Incremental Discretization and Bayes Classifiers Handles Concept Drift and Scales Very Well

Tim Menzies, *Member,IEEE,* Andres Orrego

***Abstract*—Many data sets exhibit an *early plateau* where the performance of a learner peeks after seeing a few hundred (or less) instances. When concepts drift is slower than the time to find that plateau, then a simple windowing policy and an incremental discretizer lets standard learners like NaïveBayes classifiers to scale to very large data sets. Our tookit is simple to implment; can scale to millions of instances; works as many other data mining schemes; and, with trivial modifications, can be used to detect concept drift; to repair a theory after concept drift; can reuse old knowledge when old contexts re-occur and can detect novel inputs during unsupervised learning.**

***Index Terms*—data mining, concept drift, scale up, NaïveBayes classifiers, incremental, discretization, SAWTOOTH, SPADE, novelty detection**

I. Introduction

Holte argue for *simplicity-first* approach to data mining; i.e. researchers should try simpler methods before complicating existing algorithms [1]. In their review of methods for scaling up inductive algorithms, Provost and Kolluri endorse this "simplicity-first" approach. However, they add that "it is not clear now much *leverage* can be obtained through the use of simpler classifiers to guide subsequent search to address *specific deficiencies* in their performance" [2, p32].

This paper is a simplicity-first approach to scaling up data miners. Consider what *leverage* a simple NaïveBayes classifier offers for scaling up induction. Such classifiers have two desirable features for scalable induction: fast updates and small memory footprints. NaïveBayes classifiers summarize the training data in one frequncy table per class. Hence, they consume very little memory and they can quickly and incrementally modify their knowledge by incrementing the frequency count of attribute ranges seen in new training examples.

However, simple NaïveBayes classifiers have *specific deficiencies*. When learning from a large data set, it is common for the data generating phenomenon to change and standard NaïveBayes classifiers have no mechanisms for adapting to such *concept drift*. Also, when processing very large datasets, it can be impossible to scan it multiple times or store it all in in main memory. Unfortunately, like many classifiers, NaïveBayes assumes that all the data is held in a single memory-resident table [2, p32]. Also, NaïveBaues handles numeric attributes via either *discretization* or *kernel estimation* methods. Most known discretization and estimation methods for NaïveBayes require multiple passes through the data [3]–[5].

This paper reports an experiment in applying Holte's simplicity first approach to resolving these specific deficiencies. Experiments with *incremental cross-validation*, discussed below, showed that many data sets have a *early plateau* effect where classification accuracy plateaus after a relatively small number of examples (just a few hundred). This motivated scale-up experiments with a *windowing* scheme called SAWTOOTH that enables classification only when performance is rising or falling from a plateau. In accordance with the Holte doctrine, SAWTOOTH was build by considering multiple implementation options, then always implementing the simplest one. For example, the windowing scheme was added to a simple NaïveBayes classifier. Also, our *incremental discretization method*, called SPADE, was based on the simplest discretization method that we could find.

Manuscript received XXX, 2004, revised August YYY, 2004.

Tim Menzies is with Computer Science, Portland State University; tim@barmag.net. Dr. Menzies' web site is http://barmag.net

Andres Orrego is with Computer Science, West Virginia University, USA; Andres.S.Orrego@ivv.nasa.gov

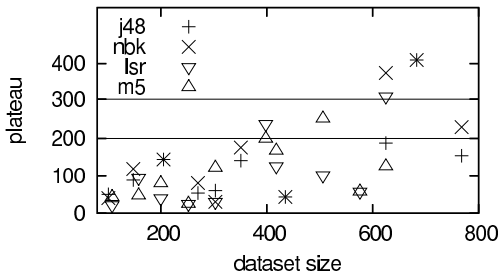

Fig. 1. *10*10* incremental cross validation experiments with J48 and Naive-Bayes (with kernel estimation) on {A:heart-c, B:zoo; C:vote; D:heart-statlog; E:lymph, F:autos. G:ionosphere, H:diabetes, I:balance-scale, J:soybean}; M5 and LSR on {K:bodyfat. L:cloud, M:fishcatch, N:sensory, O:pwLinear, Q:strike, R:pbc, S:autoMpg, T:housing}. All data sets from the UCI repository [8]. Data sets A..J have discrete classes and are scored via the *accuracy* of the learned theory; i.e % successful classifications. Data sets K..T have continuous classes and are scored by the *PRED(30)* of the learned theory; i.e. what % of the estimated values are within 30% of the actual value.

The experiment was quite successful. SAWTOOTH/SPADE can execute via one scan of the data; can scale to millions of instances; and works as well as many other schemes for scaling up data mining. With trivial modifications, SAWTOOTH/SPADE can also detect concept drift; repair a theory after concept drift; reuse old knowledge when old contexts re-occur; and could detect novel inputs during unsupervised learning.

SAWTOOTH/SPADE can be used as a simple *baseline* system to comparatively evaluate the merits of seemingly more sophisticated implementations. This evaluation has lead to the abandonment of certain lines of research. Elsewhere, we have explored novelty detection in unsupervised learning using a variety of complex methods: association rules to learn expected patterns in attribute values [6]; or SVDDs to recognize boundaries between expected and novel inputs [7]. SAWTOOTH/SPADE has now replaced those prior implementations, which we now view as needlessly complex.

The rest of this paper describes the early plateau effect, offers some background notes on NaïveBayes, SAWTOOTH/SPADE, then described experiments with numerous UCI data sets, some KDD cup data, and an aircraft flight simulator.

II. Early Plateau

A data set exhibits *early plateaus* when the peak performance of a learner requires just a few hundred instances. For datasets with plateaus, learning could proceed in *windows* of a few hundred instances. After a small number of windows, performance would peak and learning could be disabled. If the learner's performance falls off the plateau (i.e. due to concept drift), the learner could start afresh. Since learning only ever has to process a few hundred instances at a time, this approach should scale to very large data sets.

One way to find plateaus is via incremental R*N-way cross-validation. For $R = 10$ repeats, the order of the data is randomly shuffled. For each random ordering, the data is then divided *N=10* ways. Training is then conducted using the first $1 \leq i < N$ divisions and tested using remaining *N-i* divisions. As the size of the training set grows, the accuracy of the learned theory improves. At the plateau point, this improvement flattens out. After all the R repeats, this plateau point can be identified when learning on $\frac{Y}{10}$th of the data does not result in significantly greater accuracies than using $\frac{\{1..X|X<Y\}}{10}$ of the data (computed using t-tests with $\alpha = 0.05$).

The y-axis of Figure 1 show the plateau points seen in R*N-way cross validation experiments on 20 UCI data sets ($R = N = 10$)

using four different learners: J48; NBK: NaïveBayes with kernel estimation; M5, and LSR[1]. In all the experiments, a plateau was reached well before all the training instances were used. Most of the experiments ($\frac{33}{40}$) reached plateau in 200 instances or less. Further, only a handful ($\frac{3}{40}$) of experiments needed more than 300 instances to find their plateau.

This plateaus effect has been have been reported before (although this may be first report of early plateaus in M5' and LSR). Provost and Kolluri [2] make the general comment that the performance of some learners levels off quite early, without specifying exactly how early that might happen. They add that while some further accuracy improvements are seen as data set size increases, those improvements can be quite small. For example, Catlett reports differences of less than 1% (on average) between theories learned from 5000 or 2000 randomly selected instances in ten different data sets [12].

In another study, Oates and Jensen found plateaus in 19 UCI data sets using five variants of C4.5 [13]. In their results, six of their runs plateaued after seeing 85 to 100% of the data. This is much later than Figure 1 where none of our data sets needed more than 70% of the data. One possible reason for our earlier plateaus is the method used to identify start-of-plateau. Figure 1 detected plateaus using t-tests to compare performance scores seen in theories learned from M or N examples ($M < N$) and reported start-of-plateau if no significant (α=0.05) difference was detected between the N and the last M with a significant change. On the other hand, Oates and Jensen scanned the accuracies learned from $5, 10, 15\%$ etc. of the data looking for three consecutive accuracy scores that are within 1% of the score gained from a theory using all the available data. That is, Figure 1 shows the point where accuracies stop *improving* as training sets *grow* while Oates and Jensen report when accuracies start *degrading* as training sets *shrink*. Given sufficiently large standard deviations, our method will terminate on smaller training sets than Oates and Jensen.

The results below show that learning using our start-of-plateau detector can produce adequate classifiers that scale to very large data sets. Hence, at this time, we are not motivated to explore different methods for detecting start-of-plateau. In any case, regardless of *where* they found plateaus, Oates and Jensen's results endorse our general thesis that, often, learning need not process all the available examples. Rather, learning can jump through the available data in windows of a few hundred instances at a time.

III. WINDOWING

In a windowing system like SAWTOOTH, newly arrived examples are pushed into the start of sliding window of size W while the same number of older examples are popped from the end. Windowing systems need to select an appropriate window size W. If W is small relative to the rate of concept drift, then windowing guarantees the maintenance of a theory relevant to the last W examples. However, if W is too small, learning may never have find an adequate characterization of the target concept. Similarly, if W is too large, then this will slow the learner's reaction to concept drift.

Many windowing systems like SAWTOOTH and FLORA [14] select the window size dynamically: W grows till stable performance is reached; remains constant while performance is stable; then shrinks when concept drift occurs and performance drops. FLORA changes W using heuristics based on accuracy and other parameters that take into account the number of literals in the learnt theory. FLORA's authors comment that their heuristics are "very sensitive to the description language used". Hence, they claim that "it seems hopeless (or at least difficult) to make it completely free of parameters". This has not been our experience: SAWTOOTH uses the simple standardized test statistic of Equation 1 to determine window size. In all our experiments we have kept parameters of those tests constant.

$$-z(\alpha = 0.01) = -2.326 \leq \frac{\mu_j - \mu}{\frac{\sigma}{\sqrt{E}}} = \frac{S_j - (\frac{\sum_i S_i}{\sum_i E} * E)}{\frac{\sqrt{E * \frac{S_j}{E} * \left(1 - \frac{S_j}{E}\right)}}{\sqrt{E}}} \quad (1)$$

Equation 1 needs some explanation. A SAWTOOTH window is some integer number of eras of size E; i.e. $W = nE$ (default: *E=150* instances). SAWTOOTH windows grow until performance has not changed significantly in a $Stable$ (default: 2) number of eras. Each era is viewed as a binomial trial and each window is a record of trail results in the eras $1, ...i, j$ where era=j is the current era and era=1 is the first report of instability. Each era k holds S_k successful classifications and Equation 1 checks if the current era j is different to the proceedings eras $1..., i$.

On stability, SAWTOOTH disables theory updates, but keeps collecting the S statistics (i.e. keeps classifying new examples using the frozen theory). If stability changes to instability, SAWTOOTH shrinks W back to one era's worth of data and learning is then re-enabled.

IV. NAÏVEBAYES

One problem with windowing systems is the computational cost of continually re-learning. Hence SAWTOOTH uses a learner that can update its knowledge very quickly. Figure 2 shows the NaïveBayes classifier used by SAWTOOTH. The function `update` in that figure illustrates the simplicity of re-learning for a Bayes classifier: just increment a frequency table F holding counts of the attribute values seen in the new training examples.

In terms of scaling up induction, the most important property of Figure 2 is the `F` data structure that holds the frequency counts. A Bayes classifier only needs the memory required for the `F` frequency

```
# GLOBALS: "F": frequeny tables;     "I" : number of instances;
#          "C": how many classes?; "N": instances per class
function update(class,train)
  # OUTPUT: changes to the globals.
  # INPUT:  a "train"ing example containing attribute/value pairs
  #         plus that case's "class"
  I++;  if (++N[class]==1) then C++ fi
  for <attr,value>  in train
    if (value != "?")  then
      F[class,attr,range]++ fi
function classify(test)
  # OUTPUT: "what" is the most likely hypothesis for the test case.
  # INPUT:  a "test" case containing attribute/value pairs.
  k=1; m=2            # Control for Laplace and M-estimates.
  like = -100000      # Initial, impossibly small likelihood.
  for H  in N         # Check all  hypotheses.
  { prior = (N[H]+k)/(I+(k*C))                    #⇐P(H)
    temp  = log(prior)
    for <attr,value>  in attributes
    { if (value != "?")  then
      inc= F[H,attr,value]+(m*prior))/(N[H]+m)  #⇐P(E_i | H)
      temp += log(inc)  fi
    }
    if (temp >= like) then like = temp; what=class fi
  }
  return what
```

Fig. 2. A Bayes Classifier. "?" denotes "missing values". Probabilities are multiplied together using logarithms to stop numeric errors when handling very small numbers. The m and k variables handle low frequencies counts in the manner recommended by Yang and Webb [15, §3.1].

[1]All learners come from the WEKA [9]. LSR/M5' assumes values can be fitted to one/many (respectively) n-dimensional linear models [10]. J48 is a JAVA implementation of Quinlan's C4.5 decision tree learner [11].

counts plus a buffer just large enough to hold the `test` instance passed to Figure 2's `classify` function.

NaïveBayes classifiers are based on Bayes' Theorem. Informally, the theorem says $next = old{*}new$ i.e. what we'll believe *next* comes from how *new* evidence effects *old* beliefs. More formally:

$$P(H \mid E) = \frac{P(H)}{P(E)} \prod_i P(E_i \mid H)$$

i.e. given fragments of evidence E_i and a prior probability for a class $P(H)$, the theorem lets us calcuate a posteri probability $P(H \mid E)$. Technically, a Bayes classifier should return the class with highest probability. However, Figure 2 actually, computes class *likelihoods* not probabilities. Likelihoods become probabilities when they are normalized over the sum of all likelihoods. Since maximum probability comes from maximum likelihood, this code only needs to return the class with maximum likelihood. Note that unlikely instances have lower frequency counts and hence lower likelihoods. In the sequel, we will use this property of likelihoods to recognizing novel instances in unsupervised learning.

Bayes classifiers are called *naïve* since they assume that the frequencies of different attributes are independent. In practice [16], the absolute values of the classification probabilities computed by Bayes classifiers are often inaccurate. However, the relative ranking of classification probabilities is adequate for the purposes of classification. Many studies (e.g. [4], [17]) have reported that, in many domains, this simple Bayes classification scheme exhibits excellent performance compared to other learners.

Other researchers have explored incremental Bayes classifiers using modifications to the standard Bayes classifier: e.g. Gama alters the frequency counts in the summary tables according the success rate of the last N classifications [18] while Chai et.al. updates the priors via feedback from the examples seen up till now [19]. In contrast, we use standard Bayes classifiers *without* modification.

Bayes classifiers can be extended to numeric attributes using *kernel estimation* methods. The standard estimator assumes the central limit theorem and models each numeric attribute using a single gaussian. Other methods don't assume a single gaussian; e.g. John and Langley's gaussian kernel estimator models distributions of any shape as the sum of multiple gaussians [3]. Other, more sophisticated methods are well-established [20], but several studies report that even simple *discretization methods* suffice for adapting Bayes classifiers to numeric variables [4], [5].

John and Langley comment that their method must access all the individual numeric values to build their kernel estimator and this is impractical for large data sets. Many discretization methods violate the *one scan* requirement of a data miner: i.e. execute using only one scan (or less) of the data since there many not be time or memory to go back and look at a store of past instances. For example, Dougherty et.al.'s [4] *straw man* discretization method is *10-bins* which divides attribute a_i into bins of size $\frac{MAX(a_i)-MIN(a_i)}{10}$. If MAX and MIN are calculated incrementally along a stream of data, then instance data may have to be cached and re-discretized if the bin sizes change. An alternative is to calculate MAX and MIN after seeing *all* the data. Both cases require two scans through the data, with the second scan doing the actual binning. Many other discretization methods (e.g. all the methods discussed by Dougherty et.al. [4] and Yang and Webb [5]) suffer from this two-scan problem.

An incremental one scan (or less) discretization method is needed for scaling up induction. SAWTOOTH uses the SPADE method described below.

V. Handling Numeric Attributes with SPADE

Discretization converts continuous ranges to a set of bins storing the tally of numbers that fall into that bin. In order to process infinite streams of data, we developed a one-pass discretization method called SPADE (Single PAss Dynamic Enumeration).

SPADE only scans the input data once and, at anytime during the processing of X instances, SPADE's bins are available. Further, if it ever adjusts bins (e.g. when merging bins with very small tallies), the information used for that merging comes from the bins themselves, and not some second scan of the instances. Hence, it can be used for the incremental processing of very large data sets.

Unlike standard NaïveBayes classifiers, SPADE makes no assumptions about the underlying numeric distributions. SPADE is similar to *10-bins* but the MIN and MAX change incrementally. The first value N creates one bin and sets {MIN=N,MAX=N}. If a subsequent new value arrives inside the current {MIN,MAX} range, the bins from MIN to MAX are searched for an appropriate bin. Otherwise, a $SubBins$ number of new bins are created (default: SubBins=5) and MIN/MAX is extended to the new value. For example, here are four bins:

i	1	2	3	4	min	max
$border$	10	20	30	40	10	40

Each bin is specified by its lower *border* value. A variable N maps to the first/last bin if it is the current {MIN,MAX} value (respectively). Otherwise it maps to bin i where $border_i < N \leq border_{i+1}$. Assuming $SubBins = 5$, then if a new value $N = 50$ arrives, five new bins added above the old MAX to a new MAX=50:

i	1	2	3	4	5	6	7	8	9	min	max
$border$	10	20	30	40	42	44	46	48	50	10	50

If the newly created number of bins exceeds a *MaxBins* parameter (default=the square root of all the instances seen to date) then adjacent bins with a tally less than *MinInst* (default: same as *MaxBins*) are merged if the tally in the merged bins is less than a *MaxInst* parameter (default: 2**MinInst*). Preventing the creation of very few bins with big tallies is essential for a practical incremental discretizer. Hence, SPADE checks for merges only occasionally (at the end of each era), thuss allowing for the generation of multiple bins before they are merged.

SPADE runs as a pre-processor to `update` to NaïveBayes. Newly arrived numerics get placed into bins and it is this bin number that is used as the $value$ passed to `update` or Figure 2. Also, when SPADE merges bins, this causes a similar merging in frequncy tables entries (the F variable of Figure 2).

The opposite of merging would be to *split* bins with unusually large tallies. SPADE has no split operator since we did not know how to best divide up a bin *without* keeping per-bin kernel estimation data (which would be memory-expensive). Our early experiments suggested that adding $SubBins = 5$ new bins between old ranges and newly arrived out-of-range values was enough to adequately divide the range. Our subsequent experiments (see below) were so encouraging that we are not motivated to add a split operator.

Figure 3 compares results from SPADE and John and Langley's kernel estimation method using the display format proposed by Dougherty, Kohavi and Sahami [4]. In that figure, a 10*10-way cross validation used three learners: (a) NaïveBayes with a single gaussian for every numeric; (b) NaïveBayes with John and Langley's kernel estimation method (c) the Figure 2 NaïveBayes classifier using data pre-discretized by SPADE. Mean classification accuracies were collected and shown in Figure 3, sorted by the means $(c-a)-(b-a)$; that is, by the difference in the improvement seen in SPADE *or* kernel estimation *over and above* a simple single gaussian scheme. Hence,

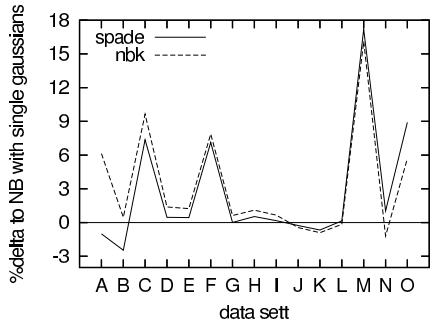

Fig. 3. Comparing SPADE and kernel estimation. Data sets: {*A=vowel, B=iris, C=ionosphere, D=echo, E=horse-colic, F=anneal, G=hypothyroid, H=hepatitis, I=heart-c, J=diabetes, K=auto-mpg, L=waveform-5000, M=vehicle, N=labor, O=segment*}.

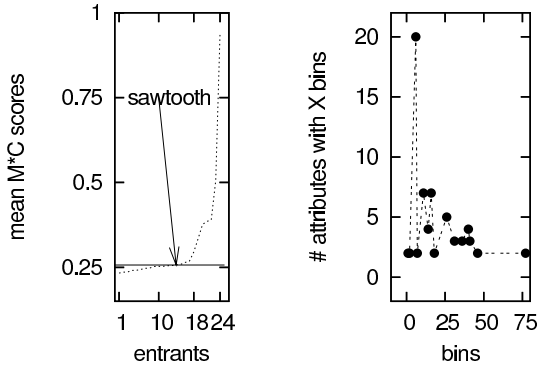

Fig. 4. SAWTOOTH and the KDD'99 data

the left-hand-side data sets of Figure 3 show examples where kernel estimation work comparatively better than SPADE while the right-hand-side shows results where SPADE did comparatively better.

Three features of Figure 3 are noteworthy. Firstly, in a finding consistent with those of Dougherty et.al. [4], discretization can sometimes dramatically improve classification the accuracy of a NaïveBayes classifier (by up to 9% to 15% in data sets C,F,M,0). Secondly, Dougherty et.al. found that even simple discretization schemes (e.g. 10-bins) can be competitive with more sophisticated schemes. We see the same result here where, in $\frac{13}{15}$ of these experiments, SPADE's mean improvement was within 3% of John and Langley's kernel estimation method. Thirdly, in two cases, SPADE's one scan method lost information and performed worse than assuming a single gaussian. In data set A, the loss was minimal (-1%) and in data set B SPADE's results were still within 3% of kernel estimation. In our view, the advantages of SPADE (incremental, one scan processing, distribution independent) compensates for its occasional performing worse than state-of-the-art alternatives which require far more memory.

VI. Experiments

In all the following experiments, SPADE was run continuously on all incoming data while SAWTOOTH worked on windows containing a variable number of eras. Also, when SAWTOOTH accuracies are reported, they are the accuracies seen on new instances *before* those instances update the frequency tables of the NaïveBayes classifier. That is, all the SAWTOOTH accuracies reported below come from data *not* (yet) used to train the classifier.

A. KDD'99 Data

In order to stress test our system, we ran it on the 5,300,000 instances used in the 1999 KDD cup[2]. KDD'99 dealt with network intrusion detection and was divided into a training set of about five million instances and a *test set* of 311,029 instances. The data comprised 6 discrete attributes, 34 continuous attributes, and 38 classes which fell into four main categories: *normal* (no attack); *probe* (surveillance and other probing); *DOS* (denial-of-service); *U2R* (unauthorized access to local super-user privileges); and *R2L* (unauthorized access from a remote machine).

The 24 KDD'99 cup entrants ran their learners to generated a matrix $M[i,j]$ showing the number of times class i was classified j. Entries were scored by computing the mean $M[i,j] * C[i,j]$ value where $C[i,j]$ was the cost of mis-classifying (e.g.) unauthorized access to super-user as (e.g.) just a simple probe. Note that $M * C$ are *mis-classification* scores so a *lower* score is better.

Figure 4 shows all the sorted *mean M*C scores* from the KDD'99 entrants. Also shown in that figure is SAWTOOTH's mean $M * C$ result. SAWTOOTH's results were close to the winning score of entrant #1; very similar to entrants 10,11,12,13,14,15,16; and much better than entrants 18,19,20,21,22,23,24. These results are encouraging since SAWTOOTH is a much simpler tool that many of the other entries. For example, the winning entrant took several runs to divide the data into smaller subsets and buid an ensemble of 50x10 C5 decision trees using an intricate cost-sensitive bagged boosting technique. This took more than a day to terminate on a dual-processor 2x300MHz Ultra-Sparc2 machine with 512MB of RAM using the commercially available implementation of C5, written in "C". In contrast, our toolkit written in interpreted scripting langauges (gawk/bash), processed all 5,300,000 instances in one scan of the data using less than 3.5 Megabytes of memory. This took 11.5 hours on a 2GHz Pentium 4, with 500MB of RAM, runing Windows/Cygwin and we conjecture that that this runtime could be greatly reduced by porting our toolkit to "C".

Another encouraging result is the *# attributes with X bins* plot of Figure 4. One concern with SPADE is that several of its internal parameters are linked to the number of processed instances; e.g. *MaxBins* is the square root of the number of instances. The 5,300,000 instances of KDD'99 could therefore generate an impractically large number of bins for each numeric attribute. This worst-case scenario would occur if each consecutive group of $SubBins$ number of numeric values has different values from the previously seen groups *and* they are sorted in ascending or descending order. If this unlikely combination of events does *not* occur then the resulting bins would have tallies than $MinInst$, encouraging it to merge with the next bin. In all our experiments, we have never seen this worst-case behavior. In KDD'99, for example, SPADE only ever generated 2 bins for 20 of the 40 attributes. Also, for only two of the attributes, did SPADE generate more than 50 bins. Further, SPADE never generated more than 100 bins for any attribute.

Attempts to test our system using other KDD cup data were not successful, for a variety of reasons[3].

[2]http://www.ai.univie.ac.at/~bernhard/kddcup99.html

[3]The KDD'04 evaluation portal was off-line during the period when SAWTOOTH was being developed. The KDD'03 problem required feature extration from free text- something that is beyond the scope of this research. The data for KDD'02 is no longer on-line. The KDD'01 had data with 130,000 attributes and we don't yet know how to extend our technqie to such a large attribute space. We had trouble following the KDD'00 documentation.

learner	win - loss	win	loss	ties
J48	10	25	15	20
NBK	8	17	9	34
SAWTOOTH	-6	12	18	30
NB	-12	9	21	30

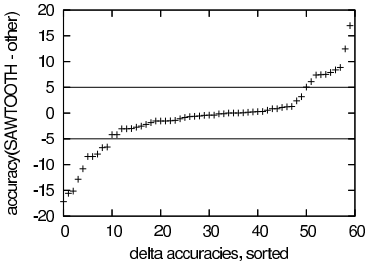

Fig. 5. SAWTOOTH exeucting on UCI data.

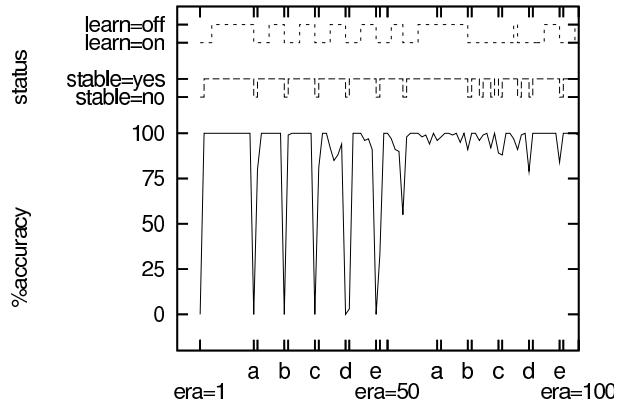

Fig. 6. SAWTOOTH and concept drift

B. UCI Data

Figure 4 explored SAWTOOTH's competencies on one large data set. Figure 5 explores SAWTOOTH's competency on many smaller data sets from the standard UCI database: {*anneal, audiology, auto-mpg, diabetes, echo, heart-c, hepatitis, horse-colic, hypothyroid, ionosphere, iris, labor, letter, primary-tumor, segment, soybean, vehicle, vote, vowel, waveform-500*}. Those data sets ranged in size from *labor*'s 57 instances to *letter*'s 20,000 instances. A standard 10*10 cross-validation experiment was conducted using SAWTOOTH/SPADE (using the Figure 2 code); or the J48 decision tree learner; or two NaïveBayes classifiers that used either a single gaussian to model continuous attributes (the "NB" learner) or a sum of gaussians (the "NBK" learner proposed by John and Langley [3]).

Using t-tests, significant differences ($\alpha = 0.05$) between the mean performance of each learner on the 20 data sets could be detected. Win/loss/ties statistics for each pair of learners on each data set was then collected. The results, shown top of Figure 5 shows SAWTOOTH performing marginally better than NB classifier but worse that both J48 and NBK. This is not surprising: Provost and Kolluri [2, p22] comment that sequential learning strategies like windowing usually performs worse than learning from the total set.

However, what is encouraging is the *size* of the difference in mean accuracies SAWTOOTH and the other learners. The plot shown bottom of Figure 5 sorts all those differences. In 80% of our experiments, SAWTOOTH performed within $\pm 5\%$ of other methods.

C. Data with Concept Drift

Figure 4 and Figure 5 showed SAWTOOTH processing static data. Figure 6 shows SAWTOOTH running on data with concept drift. To generate that figure, a flight simulator was executed where a airplane moved from a nominal mode to one of five error conditions (labeled *a,b,c,d,e*). Data was taken from the simulator in eras of size 100 instances. Each error mode lasted two eras and each such mode was encountered twice. The top of Figure 6 shows the results of SAWTOOTH's stability tests as well as when SAWTOOTH enabled or disabled learning. Each error mode introduced a period of instability which, in turn, enabled a new period of learning.

The *first* time SAWTOOTH saw a new error mode (at eras 15,23,31,39,and 47), the accuracy drops sharply and after each mode, accuracy returns to a high level (usually, over 80%). The *second* time SAWTOOTH returned to a prior error mode (at eras 63,71,79,87 and 95), the accuracies drop, but only very slightly.

Two features of Figure 6 are worthy of mention. Firstly, the large drop in accuracy when entering a new context means SAWTOOTH can be used to recognize new contexts (watch for the large drops). In terms of certifying an adaptive system, this is a very useful result: learning systems can alert their uses when they are *leaving the region of their past competency*. Secondly, and most importantly, there is no such large drop when SAWTOOTH returns to old contexts. That is, SAWTOOTH can *retain knowledge of old contexts* and reuse that knowledge *when contexts re-occur*.

D. Unsupervised Learning

Figure 4, Figure 5, and Figure 6 were all examples of *supervised learning*. In supervised learning (when each instance is stamped with a class symbol), handling concept drift means *recognizing* when the underlying data generating phenomenum has changed; and *repairing* the current classifier to cope with that change.

Figure 7 shows an *unsupervised learning* experiment (where instances lack any class symbol). In unsupervised learning, it no longer makes sense to *repair* the classifier since there are no classes to classify. However, the problem of *recognizing* novel situations remains.

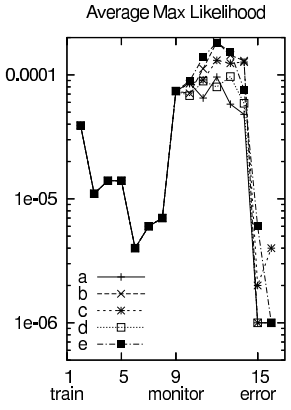

Fig. 7. Learning normal flight (eras 1 to 8); monitoring five different flights a,b,..e (eras 9 to 16); injecting errors into eras 15,16.

In the Figure 7 experiment, the class of all instances were replaced with a single label: *class*0. Eras one to eight of that figure show SAWTOOTH processing eight eras (of 100 instances) of nominal flight simulator data. Updating the frequency tables was then disabled and the system watched over five entirely different flights, each ending with one of our errors *a,b,c,d,e*. The `classify` routine of Figure 2 was modified to return the classification with the maximum likelihood *as well as* that maximum likelihood value. Figure 7 shows the average maximum likelihood seen in each era. In all cases, the era 15,16 errors dramatically changed the likelihoods: they dropped by *two orders of magnitude* from the pre-error values and they dropped *below* the liklihooods seen during training (eras 1 to 8).

This sudden drop in average maximum likelihoods could be used to recognize novel situations. Such a novelty detector could monitor (e.g.) an adapative controller for a jet fighter and propose switching to manual control (or bailing out) if the inputs are radically different to what has been seen before.

VII. Limitations

The premise of our system is that the dataset being processed comes from a data generating phenomena that with context drifts that are slower than the time required to reach plateau. If a particular data sets does not contain early plateaus then our simple toolkit should be exchanged for a more sophisticated scheme. Also, our toolkit is inappropriate if concept drift is occurring *faster* than the time required to collect enough instances to find the plateau.

Another drawback with out toolkit is that we can't guarantee that our learner operates in small constant time per incoming instance. Several of SPADE's internal parameters are functions of the total number of instances. In the worst case, this could lead to runaway generation of bins. On a more optimistic note, we note that this worst case behavior has yet to be observed in our experiments: usually, the number number of generated bins is quite small (see Figure 4).

Note that our scheme is designed for *large* data sets and so does not perform as well as other commonly used schemes when used on *smaller* data sets (but often achieves accuracy on small data sets within $\pm 5\%$ of other learners schemes- see Figure 5).

VIII. Conclusion

Before implementing seemingly more sophisticated schemes, it can be valuable to first build the simplest possible initial implementation. SAWTOOTH/SPACE were designed in accordance with Holte's simplicity-first methodology. SAWTOOTH/SPADE is much simpler than other data miners such as FLORA, the winner of KDD'99, or the SVDDs we used previously for detecting novel inputs [7]. Despite its simplicity, the system performs remarkably well. SPADE, plus SAWTOOTH executes via one scan of the data. Figure 4 shows one example where this method scales to millions of instances while working as well as many other schemes for scaling up data mining.

The system has other advantages:

- In Figure 3, the SPADE discretizer was applied to standard small UCI data sets. This discretizer performed nearly as well as other discretization methods *without* requiring multiple passes through the data.
- Figure 7 traced SAWTOOTH's behavior when used for novelty detection in unsupervised learning. Without even knowing the target classes of a system, a V&V agent could monitor the average maximum likelihood of the input examples. If that likelihood suddenly drops by orders of magnitude, then the agent could raise an alert that it is unlikely that the adaptative system is seeing inputs similar to what is has handled previously.

Acknowledgment

This research was conducted at West Virginia University, Portland State University, partially sponsored by the NASA Office of Safety and Mission Assurance under the Software Assurance Research Program led by the NASA IV&V Facility. 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.

References

[1] R.C. Holte, "Very simple classification rules perform well on most commonly used datasets," *Machine Learning*, vol. 11, pp. 63, 1993.

[2] Foster J. Provost and Venkateswarlu Kolluri, "A survey of methods for scaling up inductive algorithms," *Data Mining and Knowledge Discovery*, vol. 3, no. 2, pp. 131–169, 1999, Available from `http://citeseer.ist.psu.edu/provost99survey.html`.

[3] G.H. John and P. Langley, "Estimating continuous distributions in bayesian classifiers," in *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence Montreal, Quebec: Morgan Kaufmann*, 1995, pp. 338–345, Available from `http://citeseer.ist.psu.edu/john95estimating.html`.

[4] James Dougherty, Ron Kohavi, and Mehran Sahami, "Supervised and unsupervised discretization of continuous features," in *International Conference on Machine Learning*, 1995, pp. 194–202.

[5] Ying Yang and Geoffrey I. Webb, "A comparative study of discretization methods for naive-bayes classifiers," in *Proceedings of PKAW 2002: The 2002 Pacific Rim Knowledge Acquisition Workshop*, 2002, pp. 159–173.

[6] Y. Liu, T. Menzies, and B. Cukic, "Detecting novelties by mining association rules," 2003, Available from `http://menzies.us/pdf/03novelty.pdf`.

[7] Yan Liu, Srikanth Gururajan, Bojan Cukic, Tim Menzies, and Marcello Napolitano, "Validating an online adaptive system using svdd," in *IEEE Tools with AI*, 2003, Available from `http://menzies.us/pdf/03svdd.pdf`.

[8] C.L. Blake and C.J. Merz, "UCI repository of machine learning databases," 1998, URL: `http://www.ics.uci.edu/~mlearn/MLRepository.html`.

[9] I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*, Morgan Kaufmann, 1999.

[10] J. R. Quinlan, "Learning with Continuous Classes," in *5th Australian Joint Conference on Artificial Intelligence*, 1992, pp. 343–348, Available from `http://citeseer.nj.nec.com/quinlan92learning.html`.

[11] R. Quinlan, *C4.5: Programs for Machine Learning*, Morgan Kaufman, 1992, ISBN: 1558602380.

[12] J. Catlett, "Inductive learning from subsets or disposal of excess training data considered harmful.," in *Australian Workshop on Knowledge Acqusition for Knowledge-Based Systems, Pokolbin*, 1991, pp. 53–67.

[13] Tim Oates and David Jensen, "The effects of training set size on decision tree complexity," in *Proc. 14th International Conference on Machine Learning*, 1997, pp. 254–262, Morgan Kaufmann.

[14] Gerhard Widmer and Miroslav Kubat, "Learning in the presence of concept drift and hidden contexts," *Machine Learning*, vol. 23, no. 1, pp. 69–101, 1996, Availabel from `http://citeseer.ist.psu.edu/widmer96learning.html`.

[15] Y. Yang and G. Webb, "Weighted proportional k-interval discretization for naive-bayes classifiers," in *Proceedings of the 7th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD 2003)*, 2003, Available from `http://www.cs.uvm.edu/~yyang/wpkid.pdf`.

[16] Z. Z. Zheng and G. Webb, "Lazy learning of bayesian rules," *Machine Learning*, vol. 41, no. 1, pp. 53–84, 2000, Available from `http://www.csse.monash.edu/~webb/Files/ZhengWebb00.pdf`.

[17] M.A. Hall and G. Holmes, "Benchmarking attribute selection techniques for discrete class data mining," *IEEE Transactions On Knowledge And Data Engineering*, vol. 15, no. 6, pp. 1437– 1447, 2003.

[18] J. Gama, "Iterative bayes," *Intelligent Data Analysis*, pp. 475–488, 2000.

[19] K. Chai, H. Ng, and H. Chieu, "Bayesian online classifiers for text classification and filtering," in *Proceedings of SIGIR-02, 25th ACM International Conference on Research and Development in Information Retrieval*, M. Beaulieu, R. BaezaYates, S.H. Myaeng, and K. Jarvelin, Eds., 2002, pp. 97–104, Available from `citeseer.ist.psu.edu/chai02bayesian.html`.

[20] U M Fayyad and I H Irani, "Multi-interval discretization of continuous-valued attributes for classification learning," in *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, 1993, pp. 1022–1027.