Journal of Signal Processing Systems (2021) 93:1057–1084
https://doi.org/10.1007/s11265-021-01671-1

An Improved Boundary Uncertainty-Based Estimation for Classifier Evaluation

David Ha[1] · **Shigeru Katagiri**[1] · **Hideyuki Watanabe**[1] · **Miho Ohsaki**[1]

Received: 27 October 2020 / Revised: 3 May 2021 / Accepted: 5 May 2021 / Published online: 10 June 2021

Abstract

This paper proposes a new boundary uncertainty-based estimation method that has significantly higher accuracy, scalability, and applicability than our previously proposed boundary uncertainty estimation method. In our previous work, we introduced a new classifier evaluation metric that we termed "boundary uncertainty." The name "boundary uncertainty" comes from evaluating the classifier based solely on measuring the equality between class posterior probabilities along the classifier boundary; satisfaction of such equality can be described as "uncertainty" along the classifier boundary. We also introduced a method to estimate this new evaluation metric. By focusing solely on the classifier boundary to evaluate its uncertainty, boundary uncertainty defines an easier estimation target that can be accurately estimated based directly on a finite training set without using a validation set. Regardless of the dataset, boundary uncertainty is defined between 0 and 1, where 1 indicates whether probability estimation for the Bayes error is achieved. We call our previous boundary uncertainty estimation method "Proposal 1" in order to contrast it with the new method introduced in this paper, which we call "Proposal 2." Using Proposal 1, we performed successful classifier evaluation on real-world data and supported it with theoretical analysis. However, Proposal 1 suffered from accuracy, scalability, and applicability limitations owing to the difficulty of finding the location of a classifier boundary in a multidimensional sample space. The novelty of Proposal 2 is that it locally reformalizes boundary uncertainty in a single dimension that focuses on the classifier boundary. This convenient reduction with a focus toward the classifier boundary provides the new method's significant improvements. In classifier evaluation experiments on Support Vector Machines (SVM) and MultiLayer Perceptron (MLP), we demonstrate that Proposal 2 offers a competitive classifier evaluation accuracy compared to a benchmark Cross Validation (CV) method as well as much higher scalability than both CV and Proposal 1.

Keywords Pattern classification · Generalization ability · Bayes error · Model selection

1 Introduction

A fundamental problem in statistics, machine learning, and pattern classification is to obtain an accurate estimate for the generalization ability of a learning algorithm trained on a finite dataset. The generalization ability of a pattern classifier is traditionally measured in terms of classification error probability. The goal of pattern classification is to execute the optimal classification rule (aka a Bayes decision rule) that corresponds to the minimum classification error probability (aka a Bayes error). However, the estimate of the error probability based on a finite amount of training data is so seriously biased that it cannot directly indicate the error probability [1].

A major conventional approach to classification error probability is Structural Risk Minimization (SRM)[2]. SRM provides an analytic estimation of the classification error probability. However, the resulting estimation can be loose in practice. Furthermore, the difficulty of deriving the necessary SRM in a case-by-case manner hinders its application [1, 3].

Hold Out (HO) evaluation bypasses the bias intrinsic to the training data by splitting the available data into training data and validation data. Then, the empirical error rate of the validation data is used as an estimate of the error probability. However, the error probability estimate

This work was supported in part by JSPS KAKENHI No. 18H03266.

✉ David Ha
davidhafr@gmail.com

[1] Doshisha University, Kyotanabe-shi, Kyoto-Fu, Japan

obtained from a particular validation split may also be biased. To resolve this issue, Cross Validation (CV) [4] averages the error probability estimates obtained in turn from different validation sets partitioned from the data. The particular case where every sample is subsequently used as a validation set is called Leave-One-Out (LOO). LOO is known to converge to the expected error probability [5].

Bootstrap [6] reduces the variance of the error probability estimation by averaging the estimations obtained from different training sets by sampling with replacement. One drawback of the resampling approaches (CV, Bootstrap) is their costly training repetition that can be prohibitive in real-world tasks [7], along with the sacrifice of separate data for evaluation. In principle, training and evaluation on the training data to directly target the Bayes error would be preferable, although this has been difficult so far.

Moreover, error rate smoothing methods can improve error probability estimation [8] based only on the training set. For example, Minimum Classification Error training [9] optimally determines the degree of smoothing of the empirical error rate to target the Bayes error. However, the settings and effect of this automatic smoothing are ongoing research issues [10].

Likelihood methods such as information criteria [11] and Bayesian model selection [12, 13] rely on class posterior probability estimation instead of estimating the error probability. However, classification focuses the quality of the estimation on the boundaries that delineate class distributions, not on the class distributions themselves. Therefore, likelihood approaches are not necessarily optimal for classifier evaluation [14, 15].

The limitations described above come from the intrinsic difficulty of estimating the error probability. To circumvent these limitations, we proposed finding the optimally trained classifier through a new classifier evaluation metric that is uniquely easy to estimate in principle, which we termed "*boundary uncertainty*" or alternately "Bayes-boundary-ness" [16–18]. We chose the name "*boundary uncertainty*" because this evaluation metric measures the generalization ability of a classifier based on how equal class posterior probabilities are along the classifier boundary. It is known that the optimal classifier boundary is defined by equality among the class posterior probabilities, and this situation can be described as "uncertainty" along the boundary. Boundary uncertainty is fundamentally easier to estimate for two reasons: focusing the estimation on the classifier boundary instead of integrating estimations over the entire multidimensional space is easier, and the classifier boundary is defined based on classifier parameters that are precisely known. Furthermore, the value of boundary uncertainty implies how close we are to the Bayes risk.

In addition to defining boundary uncertainty, our previous work also proposed a boundary uncertainty estimation method that here we refer to as "Proposal 1." Through experiments and theoretical analysis, we showed that Proposal 1 could perform classifier evaluation using neither sample distribution assumptions nor data resampling methods [16–18]. However, Proposal 1 relied on largely unclear settings and heavy treatment that seriously limited both its accuracy and its scalability. This motivates our work on a new boundary uncertainty estimation method that we call "Proposal 2" and comprehensively introduce in this paper.

The paper is organized as follows. In Section 2, we prepare the formalization of the classification problem. In Section 3, we summarize the previous method, i.e., Proposal 1. In Section 4, we introduce Proposal 2 as a new method. Then, in Section 5, we discuss the time costs of classifier evaluation for the two methods. In Section 6, we perform experiments using these two methods and provide extensive discussion. In Section 7, we finally summarize the paper.

Table 1 presents the main notations used in this paper. The first column of the table indicates the section where each notation is first used.

2 Formalization of Classification Problem

2.1 Bayes Decision Rule

We assume a d-dimensional sample space $\mathcal{X}$, where we intend to discriminate between J classes. Given $j \in [\![1, J]\!]$ and $\boldsymbol{x} \in \mathcal{X}$, we denote by $P(C_j|\boldsymbol{x})$ the (true) class posterior probability of C_j given $\boldsymbol{x}$. The goal of a classifier is to minimize the following classification risk:

$$R = \int_{\boldsymbol{x}\in\mathcal{X}} R(C(\boldsymbol{x})|\boldsymbol{x})\, p(\boldsymbol{x}) d\boldsymbol{x}. \tag{1}$$

We can see that Eq. 1 operates over the entire multidimensional sample space. $C(\boldsymbol{x})$ represents the classification decision for sample $\boldsymbol{x}$ (decision is one of the classes C_r, where $r \in [\![1, J]\!]$). $R(C(\boldsymbol{x})|\boldsymbol{x})$ represents the risk when assigning class $C(\boldsymbol{x})$ to $\boldsymbol{x}$:

$$R(C_r|\boldsymbol{x}) = \sum_{j=1}^{J} \lambda(C_r|C_j) P(C_j|\boldsymbol{x}), \tag{2}$$

where $\lambda(C_r|C_j)$ denotes the cost of classifying a member of class C_j as a member of class C_r. We assume the following cost:

$$\lambda(C_r|C_j) = \begin{cases} 1, & \text{if } r \neq j, \\ 0, & \text{otherwise.} \end{cases} \tag{3}$$

This leads to the risk $R(C_r|\boldsymbol{x}) = 1 - P(C_r|\boldsymbol{x})$. This risk is minimized by choosing C_r such that $P(C_r|\boldsymbol{x})$ is the highest among all of the class posterior probabilities at $\boldsymbol{x}$. Based on this consideration, the decision rule that achieves the

Table 1 Summary of main notations.

Section	Notation	Description
Section 2.1	$\mathcal{X}$	Sample space, assumed to be fixed-dimensional in this paper for simplicity
Section 2.1	J	Number of classes
Section 2.1	d	Dimensionality of the sample space
Section 2.1	C_l	Class label, where class index $l \in [\![1, J]\!]$
Section 2.2	N	Number of training samples
Section 2.2	$\mathcal{T}$	Training set defined as $\{(\boldsymbol{x}_n, y_n)\}_{n\in[\![1,N]\!]}$. $\boldsymbol{x}_n \in \mathcal{X}$: training sample; $y_n \in [\![1, J]\!]$: its *given* class index
Section 2.2	Λ	Set of classifier parameter values ("classifier status")
Section 2.2	$C(\boldsymbol{x}; \Lambda)$	Given Λ, class label *predicted* by the classifier at $\boldsymbol{x}$
Section 2.2	$g_l(\boldsymbol{x}; \Lambda)$	Given Λ, discriminant function for class C_l at $\boldsymbol{x}$
Section 3.1.1	$\{i^*(\boldsymbol{x}), j^*(\boldsymbol{x})\}$	Given Λ, indexes of the two (true) class posterior probabilities whose values at $\boldsymbol{x}$ are highest
Section 3.1.1	B^*	Bayes boundary defined as the set of samples $\boldsymbol{x} \in \mathcal{X}$ that satisfy $P(C_{i^*(\boldsymbol{x})}\|\boldsymbol{x}) = P(C_{j^*(\boldsymbol{x})}\|\boldsymbol{x})$
Section 3.1.2	$\{i(\boldsymbol{x}; \Lambda), j(\boldsymbol{x}; \Lambda)\}$	Given Λ, indexes of the two discriminant functions whose values at $\boldsymbol{x}$ are highest
Section 3.1.2	$B(\Lambda)$	Given Λ, classifier boundary defined as the set of samples $\boldsymbol{x} \in \mathcal{X}$ that satisfy $g_{i(\boldsymbol{x};\Lambda)}(\boldsymbol{x}; \Lambda) = g_{j(\boldsymbol{x};\Lambda)}(\boldsymbol{x}; \Lambda)$
Section 3.1.3	$\hat{\cdot}$	Hat notation: estimated quantity, in contrast to its expected (true) value
Section 3.1.3	$\hat{U}(\Lambda)$	Given Λ, empirical boundary uncertainty
Section 3.1.3	$\hat{U}(\boldsymbol{x}; \Lambda)$	Given Λ, estimated local uncertainty at sample $\boldsymbol{x}$
Section 3.1.3	U_{min}, U_{max}	Minimum and maximum possible values of the local uncertainty function, respectively
Section 3.3	i_{max}	Imposed maximum number of dichotomy iterations when interpolating on a random segment
Section 3.3	R_A	Imposed number of anchors generated by Step 1 of Proposal 1
Section 3.4.1	$\mathcal{N}_B(\Lambda)$	Near-boundary set selected by Step 1 of Proposal 1
Section 3.4.2	$\|\cdot\|$	Cardinal operator
Section 3.4.2	$¶_\Lambda$	Set of clusters obtained by a single Tree Divisive Clustering (TDC)
Section 3.4.2	N_{min}, N_{max}	Imposed minimum and maximum number of samples for each cluster in ¶, respectively
Section 3.4.2	i_K	Imposed maximum number of iterations during a 2-means clustering
Section 3.4.2	R_T	Imposed number of TDC repetitions
Section 3.4.4	$\hat{U}^{(1)}(\boldsymbol{x}; \Lambda)$	Given Λ, estimate of empirical boundary uncertainty using Proposal 1
Section 4.2.1	$f_{kl}(\boldsymbol{x}; \Lambda)$	Given Λ, given class indexes $k, l \in [\![1, J]\!]$, near-boundary-ness measurement of sample $\boldsymbol{x}$ relative to class indexes k and l
Section 4.2.2	$\tilde{\cdot}$	Tilde notation: operation that describes a perturbation (e.g., $\widetilde{\mathcal{T}}$ is the perturbated version of $\mathcal{T}$).
Section 4.2.2	$\boldsymbol{x}^{(m)}$	m-th nearest training sample to $\boldsymbol{x}$
Section 4.3.1	$\mathcal{I}^*$	Set of couples of class indexes that compose the Bayes boundary B^*
Section 4.3.1	$B^*(k, l)$	Set of training samples $\boldsymbol{x} \in \mathcal{X}$ that satisfy $\{i^*(\boldsymbol{x}), j^*(\boldsymbol{x})\} = \{k, l\}$
Section 4.3.2	$P(C_{kl}\|\boldsymbol{x})$	Given $\{k, l\} \in \mathcal{I}^*$ and $\boldsymbol{x} \in \mathcal{X}$, probability that $\{i^*(\boldsymbol{x}), j^*(\boldsymbol{x})\} = \{k, l\}$
Section 4.3.2	$\lambda_U\,(C(\boldsymbol{x}; \Lambda)\|\, C_{kl})$	Given Λ, given $\{k, l\} \in \mathcal{I}^*$ and $\boldsymbol{x} \in \mathcal{X}$, boundary-wise cost of the classifier decision $C(\cdot; \Lambda)$ at $\boldsymbol{x}$ in regard to class indexes k, l
Section 4.3.2	$U(\Lambda)$	Given Λ, expected boundary uncertainty
Section 4.3.2	$U_{kl}(\Lambda)$	Given Λ, given classes indexes $\{k, l\} \in \mathcal{I}^*$, two-class boundary uncertainty relative to C_k, C_l
Section 4.4.1	$\mathcal{N}^{\mathrm{M}}(\boldsymbol{x})$	Set of M nearest training samples to $\boldsymbol{x}$
Section 4.4.1	$h_{\boldsymbol{x}}$	Given Λ, Parzen width estimated using Silverman's rule of thumb on the (perturbated) projected set $\{f_{i(\boldsymbol{x};\Lambda)j(\boldsymbol{x};\Lambda)}(\widetilde{x'}; \Lambda)\}_{\boldsymbol{x}'\in\mathcal{N}^{\mathrm{M}}(\boldsymbol{x})}$
Section 4.4.1	$\hat{k}(\boldsymbol{x}; \Lambda)$	Given Λ and a sample $\boldsymbol{x}$, count of how many samples in $\{f_{i(\boldsymbol{x};\Lambda)j(\boldsymbol{x};\Lambda)}(\widetilde{x'}; \Lambda)\}_{\boldsymbol{x}'\in\mathcal{N}^{\mathrm{M}}(\boldsymbol{x})}$ fall in $[-h_{\boldsymbol{x}}; h_{\boldsymbol{x}}]$
Section 4.4.1	$\hat{k}_m(\boldsymbol{x}; \Lambda)$	Given Λ and a sample $\boldsymbol{x}$, count of how many samples in $\{f_{i(\boldsymbol{x};\Lambda)j(\boldsymbol{x};\Lambda)}(\widetilde{x'}; \Lambda)\}_{\boldsymbol{x}'\in\mathcal{N}^{\mathrm{M}}(\boldsymbol{x})}$ with *given* class label $m \in \{i(\boldsymbol{x}; \Lambda), j(\boldsymbol{x}; \Lambda)\}$ fall in $[-h_{\boldsymbol{x}}; h_{\boldsymbol{x}}]$
Section 4.4.1	$\hat{U}^{(2)}(\Lambda)$	Given Λ, finite estimate of local uncertainty provided by Proposal 2
Section 4.4.3	$\mathcal{T}_j$	Given $j \in [\![1, J]\!]$, set of training samples with given class label C_j
Section 4.4.3	K_j	Given $j \in [\![1, J]\!]$, number of prototypes used to represent class C_j
Section 4.4.4	N_j	Given $j \in [\![1, J]\!]$, number of training samples with given class label C_j
Section 4.5	$\hat{U}^{(3)}(\Lambda)$	Given Λ, estimate of expected boundary uncertainty provided by Proposal 2 for separability cases

Table 1 (continued)

Section	Notation	Description
Section 5.1	c_{add}	Time costs of a scalar addition
Section 5.1	c_{mul}	Time costs of a scalar multiplication
Section 5.1	c_{exp}	Time costs to compute exponential function
Section 5.1	c_{comp}	Time costs of a scalar comparison
Section 5.1	$c_{CL}(d, J, \Lambda)$	Given Λ, time cost of one classification in the sample space that has d dimensions and J classes
Section 5.1	$c^{(2)}(\Lambda)$	Given Λ, time costs of Proposal 2 to estimate $U(\Lambda)$
Section 5.1	$c^{(2)}_{shared}$	Preliminary time cost of Proposal 2 to estimate $U(\Lambda)$, which is shared across L candidate classifier statuses and applied only once, regardless of L
Section 5.1	R_K	Number of values that we try for K
Section 5.2	$c^{(1)}(\Lambda)$	Given Λ, time costs of Proposal 2 to estimate $U(\Lambda)$

minimum classification error probability (aka a Bayes error) is [19]

$$C^*(\boldsymbol{x}) = C_r \Leftrightarrow r = \underset{j \in [\![1,J]\!]}{\operatorname{argmax}} P(C_j|\boldsymbol{x}). \tag{4}$$

This optimal classification decision is called the Bayes decision rule. Unfortunately, class posterior probabilities are unknown and difficult to accurately estimate in practice.

2.2 Classifier Decision Rule

We denote by $\mathcal{T} = \{(\boldsymbol{x}_n, y_n)\}_{n \in [\![1,N]\!]}$ the training set that consists of N pairs of a training sample $\boldsymbol{x}_n \in \mathcal{X}$ and of its class index $y_n \in [\![1, J]\!]$. We emphasize the difference between $y(\boldsymbol{x}_n) = y_n$, the *given* class index at $\boldsymbol{x}_n$, and $C(\boldsymbol{x}_n; \Lambda)$, the class label *predicted by the classifier* at any sample $\boldsymbol{x}_n$. The decision of most classifiers is formalized as

$$C(\boldsymbol{x}; \Lambda) = C_r \Leftrightarrow r = \underset{j \in [\![1,J]\!]}{\operatorname{argmax}} g_j(\boldsymbol{x}; \Lambda). \tag{5}$$

The parameter vector Λ encompasses trainable parameters that are optimized on the training data (e.g., network weights) as well as hyperparameters that are traditionally set using validation data (e.g., regularization parameters). We refer to a specific value of Λ as "classifier status." $g_j(\cdot; \Lambda)$ is called a discriminant function. Its value at a sample $\boldsymbol{x}$ estimates the degree to which $\boldsymbol{x}$ belongs to C_j. $g_j(\boldsymbol{x}; \Lambda)$ does not necessarily directly estimate $P(C_j|\boldsymbol{x})$ [20]. The goal of this training is to find Λ so that the values of $g_j(\cdot; \Lambda)$ result in a classifier decision rule (5) that executes the Bayes decision rule (4).

2.3 Classifier Evaluation and Classifier Selection

A classifier evaluation metric measures the generalization ability of a classifier. This evaluation metric is traditionally the error probability, and its estimate is usually obtained using validation data. A lower classification error probability value indicates a classifier decision that has higher generalization ability.

As illustrated in Fig. 1, classifier selection is the process of evaluating different classifier statuses $\Lambda_{\mathrm{TR}} = \{\Lambda_m\}_{m \in [\![1,L]\!]}$ (green boxes on the left feed into the blue evaluation box in the center) in terms of a classifier evaluation metric (values for L different classifier statuses obtained are represented by red boxes on the right) and then selecting the status that scores the best (black box on the right).

3 Previous Boundary Uncertainty Estimation: Proposal 1

3.1 Previous Definition of Boundary Uncertainty

3.1.1 Bayes Boundary

We denote by B^* the classification boundary that is uniquely defined by Eq. 4, and we term it "Bayes boundary." B^* is the optimal classification boundary that a classifier should execute. B^* is locally defined by the equality between the highest class posterior probabilities. At a given

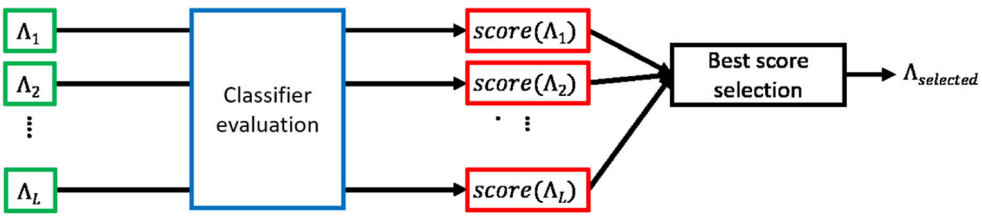

Figure 1 Classifier selection. A "classifier status" Λ corresponds to the set of values of parameters (trainable parameters and hyperparameters) of a classifier.

$\boldsymbol{x} \in \mathcal{X}$, there can be equality between the three or more highest class posterior probabilities. For simplicity, we assume that equality is achieved only between the two highest class posterior probabilities at $\boldsymbol{x}$ in practice. We denote by $\{i^*(\boldsymbol{x}), j^*(\boldsymbol{x})\}$ the indexes of the two highest (true) class posterior probabilities at $\boldsymbol{x}$. By convention, we order them so that $i^*(\boldsymbol{x}) < j^*(\boldsymbol{x})$. With these notations,

$$B^* = \left\{\boldsymbol{x} \in \mathcal{X} \middle| P(C_{i^*(\boldsymbol{x})}|\boldsymbol{x}) = P(C_{j^*(\boldsymbol{x})}|\boldsymbol{x})\right\}. \tag{6}$$

This definition conveys the true impossibility of deciding for a single class along B^*. We interpret this situation as uncertainty along B^*.

3.1.2 Classifier Boundary

We denote by $B(\Lambda)$ the classifier boundary that corresponds to the decision rule given by Eq. 5. Similarly to Section 3.1.1, we assume that $B(\Lambda)$ involves equality between only the two classes that yield the highest discriminant function values at $\boldsymbol{x}$. We denote by $\{i(\boldsymbol{x}; \Lambda), j(\boldsymbol{x}; \Lambda)\}$ the indexes of these two classes at $\boldsymbol{x}$; by convention, we order them so that $i(\boldsymbol{x}; \Lambda) < j(\boldsymbol{x}; \Lambda)$. With these notations,

$$B(\Lambda) = \left\{\boldsymbol{x} \in \mathcal{X} \middle| g_{i(\boldsymbol{x};\Lambda)}(\boldsymbol{x}; \Lambda) = g_{j(\boldsymbol{x};\Lambda)}(\boldsymbol{x}; \Lambda)\right\}. \tag{7}$$

3.1.3 Boundary Uncertainty

Boundary uncertainty relies on two principles. First, the Bayes boundary B^* solely consists of *uncertain* samples, whose two highest class posterior probabilities are equal. Second, there is a one-to-one relationship between a classifier decision and its corresponding classification boundary. Based on these considerations, we previously proposed to evaluate the generalization ability of a classifier parameterized by Λ in terms of the optimality of its classifier boundary. We defined boundary uncertainty as a classifier evaluation metric that measures the degree of equality between B^* and $B(\Lambda)$.

In order to define boundary uncertainty, our previous work considered the "local uncertainty" of samples $\boldsymbol{x}$ that are on $B(\Lambda)$ and then denoted it by $\hat{U}(\boldsymbol{x}; \Lambda)$.[1] In principle, the local uncertainty $U(\boldsymbol{x}; \Lambda)$ takes as input the class posterior probability $P(C_{i(\boldsymbol{x};\Lambda)}|\boldsymbol{x})$.[2] $U(\boldsymbol{x}; \Lambda)$ measures the degree of equality between class posterior probabilities $P(C_{i(\boldsymbol{x};\Lambda)}|\boldsymbol{x})$ and $P(C_{j(\boldsymbol{x};\Lambda)}|\boldsymbol{x})$. We only expect from $U(\cdot; \Lambda)$ that it takes a higher value, since $P(C_{i(\boldsymbol{x};\Lambda)}|\boldsymbol{x})$ and $P(C_{j(\boldsymbol{x};\Lambda)}|\boldsymbol{x})$ are closer to equality, and lower values otherwise. $U(\boldsymbol{x}; \Lambda)$ implicitly contains the top indexes $\{i(\boldsymbol{x}; \Lambda), j(\boldsymbol{x}; \Lambda)\}$ through its arguments $\boldsymbol{x}$ and Λ.

Figure 2 illustrates two possible choices of local uncertainty function: a triangle function (orange curve) and the binary Shannon entropy (blue curve). The horizontal axis of the graph is indifferently $P(C_{i(\boldsymbol{x};\Lambda)}|\boldsymbol{x})$ or $P(C_{j(\boldsymbol{x};\Lambda)}|\boldsymbol{x})$. The minimum and maximum values of these local uncertainty functions are $U_{min} = 0$ and $U_{max} = 1$, respectively. This results in a convenient range of values [0, 1] that is the same regardless of the dataset, where "1" indicates optimality of the boundary at $\boldsymbol{x}$.

Then, given a classifier status Λ, Proposal 1 empirically defined the following classifier evaluation metric that we estimated from the *training* set [18] and termed "boundary uncertainty":

$$\hat{U}(\Lambda) = \frac{1}{|\mathcal{N}_B(\Lambda)|} \sum_{\boldsymbol{x} \in \mathcal{N}_B(\Lambda)} \hat{U}(\boldsymbol{x}; \Lambda). \tag{8}$$

$\mathcal{N}_B(\Lambda)$ refers to a "near-boundary set" that ideally would consist of samples exactly on $B(\Lambda)$. $\mathcal{N}_B(\Lambda)$ acted as a practical approximation of $B(\Lambda)$ consisting of training samples that were very close to $B(\Lambda)$, all along $B(\Lambda)$. $|\cdot|$ refers to the cardinal operator. The notation $\hat{\cdot}$ in this paper refers to estimated quantities based on a finite dataset, in order to contrast with their expected value (based on an infinite dataset).

In Eq. 8, boundary uncertainty is defined as the finite expectation of the local uncertainty function over the $\mathcal{N}_B(\Lambda)$. Owing to the characterization of the Bayes boundary B^* and to the property of the local uncertainty function, boundary uncertainty reaches a higher value because the classifier boundary is more highly optimal, and lower values otherwise. Furthermore, the relationship between classification error probability and boundary uncertainty was the equivalence between achieving U_{max}

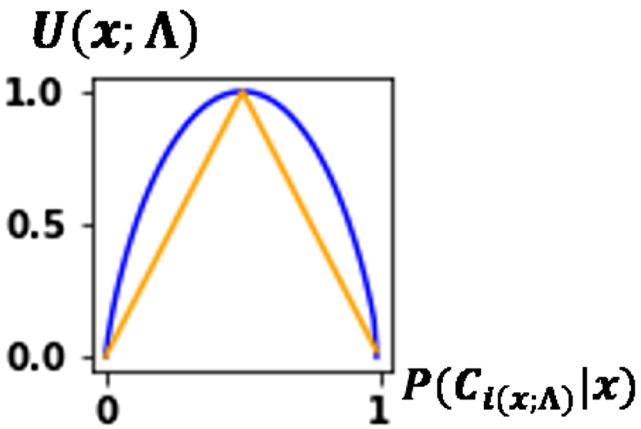

Figure 2 Two possible local uncertainty functions: binary Shannon entropy (blue) and a triangle function (orange).

[1] In our previous work, we used the notation $H(\boldsymbol{x})$ instead of $\hat{U}(\boldsymbol{x}; \Lambda)$ [18].

[2] In this paragraph, we do not use the $\hat{\cdot}$ notation because we refer to the local uncertainty function in principle, rather than its estimate measured by Proposal 1.

and achieving the minimum error probability (Bayes error). We describe this matter in more detail in later sections.

To compute (8), Proposal 1 consists of two steps described in Sections 3.3 and 3.4, respectively. Before reviewing these two steps, we review the k Nearest Neighbor (kNN) regression rule that Proposal 1 relied upon to estimate the class posterior probability $P(C_{i(\boldsymbol{x};\Lambda)}|\boldsymbol{x})$ input into the local uncertainty.

3.2 Nonparametric Posterior Probability Estimation using kNN Regression

To estimate local uncertainties, both Proposal 1 and Proposal 2 rely on the kNN regression because it estimates class posterior probabilities without globally imposing a probability model on the class distributions. Given $\boldsymbol{x} \in \mathcal{X}$ and $j \in [\![1, J]\!]$, in order to estimate $P(C_j|\boldsymbol{x})$, kNN regression requires a small volume $V(\boldsymbol{x})$ that contains $\boldsymbol{x}$. We denote by $k(\boldsymbol{x})$ the number of samples that are contained in $V(\boldsymbol{x})$, and by $k_j(\boldsymbol{x})$ the number of samples among them that carry the label C_j (the basic version of kNN regression considers a uniform k across $\mathcal{X}$, but adaptative kNN regression methods adapt k to each sample $\boldsymbol{x}$).

kNN regression gives $\hat{P}(C_j|\boldsymbol{x}) = k_j(\boldsymbol{x})/k(\boldsymbol{x})$. A higher $k(\boldsymbol{x})$ provides more samples for the estimation (lower variance), but these samples are farther away from $\boldsymbol{x}$ (higher bias). This requires a tradeoff. $k(\boldsymbol{x})$ is usually set using validation data.

3.3 Step 1: Generation of Anchors

Sections 3.3 and 3.4 refer to Panels **1A** to **1D** of Fig. 3. Panel **1A** illustrates Step 1 of Proposal 1: We consider a training set $\mathcal{T}$ whose samples of given labels C_1 and C_2 are represented by triangles and squares, respectively. Proposal 1 evaluates a candidate classifier status Λ that it takes as **Input** (left-hand side of Fig. 3). We illustrate $B(\Lambda)$ with a blue curve.

Proposal 1 directly applied the definition of $B(\Lambda)$ (7) to find $B(\Lambda)$. It searched for zeros of the function $f(\boldsymbol{x};\Lambda) = g_{i(\boldsymbol{x};\Lambda)}(\boldsymbol{x};\Lambda) - g_{j(\boldsymbol{x};\Lambda)}(\boldsymbol{x};\Lambda)$ in $\mathcal{X}$. $\boldsymbol{a}$ is characterized by $f(\boldsymbol{a};\Lambda) = 0$, so the theorem of intermediate values applied to $f(\cdot;\Lambda)$ guarantees the existence of a sample $\boldsymbol{a} \in B(\Lambda)$ on any segment $[\boldsymbol{x}, \boldsymbol{x}']$ that satisfies $f(\boldsymbol{x};\Lambda) > 0$ and $f(\boldsymbol{x}';\Lambda) < 0$. We termed such an on-boundary sample as $\boldsymbol{a}$ an "anchor" (anchors are represented by red dots in Panel **1A**). Proposal 1 therefore searched for anchors along the segments that join randomly picked couples of training samples $\{\boldsymbol{x}; \boldsymbol{x}'\}$ that satisfy $f(\boldsymbol{x};\Lambda) > 0$ and $f(\boldsymbol{x}';\Lambda) < 0$. The search along $[\boldsymbol{x}; \boldsymbol{x}']$ can be done by a dichotomy: at each dichotomy iteration, we checked the sign of $f(\cdot;\Lambda)$ at the middle of the segment, and then considered only the half of the segment that contains a change of sign of $f(\cdot;\Lambda)$ (namely, an anchor). In practice, we imposed a maximum number of dichotomy iterations that we denoted by i_{max}.

How many and where anchors should be generated was a priori not obvious. In Eq. 6, the equality of class posterior probabilities should be checked all along $B(\Lambda)$. To get as close to this ideal situation as possible, Proposal 1 generated a large number of anchors denoted by R_A, each of which we obtained from a random pair of training samples as described above. In the multiclass case, Proposal 1 had to check $i(\cdot;\Lambda)$, $j(\cdot;\Lambda)$ at each dichotomy iteration. Indeed, if we generate an anchor defined by equality between two discriminant functions whose class indexes are not locally

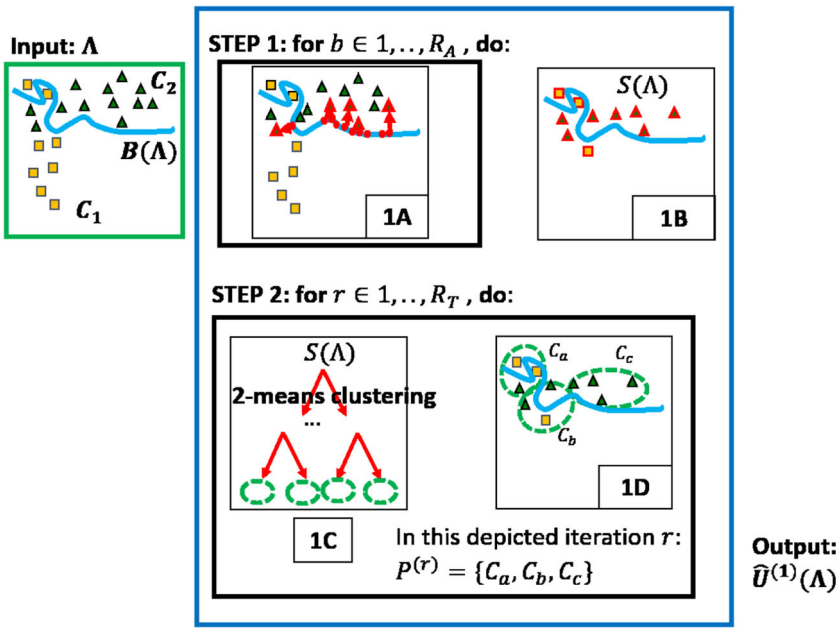

Figure 3 Estimation of $\hat{U}(\Lambda)$ by our previous Proposal 1 on two-class and two-dimensional data. Samples of given class labels C_1 and C_2 are represented by yellow squares and green triangles, respectively. For a given Λ, $B(\Lambda)$ is represented by a blue curve. We provide step-by-step explanations in Sections 3.3 and 3.4.

the highest, then this anchor is actually generated on a non-existing classifier boundary (in the sense of Eq. 7).

3.4 Step 2: Near-Boundary Local Uncertainty Estimation

3.4.1 Filtering Out Off-Boundary Samples

Given an anchor $\boldsymbol{a} \in B(\Lambda)$, the most direct way to apply kNN regression would be to use as target volume $V(\boldsymbol{a})$, the volume defined by the k training samples that are nearest to $\boldsymbol{a}$. However, this approach led to poor quality of the boundary uncertainty estimate, especially for higher-dimensional data, because a naive spherical target volume largely contains off-boundary samples that are biased in terms of boundary uncertainty estimation. Proposal 1 filtered out this bias by forming target volumes that only contain near-boundary samples. Proposal 1 defined the set of near-boundary samples denoted by $\mathcal{N}_B(\Lambda)$ as the set of one-nearest training sample for each anchor generated in Step 1 (Panel **1B**).

3.4.2 Partitioning Near-Boundary Set into Target Volumes

Proposal 1 used Tree Divisive Clustering (TDC), of which imposed number of repetitions is R_T, to break down $\mathcal{N}_B(\Lambda)$ into clusters that we used as target volumes for kNN regression. We used such a clustering method so that the obtained target volumes could adapt to the distribution. As described in the next paragraph, the iterative nature of TDC enabled us to progressively keep some control over the size of clusters.

Every step of the TDC consists of a 2-means clustering that ends after an imposed i_K number of iterations: TDC thus adaptatively broke down $\mathcal{N}_B(\Lambda)$ into clusters (Panel **1C**). We denote by $\P_\Lambda$ the set of resulting clusters, and by $\mathcal{C} \in \P_\Lambda$ one of the clusters. Here, the dependency in Λ emphasizes that the partition obtained by clustering is defined for a given $\mathcal{N}_B(\Lambda)$. Proposal 1 applied the kNN regression by using each $\mathcal{C}$ as a target volume (Panel **1D**), or in other words, the "k" of "kNN regression" is set to $|\mathcal{C}|$ for each cluster $\mathcal{C}$.

Clusters should contain enough samples to perform kNN regression, while focusing on $B(\Lambda)$ as locally as possible (local uncertainties ideally measure equality between class posteriors on points along $B(\Lambda)$). Proposal 1 therefore continued to divide $\mathcal{N}_B(\Lambda)$ into smaller clusters so long as the clusters obtained at a given TDC step contained more than N_{max} samples. Then, Proposal 1 only retained clusters that contained more than N_{min} samples.

3.4.3 Local Uncertainty Computation

Proposal 1 adopted binary Shannon entropy as the local uncertainty function (blue curve in Fig. 2). When using the kNN regression to estimate class posterior probabilities, we do not consider *individual* samples but rather *target volumes* (in our case, elements of $\P_\Lambda$). Therefore, in the following, we explicitly describe the local uncertainty of clusters rather than that of individual samples.

Given a near-boundary cluster $\mathcal{C} \in \P_\Lambda$, for each given class index $j \in [\![1, J]\!]$, we count the number of samples that have *predicted* class index j. To estimate $\{i(\mathcal{C};\Lambda), j(\mathcal{C};\Lambda)\}$ defined in Section 3.1.2, we define $\{\hat{i}(\mathcal{C};\Lambda), \hat{j}(\mathcal{C};\Lambda)\}$ as the pair of predicted class indexes that have the most samples in $\mathcal{C}$. Then, we estimate the local uncertainty of this cluster as

$$\begin{aligned}\hat{U}(\mathcal{C};\Lambda) = &-\hat{P}(C_{\hat{i}(\mathcal{C};\Lambda)}|\mathcal{C}) \log\left(\hat{P}(C_{\hat{i}(\mathcal{C};\Lambda)}|\mathcal{C})\right) \\ &-\hat{P}(C_{\hat{j}(\mathcal{C};\Lambda)}|\mathcal{C}) \log\left(\hat{P}(C_{\hat{j}(\mathcal{C};\Lambda)}|\mathcal{C})\right). \end{aligned} \tag{9}$$

Recall that our estimation goal is ideally $B(\Lambda)$ itself, so we hope that $\mathcal{C}$ is centered on $B(\Lambda)$. Unfortunately, the TDC may result in some $\mathcal{C}$s "completely on one side of $B(\Lambda)$," which results in a kind of bias in terms of local uncertainty estimation.

3.4.4 Boundary Uncertainty Computation

Sampling elements (clusters) from the partition $\P_\Lambda$ means that we are sampling independent outcomes (clusters) from the vicinity $\mathcal{N}_B(\Lambda)$ of the classifier boundary. Each cluster $\mathcal{C}$ has probability $\frac{|\mathcal{C}|}{|\P_\Lambda|}$. The finite expectation of the local uncertainty function over $\P$ is thus

$$\hat{U}^{(1)}(\P_\Lambda, \Lambda) = \sum_{\mathcal{C}\in\P_\Lambda} \hat{U}(\mathcal{C};\Lambda)\frac{|\mathcal{C}|}{|\P_\Lambda|}. \tag{10}$$

Superscript "(1)" refers to Proposal 1, in order to distinguish it from the boundary uncertainty estimate obtained by Proposal 2. $\P_\Lambda$ obtained by a TDC depends on the random clustering seeds used at each TDC step (i.e., a 2-means clustering). Step 2 of Proposal 1 accordingly performed $R_T > 1$ different runs of TDCs. Each run indexed by $r \in [\![1, R_T]\!]$ has its own different set of random clustering seeds and results in a partition we denote by $\P_\Lambda^{(r)}$. Then, Proposal 1 estimates $\hat{U}(\Lambda)$ in Eq. 8 by averaging over the

boundary uncertainty estimates obtained by each run. This results in the following estimate denoted by $\hat{U}^{(1)}(\Lambda)$:

$$\hat{U}^{(1)}(\Lambda) = \frac{1}{R_T} \sum_{r=1}^{R_T} \sum_{\mathcal{C} \in \P_\Lambda^{(r)}} \hat{U}(\mathcal{C}; \Lambda) \frac{|\mathcal{C}|}{|\P_\Lambda^{(r)}|}. \tag{11}$$

We illustrate (11) with Panel **1D** contained in the black box that indicates R_T repetitions of Step 2.

3.5 Issues in Step 1 of Proposal 1

On the one hand, anchors and then the selection of their nearest training samples answered the difficult question "what is closest to $B(\Lambda)$?" by accurately executing the definition of $B(\Lambda)$. On the other hand, an approach that revolves around anchors to search for $B(\Lambda)$ may not be practical, since it is impossible to generate infinitely many anchors as required by the definition of $B(\Lambda)$.

Moreover, generating anchors from random pairs of training samples (that may be far from each other) results in costly treatments as described in Section 3.3. For example, resuming the notations of Section 3.3, for every checked position $\boldsymbol{x}_c$ along $[\boldsymbol{x}, \boldsymbol{x}']$, finding $i(\boldsymbol{x}_c; \Lambda)$, $j(\boldsymbol{x}_c; \Lambda)$ using QuickSort costs $O(J \log(J))$ comparisons between J discriminant function values. This cost explodes as it is multiplied by the number i_{max} of dichotomy iterations, then by the large number R_A of anchors, and finally by the number of classifier statuses to evaluate.

3.6 Issues in Step 2 of Proposal 1

On the one hand, previous experiments showed that the restriction to $\mathcal{N}_B(\Lambda)$ of target volumes used for kNN regression efficiently increases the accuracy of $\hat{U}^{(1)}(\Lambda)$ [18]. On the other hand, a discrete selection of "what is close to $B(\Lambda)$ or not" unavoidably leads to the following dilemma: either taking too few samples that are not representative of $B(\Lambda)$ or taking too many samples that are farther from $B(\Lambda)$.

Moreover, the centering of target volumes on $B(\Lambda)$ is not explicitly guaranteed, which can result in a bias of the boundary uncertainty estimate. Furthermore, the formation of clusters obeys a certain clustering criterion, and we thus do not have direct control over the size of clusters (e.g., when applying the 2-means clustering, some clusters may persistently be broken into most of the cluster itself, and a "leftover" that consists of just one or two samples). This may result in a partition that has some persistently large-sized clusters that do not measure *locally* the boundary uncertainty, as well as remaining "leftover" clusters that are too small to estimate local uncertainty. Last but not least, the R_T repetitions of Step 2 increase both time and memory costs.

4 New Boundary Uncertainty Formalization and Estimation: Proposal 2

4.1 Overview of Proposal 2

To avoid the costly generation of anchors in the multidimensional space (Section 3.5), Proposal 2 *smoothly* and *implicitly* filters off-boundary samples in a single shot. This is made possible by using a single dimension that represents a kind of distance of a sample to the classifier boundary (Section 3.1.3).

In contrast to Proposal 1, which did not guarantee that near-boundary clusters are centered on $B(\Lambda)$ (Section 3.6), Proposal 2 can explicitly focus its estimation on $B(\Lambda)$ (Section 4.4.1). This is possible because the value "zero" of the aforementioned single dimension is equivalent to being on $B(\Lambda)$.

In contrast to the difficulty of controlling the size of clusters in Proposal 1 (Section 3.6), Proposal 2 uses target volumes where the number of samples is directly specified. Then, the *effectively used* number of samples in each target volume is adjusted automatically in one shot by Proposal 2. The determination of the *effective* number of samples for each cluster and each candidate classifier status is both cheap and justified (Section 4.4.2): it is determined so that the local uncertainty estimation appropriately focuses on $B(\Lambda)$ (Section 4.4.1).

The empirical formalization of boundary uncertainty used in Proposal 1 actually left some unclear items, such as the possible confusion of whether to use $\{i^*(\boldsymbol{x}), j^*(\boldsymbol{x})\}$ (top given class indexes) or $\{i(\boldsymbol{x}; \Lambda), j(\boldsymbol{x}; \Lambda)\}$ (top predicted class indexes). Such confusion was especially possible in the several intricate branching treatments that appeared in Proposal 1, as well as in the treatment of multiclass data [18]. In contrast, this paper introduces a more complete formalization of boundary uncertainty (Section 4.3). This formalization clarifies these items and permits a more systematic approach to boundary uncertainty estimation.

4.2 Near-Boundary-Ness Measurement

4.2.1 Definition of Near-Boundary-Ness Measurement

Given any two class indexes $k, l \in [\![1, J]\!]$, a classifier status Λ, and $\boldsymbol{x} \in \mathcal{X}$, we define the near-boundary-ness measurement as follows:

$$f_{kl}(\boldsymbol{x}; \Lambda) = g_k(\boldsymbol{x}; \Lambda) - g_l(\boldsymbol{x}; \Lambda). \tag{12}$$

A smaller value of $|f_{kl}(\boldsymbol{x}; \Lambda)|$ means that $\boldsymbol{x}$ is closer to $B_{kl}(\Lambda)$, hence we termed $f_{kl}(\boldsymbol{x}; \Lambda)$ a "near-boundary-ness measurement." The sign of $f_{kl}(\boldsymbol{x}; \Lambda)$ indicates which "side" of $B_{kl}(\Lambda)$ the sample $\boldsymbol{x}$ is, namely whether the classifier assigns $\boldsymbol{x}$ to C_k or to C_l. To simplify notations,

we shorten $f_{i(\boldsymbol{x};\Lambda)j(\boldsymbol{x};\Lambda)}(\boldsymbol{x};\Lambda)$ to $f(\boldsymbol{x};\Lambda)$ in the particular case $\{k,l\} = \{i(\boldsymbol{x};\Lambda), j(\boldsymbol{x};\Lambda)\}$, as well as in the two-class case. $f_{kl}(\cdot;\Lambda)$ can be computed for any classifier whose formalization follows Section 2. For example, in the case of a neural network, $f_{kl}(\boldsymbol{x};\Lambda)$ is obtained by monitoring the output layer.

4.2.2 Estimation of Near-Boundary-Ness Measurement Values at Perturbated Samples Instead of at Training Samples

The goal of this section is to provide a reliable representation of $\mathcal{X}$ in the near-boundary-ness measurement space. This lengthy section describes a preliminary treatment that we use throughout Proposal 2, so we cover it here before describing Proposal 2 itself to provide the necessary clarity for following the later sections. We temporarily assume two-class data for simplicity of notation in this section.

Application of the function $f(\cdot;\Lambda)$ on training set $\mathcal{T}$ can be understood as sampling a mapped training set in the near-boundary-ness measurement space. We denote by $f(\mathcal{T};\Lambda)$ this projection. As covered in the next sections, Proposal 2 performs some density estimations on $f(\mathcal{T};\Lambda)$ in the near-boundary-ness measurement space. The goal of such density estimations is to estimate the density of $f(\mathcal{X};\Lambda)$, or in other words to generalize effectively to the density of the entire mapped sample space. However, $f(\cdot;\Lambda)$ reflects the classifier decision, and it is well known that the classifier decision values can be quite different on the training set and on some (unseen) testing set $\mathcal{X}^{te}$. This phenomenon is one aspect of overfitting.

In other words, even if $\mathcal{T}$ and $\mathcal{X}^{te}$ follow similar distributions in the sample space, their mapped counterparts $f(\mathcal{T};\Lambda)$ and $f(\mathcal{X}^{te};\Lambda)$ may follow quite different distributions in the near-boundary-ness measurement space. This sampling issue in the near-boundary-ness measurement space may be seen as a case of covariate shift [21].

Naively using $f(\mathcal{T};\Lambda)$ may not lead to an accurate estimation of the density of $f(\mathcal{X};\Lambda)$. We propose slightly perturbating training samples and then applying $f(\cdot;\Lambda)$ at such perturbated versions of the training samples instead of at the training samples themselves. We denote by $\widetilde{\mathcal{T}}$ this perturbated version, where $\widetilde{\cdot}$ (tilde notation) denotes the operation of "perturbation." Our hope is that $f(\widetilde{\mathcal{T}};\Lambda)$ suffers the covariate shift issue to a lesser extent than $f(\mathcal{T};\Lambda)$.

Algorithm 1 describes the generation of $\widetilde{\mathcal{T}}$. Given a training sample $\boldsymbol{x}$, we denote by $\boldsymbol{x}^{(m)}$ its m-th nearest training sample. The perturbation is uniform across the features of each training sample. The amplitude of the perturbation is adaptatively set small to stay as close as possible to the training set in the sample space. A detailed description of Proposal 2 itself is given in the next section to explain why we want to stay close to the training set in the sample space.

$\widetilde{\mathcal{T}}$ may be simple to obtain in principle for a wide range of classification tasks. For example, in the case of speech data, our perturbation can be obtained by adding some noise to the speech signal, which seems to be a common practice when preparing speech data [22].

Algorithm 1 Generation of $\widetilde{\mathcal{T}}$.

Input: $\mathcal{T}$, $\{x^{(1)}\}_{\boldsymbol{x}\in\mathcal{T}}$
for $\boldsymbol{x} \in \mathcal{T}$ **do**
  Generate a d-dimensional random vector $r_{\boldsymbol{x}}$, whose features are each sampled independently and uniformly;
  $\widetilde{x} \leftarrow \boldsymbol{x} + \frac{r_x}{||r_x||}||\boldsymbol{x} - \boldsymbol{x}^{(1)}||$;
end
return $\widetilde{\mathcal{T}} = \{\widetilde{x}\}_{\boldsymbol{x}\in\mathcal{T}}$

4.3 Formalization of Boundary Uncertainty

In order to more adequately derive an estimation procedure for boundary uncertainty, we start by defining the expected boundary uncertainty that we denote by $U(\Lambda)$.

4.3.1 Notations

Recall that we denoted by $\{i^*(\boldsymbol{x}), j^*(\boldsymbol{x})\}$ in Eq. 6 the indexes of the two class posterior probabilities whose values are highest at $\boldsymbol{x}$. We denote by $\mathcal{I}^*$ the set of pairs of indexes that compose the Bayes boundary:

$$\mathcal{I}^* = \left\{\{i^*(\boldsymbol{x}), j^*(\boldsymbol{x})\}\right\}_{\boldsymbol{x}\in\mathcal{X}}. \tag{13}$$

$\forall\{k,l\} \in \mathcal{I}^*$, and we denote by $B^*(k,l)$ the set of training samples whose two top indexes are k and l:

$$B^*(k,l) = \left\{\boldsymbol{x} \in \mathcal{X} \,\middle|\, \{i^*(\boldsymbol{x}), j^*(\boldsymbol{x})\} = \{k,l\}\right\}. \tag{14}$$

4.3.2 Expected Boundary Uncertainty

Similarly to the definition of the risk in Eq. 1, we propose

$$-U(\Lambda) = \int_{\boldsymbol{x}\in(B(\Lambda)+\delta V)} R_U\left(C(\boldsymbol{x})\middle|\boldsymbol{x}\right) p(\boldsymbol{x}) d\boldsymbol{x}, \tag{15}$$

where $R_U\left(C(\boldsymbol{x})|\boldsymbol{x}\right)$ represents the penalization of non-boundary uncertainty of the classification decision $C(\boldsymbol{x})$ at $\boldsymbol{x}$. We consider $-U(\Lambda)$ instead of $U(\Lambda)$, since $U(\Lambda)$ is the opposite of a loss: Higher values of $U(\Lambda)$ correspond to more optimal classifier decisions. The infinitesimal width δV is introduced to avoid a probability measure equal to zero. We will clarify this formalization item in a future work. $p(\boldsymbol{x})$ denotes the density at $\boldsymbol{x}$.

We now consider a classification decision $C(\boldsymbol{x};\Lambda)$ made by a classifier that is parameterized by Λ, where $C(\boldsymbol{x};\Lambda)$ is described in Eq. 5. In the boundary uncertainty framework, our quantity of interest in $C(\boldsymbol{x};\Lambda)$ is the pair of the two top predicted indexes (Sections 3.1.1 and 3.1.2). In this section, we write $C_{i(\boldsymbol{x};\Lambda)j(\boldsymbol{x};\Lambda)}(\boldsymbol{x};\Lambda)$ instead of $C(\boldsymbol{x};\Lambda)$ in order to emphasize this quantity of interest. The penalization of non-boundary uncertainty of the classifier at $\boldsymbol{x}$ is

$$R_U\left(C_{i(\boldsymbol{x};\Lambda)j(\boldsymbol{x};\Lambda)}(\boldsymbol{x};\Lambda)|\boldsymbol{x}\right) = \sum_{\substack{\{k,l\}\in[\![1,J]\!]^2\\ k\neq l}} \times\lambda_U\left(C_{i(\boldsymbol{x};\Lambda)j(\boldsymbol{x};\Lambda)}(\boldsymbol{x};\Lambda)\,|C_{kl}\right)P(C_{kl}|\boldsymbol{x}), \tag{16}$$

where we introduced the output variable C_{kl}. Once more, the subscript in C_{kl} involves two class indexes $\{k,l\}$ owing to the focus of boundary uncertainty on the pair of class indexes that define the boundaries. We define C_{kl} through the following discrete probability distribution: $\forall\{k,l\}\in[\![1,J]\!]^2, k\neq l$,

$$P(C_{kl}|\boldsymbol{x}) = \begin{cases} 1, & \text{if } \{k,l\} = \{i^*(\boldsymbol{x}), j^*(\boldsymbol{x})\},\\ 0, & \text{otherwise.}\end{cases} \tag{17}$$

To evaluate the classification decision at $\boldsymbol{x}$, the "boundary-wise cost" $\lambda_U\left(C_{i(\boldsymbol{x};\Lambda)j(\boldsymbol{x};\Lambda)}(\boldsymbol{x};\Lambda)\,|C_{kl}\right)$ penalizes at $\boldsymbol{x}$ the non-uncertainty of the two-class classifier boundary that is relative to top indexes $i(\boldsymbol{x};\Lambda)$, $j(\boldsymbol{x};\Lambda)$ as follows:

$$\lambda_U\left(C_{i(\boldsymbol{x};\Lambda)j(\boldsymbol{x};\Lambda)}(\boldsymbol{x};\Lambda)\,|C_{kl}\right) = \begin{cases} -U(\boldsymbol{x};\Lambda) & \text{if } \{i(\boldsymbol{x};\Lambda), j(\boldsymbol{x};\Lambda)\} = \{k,l\},\\ -U_{min}, & \text{otherwise.}\end{cases} \tag{18}$$

The integration of Eqs. 17 and 18 into Eq. 16 reveals that at a sample $\boldsymbol{x}$, we consider only the two true top class indexes $\{i^*(\boldsymbol{x}), j^*(\boldsymbol{x})\}$, and then we evaluate the optimality of the two-class classifier boundary $B_{i(\boldsymbol{x};\Lambda)j(\boldsymbol{x};\Lambda)}(\Lambda)$ at $\boldsymbol{x}$. If the indexes $\{i(\boldsymbol{x};\Lambda), j(\boldsymbol{x};\Lambda)\}$ do not even coincide with the true top indexes $\{i^*(\boldsymbol{x}), j^*(\boldsymbol{x})\}$, this indicates that the classifier boundary is strongly biased around $\boldsymbol{x}$, and thus (18) locally assigns the default worst uncertainty value $-U_{min}$ at $\boldsymbol{x}$. If $\{i(\boldsymbol{x};\Lambda)j(\boldsymbol{x};\Lambda)\} = \{i^*(\boldsymbol{x}), j^*(\boldsymbol{x})\}$ (as it should be in most cases if the classifier training did not fail), then we evaluate the classifier boundary at $\boldsymbol{x}$ using the sign-reversed local uncertainty function $-U(\boldsymbol{x};\Lambda)$. This branching treatment in our formalization reflects the branching treatments that appeared in Proposal 1 [18].

The Bayes theorem gives $P(C_{kl}|\boldsymbol{x})p(\boldsymbol{x}) = p(\boldsymbol{x}|C_{kl})P(C_{kl})$, where $P(C_{kl})$ corresponds to the overall probability in the sample space $\mathcal{X}$ that $\{k,l\}$ are the true top class indexes. Substituting (17), (18) and using the Bayes theorem in Eq. 16 gives

$$U(\Lambda) = -\sum_{\{k,l\}\in\mathcal{I}^*} P(C_{kl}) \int_{B^*(k,l)\cap(B(\Lambda)+\delta V)} \times\lambda_U\left(C_{i(\boldsymbol{x};\Lambda)j(\boldsymbol{x};\Lambda)}(\boldsymbol{x};\Lambda)\,|C_{kl}\right)p(\boldsymbol{x}|C_{kl})d\boldsymbol{x}, \tag{19}$$

where we note that $B^*(k,l)$ is defined in Eq. 14 and $\lambda_U\left(C_{i(\boldsymbol{x};\Lambda)j(\boldsymbol{x};\Lambda)}(\boldsymbol{x};\Lambda)\,|C_{kl}\right)$ is $-U(\boldsymbol{x};\Lambda)$ in most cases. Equation 19 appears as a sum only over pairs $\{k,l\}\in\mathcal{I}^*$ because the definition in Eq. 17 implies that the terms of Eq. 19 are non-zero only for pairs $\{k,l\}\in\mathcal{I}^*$. $\forall\{k,l\}\in\mathcal{I}^*$, we define

$$U_{kl}(\Lambda) = -\int_{B^*(k,l)\cap(B(\Lambda)+\delta V)} \times\lambda_U\left(C_{i(\boldsymbol{x};\Lambda)j(\boldsymbol{x};\Lambda)}(\boldsymbol{x};\Lambda)\,|C_{kl}\right) \times p(\boldsymbol{x}|C_{kl})d\boldsymbol{x}. \tag{20}$$

Equation 19 shows that in a multiclass setting, $U(\Lambda)$ appears as a weighted combination of two-class uncertainty boundaries $\{U_{kl}(\Lambda)\}_{\{k,l\}\in\mathcal{I}^*}$. The above expressions generalize both our previous empirical understanding of boundary uncertainty defined by Eq. 8 and Proposal 1.

4.3.3 Property of Boundary Uncertainty

As we described in Section 3.1.3, our previous work implicitly assumed equivalence between U_{max} and achieving the Bayes decision rule. We now consider this statement more carefully:

$$\begin{aligned} &U(\Lambda) = U_{max}\\ &\Leftrightarrow \forall\boldsymbol{x}\in B(\Lambda),\ P(C_{i(\boldsymbol{x};\Lambda)}|\boldsymbol{x}) = P(C_{j(\boldsymbol{x};\Lambda)}|\boldsymbol{x}) && (21)\\ &\Leftrightarrow B(\Lambda)\subset B^* && (22)\\ &\text{(in practice)}\ \Leftrightarrow B(\Lambda) = B^* && (23)\\ &\Leftrightarrow \text{Bayes decision rule defined by Eq. 4 is achieved.} && (24)\end{aligned}$$

Section 6.9 discusses the successive equivalences (21) to (24) in more detail. The next sections describe the estimation of $U(\Lambda)$.

4.4 Estimation of Expected Boundary Uncertainty

4.4.1 Empirical Boundary Uncertainty and its Implicit Centering on Classifier Boundary

In Proposal 2, we attempt to estimate the expected boundary uncertainty defined by Eq. 19 with the following empirical sum:

$$\hat{U}(\Lambda) = \sum_{\{k,l\}\in\hat{\mathcal{I}}^*} \hat{P}(C_{kl}) \sum_{\boldsymbol{x}\in\hat{B}^*(k,l)\cap(B(\Lambda)+\delta V)} \times\hat{\lambda}_U\left(C_{i(\boldsymbol{x};\Lambda)j(\boldsymbol{x};\Lambda)}(\boldsymbol{x};\Lambda)\,|C_{kl}\right). \tag{25}$$

$\hat{B}^*(k,l)$ is the approximation of $B^*(k,l)$ obtained from the training set, and $\hat{\lambda}_U\left(C_{i(\boldsymbol{x};\Lambda)j(\boldsymbol{x};\Lambda)}(\boldsymbol{x};\Lambda)\,|C_{kl}\right)$ is the empirical version of Eq. 18, which is discussed in detail in Section 4.4.4. We note that execution of the sum defined by Eq. 25 requires a preliminary estimation of $\mathcal{I}^*$, and we give details of its estimate denoted by $\hat{\mathcal{I}}^*$ in Section 4.4.3.

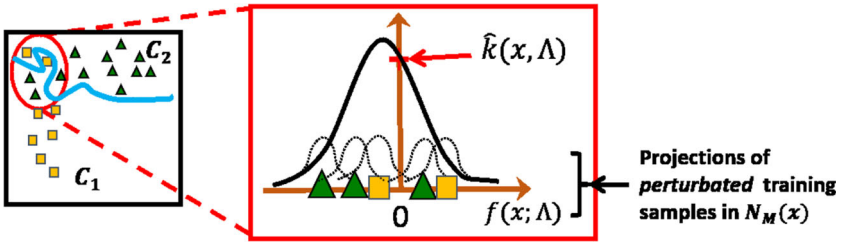

Figure 4 Illustration of the locally one-dimensional count by Proposal 2 on a two-class and two-dimensional dataset. Samples of given class labels C_1 and C_2 are represented by yellow squares and green triangles, respectively. For a given Λ, we represent $B(\Lambda)$ with a blue curve. The horizontal axis $f(\cdot; \Lambda)$ is described in Section 4.2.

A key point of Proposal 2 is to implicitly perform the filtering expressed by "$\cap\,(B(\Lambda) + \delta V)$" in Eq. 25 without having to explicitly generate anchors. This actually draws inspiration from a work that used a one-dimensional space based on the discriminant functions to estimate the classification error probability [23]. Here, we describe how to *implicitly* define clusters that are centered on $B(\Lambda)$.

Given an integer $M > 0$, $\forall \boldsymbol{x} \in \mathcal{T}$, we denote by $\mathcal{N}^{\mathrm{M}}(\boldsymbol{x})$ the set (cluster) formed by $\boldsymbol{x}$ and its $M-1$ nearest neighbors in $\mathcal{T}$:

$$\mathcal{N}^{\mathrm{M}}(\boldsymbol{x}) = \{\boldsymbol{x}, \boldsymbol{x}^{(1)}, \cdots, \boldsymbol{x}^{(M-1)}\}. \tag{26}$$

We consider a given $\boldsymbol{x} \in \mathcal{T}$. Given a kernel function (e.g., Gaussian kernel) and a small $h_{\boldsymbol{x}} > 0$ to specify, we define $\forall m \in \{i(\boldsymbol{x}; \Lambda), j(\boldsymbol{x}; \Lambda)\}$,

$$\hat{k}^m(\boldsymbol{x}; \Lambda) = \sum_{\boldsymbol{x}' \in \mathcal{N}^{\mathrm{M}}(\boldsymbol{x}) \mid y(\boldsymbol{x}') = m} \times \phi\left(\frac{0 - f_{i(\boldsymbol{x};\Lambda)j(\boldsymbol{x};\Lambda)}(\widetilde{x'}; \Lambda)}{h_{\boldsymbol{x}}}\right), \tag{27}$$

where we note that $y(\boldsymbol{x}')$ is the *given* class index of training sample $\boldsymbol{x}'$ and that $\boldsymbol{x} \in B(\Lambda) \Leftrightarrow f_{i(\boldsymbol{x};\Lambda)j(\boldsymbol{x};\Lambda)}(\boldsymbol{x}; \Lambda) = 0$. An important factor is the tilde notation (operation of perturbation) when computing the near-boundary-ness measurement in Eq. 27 according to the considerations mentioned in Section 4.2.2. $\hat{k}^m(\boldsymbol{x}; \Lambda)$ counts how many samples with given class index m fall within a distance $h_{\boldsymbol{x}}$ of $B(\Lambda)$ in the near-boundary-ness measurement space.[3] We define

$$\hat{k}(\boldsymbol{x}; \Lambda) = \hat{k}^{i(\boldsymbol{x};\Lambda)}(\boldsymbol{x}; \Lambda) + \hat{k}^{j(\boldsymbol{x};\Lambda)}(\boldsymbol{x}; \Lambda). \tag{28}$$

[3]The formalism of Eq. 27 forms the basis of deriving the Kernel Density Estimation [24].

$\hat{k}(\boldsymbol{x}; \Lambda)$ *implicitly* delineates an on-boundary cluster contained in $[-h_{\boldsymbol{x}}; h_{\boldsymbol{x}}]$ in the near-boundary-ness measurement space, and that only contains (a total of $\hat{k}(\boldsymbol{x}; \Lambda)$) samples with given class labels $C_{i(\boldsymbol{x};\Lambda)}$ and $C_{j(\boldsymbol{x};\Lambda)}$. As we describe in Section 4.4.2, $h_{\boldsymbol{x}}$ is usually a small value. The probability mass contained in a small volume is conserved through a change of variable. Therefore, the "implicitly defined on-boundary cluster that is centered on $f_{i(\boldsymbol{x};\Lambda)j(\boldsymbol{x};\Lambda)}(\cdot; \Lambda) = 0$ in the near-boundary-ness measurement space" corresponds to a "small region $\delta V(\boldsymbol{x})$ that is centered on $B(\Lambda)$ around $\boldsymbol{x}$ in the sample space." We denote by δV the union of these individual volumes $\delta V(\boldsymbol{x})$; Eqs. 27 and 28 thus adaptatively execute the filtering expressed by "$\cap\,(B(\Lambda) + \delta V)$" in Eq. 25, without having to explicitly generate anchors nor select their nearest training samples.

Figure 4 illustrates (28) with $M = 5$ on a two-class dataset. We illustrate the cluster $\mathcal{N}^{\mathrm{M}}(\boldsymbol{x})$ with a red circle, and the projection of (the perturbated) $\mathcal{N}^{\mathrm{M}}(\boldsymbol{x})$ onto $f(\cdot; \Lambda)$ with a zooming effect. The Parzen kernels are represented by black kernels centered on each projection of a perturbated sample in the cluster.

For convenience, we introduce $\forall \{k, l\} \in \hat{\mathcal{I}}^*$,

$$w_{kl}(\Lambda) = \sum_{\boldsymbol{x} \in \hat{B}^*(k,l)} \hat{k}(\boldsymbol{x}; \Lambda). \tag{29}$$

Each on-boundary implicit cluster contains $\hat{k}(\boldsymbol{x}; \Lambda)$ samples, and thus the probability of each cluster among all clusters for $\hat{B}^*(k, l)$ is $k(\boldsymbol{x}; \Lambda)/w_{kl}(\Lambda)$. Therefore, we can re-write (25) as the finite expectation of the local uncertainty function over the set of implicit on-boundary clusters described above:

$$\hat{U}^{(2)}(\Lambda) = \sum_{\{k,l\} \in \hat{\mathcal{I}}^*} \hat{P}(C_{kl}) \sum_{\boldsymbol{x} \in \hat{B}^*(k,l)} \frac{\hat{k}(\boldsymbol{x}; \Lambda)}{w_{kl}(\Lambda)} \times \hat{\lambda}_U\left(C_{i(\boldsymbol{x};\Lambda)j(\boldsymbol{x};\Lambda)}(\boldsymbol{x}; \Lambda) \mid C_{kl}\right), \tag{30}$$

where superscript "(2)" distinguishes this from the estimate obtained by Proposal 1 in Eq. 11. For convenience we define the estimated two-class boundary uncertainty: $\forall\{k,l\} \in \hat{\mathcal{I}}^*$,

$$\hat{U}_{kl}^{(2)}(\Lambda) = \sum_{\boldsymbol{x}\in\hat{B}^*(k,l)} \frac{\hat{k}(\boldsymbol{x};\Lambda)}{w_{kl}(\Lambda)}\hat{\lambda}_U \times \left(C_{i(\boldsymbol{x};\Lambda)j(\boldsymbol{x};\Lambda)}(\boldsymbol{x};\Lambda)\,|C_{kl}\right). \tag{31}$$

4.4.2 Determination of Parameters Defining On-Boundary Implicit Clusters

Given $\boldsymbol{x} \in \mathcal{T}$, the parameters $h_{\boldsymbol{x}}$ and M determine the on-boundary cluster that is implicitly defined by Eq. 27 (M determines the "radius" of $\mathcal{N}^{\mathrm{M}}(\boldsymbol{x})$, and then $h_{\boldsymbol{x}}$ is set in the direction orthogonal to $B(\Lambda)$). The appropriate setting of $h_{\boldsymbol{x}}$ temporarily requires us to consider a density estimation instead of a count estimation, since a count goes to infinity as the size of the dataset goes to infinity. The density estimation that corresponds to Eq. 27 is obtained by dividing (27) by $Mh_{\boldsymbol{x}}$ [24]. A possible way of setting $h_{\boldsymbol{x}}$ for optimal density estimation is to minimize the Mean Squared Integrated Error (MISE). This results in "Silverman's rule of thumb" [25]:

$$h_{\boldsymbol{x}} = 0.9\min\left(\hat{\sigma}_{\boldsymbol{x}}, \frac{\mathrm{IQR}_{\boldsymbol{x}}}{1.34}M^{-\frac{1}{5}}\right), \tag{32}$$

where $\hat{\sigma}_{\boldsymbol{x}}$ and $\mathrm{IQR}_{\boldsymbol{x}}$ correspond to the standard deviation and the interquartile range estimated on the distribution $\left\{f_{i(\boldsymbol{x};\Lambda)j(\boldsymbol{x};\Lambda)}(\widetilde{x'})\right\}_{\boldsymbol{x}'\in\mathcal{N}^{\mathrm{M}}(\boldsymbol{x})}$, respectively.

Given $m \in \{i(\boldsymbol{x};\Lambda), j(\boldsymbol{x};\Lambda)\}$, Eq. 32 may return $h_{\boldsymbol{x}} = 0$ if all values $\left\{f_{i(\boldsymbol{x};\Lambda)j(\boldsymbol{x};\Lambda)}(\widetilde{x'})\right\}_{\boldsymbol{x}'\in\mathcal{N}^{\mathrm{M}}(\boldsymbol{x})}$ are identical. In this case, provided this unique value is not zero (which we can safely assume), we set $\hat{k}^m(\boldsymbol{x};\Lambda) = 0$.

Equation 27 also depends on the hyperparameter M. We discuss how the setting of M is quite insensitive and is actually more controlled by practical constraints. Can the value of M be too high? Even if M is set quite high, the Parzen counts will automatically smooth out samples that are away from $B(\Lambda)$. Nevertheless, M should be as small as possible to *locally* estimate uncertainty. Can the value of M be too low? The estimation of $h_{\boldsymbol{x}}$ using Eq. 32 requires a minimum number of samples to be meaningful. These two questions compelled us to set a uniform $M = 40$ in practice (same value M for all the samples to obtain $\mathcal{N}^{\mathrm{M}}(\boldsymbol{x})$).

4.4.3 Estimation of $\mathcal{I}^*$ and its Related Quantities

By "quantities that are related to $\mathcal{I}^*$," we refer to $\forall\{k,l\} \in \mathcal{I}^*$, $B^*(k,l)$ and $P(C_{kl})$. $\mathcal{I}^*$ assumes knowledge of true class posterior probabilities. However, rather than requiring an accurate knowledge of true class posterior probabilities, $\mathcal{I}^*$ only requires knowledge of the indexes of the two highest class posterior probabilities at each sample. In other words, a rougher estimate of class posterior probabilities is sufficient for our purpose, so long as the two top indexes are correctly estimated.

Independently from the classifier model that we want to evaluate, in order to estimate class posterior probabilities, we propose using a generative classifier model that we explain in detail in Eq. 35. We denote by Λ_{gen} the trained classifier status of this generative classifier model. Then, $\forall\boldsymbol{x} \in \mathcal{T}$, we estimate $\{i^*(\boldsymbol{x}), j^*(\boldsymbol{x})\}$ as $\left\{i(\boldsymbol{x};\Lambda_{gen}), j(\boldsymbol{x};\Lambda_{gen})\right\}$. Application of Eq. 13 to these estimated pairs of indexes gives $\hat{\mathcal{I}}^*$. Then, $\forall\{k,l\} \in \hat{\mathcal{I}}^*$,

$$\hat{B}^*(k,l) = \left\{\boldsymbol{x} \in \mathcal{X} \,\middle|\, \{\hat{i}^*(\boldsymbol{x}), \hat{j}^*(\boldsymbol{x})\} = \{k,l\}\right\}, \tag{33}$$

and then

$$\hat{P}(C_{kl}) = \frac{|\hat{B}^*(k,l)|}{N}. \tag{34}$$

Regarding the choice of a generative classifier model, the above considerations imply that a simple model may be enough for our purpose. For this reason, we chose the following prototype-based classifier (PBC), which has low computation costs. $\forall j \in [\![1, J]\!]$, we denote by $\mathcal{T}_j$ the set of training samples with given class label C_j, namely $\mathcal{T}_j = \{\boldsymbol{x} \in \mathcal{T} | y(\boldsymbol{x}) = j\}$. Our PBC represents $\mathcal{T}_j$ by K_j prototypes. We denote by $\left\{p_j^k\right\}_{k\in[\![1,K_j]\!]}$ the corresponding set that consists of K_j prototypes. We obtain $\left\{p_j^k\right\}_{k\in[\![1,K_j]\!]}$ by performing K-means clustering on $\mathcal{T}_j$. The discriminant functions for such PBC are $\forall j \in [\![1, J]\!]$, $\forall\boldsymbol{x} \in \mathcal{X}$,

$$g_j(\boldsymbol{x};\Lambda_{gen}) = -\min_{k\in[\![1,K_j]\!]} ||\boldsymbol{x} - p_j^k||, \tag{35}$$

where Λ_{gen} corresponds to the set of prototypes obtained by class-wise K-means clustering.

As one possibility, $\forall j \in [\![1, J]\!]$, we set the value of K_j based on the Akaike Information Criterion (AIC). AIC can be used to determine the number of a Gaussian mixture model, and a K-means algorithm is a particular instance of the Classification Expectation algorithm for a Gaussian mixture model with equal mixture weights and equal isotropic variances [26]. We used existing results to adapt AIC to each class-wise K-means [26]. We give these results in the Appendix. Algorithm 2 summarizes the estimation of $\mathcal{I}^*$ and that of its related quantities.

Algorithm 2 Estimation of $\hat{\mathcal{I}}^*$ and $\forall\{k,l\} \in \hat{\mathcal{I}}^*$, estimation of $B^*(k,l)$ and $P(C_{kl})$.

Input: Training set $\mathcal{T}$, $K_{min} \leftarrow 1$, $K_{max} \leftarrow 40$

`/* Training of the PBC classifier */`

for $j \in [\![1, J]\!]$ **do**
  for $K \in [\![K_{min}, K_{max}]\!]$ **do**
    Apply K-means clustering to class C_j to compute $\{p_j^k\}_{k\in[\![1,K]\!]}$;
  end
  Determine the optimal K_j (Section Appendix);
  Apply K-means clustering to class C_j with the value K_j to compute $\{p_j^k\}_{k\in[\![1,K_j]\!]}$;
end
For j Denote by Λ_{gen} the resulting PBC classifier status;

`/* Classification by the PBC classifier */`

for $\boldsymbol{x} \in \mathcal{T}$ **do**
  Classify $\boldsymbol{x}$ with the $PBC(\Lambda_{gen})$;
  $\left\{\hat{i}^*(\boldsymbol{x}), \hat{j}^*(\boldsymbol{x})\right\} \leftarrow \left\{i(\boldsymbol{x};\Lambda_{gen}), j(\boldsymbol{x};\Lambda_{gen})\right\}$;
end
For j $\hat{\mathcal{I}}^* \leftarrow \left\{\hat{i}^*(\boldsymbol{x}), \hat{j}^*(\boldsymbol{x})\right\}_{\boldsymbol{x}\in\mathcal{T}}$;
$\forall\{k,l\} \in \hat{\mathcal{I}}^*$, compute $\hat{B}^*(k,l)$ (33);
$\forall\{k,l\} \in \hat{\mathcal{I}}^*$, compute $\hat{P}(C_{kl})$ (34);
return $\left\{\hat{P}(C_{kl})\right\}_{\{k,l\}\in\hat{\mathcal{I}}^*}, \left\{\hat{B}^*(k,l)\right\}_{\{k,l\}\in\hat{\mathcal{I}}^*}, \hat{\mathcal{I}}^*$

4.4.4 Estimation of Boundary-Wise Cost

Given $\{k,l\} \in \hat{\mathcal{I}}^*$ and $\boldsymbol{x} \in T$, the estimated boundary-wise cost that appears in Eq. 25 is

$$\hat{\lambda}_U\left(C_{i(\boldsymbol{x};\Lambda)j(\boldsymbol{x};\Lambda)}(\boldsymbol{x};\Lambda)\,|C_{kl}\right) = \begin{cases} -\hat{U}(\boldsymbol{x};\Lambda) & \text{if } \{\hat{i}^*(\boldsymbol{x}), \hat{j}^*(\boldsymbol{x})\} = \{k,l\}, \\ -U_{min}, & \text{otherwise.} \end{cases} \tag{36}$$

Instead of the binary Shannon entropy used by Proposal 1, Proposal 2 uses a triangle-shaped function (orange curve in Fig. 2) as a local uncertainty function. This corresponds to the following local uncertainty: $\forall m \in \{i(\boldsymbol{x};\Lambda), j(\boldsymbol{x};\Lambda)\}$,

$$\hat{U}(\boldsymbol{x};\Lambda) = 1 - \left|2\hat{P}(C_m|\boldsymbol{x};\Lambda) - 1\right|. \tag{37}$$

The reason for this choice was to achieve a more neutral penalization of non-uncertainty than the binary Shannon entropy. The binary Shannon entropy weakly penalizes non-uncertainty in a wide range of [0.3; 0.7] around 0.5, while outside this range it strongly penalizes non-uncertainty.

Just as done in Proposal 1, Proposal 2 estimates class posterior probabilities using the kNN regression rule (Section 3.2). $\forall m \in \{i(\boldsymbol{x};\Lambda), j(\boldsymbol{x};\Lambda)\}$, the kNN regression applied in the near-boundary-ness measurement space gives

$$\hat{P}(C_m|\boldsymbol{x};\Lambda) = \begin{cases} \dfrac{\hat{k}^m(\boldsymbol{x};\Lambda)}{\hat{k}(\boldsymbol{x};\Lambda)} & \text{if } \hat{k}(\boldsymbol{x};\Lambda) > 0, \\ 0, & \text{if } \hat{k}(\boldsymbol{x};\Lambda) = 0. \end{cases} \tag{38}$$

When we input $\hat{P}(C_m|\boldsymbol{x};\Lambda)$ in the local uncertainty function defined in Eq. 37, the first and second cases of the branching in Eq. 38 result in the first and second cases of the branching in Eq. 36, respectively.

$\forall j \in [\![1, J]\!]$, we denote by $\hat{P}_j$ the estimate of class prior probability, and N_j is the number of training samples whose given class label is C_j: $\hat{P}_j = N_j/N$. In order to address class imbalance [29], Proposal 1 replaced each estimate $\hat{P}(C_l|\boldsymbol{x};\Lambda)$ by $\hat{P}(C_l|\boldsymbol{x};\Lambda)/\hat{P}_l$ [18]. Proposal 2 does not perform such adjustment. Indeed, if we assume an adequate sampling, then unequal class prior probabilities are implicitly handled by the kNN regression.

4.5 Two-Class Boundary Uncertainty Estimate $\hat{U}_{kl}(\Lambda)$ in Separability Case

This section assumes a pair $\{k,l\} \in \hat{\mathcal{I}}^*$ and refines (31) to handle the case where $w_{kl}(\Lambda) = 0$ (29). This case occurs when class distributions in the sample space are well-separated.

Let us assume that class distributions are separated by a nearly empty region in the sample space. In this case, the Bayes boundary B^* is located in the empty region. We consider the three possible cases of $B(\Lambda)$ and illustrate them in Fig. 5.

Case **1**: $B(\Lambda)$ (green curve) is close to B^* (red curve). In this case, we are likely to get $w_{kl}(\Lambda) = 0$. As a result, Eq. 31 cannot be computed. However, we would like Proposal 2 to assign a best boundary uncertainty value U_{max} to $U_{kl}^{(2)}$. We notice that all samples in $\hat{B}^*(k,l)$ are correctly classified by such a $B(\Lambda)$.

Case **2**: $B(\Lambda)$ is so biased that it crosses the class distributions. Proposal 2 returns the worst values of local uncertainties whenever it crosses the class distributions, since well-separated class distributions locally have only one class label with high probability (in Panel **2**, every cluster $\mathcal{N}^{\mathrm{M}}(\boldsymbol{x})$ along $B(\Lambda)$ contains either all yellow or all purple given class labels). As a result, Eq. 31 can return the worst value U_{min}, which is already what we want.

Case **3**: $B(\Lambda)$ is so biased that it lies in an empty region outside the range of the class distributions. Similarly to Case **1**, Eq. 31 cannot be computed.

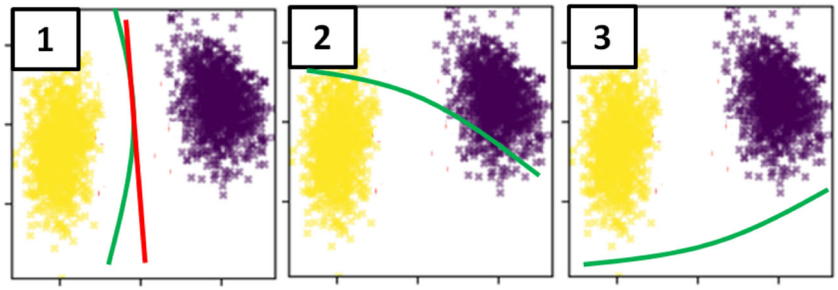

Figure 5 Illustration of separability in the sample space on two classes C_i and C_j of a two-dimensional dataset called GMM_separable. The given class labels are represented by yellow and purple crosses, respectively. The class distributions of GMM_separable are separated by an empty region. B^* is represented by a red curve in Panel **1**. We represent different $B(\Lambda)$s by a green curve in Panels **1**, **2**, and **3**. These three panels respectively correspond to Cases **1**, **2**, and **3**, which we describe in Section 4.5.

> However, we would like Proposal 2 to assign a worst value U_{min}. We note that samples of either class C_k or class C_l are all misclassified because, in this case, $B(\Lambda)$ assigns all samples of $\hat{B}^*(k,l)$ to the same class.

We must refine (31) so that it can output a reasonable value of $\hat{U}_{kl}(\Lambda)$ even in Cases **1** and **3**. Based on the observation of Cases **1** and **3**, we can quite simply identify these two cases by checking the classification error rate on $\widetilde{\mathcal{T}}$. We do not check the error rate on $\mathcal{T}$ but instead on $\widetilde{\mathcal{T}}$, since the error rate itself directly depends on the values of the near-boundary-ness measurement. As described in Section 4.2.2, we always use $f(\widetilde{\mathcal{T}};\Lambda)$ instead of $f(\mathcal{T};\Lambda)$. We denote by $\widetilde{L}_{tr}^{kl}(\Lambda)$ the classification error rate on the perturbated version of $\hat{B}^*(k,l)$ (namely, $\left\{\boldsymbol{x}+r_{\boldsymbol{x}}||\boldsymbol{x}-\boldsymbol{x}^{(1)}||\right\}_{\boldsymbol{x}\in\hat{B}^*(k,l)}$). This results in the following extended definition of $\hat{U}_{kl}(\Lambda)$ that we use instead of Eq. 31:

$$\hat{U}_{kl}^{(3)}(\Lambda)=\begin{cases} U_{max}, & \text{if } w_{kl}(\Lambda)=0 \text{ and } \widetilde{L}_{tr}^{kl}=0,\\ U_{min}, & \text{if } w_{kl}(\Lambda)=0 \text{ and } \widetilde{L}_{tr}^{kl}>0,\\ \sum\limits_{\boldsymbol{x}\in\hat{B}^*(k,l)} \hat{\lambda}_U(C_{i(\boldsymbol{x};\Lambda)j(\boldsymbol{x};\Lambda)}(\boldsymbol{x};\Lambda)|C_{kl}) & \\ \qquad\times\dfrac{\hat{k}(\boldsymbol{x};\Lambda)}{w_{kl}(\Lambda)}, & \text{otherwise.}\end{cases} \tag{39}$$

We then re-define the boundary uncertainty estimate as

$$\hat{U}^{(3)}(\Lambda)=\sum_{\{k,l\}\in\hat{\mathcal{I}}^*}\hat{P}(C_{kl})\hat{U}_{kl}^{(3)}(\Lambda). \tag{40}$$

Superscript "(3)" distinguishes this from the estimate defined by Eq. 30, but we still consider it a part of Proposal "2".

4.6 Implementation of Proposal 2

Algorithm 3 summarizes the implementation of Proposal 2.

Algorithm 3 Our current Proposal 2: boundary uncertainty estimation of L classifier statuses $\Lambda_{\mathrm{TR}}=\{\Lambda_m\}_{m\in[\![1,L]\!]}$.

Input: Training set $\mathcal{T}$, set of classifier statuses $\Lambda_{\mathrm{TR}}=\{\Lambda_m\}_{m\in[\![1,L]\!]}$, $M\leftarrow 40$
Output: $\Lambda_{selected}=\arg\max_{\Lambda\in\Lambda_{\mathrm{TR}}}\hat{U}(\Lambda)$

```
/* Preliminary treatments
   (independent of Λ_TR)                          */
```
$\forall\boldsymbol{x}\in\mathcal{T}$, store $||\boldsymbol{x}-\boldsymbol{x}^{(1)}||$, and store the sample indexes of $\mathcal{N}^{\mathrm{M}}(\boldsymbol{x})$;
Compute $\widetilde{\mathcal{T}}$ (Algorithm 1);
Estimate $\hat{\mathcal{I}}^*$, and $\forall\{k,l\}\in\hat{\mathcal{I}}^*$, compute of $\hat{P}(C_{kl})$ and $\hat{B}^*(k,l)$ (Algorithm 2);

```
/* Treatments exclusive to each
   candidate classifier status                    */
```
for $\Lambda\in\Lambda_{TR}$ **do**
  for $\boldsymbol{x}\in\mathcal{T}$ **do**
    Compute $h_{\boldsymbol{x}}$ (32);
    $\forall m\in\{i(\boldsymbol{x};\Lambda),\ j(\boldsymbol{x};\Lambda)\}$, compute $k_m(\boldsymbol{x};\Lambda)$ (27);
    $\forall m\in\{i(\boldsymbol{x};\Lambda),\ j(\boldsymbol{x};\Lambda)\}$, compute $\hat{P}(C_m|\boldsymbol{x};\Lambda)$ (38);
    Compute $\hat{U}(\boldsymbol{x};\Lambda)$ (37);
    Compute $\hat{\lambda}_U\left(C_{i(\boldsymbol{x};\Lambda)j(\boldsymbol{x};\Lambda)}(\boldsymbol{x};\Lambda)\middle|C_{kl}\right)$ (36);
  end
  for $\{k,l\}\in\hat{\mathcal{I}}^*$ **do**
    Compute $w_{kl}(\Lambda)$ (29);
    Compute $\hat{U}_{kl}^{(3)}(\Lambda)$ (39);
  end
  Compute $\hat{U}^{(3)}(\Lambda)$ (40) ;
end
return $\arg\max_{\Lambda\in\Lambda_{\mathrm{TR}}}\hat{U}^{(3)}(\Lambda)$

5 Time Costs of Classifier Evaluation

One of the goals of Proposal 2 is to achieve scalability, since the scalability of resampling-based classifier evaluation methods such as CV is limited. Therefore, this section describes the time costs of Proposal 2 to evaluate $\Lambda_{\mathrm{TR}} = \{\Lambda_m\}_{m\in[\![1,L]\!]}$. We denote the cost of one addition, multiplication, computation of the exponential function, comparison, and classification as $c_{add}, c_{mul}, c_{exp}, c_{comp}, c_{CL}(d, J, \Lambda)$, respectively. $c_{CL}(d, J, \Lambda)$ increases with d, J and with the number of parameters in Λ. For comparison, we will also describe the time costs of Proposal 1.

5.1 Time Costs of Proposal 2

We denote by $c^{(2)}_{shared}$ the costs of Proposal 2 that are shared across all classifier statuses, or in other words, that appear only once. Here, superscript "(2)" distinguishes them from the time costs of Proposal 1. We denote by $c^{(2)}_{\Lambda}$ the costs that are exclusive to the evaluation of one classifier status Λ. The costs to estimate L classifier statuses $\Lambda_{\mathrm{TR}} = \{\Lambda_m\}_{m\in[\![1,L]\!]}$ is

$$c^{(2)}(\Lambda_1, \cdots, \Lambda_L) = c^{(2)}_{shared} + \sum_{m=1}^{L} c^{(2)}(\Lambda_m). \tag{41}$$

5.1.1 Estimation of $c^{(2)}_{shared}$

Regardless of the number of candidate statuses Λ to evaluate, Proposal 2 requires us to compute and store the set of clusters $\{\mathcal{N}^{\mathrm{M}}(\boldsymbol{x})\}_{\boldsymbol{x}\in\mathcal{T}}$, as well as the set of distances $\{||\boldsymbol{x} - \boldsymbol{x}^{(1)}||\}_{\boldsymbol{x}\in\mathcal{T}}$. We obtained these clusters and distances using a KDTree, so the total cost of the construction and search of the nearest neighbors is $O(N\log(N))$ distance computations [30]. The cost of one distance computation in $\mathcal{X}$ is $O(d)(c_{add} + c_{mul})$.

As summarized in Algorithm 3, the generation of $\widetilde{\mathcal{T}}$ is done only once. The cost of generating $\widetilde{\mathcal{T}}$ is $O(dN)c_{add}$.

K-means clustering costs KN distance computations at each K-means iteration. A distance computation costs $O(d)(c_{add}+c_{mul})$. We denote by i_K the imposed maximum number of iterations of K-means. If we assume that each class contains roughly the same number of training samples N/J, then the total time costs of obtaining our trained PBC in Algorithm 2 are $O(i_K dN)(c_{add} + c_{mul})$. We repeat these costs as many times as we attempt different values for the number of prototypes per class. We denote by R_K the number of tried values of K. The preliminary time costs of Proposal 2 are thus

$$c^{(2)}_{shared} = c^{(2,shared)}_{add} + c^{(2,shared)}_{mul}, \tag{42}$$

where

$$\begin{aligned} c^{(2,shared)}_{add} &= O(dNR_K i_K)c_{add} \\ c^{(2,shared)}_{mul} &= O(dNR_K i_K)c_{mul}. \end{aligned}$$

5.1.2 Estimation of $c^{(2)}_{\Lambda}$

Given one candidate classifier status Λ, finding the top indexes $i(\boldsymbol{x};\Lambda)$, $j(\boldsymbol{x};\Lambda)$ for the N training samples using QuickSort requires $O(NJ\log(J))$ comparisons.

Then, the cost of the set of operations defined by Eqs. 27, 32, 38 is *independent* of J and d. As stated above, it only requires simple arithmetic operations (e.g., mean on M samples, multiplications, application of the exponential function) on a two-dimensional array of size at most (N, M), whose rows each correspond to a training sample $\boldsymbol{x} \in \mathcal{T}$ and whose columns each correspond to a single perturbated element of $\mathcal{N}^{\mathrm{M}}(\boldsymbol{x})$. The cost of computing near-boundary-ness measurement values on $\widetilde{\mathcal{T}}$ is the cost of classifying $\widetilde{T}$.

$$c^{(2)}(\Lambda) = c^{(2)}_{add} + c^{(2)}_{mul} + c^{(2)}_{comp} + c^{(2)}_{exp} + c^{(2)}_{CL}(d, J, \Lambda), \tag{43}$$

where

$$\begin{aligned} c^{(2)}_{add} &= O(NM)c_{add} \\ c^{(2)}_{mul} &= O(NM)c_{mul} \\ c^{(2)}_{comp} &= O(NJ\log(J))c_{comp} \\ c^{(2)}_{exp} &= O(NM)c_{exp} \\ c^{(2)}_{CL}(d, J, \Lambda) &= O(N)c_{CL}(d, J, \Lambda). \end{aligned}$$

Overall, the cost of Proposal 2 scales reasonably with J, and d mainly shows up only once when computing the clusters $\mathcal{N}^{\mathrm{M}}(\boldsymbol{x})$ in the sample space. Obtaining the rest of the costs in $c^{(2)}_{\Lambda}$ boils down to operations on the one-dimensional data obtained from the projection of each (perturbated) $\mathcal{N}^{\mathrm{M}}(\boldsymbol{x})$ on the axis $f_{i(\boldsymbol{x};\Lambda)j(\boldsymbol{x};\Lambda)}(\cdot;\Lambda)$. The operations defined by Eqs. 27, 32, 38 define simple operations on N arrays that each contain exactly M elements. These operations can be efficiently parallelized on the hardware as operations on two-dimensional arrays of size (N, M), despite whether it's on the CPU[4] or the GPU. This is another desirable characteristic of Proposal 2 in terms of scalability.

5.2 Time Costs of Proposal 1

We denote by $c^{(1)}(\Lambda)$ the time costs to estimate $U(\Lambda)$ for a single candidate classifier status Λ. We denote by $c^{(1,1)}(\Lambda)$ and $c^{(1,2)}(\Lambda)$ the time costs of Step 1 and Step 2 of Proposal 1, respectively. We detail these costs in

[4] https://numpy.org/

the following sections. Step 1 and Step 2 are executed independently, hence

$$c^{(1)}(\Lambda) = c^{(1,1)}(\Lambda) + c^{(1,2)}(\Lambda). \tag{44}$$

The evaluation of each candidate classifier status Λ is performed independently by Proposal 1, so the costs of estimating L classifier statuses $\Lambda_1, \cdots, \Lambda_L$ are

$$c^{(1)}(\Lambda_1, \cdots, \Lambda_L) = \sum_{m=1}^{L} c^{(1)}(\Lambda_m). \tag{45}$$

5.2.1 Time Costs of Step 1 in Proposal 1

Here, we take up the notations from Section 3.3. The cost of Step 1 is the cost of generating R_A anchors and then searching for their nearest neighbor. Given a random segment $[\boldsymbol{x}; \boldsymbol{x}']$, the search for an anchor between $\boldsymbol{x}$ and $\boldsymbol{x}'$ is executed by dichotomy, whose maximum number of iterations we denote by i_{max}. Each dichotomy iteration generates one artificial sample. Given one such artificial sample that we denote by $\boldsymbol{x}_c$, the corresponding treatments are the interpolation between $\boldsymbol{x}$ and $\boldsymbol{x}'$ to obtain $\boldsymbol{x}_c$ (cost: $O(d)c_{add}$), the classification of $\boldsymbol{x}_c$ (cost: $c_{CL}(d, J, \Lambda)$), the sorting among the J discriminant function values to select $\{i(\boldsymbol{x}_c; \Lambda), j(\boldsymbol{x}_c; \Lambda)\}$ (cost: $O(J \log(J))c_{comp}$), and then one subtraction $g_{i(\boldsymbol{x};\Lambda)}(\boldsymbol{x}; \Lambda) - g_{j(\boldsymbol{x};\Lambda)}(\boldsymbol{x}; \Lambda)$ (negligible cost: c_{add}).

The cost of searching for and generating one anchor is thus $O\left(i_{max}(dc_{add} + c_{CL}(d, J, \Lambda) + J \log(J)c_{comp})\right)$. Then, searching for the nearest neighbor of an anchor with a KDTree costs $O(\log(N))$ distance computations [30], namely $O(d)(c_{add} + c_{mul})$. The cost of Step 1 of Proposal 1 is thus as follows:

$$c^{(1,1)}(\Lambda) = c^{(1)}_{add} + c^{(1)}_{mul} + c^{(1)}_{comp} + c^{(1)}_{CL}(d, J, \Lambda), \tag{46}$$

where

$$\begin{aligned} c^{(1,1)}_{add} &= O\left(R_A i_{max} d + R_A \log(N)\right) c_{add} \\ c^{(1,1)}_{mul} &= O\left(R_A \log(N)\right) c_{mul} \\ c^{(1,1)}_{comp} &= O(R_A i_{max} J \log(J)) c_{comp} \\ c^{(1,1)}_{CL}(d, J, \Lambda) &= O(R_A i_{max}) c_{CL}(d, J, \Lambda). \end{aligned}$$

5.2.2 Time Costs of Step 2 in Proposal 1

We now take up the notations from Section 3.4. We assume for simplification that at the end of the TDC, each cluster contains N_{max} samples. During the first step of a given TDC, the main cost of the 2-means clustering is to compute the distance between each of the N samples and the two cluster centroids. This results in $2N$ distance computations. There are i_K iterations in 2-means clustering, so the cost of the 2-means clustering on N samples is $O(2i_K dN)c_{add}$.

During the second step of a given TDC, 2-means clustering is applied to each of the two clusters that consist of $N/2$ samples. This corresponds to the computation of two times $2N/2$ distances, namely $2N$ distance computations overall. More generally, we note that $2N$ distances are computed regardless of the dividing step in a given TDC, and thus the cost of each TDC step is always $O(2i_K dN)(c_{add} + c_{mul})$.

To obtain the total cost of a given TDC, we estimate the number of TDC steps that we denote by i_T for convenience. Assuming clusters of roughly equal size within each step, i_T satisfies $N/2^{i_T} = N_{max}$, and thus $i_T = \lceil \log(N/N_{max}) \rceil + 1$. The cost of Step 2 of Proposal 1 is therefore

$$c^{(1,2)}(\Lambda) = c^{(1,2)}_{add} + c^{(1,2)}_{mul}, \tag{47}$$

where

$$\begin{aligned} c^{(1,2)}_{add} &= O\left(R_T i_K dN \log(N/N_{max})\right) c_{add} \\ c^{(1,2)}_{mul} &= O\left(R_T i_K dN \log(N/N_{max})\right) c_{mul}. \end{aligned}$$

The time costs of Step 1 and Step 2 in Proposal 1 are simply added up, and we group operations by elementary costs for convenience:

$$\begin{aligned} c^{(1)}_{add} &= c^{(1,1)}_{add} + c^{(1,2)}_{add} \\ c^{(1)}_{mul} &= c^{(1,1)}_{mul} + c^{(1,2)}_{mul} \\ c^{(1)}_{comp} &= c^{(1,1)}_{add} \\ c^{(1)}_{CL}(d, J, \Lambda) &= c^{(1,1)}_{CL}(d, J, \Lambda). \end{aligned}$$

6 Experiments

The goal of our experiments is three-fold. Goal (a): Assess whether Proposal 2 accurately estimates the boundary uncertainty defined by Eq. 15 even without costly traditional estimation methods such as re-sampling. Goal (b): Confirm once more that boundary uncertainty is a relevant quantity to perform classifier evaluation and classifier selection. Goal (c): Compare Proposal 2 in terms of the accuracy of classifier selection and scalability with existing widely applicable and powerful methods (HO and CV), but also with Proposal 1.

6.1 Classifiers

In this early stage of our research, we focus on the careful design and analysis of boundary uncertainty estimation and thus restrict our experiments to the selection of a single hyperparameter.

We assessed Proposal 2 in the evaluation and then selection of classifier statuses of the Gaussian kernel SVM classifier for two reasons. First, this classifier has only two

hyperparameters: the Gaussian kernel width and the regularization coefficient. Second, we can easily obtain extreme cases of insufficient or excessive representation capability by controlling the values of these two hyperparameters, owing to the infinite VC capacity of Gaussian kernel SVMs and to the possibility of analytically obtaining the global minimum of training objectives of SVMs [27]. Gaussian kernel SVMs thus provide a simple way of analyzing Proposal 2 over a wide range of classifier boundary cases. We varied the hyperparameter values by powers of 2 in order to easily sweep a large range of values, while avoiding an excessively rough search (as powers of 10 may result in), e.g., $2^{-15}, 2^{-14}, \cdots, 2^{14}, 2^{15}$.

To simplify the analysis, we preliminarily selected the regularization coefficient using CV [18]. Our experiments focused on the selection of the Gaussian kernel width. Following the notations of the SVM implementation that we used,[5] we denote by γ the inverse of the Gaussian kernel width. A higher γ corresponds to a higher capacity of the classifier to draw a more complex $B(\Lambda)$. For each value of γ, we performed full SVM training, and then we evaluated the resulting classifier status with Proposal 2, HO, and CV.

Just to ensure that Proposal 2 can also perform evaluation of other classifier models, we also succinctly assessed Proposal 2 on a MultiLayer Perceptron (MLP). The goal in our experimental setting was not really to achieve state-of-the-art performance but rather to perform analysis of Proposal 2 on MLP. Therefore, we simply considered an MLP with two hidden layers of 128 and 64 units, respectively. Both layers use ReLu activation functions. The output layer uses cross entropy as the objective function that we optimized with RMSProp, using a library available online.[6] As an MLP selection experiment, we performed an early stopping experiment, namely we looked for the optimal number of training epochs inside a single instance of classifier training.

6.2 Datasets

We performed our experiments on real-world benchmark datasets available online[7] that are quite small and basic but that provide some diversity in terms of dimensionality, number of samples, nature of the features, and class overlap. Furthermore, we prepared three synthetic two-dimensional and two-class datasets using Gaussian Mixture Models: GMM, GMM_separable, and GMM_inclusion . We illustrate these datasets in Figs. 10, 14, and 15, respectively. For the GMM dataset, we generated a testing set from the same mother distribution as $\mathcal{T}$ to provide more exhaustive experimental results. Owing to its larger number of available samples, we could also afford to split the Letter Recognition dataset into a training set and a testing set of equal size.

6.3 Data Preparation

On all of the datasets, we standardized independently each feature by removing the mean and then scaling to unit variance across the entire training set.[8] For the datasets that also have a testing set, we applied the same standardization to the testing set, based on the means and variances that were measured from the training set.

To obtain $\widetilde{T}$, we applied Algorithm 1 on the standardized data, using the nearest distances $||x - x^{(1)}||$ that were measured on the standardized data.

The two hyperparameters of Proposal 2 are M and K, which appear in Sections 4.4.1 and 4.4.3, respectively. For all of the datasets, we set M to 40 because a meaningful computation of Eq. 32 seems to require at least 30-40 samples. In all of the datasets, we searched for K in the range [1, 40] with a step of 2, and then we selected its value as described in the Appendix.

6.4 Accuracy of Boundary Uncertainty-Based Classifier Selection

6.4.1 Exhaustive Results on Synthetic Data

In Fig. 6, we display the SVM selection results for the GMM dataset. There are two ideal alternatives for assessing Goal (a) (for our three goals, see the beginning of Section 6). One alternative is to visualize the similarity between B^* and $B(\Lambda)$ in the multidimensional sample space; however, this is not practical. Another alternative is to use a large testing set (not to be confused with the validation sets that are used by HO and CV) as a reference truth that gives estimates as close as possible to the expected values. In this case, for each Λ that is preliminarily obtained by training the classifier on $\mathcal{T}$, we obtain $\hat{U}_{tr}(\Lambda)$ by using $\mathcal{T}$ as input of Algorithm 3, and then we separately obtain $\hat{U}_{te}(\Lambda)$ by using the testing set as input of Algorithm 3. If Proposal 2 is an accurate boundary uncertainty estimation method, then Proposal 2 should satisfy $\forall \Lambda, \hat{U}_{tr}(\Lambda) \approx \hat{U}_{te}(\Lambda)$.

In Panel **3** of Fig. 6, we plot $-\hat{U}_{tr}(\Lambda)$ (blue) and $-\hat{U}_{te}(\Lambda)$ (black) against γ. We can see that the blue and black curves in Fig. 6 are nearly identical. This seems to validate the accurate estimation of the boundary uncertainty by Proposal 2 without relying on resampling methods, and thus it achieves Goal (a). We note that the boundary

[5] https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

[6] https://keras.io/

[7] UCI repository: https://archive.ics.uci.edu/ml/index.php

[8] https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

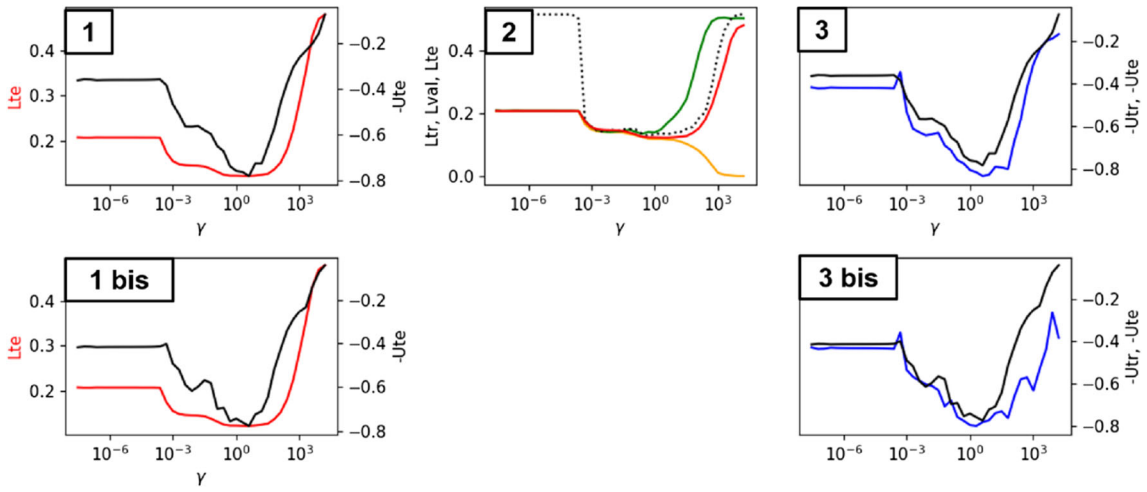

Figure 6 SVM classifier selection results using Proposal 2 on the synthetic GMM dataset. Horizontal axis for Panels **1** to **3**: inverse γ of the Gaussian kernel width. Panel **1**: $\hat{L}_{te}(\Lambda)$ (red) on the left vertical axis, $-\hat{U}_{te}(\Lambda)$ (black) on the right vertical axis. Panel **2**: L_{HO} (dotted black), $\hat{L}_{tr}(\Lambda)$ (orange), $\hat{L}_{val}(\Lambda)$ (green) on the left vertical axis. Panel **3**: $-\hat{U}_{tr}(\Lambda)$ (blue), $-\hat{U}_{te}(\Lambda)$ (black) on the right-hand axis. Following the same conventions as Panels **1** and **3**, Panels **1bis** and **3bis** were obtained by applying Proposal 2 without using $\widetilde{\mathcal{T}}$, namely by computing the values of the near-boundary-ness measurement on $\mathcal{T}$.

uncertainty estimation in Proposal 2 essentially relies on Eqs. 27 and 32. Accurate and scalable estimation of ratios for multidimensional data such as in the kNN regression usually requires advanced estimation methods [28]. In our case, accurate estimation of the ratio formed by Eqs. 27 and 28 with the accuracy shown in Fig. 6 may be explained by the local reduction of the boundary uncertainty task to the single dimension formed by the near-boundary-ness measurement. This local reduction enabled the use of analytic estimation rules such as Eq. 32, which may be sufficiently accurate on one-dimensional data by simply requiring training data.

We can assess Goal (b) by comparing the behavior of $-U(\Lambda)$ with the behavior of well-established classifier evaluation metrics such as error probability. A more optimal classifier status corresponds to a lower error probability, and to a lower sign-reversed boundary uncertainty $-U(\Lambda)$. We can thus expect the two evaluation metrics to mutually confirm their validity by following the same trends, and especially by hitting a minimum for the same Λ (hopefully the Bayes error status). Incidentally, we can assess Goal (a) by checking whether $-U(\Lambda)$ reaches its minimum value -1 when the minimum error probability is achieved. We thus check $-\hat{U}_{te}(\Lambda)$ and $\hat{L}_{te}(\Lambda)$ against γ for the GMM dataset on Panel **1** of Fig. 7, and we see that Goal (b) is achieved.

In Panel **1**, $-\hat{U}_{te}(\Lambda)$ actually shows a sharper minimum than $\hat{L}_{te}(\Lambda)$. This sharper trend can be explained by the focus of boundary uncertainty precisely on the classifier boundary: Overall similar error probabilities can correspond to quite different classifier boundaries. Boundary uncertainty-based classifier evaluation may discriminate between classifier statuses more finely than error probability, provided that the estimation method is accurate enough. This implies that even though we can expect some similar trend between $-\hat{U}_{te}(\Lambda)$ and $\hat{L}_{te}(\Lambda)$, expecting exactly the same trend between the two curves is not necessarily the goal of boundary uncertainty. We emphasize that the purpose here is **not** to provide exactly the same trend as $\hat{L}_{te}(\Lambda)$, and that $\hat{L}_{te}(\Lambda)$ is simply used as a default informative reference. This is the reason why we do **not** try

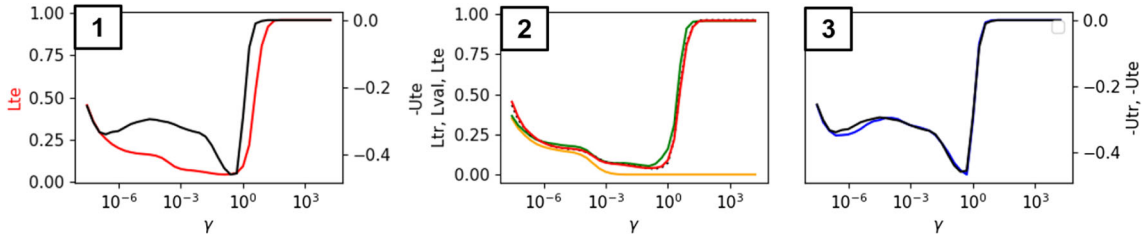

Figure 7 SVM classifier selection results using Proposal 2 on the Letter Recognition dataset. Horizontal axis for Panels **1** to **3**: inverse γ of the Gaussian kernel width. Panel **1**: $\hat{L}_{te}(\Lambda)$ (red) on the left vertical axis, $-\hat{U}_{te}(\Lambda)$ (black) on the right vertical axis. Panel **2**: L_{HO} (dotted black), $\hat{L}_{tr}(\Lambda)$ (orange), and $\hat{L}_{val}(\Lambda)$ (green) on the left vertical axis. Panel **3**: $-\hat{U}_{tr}(\Lambda)$ (blue) and $-\hat{U}_{te}(\Lambda)$ (black) on the right-hand axis.

to quantitatively measure the correlation between $-\hat{U}_{te}(\Lambda)$ and $\hat{L}_{te}(\Lambda)$, since this might imply that our goal is to fit $\hat{L}_{te}(\Lambda)$ with $-\hat{U}_{te}(\Lambda)$.

In practice, we may not always have access to a testing set. Instead, CV may be used to estimate the error probability. Therefore, we also assessed CV-based estimates of the classification error probability. Our experiments adopted stratified CV[9] with a high number of folds (10 to 40) to get closer to LOO, whose asymptotic unbiasedness is proven. We also tried HO, by viewing HO as a more practical competitor of Proposal 2 than CV in terms of computation cost. For HO, we applied a ratio (75%, 25%) for the split (train, validation).

We illustrate CV and HO in Panel **2** of Fig. 6: average $\hat{L}_{tr}(\Lambda)$ over the error probability estimated on the training folds of CV (orange); average $\hat{L}_{val}(\Lambda)$ over the error probability estimated on the validation folds of CV (green); error probability estimated on a holdout validation set $\hat{L}_{\mathrm{HO}}(\Lambda)$ (dotted-black); error probability estimated on a large testing set $\hat{L}_{te}(\Lambda)$ (red); sign-reversed boundary uncertainty estimated on the training set $-\hat{U}_{tr}(\Lambda)$ (blue); and sign-reversed boundary uncertainty estimated on a large testing set $-\hat{U}_{te}(\Lambda)$ (black).

Curiously, $\hat{L}_{\mathrm{HO}}(\Lambda)$ (dotted black) seems closer to $\hat{L}_{te}(\Lambda)$ (red) than $\hat{L}_{val}(\Lambda)$ (green). While $\hat{L}_{tr}(\Lambda)$ is a seriously biased estimate of the error probability, $-\hat{U}_{tr}(\Lambda)$ seems to estimate the expected boundary uncertainty quite accurately.

6.4.2 Results on Real-Life Data

For the Letter Recognition dataset, we have a testing set, so we separately display more detailed results in Fig. 7, that follows the same layout as in the upper row of Fig. 6. We note how $-\hat{U}_{tr}(\Lambda)$ and $-\hat{U}_{te}(\Lambda)$ are strikingly close in Panel **3**, which seems to confirm that Proposal 2 can accurately estimate boundary uncertainty. This contrasts once more with the impossibility of estimating $\hat{L}_{te}(\Lambda)$ simply with $\hat{L}_{tr}(\Lambda)$ (orange curve in Panel **2**).

We observe that the minimum value of $\hat{L}_{val}(\Lambda)$ on the Letter Recognition dataset is nearly zero, which implies that this dataset may be easy to classify and that performance for the Bayes risk can be achieved at this minimum value of $\hat{L}_{val}(\Lambda)$. The minimum of $\hat{L}_{val}(\Lambda)$ and $\hat{U}_{tr}(\Lambda)$ coincide, but the minimum value of $-\hat{U}_{tr}(\Lambda)$ is not -1, as it would be if the classifier executed B^*. If we assume that Proposal 2 accurately estimates boundary uncertainty (based on observations from Panel **3**), then it would be reasonable to assume that a classifier status could be nearly optimal in terms of classification error probability, while boundary uncertainty may not appear quite optimal so long as $B(\Lambda)$ does not even get closer to optimal $B(\Lambda)$. In other words, boundary uncertainty may evaluate the generalization ability more strictly than the classification error probability.

For the other real-life datasets, we did not have access to a large testing set. In this case, the best available estimates of the classification error probability and of $-U(\Lambda)$ are $\hat{L}_{val}(\Lambda)$ and $-\hat{U}_{tr}(\Lambda)$, respectively.[10] We apply the same checks as in Section 6.4.1 but by replacing $\hat{L}_{te}(\Lambda)$ and $\hat{U}_{te}(\Lambda)$ with $\hat{L}_{val}(\Lambda)$ and $-\hat{U}_{tr}(\Lambda)$, respectively. We expect similar trends and minimum values of $\hat{L}_{val}(\Lambda)$ and $-\hat{U}_{tr}(\Lambda)$ against the classifier status.

For comparison, we also consider $-\hat{U}_{tr}(\Lambda)$ obtained by Proposal 1 with Eq. 11. The slightly different trends of $\hat{L}_{val}(\Lambda)$ between Proposal 1 and Proposal 2 may be due to slightly different splittings between our former experiments of Proposal 1 [18] and the experiments in this paper. Owing to the range of values of the binary Shannon entropy in Proposal 1, we rescaled $-\hat{U}_{tr}(\Lambda)$ in the results of Proposal 1 by a factor $\ln(2)$ so that the range of values becomes [0, 1] as in Proposal 2. In Proposal 2, the minimum values of $-\hat{U}_{tr}(\Lambda)$ are closer to -1 for each dataset, which achieves Goal (a).

Figure 8 shows that for most of the datasets, Proposal 2 matches the benchmark CV in terms of trends and minimum, which reaches Goal (b). Additionally, the trends of $-\hat{U}_{tr}(\Lambda)$ in Proposal 2 provide a sharper minimum than either their counterpart in Proposal 1 or $\hat{L}_{val}(\Lambda)$. This higher ability to sharply select an optimal classifier status achieves Goal (c).

Figure 9 shows the MLP selection results for the Letter Recognition dataset. The layout is the same as in Fig. 7, but in this case the horizontal axis corresponds to the training epoch. Once more, Proposal 2 seems to achieve the three goals we described in Section 6.4.1 for the GMM dataset.

6.5 Effect of $\widetilde{\mathcal{T}}$

6.5.1 Illustration of Covariate Shift in Near-Boundary-Ness Measurement Space

In order to illustrate the covariate shift in the near-boundary-ness measurement space $f(\cdot;\Lambda)$, as well as the usefulness of $\widetilde{\mathcal{T}}$ covered in Section 4.2.2, we selected an SVM classifier status Λ that has a rather high representation capability. We then plotted in Fig. 10 the three distributions $f(\mathcal{T};\Lambda)$, $f(\widetilde{\mathcal{T}};\Lambda)$, $f(\mathcal{X}^{te};\Lambda)$ in the

[9] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedKFold.html

[10] One motivation of boundary uncertainty is to avoid resampling methods. Therefore, we do not estimate $U(\Lambda)$ using validation folds but using *all of the available data,* (i.e., non-testing data).

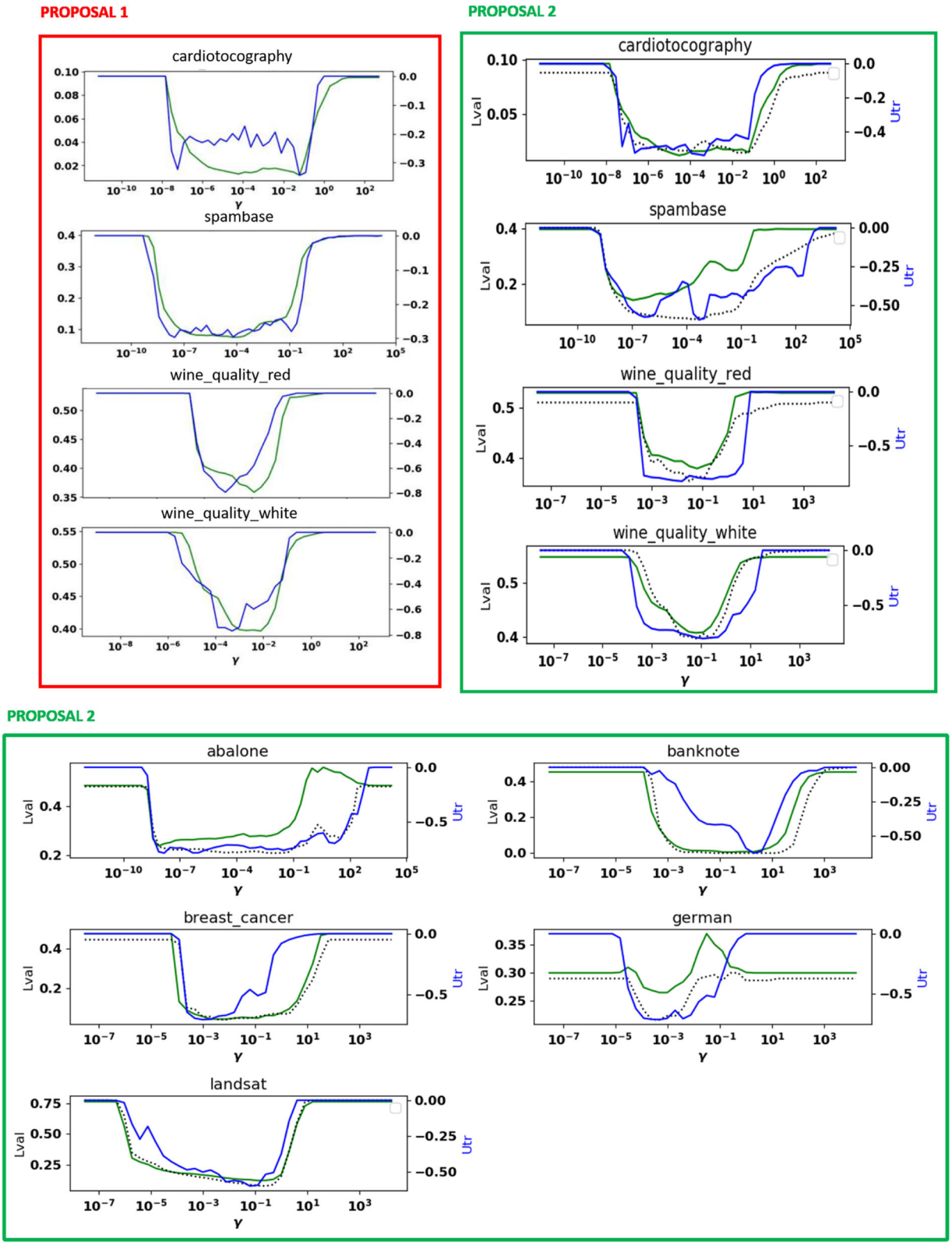

Figure 8 SVM classifier selection results using Proposal 2 on real-world datasets (green boxes). From left to right, and top to bottom: Abalone, Banknote, Breast Cancer, Cardiotocography, German, Spambase, Landsat Satellite, Letter Recognition, Wine Quality Red, and Wine Quality White. For each dataset, horizontal axis: inverse γ of the Gaussian kernel width; left vertical axis: $\hat{L}_{val}(\Lambda)$ (green), L_{HO} (dotted black); and right vertical axis: $-\hat{U}_{tr}(\Lambda)$ (blue). SVM classifier selection results using Proposal 1 on the Cardiotocography, Spambase, Wine Quality Red, and Wine Quality White datasets (red box), which we considered difficult.

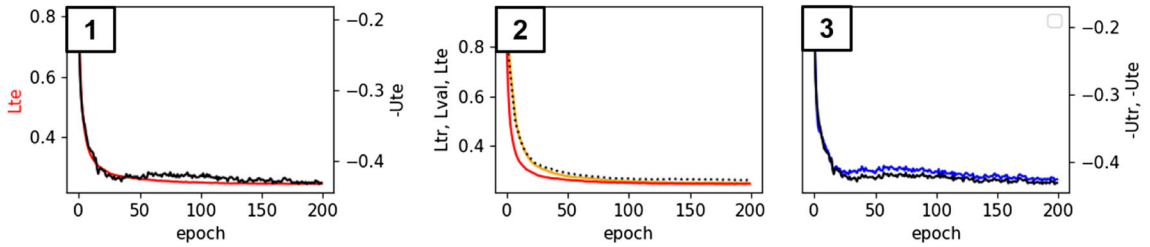

Figure 9 MLP classifier selection results using Proposal 2 on the Letter Recognition dataset. Horizontal axis for Panels **1** to **3**: index of training epochs. Panel **1**: $\hat{L}_{te}(\Lambda)$ (red) on the left vertical axis, $-\hat{U}_{te}(\Lambda)$ (black) on the right vertical axis. Panel **2**: L_{HO} (dotted black), $\hat{L}_{tr}(\Lambda)$ (orange) on the left vertical axis. Panel **3**: $-\hat{U}_{tr}(\Lambda)$ (blue), $-\hat{U}_{te}(\Lambda)$ (black) on the right-hand axis.

sample space by representing the values of the near-boundary-ness measurement using a colormap. The plots confirm that $f(\mathcal{T};\Lambda)$ is quite different from $f(\mathcal{X}^{te};\Lambda)$, whereas $f(\widetilde{\mathcal{T}};\Lambda)$ is quite closer to $f(\mathcal{X}^{te};\Lambda)$.

6.5.2 Effect of $\widetilde{\mathcal{T}}$ in Boundary Uncertainty Estimation

To show the effect of $\widetilde{\mathcal{T}}$, we display in Fig. 11 and in Panels **1bis** and **3bis** of Fig. 7 the results obtained by Proposal 2 when naively computing the values of $f(\cdot;\Lambda)$ on $\mathcal{T}$. For almost all datasets, we observe that $-\hat{U}_{tr}(\Lambda)$ assigns a favorable uncertainty value even to excessively high values of γ that are clearly not optimal (high $\hat{L}_{val}(\Lambda)$ as indicated on the left vertical axis). Such clear degradation of the accuracy of $-\hat{U}_{tr}(\Lambda)$ for higher values of γ confirms the usefulness of $\widetilde{\mathcal{T}}$. For the Spambase dataset, we see a noisier trend of $-\hat{U}_{tr}(\Lambda)$ in Fig. 8 compared to Fig. 11. This may call for refinements of the definition of $\widetilde{\mathcal{T}}$ (Algorithm 1) in order to more "naturally" perturbate $\mathcal{T}$.

6.6 Influence of Hyperparameter M on $\hat{U}^{(3)}(\Lambda)$

To ensure that Proposal 2 is not sensitive to the setting of M, we performed the above SVM selection experiments not only for $M = 40$ but also for a wide range of values: $M = 20, 40, 80, 120, 160$. We show the corresponding results in Fig. 12 for the GMM and Wine Quality White datasets. Figure 12 shows that Proposal 2 is quite insensitive to the value of M. The minimum and trend of $\hat{U}^{(3)}(\Lambda)$ seem quite insensitive to M, although the value of $\hat{U}^{(3)}(\Lambda)$ itself may slightly change. This insensitivity was observed for all datasets, so we only display the results for two datasets. Despite this apparent insensitivity, we prefer the smaller value $M = 40$ in order to have just enough samples for the estimation described by Eq. 27, while focusing *locally* along $B(\Lambda)$.

6.7 Influence of Hyperparameter K on $\hat{U}^{(3)}(\Lambda)$

In order to perform the classifier selection, as Section 4.4.3 implies, Proposal 2 requires that model selection be performed on a generative classifier model, using another model selection criterion (AIC in our case). This requirement may seem to defeat the purpose of our proposal, both conceptually and in terms of time costs. However, we show that this requirement is not a bottleneck but rather a rough initialization for Proposal 2.

First, the time costs incurred by the class-by-class search of the optimal number of prototypes are not high, as described in Section 6.10. Second, the setting of each K_j is actually quite insensitive. Figure 13 illustrates the classifier evaluation results of Proposal 2 obtained by imposing

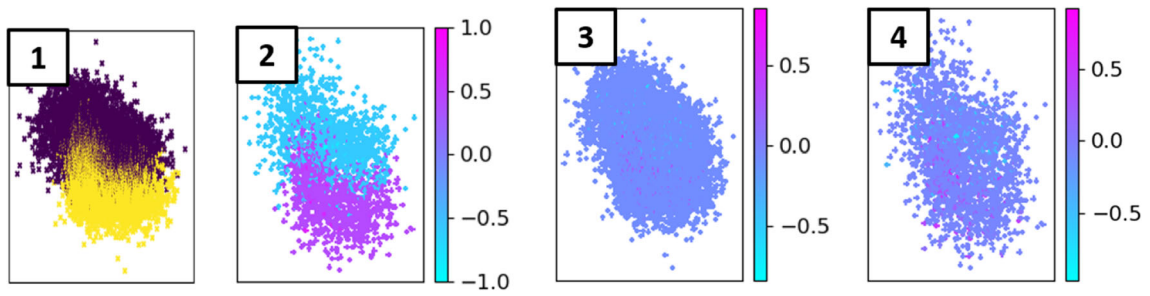

Figure 10 Usefulness of $\widetilde{\mathcal{T}}$. Panel **1**: the GMM dataset with its given class labels represented by yellow and purple crosses, respectively. The classifier used is a Gaussian kernel SVM, whose regularization coefficient was preliminarily set. For a given Λ, the three distributions $f(\mathcal{T};\Lambda)$, $f(\widetilde{\mathcal{T}};\Lambda)$, $f(\mathcal{X}^{te};\Lambda)$ in the sample space are represented using a colormap in Panels **2**, **3**, and **4**, respectively. Cyan and pink colors correspond to lower and higher algebraic values of $f(\cdot;\Lambda)$, respectively. Negative and positive values of $f(\cdot;\Lambda)$ correspond to a classifier decision that assigns data to the yellow and purple classes, respectively.

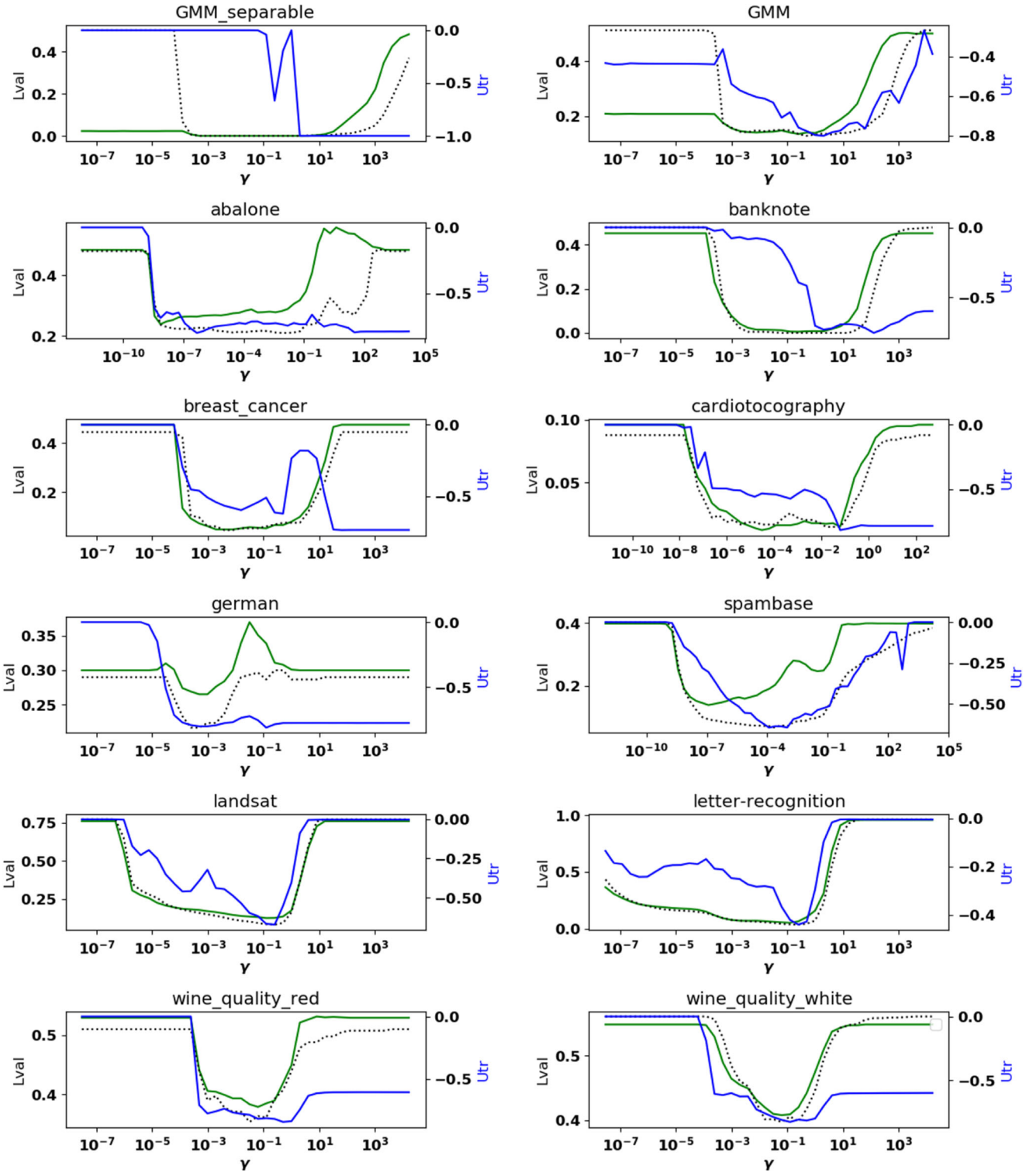

Figure 11 SVM classifier selection results using Proposal 2 by naive computation of values of the near-boundary-ness measurement on $\mathcal{T}$, without using $\widetilde{\mathcal{T}}$. From left to right and top to bottom: Abalone, Banknote, Breast Cancer, Cardiotocography, German, Spambase, Landsat Satellite, Letter Recognition, Wine Quality Red, and Wine Quality White. For each dataset, horizontal axis: inverse γ of the Gaussian kernel width; left vertical axis: $\hat{L}_{val}(\Lambda)$ (green), L_{HO} (dotted black); and right vertical axis: $-\hat{U}_{tr}(\Lambda)$ (blue).

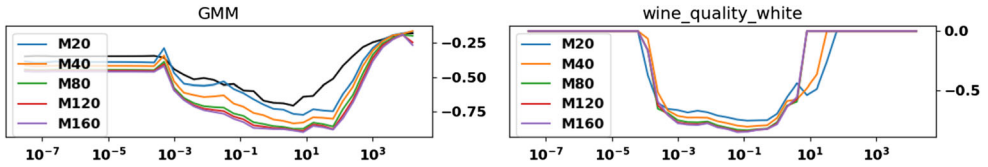

Figure 12 Influence of M over $\hat{U}^{(3)}(\Lambda)$. We displayed the SVM selection results of Proposal 2 obtained for the GMM and Wine Quality White datasets in the left and right panels, respectively. Horizontal axis: inverse γ of the Gaussian kernel width. Right vertical axis: $-\hat{U}_{tr}(\Lambda)$. Estimates of $-\hat{U}_{tr}(\Lambda)$ based on different values of $M = 20, 40, 80, 120, 160$ are distinguished by different colors. For the GMM dataset, we also plotted $\hat{U}_{te}(\Lambda)$ (black curve) to serve as a reference.

different values of the number of prototypes per class, which we set the same for all classes for simplicity and which we denote by K. We attempted the values $K = 3, 10, 20, 40$. We also display the results obtained by setting each K_j using AIC as described in Algorithm 2. $\mathcal{I}^*$ has no influence on two-class datasets, so we only display results for multi-class datasets. We only show two datasets, since results on other datasets show the same phenomenon. We set $M = 40$ in this paragraph. Figure 13 shows that results are almost the same despite the quite broad range of values for K. This insensitivity may be the result of $\mathcal{I}^*$ only requiring the determination of the *indexes* of the two highest class posterior probabilities, instead of their *actual* values.

6.8 Handling of Datasets with Well-Separated Classes

This section illustrates the usefulness of the branching treatment proposed by Eq. 39 on a synthetic dataset that features well-separable classes, which we call GMM_separable. Figure 14 shows details of the SVM evaluation results obtained on the GMM_separable dataset.

For this dataset, B^* obviously lies in the middle of the empty region that separates the two classes, as also illustrated in Fig. 5. For four classifier statuses γ_1 to γ_4, we represent the location of $B(\Lambda)$ by plotting the values of the probability weights $k(\cdot; \Lambda)/w_{12}(\Lambda)$ in a colormap. Training samples that are close to $B(\Lambda)$ are represented in cyan, and those even closer to $B(\Lambda)$ are represented in pink.

The colormap shows that γ_1 results in a biased $B(\Lambda)$ that diagonally crosses the two classes, instead of passing between them. γ_2 results in a better $B(\Lambda)$, although the classifier appears quite uncertain as $B(\Lambda)$ seems to spread widely around the empty region (samples in cyan). γ_3 executes the Bayes boundary, namely $B(\Lambda) \approx B^*$: In this case, $B(\Lambda)$ lies in the center of the empty region between the class distributions, and it is far from either class distribution, so no training sample appears in cyan or pink. γ_4 is too high: The resulting $B(\Lambda)$ unnecessarily encircles both class distributions, which shows overfitting to the training set.

While all of $\gamma_1, \gamma_2, \gamma_3, \gamma_4$ are assigned a quite favorable score $\hat{L}_{val}(\Lambda)$, Proposal 2 can discriminate quite sharply between a classifier that truly executes B^*, and a classifier that is actually quite far from executing B^*, despite apparently low classification error on the finite data at hand. This sharp evaluation ability of Proposal 2 seems to be the result of the sharp estimation focus on $B(\Lambda)$ and B^*, combined with the perturbation used when evaluating the near-boundary-ness measurement that efficiently reacts to overfitting.

6.9 Experimental Considerations of Equivalence between U_{max} and B^*

The equivalence between achieving U_{max} and achieving the Bayes decision rule is a key component of boundary uncertainty. Therefore, we further discuss the equivalences introduced in Section 4.5.

Equivalences (21) and (22) are obvious, based on the definitions of $B(\Lambda)$, B^* and $U(\Lambda)$. Equivalence (24) (i.e., equivalence between B^* and Bayes decision rule) is also quite straightforward. There may be one counter-example to Equivalence (24): Given a two-class task, we may have $B(\Lambda) = B^*$, while *all* of the class labels may be mistaken. This counter-example is highly unlikely in practice, and even more unlikely in the case of multiclass classification.

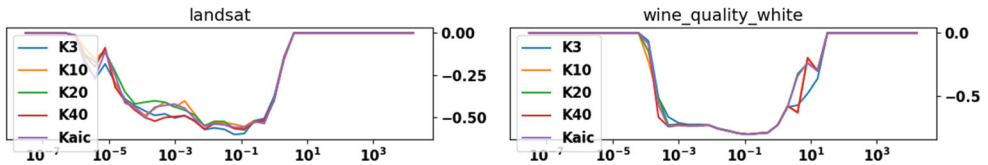

Figure 13 Influence of K over $\hat{U}^{(3)}(\Lambda)$. The SVM selection results of Proposal 2 obtained for the Landsat and Wine Quality White datasets are displayed in the left and right panels, respectively. Horizontal axis: inverse γ of the Gaussian kernel width. Right vertical axis: $-\hat{U}_{tr}(\Lambda)$. Estimates of $-\hat{U}_{tr}(\Lambda)$ based on the different values $K = 3, 10, 20, 40$ are distinguished by different colors. We also display the estimate of $-\hat{U}_{tr}(\Lambda)$ based on each K_j set by AIC, labeled as "Kaic".

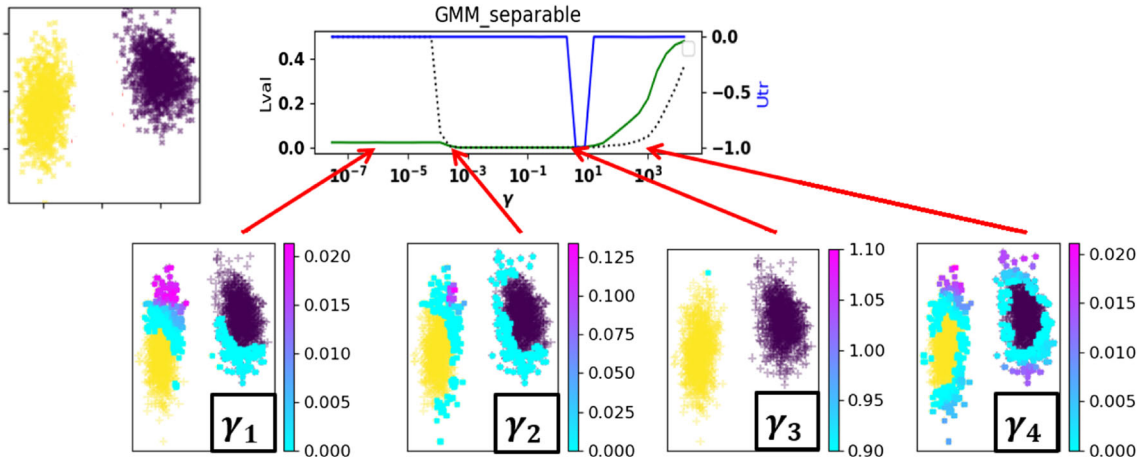

Figure 14 Handling of well-separated classes by Proposal 2. Upper row, left: scatter plot of GMM_inclusion. Training samples with given class labels C_1 and C_2 represented with yellow and purple crosses, respectively. Upper row, middle panel: SVM evaluation results obtained from Proposal 2 on the GMM_inclusion dataset; horizontal axis: inverse γ of the kernel width; left vertical axis: $\hat{L}_{val}(\Lambda)$ (green); right vertical axis: $-\hat{U}_{tr}(\Lambda)$ (blue). Lower row: for four classifier statuses obtained from $\gamma_1, \gamma_2, \gamma_3, \gamma_4$ indicated with the red arrows, we provide the scatter plot of GMM_inclusion along with the probability weights $k(\cdot; \Lambda)/w_{12}(\Lambda)$ represented in a colormap that varies from cyan to pink. Training samples that appear in more fully cyan and more fully pink values are further and closer to $B(\Lambda)$, respectively. Only *strictly* positive probability weights appear in the colormap, and thus training samples that are far from $B(\Lambda)$ just appear in yellow or purple.

We now elaborate on Equivalence (23). Strictly speaking, maximum boundary uncertainty measures the inclusion $B(\Lambda) \subset B^*$. Here, the main point is that the perfect inclusion $B(\Lambda) \subset B^*$ is unlikely, and that even a nearly included $B(\Lambda)$ is likely to have bias (un-Bayes-boundary-ness) that can be detected with an accurate boundary uncertainty estimation. We first illustrate the inclusion issue on *two-class* data using the GMM_inclusion dataset (top-left corner of Fig. 15).

We generated the GMM_inclusion dataset so that B^* consists of two fragments, denoted by $B^{*(1)}$ and $B^{*(2)}$, respectively. $B^{*(1)}$ is a gentle curve between the blue (left) and red (right) sample crowds. $B^{*(2)}$ is an ellipse surrounding the blue dense crowd at the right side. We plot the classifier selection results by Proposal 2 on the GMM_inclusion dataset with a Gaussian kernel SVM following the conventions used in Fig. 8. Λ_A and Λ_B are two trained classifier statuses that were obtained from γ_A and γ_B, respectively. Panels **A** and **B** visualize the classifier decision that corresponds to Λ_A and Λ_B, respectively, by showing the training samples with their predicted class labels (red and blue dots) as well as some estimated anchors (black dots).

$B(\Lambda_A)$ is nearly included in B^*: $B(\Lambda_A) \approx B^{*(1)}$, but $B(\Lambda_A)$ "omitted" $B^{*(2)}$. We can explain this by the insufficient representation capability of the SVM for Λ_A. $B(\Lambda_B)$ is nearly equal to B^*. Thus $\hat{U}(\Lambda_A) \approx U_{max}$, even though $B(\Lambda_A)$ is clearly not optimal. In practice, the graph in Fig. 15 shows that Proposal 2 quite clearly outputs $\hat{U}(\Lambda_B) \geq \hat{U}(\Lambda_A)$, despite the near inclusion owing to the unavoidable (although admittedly small) bias of $B(\Lambda_A)$ when trying to reproduce $B^{*(1)}$ with insufficient representation capability.

Assuming that $B(\Lambda_A)$ was even closer to inclusion in B^*, there may be a simple way of detecting the obvious non-optimality of $B(\Lambda_A)$. To fix the ideas, we represent in the rectangle box below Panel **A** the schematic histograms of the distributions $\{f(\boldsymbol{x}; \Lambda_A)\}_{\boldsymbol{x} \in C_1}$ and $\{f(\boldsymbol{x}; \Lambda_A)\}_{\boldsymbol{x} \in C_2}$ in blue and red, respectively. We note that $f(\boldsymbol{x}; \Lambda_A) = g_1(\boldsymbol{x}; \Lambda_A) - g_2(\boldsymbol{x}; \Lambda_A)$. Misclassifications are represented by the yellow areas under the histograms. We observe that the omission by $B(\Lambda_A)$ of the entire fragment of Bayes boundary $B^{*(2)}$ results in a high amount of misclassification quite far away from $f(\cdot; \Lambda_A) = 0$. However, the number of misclassifications usually become smaller and smaller as we go further away from $B(\Lambda)$. In order to safely avoid the erroneous selection of classifier statuses with excessively low representation capability, investigating the detection and incorporation of such a suspicious burst of misclassifications far from $B(\Lambda)$ in the computation of $\hat{U}(\Lambda)$ remains a possible future work.

6.10 Time Costs of Proposal 2

6.10.1 Time Cost Comparison between Proposal 2 and Proposal 1

This section quantifies the speed improvement of Proposal 2 compared to Proposal 1. For several real-life datasets used in our experiments and for any given classifier status Λ, Table 3 displays the multiplicative gain of Proposal 2 over Proposal 1 in regard to each elementary cost introduced

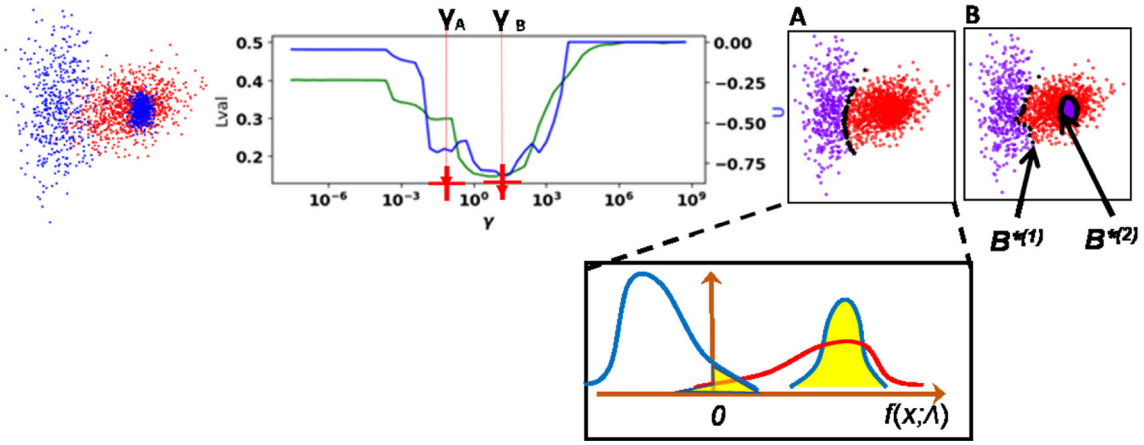

Figure 15 Discussion of Equivalence (23). Left side: scatter plot of GMM_inclusion. Training samples with given class labels C_1 and C_2 represented with blue and red dots, respectively. Middle panel: SVM evaluation results obtained from Proposal 2 on the GMM_inclusion dataset; horizontal axis: inverse γ of the kernel width; left vertical axis: $\hat{L}_{val}(\Lambda)$ (green); and right vertical axis: $-\hat{U}_{tr}(\Lambda)$ (blue). γ_A: We nearly have $B(\Lambda_A) \subset B^*$. γ_B: We nearly have $B(\Lambda_B) = B^*$. Right side (Panels **A** and **B**): training samples with the predicted class labels (purple and red dots), and on-boundary anchors (black dots) for Λ_A and Λ_B, respectively. We denote by $B^{*(1)}$, $B^{*(2)}$ the two fragments that constitute B^*. Rectangle box under Panel **A**: schematic histograms of the distributions $\{f(\boldsymbol{x};\Lambda)\}_{\boldsymbol{x}\in C_1}$ and $\{f(\boldsymbol{x};\Lambda)\}_{\boldsymbol{x}\in C_2}$ in blue and red, respectively. We note that $f(\boldsymbol{x};\Lambda) = g_1(\boldsymbol{x};\Lambda) - g_2(\boldsymbol{x};\Lambda)$. Misclassifications are represented with the yellow areas under the histograms.

in Section 5 when estimating the boundary uncertainty. We only consider the time costs that are exclusive to the evaluation of a classifier status. For example, $G_{add}(\Lambda) = c^{(1)}_{add}(\Lambda)/c^{(2)}_{add}(\Lambda)$. We could not compute a gain in regard to c_{exp} because this elementary cost was not present in Proposal 1.

To compute the values in Table 3, we optimistically assumed that $R_A \sim N$, although for higher-dimensional datasets, quite higher values of R_A seemed necessary (i.e., Step 1 of Proposal 1 kept selecting different training samples even as the number of generated anchors increased well above N). We set the following values in Proposal 1 and Proposal 2: $i_{max} = 20$, $R_T = 10$, $i_K = 20$, $M = 40$. The values of d, J, N depend on the dataset, and they are given in Table 2.

Table 3 shows that even for the relatively small datasets used in our experiments, Proposal 2 is 100 to 1000 times faster than Proposal 1, and this gain increases with d, J, N (Section 5).

6.10.2 Time Costs comparison between Proposal 2 and CV

The time costs of SVM training are between $O(N^2)$ and $O(N^3)$ [31]. Along with the cost of classifying the validation folds, such training costs are repeated as many times as there are validation folds in CV. To obtain an

Table 2 Datasets.

Dataset	N	N_{te}	d	J	Remarks
GMM	2,000	18,000	2	2	class overlap
GMM_inclusion	2,000	0	2	2	discontinuous B^*
GMM_separable	2,000	0	2	2	well-separated classes
Abalone	4,177	0	7	3	class overlap
Banknote	1,372	0	4	2	well-clustered classes
Breast Cancer	683	0	9	2	well clustered classes
Cardiotocography	1,831	0	30	2	10:1 imbalance
German	1,000	0	26	2	sparse
Letter Recognition (LR)	10,000	10,000	16	26	well-clustered classes
Satimage (Landsat Satellite)	6,435	0	36	7	
Spambase	4,601	0	57	2	sparse
Wine Quality Red	1,000	0	11	2	class overlap
Wine Quality White	3,961	0	11	3	class overlap

Table 3 Gain of elementary time costs of Proposal 2 compared to Proposal 1.

Dataset	$G_{add}(\Lambda)$	$G_{mul}(\Lambda)$	$G_{comp}(\Lambda)$	$G_{CL}(d, j, \Lambda)$
Abalone	$2E+02$	$2E+02$	$2E+01$	$2E+01$
Cardiotocography	$7E+02$	$7E+02$	$2E+01$	$2E+01$
Letter Recognition	$6E+02$	$6E+02$	$2E+01$	$2E+01$
Landsat	$1E+03$	$1E+03$	$2E+01$	$2E+01$
Spambase	$2E+03$	$2E+03$	$2E+01$	$2E+01$

accurate CV estimate, 5 to 10 folds are usually required. In contrast, Proposal 2 can be accurately evaluated through only a single instance of classifier training, directly on the training data. The time costs of classifier training are usually the predominate costs of classifying the data. Given a candidate classifier status, comparing the time costs of Proposal 2 with CV is thus equivalent to comparing the time costs of repeated classifier training with those of estimating boundary uncertainty. Equation 43 shows that the time costs of Proposal 2 are linear with N, J and nearly independent of d (d only appears in the first block in Algorithm 3). This shows that, at least for SVMs, Proposal 2 can be applied to obtain accurate classifiers even for large-scale tasks, in contrast to CV.

7 Discussion and Conclusion

In this paper, we formalized a new boundary uncertainty estimation method that provides more accurate, applicable, and scalable classifier evaluation (no anchor generation, no random repetitions, no unclear settings). The new proposal also clarified the two main reasons why boundary uncertainty can be accurately estimated based on a finite amount of training data without costly methods such as Cross Validation (CV). First, a tight focus relative to the classifier boundary implies that there is *locally* a *single dimension* of interest in the boundary uncertainty estimation task: such an estimation task is fundamentally easier and can be accurately performed even analytically. Second, both the classifier boundary and the Bayes boundary are identified by known conditions that can be accurately approximated from the training data. Performing classifier evaluation in a single shot without averaging, as done in CV, may in a sense improve interpretability: "what we got is what we evaluated." This contrasts with CV, where the final model is the result of separately re-training the entire available dataset, which is also different from each of the models averaged over the training folds.

Our approach to classifier evaluation starts from the statistical assumption that there is class overlap around the Bayes boundary. In practice, the feature-extraction step before classification aims to generate well-separated classes where no sample is close to the Bayes boundary (Bayes risk is equal to zero). In this regard, our approach may seem paradoxical. Nevertheless, in practice, the data may be high-dimensional and difficult to separate, implicitly containing irreducible misclassification. Traditional approaches based on classification error tend to easily overfit such difficult data, and the design of more and more powerful classifier models may also make them prone to overfitting unless adequate classifier evaluation is carried out. Our proposed approach may be especially useful in such settings. Meanwhile, our proposal can also handle simple cases with well-separated classes.

So far, we have focused our efforts on designing the concept of boundary uncertainty and obtaining reasonably accurate, scalable and applicable results. However, we have not yet obtained results that quantitatively show a definite gain in generalization ability compared to traditional approaches in state-of-the-art classification tasks. For now, boundary uncertainty can be used to determine whether the Bayes risk is achieved, and to design classifiers directly based on the training data. Boundary uncertainty actually defines *another* metric for generalization ability that could even be used as a reference to describe the performance on a testing set. Future steps will aim at careful consideration of how to evaluate boundary uncertainty itself, and how to make the most of it for accurate classification.

Other future steps for boundary uncertainty-based classifier evaluation include strengthening the equivalence between maximum boundary uncertainty and Bayes decision rule; deepening the formalization of boundary uncertainty and performing quantitative analysis of our classifier evaluation method, for example by establishing the rate of convergence of our boundary uncertainty estimator; and investigating applications to challenging tasks, including large-scale tasks where many parameters and hyperparameters are simultaneously optimized.

Appendix A: Class-Wise Application of AIC to Determine the Number of Prototypes per Class

We consider a class index $j \in [\![1, J]\!]$. We denote by $\mathcal{T}_j$ the set of training samples with given class label C_j and by M_j the set of prototypes that represents class C_j, namely $M_j = \{p_j^k\}_{k \in [\![1, K_j]\!]}$. Given a training sample $\boldsymbol{x} \in \mathcal{T}_j$, the likelihood of $\boldsymbol{x}$ can be calculated by assigning $\boldsymbol{x}$ to the mixture component of class C_j that has the highest probability,

namely to the prototype of class C_j that is closest to $\boldsymbol{x}$. We denote this prototype by $p_j(\boldsymbol{x})$. The likelihood of $\boldsymbol{x}$ is thus

$$P(\boldsymbol{x}|M_j, \sigma_j^2) = \frac{1}{\sqrt{2\pi\sigma_j^2}} \exp\left(-\frac{\left(\boldsymbol{x} - p_j(\boldsymbol{x})\right)^2}{2\sigma_j^2}\right), \quad (48)$$

where we take the variance σ_j^2 to be the within-cluster variance:

$$\sigma_j^2 = \frac{1}{|\mathcal{T}_j|} \sum_{\boldsymbol{x} \in \mathcal{T}_j} ||x - p_j(\boldsymbol{x})||^2. \quad (49)$$

The likelihood of $\mathcal{T}_j$ is

$$P(\mathcal{T}_j|M_j, \sigma_j^2) = \prod_{\boldsymbol{x} \in \mathcal{T}_j} P(\boldsymbol{x}|M_j, \sigma_j^2). \quad (50)$$

Finally, the AIC score for class C_j represented with the number of prototypes K_j is

$$AIC(K_j) = \log P(\mathcal{T}_j|M_j, \sigma_j^2) - (K_j d + 1). \quad (51)$$

We select the value of K_j that yields the highest value of $AIC(K_j)$.

Acknowledgements We express our deep gratitude to Nicolas Ménard and Emilie Delattre for their work involving Proposal 2 at Doshisha University, and to Prof. Wu for her continuing support.

References

1. Hastie, T., Tibshirani, R., Friedman, J. (2009). *Optimism of the error rate (in the elements of statistical learning*, 2nd edn., (pp. 228–230). New York: Springer Science+Business Media.
2. Vapnik, V. (1998). The structural risk minimization principle. In Vapnik, E. (Ed.) *Statistical learning theory*. 1st edn. (pp. 219–268). New York: Wiley.
3. Guyon, I., Vapnik, V., Boser, B., Bottou, L., Solla, S.A. (1991). Structural risk minimization for character minimization. *Proceedings of Advances in Neural Information Processing Systems*, *4*(NIPS 1991), 471–479.
4. Stone, M. (1974). Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society*, *36*(2), 111–147.
5. Chapelle, O., & Vapnik, V. (2000). Model selection for support vector machines. *Advances in Neural Information Processing Systems*, *12*.
6. Efron, B., & Tibshirani, R. (1993). *An introduction to the bootstrap*. New York: Springer Science+Business Media.
7. Demyanov, S., Bailey, J., Ramamohanarao, K., Leckie, C. (2012). AIC And BIC based approaches for SVM parameter value estimation with RBF kernels. *Journal of Machine Learning Research*, *25*, 97–112.
8. Devroye, L., Gyorfi, L., Lugosi, G. (1997). 24 deleted estimates of the error probability. In *A probabilistic theory of pattern recognition* (pp. 407–419): Springer.
9. Juang, B.-H., & Katagiri, S. (1992). Discriminative learning for minimum error classification. *IEEE Transactions on Signal Processing*, *30*(12), 3043–3054.
10. Ohashi, T., Watanabe, H., Tokuno, J., Katagiri, S., Ohsaki, M., Matsuda, S., Kashioka, H. (2012). Increasing virtual samples through loss smoothness determination in large geometric margin minimum classification error training. In *Proceedings of 2012 international conference on acoustics, speech and signal processing (ICASSP 2012)* (pp. 2081–2084).
11. Akaike, H. (1974). A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, *19*(6), 716–723.
12. Shwarz, G. (1978). Estimating the dimension of a model. *The Annals of Statistics*, *6*(2), 461–464.
13. Bishop, C. (2006). 3.4 Bayesian model selection. In *Pattern recognition and machine learning* (pp. 161–165): Bishop.
14. Bouchard, G., & Celleux, G. (2006). Selection of generative models in classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *28*(4), 544–554.
15. Guo, Y., & Greiner, R. (2005). Discriminative model selection for belief net structures. In *Proceedings of AAAI twentieth national conference on artificial intelligence (AAAI-05)* (pp. 770–776).
16. Ha, D., Watanabe, H., Tomotoshi, Y., Delattre, E., Katagiri, S., Ohsaki, M. (2018). Optimality analysis of boundary-uncertainty-based classifier model parameter status selection method. In *Proceedings of 2018 ACM international conference on signal processing and machine learning*.
17. Tomotoshi, Y., Ha, D., Delattre, E., Watanabe, H., Katagiri, S., Ohsaki, M. (2019). Optimal Classifier status selection using class boundary uncertainty measure for prototype-based and neural network classifier. In *International symposium on integrated uncertainty in knowledge modeling and decision making*.
18. Ha, D., Tomotoshi, Y., Senda, M., Watanabe, H., Katagiri, S., Ohsaki, M. (2019). A practical method based on bayes boundaryness for optimal classifier parameter status selection. *Journal of Signal Processing Systems*.
19. Bishop, C. (2006). 1.5.1. minimizing the misclassification rate. In *Pattern recognition and machine learning* (pp. 39–40): Bishop.
20. Bishop, C. (2006). 1.4.4. inference and decision. In *Pattern recognition and machine learning* (pp. 42–44): Bishop.
21. Sugiyama, M., Kraudelat, M., Muller, K.R. (2007). Covariate shift adaptation by importance weighted cross validation. *Journal of Machine Learning Research*, *8*, 985–1005.
22. Variani, E., Bagby, T., McDermott, E., Bacchiani, M. (2017). End-to-end training of acoustic models for large vocabulary continuous speech recognition with tensorflow. Interspeech 2017.
23. McDermott, E., & Katagiri, S. (2004). A derivation of minimum classification error from the theoretical classification risk using parzen estimation. *Elsevier Computer Speech and Language*, *18*, 107–122.

24. Bishop, C. (2006). 2.5.1. Kernel density estimator. In *Pattern recognition and machine learning* (pp. 122–124): Bishop.
25. Silverman, B. (1986). *Density estimation for statistics and data analysis*. London: Chapman and Hall.
26. Goutte, C., Hansen, L.K., Liptrot, M.G., Rostrup, E. (2001). Feature-space clustering for fMRI meta-analysis. *Human Brain Mapping*, *13*, 165–183.
27. Burges, C. (2009). A tutorial for support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, *2*, 1485–1510.
28. Sugiyama, M., Suzuki, T., Kanamori, T. (2010). Density ratio estimation: a comprehensive review. *RIMS Kokyuroku*, *1703*, 10–31.
29. Chawla, N.V., Bowyer, K.W., Hall, L.O., Kegelmeyer, W.P. (2002). SMOTE: synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, *16*, 321–357.
30. Cormen, T., Leiserson, C., Rivest, R. (1990). *Introduction to algorithms*. Cambridge: MIT Press and McGraw-Hill.
31. Bottou, L., & Lin, C.J. (2007). Support vector machine solvers. In *Large-scale Kernel machines* (pp. 301–320): MIT Press and McGraw-Hill.