Incremental Discretization With Bayes Classifiers Scales Very Well for Data With "Early Plateaus"

Tim Menzies, *Member,IEEE,* Andres Orrego

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

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. Further, this approach supports forgetting old theories after concept drift and learning faster when concepts drift back to a context seen previously.

Index Terms

data mining, concept drift, scale up, NaïveBayes classifiers, incremental, discretization, SAWTOOTH, SPADE

I. INTRODUCTION

Our goal is intelligent adapative runtime monitoring of procedural programs augmented with temporal logic invariants. We want data miners to monitor Monte Carlo simulations and learn constraints to input ranges which increase the number of logic violations. To this end, we explore how to simplify the task of scaling up induction to very large data sets. Such data sets are readily obtainable in our domain: just run the Monte Carlo simulations on the spare CPUs left behind when our co-workers 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.

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).

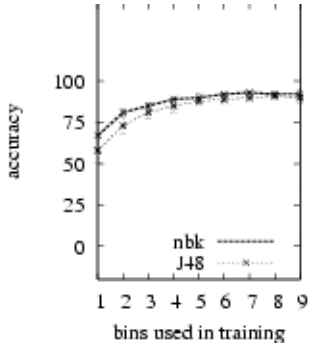


Fig. 1. Incremental 10*10 cross validation experiments on soybean. Error bars show ± 1 standard deviations for the accuracies over the repeats.

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

[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].

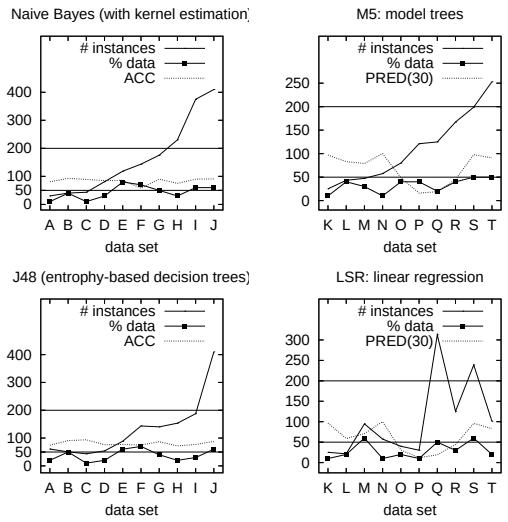


Fig. 2. *R=10*N=10* incremental cross validation experiments on 20 UCI data sets [2]. A:heart-c, B:zoo; C:vote; D:heart-statlog; E:lymph, F:autos. G:ionosphere, H:diabetes, I:balance-scale, J:soybean, K:bodyfat. L:cloud, M:fishcatch, N:sensory, O:pwLinear, Q:strike, R:pbc, S:autoMpg, T:housing. Data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

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.

This paper tests that speculation. For numerous UCI data sets, some KDD cup data, and in a aircraft flight simulator, this combination of Bayes+SAWTOOTH+SPADE performs comparatively well, uses less memory, and is simple to implement. Our conclusion is twofold. Firstly, early plateaus are common and hence very simple methods can scale to numerous very large data sets. Secondly, at least some of the recent advances in scaling up data mining are not be due to the sophistication of the data miner. Rather, they may be due to the simplicity of the data in the test cases.

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 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*. SAWTOOTH is a *windowing* data partitioning scheme where 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 is used in many systems including FLORA [9] and SAWTOOTH. If the window size W is small relative to the rate of concept drift, then windowing guarantees the maintenance of a a theory relevant to the last W examples. However, if W is too small, the learning may never have find an adequate characterization of the target concept. Similarly, if W is too large, then the this will slow the learner's reaction to concept drift.

Like many windowing systems, SAWTOOTH and FLORA 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 simple t-test to determine window size and, in all our experiments, have kept parameters of those tests constant (at $\alpha = 0.01$). 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

```
# 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 [10, §3.1]. This code computes class *likelihoods* not probabilities. Likelihoods become probabilities when they are normalized over the sum of all likelihoods. However, since maximum probability comes from maximum likelihood, this code only needs to return the class with maximum likelihood.

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. In each era k, there are S_k successful classifications. Equation 1 checks if the current era j is different to the proceedings eras $1..., i$ (at confidence $\alpha = 0.01$).

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

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 the computational cost of continually re-learning. Hence SAW-

TOOTH 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.

Apart from rapid updates, NaïveBayes classifiers have other advantages. Such classifiers use Bayes' Theorem:

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

That is, given fragments of evidence E_i and a prior probability for a class $P(H)$, a posterior probability $P(H \mid E)$ is calculated for the hypothesis given the evidence. The Bayes classifier returns the class with highest probability. Such 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.

More importantly, Bayes classifiers need very little memory and hence can scale to very large problems. These learners only need the memory 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].

Many kernel estimation and discretization methods violate the *one scan* requirement of a data miner; i.e. learning needs 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. Similarly, John and Langley's kernel estimation method can't build its distribution until *after* seeing all the data and storing every continuous attribute it sees during training. An incremental one scan (or less) discretization method is needed for scaling up induction. SAWTOOTH uses the SPADE method described below.

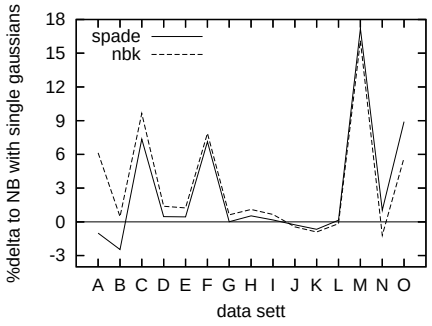


Fig. 4. Comparing SPADE and kernel estimation. Data sets: A=vowel, B=iris, C=ionosphere, D=echo, E=horse-colic, F=anneal, G=hypothyroid, H=hepatitis, I=heart-c, J=diabetes, K=auto-mpg, L=waveform-5000, M=vehicle, N=labor, O=segment.

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).

Unlike N, SPADE does not assume a normal distribution among attribute values, in fact, it does not assume any distribution at all. It 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*).

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), 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. A comparison of the algorithmic complexity of Nwith the single gaussian assumption, John and Langley's kernel estimation, and SPADE is shown in Figure **??**.

It is important to note that SPADE will adjust bins only after the current ERA has been processed, therefore allowing plenty of room for the generation of multiple bins before they are merged. Preventing the creation of very few bins with big tallies is essential to avoid a bin splitting policy which is impractical within an incremental context.

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 $(c-a)-(b-a)$; 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

	Gaussian Assumption		Kernel Estimation		SPADE	
Operation	Time	Space	Time	Space	Time	Space
Train on n instances	$O(nk)$	$O(k)$	$O(nk)$	$O(nk)$	$O(nk)$	$O(kb) = O(k)$
Test on m instances	$O(mk)$		$O(mnk)$		$O(mkb) = O(mk)$	

Fig. 5. Algorithmic complexity of three different numeric handling techniques for the NaïveBayes classifier, given n training instances and k attributes. Also, b is the number of bins generated by SPADE, which becomes a constant when the minimum and maximum values of each attribute are encountered. SPADE then has, effectively, the same unbeatable low algorithmic complexity of the single gaussian assumption

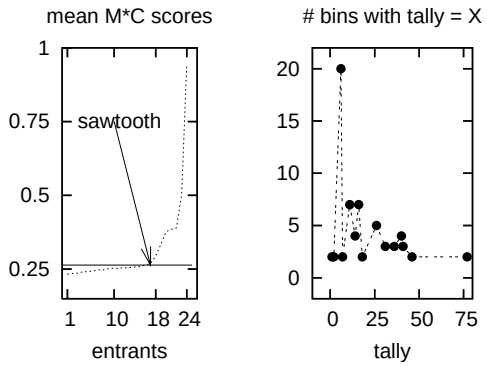
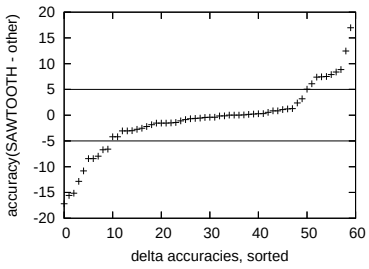


Fig. 6. SAWTOOTH and the KDD'99 data

		incremental	batch		
data set	#instances	SAWTOOTH	NB	nbk	j48
labor	57	91.13 ± 11.76	93.87 ± 11.26	92.67 ± 11.46	78.67 ± 17.65 -
iris	150	93.13 ± 6.03	95.67 ± 5.14 +	96.13 ± 4.85 +	94.53 ± 5.87
hepatitis	155	85.67 ± 9.15	83.28 ± 10.04 -	84.38 ± 9.92	78.28 ± 9.49 -
echo	130	80.08 ± 10.09	78.85 ± 11.01	80.23 ± 9.98	80.77 ± 11.34
audiology	226	61.42 ± 10.45	69.84 ± 10.35 +	69.84 ± 10.35 +	76.55 ± 9.10 +
heart-c	303	83.74 ± 6.46	83.50 ± 6.97	84.16 ± 6.24	76.23 ± 7.71 -
primary-tumor	339	47.83 ± 7.34	49.37 ± 7.09	49.37 ± 7.09	41.73 ± 7.63 -
ionosphere	351	89.54 ± 5.09	82.09 ± 7.70 -	91.73 ± 4.31 +	89.66 ± 5.28
horse-colic	368	68.41 ± 7.55	67.58 ± 7.64	68.81 ± 7.44	67.31 ± 8.18
auto-mpg	398	74.24 ± 7.48	74.85 ± 8.37	73.95 ± 7.83	75.73 ± 8.17
vote	435	89.98 ± 4.32	89.96 ± 4.35	89.96 ± 4.35	96.58 ± 2.97 +
soybean	683	89.38 ± 4.76	92.42 ± 4.21 +	92.42 ± 4.21 +	90.49 ± 6.04
diabetes	798	74.70 ± 4.81	75.54 ± 4.36	75.10 ± 4.64	74.15 ± 5.41
vehicle	846	61.62 ± 4.73	44.64 ± 4.95 -	60.77 ± 4.28	72.42 ± 5.52 +
anneal	898	94.50 ± 2.97	86.61 ± 3.59 -	94.49 ± 2.47	98.69 ± 1.02 +
vowel	990	59.72 ± 6.51	66.42 ± 5.31 +	72.56 ± 6.33 +	76.91 ± 9.14 +
segment	2310	88.91 ± 2.24	80.04 ± 2.34 -	85.73 ± 2.09 -	96.87 ± 1.19 +
hypothyroid	3772	95.36 ± 1.30	95.30 ± 1.06	95.93 ± 0.98	99.53 ± 0.32 +
waveform-500	5000	80.16 ± 1.77	80.00 ± 1.93	79.84 ± 1.95	75.12 ± 2.03 -
letter	20000	72.44 ± 1.26	64.05 ± 1.11 -	74.28 ± 1.08 +	88.01 ± 0.83 +
MEAN		79.10	77.69	80.62	81.41



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. 7. $mean \pm standard\ deviations$ seen in 10*10-way cross validation experiments on UCI Irvine data sets. “NB” and “nbk” denote NaïveBayes classifiers that use gaussians model continuous attributes. “NB” uses a single gaussian while “nbk” uses a sum of gaussians in the method recommended by John and Langley [5]. The plot top-right sorts the differences in the accuracies found by SAWTOOTH and all the other learners. Some of those differences aren’t statistically significant: the “+” or “-” in the left-hand-side table denote mean differences that are significantly different to SAWTOOTH at the $\alpha = 0.05$ level. The significant differences between all the learners are shown in the win-loss statistics of the bottom-right table.

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]. The KDD data dealt with network intrusion detection and was divided into a training set of about five million instances and a *test*

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

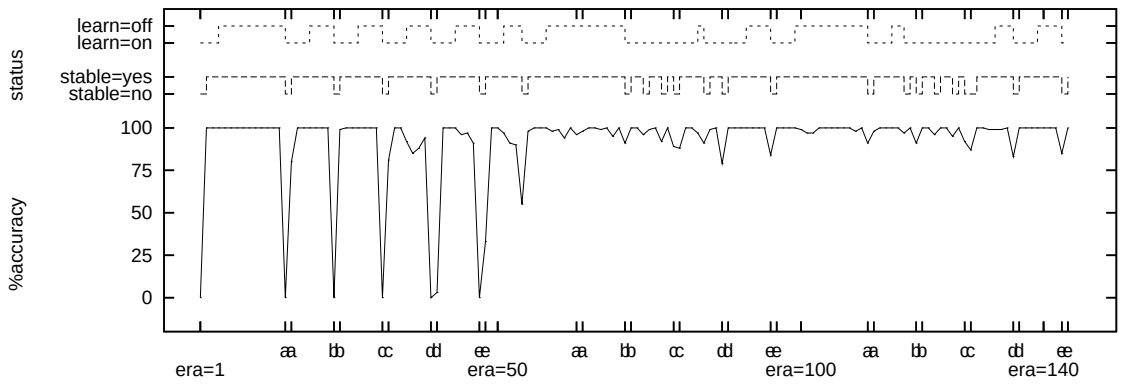


Fig. 8. SAWTOOTH and Concept Drift

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 the *mean M*C scores* for SAWTOOTH and the KDD'99 entrants. SAWTOOTH's mean $M * C$ results were close to the winning score of entrant #1; very similar to entrants 10,11,12,13,14,15,16; and better than entrants 18,19,20,21,22,23,24. These results are encouraging since SAWTOOTH is a much simpler tool that the winning entry (which classified the security incidents using an ensemble of decision trees built from a 50*10 cross-val).

Another encouraging result is the *# bins with tally=X* 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, in the worst case, generate thousands of bins for each numeric attribute. 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. Lastly, SPADE never generated more than 100 bins.

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. The large table on the left shows the mean and standard deviations of the accuracy seen in 10*10 cross-validation experiments. The win-loss table (bottom-right of Figure 6) use t-tests to compare the performance of our learners. SAWTOOTH performs marginally better than a simple NaïveBayes

classifier but is out-performed by both J48 and nbk. This is not surprising: Provost and Kolluri [1, p22] comment that sequential learning strategies like windowing usually performs worse that 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 top-right 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 three times, the simulator returned to each error mode. 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.

Three features of Figure 7 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 significant result: learning systems can alert their uses when they are *leaving the region of their past competency*. Secondly, 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*. Thirdly, between concept drifts, the accuracy stabilizes and SAWTOOTH mostly disables the learner. That is, for much of Figure 7 the SAWTOOTH "learner" is *doing no learning at all*.

V. Conclusion

Holte argue for *simplicity-first* approach to data mining; i.e. researchers should try simpler methods before complicating existing algorithms [21]. 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. Our toolkit is much simpler than other scale-up methods such as FLORA or the winner of KDD'99. Even so it performs as well as many other data mining schemes (see Figure 5). Further, the same toolkit without any

modifications can be used to detect concept drift, to repair a theory after concept drift, and can reuse old knowledge when old contexts re-occur (see Figure 7).

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).

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] T. Menzies and C. Pecheur, "Verification and validation and artificial intelligence," in *Advances in Computing*, M. Zelbowtiz, Ed. 2005 (to appear), Elsevier.

[19] Paul S. Bradley, Usama M. Fayyad, and Cory Reina, "Scaling clustering algorithms to large databases," in *Knowledge Discovery and Data Mining*, 1998, pp. 9–15, Available from `http://citeseer.ist.psu.edu/bradley98scaling.html`.

[20] G. Hulten and P. Domingos, "A general framework for mining massive data streams," *Journal of Computational and Graphical Statistics*, vol. 12, 2003, Available from `http://www.cs.washington.edu/homes/ghulten/papers/genframe-jcgs.pdf`.

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