properties. We demonstrate the utility of our system by comparing it with state-of-the-art batch algorithms on several real and synthetic datasets.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: *Information filtering, Query formulation, Selection process*. H.2.8 [**Database Applications**]: *Data mining, Scientific databases.*

General Terms

Algorithms, Design, Experimentation, Human Factors.

Keywords

Time Series, Visualization, Motif Discovery, Anomaly Detection, Pattern Discovery

Dr. Keogh is supported by NSF Career Award IIS-0237918

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
KDD '04, August 22–25, 2004, Seattle, Washington, U.S.A.
Copyright 2004 ACM 1-58113-888-1/04/0008…$5.00.

1. INTRODUCTION

One of the crucial responsibilities of The Aerospace Corporation is to provide engineering assessments for the government engineering discipline specialists who make the critical *go/no-go* decision moments before the launch of every space vehicle launched by the DoD. The cost of a false positive, allowing a launch in spite of a fault, or a false negative, stopping a potentially successful launch, can be measured in the tens of millions of dollars, not including the cost in morale and other more intangible detriments to the U.S. defense program.

The launch monitoring facilities at Aerospace are similar to the familiar Hollywood movie recreations [26]. There are several rows of work cells, each with a computer display and a headset. Each work cell is devoted to one analyst, for example, propulsion, guidance, electrical, etc. Each display presents some common data (say vehicle location and orientation), as well as data specific to that discipline.

The analyst making the engineering assessments has access to data from previous launches and must constantly monitor streaming telemetry from the current mission.

Currently, the analysts use electronic strip charts similar to those used to record earthquake shock on paper rolls. However, while these charts illustrate the recent history of each sensor, they do not provide any useful higher-level information that might be valuable to the analyst.

To reduce the possibility of wrong *go/no-go* decisions, Aerospace is continually investing in research. There are two major directions of research in this area.

- Producing better techniques to mine the archival launch data from previous missions. Finding rules, patterns, and regularities from past data can help us "*know what to expect*" for future missions, and allow more accurate and targeted monitoring, contingency planning, etc [26].
- Producing better techniques to visualize the streaming telemetry data in the hours before launch. This is particularly challenging because an analyst may have to monitor as many as dozens of rapidly changing sensors [26].

Although these two tasks are quite distinct, and are usually tackled separately, the contribution of this work is to introduce a single framework that can address both. Having a single tool for both tasks allows knowledge gleaned in the *mining* stage to be represented in the same visual language of the *monitoring* stage, thus allowing a more natural and intuitive transfer of knowledge.

More concretely, we propose VizTree, a time series pattern discovery and visualization system based on augmenting suffix trees. VizTree simultaneously visually summarizes both the global and local structures of time series data. In addition, it provides novel interactive

solutions to many pattern discovery problems, including the discovery of frequently occurring patterns (motif discovery) [7, 29, 38], surprising patterns (anomaly detection) [9, 24, 36], and query by content [11, 15, 21, 34]. The user interactive paradigm allows users to visually explore the time series, and perform real-time hypotheses testing [1, 19].

We employ the widely referenced "*Overview, zoom & filter, details on demand*" paradigm of Dr. Ben Shneiderman of the University of Maryland [37]. As we will show in this paper, our work fits neatly into these principles. We give an *overview* of the global structure of an arbitrarily long time series in constant space, while we allow the user to *zoom* in on particular local structures and patterns, and provide *details on demand* for patterns and regularities that the user has tentatively identified.

While there are several systems for visualizing time series in the literature, our approach is unique in several respects. First, almost all other approaches assume highly periodic time series [40, 41], whereas ours makes no such assumption. Other methods typically require space (both memory space, and pixel space) that grows at least linearly with the length of the time series, making them untenable for mining massive datasets. Finally, our approach allows us to visualize a much richer sets of features, including global summaries of the differences between two time series, locally repeated patterns, anomalies, etc.

While the evaluation of visualization systems is often subjective, we will evaluate our system with objective experiments by comparing our system with state-of-the-art batch algorithms on several real and synthetic datasets.

The rest of the paper is organized as follows. In Section 2 we review necessary background material and survey related work. We introduce our system VizTree in Section 3. In Section 4, we extend the idea to further allow comparison and contrast between two time series. Section 5 contains a detailed empirical evaluation of our system. We conclude in Section 6.

We note that all the figures in this text suffer from their small scale and monochromatic printing. We encourage the interested reader to visit [27] to view high-resolution full-color examples.

2. BACKGROUND AND RELATED WORK

We begin this section by briefly reviewing the most important time series data mining tasks. We will then consider current visualization techniques and explain why they are unsuited to the problem at hand.

2.1 Time Series Data Mining Tasks

For lack of space, this brief introduction to the important time series data mining tasks is necessarily subjective and somewhat domain driven. Nevertheless, these three tasks cover the majority of time series data mining research [6, 7, 9, 11, 15, 18, 22, 24, 29, 30, 31, 38].

2.1.1 Subsequence Matching

Sequence matching is perhaps the most widely studied area of time series data mining [11, 15]. The task has long been divided into two categories: whole matching and subsequence matching [11, 21].

- **Whole Matching** a query time series is matched against a database of individual time series to identify the ones similar to the query.
- **Subsequence Matching** a short query subsequence time series is matched against longer time series by sliding it along the longer sequence, looking for the best matching location.

While there are literally hundreds of methods proposed for whole sequence matching (see, e.g., [22] and references therein), in practice, its application is limited to cases where some information about the data is known *a priori*.

Subsequence matching can be generalized to whole matching by dividing sequences into non-overlapping sections. For example, we may wish to take a long electrocardiogram and extract the individual heartbeats. This informal idea has been used by many researchers and is also an optional feature of VizTree. We will therefore formally name this transformation *chunking*, and define it below.

Definition 1. *Chunking*: the process where a time series is broken into individual time series by either a specific period or, more arbitrarily, by its shape.

The former usually applies to periodic data, for example consider power usage data provided by a Dutch research facility (this dataset is used as a running example in this work, see Figures 3 and 15): the data can be chunked by days, weeks, etc. The latter applies to data having regular structure or repetitive shape, but not necessarily having the same length for each occurrence. Electrocardiogram data are such an example, and they can be separated into individual heartbeats.

There is increasing awareness that for many data mining and information retrieval tasks, very fast approximate search is preferable to slower exact search [5]. This is particularly true for exploratory purposes and hypotheses testing. Consider the stock market data. While it makes sense to look for approximate patterns, for example, *"a pattern that rapidly decreases after a long plateau"* it seems pedantic to insist on *exact* matches. As we will demonstrate in Section 5.1, our application allows rapid approximate sequence matching.

2.1.2 Anomaly Detection

In time series data mining and monitoring, the problem of detecting anomalous/surprising/novel patterns has attracted much attention [9, 30, 36]. In contrast to subsequence matching, anomaly detection is identification of previously *unknown* patterns. The problem is particularly difficult because what constitutes an anomaly can greatly differ depending on the task at hand. In a general sense, an anomalous behavior is one that deviates from "normal" behavior. While there have been numerous definitions given for anomalous or surprising behaviors, the one given by Keogh et. al. [24] is unique in that it requires no explicit formulation of what is anomalous. Instead, they simply defined an anomalous pattern as one *"whose frequency of occurrences differs substantially from that expected, given previously seen data"*. Their definition was implemented in an algorithm (called "Tarzan") that was singled out by NASA as an algorithm that has *"great promise in the long term"* [17]. As it will become clearer later, a subset of the system that we propose here includes what may be considered a visual encoding of Tarzan.

2.1.3 Time Series Motif Discovery

In bioinformatics, it is well documented that overrepresented DNA sequences often have biological significance [2, 10, 35]. Other applications that rely heavily on overrepresented (and underrepresented) pattern discovery include intrusion detection, fraud detection, web usage prediction, financial analysis, etc.

A substantial body of literature has been devoted to techniques to discover such overrepresented patterns in time series; however, each work considered a different definition of *pattern* [3, 32]. In previous work, we unified and formalized the problem by defining the concept of "time series motif" [29]. Time series motifs are close analogues of

their discrete cousins, although the definitions must be augmented to prevent certain degenerating solutions. This definition is gaining acceptance, and now being used in animation [4], mining human motion data [38], and several other applications. The naïve algorithm to discover motifs is quadratic in the length of the time series. In [29], we demonstrated a simple technique to mitigate the quadratic complexity by a large constant factor; nevertheless this time complexity is clearly untenable for most real datasets. As we shall demonstrate in Section 5.2, VizTree allows users to visually discover approximate motifs in real time.

2.2 Visualizing Time Series

Time series is perhaps the most common data type encountered in data mining, touching as it does, almost every aspect of human life, including medicine (ECG, EEG data), finance (stock market data, credit card usage data), aerospace (launch telemetry, satellite sensor data), entertainment (music, movies) [4], etc. Because time series datasets are often massive (in gigabytes or even terabytes), time- and space-complexity is of paramount importance.

Surprisingly, although the human eye is often advocated as the ultimate data-mining tool [19, 37, 39], there has been relatively little work on visualizing massive time series datasets. We have reimplemented the three most referenced approaches in the literature. Below, we will briefly review them and explain why they are not suited to the task at hand.

2.2.1 TimeSearcher

TimeSearcher [14] is a time series exploratory and visualization tool that allows users to retrieve time series by creating queries. This is achieved by use of "TimeBoxes", which are rectangular query locators that specify the region(s) in which the users are interested within any given time series. In Figure 1, three TimeBoxes have been drawn to specify time series that start low, increase, then fall once more.

The main advantage of this tool is its flexibility. In particular, unlike conventional query-by-content similarity search algorithms, TimeSearcher allows users to specify different regions of interest from a query time series, rather than feeding the entire query for matching. This is useful when users are interested in finding time series that exhibit similar behavior as the query time series in only specific regions.

While TimeSearcher and VizTree proposed here both serve as visualization and exploratory tools for time series, their functionalities are fundamentally different. For example, TimeSearcher is a query-by-example tool for multiple time series data. Even with its flexibility, users still need to specify the query regions in order to find similar patterns. In other words, some knowledge about the datasets may be needed in advance and users need to have a general idea of what is interesting. On the other hand, VizTree serves as a true pattern discovery tool for a long time series that tentatively identifies and isolates interesting patterns and invites further inspection by the analyst.

The functionality of TimeSearcher for similarity search is implicit in the design of our system: similar patterns are automatically grouped together. Furthermore, TimeSearcher suffers from its limited scalability, which restricts its utility to smaller datasets, and is impractical for the task at hand.

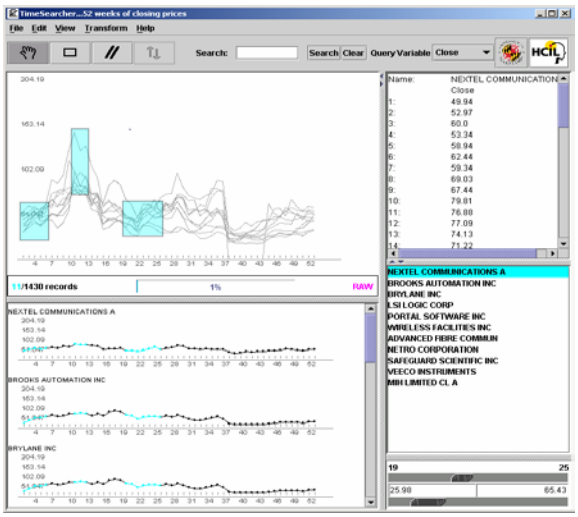

Figure 1: The TimeSearcher visual query interface. A user can filter away sequences that are not interesting by insisting that all sequences have at least one data point within the query boxes.

2.2.2 Cluster and Calendar-Based Visualization

Another time series visualization system is cluster and calendar-based, developed by [40]. The time series data are chunked into sequences of day patterns, and these day patterns are in turn clustered using a bottom-up clustering algorithm. This visualization system displays patterns represented by cluster averages, as well as a calendar with each day color-coded by the cluster that it belongs to. Figure 2 shows just one view of this visualization scheme.

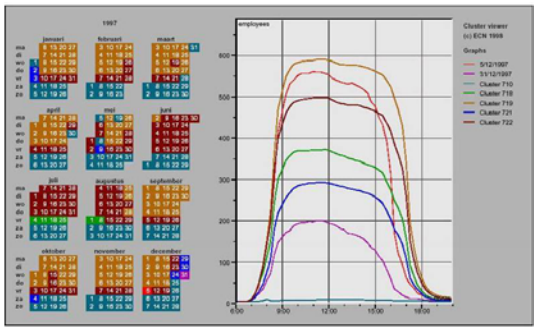

Figure 2: The cluster and calendar-based visualization on employee working hours data. It shows six clusters, representing different working-day patterns.

While the calendar-based approach provides a good overview of the data, its application is limited to calendar-based data, that is to say, data which has some regularity imposed on it by social or financial dependence on the calendar. This approach is of little utility for data without obvious daily/weekly patterns and/or *a priori* knowledge about such patterns. In short, this system works well to find patterns within a specific, known time scale, while our system aims to discover previously unknown patterns with little or no knowledge about the data.

2.2.3 Spiral

Weber et. al developed a tool that visualizes time series on spirals [41]. Each periodic section of time series is mapped onto one "ring" and attributes such as color and line thickness are used to characterize the data values. The main use of this approach is the identification of periodic structures in the data. However, the utility of this tool is

limited for time series that do not exhibit periodic behaviors, or when the period is unknown.

We reimplemented the spiral approach and ran it on the power consumption dataset. A screenshot of the resulting spiral is shown in Figure 3

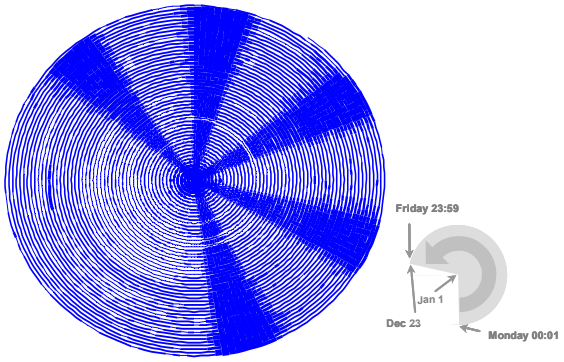

Figure 3: The Spiral visualization approach of Weber et al. applied to the power usage dataset[1].

Note that one can clearly visualize the normal "9-to-5" working week pattern. In addition, one can see several other interesting events. For example, while it is apparent that no one works during weekends in general, on one Saturday in late summer, there was a power demand suggestive of a full days shift. Surprisingly, this idea for visualizing time series predates computers, with elegant hand drawn examples dating back to at least the 1880's [12, 39].

While the Spiral approach is elegant, it does not meet our requirements for several reasons. As mentioned, it works well only for periodic data (based on the original authors' claims and our own experiments). More importantly, it requires pixel space linear in the length of the time series; this is simply untenable for our purposes.

3. OUR APPROACH: VIZ-TREE

Our visualization approach works by transforming the time series into a symbolic representation, and encoding the data in a modified suffix tree in which the frequency and other properties of patterns are mapped onto colors and other visual properties. Before explaining our approach in detail, we will present a simple problem that motivates our work.

Two sets of binary sequences of length 200 were generated: the first set by the pseudo-random-number generator by the computer, and the second set by hand by a group of volunteers. The volunteers were asked to try and make the bit strings as random as possible, and were offered a prize to motivate them. Figure 4 shows one sample sequence from each set.

By simply looking at the original bit strings, it's difficult, if not impossible, to distinguish the computer-generated from the human-constructed numbers. However, if we represent them with a tree structure where the frequencies of subsequences are encoded in the thickness of branches, the distinction becomes clear. For clarity, the trees are pruned at depth three. Each tree represents one sequence from each set, and each node in the tree has exactly two branches: the upper branch represents 1, and the lower branch represents 0. The tree is constructed as follows: starting from the beginning of each sequence, subsequences of length three are extracted with a sliding window that slides across the sequence one bit at a time. So for the first sequence we get a set of subsequences {(0,1,0), (1,0,1), (0,1,1), …}.

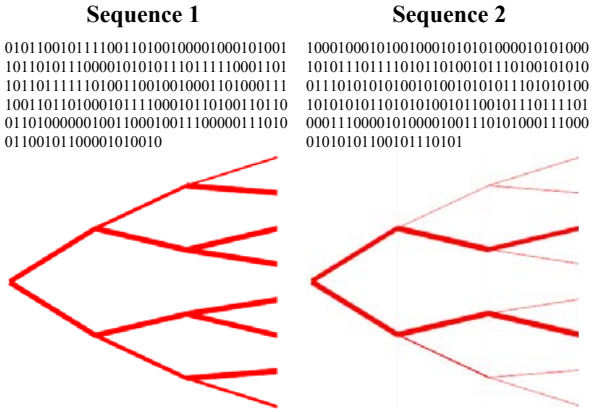

Figure 4: (*Left*) Computer-generated random bits presented as an augmented suffix tree (*Right*) Human-constructed bits presented as an augmented suffix tree.

For the tree shown on the left in Figure 4, the branches at any given level have approximately the same thickness, which means that the probabilities of subsequences at any given level are approximately evenly distributed. In contrast, the tree on the right shows that subsequences of alternating 0's and 1's dominate the whole sequence. The "motifs" for the sequence, 101 and 010, can be easily identified, since they appear more frequently than the other subsequences.

The non-randomness, which can be seen very clearly in this example, implies that humans usually try to "fake" randomness by alternating patterns [16]. Undoubtedly, there exist other solutions to uncover these "patterns" (entropy, Hidden Markov models, etc.). Nonetheless, what this visualization scheme provides is a straightforward solution that allows users to easily identify and view the patterns in a way intuitive to human perception.

The simple experiment demonstrates how visualizing augmented suffix trees can provide an overall visual summary, and potentially reveal hidden structures in the data. Since the strings represented in the tree are in fact "subsequences" rather than "suffixes," we call such trees *subsequence trees*.

This simple experiment motivates our work. Although time series are not discrete, they can be discretized with little loss of information, thus allowing the use of suffix/subsequence trees.

Our system is partly inspired by Visualysis [25], a visualization tool for biological sequences. Visualysis uses a suffix tree to store the biological sequences and, through the properties of the tree, such as bushiness, branch distribution, etc., and user navigation, interesting biological information can be discovered [25]. Visualysis incorporates algorithms that utilize suffix trees in computational biology; more specifically, exact sequence matching and tandem repeat algorithms. At a first glance, our visualization system is similar to Visualysis in the sense that it also has the objective of pattern discovery using a tree structure. However, several characteristics that are unique to our application make it more diversely functional than its computational-biology counterpart. First, although the tree structure needs the data to be discrete, the

[1] Of all the figures in this paper, this one suffers the most from the small scale of reproduction. In addition we did not optimize the anti-aliasing and other graphic tricks to make the hard copy reproduction as good as the on screen version. We encourage the interested reader to refer to the original paper [29] for much higher quality images.

original time series data is not. Using a time-series discretization method that we introduced in an earlier work [28], continuous data can be transformed into discrete domain, with certain desirable properties such as lower-bounding distance, dimensionality reduction, etc. Second, instead of using a suffix tree, we use a subsequence tree that maps all subsequences onto the branches of the tree. Thus, given the same parameters, the trees have the same overall shape for any dataset. This approach makes comparing two time series easy and anomaly detection possible.

3.1 The Utility of Discretizing Time Series

In [28], we introduced Symbolic Aggregate approximation (SAX), a novel symbolic representation for time series. It is ideal for this application since, unlike all previously proposed discretization methods for time series, SAX allows lower-bounding distance measures to be defined on the symbolic space. In addition, its dimensionality reduction feature makes approximating large dataset feasible, and its ability to convert the data using merely the local information, without having to access the entire dataset, is especially desirable for streaming time series. The utility of SAX has been demonstrated in [28], and the adaptation or extension of SAX by other researchers further shows its impact in diverse fields such as medical and video [6, 33]. For these reasons, we choose to use SAX as the discretization method for the input time series data.

Before converting a time series to symbols, it should be normalized. The importance of normalization has been extensively documented in the past [22]. Without normalization, many time series data mining tasks have little meaning [22]. After normalization[2], SAX performs the discretization in two steps. First, a time series T of length n is divided into w equal-sized segments; the values in each segment are then approximated and replaced by a single coefficient, which is their average. Aggregating these w coefficients form the Piecewise Aggregate Approximation (PAA) representation of T.

Next, to convert the PAA coefficients to symbols, we determine the breakpoints that divide the distribution space into α equiprobable regions, where α is the alphabet size specified by the user. In other words, the breakpoints are determined such that the probability of a segment falling into any of the regions is approximately the same. If the symbols were not equi-probable, some of the substrings would be more probable than others. As a consequence, we would inject a probabilistic bias in the process. In [8], Crochemore et. al. showed that a suffix tree automation algorithm is optimal if the letters are equiprobable.

Once the breakpoints are determined, each region is assigned a symbol. The PAA coefficients can then be easily mapped to the symbols corresponding to the regions in which they reside. In [28], the symbols are assigned in a bottom-up fashion so the PAA coefficient that falls in the lowest region is converted to "a," in the one above to "b," and so forth. In this paper, for reason that will become clear in the next section, we reverse the assigning order, so the regions will be labeled top-down instead (i.e. the top-most region is labeled "a," the one below it "b," and so forth). Figure 5 shows an example of a time series being converted to string *acdcbdba*. Note the general shape of the time series is preserved, in spite of the massive amount of dimensionality reduction, and the symbols are equiprobable.

[2] In the unusual event where it might be more appropriate not to normalize, for example, when offset and amplitude changes are important, VizTree provides an option to skip the normalization step.

The discretization technique can be applied to VizTree by calling SAX repeatedly for each subsequence. More specifically, subsequences of specified lengths are extracted from the input time series and normalized to have a mean of zero and a standard deviation of one. Applying SAX on these subsequences, we obtain a set of strings. From this point on, the steps are identical to the motivating example shown in the beginning of Section 3: the strings are inserted into the subsequence tree one by one.

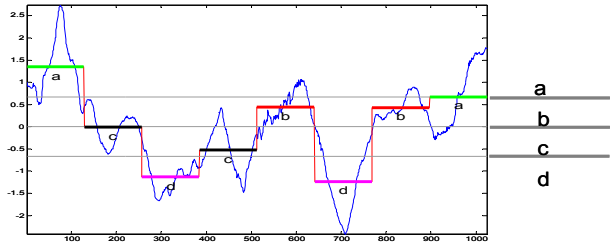

Figure 5: A time series dataset "leleccum" of length 1024 is converted into an eight-symbol string "**acdcbdba**" Note that the general shape of the time series is preserved, in spite of the massive amount of dimensionality reduction.

3.2 A First Look at VizTree

Figure 6 shows a screen shot of VizTree. When the program is executed, four blank panels and a parameter-setting area are displayed. To load a time series dataset, the user selects the input file using a familiar dropdown menu. The input time series is plotted in the top left-hand panel. Next to the time series plotting window is the parameter setting area; the analyst can enter the sliding window length, the number of SAX segments per window, and select alphabet size from a dropdown menu. Once the parameters are entered, the user can click on the "Show Tree" button to display the subsequence tree on the bottom left panel.

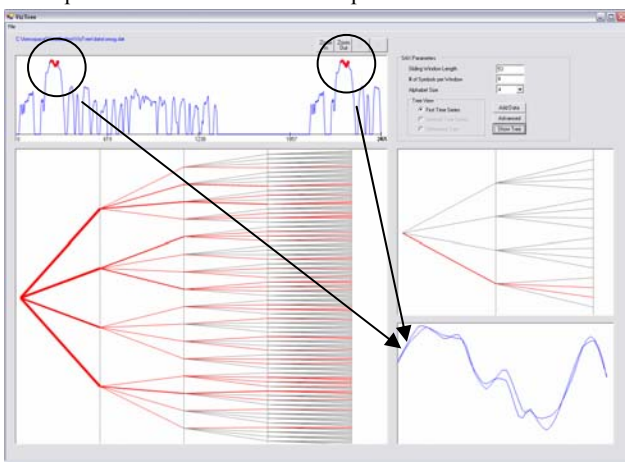

Figure 6: A screenshot of Viztree. The top panel is the input time series. The bottom left panel shows the subsequence tree for the time series. On the right, the very top is the parameter setting area. Next to the subsequence tree panel, the top window shows the zoom-in of the tree, and the bottom window plots the actual subsequences when the analyst clicks on a branch.

The time series used for this example is a real, industrial dataset of smog emissions from a motor vehicle. The length of the time series is 2478. The length of the sliding window is set to 53; the number of segments (i.e., the depth of the tree) is four, and the alphabet size (i.e., the number of children for each node) is four.

Each branch represents one pattern As mentioned in the previous section, we reverse the assigning order of the symbols from bottom-

up to top-down. The reason is that when the symbols are arranged this way, it is more consistent with the natural shape of the tree. For example, for any given node, a branch at a higher position denotes segments with higher values. Traversing breadth-first from the top-most branch of any given node, the symbols that represent the branches are *a, b, c*, and *d*, respectively. Each level of the tree represents one segment (or one symbol). To retrieve any string, we simply traverse down the appropriate branches.

Definition 2 *Pattern*: a pattern p is the SAX representation of a subsequence in the time series, denoted by the string s formed by following any path down the subsequence tree. The frequency of p in time series A is denoted by $f(p_A)$, which is the number of occurrences of p over the number of all occurrences in A.

The frequency of a pattern is encoded in the thickness of the branch. For clarity, the full tree is drawn. Branches with zero frequency are drawn in light gray, while others are drawn in red with varying thicknesses.

On the right hand side of VizTree, there are two panels. The upper one shows the zoom-in of the tree shown in the left panel. This is very useful especially for deep and bushy trees. The user can click on any node (on the subsequence tree window, or recursively, on the zoom-in window) and the sub-tree rooted at this node will be displayed in this upper panel. The sub-tree shown in Figure 6 is rooted at the node representing the string "*abxx*," where the "*xx*" denotes *don't-care* since we are not at the leaf level. If the user clicks on any branch, then the actual subsequences having the string represented by this particular branch will be displayed in the bottom panel and highlighted in the time series plot window. In the figure, subsequences encoded to "*abdb*" are shown.

3.2.1 Parameter Selection

Three parameters need to be determined: the length of the sliding window, the number of segments, and the alphabet size. In [29] we showed the trade-off between the number of segments and the alphabet size. In general, VizTree works very well even with massive dimensionality reduction, as we will demonstrate in Section 5 (in the experiments we used no more than 5 segments). The length of the sliding window is data-dependent; however, the user can drag a range over any pattern of interest on the time series plot window and the window size will be filled in automatically.

3.3 Subsequence Matching

Subsequence matching can be done very efficiently with VizTree. Instead of feeding another time series as query, the user provides the query in an intuitive way. Recall that each branch corresponds to one of the equiprobable regions that are used to convert the PAA coefficients to symbols. The top branch corresponds to the region with the highest values, and the bottom branch corresponds to the region with the lowest values. Therefore, any path can be easily translated into a general shape and can be used as a query. For example, the top-most branch at depth one (i.e., string "*axxx*") represents all subsequences that start with high values, or more precisely, whose values in the first segment have the mean value that resides in the highest region. In the previous example, the user is interested in finding a concave-down pattern (i.e., a U-shape). This particular pattern, according to the domain experts, corresponds to a change of gears in the motor vehicle during the smog emission test. From the U shape, the user can approximate the query to be something that goes down and comes up, or a string that starts and ends with high branches, with low branches in the middle. As a result, clicking on the branch representing "*abdb*" as shown in the figure uncovers the pattern of interest.

3.4 Motif Discovery & Simple Anomaly Detection

VizTree provides a straightforward way to identify motifs. Since the thickness of a branch denotes the frequency of the subsequences having the same, corresponding strings, we can identify approximate motifs by examining the subsequences represented by thick tree paths. A feature unique to VizTree is that it allows users to visually evaluate and inspect the patterns returned. This interactive feature is important since different strings can also represent similar subsequences, such as those that differ by only one symbol. In addition, the user can prune off uninteresting or expected patterns to improve the efficiency of the system and reduce false positives. For example, for ECG data, the motif algorithm will mostly likely return normal heart beats as the most important motif, which is correct but non-useful. Allowing user to manually prune off this dominant pattern, secondary yet more interesting patterns may be revealed.

Figure 7 shows such an example. The dataset used here is a *real*, industrial dataset, "winding," which records the angular speed of a reel. The subsequences retrieved in the lower right panel have the string representation "*dacb*." Examining the motifs in this dataset allowed us to discover an interesting fact: while the dataset was advertised as real, we noted that repeated patterns occur at every 1000 points. For example, in Figure 7, the two nearly identical subsequences retrieved are located at offsets 599 and 1599, exactly 1000 points apart. We checked with the original author and discovered that this is actually a synthetic dataset composed from parts of a real dataset, a fact that is not obvious from inspection of the original data.

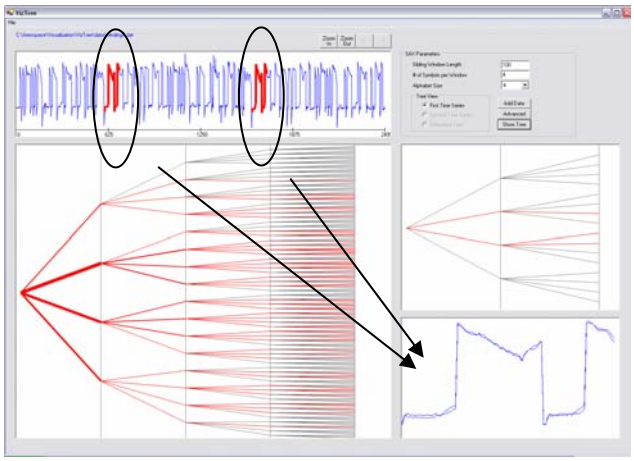

Figure 7: Example of motif discovery on the winding dataset. Two nearly identical subsequences are identified, among the other motifs.

The complementary problem of motif discovery is anomaly detection. While frequently occurring patterns can be detected by thick branches in the Viztree, simple anomalous patterns can be detected by unusually thin branches. Figure 8 demonstrates both motif discovery and simple anomaly detection on an MIT-BIH Noise Stress Test Dataset (ECG recordings) obtained from PhyioBank [13]. Here, motifs can be identified very easily from the thick branches; more remarkably, there is one very thin line straying off on its own (the path that starts with "a"). This line turns out to be an anomalous heart beat, independently annotated by a cardiologist as a premature ventricular contraction.

While anomalies can be detected this way for trivial cases, in more complex cases, the anomalies are usually detected by comparing the time series against a normal, reference time series. Anything that differs substantially from this reference time series can signal anomalies. This is exactly the objective of the Diff-Tree, as described in the next section.

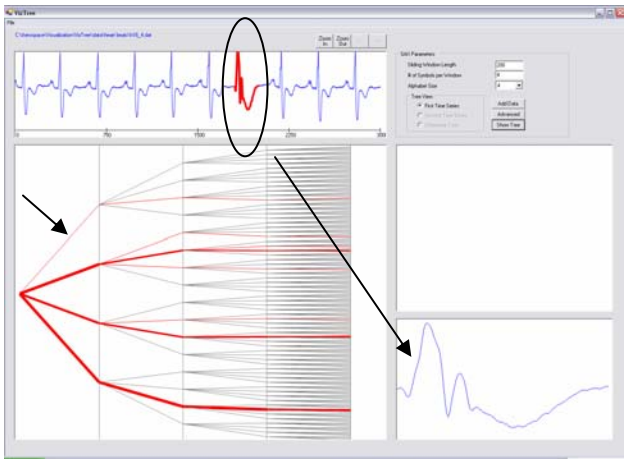

Figure 8: Heart-beat data with anomaly is shown. While the subsequence tree can be used to identify motifs, it can be used for simple anomaly detection as well.

4. DIFF-TREE

We have described how global structures, motifs, and simple anomalies can be identified by a subsequence tree. In this section, we extend these ideas to further allow the comparison of two time series by means of a "diff-tree." A diff tree is short for "difference tree," and as the name implies, shows the distinction between two time series. The construction of a diff-tree is fairly straightforward with the use of subsequence tree, since the overall tree shape is the same regardless of the strings, provided that the parameters selected (i.e., alphabet size, number of segment, etc) are the same. The diff-tree is constructed by computing the difference in thickness (i.e., frequency of occurrence) for each branch. Intuitively, time series data with similar structures can be expected to have similar subsequence trees, and in turn, a sparse diff-tree. In contrast, those with dissimilar structures will result in distinctively different subsequence trees and therefore a relatively dense diff-tree.

One or two datasets can be loaded to VizTree simultaneously. If only one is loaded, then its subsequence tree will be shown. If two datasets are loaded, the user has the option of viewing the subsequence tree of either one, or their diff-tree. The branches in the difference tree are color-coded to distinguish between the overrepresented and underrepresented patterns. Given two time series A and B, where A is the basis for comparison (the reference time series), and B is the added time series, we can define the following terms:

Definition 3. *Overrepresented Pattern* a pattern is over-represented in B if it occurs more frequently in B than it does in A.

Definition 4 *Underrepresented Pattern* a pattern is under-represented in B if it occurs more frequently in A than it does in B.

Definition 5. *Degree of Difference* the degree of difference for any pattern p between A and B is defined as follows:

$$D_p = \frac{f(p_B) - f(p_A)}{\max(\max\_freq\_in\_A, \max\_freq\_in\_B)} \quad (1)$$

Simply stated, D_p measures how a pattern (i.e. branch) differs from one time series to another, by computing the difference of frequencies between A and B and dividing by the maximum frequency in A and B. If p occurs less frequently in B than in A, then the pattern is underrepresented and $D_p < 0$, otherwise it is overrepresented and $D_p > 0$.

This is the measure encoded in the diff-tree as the thickness of the branch. Currently, discrete colors are used to distinguish overrepresentation from underrepresentation: overrepresented patterns are drawn in green (same color as the test time series); underrepresented patterns in blue (same color as the basis time series); and if the frequency is the same, then the branch is drawn in red. However, color intensity can be used to further highlight the degrees of difference.

4.1 Anomaly Detection

The datasets used for anomaly detection, constructed independently of the current authors and provided by the Aerospace Corporation for sanity check, are shown in Figure 9. The one on the top is the normal time series, and the one below is similar to a normal time series, except it has a gap in the middle as anomaly. Figure 10 shows a screenshot of the anomaly detection by diff-tree. The tree panel shows the diff-tree between the two datasets. The two thick paths denote the beginning and the end of the anomaly, respectively.

This is a very trivial example for demonstration purpose. However, the effect is similar for more complex cases.

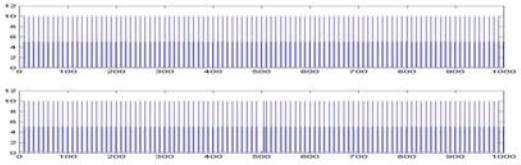

Figure 9 The input files used for anomaly detection by diff-tree. *(Top)* Normal time series. *(Bottom)* Anomaly is introduced as a gap in the middle of the dataset.

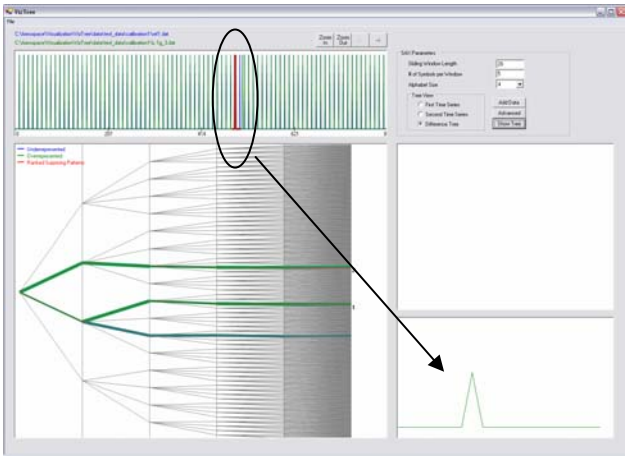

Figure 10: Diff-tree on the datasets shown in the previous figure. The gap is successfully identified.

5. EXPERIMENTAL EVALUATION

In this section we evaluate (and demonstrate) our approach on datasets which are either very intuitive to the average person or have been extensively annotated by domain experts. In particular, we will evaluate our work on human motion data and the power demand

data. Note that all datasets used here are available for free from the UCR archive [20].

5.1 Subsequence Matching

This experiment incorporates both subsequence matching and motif discovery. The dataset used is the human motion data of yoga postures. A model postured yoga routines in front of a green screen, and her motion was captured by various sensors. The motion capture is transformed into a time series by computing the aspect ratio of the minimum-bounding rectangle formed around her body. The length of the time series is approximately 26,000 (i.e. there are approximately these many frames in the original video).

Suppose we are interested in finding a yoga sequence like the one in Figure 11:

Figure 11: A sample yoga sequence for approximate subsequence matching.

Then we would expect the shape of the query to descend rapidly after the first position (the width-to-height-ratio decreases), ascend slowly after the second position, descend again, and finally ascend once more. Assume that we set the number of segments to be five (an arbitrary choice), then a reasonable start would be the branch "*adxxx*." Since there are only two paths extending from the node "*ad*," the matches are found very quickly without much refinement in the search space. The result is shown in Figure 12 and the actual yoga sequences for the matches are outlined in Figure 13. The subsequence length is 400 (i.e. about 6.5 seconds). As the figure shows, the two sequences are very similar with only very minor distinction.

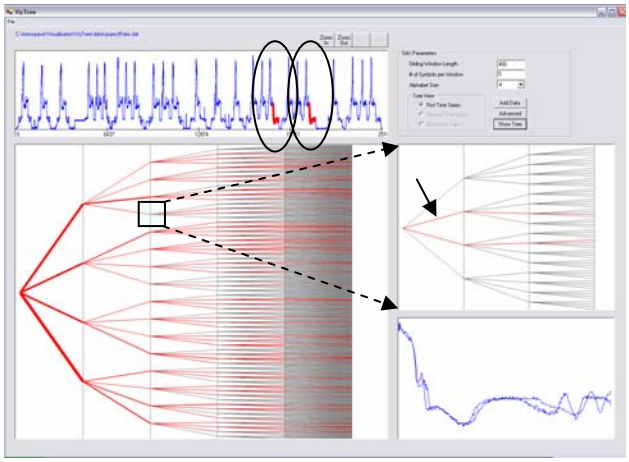

Figure 12: Matches for the yoga sequence in Figure 11. The bottom right corner shows how similar these two subsequences are.

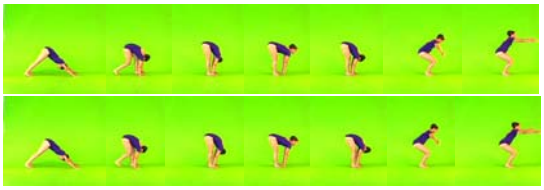

Figure 13: Outline of the actual yoga sequences that match the query.

There are several advantages of the approximate subsequence matching by VizTree. One is that this feature is built-in to the application, and it is relatively easy to specify the query without explicitly providing it. More importantly, the system retrieves the results very efficiently since the information is already stored in the tree. With the current state-of-the-art *exact* subsequence matching algorithms, retrieval is much too slow for a real time interaction.

5.2 Motif Discovery

For the motif discovery experiment, we will continue with the previous human-motion example. There are obviously some noticeable motifs such as the long spikes that occur throughout the sequence (see the time series plot in Figure 12). They denote the posture where the model is lying flat on the ground, when the aspect ratio is at its maximum. However, one of the desirable features of VizTree is that it allows users to visually identify secondary yet more interesting motifs. The matches found in the previous section are such example. We can zoom-in on these subsequences and examine their similarity.

From Figure 14 we can see that these two subsequences are indeed very similar to each other. Note that they both have a small dip towards the end of the sequence. However, there is a slight difference there – the dip for the first sequence occurs before that for the second sequence, and is followed by a plateau. Examining the motion captures we discover that the dip corresponds to the 6th position shown in Figure 13, right before the model stretched her arms straight in front of her. In addition, for the first sequence, the model held that last position for a longer period of time, thus the plateau following the dip. These subtle differences are difficult to notice without the motif discovery and/or the subsequence matching features in VizTree.

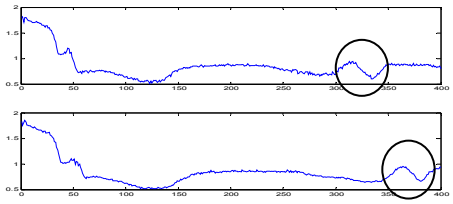

Figure 14 Zoom-ins of the two matches found in the yoga subsequence match example. Note that they both have a dip towards the end of the sequences.

For comparison, we ran the fastest known *exact* motif discovery algorithm [29] . Although the same motif can also be successfully identified, it takes minutes to compute, while VizTree gives instant (less than one second) feedback on the results. Even with the *approximate* motif discovery algorithm [7], it takes tens of seconds to complete. In addition, the visualization power of VizTree allows the user to see exactly where the motif occurs and how it maps to the original time series.

5.3 Anomaly Detection

For anomaly detection, we used the power demand data that was also used in Figure 3. Electricity consumption is recorded every 15 minutes; therefore, for the year of 1997, there are 35,040 data points. Figure 15 shows the resulting tree with the sliding window length set to 672 (exactly one week of data), and both alphabet size and number of segments to 3. The majority of the weeks follow the regular Monday-Friday, 5-working-day pattern, as shown by the thick branches. The thin branches denote the anomalies. The one circled is from the branch "*bab*." The zoom-in shows the beginning of the three-day week during Christmas (Thursday and Friday off). The

other thin branches denote other anomalies[3] such as New Year's Day, Good Friday, Queen's Birthday, etc.

While other anomaly detection algorithms such as the TSA-Tree Wavelet-based algorithm by Shahabi et. al. [36] and the Immunology-based algorithm (IMM) by Dasgupta and Forrest [9] can potentially find these anomalies as well given the right parameters, both are much more computationally intensive. While VizTree requires input of parameters, the results are almost instant. In the contrary, the TSA-Tree takes tens of seconds, and IMM needs re-training its data with every adjustment of parameters, with each training session taking several minutes. This is clearly untenable for massive datasets.

In addition to the fast computational time, anomaly detection by VizTree does not always require a training dataset. As demonstrated, simple anomalies can be identified as an inverse to the motifs.

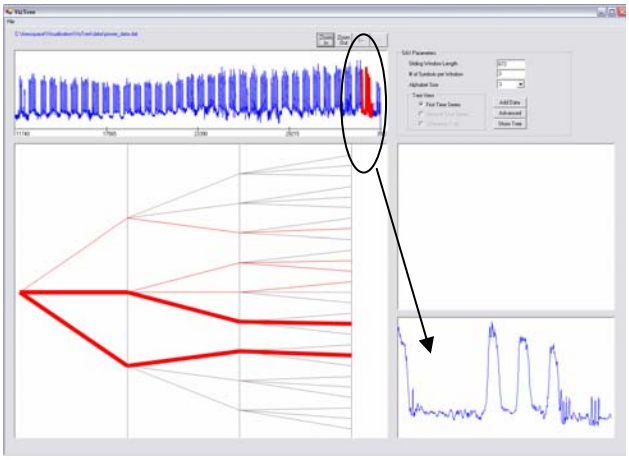

Figure 15 Anomaly detection on power consumption data. The anomaly shown here is a short week during Christmas.

5.4 Scalability

The pixel space of the subsequence tree is determined solely by the number of segments and alphabet size. In particular, we note that the pixel size of the tree is *constant* and independent to the length of time series. We have already shown that large amounts of dimensionality reduction do not greatly affect the accuracy of our results (in Section 5.3, the dimensionality is reduced from 672 to 3, a compression ratio of 224-to-1). However, the size of the dataset plays a role in memory space, since each node in the tree stores the offsets of its subsequences. However, SAX allows efficient numerosity reduction to reduce the number of subsequences being included into the tree, in addition to alleviating the problem associated with trivial matches (see below) [23, 29].

5.4.1 Numerosity Reduction

In [29] we showed that the best matches for a subsequence tend to be its immediate neighbors: the subsequence one point to the right and the subsequence one point to the left. We defined these matches to be the "trivial matches." In the smooth regions of the time series, the amount of trivial matches might be large. If we include them in any sliding-window algorithms, the trivial matches will dominate over the true patterns due to over-counting, and the results will likely be distorted, or worse, become meaningless [23]. Therefore, when extracting subsequences from the time series by a sliding window, the trivial matches should be excluded.

Different definitions can be used to identify trivial matches. The easiest way is to compare the SAX strings and only record a subsequence if its string is different from the last one recorded. In other words, no two consecutive strings should be the same.

Additionally, we can check two strings symbol-by-symbol and consider them trivial matches of one another if no pair of symbols is more than one alphabet apart. This extra check is based on the same idea as the previous numerosity reduction option, that similar subsequences have the same SAX representation. However, it is also likely that similar subsequences do not have exactly the same SAX representations; rather, they might have alphabets that differ by at most one at any given position (i.e. the values could be very close but reside on different sides of a breakpoint).

Furthermore, the second option can be extended to also exclude non-monotonic strings. Depending on the nature of the datasets, users might only be interested in finding patterns with ups and downs. Finally, the ultimate numerosity reduction can be achieved by chunking, which allows no overlapping subsequences. This has been used for many approaches; however, we would like to note that it is only useful if the dataset exhibits regular patterns, either by shape or by period. For example, if we use chunking for the power consumption data used in Section 5.3, then we get an even more distinctive tree.

6. CONCLUSIONS AND FUTURE WORK

We proposed VizTree, a novel visualization framework for time series that summarizes the global and local structures of the data. We demonstrated how pattern discovery can be achieved very efficiently with VizTree.

As mentioned, VizTree will be formally evaluated by The Aerospace Corp in the summer of 2004, and we will incorporate the feedback into the system. We believe that researchers from other sectors of the industry can greatly benefit from our system as well. For example, it could potentially be used for indexing and editing video sequences. We plan to have domain experts in other fields such as medicine and animation evaluate our system.

In the beginning of the paper we mention that the system can be used for monitoring and mining time series data. While we mainly focus on the "mining" aspect in this paper, we will extend VizTree to accept online streaming data for monitoring purposes.

Reproducible Research Statement: All datasets and code used in this work will be freely available. For higher-quality images and more information, please visit http://www.cs.ucr.edu/~jessica/VizTree.htm.

7. ACKNOWLEDGMENTS

Thanks to Victor Zordan and Bhrigu Celly for providing the yoga postures data.

8. REFERENCES

[1] Aggarwal, C. (2002). Towards Effective and Interpretable Data Mining by Visual Interaction. In *SIGKDD Explorations* Jan, 2002.

[2] Apostolico, A., Bock, M. E. & Lonardi, S. (2002). Monotony of Surprise in Large-Scale Quest for Unusual Words. In *proceedings of the 6th Int'l conference on Research in Computational Molecular Biology* Washington, D.C., Apr 18-21. pp. 22-31.

[3] Anomalies in the sense that the electricity consumption is abnormal given the day of the week.

[3] Caraca-Valente, J. P. & Lopez-Chavarrias, I. (2000). Discovering Similar Patterns in Time Series. In *proceedings of the 6th Int'l Conference on Knowledge Discovery and Data Mining*. Boston, MA. pp. 497-505.

[4] Cardle, M. (2004). Ph.D Thesis, in progress. University of Cambridge.

[5] Chang, C. L. E., Garcia-Molina, H. & Wiederhold, G. (2002). Clustering for Approximate Similarity Search in High-Dimensional Spaces. *IEEE Transactions on Knowledge and Data Engineering* vol. 14(4), July-August. pp. 792-808.

[6] Chen, L., Ozsu, T. & Oria, V. (2003). Symbolic Representation and Retrieval of Moving Object Trajectories. University of Waterloo. 2003.

[7] Chiu, B., Keogh, E. & Lonardi, S. (2003). Probabilistic Discovery of Time Series Motifs. In *proceedings of the 9th ACM SIGKDD Int'l Conference on Knowledge Discovery and Data Mining*. Washington DC, USA, Aug 24-27. pp. 493-498.

[8] Crochemore, M., Czumaj, A., Gasjeniec, L., Jarominek, S., Lecroq, T., Plandowski, W. & Rytter, W. (1994). Speeding Up Two String-Matching Algorithms. *Algorithmica* vol. 12(4/5). pp. 247-267.

[9] Dasgupta, D. & Forrest, S. (1999). Novelty Detection in Time Series Data Using Ideas from Immunology. In *proceedings of the 8th Int'l Conference on Intelligent Systems*. Denver, CO, Jun 24-26.

[10] Durbin, R., Eddy, S., Krogh, A. & Mitchison, G. (1998). Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids. Cambridge University Press.

[11] Faloutsos, C., Ranganathan, M. & Manolopulos, Y. (1994). Fast Subsequence Matching in Time-Series Databases. *SIGMOD Record* vol. 23(2), June. pp. 419-429.

[12] Gabglio, A. (1888). Theoria Generale Della Statistica, 2nd ed. Milan.

[13] Goldberger, A. L., Amaral, L. A., Glass, L., Hausdorff, J. M., Ivanov, P. C., Mark, R. G., Mietus, J. E., Moody, G. B., Peng, C. K. & Stanley, H. E. (2000). PhysioBank, PhysioToolkit, and PhysioNet: Componenets of a New Research Resource for Complex Physiologic Signals. *Circulation* vol. 101(23), June 13. pp. e215-e220 [Circulation Electronic Pages; http://circ.ahajournals.org/cgi/content/full/101/23/e215]

[14] Hochheiser, H. & Shneiderman, B. (2001). Interactive Exploration of Time-Series Data. In *proceedings of the 4th Int'l Conference on Discovery Science*. Washington D.C., Nov 25-28. pp. 441-446.

[15] Huang, Y. W. & Yu, P. S. (1999). Adaptive Query Processing for Time-Series Data. In *proceedings of the 5th ACM SIGKDD Int'l Conference on Knowledge Discovery and Data Mining*. San Diego, CA, Aug 15-18. pp. 282-286.

[16] Huettel, S., Mack, P. B. & McCarthy, G. (2002). Perceiving Patterns in Random Series: Dynamic Processing of Sequence in Prefrontal Cortex. *Nature Neuroscience* vol. 5. pp. 485-490.

[17] Isaac, D. & Lynnes, C. (2003). Automated Data Quality Assessment in the Intelligent Archive, White Paper prepared for the Intelligent Data Understanding program. 2003. pp. 17.

[18] Jin, X., Wang, L., Lu, Y. & Shi, C. (2002). Indexing and Mining of the Local Patterns in Sequence Database. In *proceedings of the 3rd Int'l Conference on Intelligent Data Engineering and Automated Learning*. Manchester, UK, Aug 12-14. pp. 68-73.

[19] Keim, D. A. (2002). Information Visualization and Visual Data Mining. *IEEE Transactions on Visualization and Computer Graphics* vol. 8(1). pp. 1-8.

[20] Keogh, E. The UCR Time Series Data Mining Archive. *http://www.cs.ucr.edu/~eamonn/tsdma/index.html*

[21] Keogh, E., Chakrabarti, K. & Pazzani, M. (2001). Locally Adaptive Dimensionality Reduction for Indexing Large Time Series Databases. *SIGMOD Record* vol. 30(2), June. pp. 151-162.

[22] Keogh, E. & Kasetty, S. (2002). On the Need for Time Series Data Mining Benchmarks: A Survey and Empirical Demonstration. In *proceedings of the 8th ACM SIGKDD Int'l Conference on Knowledge Discovery and Data Mining*. Edmonton, Alberta, Canada, July 23-26. pp. 102-111.

[23] Keogh, E. & Lin, J. (2004). Clustering of Time Series Subsequences is Meaningless: Implications for Previous and Future Research. *Knowledge and Information Systems Journal*. To Appear

[24] Keogh, E., Lonardi, S. & Chiu, B. (2002). Finding Surprising Patterns in a Time Series Database in Linear Time and Space. In *proceedings of the 8th ACM SIGKDD Int'l Conference on Knowledge Discovery and Data Mining*. Edmonton, Alberta, Canada, Jul 23-26. pp. 550-556.

[25] Kim, S., Kim, Y., Ahn, T., Nam, H. K., Han, B. J. & Kim, S. M. (2000). Visualysis: A Tool for Biological Sequence Analysis. In *proceedings of the 4th Int'l Conference on Computational Molecular Biology*. Tokyo, Japan, Apr 8-11.

[26] Lankford, J. P. & Quan, A. (2002). Evolution of Knowledge-Based Applications for Launch Support. In *proceedings of Ground System Architecture Workshop*. El Segundo, CA.

[27] Lin, J. VizTree Website. *http://www.cs.ucr.edu/~jessica/viztree.htm*

[28] Lin, J., Keogh, E., Lonardi, S. & Chiu, B. (2003). A Symbolic Representation of Time Series, with Implications for Streaming Algorithms. In *Workshop on Research Issues in Data Mining and Knowledge Discovery the 8th ACM SIGMOD*. San Diego, CA. June 13, 2003.

[29] Lin, J., Keogh, E., Patel, P. & Lonardi, S. (2002). Finding Motifs in Time Series. In *the 2nd Workshop on Temporal Data Mining the 8th ACM Int'l Conference on Knowledge Discovery and Data Mining*. Edmonton, Alberta, Canada. July 23-26, 2002.

[30] Ma, J. & Perkins, S. (2003). Online Novelty Detection on Temporal Sequences. In *proceedings of the 9th Int'l Conference on Knowledge Discovery and Data Mining*. Washington D.C., Aug 24-27.

[31] Oates, T. (1999). Identifying Distinctive Subsequences in Multivariate Time Series by Clustering. In *proceedings of the 5th Int'l Conference on Knowledge Discovery and Data Mining*. San Diego, CA, Aug 15-18. pp. 322-326.

[32] Oates, T., Schmill, M. & Cohen, P. (2000). A Method for Clustering the Experiences of a Mobile Robot that Accords with Human Judgements. In *proceedings of the 17th National Conference on Artificial Intelligence* pp. 846-851.

[33] Ohsaki, M., Sato, Y., Yokoi, H. & Yamaguchi, T. (2003). A Rule Discovery Support System for Sequential Medical Data, in the Case Study of a Chronic Hepatitis Dataset. In *Discovery Challenge Workshop the 14th European Conference on Machine Learning/the 7th European Conference on Principles and Practice of Knowledge Discovery in Databases*. Cavtat-Dubrovnik, Croatia. Sep 22-26, 2003.

[34] Park, S., Chu, W., Yoon, J. & Hsu, C. (2000). Efficient Searches for Similar Subsequences of Different Lengths in Sequence Databases. In *proceedings of the 16th IEEE Int'l Conference on Data Engineering*. San Diego, CA, Feb 28 - Mar 3. pp. 22-32.

[35] Reinert, G., Schbath, S. & Waterman, M. S. (2000). Probabilistic and Statistical Properties of Words: An Overview. *Journal of Computational Biology*. vol. 7. pp. 1-46.

[36] Shahabi, C., Tian, X. & Zhao, W. (2000). TSA-Tree: A Wavelet-Based Approach to Improve the Efficiency of Multi-Level Surprise and Trend Queries. In *proceedings of the 12th Int'l Conference on Scientific and Statistical Database Management*. Berlin, Germany, Jul 26-28. pp. 55-68.

[37] Shneiderman, B. (1996). The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations. In *proceedings of the IEEE Symposium on Visual Languages*. Boulder, CO, Sep 3-6. pp. 336-343.

[38] Tanaka, Y. & Uehara, K. (2003). Discover Motifs in Multi Dimensional Time-Series Using the Principal Component Analysis and the MDL Principle. In *proceedings of the 3rd Int'l Conference on Machine Learning and Data Mining in Pattern Recognition*. Leipzig, Germany, Jul 5-7. pp. 252-265.

[39] Tufte, E. R. (1983). The Visual Display of Quantitative Information. Cheshire, CT. Graphics Press.

[40] van Wijk, J. J. & van Selow, E. R. (1999). Cluster and Calendar Based Visualization of Time Series Data. In *proceedings of 1999 IEEE Symposium on Information Visualization*. San Francisco, CA, Oct 24-29. pp. 4-9.

[41] Weber, M., Alexa, M. & Muller, W. (2001). Visualizing Time Series on Spirals. In *proceedings of 2001 IEEE Symposium on Information Visualization*. San Diego, CA, Oct 21-26. pp. 7-14.

Improving IV&V Techniques Through the Analysis of Project Anomalies: Text Mining PITS issue reports - final report

Tim Menzies

Lane Department of Computer Science and Electrical Engineering, West Virginia University, USA
tim@menzies.us
http://menzies.us

Abstract This project is in two parts. The *second part* will try to combine two (or more) of the IV&V data sources into an active monitoring framework where data collected during an active IV&V project will trigger an alert if a project becomes unusual" (and defining "unusual" is one of the goals of this project).

But before we can generalize between sources, we need to study each source in isolation to determine its strengths and weaknesses. Hence, the *first part* of this project aims to gain experience with the various IV&V data sources available to researchers like myself; i.e.

- SILAP, from the IV&V planning and scoping team;
- James Dabney's Bayes networks that describe the IV&V business practices of the L3 IV&V contractor;
- The PITS issue tracking data;
- The LINKER database project that intends to join PITS to other data sources;
- Balanced score card strategy maps from NASA Langley.
- and the COCOMO data sets from JPL.

This is the second year of a three year project that started in June 2006. The project is data-rich project and much progress has already been achieved.

- At SAS'06, a preliminary report described what had been learned from the SILAP data. A ranking was offered on the most common IV&V work-breakdown structure (WBS) activities. This ranking can be used for (e.g.) identifying what WBS tasks would benefit most from optimization.
- This report on SILAP was finalized in the first part of 2007. In summary, there exists a very strong signal in the SILAP data for issue frequency and severity.
- In October'06, a preliminary report was delivered on the Bayes network. On a limited case study, it was shown that Bayes nets and treatment learning could generate parsimonious explanations for project events.
- A preliminary report on text mining from the PITS issues tracking database that generated an expert system which audited a test engineer's proposed severity level.

This document updates the preliminary PITS report. Before, the PITS report studied two projects with a limited range of severities (mostly severity 3 and 4). Here, we explore five projects with a much wider range of severities. The results from the PITS preliminary report is confirmed. Using text mining, PITS can be used to generate an expert system that audits a test engineer's proposed severity level for an issue.

Credits: This work was made possible due to the heroic efforts of Ken Costello (chief engineering at NASA IV&V) who provided the PITS defect reports. The text mining technology used here was inspired by the trace-ability work of Jane Hayes and Alex Dekhtyar. Alex was particularly helpful is mapping out the ABCs of text mining. Jane also offered extensive advice on how to extend the current system. This research was conducted at West Virginia University under NASA sub-contract project 100005549, task 5e, award 1002193r.

Cautions Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement by the United States Government.

Revision history : Much of the introductory and exposition text of this document comes from the preliminary report. However, all the results of this document have been recomputed for the five new projects. Also new in this report are the rule minimization results.

Contents

List of Figures

1 Introduction: We don't need another hero

NASA's software IV&V Program captures all of its findings in a database called the Project and Issue Tracking System (PITS). The data in PITS has been collected for more than 10 years and includes issues on robotic satellite missions and human-rated systems.

It is difficult, to say the least, to generate conclusions from a moving target like PITS. Several heroic studies have made significant conclusions using PITS data (see Figure 1). These studies were heroic in the sense that the "heroes" reached their goals after tedious and complex struggling. Worse, the extracted data was only accessible with the help of NASA civil servants- a scarce and expensive resource.

The problem with PITS is that there is a lack of consistency in how each of the projects collected issue data. In virtually all instances, the specific configuration of the information captured about an issue was tailored by the IV&V project to meet its needs. This has created consistency problems when metrics data is pulled across projects. While there was a set of required data fields, the majorities of those fields do not provide information in regards to the quality of the issue and are not very suitable for comparing projects.

NASA is very aware of the problems with PITS and is taking active steps to improve it. At the time of this writing, there is an on-going effort to implement a mandatory data set in each IV&V project database to support IV&V effectiveness metrics. This effort has been in development for about a year and is currently being executed by several projects. However, it is too early to make any useful observations from that data.

- Ken Costello (IV&V's chief engineer) compiled statistics for NASA headquarters that showing, in nine IV&V tasks, the majority of issues found by IV&V were found via an analysis of requirements documents.
- Marcus Fisher (IV&V's research lead) applied a "mid-course correction" to one IV&V project after checking the progress of the IV&V against historical records in PITS.
- David Raffo (University of Portland), working with Ken Costello and other civil servants, found enough cost data to partially tune his waterfall-based model of IV&V;
- In a prior report in this project, Melissa Northey (Project Manager) performed some joins across PITS to return costs for different IV&V tasks;

Fig. 1 A partial list of past heroic successes with PITS.

To be fair, PITS is hardly unique. Based on my experience with data mining at other corporations, I assert that PITS is a typical database, useful for storing day-to-day information and generating small-scale tactical reports (e.g. "list the bugs we found last Tuesday"), but difficult to use for high-end business strategic analysis (e.g.. "in the past, what methods have proved most cost effective in finding bugs?"). Like many other databases, it takes heroes to extract information from PITS. Sadly, most of the heroes I know are so busy saving their own part of the world that they have little time to save researchers like me.

Hence, in this report, we try a new approach for extracting general conclusions from PITS data. Unlike previous heroic efforts, our text mining and machine learning methods are low cost, automatic, and rapid. We find we can build an agent to automatically review issue reports and alert when a proposed severity is anomalous. Better, the way we generated the agent means that we have probabilities that the agent is correct. These probabilities can be used to intelligently guide decision making.

An extremely surprising conclusion from this report is that the unstructured text might be a better candidate for generating lessons learned than the structured data base fields.

- While the database fields in PITS keep changing, the nature of the unstructured text remains constant.
- In other words, the reason it is so hard in the past to reason about PITS is that we have been looking at the wrong data.

If we could properly understand unstructured text, this would be a result of tremendous practical importance. A recent study[1] concluded that

- 80 percent of business is conducted on unstructured information;
- 85 percent of all data stored is held in an unstructured format (e.g. the unstructured text descriptions of issues found in PITS);
- Unstructured data doubles every three months;

That is, if we can tame the text mining problem, it would be possible to reason and learn from a much wider range of NASA data than ever before.

[1] http://www.b-eye-network.com/view/2098

2 Concept of Operations

NASA uses a five-point scale to score issue severity. The scale ranges one to five, worst to dullest, respectively. A different scale is used for robotic and human-rated missions (see Figure 2 and Figure 3). The data used in this report comes from robotic missions.

Using text mining and machine learning methods, this report shows that it is possible to automatically generate a *review agent* from PITS issue reports via the process of Figure 4. This agent can check the validity of the severity levels assigned to issues:

- After seeing an issue in some *artifact*, a human *analyst* generates some text *notes* and assigned a severity level *severityX*.
- An agent learns a predictor for issue severity level from logs of $\{notes, severityX\}$. A *training* module (a) updates the agent beliefs and (b) determines how much *self-confidence* a *supervisor* might have in the *agent*'s conclusions.
- Using the learned knowledge, the *agent* reviews the *analysts*'s text and generates its own *severityY* level.
- If the agent's proposed *severityY* differs from the *severityX* level of the human *analyst*, then a human *supervisor* can decide to review the human *analyst*'s *severityX*. To help in that process, the *supervisor* can review the *self-confidence* information to decide if they trust the agent's recommendations.

This agent would be of useful under the following circumstances:

- When a less-experienced test engineer has assigned the wrong severity levels.
- When experienced test engineers are operating under urgent time pressure demands, they could use the agent to automatically and quickly audit their conclusions.
- For agents that can detect severity one and two-level errors with high probability, the agent could check for the rare, but extremely dangerous case, that an IV&V team has missed a high-severity problem.

Severity 1: Prevent the accomplishment of an essential capability; or jeopardize safety, security, or other requirement designated critical.
Severity 2: Adversely affect the accomplishment of an essential capability and no work-around solution is known ; or adversely affect technical, cost or schedule risks to the project or life cycle support of the system, and no work-around solution is known.
Severity 3: Adversely affect the accomplishment of an essential capability but a work-around solution is known; or adversely affect technical, cost, or schedule risks to the project or life cycle support of the system, but a work-around solution is known.
Severity 4: Results in user/operator inconvenience but does not affect a required operational or mission essential capability; or results in inconvenience for development or maintenance personnel, but does not affect the accomplishment of these responsibilities.
Severity 5: Any other issues.

Fig. 2 Severities for robotic missions.

Severity 1: A failure which could result in the loss of the human-rated system, the loss of flight or ground personnel, or a permanently disabling personnel injury.
Severity 1N: A failure which would otherwise be Severity 1 but where an established mission procedure precludes any operational scenario in which the problem might occur, or the number of detectable failures necessary to result in the problem exceeds requirements.
Severity 2: A failure which could result in loss of critical mission support capability.
Severity 2N: A failure which would otherwise be Severity 2 but where an established mission procedure precludes any operational scenario in which the problem might occur or the number of detectable failures necessary to result in the problem exceeds requirements.
Severity 3: A failure which is perceivable by an operator and is neither Severity 1 nor 2.
Severity 4: A failure which is not perceivable by an operator and is neither Severity 1 nor 2.
Severity 5: A problem which is not a failure but needs to be corrected such as standards violations or maintenance issues.

Fig. 3 Severities for human-rated missions.

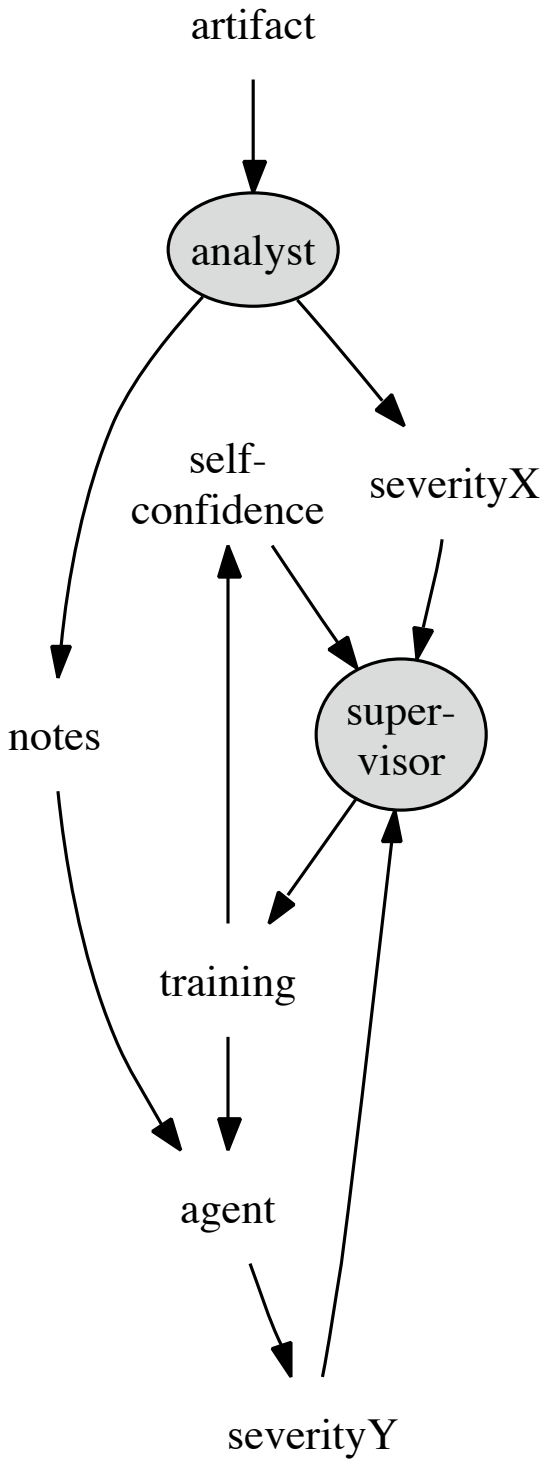

Fig. 4 An agent for reviewing issue severity levels. Gray nodes denote humans.

3 How it Works

The essential problem of text mining is *dimensionality reduction*. Standard machine learners work well for instances that are nearly all fully described using dozens (or fewer) attributes [6]. But text mining applications (e.g. analyzing PITS detect reports) must process thousands of unique words, and any particular paragraph may only mention a few of them [1, 5]. Therefore, before we can apply machine learning to text mining, we have to reduce the number of dimensions (i.e. attributes) in the problem.

There are several standard methods for dimensionality reduction such as *tokenization, stop lists, stemming, Tf*Idf* and *InfoGain*. All these methods are discussed below.

3.1 Tokenization

Figure 5 shows the tokenizer used in this study:

- `Clean` replaces certain punctuation with blanks spaces.
- The file `bad.txt` contains some non-printable escape characters which are removed from the issue reports.
- Finally, `lowerCase` sends all text to lower case.

3.2 Stop lists

Another way to reduce dimensionality is to remove "dull" words via a *stop list* of "dull" words. Figure 6 shows a sample of the stop list used in this study. IV&V's chief engineer, Ken Costello, reviewed this list and removed "counting words" such as "one", "every", etc, arguing that "reasoning about number of events could be an important requirement". Figure 7 shows code for a stop-list function. Note that our code supports use a *keep list* of words we want to retain (but, in this study, the keep list was empty).

3.3 Stemming

Terms with a common stem will usually have similar meanings. For example, all these words relate to the same concept.

```
clean() {
      gawk '{gsub(Bad,""); print $0}' \
         Bad=`cat $Here/bad.txt` $1 |
      sed 's/[\"\.\*\/\\\(\)\{\}\[\]><\(\)]/ /g'
}
lowerCase() {
      tr A-Z a-z $1 ;
}
```

Fig. 5 Tokenization

```
a           about     across    again      against
almost      alone     along     already    also
although    always    am        among      amongst
amongst     amount    an        and        another
any         anyhow    anyone    anything   anyway
anywhere    are       around    as         at
...         ...       ...       ...        ...
```

Fig. 6 24 of the 262 stop words used in this study.

```
stops()     { gawk '
   NR==1 {
        while (getline < Stops)  Stop[$0] = 1;
        while (getline < Keeps)  Keep[$0] = 1;
    }
    { for(I=1;I<=NF;I++)
        if (Stop[$I] && ! Keep[$I])
            $I=" "
       print $0
    }'  \
      Stops="$Here/stop_words.txt" \
      Keeps="$Here/keep_words.txt"  \
     $1
}
```

Fig. 7 Applying a stop-list.

```
RULE                EXAMPLE
----                -------
ATIONAL ->  ATE     relational     ->  relate
TIONAL  ->  TION    conditional    ->  condition
                    rational       ->  rational
ENCI    ->  ENCE    valenci        ->  valence
ANCI    ->  ANCE    hesitanci      ->  hesitance
IZER    ->  IZE     digitizer      ->  digitize
ABLI    ->  ABLE    conformabli    ->  conformable
ALLI    ->  AL      radicalli      ->  radical
ENTLI   ->  ENT     differentli    ->  different
ELI     ->  E       vileli         ->  vile
OUSLI   ->  OUS     analogousli    ->  analogous
IZATION ->  IZE     vietnamization ->  vietnamize
ATION   ->  ATE     predication    ->  predicate
ATOR    ->  ATE     operator       ->  operate
ALISM   ->  AL      feudalism      ->  feudal
IVENESS ->  IVE     decisiveness   ->  decisive
FULNESS ->  FUL     hopefulness    ->  hopeful
OUSNESS ->  OUS     callousness    ->  callous
ALITI   ->  AL      formaliti      ->  formal
IVITI   ->  IVE     sensitiviti    ->  sensitive
BILITI  ->  BLE     sensibiliti    ->  sensible
```

Fig. 8 Some stemming rules.

```
stemming()  { perl $Here/stemming.pl $1 ; }
```

Fig. 9 Using a downloaded stemmer.

- CONNECT
- CONNECTED
- CONNECTING
- CONNECTION
- CONNECTIONS

Porter's stemming algorithm [4] is the standard stemming tool. It repeatedly replies a set of pruning rules to the end of words until the surviving words are unchanged. The pruning rules ignore the semantics of a word and just perform syntactic pruning (e.g. Figure 8).

Porter's stemming algorithm has been coded in any number of languages[2] such as the Perl *stemming.pl* used in this study (see Figure 9).

Stemming is the end of our pre-processing (a sequence that began with the `clean` function shown above). Recall that the complete sequence was

$$clean \rightarrow lowerCase \rightarrow stops \rightarrow stems$$

This full sequence is shown in Figure 10.

[2] `http://www.tartarus.org/martin/PorterStemmer`

```
selectColumns() {
    gawk -F, '$3 {OFS=","; print $4 "," $3}' -
}
cleans() {
    for i in $Files; do
        (cd $Dir; clean tab${i}5.csv 2> /dev/null |
        lowerCase |
        stops |
        stemming |
        selectColumns > $Temp/ok_$i
    done
}
Files="a b c d e
Dir=$Root/mine/trunk/doc/07/telling/data/raw
cleans
```

Fig. 10 Preparation.

```
#update counters for all words in the record
function train()  {
    Documents++;
    for(I=1;I<NF;I++) {
        if( ++In[$I,Documents]==1)
            Document[$I]++
        Word[$I]++
        Words++
   }
}
# computer tfidf for one word
function tfidf(i) {
    return Word[i]/Words*log(Documents/Document[i])
}
```

Fig. 11 tfidf.awk.

```
tfidf() {
    gawk  -f tfidf.awk --source '
      { train() }
  END { OFS=","; for(I in Word) print I, tfidf(I) } ' $1 ;
      } ' $1
}
tfidf | sort -t, -n +0 | tail -100
```

Fig. 12 Finding the 100 highest Tf*Idf words using the $tfidf.awk$ code of Figure 11.

*3.4 Tf*IDF*

Tf*Idf is shorthand for "term frequency times inverse document frequency". This calculation models the intuition that jargon usually contains technical words that appear a lot, but only in a small number of paragraphs. For example, in a document describing a space craft, the terminology relating to the power supply may be appear frequently in the sections relating to power, but nowhere else in the document.

Calculating Tf*Idf is a relatively simple matter. If there be $Words$ number of document and each word I appear $Word[I]$ number of times inside a set of $Documents$ and if $Document[I]$ be the documents containing I, then:

$$Tf{*}Id = Word[i]/Words{*}log(Documents/Document[i])$$

The standard way to use this measure is to cull all but the k top Tf*Idf ranked stopped, stemmed tokens. This study used $k = 100$ (see Figure 11 and Figure 12).

3.5 InfoGain

According to the $InfoGain$ measure, the *best* words are those that *most simplifies* the target concept (in our case, the distribution of severities). Concept "simplicity" is measured using information theory. Suppose a data set has 80% severity=5 issues and 20% severity=1 issues. Then that data set has a class distribution C_0 with classes $c(1) = severity5$ and $c(2) = severity1$ with frequencies $n(1) = 0.8$ and $n(2) = 0.2$. The number of bits required to encode an arbitrary class distribution C_0 is $H(C_0)$ defined as follows:

$$\left.\begin{array}{r} N = \sum_{c \in C} n(c) \\ p(c) = n(c)/N \\ H(C) = -\sum_{c \in C} p(c) log_2 p(c) \end{array}\right\} \quad (1)$$

If A is a set of attributes, the number of bits required to encode a class after observing an attribute is:

$$H(C|A) = -\sum\nolimits_{a \in A} p(a) \sum\nolimits_{c \in C} p(c|a) log_2(p(c|a)$$

The highest ranked attribute A_i is the one with the largest *information gain*; i.e the one that most reduces the encoding required for the data *after* using that attribute; i.e.

$$InfoGain(A_i) = H(C) - H(C|A_i) \quad (2)$$

where $H(C)$ comes from Equation 1. In this study, we will use InfoGain to find the top $N \in \{100, 50, 25, 12, 6, 3\}$ most informative tokens.

3.6 Rule Learning

A data miner was then called to learn rules that predict for the severity attribute using the terms found above. The learner used here was a JAVA version of Cohen's RIPPER rule learner [2, 7]. RIPPER is useful for generating very small rule sets. The generated rules are of the form $if \rightarrow then$:

$$\underbrace{Feature_1 = Value_1 \wedge Feature_2 = Value_2 \wedge \ldots}_{condition} \longrightarrow \underbrace{Class}_{conclusion}$$

RIPPER, is a *covering* algorithm that runs over the data in multiple passes. Rule covering algorithms learns one rule at each pass for the *majority class*. All the examples that satisfy the conditions are marked as *covered* and removed from the data set. The algorithm then recurses on the remaining data. The output of a rule covering algorithm is an ordered *decision list* of rules where $rule_j$ is only tested if all conditions in $rule_{i<j}$ fail.

One way to visualize a covering algorithm is to imagine the data as a table on a piece of paper. If there exists a clear pattern between the features and the class,define that pattern as a rule and cross out all the rows covered by that rule. As covering recursively explores the remaining data, it keeps splitting the data into:

- what is easiest to explain, and
- any remaining ambiguity that requires a more detailed analysis.

```
  a   b   c   d   <-- classified as
--- --- --- ---
321  12  21   0 |   a = 1
157  41   8   1 |   b = 2
 49   3 259   0 |   c = 3
 21   1   2   2 |   d = 4
```

Fig. 13 Sample 10-way classification results.

$precision$	$pd = recall$	f-measure	severity
0.893	0.833	0.862	1
0.586	0.907	0.712	2
0.719	0.198	0.311	3
0.667	0.077	0.138	4

Fig. 14 Some precision, recall, f-measures from Figure 13.

3.7 Assessing the Results

It is a methodological error to assess the rules learned from a data miner using the data used in training. Such a *self-test* can lead to an over-estimate of the value of that model.

Cross-validation, on the other hand, assesses a learned model using data *not* used to generate it. The data is divided into, say, 10 buckets. Each bucket is set aside as a test set and a model is learned from the remaining data. This learned model is then assessed using the test set. Such cross-validation studies are the preferred evaluation method when the goal is to produce predictors intended to predict future events [7].

Mean results from a 10-way cross-validation can be assessed via a *confusion matrix* such as Figure 13. In that figure, some rule learner has generated predictions for classes {a,b,c,d} which denote issues of severity {1,2,3,4} (respectively). As shown top left of this matrix, the rules correctly classified issue reports of severity=1 as severity=1 321 times (mean results in 10-way cross-val). However, some severity=1 issues were incorrectly classified as severity=2 and severity=3 in 12 and 21 cases (respectively).

A confusion matrices can be summarized as follows. Let $\{A, B, C, D\}$ denote the true negatives, false negatives, false positives, and true positives (respectively). When predicting for class "a", then for Figure 13:

- A are all the examples where issues of severity=1 were classified as severity=1; i.e. A=321.
- B are all the examples where lower severity issues were classified as severity=1; i.e. B=157+49+21;
- C are all the examples where severity=1 issues were classified as something else; i.e. C=21+12
- D are the remaining examples; i.e. D=41+8+1=2+259+0+1+2+2.

A, B, C, D can be combined in many ways. Recall (or pd) comments on how much of the target was found.

$$pd = recall = D/(B + D) \tag{3}$$

Precision (or $prec$) comments on how many of the instances that triggered the detector actually containing the target concept.

$$prec = precision = D/(D + C) \tag{4}$$

The f-measure is the harmonic mean of precision and recall. It has the property that if *either* precision or recall is low, then the f-measure is decreased. The f measure is useful for dual assessments that include *both* precision and recall.

$$\textit{f-measure} = \frac{2 \cdot prec \cdot pd}{prec + pd} \tag{5}$$

Note that all these measures fall in the range

$$0 \leq \{pd, prec, f\} \leq 1$$

Also, the *larger* these values, the better the model. Figure 14 shows the $precision$, $recall$, and f-measure values for Figure 13.

4 Results

4.1 Data

The above methods where applied to {a,b,c,d,e}, five anonymous PITS projects supplied by Ken Costello (see Figure 15). All these systems were robotic. Note that this data has no severity one issues (these are quite rate and few severity five issues (these often not reported since they have such a low priority).

data set	severity	number
a	1	0
	2	311
	3	356
	4	208
	5	26
b	1	0
	2	23
	3	523
	4	382
	5	59
c	1	0
	2	0
	3	132
	4	180
	5	7
d	1	0
	2	1
	3	167
	4	13
	5	1
e	1	0
	2	24
	3	517
	4	243
	5	41

Fig. 15 Data sets in the this study.

4.2 Stopping and Stemming

Figure 16 shows some disappointing results for stopping and stemming. In these data sets, stopping and stemming methods barely reduced the number of tokens.

*4.3 Tf*Idf*

Tf*Idf proved to be more powerful than stopping or stemming. Figure 17 shows that in all data sets, there exist a very small number of words with high Tf*Idf scores. These top 100 terms are shown in Figure 18 and Figure 19. We have shown these lists to domain experts but, to date, we have not found any particular domain insights from these words. However, as shown below, even if we don't understand *why* these terms were chosen, they are can be used very effectively for the task of predicting issue severity.

4.4 Learning

The issue reports for each data set were then rewritten as frequency counts for those top 100 tokens (with the severity value for each record written to the end of line- see Figure 20). As shown in Figure 21 the resulting data sets are quite sparse. This figure shows the distributions of the frequency counts of the cells in the data sets. Note that most cells have a zero frequency count; 10% of the cells have a frequency count of one, and frequency counts higher than 10 occur in only $\frac{1}{100}\%$ of cells, or less.

Figure 22 shows the rules and confusion matrix see when learning from the top 100 tokens of data set "a". This rule

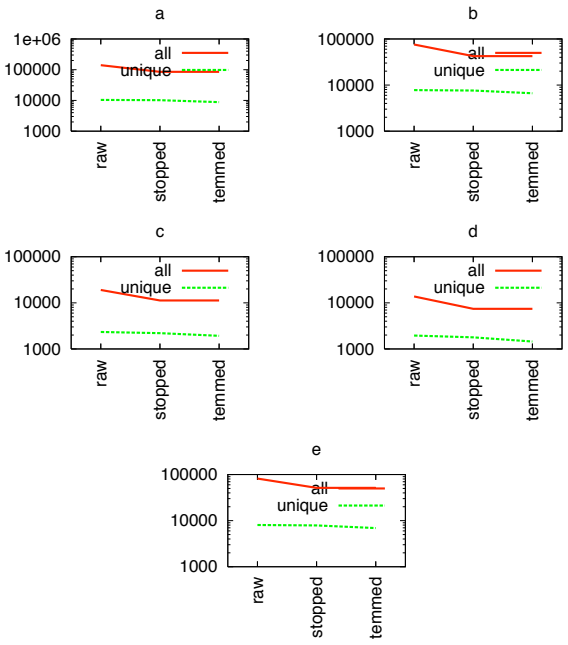

Fig. 16 Effects of stopping and stemming.

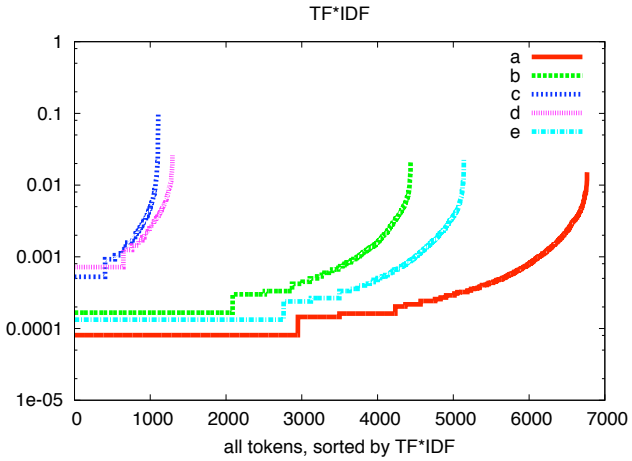

Fig. 17 Tf*Idf scoring for the stopped, stemmed tokens. Note that most tokens can be ignored since they have very low Tf*Idf scores.

	data set				
rank	a	b	c	d	e
1	rvm	fsw	softwar	switch	convent
2	sr	declar	fsw	default	the
3	script	requir	specifi	statement	capabl
4	engcntrl	arrai	command	contain	state
5	set	sr	parent	case	interfac
6	differ	parent	sc	trace	control
7	cdh	comment	trace	code	word
8	l4	us	ground	line	declar
9	indic	verifi	perform	violat	variabl
10	verifi	step	section	comment	line
11	section	gce	spec	detail	fsw
12	link	ac	matrix	avion	inst5
13	flight	scenario	cdh	spacecraft	document
14	paramet	defin	initi	appear	conduct
15	state	valid	who/what	would	septemb
16	obc	base	pip	data	set
17	system	command	child	downward	hardwar
18	onli	valu	tim	fsw	artifact
19	spacecraft	els	s919-er2342	defin	stp
20	all	test	icd	presum	condit
21	trace	includ	mu	pixel	compon
22	check	onli	spacecraft	mask	icd
23	vm	complet	verif	spec	sr
24	softwar	state	glori	process	version
25	bootload	control	comm	fpa	check
26	capabl	all	traceabl	packet	releas
27	number	page	data	on	flight
28	vml	alloc	configur	collater	statu
29	sequenc	caus	card	scienc	implement
30	l3	verif	channel	sc	current
31	specif	flag	artifact	oper	number
32	issu	inform	downlink	logic	specifi
33	oper	fail	valid	mode	power
34	messag	trace	mechan	tabl	l4
35	support	detail	satellit	document	rqmt
36	defin	read	system	initi	list
37	fp	pse	oper	ffi	macro
38	address	ivv	includ	collect	unsign
39	code	function	instrument	onli	messag
40	uplink	condit	launch	column	assign
41	document	interfac	possibl	black	fault
42	command	on	adac	within	short
43	task	tabl	scienc	zero	test
44	rt	thruster	electr	and/or	rev
45	note	document	tp	point	time
46	monitor	initi	mode	apertur	design
47	ground	true	safehold	support	mode
48	accept	the	note	fault	jpl
49	load	in	_vbuf	exist	clear
50	initi	fprintfsetup	common2/includ/vbufh	/line	data

Fig. 18 Top 1 to 50 terms found by TF*IDF, sorted by infogain.

	data set				
rank	a	b	c	d	e
51	calcul	correct	'common2 hdlclite hdlclitec'	cadenc	oper
52	attitud	rate	struct	dure	int
53	telemetri	two	refer	all	paramet
54	fsw	end	list	store	tefsw
55	within	reset	store	csc	rafsw
56	point	req	packet momentum		read
57	dure	telemetri	all	fine	all
58	log	packet	telemetri	target	inst6
59	packet	address	point	kav	refer
60	receiv	buffer	ap	protect	inst9
61	rate	counter	syspciinbyt	second	function
62	fault	list	interfac	autonom	reset
63	event	uint32	'common2 gener tlm_cmdc'	capabl	inst4
64	reset	rvtm	'glori cdh comm genportc'	level	command
65	process	error	'common2 gener com_cmdc'	note	configur
66	mode	level	detail	manag	enabl
67	refer	issu	rate	ground	dl
68	memori	note	em_map_pageunsign	command	us
69	engin	can	common2/includ/emsfunch	ccd	code
70	error	data	int	implic	trace
71	data	plan	collect	bin	long
72	second	sdn	'common2 router routerc'	long	flow
74	enabl	paramet	subsystem	appar	discret
75	perform	number	valu	perform	chart
76	ac	algorithm	'common2 vbuf vbufc'	short	c
77	transit	specifi	short	flight	b
78	includ	execut	compar	engin	gener
79	design	exampl	control	field	posit
80	time	void	capabl	set	spacecraft
81	execut	time	hlite_freestruct	hardwar	section
82	arrai	line	us	valu	m
83	specifi	current	soh	softwar	defin
84	control	set	float	implement	case
85	respons	review	char	enter	fail
86	current	indic	vbuf_freestruct	respect	call
87	checksum	case	power	could	initi
88	interrupt	variabl	unsign	nim	l5
89	power	specif	accuraci	smear	actuat
90	case	chang	commun	us	descript
91	tabl	section	asec	fg	tabl
92	singl	code	document	signal	subsystem
93	list[	statement	'common2 gener mm_utilc'	segment	valu
94	dump	instanc	long	fulli	softwar
95	us	updat	specif	error	issu
96	valu	procedur	compon	safe	error
97	scrub	need	orbit	design	switch
98	safe	check	ignor	requir	level
99	procedur	softwar	_hlite_cntl_blk	check	requir
100	word	charact	common2/includ/hl_protoh	period	temperatur

Fig. 19 Top 51 to 100 terms found by TF*IDF, sorted by infogain.

```
NR ==1 {
# grab the words we want to count
while (getline < "top100") Want[$0] = 1;
    # write the header
    for(I in Want)
            printf("%s,",I);
    print "severity"
}
NR > 1 { # rewrite each record as counts of "Want"
        gsub(/ /,"",$2);  counts($1,Want,  $2)
}
function counts(str,want,klass,   sum,out,i,j,n,tmp,got) {
    n=split(str,tmp," ");
    for(i=1;i<=n;i++)
        if (tmp[i] in want)
            got[tmp[i]]++;
    for(j in want) {
        sum += got[j]
        out = out got[j]+0 ",";
    }
    if (sum)
        print out "_" klass
}
```

Fig. 20 Re-writing issue reports as frequency counts.

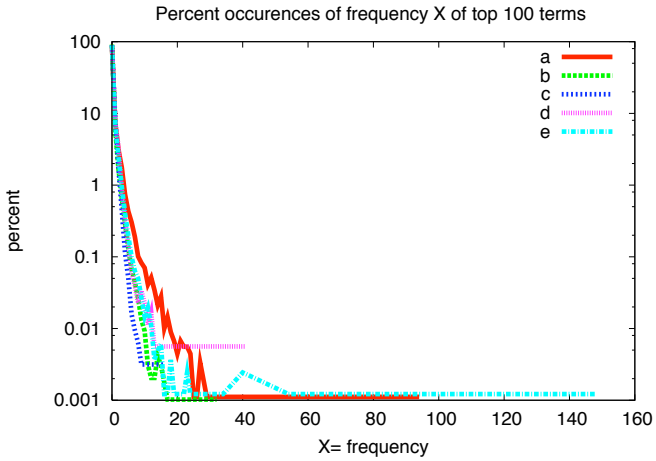

Fig. 21 Frequency counts seen in the cells of our data. Note that our data is mostly sparse: only 10% of the time (or less) were there frequency counts at or over 1. In most data sets, frequency counts of 0 (i.e. empty cell) appeared in 90% of the cells..

set is a little hard to read so Figure 23 shows the same experiment, but only using the top 3 ranked tokens. In those rules `sr` is a stemmed version of `srs`; i.e. systems requirements specification. Note that the rules of Figure 22 use only a subset of the 100 terms in the data set. That is, for data set "a", there exists a handful of terms that most predict for issue severity. Similar results hold for same results repeat for data sets {b,c,d,e} (see Figure 24 to Figure 31). That is, even when learning from all 100 tokens, most of the rules use a few dozens terms or less. Even though few tokens were used, in many cases, the f measures are quite large:

- Data set "a", for issues of severity=2, $f = 78...82\%$;
- Data set "a", for issues of severity=3, $f = 69...71\%$;
- Data set "b", for issues of severity=3, $f = 70...71\%$;
- Data set "c", for issues of severity=3, $f = 80...92\%$;
- Data set "c", for issues of severity=4, $f = 86...92\%$;
- Data set "d", for issues of severity=3, $f = 96...98\%$;
- Data set "d", for issues of severity=4, $f = 87...87\%$;
- Data set "e", for issues of severity=3, $f = 79...80\%$;

These results are better than they might first appear:

- These results are listed in the format, e.g. of $f = 79...80\%$. and show the results from using the $N = 3...N = 100$ tokens. Note how using just a vanishingly small number of tokens performed nearly as well as using a much larger number of tokens.
- Recall that these are all results from a 10-way cross-validation which usually over-estimates model error [3], That is, the real performance values are *higher* than the values shown above.

For other severities, the results are not as positive. Recalling Figure 15, none of our data sets had severity=1 errors so the absence of severity=1 results in the above list is not a concern. However, not all datasets resulted in good predictors for severity=2 errors. In all cases where this was observed, the data set had very few examples of such issues:

- Data set “b”, only has 22 records of severity=2;
- Data set “c”, has zero records of severity=2;
- Data set “d”, only has 1 record of severity=2;
- Data set “e”, only has 21 record of severity=2;

5 Discussion

Over the years, the Project Issue Tracking System (PITS) has been extensively and repeatedly modified. Prior attempts at generating generalized conclusions from PITS have required significant levels of manual, hence error-prone, processing.

Here, we show that conclusions can be reached from PITS without heroic effort. Using text mining and machine learning methods, we have shown that it is possible to automatically generate predictors for severity levels from the free text entered into PITS.

Better yet, our rules are self-certifying. Our data mining generation methods builds the rules and prints performance statistics (the confusion matrix). With those statistics, these rules support the following dialogue:

> *Tim wrote the problem report and he says this is a severity 5 issue. But the agent says that its a severity 3 issue with probability 83%. Hmmm... the agent seems pretty sure of itself- better get someone else to take a look at the issue.*

When this work began, we thought that we were conducting a baseline text mining experiment that would serve as a (low) baseline against which we could assess more sophisticated methods. However, for data sets with more that 30 examples of high severity issues, we always found good issue predictors (with high f-measures).

Further, we did so using surprisingly little domain knowledge. In call cases where large f-measures were seen using the top 100 terms, similar f measures were seen when using 3 terms. This is a very exciting result since it speaks to the *usability* of this work. It would be a simple to matter to apply these rules. E Given that a few frequency counts are enough to predict for issue severity, even a manual method would suffice.

We end this report with one caution. As seen in Figure 22 to Figure 31, the learned predictors are different for different data sets. We hence recommend adding these text mining tools to PITS and, on a regular basis, generate new rules relevant to just one project.

References

1. R. A. Baeza-Yates and B. Ribeiro-Neto, editors. *Modern Information Retrieval.* Addison-Wesley, 1999.
2. W. Cohen. Fast effective rule induction. In *ICML'95*, pages 115–123, 1995. Available on-line from `http://www.cs.cmu.edu/~wcohen/postscript/ml-95-ripper.ps`.
3. J. Demsar. Statistical comparisons of clasifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006. Avaliable from `http://jmlr.csail.mit.edu/papers/v7/demsar06a.html`.
4. M. Porter. An algorithm for suffix stripping. In K. S. Jones and P. Willet, editors, *Readings in Information Retrieval, San Francisco: Morgan Kaufmann*. 1997.
5. G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval.* McGraw Hill, 1983.
6. I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations.* Morgan Kaufmann, 1999.
7. I. H. Witten and E. Frank. *Data mining. 2nd edition.* Morgan Kaufmann, Los Altos, US, 2005.

```
     if  CONDITION                                                          then  SEVERITY  USED / INCORRECT
     if (script <= 0) and (section >= 2) and (14 >= 1) and (cdh >= 1) then    4      35.0 /   0.0
else if (sr <= 1) and (issu >= 1) and (code >= 3)                     then    4      12.0 /   1.0
else if (sr >= 2) and (rvm >= 1)                                      then    2     183.0 /   9.0
else if (sr >= 2) and (14 >= 1)                                       then    2      55.0 /   1.0
else if (within >= 2) and (state <= 0) and (system <= 0)              then    2      22.0 /   2.0
else if (verifi >= 1) and (fsw >= 1)                                  then    2      10.0 /   1.0
else if (control >= 1) and (code >= 1) and (attitud >= 4)             then    2       5.0 /   0.0
else if (13 >= 2) and (obc <= 0) and (perform <= 0)                   then    2      19.0 /   7.0
else if (script >= 1) and (trace >= 1)                                then    2       3.0 /   0.0
else if true                                                          then    3     554.0 / 219.0

   a   b   c   d   <-- classified as
 321  12  21   0 |   a = 3
 157  41   8   1 |   b = 4
  49   3 259   0 |   c = 2
  21   1   2   2 |   d = 5
```

Fig. 22 Data set “a”; top 100 tokens; learned rules.

```
     if  CONDITION             then  SEVERITY  USED / INCORRECT
     if (rvm <= 0) and (sr = 3) then     4       52.0 /  21.0
else if (sr >= 2)              then     2      289.0 /  54.0
else if true                   then     3      557.0 / 245.0


  a   b   c   d   <-- classified as
314  13  27   0 |   a = 3
158  25  24   0 |   b = 4
 69  10 232   0 |   c = 2
 25   0   1   0 |   d = 5
```

Fig. 23 Data set “a”; top 3 tokens; learned rules.

```
     if  CONDITION                                                        then  SEVERITY  USED / INCORRECT
     if (arrai >= 2) and (line >= 1) and (us <= 1)                        then    2       7.0 /   0.0
else if (base >= 4)                                                       then    2       3.0 /   0.0
else if (fsw <= 0) and (declar >= 1)                                      then    4      86.0 /  26.0
else if (fsw <= 0) and (complet >= 1) and (section <= 0)                  then    4      27.0 /   4.0
else if (fsw <= 0) and (statement >= 1) and (need <= 0) and (valu <= 0) then    4      36.0 /  10.0
else if true                                                              then    3     819.0 / 332.0

   a   b   c   d   <-- classified as
 120 253   0   4 |   a = 4
  69 445   0   7 |   b = 3
  11  47   0   0 |   c = 5
   2   9   0  11 |   d = 2
```

Fig. 24 Data set “b”; top 100 tokens; learned rules.

```
     if  CONDITION                  then  SEVERITY  USED / INCORRECT
     if (fsw <= 0) and (declar >= 1) then    4      94.0 /  34
else if true                        then    3     884.0 / 383.0

   a   b   c   d   <-- classified as
  60 317   0   0 |   a = _4
  20 501   0   0 |   b = _3
   3  55   0   0 |   c = _5
  11  11   0   0 |   d = _2
```

Fig. 25 Data set “b”; top 3 tokens; learned rules.

```
     if  CONDITION                                  then  SEVERITY  USED / INCORRECT
    if (section >= 2) and (matrix >= 1) and (icd >= 1) then      5      7.0 /  1.0
else if (softwar >= 1)                              then      3     95.0 /  3.0
else if (parent <= 0) and (trace <= 0)              then      3     48.0 / 14.0
else if true                                        then      4    167.0 /  4.0

   a   b   c   <-- classified as
 162  14   4 |   a = _4
   7 123   0 |   b = _3
   2   1   4 |   c = _5
```

Fig. 26 Data set “c”; top 100 tokens; learned rules.

```
     if  CONDITION              then  SEVERITY  USED / INCORRECT
     if  (softwar >= 1)         then      3     95.0 /  3.0
else if  true                   then      4    222.0 / 44.0

   a   b   c   <-- classified as
 169  11   0 |   a = _4
  37  93   0 |   b = _3
   6   1   0 |   c = _5
```

Fig. 27 Data set “c”; top 3 tokens; learned rules.

```
     if  CONDITION              then  SEVERITY  USED / INCORRECT
     if (switch >= 2)           then     4     11.0 / 1.0
else if true                    then     3    167.0 / 4.0)

   a   b   c   d   <-- classified as
   0   0   1   0 |   a = _2
   0  10   2   0 |   b = _4
   0   1 163   0 |   c = _3
   0   0   1   0 |   d = _5
```

Fig. 28 Data set “d”; top 100 tokens; learned rules.

```
     if  CONDITION              then  SEVERITY  USED / INCORRECT
     if  (switch >= 2)          then     4     11.0 / 1.0
else if  true                   then     3     167.0 / 4.0)

   a   b   c   d   <-- classified as
   0   0   1   0 |   a = _2
   0  10   2   0 |   b = _4
   0   1 163   0 |   c = _3
   0   0   1   0 |   d = _5
```

Fig. 29 Data set “d”; top 3 tokens; learned rules.

```
     if  CONDITION                                                  then  SEVERITY  USED / INCORRECT
     if (trace >= 1) and (test >= 1) and (case >= 3)                then     2     3.0  /   0.0
else if (error >= 1) and (line >= 2)                                then     2     3.0  /   0.0
else if (convent >= 2) and (declar <= 0) and (function <= 0)        then     5     6.0  /   0.0
else if (case >= 2) and (sr >= 3)                                   then     5     6.0  /   1.0
else if (refer >= 1) and (section >= 3) and (refer <= 1)            then     5     7.0  /   2.0
else if (the >= 1) and (fsw >= 2)                                   then     4    47.0  /   5.0
else if (control <= 0) and (convent >= 3)                           then     4    25.0  /   3.0
else if true                                                        then     3   723.0  / 214.0

   a   b   c   d   <-- classified as
   1  20   0   0 |   a = _2
   3 490   3  20 |   b = _3
   0  26   9   6 |   c = _5
   0 167   1  74 |   d = _4
```

Fig. 30 Data set “e”; top 100 tokens; learned rules.

```
     if  CONDITION                    then  SEVERITY  USED / INCORRECT
     if  (the >= 1) and (convent >= 3) then    4       30.0 /   8.0
else if  true                                  3      790.0 / 275.0

   a   b   c   d   <-- classified as
   0  21   0   0 |   a = _2
   0 515   0   1 |   b = _3
   0  34   0   7 |   c = _5
   0 222   0  20 |   d = _4
```

Fig. 31 Data set “e”; top 3 tokens; learned rules.