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

Our goal is intelligent adapative controllers of running programs that are monitored by temporal logic invariants. We want to find bugs, or ways to dodge them. Hence, we want data miners to watch Monte Carlo simulations and learn constraints to input ranges which increase/decrease the number of logic violations. To this end, we explore how to simplify the task of scaling up induction to very large data sets. Given the right systems software such data sets are readily obtainable: just run the Monte Carlo simulations on the spare CPUs left behind when everyone go home for the night.

Standard classifiers are not designed to handle very large data sets: they usually assume that the data will be represented as a single, memory-resident table [1]. Nevertheless, for data sets that exhibit *early plateaus* we show here that the following combination tools can scale to very large data sets: (i) a NaïveBayes classifier; (ii) an incremental discretizer called SPADE; (iii) a simple windowing sampling policy called SAWTOOTH.

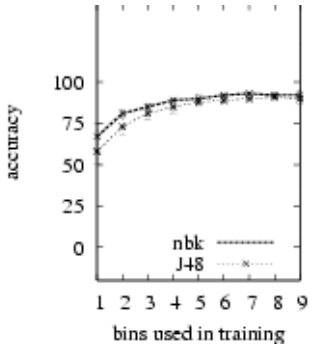

Fig. 1. Incremental 10*10 cross validation experiments on *soybean*. Error bars show ± 1 standard deviations for the accuracies over the repeats.

A data set exhibits *early plateaus* when the peak performance of a learner requires just a few hundred instances. Figure 1 shows such an early plateau in the *soybean* [2] data set. Figure 1 is a *incremental R*N-way cross-validation* experiment. For $R = 10$ repeats, data was randomly shuffled. The data in each random ordering was then divided *N=10* ways. Two learners, J48 and nbk[1], were then trained using the first i divisions and tested using remaining *N-i* divisions. As might be expected, as more data is used for training (i.e. running for *i=1...(N-1))*, the accuracy of the learned theory increased. For the *soybean* data set, this increase peaked 88%/91% for J48/nbk respectively after seeing 60% of the data (409 instances).

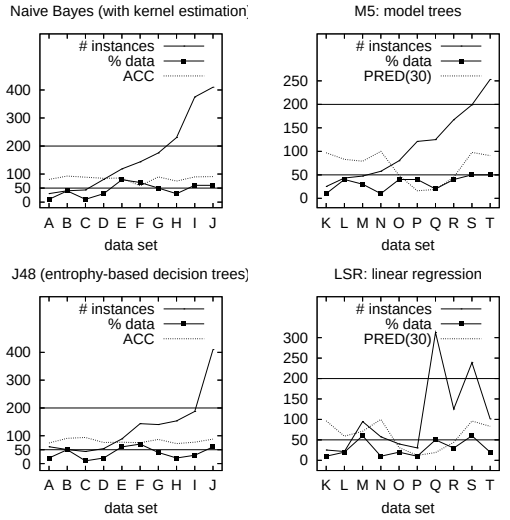

Fig. 2. *10*10* incremental cross validation experiments on 20 UCI data sets [2]: {*A:heart-c, B:zoo; C:vote; D:heart-statlog; E:lymph, F:autos. G:ionosphere, H:diabetes, I:balance-scale, J:soybean, K:bodyfat. L:cloud, M:fishcatch, N:sensory, O:pwLinear, Q:strike, R:pbc, S:autoMpg, T:housing*}. Data 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. Data sets are sorted according to how many instances were required. to reach plateau using nbk (left-hand side) or M5' (right-hand side).

Other data sets plateau even earlier than *soybean*. Figure 2 shows plateaus found by four learners (J48,nbk, LSR, and M5'[2]) in 20 data sets. Those plateaus were found before seeing most of the data: usually, after seeing 50%, and mostly before after seeing 60% (exceptions: E and F). For any learner in Figure 2, in $\frac{8}{10}$ths of the data sets, learning did not improve significantly (computed using t-tests with $\alpha = 0.05$) after seeing 200 instances.

Figure 2 suggests that, for many data sets and many learners, learning could proceed in *windows* of a few hundred instances. Learning could be disabled once performance peaks within those windows. If the learner's performance falls off the plateau (i.e. due to concept drift), the learner could start afresh to learn a new theory. Given early plateaus like those in Figure 2, this new learning should take only a few more hundred instances. Further, since learning only ever has to process a few hundred instances at a time, this approach should scale to very large data sets.

The rest of this tests that speculation using numerous UCI data sets, some KDD cup data, and an aircraft flight simulator. Our simple toolkit easily extends to handle concept drift problems such as recurring contexts and, in unsupervised learning, can be used to detect novel inputs. The toolkit performs comparatively well compared to other data miners. Our only explanation for the success of such a simple toolkit is that the assumption of early plateaus holds in many datasets.

II. Related Work

Provost and Kolluri [1] note that while the performance of some learners level off quite early, other learners continue to show accuracy increases as data set size increases. However, that improvement can be

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

[1]Both learners come from the WEKA toolkit [3]. J48 is release eight of C4.5 [4] ported to JAVA and nbk is a NaïveBayes classifier using John and Langley's kernel estimation method [5]

[2]LSR and M5' come from the WEKA [3]. LSR/M5' assumes values can be fitted to one/many (respectively) n-dimensional linear models [6].

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 [7].

Plateaus like Figure 2 have been reported before (although this may be first report of early plateaus in M5' and LSR). Oates and Jensen found plateaus in 19 UCI data sets using five variants of C4.5 [8]. In their results, six of their runs plateaued after seeing 85 to 100% of the data. This is much later than Figure 2 where none of our data sets needed more than 70% of the data.

We are not motivated to explore different methods for detecting start-of-plateau. The results below show that learning using our start-of-plateau detector can produce adequate classifiers that scale to very large data sets. Nevertheless, one possible reason for our earlier plateaus is the method used to identify start-of-plateau. Figure 2 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. Note that 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.

Assuming early plateaus, then very simple learners should scale to large data sets. Our reading of the literature is that Bayes+SAWTOOTH+SPADE is much simpler than other methods for scaling up data mining. Provost and Kolluri distinguish three general types of scale-up methods. Firstly, there are *relational representation* methods that reject the assumption that we should learn from a single memory-resident table. Secondly, there are *faster algorithms* that (e.g.) exploits *parallelism*. Thirdly there are *data partitioning* methods to learn from (e.g.) some *subset of the attributes*. Two such strategies are *windowing* and *stratified sampling*. In *windowing*, 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. In *straified sampling*, instances are sampled according to the relative frequency of their classes; e.g. instances from the minority classes are selected with a higher frequency that instances from the majority classes. SAWTOOTH uses an *explicit* windowing policy and, as discussed later, uses *implicit* stratified sampler.

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 [9] 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.

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

```
# 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. 3. 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 [10, §3.1].

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.

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 3 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.

NaïveBayes classifiers are based on Bayes' Theorem. Informally, the theorem says $new = old * evidence$; i.e. our *new* beliefs are a function of our *old* and any *new*. 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 3 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 [11], 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. [12], [13]) have reported that, in many domains, this simple Bayes classification scheme exhibits excellent performance compared to other learners.

In terms of scaling up induction, the most important property of a Bayes classifier is that they need very little memory: i.e. only what required for the frequency counts plus a buffer just large enough to hold a single instance.

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 [14] while Chai et.al. updates the priors via feedback from the examples seen up till now [15]. 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 [5]. Other, more sophisticated methods are well-established [16], but several studies report that even simple *discretization methods* suffice for adapting Bayes classifiers to numeric variables [13], [17].

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 [13] *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. [13] and Yang and Webb [17]) 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.

III. 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 3. Also, when SPADE merges bins, this causes a similar merging in frequncy tables entries (the F variable of Figure 3).

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 4 compares results from SPADE and John and Langley's kernel estimation method using the display format proposed by Dougherty, Kohavi and Sahami [13]. 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 3 NaïveBayes classifier using data pre-discretized by SPADE. Mean classification accuracies were collected and shown in Figure 4, sorted by the means $(c-a)-(b-a)$;

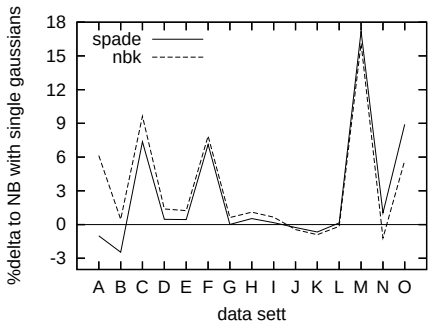

Fig. 4. Comparing SPADE and kernel estimation. Data sets: {*A=vowel, B=iris, C=ionosphere, D=echo, E=horse-colic, F=anneal, G=hypothyroid, H=hepatitis, I=heart-c, J=diabetes, K=auto-mpg, L=waveform-5000, M=vehicle, N=labor, O=segment*}.

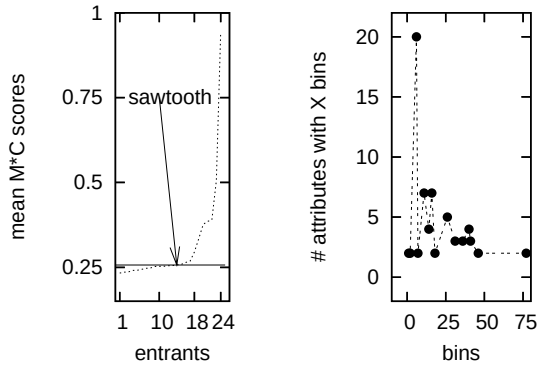

Fig. 5. SAWTOOTH and the KDD'99 data

that is, by the difference in the improvement seen in SPADE *or* kernel estimation *over or above* a simple single gaussian scheme. Hence, the left-hand-side data sets of Figure 4 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 4 are noteworthy. Firstly, in a finding consistent with those of Dougherty et.al. [13], 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.

IV. 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[3]. 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 5 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 5. 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[4].

B. UCI Data

Figure 5 explored SAWTOOTH's competencies on one large data set. Figure 6 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+Bayes (using the Figure 3 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 [5]).

Using t-tests, significant differences ($\alpha = 0.05$) between the mean performance of each learner on the 20 data sets could be

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

[4]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

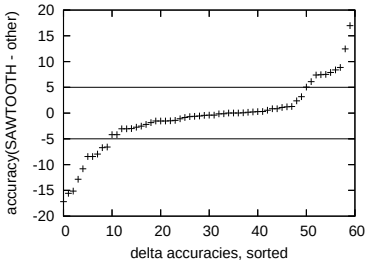

Fig. 6. SAWTOOTH exeucting on UCI data.

detected. Win/loss/ties statistics for each pair of learners on each data set was then collected. The results, shown top of Figure 6 shows SAWTOOTH performing marginally better than NB classifier but worse that both J48 and NBK. This is not surprising: Provost and Kolluri [1, 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 6 sorts all those differences. In 80% of our experiments, SAWTOOTH performed within $\pm 5\%$ of other methods.

C. Data with Concept Drift

Figure 5 and Figure 6 showed SAWTOOTH processing static data. Figure 7 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 7 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.

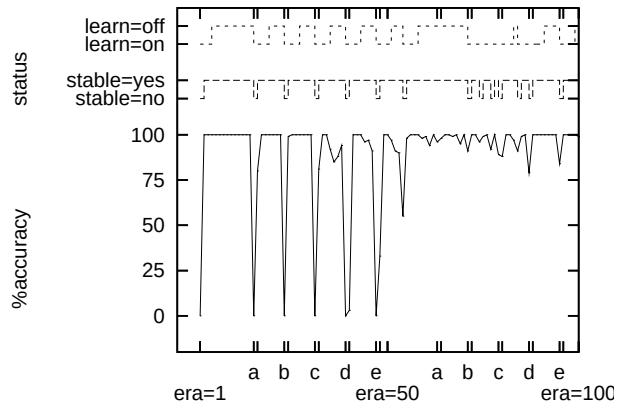

Fig. 7. SAWTOOTH and concept drift

Three features of Figure 7 are worthy of mention. Firstly, between concept drifts, the accuracy stabilizes and SAWTOOTH mostly disables the learner, thus implementing the *implicit stratified sampling* policy mentioned above in §II. Consider a run containing eras 1,2,...,50 where all eras come from class1 except eras 10,11 which contain class2. If plateau takes two eras, then, at most, SAWTOOTH would only use two eras of class2 data (since that is all there is) and four eras of class1 data (the first two eras after the start and two more eras after the class2 eras). This means that SAWTOOTH would under-sample the majority class and ignore 88% of the class1 data.

Secondly, 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*. Thirdly, 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 5, Figure 6, and Figure 7 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 8 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. Such novelty detectors 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.

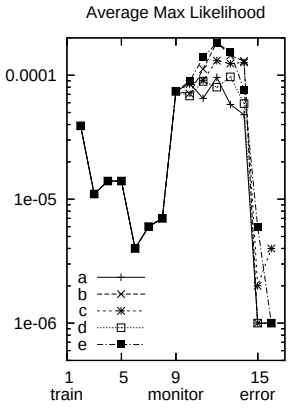

Fig. 8. Learning normal flight (eras 1 to 8); monitoring five different flights a,b,..e (eras 9 to 16); injecting errors into eras 15,16.

In the Figure 8 experiment, the class of all instances were replaced with a single label: $class0$. 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 3 was modified to return the classification with the maximum likelihood *as well as* that maximum likelihood value. Figure 8 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).

Figure 8 suggets that SAWTOOTH could be used for novelty detection in unsupervised learning as follows. 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.

Elsewhere, we have explored novelty detection in unsupervised learning using a variety of complex methods: association rules to learn expected patterns in attribute values [18]; or SVDDs to recognize boundaries between expected and novel inputs [19]. Figure 8 suggests that SAWTOOTH could perform the same task, but in a much simpler way.

V. Discussion

Holte argue for *simplicity-first* approach to data mining; i.e. researchers should try simpler methods before complicating existing algorithms [20]. While Provost and Kolluri endorse "simplicity-first", they note in their review of methods for scaling up inductive algorithms 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" [1, p32].

This paper has been a simplicity-first approach to scaling up data miners. We have *levered* two features of NaïveBayes classifiers that make them good candidates for handling large datasets: fast updates of the current theory and small memory foot print. Several *deficiencies* with NaïveBayes classifiers have been addressed: incremental discretization and dynamic windowing means that Bayes classifiers need not hold all the data in RAM at one time.

Our Bayes+SAWTOOTH+SPADE toolkit works via one scan of the data and can scale to millions of instances and works as many other data mining schemes (see Figure 5). The same toolkit 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 (see Figure 7); and can detect novel inputs during unsupervised learning (see Figure 8).

A 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 5).

Our toolkit much simpler than other data miners such as FLORA, the winner of KDD'99, or the SVDDs we used previously for detecting novel inputs. Why can such a simple toolkit like Bayes+SAWTOOTH+SPADE be so competent? Our answer is that many data sets (such as all those processed in our experiments) exhibit early plateaus and such early plateaus can be exploited to build very simple learners. 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. Further, 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 6).

Finally, we recommend that other data mining researchers check for early plateaus in their data sets. If such plateaus are a widespread phenmema, then *very simple tools* (like Bayes+SAWTOOTH+SPADE) should be adequate for the purposes of scaling up induction.

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] 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`.

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

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

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

[5] 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`.

[6] 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`.

[7] 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.

[8] 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.

[9] 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`.

[10] 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`.

[11] 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`.

[12] 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.

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

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

[15] 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`.

[16] 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.

[17] 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.

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

[19] 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`.

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