Mach Learn (2009) 77: 61–102
DOI 10.1007/s10994-009-5118-6

Anytime classification for a pool of instances

Bei Hui · Ying Yang · Geoffrey I. Webb

Received: 2 April 2008 / Revised: 27 April 2009 / Accepted: 28 April 2009 / Published online: 2 July 2009

Abstract In many real-world applications of classification learning, such as credit card transaction vetting or classification embedded in sensor nodes, multiple instances simultaneously require classification under computational resource constraints such as limited time or limited battery capacity. In such a situation, available computational resources should be allocated across the instances in order to optimize the overall classification efficacy and efficiency. We propose a novel anytime classification framework, Scheduling Anytime Averaged Probabilistic Estimators (SAAPE), which is capable of classifying a pool of instances, delivering accurate results whenever interrupted and optimizing the collective classification performance. Following the practice of our previous anytime classification system AAPE, SAAPE runs a sequence of very efficient Bayesian probabilistic classifiers to classify each single instance. Furthermore, SAAPE implements seven alternative scheduling schemes to decide which instance gets available computational resources next such that a new classifier can be applied to refine its classification. We formally present each scheduling scheme's definition, rationale and time complexity. We conduct large-scale experiments using 60 benchmark data sets and diversified statistical tests to evaluate SAAPE's performance on zero-one loss classification as well as on probability estimation. We analyze each scheduling scheme's advantages and disadvantages according to both theoretical understandings and empirical observations. Consequently we identify effective scheduling schemes that enable SAAPE to accomplish accurate anytime classification for a pool of instances.

Editor: David Page.

B. Hui (✉)
School of Computer Science and Engineering, University of Electronic Science and Technology of China, ChengDu, China
e-mail: bhui@uestc.edu.cn

Y. Yang
Australian Taxation Office, Melbourne, Australia
e-mail: ying.yang@ato.gov.au

G.I. Webb
Clayton School of Information Technology, Monash University, Clayton, Australia
e-mail: geoff.webb@infotech.monash.edu.au

Keywords Anytime classification · Computational resource constraints · Bayesian probabilistic classifiers · Ensemble learning

1 Introduction

The goal of our research is to develop classification learning systems that are able to handle varying computational resources and to optimize classification results within the constraints imposed by those resources. This contrasts with conventional classification techniques that are designed to operate within the constraints of the minimum expected resources and fail to utilize further resources when they are available, as a result of which less accurate classification may occur than might be achievable within the available resources if a less efficient but more accurate learner were employed.

Among four distinct computational resources, *training time*, *training space*, *classification time* and *classification space* (Yang et al. 2007), our current research addresses one specific dimension, *classification time*.[1] We believe this dimension has wide practical application and is typical of many online applications where naïve Bayes (NB) (Langley et al. 1992) is deployed. For instance, interactive systems must complete their task within strict elapsed time limits in order to gain and retain user acceptance. Some of the many examples of such applications include information retrieval (Baeza-Yates and Ribeiro-Neto 1999), recommender systems (Resnick and Varian 1997), user modeling (Webb et al. 2001) and online fraud detection (Chan et al. 1999).

In this context, constraints on training space and time are not a major issue, because classifiers are developed infrequently off-line and then employed many times. Further, constraints on classification space do not apply to classifiers, because a single classifier can be shared among all jobs and hence the constraints relating to the classification space for a particular job relate only to the job's additional space requirements. As a result, our focus is on *anytime classification*. We assume that classifiers have been learned beforehand, and the classification time budget is sufficient to apply at least one of the classifiers to each instance. The idea is to run an ordered sequence of (very efficient) classifiers until classification time runs out.

In a previous paper (Yang et al. 2007) we have proposed Anytime Averaged Probabilistic Estimators (AAPE), an anytime classification system that uses an ensemble of Bayesian probabilistic estimators and that is able to deliver accurate classification for an instance whenever the classification is interrupted. In particular, to classify an instance, AAPE first computes NB and then utilizes any additional time to refine the probability estimates by invoking superparent-one-dependence estimators (SPODEs) (Keogh and Pazzani 2002) in turn. The classification can stop anywhere in the ordered sequence of SPODEs. The probabilistic classifiers invoked up to the interruption time compose an ensemble which will carry out the classification.

The existing AAPE framework, and other existing anytime classification systems such as anytime support vector machines (DeCoste 2002), anytime nearest neighbors (Ueno et al. 2006) and anytime Bayesian belief networks (Liu and Wellman 1996; Jitnah and Nicholson 1997) have focused on the scenario where only one instance or query needs to be evaluated at one time. However it often happens that multiple instances simultaneously need to be processed. In such a situation, the available computational resources must be allocated

[1] In our context, the classification time also reflects constraints such as the sensor battery capacity.

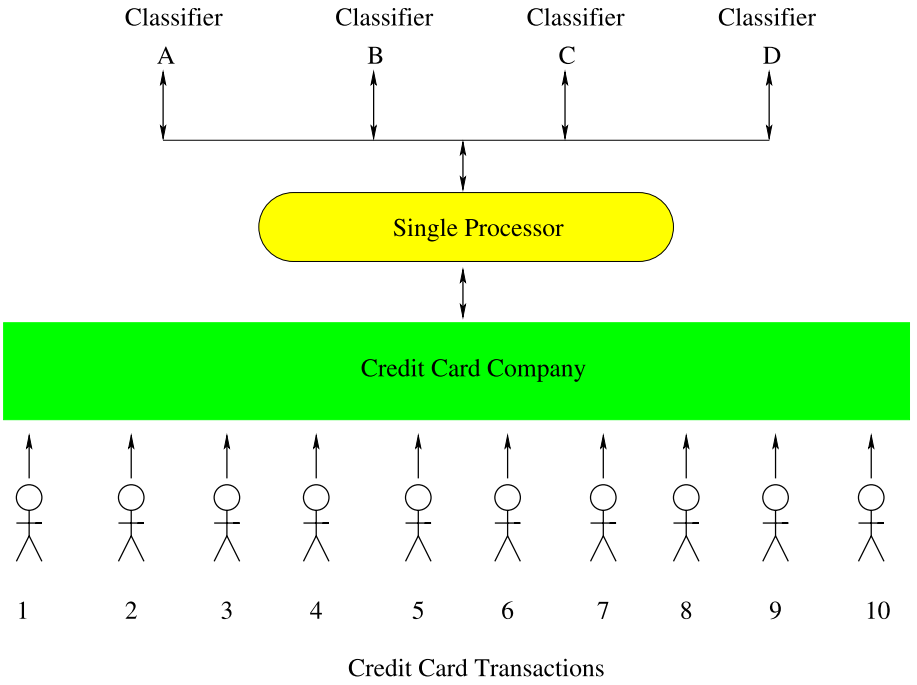


Fig. 1 When a credit card transaction takes place, the credit card company passes it to classifiers to decide whether to approve or to reject it. There can be multiple transactions happening at the same time. (Note that we restrict this example to a single processor for the sake of simplicity. The problem and our solutions generalize to multi-processor contexts.)

across the instances in order to optimize the overall classification efficacy and efficiency. Ye, Wang, Yankov and Keogh have previously studied anytime classification for multiple instances in the context of nearest neighbor classification learning (2008). The current paper will examine this problem in the context of Bayesian probabilistic classification learning, where a choice has to be made about which instance should get available computational resources next such that a new SPODE can be applied to refine the instance's class probability estimates. We use the application of credit card transaction approval to demonstrate such a scenario. Each transaction is an instance to be classified. Note that the values used here are toy examples devised for illustrative purposes. As shown in Fig. 1, when a credit card transaction takes place, the credit card company passes it to classifiers to decide whether to approve or to reject it. Assume that ten transactions happen at approximately the same time, and that a service level agreement requires that a decision takes no more than 30 seconds to deliver. Also assume that the company has four classifiers, and has a single processor that can only apply a single classifier to a single transaction at one time.[2] Further assume that it takes 2 seconds for each classifier to examine each transaction. The more classifiers that are applied to a transaction, the higher confidence the company has in its decision. In this scenario, AAPE will consult each of the four classifiers in turn to examine Transaction 1, which takes 8 seconds. Then AAPE will consult each of the four classifiers in turn for

[2]The problem and our solutions generalize to multi-processor contexts. We restrict this example to a single processor for the sake of simplicity.

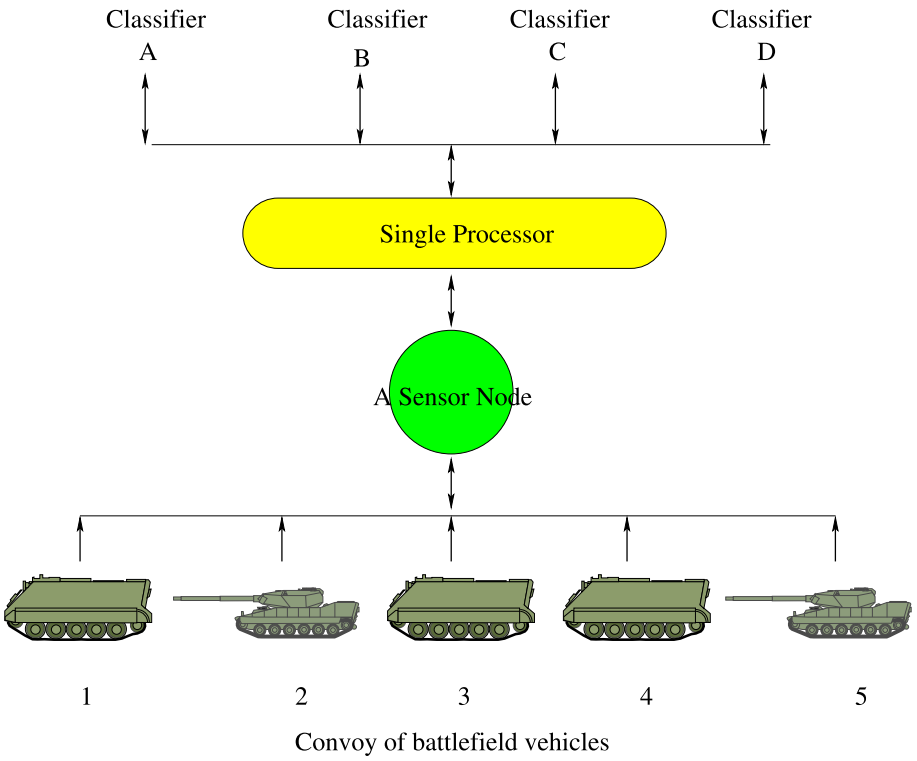


Fig. 2 When a convoy of vehicle pass a sensor node, the sensor node needs to classify each vehicle type within its available energy. The vehicle artwork (©2001, Carlo Kopp) is distributed as part of the xfig package

Transaction 2, which takes another 8 seconds. This procedure continues until some intervention occurs. AAPE is optimal from the individual point of view of a single application. If AAPE finishes processing an application, it means that this application has been allocated as many resources as possible so that the decision about it is most appropriate. However, AAPE is sub-optimal from the collective point of view regarding all the ten transactions. In this case, 30 seconds will have passed before the company finishes processing Transaction 4. Transactions 5 to 10 will not have been processed within an acceptable time frame.

A similar problem exists in sensor networks. For example, as illustrated in Fig. 2 one of the important sensor network applications is to identify vehicle types in a moving convoy in a battlefield (Meesookho et al. 2002; Sun and Daigle 2006; Duarte and Hu 2004). A sensor node collects acoustic and seismic signals of each vehicle moving past, and carries out the classification. Because a sensor node has very limited battery capacity which varies with time and usage, the challenge is to maximize the classification accuracy across the convoy of vehicles, each time according to the available energy at that time.

To solve these problems, we propose a new anytime classification framework, which is capable of classifying a pool of instances, delivering accurate results whenever interrupted and optimizing the overall classification accuracy regarding the whole pool. This problem is analogous to the computer scheduling problem where multiple processes compete for the CPU at the same time (Tanenbaum 2001; Deitel et al. 2004; Stallings 2004). Hence we

name this new system Scheduling Anytime Averaged Probabilistic Estimators (SAAPE) and sometimes employ scheduling terminology.

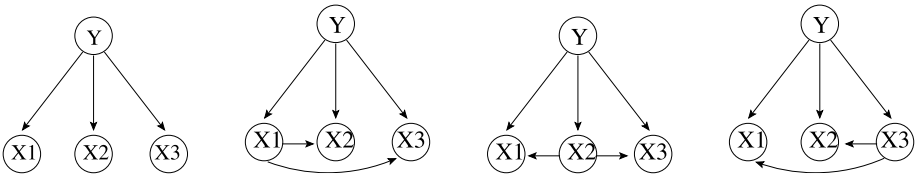
The remainder of this paper is organized as follows. Section 2 recalls the previous AAPE system which is the base of the new SAAPE system. Section 3 proposes the new SAAPE system and presents a wide variety of seven alternative scheduling algorithms therein. The pseudo codes of the SAAPE system and the scheduling algorithms are presented in Appendix B to avoid distraction from the main text. Section 4 analyzes the classification time complexity of SAAPE and each scheduling algorithm. Section 5 conducts large-scale experiments using 60 benchmark data sets to test the anytime classification performance of SAAPE. Diversified statistical tests are employed to thoroughly evaluate and rank alternative scheduling schemes. Section 6 gives concluding remarks and discusses future work.

2 The AAPE system

Our new SAAPE system for anytime classification of a pool of instances arises from our previous AAPE system that delivers anytime classification for individual instances. Hence we first briefly describe the AAPE system for a better understanding of the SAAPE system.

To classify an instance $\mathbf{x}$, AAPE deploys the Bayesian probabilistic classification methodology. It utilizes Bayes formula to calculate $\hat{P}(y \mid \mathbf{x})$, an estimate of $P(y \mid \mathbf{x})$ that is the probability of each class y given the instance to be classified, $\mathbf{x}$. It assigns the class with the highest estimated probability to $\mathbf{x}$. To achieve anytime classification for $\mathbf{x}$, AAPE employs both naive Bayes (NB) and relaxations thereof called superparent-one-dependence estimators (SPODEs). It starts with invoking NB to compute $\hat{P}(y \mid \mathbf{x})$. It then proceeds to invoke each candidate SPODE in turn to compute $\hat{P}(y \mid \mathbf{x})$ until interruption intervenes or classification time runs out. At this point, AAPE averages the class probabilities given $\mathbf{x}$ estimated by all the invoked classifiers so far and delivers them to the user. Figure 3 illustrates NB and all possible SPODEs for a data set with three attributes and one class variable.

NB and SPODEs are deployed in the AAPE system because they are efficient Bayesian classifiers and hence can minimize the risk that a single computation step is interrupted (Grass and Zilberstein 1996; Yang et al. 2007). NB assumes attributes conditionally



(a) NB (b) SPODE whose superparent is X_1 (c) SPODE whose superparent is X_2 (d) SPODE whose superparent is X_3

Fig. 3 Illustration of NB and all possible three SPODEs for a data set with three attributes and one class variable. An arc points from a parent to a child. A child only depends on its parents. NB assumes each attribute only depends on the class Y and is independent of other attributes given Y. A SPODE assumes that each attribute can depend on a *common* attribute (the superparent) in addition to the class. For a data set with m attributes, there are m possible SPODEs, each taking a different attribute as its superparent

independent of each other given the class. Thus it delivers class probability estimates using Eq. 1 where m is the number of attributes:

$$\hat{P}(y \mid \mathbf{x}) = \frac{\hat{P}(y)\hat{P}(\mathbf{x} \mid y)}{\hat{P}(\mathbf{x})} = \frac{\hat{P}(y)\prod_{i=1}^{m}\hat{P}(x_i \mid y)}{\hat{P}(\mathbf{x})}. \quad (1)$$

A SPODE relaxes NB's attribute independence assumption by allowing all attributes to depend on a common attribute, the superparent, in addition to the class. For a data set with m attributes, there are m possible SPODEs, each taking a different attribute as the superparent. A SPODE whose superparent is the attribute X_p delivers class probability estimates using Eq. 2:

$$\hat{P}(y \mid \mathbf{x}) = \frac{\hat{P}(x_p, y)P(\mathbf{x} \mid x_p, y)}{\hat{P}(\mathbf{x})} = \frac{\hat{P}(x_p, y)\prod_{i=1}^{m}\hat{P}(x_i \mid x_p, y)}{\hat{P}(\mathbf{x})}. \quad (2)$$

When the classification time budget has been exhausted, AAPE terminates and returns the probability estimates $\hat{P}(y \mid \mathbf{x})$ averaged on all classifiers invoked so far. This follows the practice of the AODE ensemble algorithm (Webb et al. 2005) and is computed as follows. Assume that S denotes the subset of SPODEs that have been invoked before an interrupt occurs. Assume $|S|$ denotes the number of SPODEs in S. Averaging the probability estimates of NB and SPODEs, we get:

$$\hat{P}(y \mid \mathbf{x}) = \frac{\hat{P}(y)\hat{P}(\mathbf{x} \mid y) + \sum_{p \in S}\hat{P}(x_p, y)\hat{P}(\mathbf{x} \mid x_p, y)}{\hat{P}(\mathbf{x}) \times (|S| + 1)} = \frac{\hat{P}(y)\prod_{i=1}^{m}\hat{P}(x_i|y) + \sum_{p \in S}\hat{P}(x_p, y)\prod_{i=1}^{m}\hat{P}(x_i|x_p, y)}{\hat{P}(\mathbf{x}) \times (|S| + 1)} \quad (3)$$

where the first addend in the numerator of Eq. 3 is NB's estimate and the second addend in the numerator of Eq. 3 sums SPODEs' estimates, each SPODE with the attribute x_p being the superparent. For the details of the induction of these equations, please refer to our previous paper on AAPE (Yang et al. 2007).

Table 1 presents the AAPE algorithm. Lines 1 to 7 compute NB. It is assumed that this computation can be completed within the contract time budget and hence interrupts are only enabled on Line 8. Line 10 starts the loop over the ordered set of qualified attributes S that are each the superparent for a SPODE to be invoked. Lines 12 to 17 compute the probability estimates TP for each SPODE. Lines 19 to 22 update the average conditional probability estimate for each class to account for a new set of TP values. Interrupts are suspended during this process as the P values are in an unstable state. In consequence, if the time budget expires during this process there will be a slight delay before computation terminates.

Table 1 The AAPE algorithm

Algorithm: AAPE
Inputs:

- **x**: the instance to be classified.
- S: a set of ordered SPODEs.
- $PP[c_1 \dots c_k]$: prior probability estimate for each class. This and the following estimates are based on the observed frequency in the training data with possible correction for sampling error such as a Laplace correction.
- $PP[c_1 \dots c_k, attvals]$: prior probability estimates for each class and attribute value pair.
- $CP[c_1 \dots c_k, attvals]$: conditional probability estimates for each attribute value given each class.
- $CP[c_1 \dots c_k, superparentvals, attvals]$: conditional probability estimates for each attribute value (the final index) given each class and each superparent value.

Outputs:

- $P[c_1 \dots c_k]$: the estimated probability of each class given **x**.

```
1: for y := c_1 ... c_k do
2:    P[y] := PP[y];
3:    for i := 1 ... m do
4:       P[y] := P[y] × CP[y, x_i];
5:    end for
6: end for
7: normalize(P);
8: on interrupt goto 25;
9: count := 1;
10: for each p ∈ I in turn do
11:    count := count + 1;
12:    for y := c_1 ... c_k do
13:       TP[y] := PP[y, x_p];
14:       for i := 1 ... m do
15:          TP[y] := TP[y] × CP[y, x_p, x_i];
16:       end for
17:    end for
18:    suspend interrupts;
19:    for y := c_1 ... c_k do
20:       P[y] := (P[y] × (count − 1) + TP[y]) / count;
21:    end for
22:    normalize(P);
23:    restore interrupts;
24: end for
25: return P;
```

3 The SAAPE system

The SAAPE system is an extension to the AAPE system. To classify an instance I, SAAPE uses the same Eq. 3 to estimate the probability of each class given I, and assigns the class of the highest probability to I. SAAPE maintains AAPE's capability of delivering classification results whenever the system is interrupted. In addition, SAAPE is able to handle the more general scenario where multiple instances (instead of one instance as AAPE addresses) require classification simultaneously. In such a situation, it is necessary to select which instance should be allocated computational resources next in order to optimize the overall classification efficacy and efficiency. This problem is analogous to the computer scheduling problem where multiple processes compete for the CPU at the same time, and is to be solved by SAAPE.

Table 2 Three credit card transactions (instances) I_1, I_2 and I_3 are waiting for approval simultaneously. Assume that two classifiers have been invoked for each instance. The probability of each class, 'accept' or 'reject', given each instance estimated by each invoked classifier is listed here. A scheduling scheme will assign a priority to each instance so as to decide which instance gets computed by a new SPODE if time allows

			Classifier_1	Classifier_2	SAAPE output
For instance I_1	Class label	'accept'	55%	55%	55%
		'reject'	45%	45%	45%
For instance I_2	Class label	'accept'	75%	35%	55%
		'reject'	25%	65%	45%
For instance I_3	Class label	'accept'	95%	55%	75%
		'reject'	5%	45%	25%

As a matter of demonstration convenience, we use the two-class credit card transaction approval application as a vehicle of illustration throughout this paper. Nonetheless, every algorithm proposed here is directly applicable to multiple-class problems. As shown in Table 2, three credit card transactions (instances) I_1, I_2 and I_3 are waiting for approval simultaneously. Assume that two classifiers have been invoked for each instance[3] and the probability of each class, 'accept' or 'reject', given each instance estimated by each invoked classifier is listed in Table 2. If classification time has not been exhausted yet, we need to assign each instance a priority value and the one with the highest priority will receive new computational resources, that is, to be classified by a new SPODE. We propose seven alternative scheduling strategies that calculate the priority of each instance as follows.

Note that in our current study SPODEs are invoked by the natural order of the superparent attributes that is inherent in the data set. We focus on comparing different scheduling schemes under this natural order.

3.1 First-come first-served (FCFS)

FCFS maintains a list of ordered probabilistic classifiers, NB, SPODE_1, $\text{SPODE}_2, \ldots,$ SPODE_m. Instances are (randomly) put into a single queue. Without losing generality, assume that the queue is I_1 followed by I_2 and then I_3. SAAPE with FCFS starts by processing I_1. It first computes NB for I_1 and then utilizes any additional time to refine I_1's probability estimates by invoking SPODEs in order. Only when all candidate SPODEs have finished examining I_1, the system proceeds to process I_2 in the queue and so on. This algorithm undertakes exhaustive application of the AAPE system to each individual instance in the instance pool in turn.

3.2 Round robin (RR)

RR maintains a list of ordered probabilistic classifiers, NB, SPODE_1, $\text{SPODE}_2, \ldots,$ SPODE_m. Instances are (randomly) put into a single queue $\langle I_1, I_2, I_3 \rangle$. SAAPE with RR starts with computing NB for each I_i $(i \in [1, 3])$. Next, it computes SPODE_1 for each I_i. The process keeps going until SPODE_m is applied to each I_i or whenever an interruption takes place.

[3] In practice, the number of invoked classifiers for each instance can be different.

3.3 Minimum-margin instance first (MMIF)

This approach is motivated by margin-based learning (Schapire et al. 1998; Bartlett 2007). To classify an instance I, SAAPE estimates the probability of each class label given I and assigns the class of the highest probability to I. Assume that the class label c_1 attains the highest predicted probability ($prob_1$) and the class label c_2 attains the second highest probability ($prob_2$). A *minimum-margin instance* I_{MMIF} is an instance whose $prob_1$–$prob_2$ is the smallest. It indicates that SAAPE's decision of classifying I_{MMIF} into c_1 is the most ambiguous among all test instances. Hence more resources need to be allocated to I_{MMIF} in order to differentiate between c_1 and c_2. MMIF will dispatch a new SPODE to I_{MMIF} first if there is still classification time available. If multiple instances obtain the same highest priority, the scheme randomly chooses one.

We now use the examples in Table 2 to demonstrate how MMIF works. For I_1, SAAPE predicts the class label 'accept' to have the highest probability ($prob_1 = 55\%$), and the class label 'reject' to have the second highest probability ($prob_2 = 45\%$). Hence the value of ($prob_1$–$prob_2$) for I_1 equals 10%. Likewise, the value of ($prob_1$–$prob_2$) for I_2 equals 10% (55%–45%), and for I_3 equals 50% (75%–25%). Thus both I_1 and I_2 are minimum-margin instances in this case. MMIF will randomly choose one to dispatch a new SPODE. Meanwhile I_3 is of a lower priority to get further computational resources because its classification if relatively clear-cut.

A potential problem of MMIF can occur if there exist many instances like I_1. As shown in Table 2, I_1 retains the minimum margin that does not change regardless of additional classifiers being applied to it. In this case, MMIF will futilely keep dispatching SPODEs to I_1 and waste much time on it. This consideration motivates the following scheduling schemes that can avoid this problem.

3.4 Controversial instance first

This family of scheduling schemes consider the variance among invoked classifiers' opinions regarding classifying an instance. The higher the variance, the more disagreement among invoked classifiers on how to classify an instance, and the higher priority this instance has to get a new SPODE. There are alternative ways to calculate the variance. We study three approaches here, variance regarding predicted class label, variance averaged on all class labels, and symmetric Kullback and Leibler distance. For each of the three schemes, if multiple instances obtain the same highest priority, the scheme randomly chooses one to dispatch a new SPODE.

3.4.1 Variance regarding predicted class label (VRPCL)

Given a test instance I, if I is predicted by SAAPE to belong to the class c_i, the variance equals to the variance of the probability $prob(c_i|I)$ estimated by each invoked classifier:

$$Var(X) = E((X - \mu)^2),$$

where the random variable X is each classifier's estimate of $prob(c_i|I)$, and $\mu = E(X)$ is the expected value (mean) of X and equals to SAAPE's output probability of $prob(c_i|I)$ according to Eq. 3.

We now use the examples in Table 2 to demonstrate how VRPCL works. For I_1, SAAPE predicts it to belong to 'accept' and its variance equals to $(55\% - 55\%)^2 +$

$(55\% - 55\%)^2 = 0$. For I_2, SAAPE predicts it to belong to 'accept' and its variance equals to $(75\% - 55\%)^2 + (35\% - 55\%)^2 = 0.08$. For I_3, SAAPE predicts it to belong to 'accept' and its variance equals to $(5\% - 25\%)^2 + (45\% - 25\%)^2 = 0.08$. Thus both I_2 and I_3 have the highest variance regarding the predicted class label. VRPCL will randomly choose one to dispatch a new SPODE. Meanwhile I_1 is of a lower priority to get further computational resources because its variance is very low.

3.4.2 Variance averaged on all class labels (VAOACL)

VAOACL is an extension to VRPCL. Given a test instance I, VAOACL first calculates the variance of *every* class label following the practice of VRPCL as in Sect. 3.4.1. It then averages the variance on all class labels. It is worth mentioning that in the two-class case, variance regarding predicted class label is equivalent to variance averaged on all class labels, whereas in the multiple-class case, they can be different. In Appendix A, we have first proved the equivalence case and then given an example to attest the difference case.

3.4.3 Symmetric Kullback and Leibler distance (SKLD)

The Kullback Leibler distance (KL-distance) (Kullback and Leibler 1951) is a natural distance function from one (true) probability distribution, p, to another (target) probability distribution, q. It can be interpreted as the expected extra message-length per datum due to using a code based on the target distribution compared to using a code based on the true distribution.

For discrete (not necessarily finite) probability distributions, $p = \langle p_0, p_1, \ldots, p_k \rangle$ and $q = \langle q_0, q_1, \ldots, q_k \rangle$, the KL-distance is defined to be

$$KL(p, q) = \sum_{i=0\ldots k} p_i \log_2 \left(\frac{p_i}{q_i} \right).$$

Note that the KL-distance is not, in general, symmetric, that is $KL(p, q) \neq KL(q, p)$. However in our context, we do not differentiate the directions from one classifier to another. Hence here we use the symmetric version of the Kullback and Leibler distance, known also as the J-divergence (Bekara et al. 2006; Jeffreys 1946) as a measure of the difference between two classifiers' estimates on the class distribution of an instance:

$$SKL(p, q) = KL(p, q) + KL(q, p).$$

Our system can invoke many classifiers for an instance if time allows. This algorithm first calculates the symmetric Kullback and Leibler distance for each pair of classifiers. It then obtains the distance averaged on all the pairs. The bigger the averaged distance, the more different each classifier's judgement on the class distribution of this test instance, and the higher priority this instance has to get a new SPODE. In the case of I_2 in Table 2, the symmetric Kullback and Leibler distance between Classifier_1 and Classifier_2 is calculated as Eq. 4.

$$\begin{aligned} &SKL(\langle 75\%, 25\% \rangle, \langle 35\%, 65\% \rangle) \\ &\quad = KL(\langle 75\%, 25\% \rangle, \langle 35\%, 65\% \rangle) \\ &\qquad + KL(\langle 35\%, 65\% \rangle, \langle 75\%, 25\% \rangle) \end{aligned}$$

$$= 75\% \times \log_2\left(\frac{75\%}{35\%}\right) + 25\% \times \log_2\left(\frac{25\%}{65\%}\right)$$
$$+ 35\% \times \log_2\left(\frac{35\%}{75\%}\right) + 65\% \times \log_2\left(\frac{65\%}{25\%}\right)$$
$$= 0.9912. \quad (4)$$

Following the same calculation, the symmetric Kullback and Leibler distance for I_1 is 0 and for I_3 is 1.5834. Hence I_3 obtains the highest priority and gets the next new classifier. I_2 obtains a medium priority and I_1 obtains the lowest priority.

3.5 A hybrid approach (Hybrid)

This scheduling algorithm incorporates the ideas of the minimum-margin instance first scheduling and the controversial instance first scheduling. For each instance, not only does it consider the difference between the highest probability estimate $prob_1$ and the second highest probability estimate $prob_2$, it also takes into account the deviation when SAAPE derives $prob_1$ and $prob_2$ from each invoked classifier's estimate.

The hybrid approach assumes that different classifiers' probability estimates of a class given an instance I follow a Gaussian distribution. It then measures the priority by the overlapping area of two Gaussian curves G_1 and G_2, where G_1 is the curve of the class assigned with the highest probability ($prob_1$) by SAAPE and G_2 the second highest ($prob_2$). The bigger the overlapping area, the more similarity between the most and second most probable class for I, and the higher the priority for I to get a new SPODE to differentiate between the two class labels.

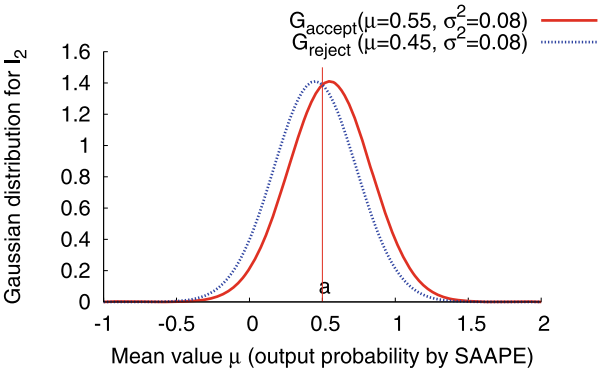
We now use the examples in Table 2 to demonstrate how this hybrid scheduling scheme works. For I_2 and the highest probability class 'accept', the mean value (μ, also known as $prob_1$ and SAAPE's output probability) of different classifiers' probability estimates is $\frac{75\%+35\%}{2} = 0.55$; and the variance σ^2 is $(75\% - 55\%)^2 + (35\% - 55\%)^2 = 0.08$. Hence the Gaussian curve is as the red (solid) curve in Fig. 15(a). Likewise, for I_2 and the second highest probability class 'reject', the Gaussian curve is as the blue (dotted) curve in Fig. 15(a). For I_3, the 'accept' and 'reject' classes are respectively associated with the red and blue curves in Fig. 15(b). The overlapping area is calculated as:

$$\int_{-\infty}^{a} G_{accept}(\mu, \sigma) + \int_{a}^{\infty} G_{reject}(\mu, \sigma)$$

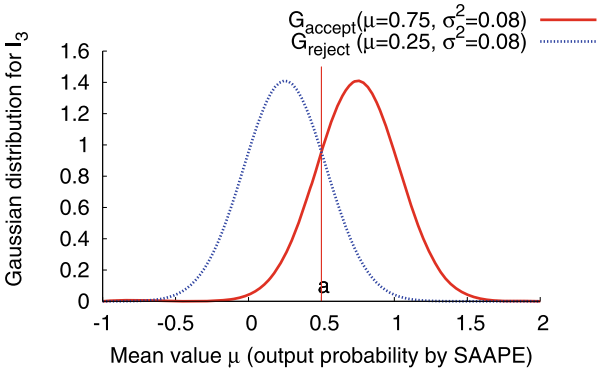
where $G(\mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} \exp(-\frac{(x-\mu)^2}{2\sigma^2})$ is the Gaussian probability density function, and a is the horizontal value of the intersection point of the two curves.

One can see from Fig. 4 that the overlapping area of I_2 is bigger than that of I_3. As for I_1, because its σ^2 equals to 0, there is no explicit Gaussian curve associated with it, and the hybrid scheduling simply defines its overlapping area as 0. This might be justified on the assumption that if multiple classifiers agree, the probability of further classifiers differing substantially is low. Instead, one might need to add more informing attributes to describe this instance in order to classify it. As a result, the hybrid scheduling will prioritize I_2 followed by I_3 and lastly I_1.

Note that if multiple instances have the same overlapping area, we use the value ($prob_1$–$prob_2$) as the tie breaker. Whichever instance has the smallest value of ($prob_1$–$prob_2$), the difference between the highest predicted class probability and the second highest predicted



(a) I_2



(b) I_3

Fig. 4 Priority equals to the overlapping area of curves G_{accept} and G_{reject}

class probability, gets a new SPODE. This has the same effect of the minimum-margin instance first scheduling in Sect. 3.3 and is inspired by our observation of the very good empirical performance of the minimum-margin instance first scheduling scheme.

The pseudo codes of the SAAPE system and every scheduling algorithm are presented in Appendix B to avoid distraction from the main text.

4 Classification time complexity analysis

All the scheduling work takes place at the classification time. Assume the number of attributes, classes and test instances are respectively m, k and n.

NB is applied to an instance to estimate the probability of every class given this instance. This incurs a complexity of $\mathbf{O}(\mathbf{mk})$ because for each class ($O(k)$), NB iterates through each attribute x_i ($O(m)$) to calculate $\hat{P}(x_i \mid y)$ as in Eq. 1.

SPODE, at each step, is dispatched to an instance to estimate the probability of every class given this instance. This incurs a complexity of $O(mk)$ because for each class ($O(k)$), the SPODE iterates through each attribute x_i other than the superparent x_p ($O(m)$) to calculate $\hat{P}(x_i \mid x_p, y)$ as in Eq. 2. In the context of anytime classification, the previous average probability estimate for each class is also updated to account for this newly applied SPODE

($O(k)$). As a result, at each step the complexity of dispatching a SPODE to an instance is $O(mk + k) = \mathbf{O(mk)}$.

All the scheduling schemes share three complexity components. First, apply NB to all the n test instances in the contract portion of the classification time budget. This complexity is of the order $O(mkn)$, resulting from an $O(mk)$ algorithm applied over an $O(n)$ sized test set. Second, identify the instance I that attains the highest priority ($O(n)$). Third, dispatch a SPODE to I ($O(mk)$).

What differs is how each scheduling scheme prioritizes instances *at each step*, that is, how each scheme decides which instance to get a new SPODE. This distinguishing complexity component is analyzed for each scheduling scheme as follows.

First-come first-served scheduling simply dispatches all SPODEs in turn to an instance. Hence, the complexity of prioritizing is $\mathbf{O(1)}$.

Round robin scheduling simply dispatches a single SPODE to all instances in turn. Hence, the complexity of prioritizing is $\mathbf{O(1)}$.

Borderline instance first scheduling iterates through every test instance ($O(n)$), identifying its $prob_1$ and $prob_2$ ($O(k)$) and calculating the value of ($prob_1$–$prob_2$) ($O(1)$) . Hence, the complexity of prioritizing is $\mathbf{O(nk)}$.

Controversial instance first (*variance regarding predicted class label*) *scheduling* iterates through every test instance ($O(n)$), identifying the predicted class label ($O(k)$) and calculating the variance on the predicted class label among invoked classifiers ($O(m)$). Hence, the complexity of prioritizing is $\mathbf{O(n(k + m))}$.

Controversial instance first (*variance averaged on all class labels*) *scheduling* iterates through every test instance ($O(n)$). For each instance, firstly it loops through every class label ($O(k)$) to calculate the variance among invoked classifiers ($O(m)$); and secondly it averages all the class labels' variance ($O(m)$). Hence, the complexity of prioritizing is $O(n(km + m)) = \mathbf{O(nkm)}$.

Controversial instance first (*symmetric Kullback and Leibler distance*) *scheduling* iterates through every test instance ($O(n)$), calculating its symmetric Kullback and Leibler distance ($O(km^2)$). Hence, the complexity of prioritizing is $\mathbf{O(nkm^2)}$.

A hybrid approach scheduling iterates through every test instance ($O(n)$), identifying $prob_1$ and $prob_2$ ($O(k)$), computing μ and σ of the top two class labels ($O(m)$), and calculating the overlapping area of their Gauss curves ($O(1)$). Hence, the complexity of prioritizing is $\mathbf{O(n(k + m))}$.

5 Experiments

We conduct large-scale experiments to test the anytime classification performance of SAAPE by using an extensive suite of 60 benchmark data sets and a wide variety of statistical tests. Our goal is to thoroughly evaluate and rank rival alternative scheduling schemes in SAAPE under varying classification time resources including first-come first-served (FCFS); round robin (RR); minimum-margin instance first (MMIF); controversial instance first that comprises variance regarding predicted class label (VRPCL), variance averaged on all class labels (VAOACL), and symmetric Kullback and Leibler distance (SKLD); and lastly, the hybrid approach (Hybrid).

Note that the FCFS scheme equals to exhaustive application of the AAPE system to each individual instance in the instance pool in turn. Hence the observations and evaluations for FCFS are the same as those for AAPE classifying multiple instances under computational resource constraints.

Because the SAAPE system can deliver probability estimates of class labels given each test instance as well as output zero-one loss classification, we will study each scheduling scheme's performance regarding both outputs.

5.1 Experimental design

Details of our experimental setup are presented here.

Data We use a large suite of 60 benchmark data sets from the UCI Machine Learning Repository (Asuncion and Newman 2007), as described in Table 3.

Table 3 Description of 60 experimental data sets: each data set's name, number of instances (Ins.), number of attributes (Att.) and number of class labels (Cls.)

Data set	Ins.	Att.	Cls.	Data set	Ins.	Att.	Cls.
Abalone	4177	8	3	Annealing	898	38	6
Audiology	226	69	24	Automoblie	205	25	7
BalanceScale	625	4	3	BreastCancer	699	9	2
Bands	539	36	2	Bridges	105	12	6
LiverDisorders	345	6	2	Chess	551	39	2
Car	1728	6	4	ContraceptiveMethodChoice	1473	9	3
CreditApproval	690	15	2	Diabetes	768	8	2
Echocardiogram	131	6	2	Flags	194	29	8
German	1000	20	2	GlassIdentification	214	9	7
HeartDiseaseCleveland	303	13	5	Hepatitis	155	19	2
HorseColic	368	22	2	HouseVotes84	435	16	2
HeartDiseaseHungarian	294	13	2	Hypothyroid	3772	29	4
Ionosphere	351	34	2	IrisPlant	150	4	3
KRvsKP	3196	36	2	LaborRelations	57	16	2
LEDDisplay	1000	7	10	LetterRecognition	20000	16	26
LungCancer	32	56	3	Lymphography	148	18	4
MultipleFeaturesMorphological	2000	6	10	Mushrooms	8124	22	2
Musk	476	166	2	NewThyroid	215	5	3
Optdigits	5620	48	10	PageBlocks	5473	10	5
PenBasedRecognition	10992	16	10	PhoneNettalkPhonememe	5438	7	50
Postoperative	90	8	3	PrimaryTumor	339	17	22
Promoter	106	57	2	SatelliteImage	6435	36	6
ImageSegmentation	2310	19	7	SickEuthyroid	3772	29	2
AustralianSignLanguage	12546	8	3	SolarFlare	1389	9	2
Sonar	208	60	2	Soybean	683	35	19
SpliceJunctionice	3190	61	3	SyntheticControl	600	60	6
TicTacToe	958	9	2	Vehicle	846	18	4
Volcanoes	1520	3	4	Vowel	990	13	16
Waveform	5000	40	3	Wine	178	13	3
Yeast	1484	8	10	Zoo	101	17	7

Simulation of varying classification time We use *step*s to simulate the varying classification time resources. One step equals to the amount of time that allows to apply one SPODE to one instance. The maximum meaningful number of steps for a test data set $\mathcal{D}$ with n instances and m attributes equals $n \times m$. If the SAAPE system has time available more than $n \times m$ steps when classifying D, it will finish applying every candidate SPODE to every instance and its performance will remain constant afterwards. As for our experimental suite of data sets, 'LetterRecognition' requires the biggest number of meaningful steps, 32000 resulting from 2000 test instances multiplying 16 attributes in each fold of the stratified 10-fold cross validation that is to be explained next.

Performance measured The SAAPE system is implemented in the WEKA machine learning environment (Witten and Frank 2005). To evaluate SAAPE's performance, on each data set, the WEKA's default stratified 10-fold cross validation is performed. Each data set is divided into 10 mutually exclusive subsets (folds) $D_1, D_2, \ldots, D_{10}$ of approximately equal size and approximately the same class proportions as in the full data set D. The classifier is trained and tested 10 times. Each time $t \in \{1, 2, \ldots, 10\}$, it is trained on $D - D_t$ and tested on D_t (Kohavi 1995; Kohavi and Provost 1998). As a result, each instance of D is taken as a test instance and classified once.

The following performance measures are recorded for each alternative scheduling scheme on each data set under the stratified 10-fold cross validation.

- *Zero-one loss classification error rate* measures the overall classification performance of a classifier under zero-one loss (Witten and Frank 2005). The classifier predicts the class of each test instance. If the prediction is correct, it is counted as a *success* and the loss is zero; if not, it is counted as an *error* and the loss is one. The zero-one loss classification error rate is the proportion of errors made over all the test instances.
- *Root mean squared error* measures the quality of the probability estimates when a classifier outputs a probability associated with each prediction (Witten and Frank 2005). Assume the actual probability of a test instance I_i belonging to each of the k classes is $a_{i1}, a_{i2}, \ldots, a_{ik}$ respectively; and the probability predicted by the classifier is $p_{i1}, p_{i2}, \ldots, p_{ik}$ respectively. The root mean squared error of the classifier over n test instances equals to

$$\sqrt{\frac{\sum_{i=1}^{n} \left(\frac{(p_{i1}-a_{i1})^2+\cdots+(p_{ik}-a_{ik})^2}{k} \right)}{n}}.$$

- *Rank* orders competing algorithms for each data set following the practice of Friedman (1937, 1940). The algorithm that attains the best performance is ranked 1, the second best ranked 2, so on and so forth. An algorithm's average rank can be obtained by averaging its ranks across all data sets. Compared with mean value such as the arithmetic mean of the classification error rates across all data sets, average rank can reduce the susceptibility to outliers that, for instance, allows a classifier's excellent performance on one data set to compensate for its overall bad performance (Demsar 2006).
- *Anytime classification curve* of an algorithm A on a data set D (or averaged on multiple data sets) is a two-dimensional curve whose horizontal axis corresponds to the exhausted classification time resource (number of steps) by the time t, and whose vertical axis corresponds to A's some performance measure such as the classification error rate by the time t. If the number of steps becomes bigger than the meaningful number of steps required by a data set, the error rate value remains that of the full NB+SPODEs ensemble. In practice, this means that if the classification time budget is more than what SAAPE needs to finish, SAAPE always returns its complete calculation.

Statistical tests employed A variety of statistical tests are employed to evaluate measured performance of each competing scheme.

- *Win/lose/tie record and binomial sign test* can be applied to *compare each pair of algorithms across multiple data sets*. A win/lose/tie record can be calculated for each pair of competing algorithms A and B with regard to a performance measure M. The record represents the number of data sets in which A respectively beats, loses to or ties with B on M. A one-tailed binomial sign test can be applied to wins versus losses. If its result is less than the critical level of 0.05, the wins against losses are statistically significant, supporting the claim that the winner has a systematic (instead of by chance) advantage over the loser.
- *Friedman test* and *Nemenyi test* can be applied to *compare multiple classifiers across multiple data sets*. As recommended by Demsar (2006), the Friedman test compares schemes' average ranks to decide whether to reject the null-hypothesis, which states that all the schemes are equivalent and so their ranks should be equal. If the Friedman test rejects its null-hypothesis, we can proceed with a post-hoc test, the Nemenyi test (Demsar 2006). It can be applied to average ranks of competing schemes and indicate whose performances have statistically significant differences (here we use the 0.05 critical level).

5.2 Observations and analysis

Rival algorithms' anytime classification performances are presented and analyzed in this section. Because the SAAPE system can deliver probability estimates of class labels given each test instance as well as output zero-one loss classification, we will study each scheduling scheme's performance regarding both outputs.

5.2.1 Performance regarding zero-one loss classification

For each alternative scheduling scheme Fig. 5 illustrates its anytime classification curve regarding SAAPE's zero-one loss classification error rate averaged on 60 experimental data sets. Every scheduling scheme starts with NB, resulting in the same classification error rate at $step = 0$. A rapid decrease in error is witnessed at the early stages (approximately before $step = 50 \times 100$) of dispatching SPODEs to test instances. After $step = 50 \times 100$, the error decrease slows down and the relative performance among alternative scheduling schemes largely remains the same. When available classification time is long enough so that SAAPE dispatches every candidate SPODE to every test instance, all scheduling schemes converge to the same classification error rate. Because most change takes place before $step = 50 \times 100$, we zoom in the part of Fig. 5 where $step \in [0, 50 \times 100]$ to have a detailed study. This results in Fig. 6.

For each alternative scheduling scheme Fig. 7 illustrates its rank averaged on 60 experimental data sets regarding SAAPE's zero-one loss classification rate under varying classification time. Figure 8 is the zoom in version of Fig. 7 where $step \in [0, 50 \times 100]$.

Experimental results on both error rates and ranks (Figs. 5, 6, 7, 8) reveal that in general Hybrid is the most effective at using extra classification time to reduce classification error rate. MMIF is the second best followed by RR and FCFS. The three versions of controversial instance first scheduling, VRPCL, VAOACL and SKLD, are the least effective at reducing classification error rate and do not have a big performance difference among themselves although VAOACL often gets the worst ranking.

It is interesting to note that at the very beginning MMIF obtains lower error than Hybrid more often than not. We suggest the reason is that Hybrid must first allocate every test instance a SPODE in addition to NB in order to calculate μ and σ for the Gaussian curve.

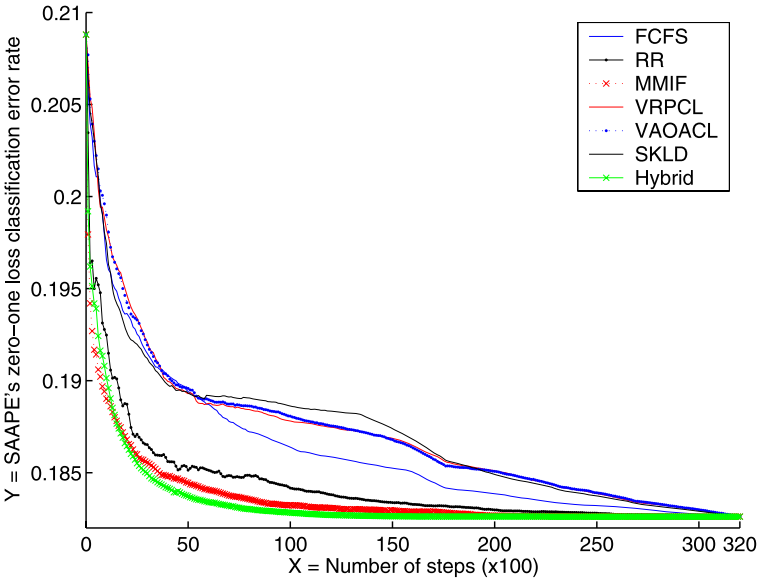


Fig. 5 Anytime classification curve of each scheduling algorithm's zero-one loss classification performance. *The horizontal axis* is the number of steps simulating the available classification time resource. *The vertical axis* is SAAPE's classification error rate averaged on 60 experimental data sets

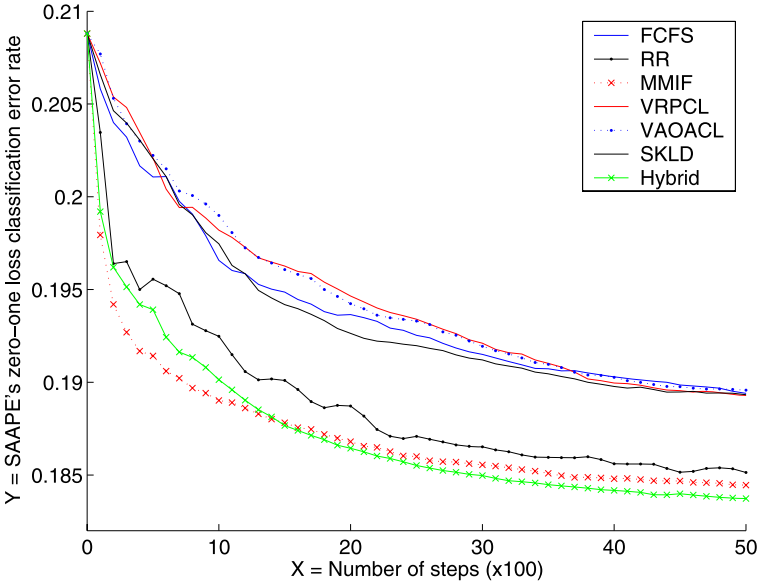


Fig. 6 Zoom in the part of Fig. 5 where $step \in [0, 50 \times 100]$

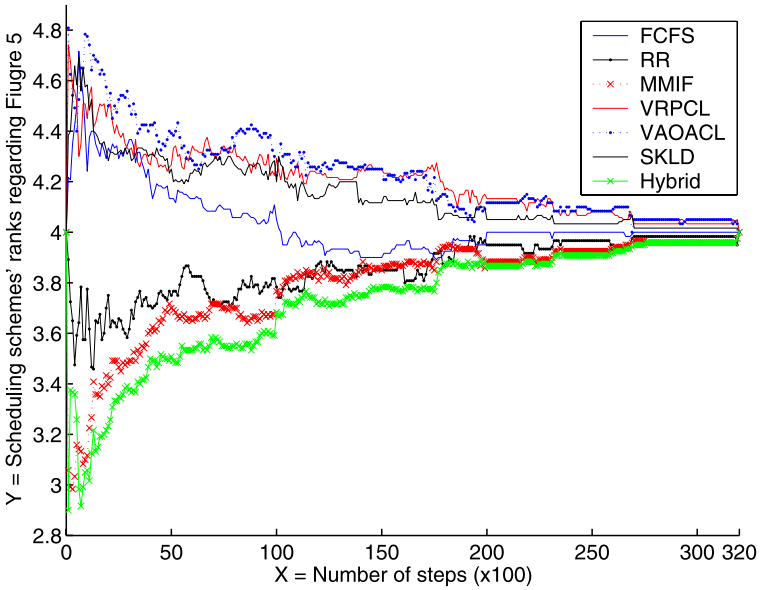


Fig. 7 Anytime classification curve of each scheduling algorithm's average rank regarding zero-one loss classification performance. *The horizontal axis* is the number of steps simulating the available classification time resource. *The vertical axis* is each scheduling algorithm's rank in terms of SAAPE's classification error rate and averaged on 60 experimental data sets

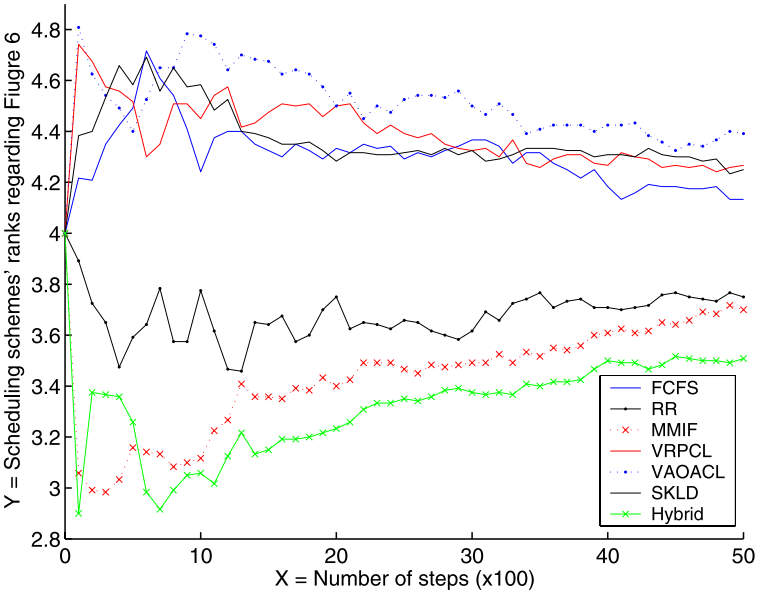


Fig. 8 Zoom in the part of Fig. 7 where $step \in [0, 50 \times 100]$

This has an effect similar to RR (Round Robin)[4] and is less effective than MMIF that can keep adding SPODEs to refine the class probability estimates for the minimum-margin instance(s) so long as every test instance has been assigned its initial NB. Nonetheless, as time progresses Hybrid gradually overtakes MMIF and becomes the most effective scheduling strategy. We suggest the reason is because Hybrid not only considers the difference between the highest probability estimate $prob_1$ and the second highest probability estimate $prob_2$, it also takes into account the deviation when SAAPE derives $prob_1$ and $prob_2$ from each invoked classifier's estimate.

It is also worth noting that RR outperforms FCFS. This confirms our previous observations in the AAPE system that when adding SPODEs into the classifier ensemble to classify an instance, very often the biggest improvements of classification performance take place at the early stages, such as when adding the first few SPODEs (Yang et al. 2007). Hence when classifying a pool of instances under constraint time resources, it is more effective to dispatch the first SPODE to every instance, then the second SPODE to every instance and so on (RR), rather than to dispatch all the SPODEs to the first instance, then all the SPODEs to the second instance and so on (FCFS). RR exchanges time for substantial improvements by focusing on early classification stages, while FCFS exchanges time for trivial as well as big improvements by always proceeding to the later classification stages, which is not so rewarding.

Meanwhile we can conduct various statistical tests at each step to find out exactly which schemes are significantly different at this time point. To statistically compare each pair of algorithms across 60 experimental data sets, a win/lose/tie (w/l/t) record is calculated with regard to the classification error rate. For example, Table 4(a) lists the win/lose/tie records when $step = 200$. A boldface entry indicates that the wins against losses are statistically significant using a one-tailed binomial sign test at the 0.05 critical level. For instance, it is observed that when $step = 200$, RR achieves lower error than VRPCL (w/l/t = 24/10/26), VAOACL (w/l/t = 25/8/27) and SKLD (w/l/t = 22/11/27) with statistically significant frequency. Both MMIF and Hybrid achieve lower error than FCFS, VRPCL, VAOACL and SKLD with statistically significant frequency. MMIF wins more often than not compared with both RR and Hybrid. The same statistical test procedure can be applied to any time point. We have also listed two sets of representative results when $step = 2{,}000$ and $step = 20{,}000$ in Table 4.

To compare all algorithms in one go across 60 experimental data sets, we use the Friedman test and the Nemenyi test following Demsar's proposal (Demsar 2006). For example, when $step = 200$, the average ranks of FCFS, RR, MMIF, VRPCL, VAOACL, SKLD and Hybrid are respectively 4.2083, 3.725, 2.9917, 4.675, 4.625, 4.4 and 3.375. When we apply the Friedman test, with 7 algorithms and 60 data sets, F_F is distributed according to the F distribution with $(7-1) = 6$ and $(7-1) \times (60-1) = 354$ degrees of freedom. The critical value of F(6, 354) at the 0.05 critical level is 2.12. F_F calculated from the average ranks is 5.87. Since $5.87 > 2.12$, we can reject the null hypothesis and infer that there exists significant difference among rival schemes. To find out exactly which schemes are significantly different, we proceed to the Nemenyi test, whose results are illustrated in Fig. 9(a). The average rank of each scheme is pointed by a circle. The horizontal bar across each circle indicates the critical difference which equals to 1.16 in this case. The performance of two methods is

[4]But Hybrid is not initially the same as round robin because in this particular round of dispatching the first SPODE to test instances, under hybrid scheduling the instance that has the smallest value of $(prob_1 – prob_2)$ will get the SPODE first, while under round robin scheduling test instances get the SPODE following their natural order in the data set.

Table 4 Given a time point, every pair of algorithms' win/lose/tie records with regard to the classification error rate across 60 data sets. Each entry indicates that the algorithm of the row compares against the algorithm of the column. A statistically significant record (at the 0.05 critical level) is indicated *in bold*

	Algorithm	FCFS	RR	MMIF	VRPCL	VAOACL	SKLD	Hybrid
(a) $step = 200$	FCFS	–	22/23/15	12/34/14	24/18/18	23/20/17	23/20/17	14/28/18
	RR	23/22/15	–	19/27/14	**24/10/26**	**25/8/27**	**22/11/27**	21/23/16
	MMIF	**34/12/14**	27/19/14	–	**39/8/13**	**37/10/13**	**36/9/15**	22/16/22
	VRPCL	18/24/18	10/24/26	8/39/13	–	8/9/43	7/14/39	9/31/20
	VAOACL	20/23/17	8/25/27	10/37/13	9/8/43	–	6/14/40	9/30/21
	SKLD	20/23/17	11/22/27	9/36/15	14/7/39	14/6/40	–	9/31/20
	Hybrid	**28/14/18**	23/21/16	16/22/22	**31/9/20**	**30/9/21**	**31/9/20**	–
(b) $step = 2{,}000$	FCFS	–	5/14/41	2/17/41	10/9/41	10/9/41	9/10/41	1/18/41
	RR	**14/5/41**	–	4/15/41	**17/0/43**	**17/0/43**	**16/2/42**	1/17/42
	MMIF	**17/2/41**	**15/4/41**	–	**19/1/40**	**19/1/40**	**17/3/40**	4/8/48
	VRPCL	9/10/41	0/17/43	1/19/40	–	4/4/52	3/8/49	0/19/41
	VAOACL	9/10/41	0/17/43	1/19/40	4/4/52	–	3/8/49	0/19/41
	SKLD	10/9/41	2/16/42	3/17/40	8/3/49	8/3/49	–	0/17/43
	Hybrid	**18/1/41**	**17/1/42**	8/4/48	**19/0/41**	**19/0/41**	**17/0/43**	–
(c) $step = 20{,}000$	FCFS	–	0/3/57	0/3/57	3/0/57	3/0/57	3/0/57	0/3/57
	RR	3/0/57	–	0/3/57	3/0/57	3/0/57	3/0/57	0/3/57
	MMIF	3/0/57	3/0/57	–	3/0/57	3/0/57	3/0/57	0/1/59
	VRPCL	0/3/57	0/3/57	0/3/57	–	1/2/57	0/3/57	0/3/57
	VAOACL	0/3/57	0/3/57	0/3/57	2/1/57	–	0/3/57	0/3/57
	SKLD	0/3/57	0/3/57	0/3/57	3/0/57	3/0/57	–	0/3/57
	Hybrid	3/0/57	3/0/57	1/0/59	3/0/57	3/0/57	3/0/57	–

significantly different if their corresponding average ranks differ by at least the critical difference. In other words, two methods are significantly different if their horizontal bars are not overlapping. For instance, Fig. 9(a) reveals that when $step = 200$, MMIF is ranked the best and is statistically significantly better than FCFS, VRPCL, VAOACL and SKLD. The same statistical test procedure can be applied to any time point. We have also illustrated two sets of representative results when $step = 2{,}000$ and $step = 20{,}000$ in Fig. 9, from which we observe that MMIF is ranked better than all the other scheduling schemes at the beginning ($step = 200$), Hybrid overtakes MMIF and becomes the best latter on ($step = 2{,}000$), and all schemes converge to the same performance eventually.

5.2.2 *Performance regarding probability estimation*

Alternative scheduling schemes' relative performance on probability estimation presents a general trend similar to that of their zero-one loss classification performance. For each alternative scheduling scheme Fig. 10 illustrates its anytime classification curve regarding SAAPE's root mean squared error averaged on 60 experimental data sets. Figure 11 zooms in the part of Fig. 10 where $step \in [0, 50 \times 100]$ because substantial performance changes take place during this period. For each alternative scheduling scheme Fig. 12 illustrates its rank averaged on 60 experimental data sets regarding SAAPE's root mean squared error

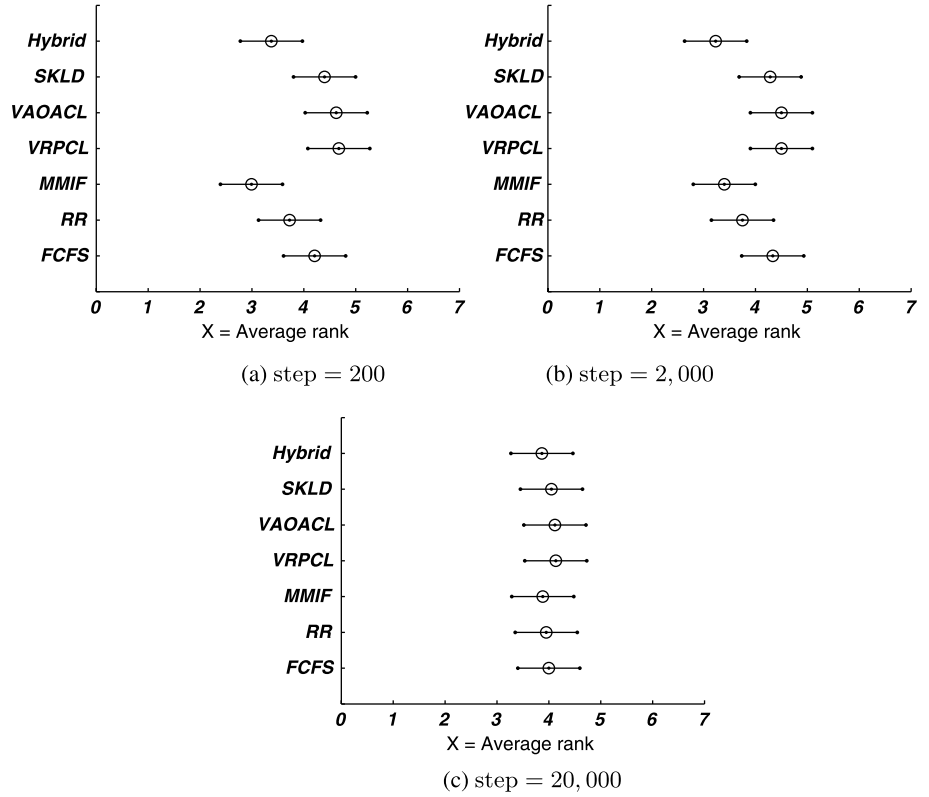


Fig. 9 Apply the Nemenyi test to alternative schemes' average ranks of classification error rate

under varying classification time and Fig. 13 is the zoom in version of Fig. 12 where *step* $\in [0, 50 \times 100]$.

In general, Hybrid is the most effective at using extra classification time to reduce probability estimation error for SAAPE although MMIF wins Hybrid more often than not at the very beginning. FCFS presents mediocre performance. The three versions of controversial instance first scheduling, VRPCL, VAOACL and SKLD, are the least effective at reducing probability estimation error and do not have a big performance difference among themselves although VAOACL often gets the worst ranking.

One observation worth mentioning is that the performance difference between competitive methods such as RR and MMIF has slightly narrowed down when the performance is measured by probability estimate error rather than by zero-one loss classification error (contrasting Fig. 6 and Fig. 11, or contrasting Fig. 8 and Fig. 13). We suggest that the reason lies in the distinction between the two measurements' nature. We measure probability estimate by root mean squared error. Given an instance (I) and two candidate class labels ($+$, $-$), assume the true class label of I is $+$. Assume a Bayesian classifier ($\mathcal{B}_1$) predicts $\hat{P}(+|I) = 60\%$ and $\hat{P}(-|I) = 40\%$, and another Bayesian classifier ($\mathcal{B}_2$) predicts $\hat{P}(+|I) = 45\%$ and $\hat{P}(-|I) = 55\%$. Then $\mathcal{B}_1$'s probability estimate error equals to $\sqrt{(1-0.6)^2 + (0-0.4)^2} = 0.57$ and $\mathcal{B}_2$'s equals to 0.78. The difference between $\mathcal{B}_1$ and $\mathcal{B}_2$ is thus 0.21. In contrast, $\mathcal{B}_1$'s zero-one loss classification error equals to 0 since $\hat{P}(+|I) > \hat{P}(-|I)$, and $\mathcal{B}_2$'s equals to 1 since $\hat{P}(+|I) < \hat{P}(-|I)$. The difference between

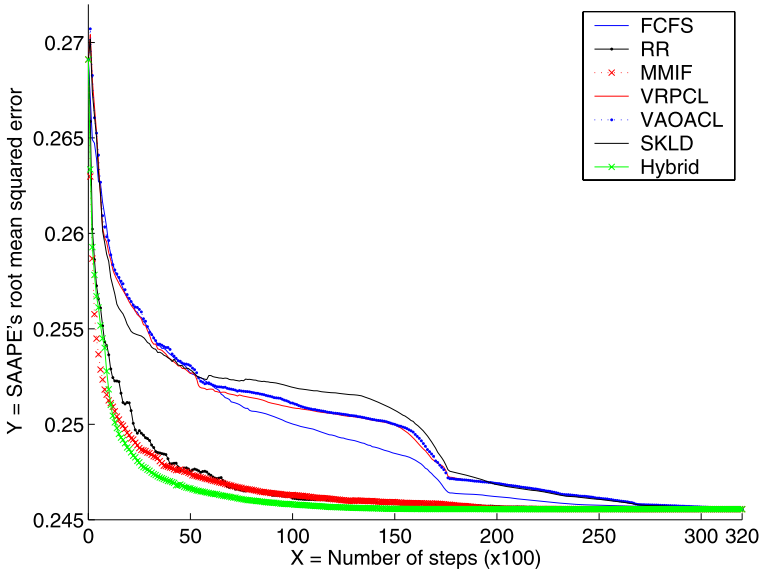


Fig. 10 Anytime classification curve of each scheduling algorithm's probability estimation performance. *The horizontal axis* is the number of steps simulating the available classification time resource. *The vertical axis* is SAAPE's root mean squared error averaged on 60 experimental data sets

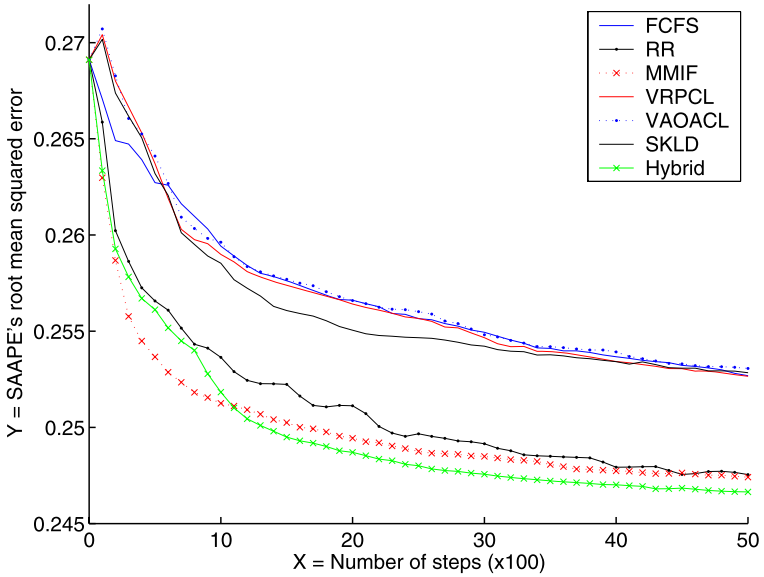


Fig. 11 Zoom in the part of Fig. 10 where $step \in [0, 50 \times 100]$

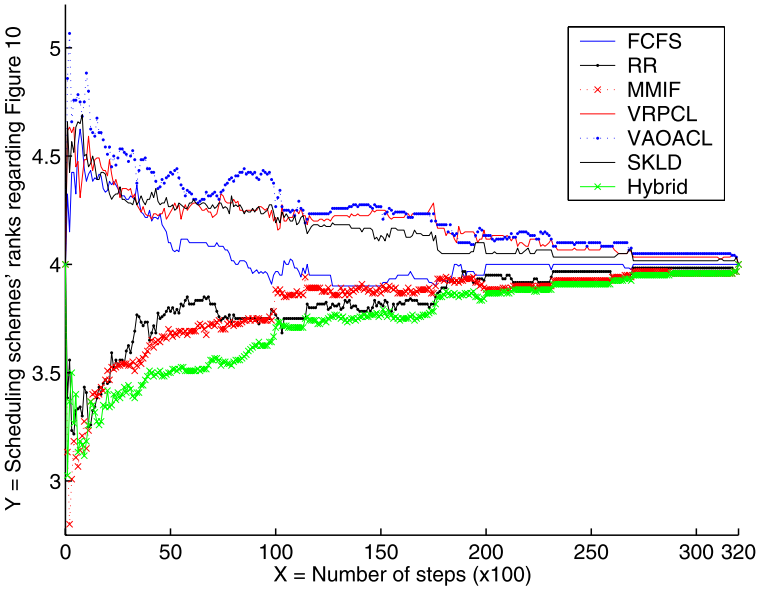


Fig. 12 Anytime classification curve of each scheduling algorithm's average rank regarding probability estimation performance. *The horizontal axis* is the number of steps simulating the available classification time resource. *The vertical axis* is each scheduling algorithm's rank in terms of SAAPE's root mean squared error and averaged on 60 experimental data sets

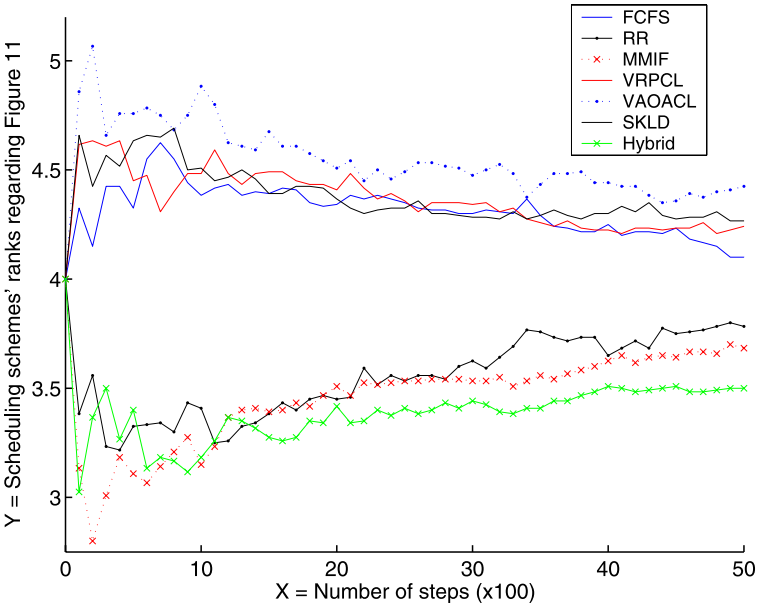


Fig. 13 Zoom in the part of Fig. 12 where *step* $\in [0, 50 \times 100]$

Table 5 Given a time point, every pair of algorithms' win/lose/tie records with regard to the root mean squared error across 60 data sets. Each entry indicates that the algorithm of the row compares against the algorithm of the column. A statistically significant record (at the 0.05 critical level) is indicated *in bold*

	Algorithm	FCFS	RR	MMIF	VRPCL	VAOACL	SKLD	Hybrid
(a) $step = 200$	FCFS	–	22/27/11	12/37/11	27/20/13	29/18/13	27/22/11	19/30/11
	RR	27/22/11	–	17/33/10	**30/6/24**	**31/6/23**	**28/11/21**	22/24/14
	MMIF	**37/12/11**	**33/17/10**	–	**42/8/10**	**43/7/10**	**38/13/9**	28/20/12
	VRPCL	20/27/13	6/30/24	8/42/10	–	**20/6/34**	15/16/29	10/34/16
	VAOACL	18/29/13	6/31/23	7/43/10	6/20/34	–	9/22/29	8/37/15
	SKLD	22/27/11	11/28/21	13/38/9	16/15/29	**22/9/29**	–	14/32/14
	Hybrid	30/19/11	24/22/14	20/28/12	**34/10/16**	**37/8/15**	**32/14/14**	–
(b) $step = 2{,}000$	FCFS	–	3/17/40	4/16/40	9/11/40	11/9/40	10/10/40	2/17/41
	RR	**17/3/40**	–	9/11/40	**19/0/41**	**19/0/41**	**18/1/41**	9/10/41
	MMIF	**16/4/40**	11/9/40	–	**18/2/40**	**19/1/40**	**17/3/40**	8/11/41
	VRPCL	11/9/40	0/19/41	2/18/40	–	8/6/46	7/8/45	1/18/41
	VAOACL	9/11/40	0/19/41	1/19/40	6/8/46	–	7/8/45	0/19/41
	SKLD	10/10/40	1/18/41	3/17/40	8/7/45	8/7/45	–	2/17/41
	Hybrid	**17/2/41**	10/9/41	11/8/41	**18/1/41**	**19/0/41**	**17/2/41**	–
(c) $step = 20{,}000$	FCFS	–	0/3/57	0/3/57	3/0/57	3/0/57	3/0/57	0/3/57
	RR	3/0/57	–	0/3/57	3/0/57	3/0/57	3/0/57	0/3/57
	MMIF	3/0/57	3/0/57	–	3/0/57	3/0/57	3/0/57	0/1/59
	VRPCL	0/3/57	0/3/57	0/3/57	–	2/1/57	0/3/57	0/3/57
	VAOACL	0/3/57	0/3/57	0/3/57	1/2/57	–	0/3/57	0/3/57
	SKLD	0/3/57	0/3/57	0/3/57	3/0/57	3/0/57	–	0/3/57
	Hybrid	3/0/57	3/0/57	1/0/59	3/0/57	3/0/57	3/0/57	–

$\mathcal{B}_1$ and $\mathcal{B}_2$ is thus 1. In this sense, the probability estimate measurement resembles a continuous variable while the classification measurement resembles a categorical variable obtained by discretizing probabilities into either 0 or 1. As a result, zero-one loss classification error tends to magnify the difference between rival methods.

Following the practice detailed in Sect. 5.2.1, we can conduct various statistical tests at each step to find out exactly which schemes are significantly different at this time point. To statistically compare each pair of algorithms across 60 experimental data sets, a win/lose/tie (w/l/t) record is calculated with regard to the root mean squared error. Table 5 lists three sets of representative results when $step = 200$, $step = 2{,}000$ and $step = 20{,}000$. For instance, it is observed that when $step = 200$, MMIF achieves lower error than FCFS (w/l/t = 37/12/11), RR (w/l/t = 33/17/10), VRPCL (w/l/t = 42/8/10), VAOACL(w/l/t = 43/7/10) and SKLD (w/l/t = 38/13/9) with statistically significant frequency; and MMIF wins Hybrid more often than not (w/l/t = 28/20/12), although not statistically significantly. When $step = 2{,}000$, Hybrid achieves lower error than FCFS, VRPCL, VAOACL and SKLD with statistically significant frequency; and Hybrid wins RR and MMIF more often than not, although not statistically significantly.

To compare all algorithms in one go across 60 experimental data sets, we use the Friedman test and the Nemenyi test following Demsar's proposal (Demsar 2006). Figure 14 illustrates three sets of representative results when $step = 200$, $step = 2{,}000$ and $step = 20{,}000$.

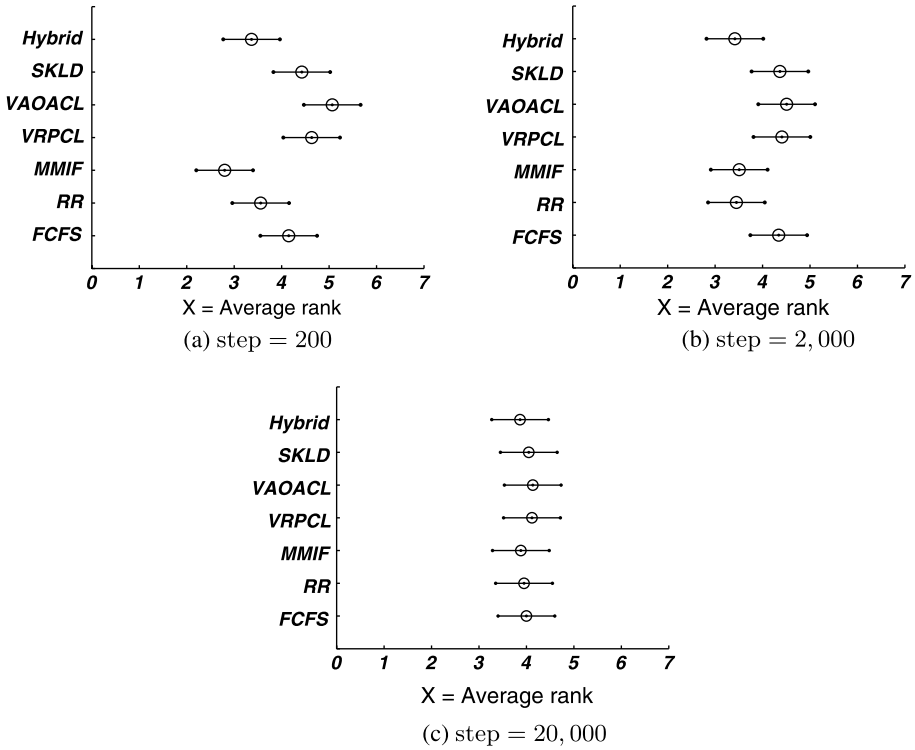


Fig. 14 Apply the Nemenyi test to alternative schemes' average ranks of root mean squared error

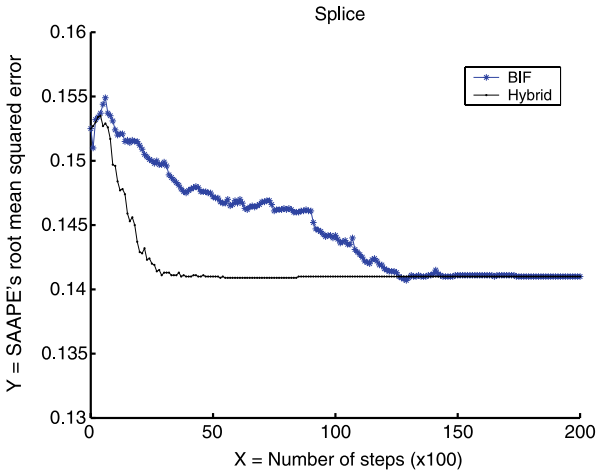
It shows that at the beginning ($step = 200$), MMIF is ranked the best and is statistically significantly better than FCFS, VRPCL, VAOACL and SKLD. Hybrid is ranked the second best and is statistically significantly better than VRPCL and VAOACL. Later on ($step = 2{,}000$), Hybrid overtakes MMIF and becomes the best although the performance difference is statistically insignificant if all seven schemes are considered in one go. Eventually, all schemes converge to the same performance.

5.2.3 MMIF versus Hybrid

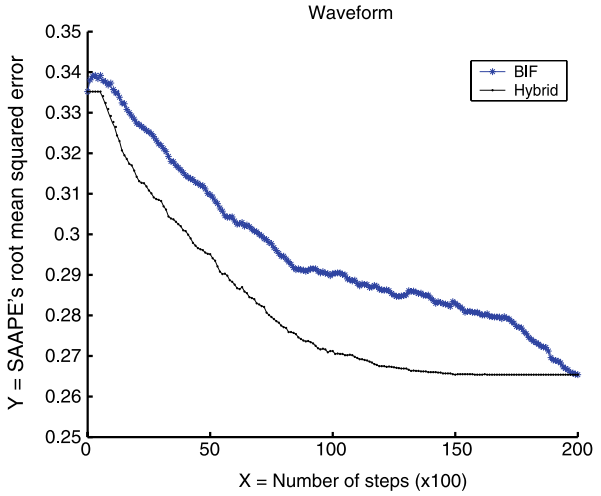
Because the empirical evidence has suggested that Hybrid and MMIF are the best two scheduling methods in general, we conduct a further comparison between the two top winners.

As we have reasoned in Sect. 3.3, a potential pitfall for MMIF is when there exist many instances like I_1 in Table 2, which retains the minimum margin that does not change regardless of additional classifiers being applied to I_1. In this case, MMIF will futilely keep dispatching SPODEs to I_1 and waste much time on it. This problem is similar to the medical situation where multiple doctors agree that a patient diagnosis is not a clear cut. In such a case, it might not help much to consult more doctors. Instead one might need to add more informing attributes, such as a new blood test.

When it is not possible to bring new attributes into the data, as is often the case, Hybrid offers another way out. We expect Hybrid to circumvent the pitfall because for each instance, not only does Hybrid consider the difference between the highest probability estimate $prob_1$



(a) Splice



(b) Waveform

Fig. 15 MMIF is less effective than Hybrid at lowering error for data sets that have many instances like I_1

and the second highest probability estimate $prob_2$, it also takes into account the deviation when SAAPE derives $prob_1$ and $prob_2$ from each invoked classifier's estimate.

The Waveform and Splice data sets in our experimental data suite are characterized with the above-mentioned problem. For example, the Waveform data set offers 40 candidate SPODEs. The 340th instance is prioritized as the minimum-margin instance for 38 continuous times out of 40. Altogether Waveform has about 180 such instances and Splice has about 120. Figure 15 illustrates the anytime classification curves of MMIF versus Hybrid regarding SAAPE's root mean squared error on the two data sets respectively. It is observed that Hybrid is more effective at lowering error than MMIF in both cases. The zero-one loss classification error has the same story.

Nonetheless, we suggest that this type of pitfalls does not occur very often. Among the comprehensive set of 60 data sets that we have studied, only two data sets (Waveform and Splice) have a non-negligible number of instances like I_1. Since MMIF also has a strong ad-

vantage in terms of efficiency, we suggest that in general MMIF is still an effective scheduling method.

6 Conclusion and future work

We have proposed an anytime classification system, Scheduling Anytime Averaged Probabilistic Estimators (SAAPE), which is capable of classifying a pool of instances, delivering accurate results whenever interrupted and optimizing the overall classification performance regarding the whole pool. When multiple instances simultaneously require classification, a choice has to be made which instance to get available computational resources next. We have proposed seven alternative scheduling strategies that calculate the priority of each instance: first-come first-served (FCFS); round robin (RR); minimum-margin instance first (MMIF); controversial instance first that comprises variance regarding predicted class label (VRPCL), variance averaged on all class labels (VAOACL), and symmetric Kullback and Leibler distance (SKLD); and lastly, the hybrid approach (Hybrid). Based on our large-scale experiments that utilize an extensive suite of 60 benchmark data sets and a wide variety of statistical tests, we draw the following conclusions.

1. In general, alternative scheduling schemes' relative performances on zero-one loss classification and on probability estimation present a similar trend.
2. Overall Hybrid is the most effective at using extra classification time to reduce zero-one loss classification error and probability estimation error. However, it loses to MMIF more often than not at the very beginning of anytime classification. We suggest the reason is that Hybrid need first allocate every test instance a SPODE in addition to NB in order to calculate μ and σ^2 for the Gaussian curve. This has an effect similar to RR (round robin) and is less effective than MMIF that can keep adding SPODEs to refine the class probability estimates for the minimum-margin instance(s) so long as every test instance has been assigned its initial NB.
3. Overall MMIF is the second most effective at using extra classification time to reduce zero-one loss classification error and probability estimation error. Considering that its performance is close to that of Hybrid, and it is more efficient than Hybrid at calculating each instance's priority (the simple subtract $prob_1$–$prob_2$ in contrast to the overlapping area of two Gaussian curves), it is the method of choice if one needs both high accuracy and high efficiency.
4. RR outperforms FCFS. This is because very often the biggest improvements for classifying an instance take place when adding the first few SPODEs. Hence when classifying a pool of instances under time resource constraints, it is more effective to dispatch the first SPODE to every instance, then the second SPODE to every instance and so on (RR), rather than to dispatch all the SPODEs to the first instance, then all the SPODEs to the second instance and so on (FCFS).
5. FCFS only presents mediocre performance. Since FCFS equals to applying the previous AAPE system to classify a pool of instances under time resource constraints, this observation suggests that SAAPE is more capable of handling anytime classification for multiple instances when coupled with appropriate scheduling schemes such as Hybrid, MMIF or RR. This is an important improvement because in reality it often happens that multiple instances simultaneously require classification such as in the credit card transaction approval application.

6. The three versions of controversial instance first scheduling, VRPCL, VAOACL and SKLD, are the least effective at reducing classification error rate as well as reducing probability estimation error. They do not have a big performance difference among themselves although VAOACL often gets the worst ranking. Hence we do not recommend to deploy them in the SAAPE system.

There are still various interesting issues to further investigate in our research on anytime classification for multiple instances. We name a few as follows.

- In our current study SPODEs are invoked by the natural order of the superparent attributes that is inherent in the data set. We focus on comparing different scheduling schemes under this natural order. However, one might suspect that the available SPODEs be of varying quality. It will be desirable to invoke the most effective SPODE first. Meanwhile, different scheduling schemes might require different optimal SPODEs ordering. How to order SPODEs under alternative scheduling schemes is a research topic for further investigation.
- Although we have used Bayesian probabilistic classifiers (NB and SPODEs) as a vehicle of illustration throughout this paper, the proposed anytime classification framework is a generic framework that can also accommodate other types of classifiers such as TAN, decision trees and SVM. Any ensemble learner could be converted into an anytime classifier by evaluating at classification time only as many of the available ensemble members as time allows. It would be very interesting to investigate alternative classifiers' efficacy and efficiency for anytime classification, and study which scheduling methods are optimal for those alternatives.
- Our future research will also explore the benefit of user interaction for anytime classification systems. In such an interaction scenario, when a user intends to terminate the classification process, the system can inform the user such as "If given this much more time, I could further improve the classification accuracy by this much". This type of information can be very useful in practice because if the improvement in accuracy is substantial while the extra time demand is trivial, the user may well choose to interrupt a little later in order to gain much better classification performance.

Acknowledgements We gratefully acknowledge Dr. Zhen He, Dr. Lloyd Allison and Dr. Enes Makalic for their thoughtful and constructive comments on this paper. This research was supported by Australian Research Council Grants DP0556279 and DP0770741.

Appendix A: Relationship between VRPCL and VAOACL

In the two-class case, variance regarding predicted class label is equivalent to variance averaged on all class labels, whereas in the multiple-class case, they can be different. We first prove the equivalence case and then give an example to attest the difference case.

Theorem 1 (Equivalence) *In the two-class case, variance regarding predicted class label is equivalent to variance averaged on all class labels.*

Proof Assume there are two possible class labels y_1 and y_2. Assume that l classifiers have been invoked to classify an instance I. Assume that the ith classifier estimates the probability of y_1 given I is a_i. Because there are only two possible class labels, the ith classifier estimates the probability of y_2 given I as $(1 - a_i)$. Without losing generality, assume that y_1 is the class label predicted by SAAPE for I.

Table 6 If there are more than two classes, the two scheduling schemes VRPCL and VAOACL can be different

			Classifier$_1$	Classifier$_2$	SAAPE output
For instance I_4	Class label	y_1	80%	60%	70%
		y_2	10%	20%	15%
		y_3	10%	20%	15%
For instance I_5	Class label	y_1	80%	60%	70%
		y_2	20%	0%	10%
		y_3	0%	40%	20%

By the definition of variance, the variance regarding the predicted class label y_1 is:

$$Var(y_1) = \sum_{i=1}^{l} \left(a_i - \frac{\sum_{i=1}^{l} a_i}{l} \right)^2, \tag{5}$$

where the term $\frac{\sum_{i=1}^{l} a_i}{l}$ is the mean value of all classifiers' probability estimates of y_1 given I.

Likewise, the variance regarding the other class label y_2 is:

$$\begin{aligned} Var(y_2) &= \sum_{i=1}^{l} \left((1 - a_i) - \frac{\sum_{i=1}^{l} (1 - a_i)}{l} \right)^2 \\ &= \sum_{i=1}^{l} \left((1 - a_i) - \left(1 - \frac{\sum_{i=1}^{l} a_i}{l} \right) \right)^2 \\ &= \sum_{i=1}^{l} \left(\frac{\sum_{i=1}^{l} a_i}{l} - a_i \right)^2. \end{aligned} \tag{6}$$

According to Eqs. 5 and 6, we get

$$Var(y_1) = Var(y_2).$$

As a result

$$average\big(Var(y_1), Var(y_2)\big) = Var(y_1). \qquad \square \tag{7}$$

The above theorem and proof assure us that the two scheduling schemes VRPCL and VAOACL are identical for two-class classification applications.

Nonetheless, if there are more than two classes, VRPCL and VAOACL can assign different priorities to an instance. One example is given in Table 6. For I_4, the predicted class is y_1 and its variance equals $(0.8 - 0.7)^2 + (0.6 - 0.7)^2 = 0.02$. I_4's variance averaged across all class labels equals to

$$\frac{(0.8-0.7)^2+(0.6-0.7)^2+(0.1-0.15)^2+(0.2-0.15)^2+(0.1-0.15)^2+(0.2-0.15)^2}{3} = 0.03.$$

For I_5, the predicted class is y_1 and its variance equals $(0.8-0.7)^2+(0.6-0.7)^2=0.02$. I_5's variance averaged across all class labels equals to

$$\frac{(0.8-0.7)^2+(0.6-0.7)^2+(0.2-0.1)^2+(0-0.1)^2+(0-0.2)^2+(0.4-0.2)^2}{3}=0.12.$$

As a result, under the scheduling scheme VRPCL, I_4 and I_5 obtain the same highest priority to get the next SPODE. In contrast, under the scheduling scheme VAOACL, I_5 attains the higher priority because $0.12 > 0.03$.

Appendix B: Pseudo codes of the SAAPE system

The pseudo codes of the SAAPE system and every scheduling algorithm are presented here. To be succinct, if an algorithm repeats some operations that have taken placed and thus have been explained in previous algorithms, we will skip the redundant explanation.

Table 7 The parameters

Explanation: Table 7 lists the parameters of our algorithms. In particular, an attribute value's frequency must be equal to or more than *m_Limit* in the training data in order to be qualified as a superparent. In our experiments *m_Limit* is set as 1 following AAPE's practice. The maximum number of qualified superparents equals the number of attributes m. We use *available_steps* to simulate the time variable. One step equals to the amount of time that allows to apply one SPODE to one instance. The maximum meaningful number of steps for a test data set $\mathcal{D}$ with n instances and m attributes equals $n \times m$. If the SAAPE system has time available more than $n \times m$ steps when classifying D, it will finish applying every candidate SPODE to every instance and its performance will remain constant afterwards.

`Algorithm: The parameters of SAAPE`

`Input:`

- $\mathcal{D}[I_1..I_n]$: test data set that comprises n instances to be classified. Each instance I_i has m attributes;
- $\mathcal{C}[c_1..c_k]$: class attribute with k class labels;
- **attvals**$[att_1..att_m]$: stores attribute values;
- *m_Limite*: the lower threshold of an attribute value's frequency in training data in order to qualify for being a superparent;
- $PP[\mathcal{C}]$: prior probability of each class label estimated from the training data;
- $PP[\mathcal{C}$, **attvals**$]$: prior probability of each class label and attribute-value pair estimated from the training data;
- $CP[\mathcal{C}$, **attvals**$]$: conditional probability of each attribute value given each class label estimated from the training data;
- $CP[\mathcal{C}$, **parent**, **attvals**$]$: conditional probability of each attribute value (the final index) given each class and each super-parent estimated from the training data;
- *available_steps*: simulates the time variable. One step equals the amount of time that allows to apply one SPODE to one instance;

`Output:`

- $P[\mathcal{D}, \mathcal{C}]$: the post probability of each class label given each instance in $\mathcal{D}$.

Table 8 The SAAPE algorithm

Explanation: Table 8 presents the framework of the SAAPE system. Same as the AAPE algorithm, Lines 1 to 10 compute NB. Interrupts are only enabled on Line 11 because we assume that the contract portion of the classification time budget is sufficient to allow standard NB classification to be performed. Line 6 stores the probability of each class label estimated by NB in the *NP* array because some scheduling algorithms (in Sect. 3.4) need this information later. Line 12 starts simulating the situation when there is still classification time available. Lines 13 to 21 initialize two arrays for every scheduling function: *superparentCount* and *retainParent*. The one-dimensional array *superparentCount*[$\mathcal{D}$] records how many SPODEs have been invoked for each instance in addition to NB, across all of which the class probability estimates given this instance will be averaged. The two-dimensional array *retainParent*[$\mathcal{D}$, **attvals**] is initialized to include every qualified superparent. For each test instance I in $\mathcal{D}$ a superparent will be removed from *retainParent*[I] once its corresponding SPODE has been dispatched to I by a scheduling function. Depending on which scheduling scheme (Line 22) is used, different instances get possibly different SPODEs to update their predicted class probabilities. The update is performed by calling the function *update_prob*() defined by Lines 24 to 31 where $P[\mathcal{C}]$ is the existent class probability estimates given I, x_p is the superparent whose SPODE is newly dispatched to I, $TP[\mathcal{C}]$ is the class probabilities estimated by x_p and *count* is the number of SPODEs I has received so far (including x_p). Interrupts are suspended during this process as the P values are in an unstable state. In consequence, if the time budget expires during this process there will be a slight delay before computation terminates.

```
Algorithm: SAAPE

 1: for num := I_1..I_n do
 2:     for y := c_1..c_k do
 3:         P[num, y] := PP[num, y];
 4:         for i := 1..m do
 5:             P[num, y] := P[num, y] × CP[y, x_i];
 6:             NP[num, y] := P[num, y];
 7:         end for
 8:     end for
 9: end for
10: normalize(P);
11: on interrupt goto Line 23;
12: step := 0;
13: for num := I_1 ... I_n do
14:     superparentCount[num] := 1;  /* Already has NB computed. */
15:     retainParent[num] := ∅;
16:     for sp := att_1 ... att_m do
17:         if sp's value for I_num has frequency no less than m_Limit in the training data then
18:             add sp into retainParent[num];
19:         end if
20:     end for
21: end for

22: call scheduling function;

23: return P;

24: update_prob ( P[C], x_p, TP[C], appliedClassifierCount ){
25: suspend interrupts;
26: for y := c_1..c_k do
27:     P[y] := (P[y]×(appliedClassifierCount−1)+TP[y]) / appliedClassifierCount;
28: end for
29: restore interrupts;
30: return P[c_1..c_k];
31: }
```

Table 9 First-come first-served scheduling

Explanation: Table 9 presents the first-come first-served algorithm. Line 7 adds one step which means SAAPE dispatches a new SPODE to an instance I_{num}, and Line 8 increments *superparentCount*[*num*] by one accordingly. Lines 9 to 14 compute *TP*, the temporary class probabilities given I_{num} that are estimated by this new SPODE. Line 15 calls the function *update_prob*( ) to update the average probability estimates of each class label given I_{num}, accounting for this newly applied SPODE.

```
Algorithm: First-come first-served scheduling
```

```
 1: if scheduling = First-come first-served then
 2:     for num := I_1..I_n do
 3:         for each x_p in retainParent[num] do
 4:             if step ≥ available_steps then
 5:                 return P; /* Classification time resources have been exhausted. */
 6:             end if
 7:             step++;
 8:             superparentCount[num]++;
 9:             for y := c_1..c_k do
10:                 TP[num, x_p, y] := PP[y, x_p];
11:                 for i := 1..m do
12:                     TP[num, x_p, y] := TP[num, x_p, y] × CP[y, x_p, x_i];
13:                 end for
14:             end for
15:             P[num, C] = update_prob(P[num, C], x_p, TP[num, x_p, C], superparentCount[num] + 1);
                /*(superparentCount[num] + 1) counts invoked SPODEs and NB. */
16:             remove x_p from retainParent[num];
17:         end for
18:     end for
19:     normalize(P);
20: end if
```

Table 10 Round robin scheduling

Explanation: Table 10 represents the round robin algorithm. Line 3 decides which instance gets a new SPODE. The % is the modular operator and n is the number of test instances in $\mathcal{D}$. Line 5 selects a superparent x_p for the instance I_{num} from the *retainParent*[*num*] array. Line 6 removes x_p from *retainParent*[*num*] since it has been scheduled to dispatch, and accordingly Line 7 increments by 1 the number of classifiers applied to I_{num}. Lines 8 to 13 compute the temporary class probabilities given I_{num} that are estimated by the SPODE whose superparent is x_p.

`Algorithm: Round robin scheduling`

1: **if** scheduling = Round robin **then**
2: **while** $step < available\_steps$ && $retainParent \neq \emptyset$ **do**
3: $num := step \ \% \ n$;
4: $step++$;
5: select x_p in $retainParent[num]$;
6: remove x_p from $retainParent[num]$;
7: $superparentCount[num]++$;
8: **for** $y := c_1..c_k$ **do**
9: $TP[num, x_p, y] := PP[y, x_p]$;
10: **for** $i := 1..m$ **do**
11: $TP[num, x_p, y] := TP[num, x_p, y] \times CP[y, x_p, x_i]$;
12: **end for**
13: **end for**
14: $P[num, \mathcal{C}] = update\_prob(P[num, \mathcal{C}], x_p, TP[num, x_p, \mathcal{C}], superparentCount[num] + 1)$;
15: **end while**
16: $normalize(P)$;
17: **end if**

Table 11 Borderline instance first scheduling

Explanation: Table 11 presents the borderline instance first scheduling algorithm. Line 2 copies the test data set $\mathcal{D}$ to $\mathcal{D}_{tmp}$. Lines 4 to 10 calculate for each instance ($prob_1$–$prob_2$) where $prob_1$ is the highest predicted class probability and $prob_2$ is the second highest predicted class probability. Line 11 chooses the instance I_{num} which has the smallest value of ($prob_1$–$prob_2$) and thus which gets a new SPODE. If the instance I_{num} has no more SPODEs to be invoked, we remove it from $\mathcal{D}_{tmp}$ (Lines 23 to 25).

`Algorithm: Borderline instance first scheduling`

```
 1: if scheduling = Borderline instance first then
 2:     D_tmp := D;
 3:     while step < available_steps && D_tmp ≠ ∅ do
 4:         for num := I_1..I_n do
 5:             if I_num in D_tmp then
 6:                 prob_1 := highest predicted class probability value;
 7:                 prob_2 := second highest predicted class probability value;
 8:                 priority[num] := prob_1 − prob_2;
 9:             end if
10:         end for
11:         num := arg min_num(priority[I_num]);
12:         select x_p from retainParent[num];
13:         remove x_p from retainParent[num];
14:         step++;
15:         superparentCount[num]++;
16:         for y := c_1..c_k do
17:             TP[num, x_p, y] := PP[y, x_p];
18:             for i := 1..m do
19:                 TP[num, x_p, y] := TP[num, x_p, y] × CP[y, x_p, x_i]
20:             end for
21:         end for
22:         P[num, C] = update_prob(P[num, C], x_p, TP[num, x_p, C], superparentCount[num] + 1);
23:         if retainParent[num] is null then
24:             remove I_num from D_tmp
25:         end if
26:     end while
27:     normalize(P);
28: end if
```

Table 12 The function to calculate variance among invoked classifiers' opinions regarding a class label given an instance

Explanation: Table 12 presents the function $Var(c_i, num)$ that calculates the variance among invoked classifiers' opinions regarding a class label c_i given an instance I_{num}. This function will be called by two scheduling algorithm: variance regarding predicted class label (Sect. 3.4.1) and variance averaged on all class labels (Sect. 3.4.2). $NP[num, c_i]$ records the probability of c_i given I_{num} estimated by NB, and $TP[num, x_p, c_i]$ records the probability of c_i given I_{num} estimated by the SPODE whose superparent is x_p.

`Function:` $Var(c_i, num)$

1: $Var(c_i, num)\{$
2: $\mu_i := NP[num, c_i]$;
3: **for** $x_p := 0..(superparentCount[num] - 1)$ **do**
4: $\quad \mu_i := \mu_i + TP[num, x_p, c_i]$;
5: **end for**
6: $\mu_i := \frac{\mu_i}{superparentCount[num]+1}$;
7: $\sigma_i := (NP[num, c_i] - \mu_i)^2$;
8: **for** $x_p := 0..(superparentCount[num] - 1)$ **do**
9: $\quad \sigma_i := \sigma_i + (TP[num, x_p, c_i] - \mu_i)^2$;
10: **end for**
11: $\sigma_i := \frac{\sigma_i}{superparentCount[num]+1}$;
12: return $\sqrt{\sigma_i}$;
13: $\}$

Table 13 Controversial instance first (variance regarding predicted class label) scheduling

Explanation: Table 13 presents the scheduling algorithm of controversial instance first (variance regarding predicted class label). Lines 2 to 6 initialize the parameters. Line 3 sets a flag in order to make sure that each instance has been estimated by one SPODE in addition to NB before starting calculating variance (Lines 8 and 9). This is necessary for getting σ_i for the class label c_i. In Line 5, the array $variance[\mathcal{D}]$ is used to record the priority (variance) for each instance and is initialized by 0. Line 7 keeps running until the classification time resource has exhausted or every instance has obtained all qualified SPODEs. Line 11 picks out the instance I_{num} to get a new SPODE. Line 24 identifies the predicted class label *predictedClass* for I_{num} and Line 25 updates variance among invoked classifiers' opinions regarding *predictedClass* given I_{num}.

`Algorithm: Controversial instance first (variance regarding predicted class label) scheduling`

1: **if** *scheduling* = *Controversial* instance first (variance regarding predicted class label) **then**
2: $\mathcal{D}_{tmp} := \mathcal{D}$;
3: $flag := 0$;
4: **for** $j := I_1..I_n$ **do**
5: $variance[j] := 0$;
6: **end for**
7: **while** $step < available\_steps$ && $\mathcal{D}_{tmp} \neq \emptyset$ **do**
8: **if** $flag < n$ **then**
9: $num := flag++$;
10: **else**
11: $num := \arg\max_j (variance[I_j])$;
12: **end if**
13: $step++$;
14: $superparentCount[num]++$;
15: select x_p from $retainParent[num]$;
16: remove x_p from $retainParent[num]$;
17: **for** $y := c_1..c_k$ **do**
18: $TP[num, x_p, y] := PP[y, x_p]$
19: **for** $i := 1..m$ **do**
20: $TP[num, x_p, y] := TP[num, x_p, y] \times CP[y, x_p, x_i]$
21: **end for**
22: **end for**
23: $P[num, \mathcal{C}] = update\_prob(P[num, \mathcal{C}], x_p, TP[num, x_p, y], superparentCount[num])$;
24: $predictedClass := \arg\max_i (P[num, c_i])$;
25: $variance[num] := Var(predictedClass, num)$;
26: **if** $retainParent[num]$ *is null* **then**
27: remove I_{num} from $\mathcal{D}_{tmp}$
28: **end if**
29: **end while**
30: $normalize(P)$;
31: **end if**

Table 14 Controversial instance first (variance averaged on all class labels) scheduling

Explanation: Table 14 presents the scheduling algorithm of controversial instance first (variance averaged on all class labels). The key difference between this algorithm and the previous one in Table 13 is that it calculates the variance regarding each class and then averages across them to get the priority of each instance (Lines 25 to 27).

```
Algorithm: Controversial instance first (variance averaged on all class
labels) scheduling
```

```
 1: if scheduling = Controversial instance first (variance averaged on all class labels) then
 2:     D_tmp := D;
 3:     flag := 0;
 4:     for j := I_1..I_n do
 5:         variance[j] := 0;
 6:     end for
 7:     while step < available_steps && D_tmp ≠ ∅ do
 8:         if flag < n then
 9:             num := flag++;
10:         else
11:             num := arg max_j (variance[I_j]);
12:         end if
13:         step++;
14:         superparentCount[num]++;
15:         select x_p from retainParent[num];
16:         remove x_p from retainParent[num];
17:         for y := c_1..c_k do
18:             TP[num, x_p, y] := PP[y, x_p]
19:             for i := 1..m do
20:                 TP[num, x_p, y] := TP[num, x_p, y] × CP[y, x_p, x_i]
21:             end for
22:         end for
23:         P[num, C] = update_prob(P[num, C], x_p, TP[num, x_p, C], superparentCount[num] + 1);
24:         sum := 0.0;
25:         for y := c_1..c_k do
26:             sum := sum + Var(y, num);
27:         end for
28:         variance[num] := sum/k;
29:         if retainParent[num] is null then
30:             remove I_num from D_tmp
31:         end if
32:     end while
33: end if
```

Table 15 The function of calculating symmetric Kullback and Leibler distance averaged across all classifier pairs

Explanation: Table 15 implements the function calculating the symmetric Kullback and Leibler distance (SKL) averaged across all pairs of invoked classifiers regarding an instance I_{num}. Lines 5 to 9 compute the SKL distance between NB and each invoked SPODE. Lines 4 to 17 compute the SKL distance between each pair of invoked SPODEs. Line 18 returns the average value across all computed SKL distances.

```
Function: SKL(num)
```

1: $SKL(num)\{$
2: $sum := 0.0;$
3: $count := 0;$
4: **for** $sp := 0..(superparentCount[num] - 1)$ **do**
5: **for** $y := c_1..c_k$ **do**
6: $sum := sum + NP[num, y] \times \log \frac{NP[num,y]}{TP[num,sp,y]};$
7: $sum := sum + TP[num, sp, y] \times \log \frac{TP[num,sp,y]}{NP[num,y]};$
8: $count++;$
9: **end for**
10: **for** $sq := (sp + 1)..(superparentCount[num] - 1)$ **do**
11: **for** $y := c_1..c_k$ **do**
12: $sum := sum + TP[num, sp, y] \times \log \frac{TP[num,sp,y]}{TP[num,sq,y]};$
13: $sum := sum + TP[num, sq, y] \times \log \frac{TP[num,sq,y]}{TP[num,sp,y]};$
14: $count++;$
15: **end for**
16: **end for**
17: **end for**
18: return $\frac{sum}{count};$
19: $\}$

Table 16 Controversial instance first (symmetric Kullback and Leibler distance) scheduling

Explanation: Table 16 presents the scheduling algorithm of controversial instance first (symmetric Kullback and Leibler distance). Lines 8 to 12 select the instance I_{num} who has the highest averaged SKL distance across all pairs of invoked classifiers and thus who gets allocated a new SPODE. Line 24 updates I_{num}'s SKL distance by calling the function $SKL(num)$.

```
Algorithm: Controversial instance first (symmetric Kullback and Leibler
distance) scheduling
```

```
 1: if scheduling = Controversial instance first (symmetric Kullback and Leibler distance) then
 2:     D_tmp := D;
 3:     flag := 0;
 4:     for j := I_1..I_n do
 5:         variance[j] := 0;
 6:     end for
 7:     while step < available_steps && D_tmp ≠ ∅ do
 8:         if flag < n then
 9:             num := flag++;
10:         else
11:             num := arg max_j (variance[I_j]);
12:         end if
13:         select x_p from retainParent[num];
14:         remove x_p from retainParent[num];
15:         step++;
16:         superparentCount[num]++;
17:         for y := c_1..c_k do
18:             TP[num, x_p, y] := PP[y, x_p]
19:             for i := 1..m do
20:                 TP[num, x_p, y] := TP[num, x_p, y] × CP[y, x_p, x_i]
21:             end for
22:         end for
23:         P[num, C] = update_prob(P[num, C], x_p, TP[num, x_p, C], superparentCount[num] + 1);
24:         variance[num] := SKL(num);
25:         if retainParent[num] is null then
26:             remove I_num from D_tmp;
27:         end if
28:     end while
29: end if
```

Table 17 A hybrid approach for scheduling

Explanation: Table 17 presents the algorithm of the hybrid approach. Line 2 copies $\mathcal{D}$ to two temporary data sets $\mathcal{D}_{tmp}$ and $\mathcal{TD}$. Because calculating σ of the Gaussian distribution requires at least two values, we need first allocate every test instance a SPODE in addition to NB. In this particular round of allocating the first SPODE, the instance that gets the smallest value of $prob_1 - prob_2$ will get the SPODE first, which is different from the round robin scheduling and is implemented by Lines 8 to 16. $\mathcal{TD}$ registers instances which still need add the first SPODE. When $\mathcal{TD}$ becomes empty, it means every instance has got its first SPODE. Given an instance I_{num} which has obtained multiple classifiers, Lines 31 and 32 identify its class labels of the highest and second highest predicted probability among all labels. We only need to calculate the overlapping area of these two class labels' Gaussian distributions (Lines 33 to 35). Line 18 identifies the instance which has the highest priority (overlapping area) and thus which gets a new SPODE by calling the function *get_Instance*().

`Algorithm: A hybrid approach for scheduling`

```
 1: if scheduling = A hybrid approach then
 2:     D_tmp := D; TD := D;
 3:     flag := 0;
 4:     for j := I_1..I_n do
 5:         variance[j] := 0;
 6:     end for
 7:     while step < available_steps && D_tmp ≠ ∅ do
 8:         if flag < n then
 9:             for num ∈ TD do
10:                 prob_1 := highest predicted class probability value;
11:                 prob_2 := second highest predicted class probability value;
12:                 priority[num] := prob_1 − prob_2;
13:             end for
14:             num := arg min_num(priority[num]);
15:             remove num from TD;
16:             flag + +;
17:         else
18:             num := get_Instance( );
19:         end if
20:         select x_p from retainParent[num];
21:         remove x_p from retainParent[num];
22:         step + +;
23:         superparentCount[num] + +;
24:         for y := c_1..c_k do
25:             TP[num, x_p, y] := PP[y, x_p];
26:             for i := 1..m do
27:                 TP[num, x_p, y] := TP[num, x_p, y] × CP[y, x_p, x_i];
28:             end for
29:         end for
30:         P[num, C] = update_prob(P[num, C], x_p, TP[num, x_p, C], superparentCount[num] + 1);
31:         y1 := arg max_y (P[num, C]); /* Class with highest predicted probability.*/
32:         y2 := arg max_y (P[num, C] − P[num, y1]); /* Class with second highest predicted probability.*/
33:         CDF_1 = Gaussian(y1);
34:         CDF_2 = Gaussian(y2);
35:         variance[num] := overlap(CDF_1, CDF_2);
36:         if retainParent[num] is null then
37:             remove I_num from D_tmp;
38:         end if
39:     end while
40: end if
```

Table 18 The function of *get_Instance*

Explanation: Table 18 presents the *get_Instance*( ) function that is called by the hybrid approach in Table 17. Lines 7 to 11 identify the instance which has the highest priority (overlapping area) and thus which gets a new SPODE. This function also implements a tie breaker strategy from Lines 12 to 17. If two instances attain the same highest priority, we use their values of $(prob_1 - prob_2)$ as the tie breaker. Whichever instance has the smallest value gets a new SPODE. This is inspired by our observation of the very good empirical performance of the borderline instance first scheduling scheme. Note that $prob_1$ and $prob_2$ equal to μ_1 and μ_2 respectively.

`Function:` *get_Instance*( )

```
 1: get_Instance( ){
 2: tmpPriority := 0.0;
 3: diff := 0.0;
 4: for i := 1..n do
 5:     prob1 := highest predicted class probability value of instance Ii;
 6:     prob2 := second highest predicted class probability value of instance Ii;
 7:     if variance[i] > tmpPriority then
 8:         tmp := i;
 9:         tmpPriority := variance[i];
10:         diff := prob1 − prob2;
11:     end if
12:     if variance[i] = tmpPriority then
13:         if (prob1 − prob2) < diff then
14:             tmp := i;
15:             diff := prob1 − prob2;
16:         end if
17:     end if
18: end for
19: return tmp;
20: }
```

References

Asuncion, A., & Newman, D. J. (2007). *UCI machine learning repository*. University of California, Irvine, School of Information and Computer Sciences.

Baeza-Yates, R., & Ribeiro-Neto, B. (1999). *Modern information retrieval*. Reading/Harlow: Addison-Wesley/Longman.

Bartlett, P. L. (2007). *Pattern classification and large margin classifiers*. Machine Learning Summer School 2006, Taipei.

Bekara, M., Knockaert, L., Seghouane, A.-K., & Fleury, G. (2006). A model selection approach to signal denoising using Kullback's symmetric divergence. *Signal Processing*, *86*, 1400–1409.

Chan, P., Fan, W., Prodromidis, A., & Stolfo, S. (1999). Distributed data mining in credit card fraud detection. *IEEE Intelligent Systems*, *14*(6), 67–74.

DeCoste, D. (2002). Anytime interval-valued outputs for kernel machines: Fast support vector machine classification via distance geometry. In *Proceedings of the 19th international conference on machine learning* (pp. 99–106).

Deitel, H. M., Deitel, P. J., & Choffnes, D. R. (2004). *Operating systems* (3rd ed.). New York: Prentice Hall.

Demsar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, *7*, 1–30.

Duarte, M. F., & Hu, Y. H. (2004). Vehicle classification in distributed sensor networks. *Journal of Parallel and Distributed Computing*, *64*(7), 826–838.

Friedman, M. (1937). The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association*, *32*, 675–701.

Friedman, M. (1940). A comparison of alternative tests of significance for the problem of m rankings. *Annals of Mathematical Statistics*, *11*, 86–92.

Grass, J., & Zilberstein, S. (1996). Anytime algorithm development tools. M. Pittarelli (Ed.). *SIGART Bulletin Special Issue on Anytime Algorithms and Deliberation Scheduling*, *7*(2), 20–27.

Jeffreys, H. (1946). An invariant form for the prior probability in estimation problems. *Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences*, *186*, 453–461.

Jitnah, N., & Nicholson, A. (1997). treeNets: A framework for anytime evaluation of belief networks. In *Proceedings of the international joint conference on qualitative and quantitative practical reasoning* (pp. 350–364).

Keogh, E. J., & Pazzani, M. J. (2002). Learning the structure of augmented Bayesian classifiers. *International Journal on Artificial Intelligence Tools*, *11*(40), 587–601.

Kohavi, R. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the 14th international joint conference on artificial intelligence* (pp. 1137–1145).

Kohavi, R., & Provost, F. (1998). Glossary of terms, special issue on applications of machine learning and the knowledge discovery process. *Machine Learning*, *30*, 271–274.

Kullback, S., & Leibler, R. A. (1951). On information and sufficiency. *Annals of Mathematical Statistics*, *22*(1), 79–86.

Langley, P., Iba, W., & Thompson, K. (1992). An analysis of Bayesian classifiers. In *Proceedings of the 10th national conference on artificial intelligence* (pp. 223–228).

Liu, C.-L., & Wellman, M. P. (1996). On state-space abstraction for anytime evaluation of Bayesian networks. *ACM SIGART Bulletin*, *7*(2), 50–57.

Meesookho, C., Narayanan, S., & Raghavendra, C. S. (2002). Collaborative classification applications in sensor networks. In *Proceedings of the sensor array and multichannel signal processing workshop* (pp. 370–374).

Resnick, P., & Varian, H. R. (1997). Recommender systems. *Communications of the ACM*, *40*(3), 56–58.

Schapire, R., Freund, Y., Bartlett, P., & Lee, W. (1998). Boosting the margin: A new explanation for the effectiveness of voting methods. *Annals of Statistics*, *26*(5), 1651–1686.

Stallings, W. (2004). *Operating systems: Internals and design principles* (5th ed.). New York: Prentice Hall.

Sun, Y., & Daigle, J. N. (2006). A PCA-based vehicle classification system in wireless sensor networks. In *Proceedings of the wireless communications and networking conference* (Vol. 4, pp. 2193–2198).

Tanenbaum, A. S. (2001). *Modern operating systems* (2nd ed.). New York: Prentice Hall.

Ueno, K., Xi, X., Keogh, E., & Lee, D.-J. (2006). Anytime classification using the nearest neighbor algorithm with applications to stream mining. In *Proceedings of the 6th international conference on data mining* (pp. 623–632).

Webb, G. I., Boughton, J., & Wang, Z. (2005). Not so naive Bayes: Averaged one-dependence estimators. *Machine Learning*, *58*(1), 5–24.

Webb, G. I., Pazzani, M. J., & Billsus, D. (2001). Machine learning for user modeling. *User Modeling and User-Adapted Interaction*, *11*(1–2), 19–29.

Witten, I. H., & Frank, E. (2005). *Data mining: Practical machine learning tools and techniques with java implementations* (2nd ed.). San Mateo: Morgan Kaufmann.

Yang, Y., Webb, G. I., Korb, K., & Ting, K. M. (2007). Classifying under computational resource constraints: anytime classification using probabilistic estimators. *Machine Learning*, *69*, 35–53.

Ye, L., Wang, X., Yankov, D., & Keogh, E. (2008). The asymmetric approximate anytime join: A new primitive with applications to data mining. In *Proceedings of the SIAM conference on data mining*.