Machine Learning (2025) 114:293
https://doi.org/10.1007/s10994-025-06878-y

LEAP: Linear equations for classifier accuracy prediction under prior probability shift

Lorenzo Volpi[1] · **Alejandro Moreo**[1] · **Fabrizio Sebastiani**[1]

Received: 7 April 2025 / Revised: 19 July 2025 / Accepted: 25 August 2025

Abstract
The standard technique for predicting the accuracy that a classifier will have on unseen data (*classifier accuracy prediction*—CAP) is cross-validation (CV). However, CV relies on the assumption that the training data and the test data are sampled from the same distribution, an assumption that is often violated in many real-world scenarios. When such violations occur (i.e., in the presence of *dataset shift*), the estimates returned by CV are unreliable. The contribution of this paper is three-fold. First, we propose a CAP method specifically designed to work under *prior probability shift* (PPS), an instance of dataset shift in which the training and test distributions are characterized by different class priors. This method estimates the n^2 entries of the contingency table of the test data (thus allowing to estimate the value of any specific evaluation measure) by solving a system of n^2 independent linear equations, with n the number of classes. Second, we show that the equations that the cells of the contingency table must satisfy are actually more than n^2, which gives rise to an overconstrained problem, and present a family of methods each based on a different selection of n^2 such equations. Third, we observe that, since a key step of the above methods involves predicting the class priors of the test data, one can exploit intuitions from the field of class prior estimation (a.k.a. "quantification"). Our experiments show that, when combined with state-of-the-art quantification techniques, under PPS our methods tend to outperform existing CAP methods.

Keywords Classifier accuracy prediction · Prior probability shift · Label shift · Quantification

Editors: Riccardo Guidotti, Anna Monreale, Dino Pedreschi.

✉ Lorenzo Volpi
lorenzo.volpi@isti.cnr.it

Alejandro Moreo
alejandro.moreo@isti.cnr.it

Fabrizio Sebastiani
fabrizio.sebastiani@isti.cnr.it

[1] Istituto di Scienza e Tecnologie dell'Informazione, Consiglio Nazionale delle Ricerche, Via Giuseppe Moruzzi 1, 56124 Pisa, Italy

Published online: 02 December 2025

1 Introduction

In machine learning, estimating the accuracy that a classifier will have on unseen data (*classifier accuracy prediction*—CAP) is a fundamental step towards ensuring that the classifier we deploy is trustworthy. The accuracy of a classifier (where "accuracy" is broadly understood as any effectiveness metric—e.g., vanilla accuracy, or F_1, or other) is typically estimated by means of cross-validation (CV) on the training data. However, CV relies on the so-called IID assumption, according to which the training data and the test data are sampled from the same distribution. Such an assumption is often violated in many real-world scenarios. When this happens, the use of CV often leads to unreliable estimates of classifier accuracy, ultimately compromising the choice of the classifier to be deployed and/or misleading its optimization process. When the IID assumption is not verified, i.e., when *dataset shift* (Storkey, 2009) is present, CAP is thus an open problem.

There exist several different types of dataset shift (see e.g., Moreno-Torres et al., 2012; Storkey, 2009), and the particular type that affects a given learning problem can often be inferred from the nature of the problem itself. More specifically, the key question is whether, in the problem we face, we are learning from causes ("causal learning") or from effects ("anti-causal learning") (Schölkopf et al., 2012).[1] In this paper we focus on the latter case, which typically gives rise to a type of dataset shift called *prior probability shift* (PPS). PPS is the type of shift in which (a) the class priors differ between the training distribution and the test distribution, while (b) the class-conditional distribution of the covariates remains invariant across the training distribution and the test distribution. These distributional assumptions are reasonable in all problems affected by dataset shift and in which the labels represent the underlying causes and the features (i.e., covariates) are their observable effects.

A prototypical example of this arises in healthcare-related applications. In such contexts, it is common to deploy a system that, based on symptoms (i.e., the covariates describing each individual), aims to predict the pathology that caused them (i.e., the class label). When a pandemic occurs, we may expect changes in class prevalence within the population (e.g., we may expect a higher proportion of people affected by the pathology), while it is reasonable to assume that the class-conditional distributions remain stable (e.g., that the individuals affected by the pathology exhibit similarly distributed symptoms).

PPS is particularly relevant in fields such as epidemiology (Patrone & Kearsley, 2024; Godau et al., 2025), where CAP methods can be employed to select, from a pool of already-trained classifiers, the one most likely to yield accurate predictions, so that these latter can support high-stakes decision-making. Additionally, the fact that our method relies on distributional assumptions is essential in order to avoid falling under the provisions of the no-free-lunch (NFL) theorem (Wolpert & Macready, 1997), which states that, in the absence of distributional assumptions, no classifier can outperform others on average. Under such conditions, attempting to predict classifier performance would indeed be illusory (Garg et al., 2022, Proposition 1). Our method is deliberately restricted to problems where the PPS

[1] An example of causal learning is inferring the presence or absence of a given pathology from covariates representing possible causes of the pathology (e.g., age, smoker or non-smoker, presence of hypercholesterolemia, etc.); an example of anti-causal learning is, symmetrically, inferring the presence or absence of a given pathology from covariates representing possible effects (or symptoms) of the pathology (e.g., abdominal pain, nausea, itching, etc.).

assumptions hold, a necessary limitation that introduces a well-defined inductive bias and enables meaningful reasoning about classifier performance on future data from anti-causal learning problems.

1.1 Our contributions

In this paper we propose LEAP (for "Linear Equations for Classifier Accuracy Prediction"), a simple yet effective family of methods for CAP which are specifically tailored to work under PPS.

The contribution of this paper is three-fold. First, we argue that, given that the vast majority of classifier accuracy measures are computed on a contingency table that relates the true labels to the predicted labels, CAP can be approached by treating the entries of this table as unknowns that require estimation. Using these estimates, the accuracy of the classifier on the test data can be estimated too, according to one or more accuracy measures at the same time; this is in contrast to previously proposed methods, that estimate accuracy according to a single measure and thus need retraining if the estimate of a different measure is needed. In a classification problem with n classes, the contingency table has n^2 entries; we show that these entries must satisfy a number of constraints, we show that these constraints are linear equations, and we present a system of n^2 independent such equations, whose solution results in the prediction of the n^2 entries.

Our second contribution is based on the observation that the equations that the cells of the contingency table must satisfy are actually more than n^2, which gives rise to an overconstrained problem. Following this observation, we present a family of methods each based on a different choice of n^2 equations that allows estimating the entries of the contingency table.

Our third contribution starts from the observation that a key step in our method involves predicting the prevalence values (i.e., the priors, or relative frequencies) of the classes in the test data. This suggests a connection with the field of *learning to quantify* (Forman, 2005; González et al., 2017; Esuli et al., 2023), the supervised learning task of predicting the distribution of the class labels in the test data. We thus propose CAP methods whereby, for the goal of solving the above-mentioned n^2 equations, the class priors of the test data are estimated by "quantification" algorithms tailored to scenarios affected by PPS.

Why was a new CAP method necessary? The simple answer is that the methods in the LEAP family represent a substantial improvement over previous CAP methods, under several aspects. The first such aspect is that, as shown in Sect. 5, once they are equipped with state-of-the-art quantification technology, LEAP methods tend to outperform existing CAP methods. The second aspect is that, as remarked above, LEAP methods allow estimating one or more accuracy measures at the same time, which is unlike previous methods, that require multiple retrainings if estimates of multiple measures are needed. The third aspect is that methods in the LEAP family have a solid theoretical foundation; while other state-of-the-art methods (e.g., Garg et al., 2022; Guillory et al., 2021) are more empirical in nature, and do not draw a clear line between the method and the type of dataset shift they try to address, our method is based on a formalization, in terms of linear equations, of the assumptions that underlie our learning problem (i.e., learning under PPS), and on the solution of the resulting system of linear equations.

This paper is an extended version of Volpi et al. (2024), where we originally discuss the LEAP_{ACC} and $\text{LEAP}_{\text{KDEy}}$ methods of Sects. 4.3 and 4.4, respectively. In the present

paper we generalize LEAP to a family of methods, of which LEAP_{ACC} and $\text{LEAP}_{\text{KDEy}}$ are specific instances, and propose two additional methods, the S-LEAP and O-LEAP methods of Sects. 4.2 and 4.5, respectively, which we show to systematically outperform LEAP_{ACC} and $\text{LEAP}_{\text{KDEy}}$. We also extend our experimentation substantially, since (a) while in Volpi et al. (2024) the experiments only dealt with the binary setting, we now carry out extensive experimentation of all the studied CAP methods for the multiclass case too; (b) we here perform (see Sect. 5.10) a study of the uncertainty of CAP predictions, treating CAP models as black-box point estimators and studying their uncertainty by deriving empirical confidence intervals; (c) we here carry out a study of empirical sampling error (see Sect. 5.11), so as to establish how it affects CAP accuracy; (d) we here add random forests (RFs) to the set of classifiers on which we run our experiments; (e) we add four CAP methods (those discussed in Xie et al., 2023; Deng et al., 2023; Lu et al., 2023; Kivimäki et al., 2025), all appeared in the recent literature, to the set of our baselines; (f) we analyse the efficiency of the methods in the LEAP family and propose an efficient implementation of them (see Sect. 5.12); (g) we investigate the impact of sampling error on CAP performance (see Sect. 5.13), and (h) we also analyse the case in which F_1 (instead of vanilla accuracy) is our target classifier accuracy measure (see Sect. 5.14).

The rest of the paper is structured as follows. Section 2 reviews the related literature on classifier accuracy prediction. In Sect. 3 we define the notation and the concepts we use in the rest of the paper. Section 4 is devoted to explaining our novel methods, the set of equations on which they are based, and their connections with quantification learning. We present our experimental results in Sect. 5, while Sect. 6 wraps up and discusses potential ideas for future work.

2 Related work

A few methods for CAP under dataset shift have emerged in recent years (Garg et al., 2022; Guillory et al., 2021; Elsahar & Gall, 2019; Chen et al., 2021a, 2021b; Jiang et al., 2022).

The *Average Thresholded Confidence* (ATC) method of Garg et al. (2022) learns a threshold t on the confidence (viewed in terms of posterior probabilities) that the classifier has in a classification decision, such that datapoints classified with confidence higher than t are assumed to be correctly classified. The authors suggest two functions for computing this score, based, respectively, on maximum confidence and negative entropy. Threshold t can be learned on the validation set as the value such that the portion of datapoints classified with confidence higher than it corresponds to the portion of correctly classified datapoints.

The *Difference of Confidence* (DoC) method of Guillory et al. (2021) also leverages the confidence a model shows in its predictions, since it uses the confidence scores returned by the model to train a regressor that estimates the accuracy of the model on the test set. In addition to the validation set V, the method assumes the existence of several validation samples V_i exhibiting some type of dataset shift with respect to V (and ideally resembling the type of dataset shift to be faced in deployment conditions). For every validation sample V_i, the average max confidence of the classifier (C^{V_i}) is computed. The scores used to train the regressor are computed as the "difference of confidences" (DoC—hence the name of the method) between V and each V_i, i.e., as $\text{DoC}(V, V_i) = C^V - C^{V_i}$. A regressor is thus trained on these scores to predict the difference in accuracy that h exhibits on V and U. The

accuracy of h on U can be computed straightforwardly from the predicted score $\mathrm{DoC}(V, U)$ and the (known) accuracy of h on V.

The *Dispersion Score* (DS) method of Xie et al. (2023) similarly relies on a regressor. The score on which the regressor is trained (the DS) is aimed at quantifying class separability in the feature space, on the grounds that good separation should correlate with high accuracy, and is computed as $\mathrm{DS} = \log \sum_{i=1}^{n} \left(m_i \cdot ||\bar{\boldsymbol{\mu}} - \tilde{\boldsymbol{\mu}}_i||_2^2 \right) - \log(n-1)$, i.e., a weighted sum of squared distances between the class centroids ($\tilde{\boldsymbol{\mu}}_i$) and the global centroid ($\bar{\boldsymbol{\mu}}$), where m_i is the number of datapoints predicted to belong to the ith class.

The Nuclear Norm (NN) method of Deng et al. (2023) extends the idea of DS (Xie et al., 2023), by proposing a new score which combines, using the NN of the matrix of the predicted posteriors, two facets of the problem, i.e., the *confidence* of the predictions, and their *dispersion*, where the latter measures the extent to which the predicted classes are diverse and well-distributed.

The *Reverse Classification Accuracy* (RCA) method of Elsahar and Gall (2019) is based on a "back-and-forth" process in which the classifier h to be evaluated is compared with a classifier h' trained on the (pseudo-)labels generated by h on U; both classifiers are then applied to validation set V, and their difference in accuracy (the "RCA score") is input to a regressor that returns the estimated accuracy of h on U. In order to train this regressor, a number of validation samples V_i must be available, where a training example for training the regressor is composed of (a) the difference in the accuracy values obtained on V by classifier h and a classifier h' trained on V_i (the covariate), and (b) the accuracy of h on V_i (the label).

Mandoline (Chen et al., 2021a) belongs instead to the family of "importance-weighting methods" (Sugiyama et al., 2007). Methods in this family try to estimate the accuracy of h on U using estimates $\hat{r}(\mathrm{x})$ of the true *density ratio* $r(\mathrm{x}) = \frac{q(\mathrm{x})}{p(\mathrm{x})}$, with p and q the density functions of the train and test probability distributions P and Q, respectively. Mandoline relies on user-defined so-called "slicing" functions to create common representations and compare training data with test data.

Chen et al. (2021b) exploit model agreement in an iterative framework, training an ensemble of auxiliary models for each iteration and using their agreement ratio to estimate the accuracy of the original model.

Generalisation Disagreement Equality (GDE) (Jiang et al., 2022) is instead based on the observation that the degree of disagreement on in-distribution (i.e., IID) data between identical neural architectures that have been initialised differently correlates strongly with their accuracy on (non-IID) test data.

The *Confidence Optimal Transport* (COT) method of Lu et al. (2023) frames the CAP problem as one of optimal transport, i.e., the task of finding the best plan for transporting the mass of posterior probabilities generated by the classifier on the test sample, to the distribution of ground-truth labels of the same sample (represented as one-hot vectors), where the target distance to minimize is the Earth Mover's Distance (EMD). As the true labels of the sample are unknown, the authors propose replacing them with the training labels, under the assumption that both sets exhibit approximately similar prevalence values. However, note that this assumption does not hold under PPS, which makes the method unsuitable to our case. In Sect. 5.5 we thus propose an adaptation of this method to PPS scenarios, and use it as a baseline in our experiments.

The *Confidence-Based Performance Estimators* (CBPE) method of Kivimäki et al. (2025) follows a different approach, close in spirit to ours, based on the idea of predicting the cells of the contingency table resulting from the application of the classifier to the unlabelled test sample.[2] Once the contingency table is predicted, many different accuracy measures can be computed. The method comes down to grouping the posterior probabilities by their predicted class label, and then using the average of the posteriors in each group scaled by the fraction of instances falling within the group, to populate the contingency table. The method, however, assumes that the classifier is well-calibrated for the target distribution, a condition which is not easy to attain under dataset shift (Ovadia et al., 2019). In Sect. 5.5 we propose a variant of it that calibrates the classifier for the target distribution before estimating the contingency table, and use it as a baseline in our experiments.

The methods we present in this paper are very different from all the methods we have mentioned above since, unlike the latter, they are *not* based on estimating an accuracy measure directly, but are based on estimating (by solving a system of linear equations) the cells of the contingency table, from which any measure can in turn be computed. In cases where more than one evaluation measure needs to be estimated, the methods presented above require retraining, while our methods allow estimating several evaluation measures at once.

3 Background

3.1 Notation

Throughout this paper we use the following notation. By $\mathcal{X} = \mathbb{R}^d$ and $\mathcal{Y} = \{\omega_1, \ldots, \omega_n\}$ we denote the input space (vectors of covariates) and the output space (classes), respectively; when dealing with binary problems we will use the shorthand $\mathcal{Y} = \{\oplus, \ominus\}$. By $D = \{(\mathrm{x}_l, y_l)\}_{l=1}^m$ we denote a generic labelled set consisting of pairs $\mathrm{x}_l \in \mathcal{X}$ and $y_l \in \mathcal{Y}$, that we will use for training or testing our models.

By $h : \mathcal{X} \to \mathcal{Y}$ we denote a (crisp) classifier mapping vectors of covariates into classes. By $h : \mathcal{X} \to \Delta^{n-1}$ we instead denote a probabilistic classifier mapping vectors of covariates into vectors of posterior probabilities lying in the probability simplex $\Delta^{n-1} = \{(\delta_1, \ldots, \delta_n) : \delta_i \geq 0, \sum_{i=1}^n \delta_i = 1\}$. From a probabilistic classifier one can easily obtain a crisp classifier by returning, for a given datapoint, the class corresponding to the largest posterior probability.

3.2 Dataset Shift

Dataset shift is defined as the situation in which the training set is drawn from a joint distribution P (hereafter: "the training distribution") and the test set is drawn from another joint distribution Q ("the test distribution") such that $P(X, Y) \neq Q(X, Y)$.

In order to analyse the type of dataset shift at play, it is useful to decompose a joint distribution into factors. may be factored as

[2] Note that this idea was first proposed, in 2024, in the earlier version of the present paper (Volpi et al., 2024; Kivimäki et al., 2025) independently developed it in 2025, unaware of our previous publication.

$$D(X,Y) = D(Y|X)D(X) \tag{1}$$

or as

$$D(X,Y) = D(X|Y)D(Y) \tag{2}$$

While both factorisations are correct, the one of Eq. 1 is more conventient when analysing *causal* learning problems, since factor makes explicit the dependence of the label on the covariates (i.e., on the *causes* of the phenomenon under study. Symmetrically, the factorisation of Eq. 2 is more convenient when analysing *anti-causal* learning problems, since factor makes explicit the dependence of the covariates (i.e., the sumptoms, or efects, of the phenomenon) on the label.

In this paper we focus on prior probability shift, a type of dataset shift typical of anti-causal learning problems; we will thus only use the factorisation of Eq. 2.

3.3 Prior Probability Shift

Prior probability shift (PPS—sometimes called *target shift* (Zhang et al., 2013) or *label shift* (Lipton et al., 2018)) is a type of dataset shift defined as the case in which

$$\begin{aligned} P(Y) &\neq Q(Y) \\ P(X|Y) &= Q(X|Y) \end{aligned} \tag{3}$$

i.e., the case in which the prevalence values of the classes can change between the training distribution and the test distribution but the class-conditional distribution of the covariates does not change. If $Z = f(X)$, with f an X-measurable function, from $P(X|Y) = Q(X|Y)$ it follows (see Tasche, 2024) that $P(Z|Y) = Q(Z|Y)$. In particular, if we take $f = h$, from $P(X|Y) = Q(X|Y)$ follows that the distribution of the class-conditional predictions $\hat{Y}$ issued by the classifier is the same for the training distribution P and the test distribution Q, i.e., it holds that

$$P(\hat{Y}|Y) = Q(\hat{Y}|Y) \tag{4}$$

3.4 Problem setting

We assume a classifier h trained on a set L of labelled datapoints, which we assume may no longer be available; we also assume a validation set V of labelled datapoints, with L and V drawn from the same distribution P. CAP consists of predicting the accuracy that our classifier h will exhibit on a given test set U (for which we assume the labels are not available) in terms of an accuracy measure A, with U drawn from a distribution Q such that P and Q are related by PPS. Note that h is assumed already trained, and that our goal is *not* to improve its performance on U but just to estimate how well it will fare on U.

4 Method

Most popular measures used for evaluating the performance of a classifier h can be computed in terms of a *contingency table* (also known as a *confusion matrix*) that relates the outcomes of the random variable Y (which denotes the true class of a datapoint) with those of the random variable $\hat{Y}$ (which denotes the class predicted by h of the datapoint). A contingency table is a matrix where entry (i, j) denotes the number of datapoints whose true class is ω_i and whose predicted class is ω_j. However, without loss of generality, from now on we will take the value of each entry to be normalized by the total number of datapoints, i.e., we will consider entry (i, j) of a contingency table as the *fraction* of datapoints whose predicted class is ω_i and whose true class is ω_j; in other words, given that these fractions sum up to 1, we will see a contingency table as an empirical *probability distribution*.

Let us consider the case in which the contingency table is obtained by applying a classifier h to a validation set $V = \{(\mathrm{x}_l, y_l)\}_{l=1}^m$ drawn from the training distribution P. Applying h to V maps V into $V' = \{(h(\mathrm{x}_l), y_l)\}_{l=1}^m$, from which we can obtain a contingency table V with entries

$$c_{ij}^V = \frac{|\{(h(\mathrm{x}_l), y_l) \in V' \ : \ h(\mathrm{x}_l) = \omega_i \ \wedge \ y_l = \omega_j\}|}{|V'|} \tag{5}$$

Note that, when $|V|$ is large enough, it holds that $c_{ij}^V \approx P(\hat{Y} = \omega_i, Y = \omega_j)$.

As noted in the introduction, if the distributions P and Q from which V and U (resp.) are drawn are related by PPS, the contingency table V cannot be used to reliably estimate the accuracy of h on U. Ideally, to estimate the accuracy of h on U we would like to directly estimate the entries c_{ij}^U of the contingency table U that results when h classifies the datapoints in U, but this is not trivial, since on U we can observe the predicted labels (by simply applying h to the datapoints of U) but not the true labels. If we had access to these true labels, any evaluation measure could be computed; for example, vanilla accuracy for a test set U can be expressed as

$$\mathrm{Acc}(h, U) = \sum_{i=1}^{n} c_{ii}^U \tag{6}$$

(where the absence of denominators is due to the fact that c_{ii}^U is not a count but a fraction).

In the binary case we will indicate fractions c_{11}^V and c_{11}^U as TP^V and TP^U, respectively, where TP stands for "true positives" and the V and U subscripts indicate from which set of datapoints (the validation set V or the test set U) these fractions originate[3] analogously, FP, FN, TN, will stand for "false positives", "false negatives", "true negatives", respectively. We will also use the shorthands

$$\mathrm{tpr} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} \quad \mathrm{fpr} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}} \quad \mathrm{tnr} = \frac{\mathrm{TN}}{\mathrm{FP} + \mathrm{TN}} \quad \mathrm{fnr} = \frac{\mathrm{FN}}{\mathrm{TP} + \mathrm{FN}} \tag{7}$$

[3] The term "true positives" is typically used to refer to the *number*; (and not the *prevalence*) of datapoints which have correctly been predicted to belong to class $\oplus$; however, in this paper it is useful for us to always think in terms of prevalence values.

to refer to the "true positive rate", the "false positive rate", the "true negative rate", and the "false negative rate" of classifier h. We use the suffix V or U to indicate the contingency table in which the rate has been computed, e.g., tpr^V or tpr^U.

4.1 The governing equations of the problem

In this section we show that the values c_{ij}^U that we would like to estimate need to satisfy a number of constraints, and that these constraints are linear equations (the *governing equations* of our problem) in which the c_{ij}^U's are the unknowns. Since the number of values c_{ij}^U that we want to estimate is n^2, if we have a system of n^2 independent equations we can estimate them all by solving it. As anticipated in the introduction, the main contribution of this paper is a simple yet effective family of methods for CAP, called LEAP (for "Linear Equations for Classifier Accuracy Prediction"), that are indeed based on identifying a system of n^2 independent equations whose solution allows us to estimate the c_{ij}^U values (and, from them, any accuracy measure based on a contingency table).

Sections 4.1.1 to 4.1.4 each describe a group of linear equations, and our method consists of solving (in one of several ways described in Sects. 4.2 to 4.5) the system of linear equations resulting from pooling together these four groups. Each section from 4.1.1 to 4.1.4 describes what the respective group of equations amounts to (i) in the multiclass case ($n > 2$), and, as a particular case, (ii) in the binary case ($n = 2$). The multiclass methods are the ones resulting from pooling together the equations for the multiclass case, while the binary methods are the ones resulting from pooling together the equations for the binary case. So, Sects. 4.1.1 to 4.1.4 altogether describe both our multiclass methods and, as a particular case, our binary methods.

Before proceeding to the next sections, let us recall that $0 \leq c_{ij}^U \leq 1$ for all $i, j \in \{1, \ldots, n\}$.

4.1.1 The first group of equations

This group actually consists of a single (trivial) equation; the entries of the contingency table are probabilities of disjoint events that cover the entire event space, and they thus must sum to one, which means that our first equation is

$$\sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij}^U = 1 \tag{8}$$

In the binary case, this corresponds to equation

$$\text{TP}^U + \text{FP}^U + \text{FN}^U + \text{TN}^U = 1 \tag{9}$$

4.1.2 The second group of equations

This group relies on the fact that the sum of the entries in row i of U must equal the (known) fraction of datapoints in U that h has assigned to class ω_i. Let us define the vector $\boldsymbol{\gamma} = (\gamma_1, \ldots, \gamma_n) \in \Delta^{n-1}$ of these observed fractions, where

$$\gamma_i = \frac{1}{|U|} \sum_{\mathrm{x} \in U} \mathbf{1}[h(\mathrm{x}) = \omega_i] \tag{10}$$

This adds to the system n equations of type

$$\sum_{j=1}^{n} c_{ij}^{U} = \gamma_i \qquad \text{for } i \in \{1, \ldots, n\} \tag{11}$$

At this point, it is worth noting that the $(n+1)$ equations deriving from Eqs. 8 and 11 are not independent of each other; indeed, one could pick any of them and obtain it as a linear combination of the other n. However, we put aside this problem for the moment being, and will come back to it in the next section.

In the binary case, Eqs. 11 amount to

$$\begin{aligned} \mathrm{TP}^U + \mathrm{FP}^U &= \gamma_{\oplus} \\ \mathrm{FN}^U + \mathrm{TN}^U &= \gamma_{\ominus} \end{aligned} \tag{12}$$

4.1.3 The third group of equations

This group derives from the assumption that the type of dataset shift affecting distributions P and Q is PPS. As indicated in Sect. 3.2, this entails that $P(\hat{Y}|Y) = Q(\hat{Y}|Y)$ (see Eq. 4), which means that the classification rates do not change across the training and test distributions. Since the validation set V is sampled from the training distribution P, this means that

$$\frac{c_{ij}^{U}}{\sum_{k=1}^{n} c_{kj}^{U}} = \frac{c_{ij}^{V}}{\sum_{k=1}^{n} c_{kj}^{V}} \qquad \text{for } i, j \in \{1, \ldots, n\} \tag{13}$$

If we use the shorthand

$$r_{ij}^{V} = \frac{c_{ij}^{V}}{\sum_{k=1}^{n} c_{kj}^{V}} \tag{14}$$

(where r_{ij}^{V} denotes the classification rates in V) we can rewrite Eq. 13 as

$$c_{1j}^{U} \cdot r_{ij}^{V} + \cdots + c_{ij}^{U} \left(r_{ij}^{V} - 1\right) + \cdots + c_{nj}^{U} \cdot r_{ij}^{V} = 0 \qquad \text{for } i, j \in \{1, \ldots, n\} \tag{15}$$

Since values r_{ij}^{V} can be computed by classifying the datapoints in V via h, Eq. 15 is actually a set of n^2 equations (one for each value r_{ij}^{V}) in n^2 unknowns (the c_{ij}^{U} values).

In the binary case, this is expressed through one equation for each classification rate (tpr, fpr, tnr, fnr). For tpr (the other three cases are analogous) we have

$$\frac{\mathrm{TP}^U}{\mathrm{TP}^U + \mathrm{FN}^U} = \mathrm{tpr}^V \tag{16}$$

which can be more conveniently written as

$$\mathrm{FN}^U \cdot \mathrm{tpr}^V + \mathrm{TP}^U \cdot (\mathrm{tpr}^V - 1) = 0 \tag{17}$$

where FN^U and TP^U are the unknowns. Note that, as for the second group of equations, the $(n^2 + 1)$ equations deriving from Eqs. 8 and 15 are not independent; e.g., in the binary case, one of the four equations of which Eq. 17 is an example is, if taken together with Eq. 9, redundant. Here too, we momentarily set aside the issue of independence and defer its discussion to later on.

4.1.4 The fourth group of equations

This group relies on the fact that, in contingency table U, the sum of the elements on column j is the true prevalence of class ω_j in the test set U. At first sight this would seem hardly useful, since the true prevalence values of the classes in U are unknown. However, it turns out that these values can be estimated from contingency table V with the aid of Eq. 4. This is best illustrated in the binary case by noting that

$$\begin{aligned} Q(\hat{Y} = \oplus) &= Q(\hat{Y} = \oplus | Y = \oplus)\, Q(Y = \oplus) + Q(\hat{Y} = \oplus | Y = \ominus)\, Q(Y = \ominus) \\ &= P(\hat{Y} = \oplus | Y = \oplus)\, Q(Y = \oplus) + P(\hat{Y} = \oplus | Y = \ominus)\, Q(Y = \ominus) \end{aligned} \tag{18}$$

where the first step derives from the law of total probability and the second step derives from the fact that, since we assume PPS, Eq. 4 holds. By taking into account (i) the fact that $Q(Y = \ominus) = (1 - Q(Y = \oplus))$; (ii) the fact that, because of Eq. 4, $P(\hat{Y} = \oplus | Y = \oplus) \approx \mathrm{tpr}^V$ and $P(\hat{Y} = \oplus | Y = \ominus) \approx \mathrm{fpr}^V$, and (iii) the fact that an estimate of $Q(\hat{Y} = \oplus)$ (denoted by $\gamma_\oplus$) can be obtained from Eq. 10, we obtain that

$$Q(Y = \oplus) \approx \frac{\gamma_\oplus - \mathrm{fpr}^V}{\mathrm{tpr}^V - \mathrm{fpr}^V} \tag{19}$$

Since $\gamma_\oplus = \mathrm{FP}^U + \mathrm{TP}^U$ and $Q(Y = \oplus) \approx \mathrm{FN}^U + \mathrm{TP}^U$ Eq. 19 leads to equation

$$\mathrm{FP}^U + \mathrm{FN}^U \cdot (\mathrm{fpr}^V - \mathrm{tpr}^V) + \mathrm{TP}^U \cdot (1 + \mathrm{fpr}^V - \mathrm{tpr}^V) = \mathrm{fpr}^V \tag{20}$$

An analogous to Eq. 19 can be formulated for $Q(Y = \ominus)$, which generates an equation analogous to Eq. 20. In the multiclass case, there are n such equations. The derivation is slightly more complicated, as it entails writing Eq. 18 as a system of linear equations, one per class, that involves the classification rates $P(\hat{Y} = \omega_i | Y = \omega_j)$. This problem can be written in matrix form as $\boldsymbol{\gamma} = \mathrm{R}^\top \mathrm{q}$, where $\boldsymbol{\gamma} = (\gamma_1, \ldots, \gamma_n) \in \Delta^{n-1}$ is the vector representing the distribution of the predicted labels, R is our matrix of classification rates, and

$\mathrm{q} = (q_1, \ldots, q_n) \in \Delta^{n-1}$ is the sought class distribution on the test data. The solution thus comes down to solving the system as $\hat{\mathrm{q}} = (\mathrm{R}^\top)^{-1}\boldsymbol{\gamma}$, and then adding the *n* equations

$$\sum_{i=1}^{n} c_{ij}^{U} = \hat{q}_j \qquad \text{for } j \in \{1, \ldots, n\} \tag{21}$$

to our system.

4.1.5 Summing up...

To sum up, by taking the first equation (Eq. 8), the *n* equations from the second group (Eq. 11), the n^2 equations from the third group (Eq. 15), and the *n* equations from the fourth group (Eq. 21), we obtain a system of exactly $(1 + n + n^2 + n) = (1 + n)^2$ equations. Since the number of unknowns is only n^2, the problem is *overconstrained*. In the following sections we discuss different alternatives (which give rise to different CAP systems in the LEAP family) for dealing with this overabundance of constraints, including (a) two reasonable choices of n^2 equations that a linear solver can easily address, and (b) a method that takes all the equations into account.

4.2 S-LEAP: a direct solution based on the PPS assumptions

Perhaps the simplest way to address the problem is to start from the n^2 equations represented by Eq. 13, i.e., the ones that derive exclusively from the fact that *P* and *Q* are related by PPS. Here, we observe that the denominators of the classification rates involved in Eq. 13 turn out to be nothing else than the class priors of the validation and test sets. More formally, let $\mathrm{p} = (p_1, \ldots, p_n) \in \Delta^{n-1}$ and $\mathrm{q} = (q_1, \ldots, q_n) \in \Delta^{n-1}$ represent the (true) class priors in the validation and test sets, respectively, where $p_j = \sum_{i=1}^{n} c_{ij}^{V}$ and $q_j = \sum_{i=1}^{n} c_{ij}^{U}$. Equation 13 can thus be rewritten as

$$c_{ij}^{U} = c_{ij}^{V} \cdot \frac{q_j}{p_j} \qquad \text{for } i, j \in \{1, \ldots, n\} \tag{22}$$

which allows to directly estimate each entry of the test contingency table by simply rescaling the corresponding entry from the validation contingency table.

Of course, while vector p is observed, q remains unknown. However, in Sect. 4.1.4 we showed a method for estimating it, which involves solving the problem $\boldsymbol{\gamma} = \mathrm{R}^\top \mathrm{q}$. By doing so, we reduce the original problem, which consists of solving a system of n^2 linear equations, to one of solving a linear system of only *n* equations. We call this method S-LEAP, since it is akin to *scaling* the contingency table entries by the ratios q_j/p_j between the priors.

While this method is appealing and direct, it is not without problems. The method relies entirely and solely on the PPS assumptions, which may not hold *exactly* in practice, because of the presence of additional types of shift (e.g., covariate shift), or simply because of random sampling error. Aside from this, by relying on a single group of equations, many aspects of the problem are hopelessly lost. For instance, excluding the second group of equations may well lead to column sums deviating from the known observed counts $\boldsymbol{\gamma} = (\gamma_1, \ldots, \gamma_n)$.

4.3 LEAP$_{\mathrm{ACC}}$: selecting a more representative subset of equations

One way to address the limitations of the S-LEAP method is to involve all four batches of equations into the final selection of n^2 equations. The method we present here follows this strategy, by selecting from the pool (i) the (only) equation from the first group (Eq. 8), (ii) $(n-1)$ equations from the second group (Eq. 11), (iii) $(n-1)^2$ equations from the third group (Eq. 15), and (iv) $(n-1)$ equations from the fourth group (Eqs. 21), thus totalling exactly n^2 equations. The rationale behind this choice is to select equations from all batches while ensuring that no redundant (i.e., linearly dependent) equations are taken from the same group. The choice of the equation to be left out from each group is arbitrary; we choose to drop the equation corresponding to the index $i = j = 1$, i.e., to the first class ω_1.

In this case, it is interesting to see that the resulting system of four equations in the binary case is

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & \mathrm{tpr}^V & 0 & (\mathrm{tpr}^V - 1) \\ 0 & 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} \mathrm{TN}^U \\ \mathrm{FN}^U \\ \mathrm{FP}^U \\ \mathrm{TP}^U \end{pmatrix} = \begin{pmatrix} 1 \\ \gamma_\oplus \\ 0 \\ q_\oplus \end{pmatrix} \tag{23}$$

which has the solution

$$\begin{aligned} \mathrm{TN}^U &= 1 - \gamma_\oplus + q_\oplus(\mathrm{tpr}^V - 1) \\ \mathrm{FN}^U &= q_\oplus(1 - \mathrm{tpr}^V) \\ \mathrm{FP}^U &= \gamma_\oplus - q_\oplus \cdot \mathrm{tpr}^V \\ \mathrm{TP}^U &= q_\oplus \cdot \mathrm{tpr}^V \end{aligned} \tag{24}$$

We can now derive the closed form of many well-known evaluation measures under PPS. For example, vanilla accuracy as computed on the test contingency table U (Eq. 25) can be expressed in compact form with the aid of $\boldsymbol{\gamma}$ and q (Eq. 26), or expanded as a function that uses only $\boldsymbol{\gamma}$ along with counts from the validation contingency table V (Eq. 27), as

$$\mathrm{Acc}(h, U) = \frac{\mathrm{TP}^U + \mathrm{TN}^U}{\mathrm{TP}^U + \mathrm{TN}^U + \mathrm{FN}^U + \mathrm{FP}^U} \tag{25}$$

$$= 1 + q_\oplus(2\mathrm{tpr}^V - 1) - \gamma_\oplus \tag{26}$$

$$= \frac{(\gamma_\oplus \cdot (\mathrm{TN}^V + \mathrm{FP}^V) - \mathrm{FP}^V)(\mathrm{TP}^V - \mathrm{FN}^V)}{\mathrm{TN}^V \mathrm{TP}^V - \mathrm{FP}^V \mathrm{FN}^V} + 1 - \gamma_\oplus \tag{27}$$

The fact that this evaluation measure can be expressed in closed form in binary problems is interesting, because it means that in order to perform CAP we simply need to compute Eq. 27 from the counts obtained from the contingency table of the validation set, *without actually solving a system of linear equations at prediction time*. In Sect. 5.14, we show analogous results for the case in which we use F_1 as our evaluation measure instead of Acc.

When, in the equations above, we replace the true prior q with our estimate $\hat{\mathrm{q}}$ (obtained via Eq. 19), we obtain an instance of LEAP that we dub $\mathrm{LEAP}_{\mathrm{ACC}}$. Note that the compact formula above (Eqs. 26) is generic, and can be used also in cases in which the class prevalence q is obtained by means of other techniques. We discuss this topic in the following section, in which the rationale behind the name $\mathrm{LEAP}_{\mathrm{ACC}}$ will become evident.

4.4 $\mathrm{LEAP}_{\mathrm{KDEy}}$: enhancing LEAP via quantification

The fourth group of equations (see Sect. 4.1.4) requires estimating the class prevalence values q_i of the test set. The solution we have shown in Eqs. 18 and 19 (upon which $\mathrm{LEAP}_{\mathrm{ACC}}$ relies) is actually borrowed from a well-known quantification (i.e., class prevalence estimation) method, called *adjusted classify and count* (ACC) (Forman, 2005, 2008; Bella et al., 2010); the method has also been called "Black-Box Shift Estimator" (BBSE) (Lipton et al., 2018). This method is of particular interest for our derivation since it only involves components (such as tpr and fpr) which can be derived from the contingency table of the validation set. The implementation we use in this paper incorporates the improvements brought about by Bunse (2022), which come down to better handling the cases in which the matrix $\mathrm{R}^{\top}$ of the classification rates is non-invertible.

However, ACC is by no means the only method available for estimating class prevalence values, nor the most sophisticated one, and there is an entire body of literature devoted to devising better ways for estimating these values (González et al., 2017; Esuli et al., 2023). In this section, we propose an enhanced version of our LEAP method where more sophisticated quantification algorithms are used for solving the fourth group of equations.

More formally, a quantifier is a function $\lambda : \mathcal{S} \to \Delta^{n-1}$ mapping samples $S \in \mathcal{S}$ of instances from the input space $\mathcal{X}$ to the probability simplex, where $\lambda(S) = \hat{\mathrm{q}}$ is an estimate of the true class prior distribution q in S. We focus our attention to the so-called *distribution-matching* approaches to quantification, which take a permutation-invariant function Φ to represent samples S, and solve for q (the sought test class prevalence vector) the equation $\mathrm{t} = \mathrm{z}^{\top}\mathrm{q}$ where $\mathrm{t} = \Phi(U)$ is the representation of test sample U and $\mathrm{z} = [\Phi(V_1), \ldots, \Phi(V_n)]$ contains the class-wise representations of validation sets $V_i = \{\mathrm{x}_k : (\mathrm{x}_k, y_k) \in V, y_k = \omega_i\}$. Specifically, we adopt the KDEy-ML variant proposed in Moreo et al. (2025) (hereafter simply called KDEy), which relies on kernel density estimation for representing the (posterior probabilities of the) samples as density functions in the probability simplex. KDEy then solves the distribution matching problem via maximum likelihood, i.e., it computes

$$\hat{\mathrm{q}} = \underset{\mathrm{q} \in \Delta^{n-1}}{\arg\min}\, \mathcal{D}_{\mathrm{KL}}(\mathrm{t} \;||\; \mathrm{z}^{\top}\mathrm{q}) \tag{28}$$

where $\mathcal{D}_{\mathrm{KL}}$ is the well-known Kullback–Leibler divergence between the density model of the test set and the density model of a mixture parameterized by $\hat{\mathrm{q}}$. We call this extension $\mathrm{LEAP}_{\mathrm{KDEy}}$. (Using a similar notation, by $\mathrm{S\text{-}LEAP}_{\mathrm{KDEy}}$ we will denote the S-LEAP method enhanced by the use of KDEy for estimating the class prevalence values.)

Yet another fundamental difference between $\mathrm{LEAP}_{\mathrm{KDEy}}$ and $\mathrm{LEAP}_{\mathrm{ACC}}$ lies in the fact that KDEy internally uses a surrogate classifier trained on the validation set, distinct from the target classifier h whose accuracy we aim to estimate. This seems a reasonable choice,

as the tool used to measure the classifier's accuracy should arguably be independent of the classifier itself.[4]

4.5 O-LEAP: overconstrained LEAP

While the method discussed in Sect. 4.2 can be solved directly, the methods presented in Sects. 4.3 and 4.4 require solving a system of n^2 equations. Such methods can be expressed in matrix form as $\mathrm{AX} = \mathrm{b}$, where $\mathrm{X} = (c^U_{11}, \ldots, c^U_{nn})$ represents the unknowns. However, in degenerate cases in which the PPS assumptions do not hold, or the estimated class prevalence values deviate much from the true value, the system may not be solvable via matrix inversion ($\mathrm{X} = \mathrm{A}^{-1}\mathrm{b}$), or the solution may be unfeasible (e.g., some $\hat{c}^U_{ij}$ may fall outside the [0, 1] range). In cases like this, we may resort to solving the constrained problem $\hat{\mathrm{X}} = \arg\min_{\mathrm{X}\in\Delta^{n^2-1}} ||\mathrm{AX} - \mathrm{b}||_2$, with $0 \leq c^U_{ij} \leq 1$; modern software packages solve this type of system efficiently (more on this in Sect. 5.12).

That said, all the methods discussed so far sacrifice, in one way or another, some of the problem's constraints (exactly $1 + 2n$ of them are dropped). This causes an asymmetry in the way different classes are treated in the solution.[5] We thus introduce a new method that instead incorporates all our $1 + 2n + n^2$ constraints; we call this method O-LEAP, for "Overconstrained LEAP", as it deals with an overconstrained system of equations ($1 + 2n + n^2$ equations for only n^2 unknowns); when we want to explicitly indicate the type of quantification method used we indicate it as a subscript (say, O-LEAP$_{\mathrm{KDEy}}$). The matrix $\mathrm{A} \in \mathbb{R}^{(1+2n+n^2)\times n^2}$ that derives from the entire set of constraints is rectangular, and thus non-invertible. However, the same strategy employed for "degenerate cases" can be used here, that is, we minimize the Frobenius norm of the residual $\mathrm{AX} - \mathrm{b}$ to find a solution in the overconstrained setting, aiming to satisfy, at least approximately, all the equations simultaneously.

5 Experiments

In this section we turn to describing the experiments we have carried out in order to assess the effectiveness of the LEAP family of methods.

5.1 Experimental setup

The effectiveness measure we use in order to assess the quality of the predictions of CAP methods is absolute error (AE). For a generic classifier accuracy measure A, AE is defined as $|A(h, U) - \hat{A}(h, U)|$, where $A(h, U)$ is the true accuracy of h on U (which in lab experi-

[4] As the surrogate classifier, that we train via cross-validation on the validation set, we choose the implementation of Multi-Layer Perceptron (MLP) available in scikit-learn, with default hyperparameters. Preliminary experiments we have run show that this choice is a good one.

[5] Indeed, note that there are no fewer than $\binom{1+2n+n^2}{n^2} = \frac{(1+2n+n^2)!}{n^2(1+2n)!}$ possible ways of dropping $(1 + 2n)$ constraints; of course, the idea of exploring them all is hopeless.

ments we can determine since we know the actual labels of U), and where $\hat{A}(h, U)$ is the accuracy of h on U as predicted by the CAP method.

As our classifier accuracy measure A we here employ vanilla accuracy (Acc – see Eq. 6). Aside from the fact that it is one of the most important measures of classifier effectiveness, we choose Acc for the sake of fair experimental comparison, given that some of our baselines only work for this evaluation measure.

5.2 Experimental protocol

The experimental protocol we adopt is as follows. Given a labelled collection $D = \{(\mathrm{x}_l, y_l)\}_{l=1}^{m}$, we split it into a training set (70%) and a test set U (30%) via stratified sampling. We further split the training set into a set L (50%) that we use for training a classifier h, and a validation set V (50%) that we use for training a CAP method, again using stratification. We train and test all competing methods on exactly the same partitions.

We then draw, uniformly at random, 1000 vectors $\mathrm{v}_1, ..., \mathrm{v}_{1000}$ of prevalence values from the unit simplex Δ^{n-1} (using the Kraemer sampling algorithm—Smith & Tromble, 2004), and for each vector v_i we randomly extract from U a bag U_i of $|U_i| = 100$ elements such that U_i satisfies the prevalence distribution of v_i. This method of generating bags U_i each complying with a class prevalence distribution sampled uniformly at random from the unit simplex, is called the *Artificial Prevalence Protocol* (APP), and is the standard protocol for simulating prior probability shift (Esuli et al., 2023; Forman, 2008). Note that L and V (as well as the original U) are from the same distribution P, while the test samples U_i extracted from U are instead from a different distribution Q_i which is related to P via PPS.

Indeed, note that this experimental protocol clearly simulates PPS, since the distribution of the covariates X conditional on the distribution of the labels Y is the same for the the training set and the U_i's, but the distribution of the labels Y is not the same for the training set and the U_i's (in the language of Sect. 3.3, our protocol simulates a situation in which $P(X|Y) = Q(X|Y)$ and $P(Y) \neq Q(Y)$, i.e., enforces the PPS assumptions).

For each test sample U_i we ask the CAP method to predict the accuracy (Acc) of h. We then compare (via AE) the predicted score with the true one and report averaged values across the 1000 tests.

5.3 Datasets

As our datasets, we use 29 binary datasets and 24 multiclass datasets from the UCI machine learning repository (Kelly et al., n.d.). We restrict our attention to those datasets that can be directly imported through the UCI's Python API, and discard datasets with fewer than 3 features or fewer than 150 instances. The number of instances in the binary datasets varies from a minimum of 150 (iris.2, iris.3) to a maximum of 5,473 (pageblocks.5), while the number of features varies from a minimum of 3 (haberman) to a maximum of 256 (semeion). The training sets display varying degrees of balance, ranging from datasets with only 2.1% positive instances (pageblocks.5) to almost perfectly balanced datasets (mammographic), to datasets with 77.8% positive instances (ctg.1). For the multiclass case, the number of instances varies from 1,014 instances (mhr) to 1,024,985 (poker-hand), while the number of features varies from 7 (shuttle) to 227 (molecular). The number of classes varies from a 3 (mhr, connect-4, molecular, waveform-v1, cmc, academic-success)

to 26 (letter and isolet). Class balance varies from highly imbalanced datasets, with some classes representing less than 1% of instances (e.g., one of the classes in poker-hand), to perfectly balanced datasets (e.g., the classes of image-seg).

5.4 Classifier-learning algorithms

For generating the classifiers whose accuracy we want to test, we consider five learning algorithms, i.e., logistic regression (LR), kNN (with $k = 10$), support vector machines (SVMs—using the RBF kernel), multilayer perceptron (MLP – with hidden sizes of 100 and 15 neurons and the ReLU activation function), and random forests (RFs). In all cases, we rely on the implementations provided by scikit-learn (Pedregosa et al., 2011), leaving the rest of the hyperparameters at their default values.

5.5 Baselines

As the baselines against which we test the performance of our methods, we use ATC (with the *maximum confidence* function) (Garg et al., 2022), DoC (Guillory et al., 2021), DS (Xie et al., 2023), L-CBPE (our own variant of CBPE, Kivimäki et al., 2025, see below), NN (Deng et al., 2023), and Q-COT (our own variant of COT, Lu et al., 2023, see below); the original methods are all discussed in Sect. 2.

L-CBPE is our own variant of CBPE (Kivimäki et al., 2025), a variant which applies LasCal (Popordanoska et al., 2024), a calibration method specifically devised for PPS, before estimating the contingency table.[6]

While the proposers of NN (Deng et al., 2023) do not provide details about the regression model they use, we adopt the same strategy used in DoC and DS, namely, training a regressor on the NN score.

Q-COT is our own adaptation of Lu et al. (2023)'s COT method to operate under PPS. The original COT method assumes that the training label distribution is representative of the test distribution, an assumption that is violated under PPS. In our variant, Q-COT first quantifies the class prevalence values in the test sample using KDEy, and then generates, via resampling, a training distribution that matches these prevalences. From this point on, Q-COT is identical to COT.

For ATC, Mandoline, RCA, and COT (the original method on top of which we build Q-COT) we use the code provided by the authors, while for DoC, DS, NN, and L-CBPE we use our own implementation since no user-provided code is available. We do not report results for Mandoline (Chen et al., 2021a) and RCA (Elsahar & Gall, 2019) since, in the experiments we have run, they are competitive neither with methods in the LEAP family, nor with the baselines we do include.[7] Similarly, we do not include the methods proposed in Chen et al.(2021b) and Jiang et al. (2022) since they are based on assumptions (namely, that the classifiers whose accuracy has to be predicted are trained via neural networks)

[6] Kivimäki et al. (2025) do not specify the calibration method they use in their experiments.

[7] The likely reason why the performance of Mandoline is not competitive, in our experiments, with other baselines is that Mandoline heavily relies on user-defined transformations, called *slicing functions* which require manual intervention, for each dataset, on the part of the designer. Regarding RCA, it is likely that the "back-and-forth" approach is not able to retain the correct amount of information about the accuracy of h during the training of a clone h' on the pseudo-labelled set.

which are too restrictive, and which ultimately do not match our setting. We report the results of Naïve, a method that assumes the training and test data to be IID, and reports the vanilla accuracy score obtained on the contingency table generated by h on V.

For the implementation of the quantification algorithms (ACC and KDEy) we rely on the QuaPy[8] package (Moreo et al., 2021). The code to reproduce all our experiments is available on GitHub.[9]

5.6 Results

Tables 1 (for the binary case) and 2 (for the multiclass case) show the results we obtain in our experiments in terms of AE (lower is better) for the LR classifier; the tables for kNN, SVMs, MLP, display similar trends, and are reported in Appendix A.

Overall, our results show that O-LEAP tends to obtain, on average, the best results across all four classifiers, both in the binary case and in the multiclass case. When considering either the best result for each dataset (in boldface) or results that are not statistically significantly different from the best (indicated with †), O-LEAP tends to achieve such results most frequently than any other method, across both binary and multiclass settings, as well as across all four classifier-training methods.

Concerning the baselines, our results clearly demonstrate the superiority of DoC with respect to the other baselines, particularly in the multiclass setting, where DoC almost always achieves the best result among the baselines.

The superiority of $\text{LEAP}_{\text{KDEy}}$ over LEAP_{ACC} supports the idea that replacing a simple quantification method like ACC with a more sophisticated one, such as KDEy, is beneficial, i.e., shows that improved class prior predictions ultimately lead to improved classifier

Table 1 Values of AE obtained in our binary experiments for different CAP methods for the LR classifier

	Baselines							*Our Methods*			
	Naïve	ATC	DoC	DS	L-CBPE	NN	Q-COT	LEAP_{ACC}	$\text{LEAP}_{\text{KDEy}}$	S-$\text{LEAP}_{\text{KDEy}}$	O-$\text{LEAP}_{\text{KDEy}}$
pageblocks.5	.341 ± .208	.129 ± .090	.084 ± .062	.156 ± .097	.137 ± .115	.150 ± .100	.048 ± .032	.080 ± .065	.042 ± .038	.035† ± .029	**.034 ± .027**
wine-q-white	.120 ± .083	.096 ± .072	.052 ± .038	.104 ± .065	.060 ± .050	.098 ± .062	.051 ± .037	.058 ± .051	.044† ± .034	**.043 ± .035**	.043† ± .035
spambase	.024 ± .018	.025 ± .019	.023† ± .019	.025 ± .018	.065 ± .029	.025 ± .018	.030 ± .020	.022 ± .017	.031 ± .024	.022† ± .017	**.022 ± .016**
ctg.1	.044 ± .032	.032 ± .025	**.029 ± .021**	.061 ± .038	.061 ± .044	.056 ± .044	.042 ± .033	.049 ± .035	.031† ± .022	.048 ± .032	.047 ± .031
ctg.2	.100 ± .076	.076 ± .058	.079 ± .055	.106 ± .062	.140 ± .080	.085 ± .076	**.059 ± .050**	.123 ± .075	.131 ± .084	.079 ± .059	.079 ± .056
ctg.3	.109 ± .076	.034 ± .029	.043 ± .033	.065 ± .041	.088 ± .072	.065 ± .044	.028† ± .024	.033 ± .027	.042 ± .035	**.028 ± .023**	.028 ± .023
wine-q-red	.038† ± .027	.069 ± .049	.050 ± .035	.040 ± .029	.039† ± .031	.040 ± .029	.062 ± .043	**.036 ± .027**	.042 ± .032	.039† ± .028	.038† ± .027
semeion	.127 ± .088	.095 ± .064	**.025 ± .022**	.072 ± .063	.137 ± .085	.071 ± .066	.028 ± .021	.057 ± .047	.040 ± .030	.051 ± .040	.039 ± .030
yeast	.134 ± .093	.092 ± .069	.062 ± .047	.129 ± .080	.063 ± .051	.126 ± .074	**.056 ± .044**	.074 ± .061	.063† ± .053	.061† ± .045	.059† ± .046
cmc.1	.074 ± .048	.082 ± .057	.086 ± .057	.089 ± .060	.060† ± .046	.076 ± .055	.065 ± .047	.074 ± .055	.075 ± .059	**.055 ± .042**	.061 ± .044
cmc.2	.268 ± .196	.177 ± .133	.129 ± .080	.198 ± .129	.164 ± .111	.133 ± .092	**.087 ± .059**	.168 ± .128	.118 ± .088	.121 ± .084	.116 ± .081
cmc.3	.184 ± .126	.165 ± .122	.104 ± .077	.190 ± .125	.151 ± .119	.093 ± .069	.111 ± .069	.172 ± .139	.078† ± .064	**.075 ± .059**	.076† ± .063
german	.127 ± .088	.103 ± .073	.063† ± .044	.105 ± .067	.088 ± .070	.111 ± .075	.073 ± .052	.088 ± .072	.074 ± .058	**.059 ± .046**	.065 ± .053
tictactoe	.017 ± .013	.023 ± .022	**.012 ± .010**	.017 ± .012	.020 ± .017	.018 ± .013	.153 ± .030	.013 ± .011	.016 ± .013	.013 ± .011	.013 ± .011
mammographic	.060 ± .039	.070 ± .042	.087 ± .042	.053 ± .038	.100 ± .048	.055 ± .038	**.041 ± .031**	.060 ± .040	.047 ± .034	.060 ± .040	.060 ± .039
transfusion	.277 ± .201	.203 ± .148	.096† ± .070	.201 ± .128	.187 ± .140	.252 ± .166	.115 ± .077	.165 ± .140	.113 ± .086	.112 ± .085	**.093 ± .074**
breast-cancer	.025 ± .018	.020 ± .016	.014 ± .010	.031 ± .018	.044 ± .029	.031 ± .019	**.012 ± .009**	.022 ± .015	.012† ± .011	.022 ± .015	.021 ± .015
balance.1	.022† ± .014	.033 ± .025	.027 ± .015	.022 ± .013	.031 ± .019	**.022 ± .013**	.101 ± .018	.025 ± .018	.033 ± .028	.025 ± .018	.025 ± .017
balance.3	.028 ± .021	.049 ± .034	.032 ± .026	.027 ± .022	.049 ± .034	.028 ± .022	.059 ± .030	.019 ± .017	.022 ± .018	.019 ± .016	**.019 ± .015**
wdbc	.024 ± .020	.030 ± .022	.030 ± .020	.022 ± .018	.059 ± .027	.021 ± .017	**.018 ± .014**	.025 ± .018	.039 ± .032	.026 ± .019	.025 ± .018
ionosphere	.056† ± .042	.071 ± .047	.099 ± .071	.056† ± .036	.101 ± .063	**.053 ± .036**	.053† ± .042	.095 ± .066	.106 ± .069	.084 ± .063	.080 ± .057
haberman	.223 ± .157	.145 ± .101	.109 ± .086	.167 ± .100	.186 ± .137	.200 ± .114	**.093 ± .061**	.189 ± .141	.098† ± .082	.103 ± .085	.112 ± .087
spectf	.133 ± .097	.096 ± .073	**.050 ± .042**	.081 ± .056	.117 ± .091	.085 ± .058	.074 ± .050	.066 ± .058	.052† ± .043	.076 ± .058	.068 ± .052
sonar	.075 ± .049	.121 ± .080	.165 ± .076	.073 ± .049	.183 ± .099	.077 ± .051	.072 ± .052	**.049 ± .036**	.061 ± .047	.069 ± .045	.068 ± .044
wine.1	.032 ± .000	.028 ± .023	**.000 ± .000**	.046 ± .005	.004 ± .004	.045 ± .002	.080 ± .033	.052 ± .029	.124 ± .081	.048 ± .024	.048 ± .026
wine.2	.019† ± .011	.039 ± .025	.020 ± .011	.023 ± .013	**.017 ± .016**	.024 ± .013	.069 ± .017	.022 ± .017	.026 ± .021	.021 ± .016	.023 ± .016
wine.3	.018 ± .015	.026 ± .022	.026 ± .022	.020 ± .018	.023 ± .020	.019 ± .018	.037 ± .016	.017 ± .016	.016 ± .015	.017 ± .016	**.016 ± .015**
iris.2	.108 ± .070	.072 ± .047	.212 ± .152	.166 ± .111	.154 ± .096	.078 ± .062	**.036 ± .027**	.353 ± .119	.208 ± .114	.100 ± .078	.115 ± .075
iris.3	.032 ± .021	.050 ± .031	.067 ± .025	.022 ± .017	.046 ± .031	**.021 ± .017**	.124 ± .053	.031 ± .021	.035 ± .030	.032 ± .022	.030 ± .021
Average	.098 ± .124	.078 ± .082	.065 ± .072	.082 ± .087	.089 ± .091	.074 ± .083	.065 ± .053	.077 ± .099	.063 ± .069	.053† ± .055	**.053 ± .054**

Boldface indicates the best method for a given dataset. Superscript † denotes the methods (if any) whose scores are not statistically significantly different from the best one according to a Wilcoxon signed-rank test at 0.01 confidence level. Cells are colour-coded so as to facilitate readability and allow for quick comparisons across results, with green indicating best and red indicating worst

[8] https://github.com/HLT-ISTI/QuaPy

[9] https://github.com/lorenzovolpi/LEAP.

Table 2 Values of AE obtained in our multiclass experiments for different CAP methods for the LR classifier

	Baselines							*Our Methods*			
	Naïve	ATC	DoC	DS	L-CBPE	NN	Q-COT	LEAP_{ACC}	$\text{LEAP}_{\text{KDEy}}$	$\text{S-LEAP}_{\text{KDEy}}$	$\text{O-LEAP}_{\text{KDEy}}$
poker-hand	.375 ± .102	.368 ± .114	.082 ± .065	.082 ± .065	.368 ± .102	.081 ± .067	**.074 ± .068**	.100 ± .079	.103 ± .091	.107 ± .097	.094 ± .079
connect-4	.254 ± .185	.164 ± .132	.093 ± .070	.164 ± .108	.185 ± .145	.112 ± .090	.078 ± .062	.242 ± .188	.070† ± .056	.078 ± .061	**.063 ± .049**
shuttle	.250 ± .188	.230 ± .187	.151 ± .101	.153 ± .101	.209 ± .177	.154 ± .106	.037 ± .032	.128 ± .104	.037 ± .030	**.034 ± .029**	.041 ± .038
chess	.052 ± .040	.085 ± .061	.045 ± .033	.051 ± .039	.052 ± .040	.047 ± .035	.072 ± .045	.056 ± .043	.043 ± .033	.044 ± .032	**.040 ± .030**
letter	.037 ± .028	.037 ± .029	**.030 ± .023**	.036 ± .028	.149 ± .084	.036 ± .027	.130 ± .037	.038 ± .028	.062 ± .039	.034 ± .025	.035 ± .026
dry-bean	.024 ± .017	.022 ± .018	**.019 ± .014**	.021 ± .016	.029 ± .020	.022 ± .016	.034 ± .020	.021 ± .015	.025 ± .019	.020 ± .015	.020† ± .015
nursery	.082 ± .070	.042 ± .037	.033 ± .027	.056 ± .050	.057 ± .047	.063 ± .050	.031 ± .021	.038 ± .035	.036 ± .025	.029 ± .026	**.022 ± .017**
hand-digits	.021 ± .016	.019 ± .015	**.017 ± .013**	.022 ± .016	.039 ± .024	.021 ± .016	.054 ± .021	.020 ± .015	.032 ± .019	.019 ± .014	.018 ± .013
isolet	.018 ± .014	.018 ± .015	**.015 ± .011**	.018 ± .013	.193 ± .082	.018 ± .014	.080 ± .024	.019 ± .014	.053 ± .025	.018 ± .013	.026 ± .016
wine-quality	.250 ± .121	.220 ± .114	.100 ± .068	.100 ± .067	.284 ± .160	.105 ± .069	.121 ± .072	.256 ± .142	.126 ± .076	**.083 ± .063**	.117 ± .075
satellite	.070 ± .062	.037 ± .030	.036 ± .027	.060 ± .047	.055 ± .045	.061 ± .048	.030† ± .024	.044 ± .036	.043 ± .032	**.030 ± .023**	.030† ± .023
digits	.016 ± .013	.016† ± .014	.014† ± .012	.017 ± .014	.017 ± .014	.016 ± .013	.037 ± .017	.015 ± .012	.027 ± .016	.015 ± .012	**.014 ± .010**
page-block	.260 ± .135	.178 ± .117	.101 ± .087	.128 ± .106	.166 ± .117	.126 ± .097	**.057 ± .045**	.141 ± .094	.080 ± .053	.104 ± .068	.081 ± .055
waveform-v1	.031 ± .023	.038 ± .029	.030 ± .023	.031 ± .023	.034 ± .026	.031 ± .024	.034 ± .026	**.028 ± .021**	.038 ± .028	.028 ± .021	.028 ± .021
academic-success	.139 ± .120	.078 ± .062	.082 ± .056	.112 ± .092	.108 ± .084	.113 ± .090	.069 ± .054	.077 ± .065	.058 ± .045	**.051 ± .041**	.052† ± .044
abalone	.057 ± .042	.068 ± .054	.045 ± .035	.054 ± .042	.071 ± .049	.052 ± .040	.042 ± .032	.075 ± .053	.052 ± .039	**.039 ± .030**	.040† ± .032
molecular	.027 ± .022	.025 ± .021	**.022 ± .018**	.025 ± .020	.035 ± .024	.025 ± .019	.026 ± .020	.024 ± .019	.026 ± .021	.025 ± .019	.024 ± .018
image-seg	.030 ± .023	.023 ± .019	.020† ± .016	.029 ± .022	.062 ± .033	.030 ± .023	.065 ± .024	.019 ± .017	.031 ± .023	**.018 ± .015**	.019† ± .015
obesity	.052 ± .037	.049 ± .039	**.033 ± .025**	.051 ± .037	.124 ± .056	.048 ± .035	.134 ± .040	.038 ± .030	.063 ± .042	.034† ± .026	.035† ± .026
cmc	.079 ± .053	.065 ± .048	.047† ± .034	.070 ± .049	.056 ± .040	.055 ± .039	.057 ± .041	.057 ± .048	.053 ± .042	.050 ± .036	**.045 ± .034**
hcv	.044† ± .032	.079 ± .054	**.042 ± .031**	.046 ± .034	.161 ± .052	.046 ± .034	.103 ± .053	.061 ± .044	.058 ± .042	.048 ± .035	.050 ± .036
phishing	.203 ± .179	.133 ± .119	.095 ± .075	.152 ± .102	.158 ± .143	.176 ± .112	.081 ± .061	.144 ± .116	.051 ± .039	.046† ± .037	**.044 ± .036**
yeast	.074 ± .052	.086 ± .062	.080 ± .055	.081 ± .058	.088 ± .062	.082 ± .057	.062† ± .044	.082 ± .058	.069 ± .053	**.057 ± .044**	.062 ± .047
mhr	.099 ± .061	.071 ± .052	.083 ± .053	.097 ± .062	.092 ± .059	.099 ± .062	.071 ± .047	.078 ± .061	**.055 ± .045**	.058† ± .043	.060 ± .045
Average	.106 ± .133	.090 ± .113	.055 ± .060	.069 ± .074	.116 ± .120	.067 ± .073	.066 ± .052	.075 ± .096	.054 ± .049	.045† ± .047	**.044 ± .045**

Boldface, superscript †, and colour coding are used in the same way as in Table 1

accuracy prediction. While the adoption of KDEy brings about superior performance in our experimental evaluation, KDEy is a *replaceable* component of LEAP; should more accurate quantification methods become available, LEAP could directly benefit from their adoption (in place of KDEy) as the quantification module.

While S-LEAP and $\text{LEAP}_{\text{KDEy}}$ perform comparably, S-LEAP tends to achieve slightly better results on average. This is particularly noteworthy because, as shown in Sect. 4.2, S-LEAP reduces the computational cost to that needed for estimating the class priors (typically equivalent to solving an additional system of n equations), while $\text{LEAP}_{\text{KDEy}}$ (like LEAP_{ACC}) requires class prior estimation (thus incurring the same cost as S-LEAP) *and* solving a system of n^2 equations. In other words, S-LEAP is $\mathcal{O}(n)$, whereas other LEAP methods are $\mathcal{O}(n^2)$. In the binary case, both systems are computationally efficient at inference time, and the differences in execution times are almost negligible. However, in the multiclass scenario, S-LEAP holds a clear advantage over the other LEAP methods, which do not scale well when the number of classes increases. In our experiments, the largest number of classes is reached by the chess dataset, with $n = 15$, thus requiring the solution of a system of $n^2 = 225$ independent equations (LEAP_{ACC}, $\text{LEAP}_{\text{KDEy}}$), or $(n+1)^2$ equations in the overconstrained case (O-LEAP). Figure 5 (in Sect. 5.12) displays the run times we clock in our experiments for the different CAP methods.

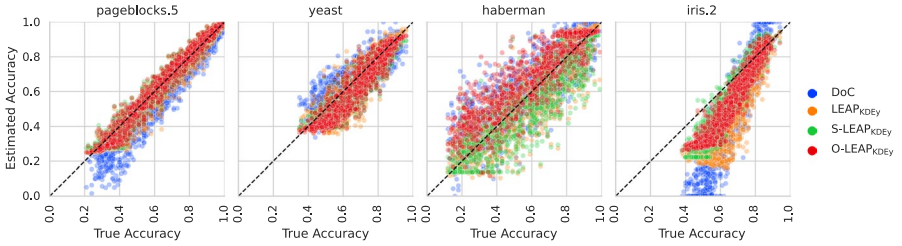
Nevertheless, our experiments also show that O-LEAP clearly outperforms S-LEAP, suggesting that, when the computational cost is affordable, solving the full set of $(n+1)^2$ equations, as O-LEAP does, is beneficial and leads to a better solution.

5.7 On the robustness of LEAP methods

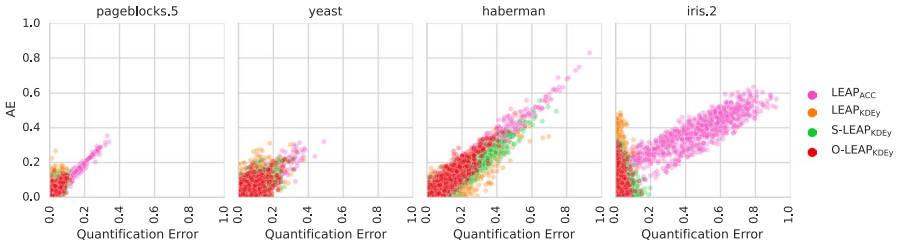
As all CAP methods, the methods from the LEAP family are potentially exposed to challenging situations. Two of them are (a) the situation in which the classifier h to be tested has very low accuracy, and (b) the situation in which the quantifier used in the CAP method has very low accuracy. In this section we report the results of experiments aimed at testing how our CAP methods cope with these two situations.

Figure 1a shows a selection of prototypical diagonal plots for the LR classifier (the full set of plots for all datasets and all classifiers is online).[10] These plots display the estimated accuracy of the classifier as a function of its accuracy, with each dot representing a set of test datapoints; a perfect CAP method would give rise only to values lying on the diagonal from (0,0) to (1,1). For the sake of clarity, we have omitted methods Naïve (which returns dots all lying on a horizontal line, and is thus uninteresting), the weakest baselines (ATC, DS, L-CBPE, NN, Q-COT), and our weakest method (LEAP_{ACC}). Note that O-LEAP tends to display the smallest deviation from the main diagonal, and that LEAP methods tend to remain close to the diagonal even for low values of the true classifier accuracy.

Figure 1b shows diagonal plots relating CAP error to the quantification error (i.e., to the absolute error between the true prevalence values of the test sample and the values predicted by the quantifier) on the same selection of datasets of Fig. 1a. The plots compare the results for LEAP_{ACC}, $\text{LEAP}_{\text{KDEy}}$, $\text{S-LEAP}_{\text{KDEy}}$, and $\text{O-LEAP}_{\text{KDEy}}$, i.e., the methods impacted by the quantification error. For $\text{LEAP}_{\text{KDEy}}$, $\text{S-LEAP}_{\text{KDEy}}$, and $\text{O-LEAP}_{\text{KDEy}}$ it is clear that the impact of the quantification error is limited in most cases, with low correlation (pageblocks.5 and yeast) or no correlation (iris.2) between CAP error and the quantification error. Some cases (haberman) present a higher level of correlation between the two errors, but in all cases the amount of test samples for which quantification error is high is limited.



(a) Diagonal plots relating classification accuracy (x-axis) with CAP accuracy (y-axis) a classifier for different CAP methods. For better readability we focus on the best baseline (DoC) and on the three best methods in the LEAP family ($\text{LEAP}_{\text{KDEy}}$, $\text{S-LEAP}_{\text{KDEy}}$, $\text{O-LEAP}_{\text{KDEy}}$).



(b) Diagonal plots relating quantification error (x-axis) with CAP error (y-axis) for CAP methods in which a quantifier is involved.

Fig. 1 Diagonal plots for evaluating the robustness of CAP methods to low classifier accuracy (top) and to low quantifier accuracy (bottom). All results are relative to the LR classifier and vanilla accuracy

[10] https://github.com/lorenzovolpi/LEAP/tree/main/output/plots

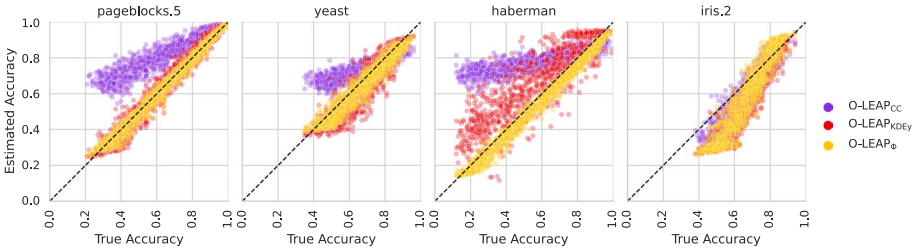


Fig. 2 Diagonal plots showing the same experimental setup as in Fig. 1a, but comparing O-LEAP$_{\mathrm{KDEy}}$ with O-LEAP$_{\mathrm{CC}}$ (lower-bound) and O-LEAP$_{\Phi}$ (upper-bound)

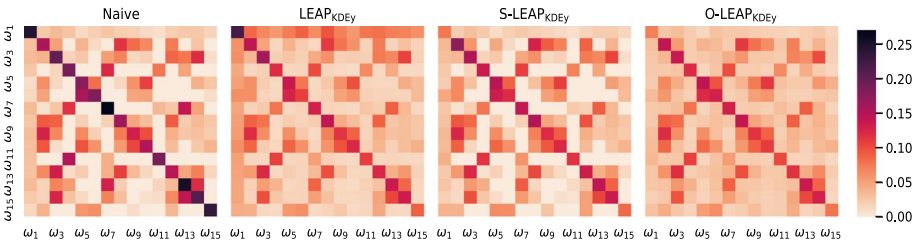


Fig. 3 Heat maps representing the prediction errors $e_{ij} = |\hat{c}^U_{ij} - c^U_{ij}|$ for the contingency table of dataset chess (with $n = 15$ classes) for classifier MLP. The errors e_{ij} have been transformed using $\sqrt{e_{ij}}$ so as to enhance contrast and improve visualization

LEAP$_{\mathrm{ACC}}$ is instead affected by low quantification performance to a higher degree, but this aspect is more related to the *choice* of the quantifier, an issue we discuss in the next section.

5.8 On the role of the quantifier in LEAP's performance

As shown in our experiments, the modular design of LEAP allows for the replacement of its quantification component, and doing so (e.g., replacing ACC with KDEy) can improve CAP accuracy. To better understand how CAP accuracy depends on the quantifier, we compare O-LEAP$_{\mathrm{KDEy}}$ with O-LEAP$_{\mathrm{CC}}$ (i.e., the version of O-LEAP that, as the quantification method, employs the simplistic "classify and count") and O-LEAP$_{\Phi}$ (the version of O-LEAP endowed with an oracle that always predicts the true prevalence values). As shown in Fig. 2, O-LEAP$_{\mathrm{CC}}$ performs markedly worse than O-LEAP, thus confirming that KDEy plays a key role in the effectiveness of O-LEAP$_{\mathrm{KDEy}}$. While O-LEAP$_{\Phi}$ is an ideal upper bound for O-LEAP, the results show that the gap between O-LEAP$_{\Phi}$ and O-LEAP$_{\mathrm{KDEy}}$ is relatively narrow, which indicates that O-LEAP$_{\mathrm{KDEy}}$ is a good method. Aside from this, given that the experiments are designed to generate PPS, any deviation of O-LEAP$_{\Phi}$ from the main diagonal must be caused by random error sampling; indeed, O-LEAP$_{\Phi}$ shows the greatest deviation from the diagonal in the smallest dataset, i.e., iris.2.

5.9 Error patterns in the contingency table

Figure 3 displays the prediction errors $e_{ij} = |\hat{c}^U_{ij} - c^U_{ij}|$ for the entries of the test contingency table U for dataset chess and for different methods. Naïve corresponds to the error

generated by assuming that the training set and the test set are IID, i.e., by disregarding the presence of PPS. This method tends to make most of the errors along the diagonal $(e_{11}, \ldots, e_{nn})$, which is critical for computing Acc (as it corresponds to the numerator of the measure), thus resulting in large CAP error. Conversely, $\text{LEAP}_{\text{KDEy}}$ tends to allocate more errors across the first row $(e_{11}, \ldots, e_{1n})$ and the first column $(e_{11}, \ldots, e_{n1})$; this can be explained by the fact that this method drops the constraints associated with class ω_1 from all equation groups (Sect. 4.2), thus misrepresenting this class relative to the others. S-LEAP does not suffer from this issue but exhibits higher variance in errors. Finally, O-LEAP seems to produce a more homogeneous distribution of errors and, in particular, tends to yield fewer errors along the diagonal when compared to the other methods.

5.10 Uncertainty estimation in LEAP predictions

As a point estimator, LEAP is inevitably affected by error. In this section we study the per-case associated uncertainty.

Unfortunately, deriving analytical error bounds for our method is non-trivial, given that LEAP relies on optimization routines for cases in which the system matrix is non-invertible, and given that the method's uncertainty inherently depends on the uncertainty of the quantifier, which is an independent module.

For this reason, and in order to facilitate a comparison across different CAP methods, we treat these latter as black-box point estimators and study their uncertainty by deriving empirical confidence intervals (CIs) via standard bootstrapping. For this experiment we concentrate on the binary case, and compare O-LEAP (our best method) against DoC (the best-performing baseline); we increase the sample size to 500 and apply 500 repetitions of bootstrap resampling to the 1000 test samples.

We evaluate the quality of the CIs in terms of their *amplitude* ($\mathcal{A}$), defined as the difference between the upper and the lower bound of the interval, and their *coverage* ($\mathcal{C}$), defined as how frequently the true accuracy value falls in the CI. A good CI has thus small amplitude and high coverage.

Table 3 reports the results we have obtained for the LR classifier using vanilla accuracy in the binary setting. Several insights emerge from this table: (i) DoC tends to achieve better coverage, while $\text{O-LEAP}_{\text{KDEy}}$ performs better in terms of amplitude; (ii) both methods (especially $\text{O-LEAP}_{\text{KDEy}}$) yield rather low levels of coverage, despite the generally small AE. A good method should perform well on both aspects (despite the fact that performing well in terms of amplitude *inevitably* makes it difficult to perform well also in terms of coverage), and if a method performs badly in terms of one, there is little to rejoice in the fact that it performs well in terms of the other. In summary, these results suggest that there is substantial room for improvement regarding uncertainty estimation in current CAP approaches. This is something we plan to address in future work.

5.11 Empirical sampling error

When the bags that we extract via the artificial prevalence protocol (see Sect. 5.2) are very large, we approach "pure" prior probability shift, i.e., the scenario in which Eqs. 3 hold. However, the smaller these bags are, the more our scenario is affected, aside from PPS, by *sampling error*. This error naturally arises from the fact that we never have access to the true

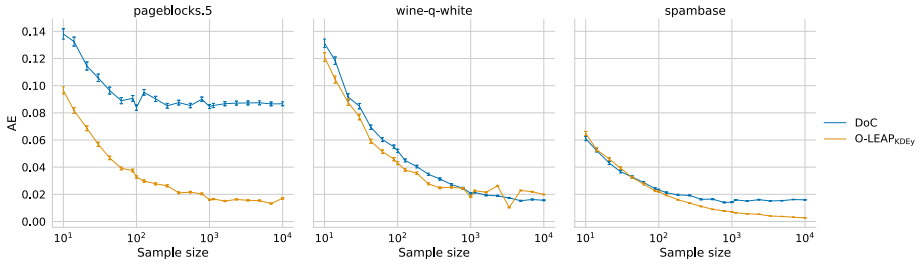
Table 3 Empirical confidence intervals obtained for DoC and O-LEAP$_{\text{KDEy}}$ on binary datasets for LR using vanilla accuracy

	DoC			O-LEAP$_{\text{KDEy}}$		
	AE	$\mathcal{A}$	$\mathcal{C}$	AE	$\mathcal{A}$	$\mathcal{C}$
pageblocks.5	.088	.102	.139	**.019**	**.064**	**.838**
wine-q-white	.025	.095	**.858**	**.021**	**.053**	.696
spambase	.018	.017	**.241**	**.010**	**.004**	.117
ctg.1	**.023**	.035	**.335**	.043	**.018**	.051
ctg.2	**.072**	.045	**.148**	.076	**.043**	.121
ctg.3	.051	.056	.211	**.013**	**.024**	**.577**
wine-q-red	.035	.037	**.248**	**.020**	**.011**	.192†
semeion	**.011**	.051	**.923**	.033	**.023**	.079
yeast	**.033**	.123	**.871**	.038	**.077**	.590
cmc.1	.113	.182	**.434**	**.045**	**.057**	.305
cmc.2	.140	**.140**	.058	**.111**	.164	**.390**
cmc.3	.102	.168	.382	**.054**	**.136**	**.678**
german	**.044**	.088	**.530**	.049†	**.071**	.403
tictactoe	**.006**	.008	**.458**	.007	**.005**	.242
mammographic	.083	.003	**.000**	**.058**	**.001**	**.000**
transfusion	.097	.184†	.522	**.049**	**.178**	**.862**
breast-cancer	**.007**	.021	**.722**	.015	**.006**	.073
balance.1	.025	.026	**.119**	**.022**	**.006**	.062
balance.3	.030	.007	.075†	**.013**	**.005**	**.108**
wdbc	.046	.072	**.432**	**.018**	**.008**	.132
ionosphere	.168	**.001**	.002	**.074**	.040	**.167**
haberman	**.063**	.247	**.890**	.170	**.166**	.212
spectf	**.039**	**.050**	**.432**	.040†	.053	.399†
sonar	.205	.031	.000	**.063**	**.006**	**.015**
wine.1	**.000**	**.000**	**1.000**	.048	.008	.000
wine.2	**.019**	.015	**.051**	.026	**.011**	.013
wine.3	.026	**.000**	.036	**.014**	.005	**.080**
iris.2	.214	.148	**.277**	**.112**	**.062**	.200
iris.3	.067	**.000**	.000	**.027**	.011	**.099**

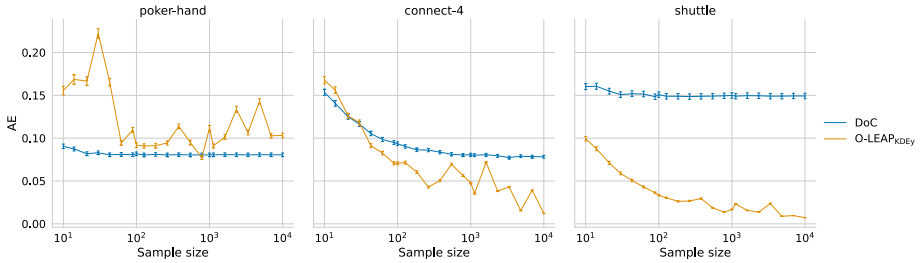
Green and red indicate the better and the worse results, respectively, independently for each evaluation measure

underlying distributions, but only to some empirical realisations thereof. In this regard, one might expect that larger empirical samples display smaller sampling error, for the simple fact that the larger the sample, the better it represents the underlying distribution. For this reason, we study the stability of the methods' performance as a function of the sampling size, assuming the latter to be a proxy of sampling error.

Figure 4a, b display CAP error as a function of sample size for the three largest binary datasets (top) and for the three largest multiclass datasets (bottom), for the LR classifier and using vanilla accuracy as the target classifier accuracy measure. The plots show the curves for the best-performing baseline (DoC) and for the best-performing method in the LEAP family (O-LEAP$_{\text{KDEy}}$), along with standard error bands. These results are from a series of experiments in which we vary the sample size from 10 to 10,000, with 1000 test evaluations per configuration. While DoC appears more stable to variations in sample size, it also tends to exhibit larger predictive error. Notable exceptions include cases where DoC performs comparably (e.g., the `wine-q-white` binary dataset) or even better than O-LEAP$_{\text{KDEy}}$ (e.g., the `poker-hand` multiclass dataset) in the long term. However, DoC often plateaus after a certain sample size (see e.g., `pageblock.5` from sample size 100 onwards, or `poker-hand` and `shuttle` across almost the entire spectrum), while O-LEAP$_{\text{KDEy}}$ steadily improves as more training data become available. Nonetheless, the improvements of O-LEAP$_{\text{KDEy}}$ tend to be less smooth, with larger performance fluctuations visible in the learning curves.



(a) Results on the three largest binary datasets.



(b) Results on the three largest multiclass datasets.

Fig. 4 CAP error as a function of sample size for the LR classifier, using vanilla accuracy as the target classifier accuracy measure

5.12 Efficiency of LEAP

The LEAP methods aim at predicting the n^2 entries (the unknowns) of the contingency table, where n is the number of classes. To this end, a system $\mathrm{A}\mathbf{x} = \mathbf{b}$ is constructed, with $\mathrm{A} \in \mathbb{R}^{n^2 \times n^2}$ ($\mathrm{LEAP}_{\mathrm{ACC}}$ and $\mathrm{LEAP}_{\mathrm{KDEy}}$). This setup has clear implications in terms of efficiency and scalability for large values of n. While the S-LEAP variant alleviates this issue, our best-performing variant, O-LEAP, instead exacerbates it, as it constructs a matrix $\mathrm{A} \in \mathbb{R}^{(n+1)^2 \times n^2}$.

Fortunately, although the space complexity is $\mathcal{O}(n^4)$, matrix A turns out to be fairly sparse. In terms of density,[11] note that the second group, third group, and fourth group of equations have density n/n^2. The first group (Eq. 8) is an exception to this, since it is fully dense; however, it contributes only a single equation to the system. While the method still has polynomial space complexity, using an efficient sparse matrix representation reduces complexity by one order of magnitude, down to $\mathcal{O}(n^3)$.

In this section we compare a naïve implementation of LEAP (which is actually the one we used in Volpi et al., 2024) with an implementation based on sparse matrix representation. Aside from reducing the space complexity, the latter implementation allows us to take advantage of modern optimizers, which drastically speed-up the computation. For the naïve implementation, in Volpi et al. (2024) we used the *sequential least squares programming* (SLSQP) routine of the `scipy.optimize` package, while for the implementation

[11] The density of a matrix A is computed as the fraction of non-zero elements it contains, i.e., as $\mathrm{density}(\mathrm{A}) = \frac{|\{a_{i,j} \in \mathrm{A} : a_{i,j} \neq 0\}|}{\#\mathrm{A}}$

based on sparse matrix representation we adopt the *Operator Splitting Quadratic Program* (OSQP) from the `cvxpy` package (Diamond & Boyd, 2016).

Figure 5 display clocked times (at inference time) for different methods.[12] The plot displays averaged times across all multiclass datasets as a function of the number of classes; the comparison includes the two best-performing baselines (ATC and DoC), the variants of our methods (denoted LEAP$^{\text{dense}}$ and O-LEAP$^{\text{dense}}$) based on the naïve implementation, and the variants of the same methods based on sparse matrix representation (denoted LEAP$^{\text{sparse}}$ and O-LEAP$^{\text{sparse}}$); note that for S-LEAP we do not include two different implementations since this method does not need to solve the system of equations (see Sect. 4.2). For the LEAP variants, we omit the name of the surrogate quantifier, which is KDEy in all cases.

This experiment reveals that the sparse implementation leads to a substantial performance gain, making LEAP and O-LEAP comparable, in terms of running time, to S-LEAP. As S-LEAP avoids solving the system of equations altogether, this result indicates that the sparse implementation virtually removes the computational overhead of solving such a system. While ATC and DoC remain the fastest methods overall (requiring less than a millisecond in all cases), the methods based on sparse matrix representation require no more than 0.1 s in the worst case, and appear to scale well with the number of classes.

Note that we show results only for the experiments regarding vanilla accuracy as the accuracy measure, and for the LR classifier. This is due to the fact that the classification model and the accuracy measure do not impact significantly on the execution time of the CAP methods (this is true both for the baselines and for our methods), and for any other combination of classifier and accuracy measure the results would be similar.

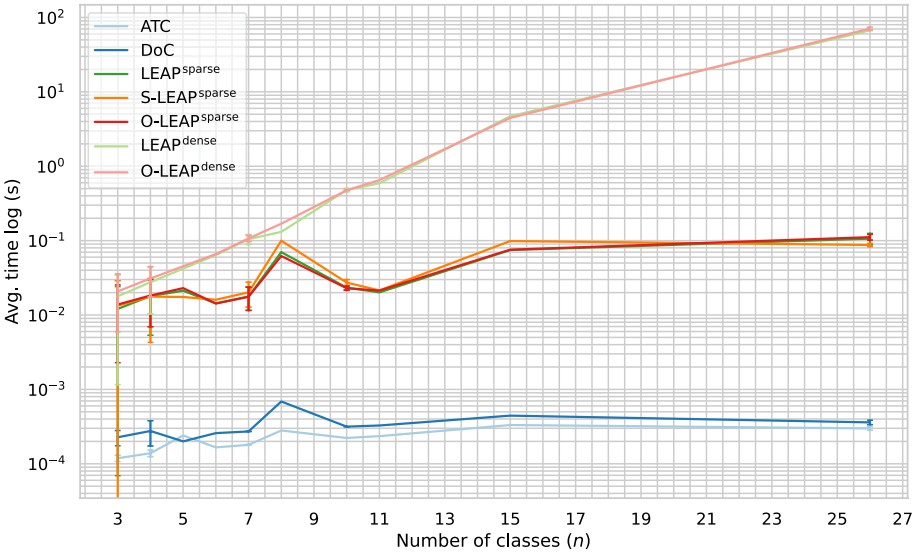


Fig. 5 Average execution times (at inference time) for the multiclass experiments for the LR classifier, expressed as a function of the number of classes. Note the log scale along the y-axis

[12] We clocked times on a server computer equipped with two Intel(R) Xeon(R) Platinum 8480C processors and 2TB of RAM, running Ubuntu 22 LTS; we run every process in single-thread mode.

5.13 Impact of PPS on accuracy prediction error

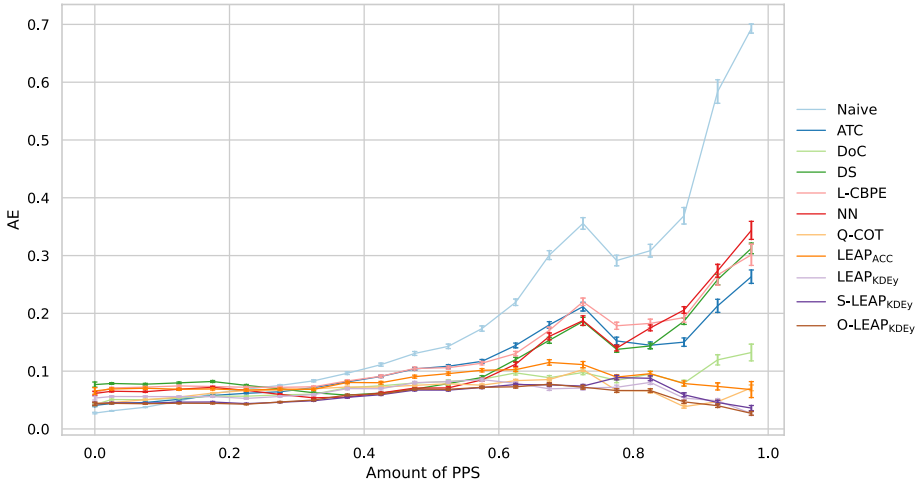
The experiments discussed so far are all based on the simulation of different levels of Prior Probability Shift (PPS). Since we have always reported average error results across all the generated test samples, an open question remains: how well do CAP methods perform as a function of the *amount* of shift? This is a relevant point, as the APP generates a wide range of amounts of shift, i.e., some samples exhibit a substantial shift while others show little or none.

Figure 6 reports average results from all binary (top) and multiclass (bottom) experiments using the LR classifier, plotted as a function of the amount of PPS. We measure the amount of PPS simply as the L1 distance between the training prevalence vector and the test sample prevalence vector. Note that the number of experiments contributing to each point along the x axis varies, since the maximum attainable L1 distance depends on the dataset. For instance, the maximum L1 distance for a binary dataset with a training prevalence of 0.3 is 0.7, meaning that this dataset can only contribute to the [0, 0.7] region of the plot. As a result, the instabilities observed on the rightmost part of the plots are due to a smaller number of contributing experiments.

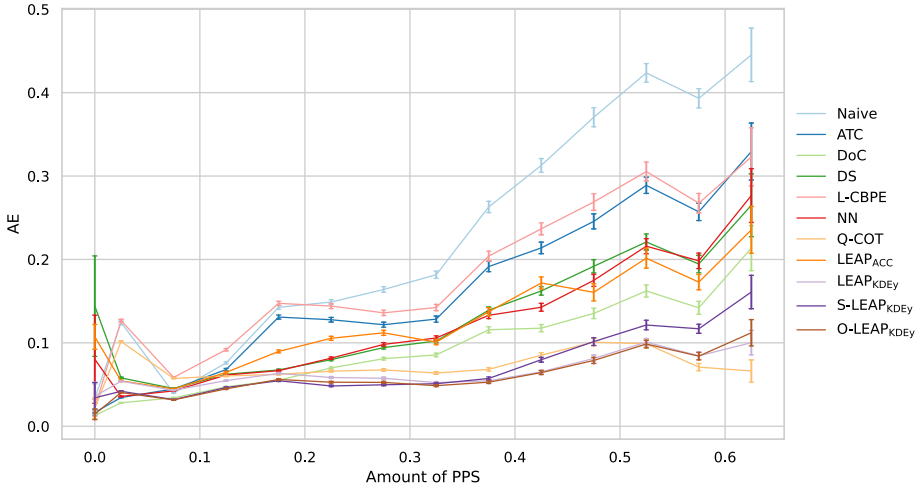
Several insights can be obtained from these plots. First, all methods perform well in the binary setting when the amount of PPS is very low, i.e., under conditions close to the IID assumption. In the multiclass case, some instabilities are observed near the IID regime for methods such as DS, LEAP_{ACC}, and (to a smaller extent) NN, but overall, all methods still perform reasonably well. Interestingly, when the amount of PPS is minimal, the Naïve approach often outperforms the other methods; this is particularly evident in the binary plot. Concerning the methods in the LEAP family, this experiment shows that, despite the fact that these methods strongly rely on the PPS assumptions, they do not fail under conditions close to the IID case.

Second, the binary plot shows a clear separation between two groups of methods, one (comprising ATC, NN, L-CBPE, and DS) for which performance degrades as the amount of shift increases, and another (including DoC, Q-COT, and the LEAP methods) which remain fairly stable across the entire range of shift values. In the multiclass scenario, this distinction is less pronounced, as all methods (LEAP methods included) tend to suffer from performance degradation under high levels of PPS. Q-COT seems to be the best-performing method for the two highest levels of shift in multiclass problems; however, measured values for these levels of shift are less reliable, due to the instability discussed above, and S-$\text{LEAP}_{\text{KDEy}}$ and O-$\text{LEAP}_{\text{KDEy}}$ seem to consistently dominate the rest of the plot.

Overall, LEAP methods demonstrate strong performance across the full spectrum of PPS in both binary and multiclass settings. The notable exception is LEAP_{ACC}, which, while effective in the binary case, shows a marked performance drop in the multiclass case. The fact that its counterpart $\text{LEAP}_{\text{KDEy}}$ performs well in both settings suggests that the limitations of LEAP_{ACC} stem from the fact that ACC is a relatively weak quantifier for multiclass problems. Overall, the best-performing CAP method across both settings and all levels of PPS appears to be O-$\text{LEAP}_{\text{KDEy}}$.



(a) Results on the binary datasets.



(b) Results on the multiclass datasets.

Fig. 6 CAP error as a function of the amount of PPS (measured by the L1 distance between the training and the test class prevalence values) for the LR classifier, averaged across all datasets

5.14 Exploring additional evaluation measures

While vanilla accuracy is the most popular evaluation measure for evaluating the accuracy of a classifier, several other measures exist. Among the most important ones is F_1, the harmonic mean of precision and recall, which is of special interest in imbalanced scenarios. In this section we turn to analysing the extent to which our method accurately predicts F_1.

In Sect. 4.3, we have shown the closed-form solution of LEAP_{ACC} for the case in which vanilla accuracy (Acc) is our classifier accuracy measure for a binary problem. Following

the same rationale, F_1 (Eq. 29) can be expressed in compact form using γ and q (Eq. 30), and expanded as a function that uses only γ along with counts from the validation contingency table V (Eq. 31), i.e.,

$$F_1(h, U) = \frac{2 \cdot \mathrm{TP}^U}{2 \cdot \mathrm{TP}^U + \mathrm{FP}^U + \mathrm{FN}^U} \tag{29}$$

$$= \frac{2q_{\oplus} \cdot \mathrm{tpr}^V}{\gamma_{\oplus} + q_{\oplus}} \tag{30}$$

$$= \frac{2 \cdot \mathrm{TP}^V(\gamma_{\oplus} \cdot \mathrm{TN}^V - \gamma_{\ominus} \cdot \mathrm{FP}^V)}{\gamma_{\oplus} \cdot \mathrm{TN}^V(2 \cdot \mathrm{TP}^V + \mathrm{FN}^V) - \mathrm{FP}^V(\mathrm{FN}^V + \gamma_{\ominus} \cdot \mathrm{TP}^V)} \tag{31}$$

In the binary case, we compute F_1 as in Eq. 29 and assign an F_1 score of 1 when the denominator is zero (since the classifier has correctly classified all instances as true negatives); for the multiclass case we use "macro"-averaged F_1, which consists of computing *n binary* F_1 scores, one for each class, and then taking the average.

Tables 4 and 5 show the results for the experiments that use F_1 as the classifier accuracy measure and the LR classifier in the binary and multiclass settings, respectively. We report results for all CAP methods except ATC and Q-COT, since these methods are explicitly designed for estimating vanilla accuracy.

Similarly to the results for vanilla accuracy, O-LEAP$_{\mathrm{KDEy}}$ and S-LEAP$_{\mathrm{KDEy}}$ show good results in comparison to the baselines, both in the binary and in the multiclass settings. O-LEAP$_{\mathrm{KDEy}}$ achieves the highest number of best results (in boldface) and results that are not statistically significantly different from the best result (marked with †), across both the binary case and the multiclass case. Also S-LEAP$_{\mathrm{KDEy}}$ shows comparable results, while slightly inferior to O-LEAP$_{\mathrm{KDEy}}$. DoC remains the strongest baseline in this case as well.

Table 4 CAP results, in terms of AE obtained on F_1 evaluation measure, for binary classifiers trained via LR

	Baselines					*Our Methods*			
	Naïve	DoC	DS	L-CBPE	NN	LEAP$_{\mathrm{ACC}}$	LEAP$_{\mathrm{KDEy}}$	S-LEAP$_{\mathrm{KDEy}}$	O-LEAP$_{\mathrm{KDEy}}$
pageblocks.5	.086 ± .084	.215 ± .091	.217 ± .153	.325 ± .213	.223 ± .180	.086 ± .131	.081 ± .099	.079 ± .091	**.060 ± .078**
wine-q-white	.214 ± .210	.098 ± .081	.195 ± .147	.126 ± .125	.189 ± .138	.090 ± .104	.062 ± .066	**.058 ± .063**	.058† ± .064
spambase	.074 ± .116	.079 ± .105	.072 ± .123	.115 ± .132	.073 ± .126	.046† ± .084	.048 ± .062	**.039 ± .063**	.042† ± .071
ctg.1	.142 ± .190	.076 ± .097	.141 ± .147	.130 ± .165	.137 ± .144	.113 ± .145	**.051 ± .067**	.064 ± .073	.086 ± .108
ctg.2	.205 ± .075	.138 ± .076	**.089 ± .112**	.243 ± .139	.132 ± .120	.126 ± .090	.186 ± .105	.145 ± .091	.122 ± .087
ctg.3	.057 ± .081	.058 ± .082	.053 ± .095	.132 ± .103	.060 ± .101	.048 ± .081	.068 ± .092	.047 ± .075	**.045 ± .068**
wine-q-red	.147 ± .158	.117 ± .110	.148 ± .150	.135 ± .156	.146 ± .146	.080 ± .106	.071 ± .076	.064 ± .073	**.063 ± .075**
semeion	**.063 ± .077**	.071 ± .070	.091 ± .095	.183 ± .099	.095 ± .098	.074 ± .071	.074 ± .115	.074 ± .076	.067 ± .079
yeast	.115 ± .082	.101 ± .078	.127 ± .141	.217 ± .198	.143 ± .129	.084† ± .093	.089 ± .088	.082 ± .084	**.082 ± .091**
cmc.1	.130 ± .100	.149 ± .100	.145 ± .136	.200 ± .168	.174 ± .154	.075† ± .084	.087 ± .067	.076 ± .062	**.066 ± .062**
cmc.2	.072 ± .062	.115 ± .069	.316 ± .165	.372 ± .189	.370 ± .186	.061 ± .058	.075 ± .084	.080 ± .086	**.057 ± .061**
cmc.3	.091† ± .086	**.081 ± .063**	.321 ± .126	.357 ± .206	.273 ± .185	.102† ± .102	.112 ± .104	.105 ± .093	.098† ± .096
german	.215 ± .209	.144 ± .094	.190 ± .147	.140 ± .141	.199 ± .136	.140 ± .146	.101 ± .101	.089† ± .091	**.087 ± .092**
tictactoe	.018 ± .017	**.016 ± .018**	.017† ± .017	.027 ± .073	.017† ± .018	.019 ± .023	.043 ± .143	.018 ± .017	.032 ± .118
mammographic	.152 ± .182	.150 ± .140	.150 ± .181	.175 ± .182	.151 ± .181	.106 ± .122	**.061 ± .069**	.078 ± .085	.090 ± .099
transfusion	.076 ± .074	.093 ± .084	.422 ± .139	.490 ± .234	.448 ± .118	.083 ± .143	.091 ± .133	.074 ± .083	**.059 ± .078**
breast-cancer	.084 ± .143	.068 ± .124	.086 ± .132	.099 ± .151	.086 ± .132	.053 ± .097	**.021 ± .059**	.029 ± .036	.038 ± .063
balance.1	.045† ± .084	.080 ± .072	.043 ± .087	.057 ± .099	.042 ± .086	.038 ± .070	**.032 ± .028**	.034 ± .052	.035 ± .060
balance.3	**.034 ± .068**	.046 ± .066	.036 ± .071	.070 ± .090	.036 ± .071	.047† ± .116	.050 ± .131	.053† ± .145	.047† ± .126
wdbc	.063† ± .117	.083 ± .092	.066† ± .126	.102 ± .120	.065† ± .124	.057 ± .106	**.043 ± .040**	.046 ± .067	.051 ± .081
ionosphere	.101† ± .098	.179 ± .083	**.094 ± .127**	.183 ± .141	.120 ± .162	.096† ± .098	.109 ± .056	.101† ± .104	.096 ± .102
haberman	.094 ± .065	**.084 ± .080**	.302 ± .158	.467 ± .261	.293 ± .106	.123 ± .103	.126 ± .102	.098 ± .088	.093 ± .091
spectf	.125 ± .091	.078† ± .076	.163 ± .159	.234 ± .238	.170 ± .146	.077 ± .094	.115 ± .108	.222 ± .172	**.075 ± .078**
sonar	.090 ± .116	.130 ± .089	.093 ± .126	.256 ± .160	.088† ± .120	.237 ± .242	.093 ± .092	**.067 ± .067**	.103 ± .119
wine.1	.053 ± .000	**.000 ± .000**	.046 ± .005	.009 ± .070	.045 ± .002	.052 ± .013	.124 ± .061	.053 ± .000	.059 ± .056
wine.2	.025 ± .019	.035 ± .019	.023 ± .019	**.023 ± .073**	.023 ± .019	.025 ± .019	.051 ± .128	.025 ± .019	.035 ± .107
wine.3	.064 ± .121	.071 ± .131	.065 ± .128	.068 ± .128	.065 ± .128	.046 ± .089	**.038 ± .078**	.046 ± .091	.042 ± .081
iris.2	.223 ± .093	.179 ± .102	**.099 ± .092**	.126 ± .122	.205 ± .157	.163 ± .069	.256 ± .092	.209 ± .087	.146 ± .073
iris.3	.075 ± .132	.117 ± .140	.076 ± .134	.108 ± .112	.078 ± .130	.075† ± .132	**.049 ± .059**	.075 ± .133	.065† ± .098
Average	.101 ± .127	.098 ± .102	.134 ± .160	.178 ± .198	.143 ± .166	.083 ± .116	.083 ± .102	.077 ± .097	**.069 ± .091**

Table 5 CAP results, in terms of AE obtained on F_1 evaluation measure, for multiclass classifiers trained via LR

	Baselines					*Our Methods*			
	Naïve	DoC	DS	L-CBPE	NN	LEAP_{ACC}	$\text{LEAP}_{\text{KDEy}}$	$\text{S-LEAP}_{\text{KDEy}}$	$\text{O-LEAP}_{\text{KDEy}}$
poker-hand	.065 ± .029	**.042 ± .053**	.086 ± .035	.092 ± .046	.083 ± .032	.046 ± .051	.101 ± .137	.141 ± .111	.048 ± .075
connect-4	.144 ± .100	.062 ± .053	.156 ± .101	.182 ± .108	.150 ± .077	.151 ± .110	.052†± .049	**.049 ± .048**	.049†± .047
shuttle	.155 ± .109	.092 ± .072	.152 ± .106	.189 ± .121	.155 ± .110	.096 ± .083	.046†± .054	.049 ± .053	**.045 ± .050**
chess	.058 ± .041	**.043 ± .033**	.057 ± .040	.087 ± .066	.064 ± .042	.069 ± .050	.056 ± .043	.046†± .036	.051 ± .040
letter	.088 ± .055	**.050 ± .038**	.090 ± .056	.119 ± .080	.089 ± .053	.109 ± .072	.099 ± .060	.062 ± .050	.071 ± .053
dry-bean	.053 ± .052	.053 ± .035	.054 ± .052	.084 ± .059	.052†± .051	.063 ± .060	.064 ± .058	**.048 ± .046**	.053 ± .052
nursery	.084 ± .063	.054 ± .046	.074 ± .063	.112 ± .078	.073 ± .059	.061 ± .059	.090 ± .086	**.042 ± .047**	.044†± .054
hand-digits	.048 ± .044	**.042 ± .030**	.046†± .042	.073 ± .053	.046†± .042	.065 ± .058	.084 ± .060	.053 ± .049	.056 ± .052
isolet	.035†± .029	**.033 ± .022**	.035†± .029	.245 ± .083	.034†± .029	.116 ± .066	.153 ± .092	.084 ± .053	.124 ± .064
wine-quality	.117 ± .055	**.052 ± .047**	.141 ± .066	.169 ± .086	.134 ± .063	.106 ± .064	.066 ± .056	.098 ± .095	.056†± .048
satellite	.091 ± .068	.059 ± .044	.082 ± .063	.121 ± .094	.080 ± .060	.077 ± .062	.083 ± .065	**.051 ± .043**	.053†± .046
digits	.041 ± .040	**.033 ± .030**	.043 ± .041	.068 ± .051	.042 ± .041	.058 ± .055	.083 ± .061	.046 ± .044	.052 ± .052
page-block	.178 ± .100	.104 ± .070	.153 ± .101	.217 ± .112	.145 ± .094	.142 ± .086	.101 ± .063	.121 ± .069	**.083 ± .058**
waveform-v1	.084 ± .081	.071 ± .058	.082 ± .081	.133 ± .102	.084 ± .084	.059 ± .060	.067 ± .066	**.051 ± .054**	.053 ± .053
academic-success	.131 ± .102	.079 ± .056	.116 ± .080	.148 ± .098	.119 ± .078	.083 ± .068	.066 ± .053	.062 ± .047	**.057 ± .045**
abalone	.048 ± .034	**.043 ± .037**	.050 ± .036	.059 ± .047	.050 ± .036	.107 ± .058	.104 ± .054	.066 ± .049	.070 ± .051
molecular	.084 ± .084	.064 ± .060	.085 ± .085	.156 ± .088	.084 ± .085	.059 ± .062	.053 ± .051	**.044 ± .042**	.046†± .047
image-seg	.046 ± .042	.040 ± .033	.048 ± .043	.075 ± .056	.047 ± .041	.055 ± .056	.073 ± .063	**.035 ± .034**	.049 ± .050
obesity	.076 ± .059	**.056 ± .042**	.082 ± .062	.126 ± .091	.081 ± .058	.081 ± .068	.118 ± .075	.057†± .047	.063†± .053
cmc	.107 ± .069	**.057 ± .047**	.110 ± .070	.118 ± .090	.101 ± .069	.071 ± .056	.071 ± .054	.060†± .048	.058†± .046
hcv	.067 ± .040	**.039 ± .029**	.072 ± .042	.100 ± .072	.069 ± .041	.058 ± .042	.048 ± .037	.044 ± .033	.045 ± .032
phishing	.134 ± .110	.078 ± .064	.141 ± .108	.284 ± .180	.132 ± .110	.142 ± .111	.061 ± .053	.054 ± .051	**.050 ± .048**
yeast	.121 ± .070	.083 ± .061	.113 ± .069	.212 ± .120	.116 ± .067	.080 ± .064	.080 ± .062	.076 ± .058	**.074 ± .057**
mhr	.074 ± .068	**.067 ± .051**	.072†± .063	.132 ± .107	.071†± .064	.102 ± .078	.088 ± .070	.068†± .053	.072†± .057
Average	.089 ± .079	**.058 ± .052**	.089 ± .077	.137 ± .109	.088 ± .075	.086 ± .075	.079 ± .071	.063 ± .061	.059†± .055

While its performance on multiclass datasets ranks among the best, it drops significantly in the binary setting, where it is outperformed on most datasets by all members of the LEAP family. This may be partly explained by the difficulty of regressing a non-linear measure like F_1, which places uneven emphasis on the positive class in binary settings. This asymmetry is mitigated in the macro-averaged version, where all classes contribute equally to the final score.

6 Conclusions

In this paper we have proposed LEAP, a family of methods for classifier accuracy prediction (CAP) specifically designed to address situations affected by prior probability shift (PPS). Our methods work by estimating the entries of the contingency table that results from classifying the test set, from which any evaluation measure can be computed. As a result, and in contrast to previously existing method, LEAP is not limited to any specific evaluation measure, and can compute more than one such measure at the same time, without requiring retraining. We have also drawn a connection with the field of quantification, and through several experiments we have shown that, when our CAP methods are endowed with state-of-the-art quantification methods, they tends to outperform existing CAP methods.

We have also shown that the problem is overconstrained, and have discussed different strategies for selecting the system of equations that, once solved, provides the required estimates. In particular, O-LEAP, which considers *all* the constraints governing the problem, tends to achieve the best results. Additionally, we have shown that some of our proposed methods do not scale well when the number of classes is large; in such cases, our S-LEAP method should be preferred to O-LEAP, since it strikes a good balance between CAP accuracy and computational cost.

Possible directions for future work include considering multi-objective optimization problems, in which our method could be especially useful due to its ability to estimate more than one evaluation measure simultaneously. Additionally, we aim to extend our methods

Table 6 CAP results, in terms of AE, for binary classifiers trained via *k*NN

	Baselines							*Our Methods*			
	Naïve	ATC	DoC	DS	L-CBPE	NN	Q-COT	LEAP_{ACC}	$\text{LEAP}_{\text{KDEy}}$	$\text{S-LEAP}_{\text{KDEy}}$	$\text{O-LEAP}_{\text{KDEy}}$
pageblocks.5	.355 ± .217	.242 ± .148	.041 ± .034	.170 ± .105	.156 ± .117	.169 ± .096	.067 ± .039	.094 ± .077	.048 ± .044	.037 ± .033	**.035 ± .030**
wine-q-white	.077 ± .054	.047 ± .037	.042 ± .032	.067 ± .047	.047 ± .035	.064 ± .049	.051 ± .043	.041 ± .035	.042 ± .033	.038† ± .030	**.037 ± .030**
spambase	.045 ± .031	.074 ± .048	.029† ± .023	.050 ± .031	.035 ± .027	.048 ± .030	.036 ± .026	.029 ± .021	.037 ± .031	.027† ± .020	**.027 ± .020**
ctg.1	.068 ± .052	.100 ± .062	.060 ± .041	.069 ± .042	.057 ± .037	.070 ± .044	.039 ± .029	.049 ± .039	**.025 ± .019**	.043 ± .033	.044 ± .033
ctg.2	.170 ± .125	.184 ± .107	**.029 ± .022**	.136 ± .075	.050 ± .036	.104 ± .076	.036 ± .026	.116 ± .083	.101 ± .078	.062 ± .052	.057 ± .047
ctg.3	.203 ± .136	.126 ± .085	.071 ± .049	.114 ± .078	.068 ± .051	.115 ± .087	.043 ± .034	.070 ± .060	.043 ± .035	.053 ± .045	**.041 ± .035**
wine-q-red	.039† ± .030	.147 ± .061	.053 ± .037	.039† ± .030	.055 ± .041	.039 ± .030	.056 ± .049	**.036 ± .028**	.043 ± .035	.040 ± .030	.038† ± .029
semeion	.149 ± .105	.085 ± .059	.031 ± .025	.091 ± .058	.055 ± .042	.084 ± .055	.028† ± .020	.041 ± .037	.038 ± .032	.032 ± .028	**.027 ± .022**
yeast	.137 ± .097	.109 ± .078	.072† ± .053	.136 ± .087	.073† ± .051	.119 ± .074	**.070 ± .047**	.143 ± .097	.100 ± .076	.072† ± .055	.078† ± .057
cmc.1	.149 ± .091	.230 ± .147	.131 ± .094	.132 ± .088	.102 ± .084	.129 ± .082	.095 ± .064	.094 ± .079	.065† ± .049	**.064 ± .046**	.072 ± .049
cmc.2	.249 ± .181	.346 ± .208	.089† ± .068	.196 ± .120	.169 ± .129	.145 ± .099	**.083 ± .055**	.172 ± .142	.109 ± .080	.108 ± .080	.101 ± .074
cmc.3	.177 ± .120	.247 ± .164	.125 ± .079	.163 ± .098	.126 ± .096	.185 ± .111	.113 ± .069	.129 ± .105	**.076 ± .061**	.094 ± .071	.101 ± .076
german	.146 ± .100	.237 ± .124	.094 ± .068	.126 ± .083	.095 ± .060	.092 ± .070	**.069 ± .049**	.138 ± .096	.079 ± .061	.075 ± .058	.082 ± .059
tictactoe	.075 ± .052	.069 ± .046	.052 ± .033	.064 ± .040	.070 ± .044	.067 ± .042	.072 ± .027	.030 ± .026	.029 ± .031	.024 ± .020	**.021 ± .018**
mammographic	.055 ± .037	.104 ± .047	.056 ± .037	.050† ± .035	.089 ± .046	**.050 ± .035**	.059 ± .039	.055 ± .037	.062 ± .044	.055 ± .037	.055 ± .037
transfusion	.229 ± .166	.190 ± .132	.074† ± .058	.179 ± .139	.133 ± .100	.206 ± .158	**.070 ± .055**	.163 ± .120	.129 ± .093	.080 ± .060	.128 ± .085
breast-cancer	.017 ± .011	.024 ± .018	.020 ± .012	.020 ± .012	.031 ± .030	.020 ± .012	.024 ± .016	.017 ± .011	**.011 ± .008**	.017 ± .010	.017 ± .010
balance.1	.039 ± .025	.049 ± .035	.068 ± .036	.040 ± .026	.043 ± .031	.039 ± .025	.096 ± .029	.031 ± .025	.041 ± .037	.029 ± .023	**.029 ± .023**
balance.3	.026 ± .021	.071 ± .031	.030 ± .024	.026 ± .021	.027 ± .021	.027 ± .022	.090 ± .025	.021† ± .017	.025 ± .021	.021 ± .017	**.021 ± .016**
wdbc	.044 ± .032	.033 ± .024	.029 ± .021	.036 ± .029	.029† ± .023	.036 ± .030	**.026 ± .019**	.031 ± .022	.029† ± .023	.032 ± .023	.031 ± .022
ionosphere	.153 ± .103	.056 ± .048	.083 ± .050	.135 ± .078	.076 ± .050	.142 ± .086	**.035 ± .028**	.071 ± .056	.083 ± .054	.073 ± .052	.079 ± .053
haberman	.236 ± .167	.173 ± .128	.174 ± .111	.233 ± .135	.212 ± .168	.267 ± .190	.197 ± .140	.266 ± .201	.103 ± .077	**.090 ± .072**	.095† ± .078
spectf	.216 ± .157	.257 ± .130	.132 ± .077	.127 ± .090	.171 ± .124	**.082 ± .058**	.142 ± .101	.379 ± .203	.088† ± .067	.096 ± .069	.134 ± .084
sonar	.140 ± .092	.118 ± .085	.120 ± .085	.163 ± .088	.099 ± .065	.167 ± .095	.105 ± .072	.190 ± .118	.170 ± .108	.103 ± .054	**.091 ± .048**
wine.1	**.012 ± .008**	.014† ± .014	.014† ± .014	.014 ± .008	.018 ± .017	.014 ± .009	.051 ± .023	.023 ± .015	.040 ± .029	.021 ± .014	.022 ± .015
wine.2	.084 ± .032	.090 ± .060	.137 ± .056	.117 ± .034	**.073 ± .062**	.109 ± .032	.100 ± .036	.150 ± .078	.195 ± .106	.128 ± .072	.123 ± .069
wine.3	.034 ± .026	.053 ± .037	.047 ± .032	.038 ± .030	.030 ± .027	.038 ± .030	.016† ± .012	.033 ± .028	**.014 ± .014**	.033 ± .027	.026 ± .021
iris.2	.059 ± .015	**.017 ± .016**	.044 ± .030	.071 ± .034	.037 ± .047	.064 ± .042	.160 ± .049	.086 ± .046	.129 ± .082	.083 ± .049	.084 ± .044
iris.3	.121 ± .091	.037 ± .030	.037 ± .028	.106 ± .077	.083 ± .053	.106 ± .082	**.031 ± .021**	.085 ± .063	.049 ± .032	.056 ± .046	.043 ± .032
Average	.121 ± .130	.122 ± .126	.068 ± .066	.100 ± .093	.080 ± .085	.096 ± .097	.071 ± .066	.096 ± .117	.069 ± .073	**.058 ± .055**	.060† ± .059

Table 7 CAP results, in terms of AE, for multiclass classifiers trained via *k*NN

	Baselines							*Our Methods*			
	Naïve	ATC	DoC	DS	L-CBPE	NN	Q-COT	LEAP_{ACC}	$\text{LEAP}_{\text{KDEy}}$	$\text{S-LEAP}_{\text{KDEy}}$	$\text{O-LEAP}_{\text{KDEy}}$
poker-hand	.391 ± .084	.424 ± .091	.066† ± .050	.069 ± .050	.423 ± .087	**.065 ± .048**	.102 ± .078	.093 ± .072	.099 ± .072	.101 ± .062	.111 ± .069
connect-4	.238 ± .161	.183 ± .125	.083 ± .061	.146 ± .098	.141 ± .104	.105 ± .082	.085 ± .068	.118 ± .096	.063† ± .050	.063† ± .050	**.061 ± .047**
shuttle	.185 ± .146	.085 ± .072	.072 ± .065	.118 ± .085	.109 ± .104	.117 ± .088	**.033 ± .030**	.062 ± .057	.034† ± .031	.036 ± .033	.036 ± .033
chess	.076 ± .049	.069 ± .050	.044 ± .033	.048 ± .036	.084 ± .050	.049 ± .037	.041† ± .030	.053 ± .039	.066 ± .046	.039† ± .029	**.039 ± .028**
letter	.026 ± .019	.060 ± .030	**.022 ± .016**	.026 ± .019	.106 ± .067	.026 ± .019	.139 ± .030	.029 ± .021	.068 ± .033	.026 ± .019	.033 ± .022
dry-bean	.025 ± .018	.040 ± .026	**.019 ± .014**	.022 ± .016	.026 ± .019	.023 ± .017	.028 ± .018	.021 ± .016	.027 ± .020	.021 ± .015	.021 ± .015
nursery	.161 ± .140	.184 ± .135	.081 ± .061	.112 ± .082	.165 ± .088	.128 ± .087	.163 ± .075	.080 ± .065	.058 ± .039	.032† ± .024	**.032 ± .023**
hand-digits	.011 ± .007	.011 ± .011	**.009 ± .007**	.011 ± .008	.012 ± .010	.011 ± .008	.033 ± .014	.010 ± .008	.016 ± .008	.010 ± .008	.009† ± .006
isolet	.033 ± .024	.031 ± .023	**.025 ± .019**	.035 ± .025	.107 ± .079	.035 ± .025	.165 ± .034	.039 ± .027	.078 ± .035	.029 ± .021	.043 ± .026
wine-quality	.222 ± .109	.213 ± .099	**.087 ± .061**	.099 ± .067	.192 ± .113	.093 ± .063	.173 ± .088	.140 ± .100	.113 ± .068	.117 ± .073	.091† ± .064
satellite	.041 ± .032	.066 ± .040	**.028 ± .021**	.040 ± .029	.038 ± .029	.044 ± .031	.029† ± .021	.036 ± .030	.058 ± .040	.030† ± .023	.032 ± .025
digits	.018 ± .014	.017 ± .015	.017 ± .014	.019 ± .016	.038 ± .023	.018 ± .015	.051 ± .020	.017 ± .014	.031 ± .017	.017 ± .014	**.015 ± .012**
page-block	.325 ± .171	.239 ± .165	.122 ± .099	.170 ± .131	.223 ± .159	.146 ± .105	.088 ± .056	.153 ± .126	**.066 ± .050**	.090 ± .069	.075 ± .056
waveform-v1	.031† ± .023	.078 ± .040	**.029 ± .022**	.031† ± .023	.063 ± .036	.031† ± .024	.060 ± .033	.030† ± .023	.035 ± .026	.030† ± .023	.029† ± .022
academic-success	.174 ± .136	.169 ± .117	.081 ± .059	.132 ± .100	.137 ± .104	.130 ± .099	.078 ± .059	.156 ± .135	**.055 ± .044**	.056† ± .043	.071 ± .055
abalone	.059 ± .041	.074 ± .051	.046† ± .036	.050 ± .041	.163 ± .063	.050 ± .040	.054 ± .037	.055 ± .042	.053 ± .040	**.042 ± .031**	.043† ± .031
molecular	.048 ± .034	.095 ± .050	.042 ± .030	.043 ± .032	.173 ± .049	.043 ± .032	.172 ± .040	.032 ± .024	.040 ± .032	.031† ± .023	**.030 ± .022**
image-seg	.045 ± .037	.036 ± .032	.032 ± .025	.043 ± .035	.042 ± .032	.046 ± .037	.033 ± .024	.029 ± .024	.036 ± .026	.025 ± .022	**.023 ± .019**
obesity	.069 ± .051	.047 ± .037	.039 ± .029	.069 ± .051	.062 ± .048	.064 ± .050	.039 ± .029	.050 ± .042	.060 ± .043	.035† ± .027	**.035 ± .028**
cmc	.052† ± .039	.295 ± .068	.057 ± .042	.054† ± .041	.081 ± .053	.058 ± .043	.052† ± .038	.068 ± .046	.086 ± .059	.050† ± .038	**.049 ± .037**
hcv	.052 ± .035	.569 ± .063	.045† ± .033	.053 ± .036	.186 ± .052	.053 ± .036	.123 ± .057	.064 ± .043	.051 ± .040	.055 ± .037	**.045 ± .032**
phishing	.204 ± .178	.115 ± .097	.064 ± .049	.159 ± .112	.136 ± .120	.137 ± .115	.077 ± .062	.111 ± .090	.044 ± .035	.039 ± .029	**.036 ± .026**
yeast	.069 ± .049	.089 ± .064	.065 ± .045	.070 ± .050	.081 ± .058	.070 ± .050	.058† ± .042	.055† ± .045	.056† ± .044	.057 ± .043	**.053 ± .041**
mhr	.067 ± .045	.076 ± .053	.068 ± .045	.063 ± .043	.072 ± .055	.068 ± .046	.061 ± .041	.051 ± .042	.053 ± .045	.050 ± .038	**.044 ± .035**
Average	.109 ± .134	.136 ± .152	.052 ± .052	.070 ± .076	.119 ± .114	.067 ± .070	.081 ± .067	.065 ± .075	.056 ± .047	.045 ± .045	**.044 ± .043**

to other types of dataset shift, incorporating different assumptions about data distributions, such as covariate shift. Finally, we plan to explore their effectiveness in estimating the accuracy of classifiers trained via deep learning.

Results for classifiers trained via kNN, SVMs, MLP, RFs

In this Appendix we report the results of the complete set of experiments we have carried out for the classifiers trained via *k*NN (Tables 6 and 7), SVMs (Tables 8 and 9), MLP (Tables 10 and 11), and RFs (Tables 12 and 13).

We also report, in Fig. 5, the average inference times (note the log scale on the y-axis) as a function of the number of classes for the multiclass setting. ATC and DoC are markedly faster than the proposed methods, consistently achieving execution times below a millisecond. However, this is not a major concern, as the CAP problem is rarely affected by real-time constraints, and since our methods require, in the worst case we have tested (i.e., for

Table 8 CAP results, in terms of AE, for binary classifiers trained via SVMs

	Baselines							*Our Methods*			
	Naïve	ATC	DoC	DS	L-CBPE	NN	Q-COT	LEAP_{ACC}	$\text{LEAP}_{\text{KDEy}}$	$\text{S-LEAP}_{\text{KDEy}}$	$\text{O-LEAP}_{\text{KDEy}}$
poker-hand	.405 ± .090	.424 ± .109	.078 ± .052	.073† ± .055	.401 ± .092	.069† ± .055	.088 ± .068	**.069 ± .053**	.103 ± .071	.095 ± .072	.081 ± .055
connect-4	.264 ± .189	.159 ± .124	.081 ± .059	.166 ± .117	.175 ± .142	.143 ± .105	.097 ± .077	.186 ± .152	.076† ± .062	**.070 ± .055**	.082 ± .064
shuttle	.208 ± .162	.137 ± .107	.069 ± .062	.131 ± .111	.157 ± .143	.116 ± .109	.080 ± .066	.062 ± .055	**.047 ± .044**	.058 ± .049	.059 ± .052
chess	.093 ± .058	.075 ± .055	.060 ± .043	.060 ± .043	.080 ± .051	.062 ± .043	**.042 ± .033**	.056 ± .043	.049 ± .037	.063 ± .043	.046† ± .035
letter	.024 ± .018	.023 ± .018	**.020 ± .015**	.024 ± .019	.185 ± .079	.024 ± .019	.113 ± .028	.032 ± .021	.053 ± .029	.023 ± .018	.025 ± .018
dry-bean	.023 ± .017	.021 ± .016	**.018 ± .013**	.021 ± .016	.025 ± .019	.022 ± .016	.028 ± .017	.021 ± .016	.024 ± .017	.020 ± .015	.020 ± .014
nursery	.124 ± .105	.115 ± .096	.116 ± .097	.114 ± .101	.099 ± .094	.117 ± .104	.079 ± .072	**.062 ± .061**	.095 ± .086	.114 ± .096	.109 ± .092
hand-digits	.008 ± .006	**.006 ± .007**	.006† ± .005	.008 ± .006	.018 ± .011	.008 ± .006	.043 ± .012	.008 ± .006	.010 ± .006	.008 ± .006	.007 ± .005
isolet	.018† ± .013	.019† ± .015	.019 ± .013	.018† ± .014	.198 ± .082	.018† ± .013	.168 ± .026	.027 ± .017	.048 ± .024	**.017 ± .013**	.024 ± .016
wine-quality	.272 ± .127	.244 ± .122	.113† ± .074	.112† ± .074	.278 ± .147	.118† ± .080	.114† ± .073	.207 ± .135	**.111 ± .078**	.121 ± .083	.143 ± .084
satellite	.051 ± .043	.033 ± .027	**.027 ± .021**	.043 ± .034	.046 ± .034	.045 ± .036	.033 ± .025	.035 ± .027	.039 ± .029	.028† ± .021	.027† ± .020
digits	.015 ± .013	.014 ± .012	.014 ± .011	.016 ± .013	.041 ± .020	.016 ± .013	.061 ± .016	.014 ± .010	.021 ± .013	.015 ± .013	**.011 ± .009**
page-block	.303 ± .167	.159 ± .109	.122 ± .092	.146 ± .120	.241 ± .154	.138 ± .121	**.071 ± .047**	.152 ± .111	.086 ± .054	.100 ± .065	.086 ± .052
waveform-v1	.031 ± .024	.037 ± .029	.033 ± .024	.031 ± .024	.033 ± .026	.031 ± .024	.031† ± .023	.030 ± .023	.037 ± .028	.029 ± .022	**.028 ± .022**
academic-success	.152 ± .119	.090 ± .068	.063† ± .044	.115 ± .090	.112 ± .083	.114 ± .089	.061† ± .046	.087 ± .072	**.056 ± .043**	.058† ± .046	.056† ± .044
abalone	.060 ± .045	.084 ± .061	.047 ± .037	.059 ± .047	.102 ± .059	.054 ± .043	**.044 ± .034**	.079 ± .057	.063 ± .048	.052 ± .039	.046† ± .035
molecular	.022 ± .016	.030 ± .022	**.020 ± .015**	.024 ± .016	.035 ± .028	.024 ± .016	.032 ± .021	.023 ± .016	.026 ± .019	.023 ± .016	.023 ± .016
image-seg	.038 ± .033	.039 ± .038	.032 ± .028	.037 ± .032	.052 ± .033	.039 ± .033	.047 ± .028	.025 ± .021	.023† ± .018	.027 ± .024	**.022 ± .019**
obesity	.046 ± .034	.041 ± .031	.037 ± .028	.043 ± .034	.077 ± .049	.043 ± .033	.085 ± .038	.043 ± .032	.071 ± .044	**.031 ± .024**	.037 ± .027
cmc	.064 ± .046	.059 ± .046	.062 ± .045	.059 ± .043	.052† ± .040	**.052 ± .038**	.071 ± .045	.065 ± .053	.060 ± .045	.052† ± .040	.062 ± .047
hcv	.048 ± .035	.055 ± .042	.054 ± .041	.048† ± .035	.052 ± .037	.060 ± .041	**.047 ± .034**	.149 ± .098	.099 ± .071	.125 ± .087	.083 ± .059
phishing	.216 ± .194	.118 ± .104	.072 ± .052	.162 ± .108	.181 ± .164	.190 ± .117	.070 ± .053	.157 ± .133	.042† ± .033	.048 ± .037	**.041 ± .034**
yeast	.068 ± .047	.071 ± .054	.078 ± .056	.068 ± .048	.073 ± .051	.071 ± .050	.060† ± .047	.060 ± .045	.064 ± .049	.056 ± .042	**.055 ± .041**
mhr	.106 ± .064	.078 ± .055	.099 ± .059	.106 ± .064	.106 ± .065	.103 ± .061	.101 ± .063	.055† ± .045	.055† ± .044	.057 ± .042	**.054 ± .041**
Average	.111 ± .141	.089 ± .113	.056 ± .058	.070 ± .080	.117 ± .125	.070 ± .080	.069 ± .056	.071 ± .088	.057 ± .054	.054 ± .058	**.051 ± .054**

Table 9 CAP results, in terms of AE, for multiclass classifiers trained via SVMs

	Baselines							*Our Methods*			
	Naïve	ATC	DoC	DS	L-CBPE	NN	Q-COT	LEAP_{ACC}	$\text{LEAP}_{\text{KDEy}}$	$\text{S-LEAP}_{\text{KDEy}}$	$\text{O-LEAP}_{\text{KDEy}}$
poker-hand	.405 ± .090	.424 ± .109	.078 ± .052	.073† ± .055	.401 ± .092	.069† ± .055	.088 ± .068	**.069 ± .053**	.103 ± .071	.095 ± .072	.081 ± .055
connect-4	.264 ± .189	.159 ± .124	.081 ± .059	.166 ± .117	.175 ± .142	.143 ± .105	.097 ± .077	.186 ± .152	.076† ± .062	**.070 ± .055**	.082 ± .064
shuttle	.208 ± .162	.137 ± .107	.069 ± .062	.131 ± .111	.157 ± .143	.116 ± .109	.080 ± .066	.062 ± .055	**.047 ± .044**	.058 ± .049	.059 ± .052
chess	.093 ± .058	.075 ± .055	.060 ± .043	.060 ± .043	.080 ± .051	.062 ± .043	**.042 ± .033**	.056 ± .043	.049 ± .037	.063 ± .043	.046† ± .035
letter	.024 ± .018	.023 ± .018	**.020 ± .015**	.024 ± .019	.185 ± .079	.024 ± .019	.113 ± .028	.032 ± .021	.053 ± .029	.023 ± .018	.025 ± .018
dry-bean	.023 ± .017	.021 ± .016	**.018 ± .013**	.021 ± .016	.025 ± .019	.022 ± .016	.028 ± .017	.021 ± .016	.024 ± .017	.020 ± .015	.020 ± .014
nursery	.124 ± .105	.115 ± .096	.116 ± .097	.114 ± .101	.099 ± .094	.117 ± .104	.079 ± .072	**.062 ± .061**	.095 ± .086	.114 ± .096	.109 ± .092
hand-digits	.008 ± .006	**.006 ± .007**	.006† ± .005	.008 ± .006	.018 ± .011	.008 ± .006	.043 ± .012	.008 ± .006	.010 ± .006	.008 ± .006	.007 ± .005
isolet	.018† ± .013	.019† ± .015	.019 ± .013	.018† ± .014	.198 ± .082	.018† ± .013	.168 ± .026	.027 ± .017	.048 ± .024	**.017 ± .013**	.024 ± .016
wine-quality	.272 ± .127	.244 ± .122	.113† ± .074	.112† ± .074	.278 ± .147	.118† ± .080	.114† ± .073	.207 ± .135	**.111 ± .078**	.121 ± .083	.143 ± .084
satellite	.051 ± .043	.033 ± .027	**.027 ± .021**	.043 ± .034	.046 ± .034	.045 ± .036	.033 ± .025	.035 ± .027	.039 ± .029	.028† ± .021	.027† ± .020
digits	.015 ± .013	.014 ± .012	.014 ± .011	.016 ± .013	.041 ± .020	.016 ± .013	.061 ± .016	.014 ± .010	.021 ± .013	.015 ± .013	**.011 ± .009**
page-block	.303 ± .167	.159 ± .109	.122 ± .092	.146 ± .120	.241 ± .154	.138 ± .121	**.071 ± .047**	.152 ± .111	.086 ± .054	.100 ± .065	.086 ± .052
waveform-v1	.031 ± .024	.037 ± .029	.033 ± .024	.031 ± .024	.033 ± .026	.031 ± .024	.031† ± .023	.030 ± .023	.037 ± .028	.029 ± .022	**.028 ± .022**
academic-success	.152 ± .119	.090 ± .068	.063† ± .044	.115 ± .090	.112 ± .083	.114 ± .089	.061† ± .046	.087 ± .072	**.056 ± .043**	.058† ± .046	.056† ± .044
abalone	.060 ± .045	.084 ± .061	.047 ± .037	.059 ± .047	.102 ± .059	.054 ± .043	**.044 ± .034**	.079 ± .057	.063 ± .048	.052 ± .039	.046† ± .035
molecular	.022 ± .016	.030 ± .022	**.020 ± .015**	.024 ± .016	.035 ± .028	.024 ± .016	.032 ± .021	.023 ± .016	.026 ± .019	.023 ± .016	.023 ± .016
image-seg	.038 ± .033	.039 ± .038	.032 ± .028	.037 ± .032	.052 ± .033	.039 ± .033	.047 ± .028	.025 ± .021	.023† ± .018	.027 ± .024	**.022 ± .019**
obesity	.046 ± .034	.041 ± .031	.037 ± .028	.043 ± .034	.077 ± .049	.043 ± .033	.085 ± .038	.043 ± .032	.071 ± .044	**.031 ± .024**	.037 ± .027
cmc	.064 ± .046	.059 ± .046	.062 ± .045	.059 ± .043	.052† ± .040	**.052 ± .038**	.071 ± .045	.065 ± .053	.060 ± .045	.052† ± .040	.062 ± .047
hcv	.048 ± .035	.055 ± .042	.054 ± .041	.048† ± .035	.052 ± .037	.060 ± .041	**.047 ± .034**	.149 ± .098	.099 ± .071	.125 ± .087	.083 ± .059
phishing	.216 ± .194	.118 ± .104	.072 ± .052	.162 ± .108	.181 ± .164	.190 ± .117	.070 ± .053	.157 ± .133	.042† ± .033	.048 ± .037	**.041 ± .034**
yeast	.068 ± .047	.071 ± .054	.078 ± .056	.068 ± .048	.073 ± .051	.071 ± .050	.060† ± .047	.060 ± .045	.064 ± .049	.056 ± .042	**.055 ± .041**
mhr	.106 ± .064	.078 ± .055	.099 ± .059	.106 ± .064	.106 ± .065	.103 ± .061	.101 ± .063	.055† ± .045	.055† ± .044	.057 ± .042	**.054 ± .041**
Average	.111 ± .141	.089 ± .113	.056 ± .058	.070 ± .080	.117 ± .125	.070 ± .080	.069 ± .056	.071 ± .088	.057 ± .054	.054 ± .058	**.051 ± .054**

Table 10 CAP results, in terms of AE, for binary classifiers trained via MLP

	Baselines							*Our Methods*			
	Naïve	ATC	DoC	DS	L-CBPE	NN	Q-COT	LEAP_{ACC}	$\text{LEAP}_{\text{KDEy}}$	$\text{S-LEAP}_{\text{KDEy}}$	$\text{O-LEAP}_{\text{KDEy}}$
poker-hand	.405 ± .090	.424 ± .109	.078 ± .052	.073† ± .055	.401 ± .092	.069† ± .055	.088 ± .068	**.069 ± .053**	.103 ± .071	.095 ± .072	.081 ± .055
connect-4	.264 ± .189	.159 ± .124	.081 ± .059	.166 ± .117	.175 ± .142	.143 ± .105	.097 ± .077	.186 ± .152	.076† ± .062	**.070 ± .055**	.082 ± .064
shuttle	.208 ± .162	.137 ± .107	.069 ± .062	.131 ± .111	.157 ± .143	.116 ± .109	.080 ± .066	.062 ± .055	**.047 ± .044**	.058 ± .049	.059 ± .052
chess	.093 ± .058	.075 ± .055	.060 ± .043	.060 ± .043	.080 ± .051	.062 ± .043	**.042 ± .033**	.056 ± .043	.049 ± .037	.063 ± .043	.046† ± .035
letter	.024 ± .018	.023 ± .018	**.020 ± .015**	.024 ± .019	.185 ± .079	.024 ± .019	.113 ± .028	.032 ± .021	.053 ± .029	.023 ± .018	.025 ± .018
dry-bean	.023 ± .017	.021 ± .016	**.018 ± .013**	.021 ± .016	.025 ± .019	.022 ± .016	.028 ± .017	.021 ± .016	.024 ± .017	.020 ± .015	.020 ± .014
nursery	.124 ± .105	.115 ± .096	.116 ± .097	.114 ± .101	.099 ± .094	.117 ± .104	.079 ± .072	**.062 ± .061**	.095 ± .086	.114 ± .096	.109 ± .092
hand-digits	.008 ± .006	**.006 ± .007**	.006† ± .005	.008 ± .006	.018 ± .011	.008 ± .006	.043 ± .012	.008 ± .006	.010 ± .006	.008 ± .006	.007 ± .005
isolet	.018† ± .013	.019† ± .015	.019 ± .013	.018† ± .014	.198 ± .082	.018† ± .013	.168 ± .026	.027 ± .017	.048 ± .024	**.017 ± .013**	.024 ± .016
satellite	.051 ± .043	.033 ± .027	**.027 ± .021**	.043 ± .034	.046 ± .034	.045 ± .036	.033 ± .025	.035 ± .027	.039 ± .029	.028† ± .021	.027† ± .020
digits	.015 ± .013	.014 ± .012	.014 ± .011	.016 ± .013	.041 ± .020	.016 ± .013	.061 ± .016	.014 ± .010	.021 ± .013	.015 ± .013	**.011 ± .009**
page-block	.303 ± .167	.159 ± .109	.122 ± .092	.146 ± .120	.241 ± .154	.138 ± .121	**.071 ± .047**	.152 ± .111	.086 ± .054	.100 ± .065	.086 ± .052
waveform-v1	.031 ± .024	.037 ± .029	.033 ± .024	.031 ± .024	.033 ± .026	.031 ± .024	.031† ± .023	.030 ± .023	.037 ± .028	.029 ± .022	**.028 ± .022**
academic-success	.152 ± .119	.090 ± .068	.063† ± .044	.115 ± .090	.112 ± .083	.114 ± .089	.061† ± .046	.087 ± .072	**.056 ± .043**	.058† ± .046	.056† ± .044
abalone	.060 ± .045	.084 ± .061	.047 ± .037	.059 ± .047	.102 ± .059	.054 ± .043	**.044 ± .034**	.079 ± .057	.063 ± .048	.052 ± .039	.046† ± .035
molecular	.022 ± .016	.030 ± .022	**.020 ± .015**	.024 ± .016	.035 ± .028	.024 ± .016	.032 ± .021	.023 ± .016	.026 ± .019	.023 ± .016	.023 ± .016
image-seg	.038 ± .033	.039 ± .038	.032 ± .028	.037 ± .032	.052 ± .033	.039 ± .033	.047 ± .028	.025 ± .021	.023† ± .018	.027 ± .024	**.022 ± .019**
obesity	.046 ± .034	.041 ± .031	.037 ± .028	.043 ± .034	.077 ± .049	.043 ± .033	.085 ± .038	.043 ± .032	.071 ± .044	**.031 ± .024**	.037 ± .027
cmc	.064 ± .046	.059 ± .046	.062 ± .045	.059 ± .043	.052† ± .040	**.052 ± .038**	.071 ± .045	.065 ± .053	.060 ± .045	.052† ± .040	.062 ± .047
hcv	.048 ± .035	.055 ± .042	.054 ± .041	.048† ± .035	.052 ± .037	.060 ± .041	**.047 ± .034**	.149 ± .098	.099 ± .071	.125 ± .087	.083 ± .059
phishing	.216 ± .194	.118 ± .104	.072 ± .052	.162 ± .108	.181 ± .164	.190 ± .117	.070 ± .053	.157 ± .133	.042† ± .033	.048 ± .037	**.041 ± .034**
yeast	.068 ± .047	.071 ± .054	.078 ± .056	.068 ± .048	.073 ± .051	.071 ± .050	.060† ± .047	.060 ± .045	.064 ± .049	.056 ± .042	**.055 ± .041**
mhr	.106 ± .064	.078 ± .055	.099 ± .059	.106 ± .064	.106 ± .065	.103 ± .061	.101 ± .063	.055† ± .045	.055† ± .044	.057 ± .042	**.054 ± .041**
Average	.104 ± .137	.082 ± .107	.053 ± .056	.068 ± .080	.111 ± .119	.068 ± .079	.067 ± .054	.065 ± .080	.054 ± .051	.051 ± .055	**.047 ± .049**

Table 11 CAP results, in terms of AE, for multiclass classifiers trained via MLP

	Baselines							*Our Methods*			
	Naïve	ATC	DoC	DS	L-CBPE	NN	Q-COT	LEAP_{ACC}	$\text{LEAP}_{\text{KDEy}}$	$\text{S-LEAP}_{\text{KDEy}}$	$\text{O-LEAP}_{\text{KDEy}}$
poker-hand	.347 ± .090	.326 ± .090	**.067 ± .049**	.074 ± .053	.341 ± .092	.073 ± .053	.081 ± .065	.094 ± .066	.091 ± .058	.077 ± .058	.072† ± .052
connect-4	.178 ± .134	.088 ± .073	.077 ± .059	.121 ± .081	.244 ± .136	.109 ± .083	.104 ± .069	.069 ± .058	.074 ± .060	.063 ± .052	**.055 ± .045**
shuttle	.122 ± .098	.054 ± .053	.050 ± .039	.078 ± .061	.094 ± .083	.072 ± .064	.038 ± .036	.037 ± .035	.032† ± .029	.032† ± .029	**.031 ± .028**
chess	.068 ± .048	.048 ± .037	**.039 ± .030**	.056 ± .042	.076 ± .051	.052 ± .039	.046 ± .034	.049 ± .037	.075 ± .048	.042 ± .032	.041† ± .030
letter	.021 ± .017	.021 ± .017	**.018 ± .014**	.022 ± .017	.190 ± .078	.021 ± .015	.072 ± .026	.022 ± .017	.056 ± .027	.021 ± .017	.025 ± .017
dry-bean	.023 ± .017	.022 ± .018	**.018 ± .014**	.021 ± .016	.021 ± .017	.021 ± .016	.023 ± .016	.020 ± .015	.023 ± .017	.021 ± .016	.020 ± .014
nursery	.006 ± .008	.003 ± .006	**.001 ± .001**	.006 ± .006	.017 ± .012	.010 ± .010	.031 ± .017	.007 ± .009	.010 ± .008	.003 ± .002	.005 ± .005
hand-digits	.008† ± .006	.007† ± .008	.007† ± .006	.008† ± .006	.008† ± .007	.007† ± .006	.030 ± .013	.007† ± .006	.009 ± .006	.008† ± .006	**.007 ± .005**
isolet	.018 ± .014	.020 ± .016	**.016 ± .012**	.018 ± .014	.191 ± .083	.019 ± .014	.070 ± .024	.019 ± .014	.051 ± .023	.018 ± .013	.024 ± .015
wine-quality	.256 ± .119	.241 ± .120	.113 ± .077	.107† ± .072	.252 ± .137	.111† ± .077	.126 ± .082	.162 ± .114	.125 ± .076	.170 ± .082	**.101 ± .072**
satellite	.051 ± .046	.036 ± .030	.030† ± .025	.043 ± .036	.044 ± .037	.043 ± .036	**.027 ± .020**	.033 ± .027	.036 ± .030	.028† ± .023	.028† ± .022
digits	.014 ± .011	.015 ± .013	**.012 ± .010**	.014 ± .012	.013† ± .011	.014 ± .012	.031 ± .017	.015 ± .012	.025 ± .015	.014 ± .012	.014 ± .010
page-block	.164 ± .101	.053 ± .044	.039 ± .029	.084 ± .066	.092 ± .073	.082 ± .060	**.034 ± .028**	.053 ± .049	.044 ± .034	.041 ± .035	.045 ± .039
waveform-v1	.032† ± .024	.036 ± .028	.034 ± .024	.032† ± .024	.051 ± .034	.030† ± .024	.034 ± .028	.029† ± .021	.034 ± .026	.030† ± .022	**.029 ± .021**
academic-success	.109 ± .090	.082 ± .061	.075 ± .055	.088 ± .077	.166 ± .101	.090 ± .062	.099 ± .063	.059 ± .050	.047† ± .039	.050 ± .039	**.045 ± .034**
abalone	.054 ± .039	.055 ± .041	**.039 ± .030**	.053 ± .042	.088 ± .053	.050 ± .040	.040† ± .031	.065 ± .046	.065 ± .046	.041† ± .032	.049 ± .037
molecular	.024 ± .016	.034 ± .026	.025 ± .018	.026 ± .017	.026 ± .022	.023† ± .017	**.021 ± .016**	.025 ± .018	.026 ± .019	.025 ± .018	.025 ± .018
image-seg	.032 ± .027	.020† ± .017	.023 ± .020	.031 ± .026	.039 ± .027	.029 ± .024	.035 ± .021	.020 ± .017	.026 ± .020	.019† ± .018	**.019 ± .015**
obesity	.042 ± .032	.036 ± .027	.028 ± .019	.042 ± .032	.042 ± .032	.042 ± .031	.052 ± .032	.037 ± .028	.063 ± .038	**.025 ± .018**	.035 ± .026
cmc	.082 ± .055	.059 ± .045	.057 ± .041	.071 ± .048	.110 ± .058	.063 ± .042	.059 ± .044	.078 ± .053	.070 ± .050	**.044 ± .033**	.055 ± .040
hcv	**.046 ± .032**	.062 ± .047	.060 ± .040	.049 ± .034	.384 ± .051	.046† ± .035	.248 ± .082	.053 ± .037	.050† ± .038	.048† ± .036	.063 ± .038
phishing	.154 ± .136	.054 ± .045	.072 ± .062	.130 ± .085	.115 ± .102	.146 ± .109	.068 ± .047	.101 ± .082	.060 ± .047	**.038 ± .030**	.041† ± .030
yeast	.064 ± .046	.061 ± .046	.077 ± .052	.074 ± .053	.094 ± .060	.077 ± .053	.054 ± .040	.061 ± .045	.074 ± .054	**.047 ± .036**	.060 ± .045
mhr	.096 ± .066	**.050 ± .038**	.063 ± .042	.090 ± .065	.075 ± .052	.082 ± .059	.059 ± .041	.060 ± .047	.056† ± .047	.055† ± .037	.059 ± .041
Average	.084 ± .106	.062 ± .085	.043 ± .046	.056 ± .059	.115 ± .123	.055 ± .060	.062 ± .063	.049 ± .057	.051 ± .048	.040 ± .047	**.039 ± .040**

Table 12 CAP results, in terms of AE, for binary classifiers trained via RFs

	Baselines							*Our Methods*			
	Naïve	ATC	DoC	DS	L-CBPE	NN	Q-COT	LEAP_{ACC}	$\text{LEAP}_{\text{KDEy}}$	$\text{S-LEAP}_{\text{KDEy}}$	$\text{O-LEAP}_{\text{KDEy}}$
pageblocks.5	.329 ± .201	.172 ± .110	.050 ± .041	.147 ± .096	.118 ± .131	.187 ± .120	.058 ± .036	.084 ± .071	.033 ± .027	.037 ± .030	**.032 ± .026**
wine-q-white	.086 ± .060	.066 ± .050	.037† ± .029	.078 ± .050	.062 ± .050	.072 ± .055	.052 ± .036	.043 ± .039	.044 ± .034	.038 ± .030	**.036 ± .029**
spambase	.022 ± .015	.023 ± .017	**.019 ± .013**	.023 ± .016	.040 ± .024	.021 ± .015	.062 ± .023	.021 ± .015	.035 ± .026	.021 ± .015	.020 ± .015
ctg.1	.062 ± .046	.061 ± .042	.033 ± .028	.045 ± .032	.072 ± .047	.042 ± .029	.059 ± .030	.030 ± .026	.024† ± .018	.027 ± .022	**.023 ± .019**
ctg.2	.090 ± .068	**.036 ± .031**	.056 ± .044	.095 ± .053	.053 ± .044	.083 ± .054	.056 ± .038	.087 ± .059	.106 ± .073	.062 ± .049	.064 ± .048
ctg.3	.076 ± .053	**.031 ± .027**	.060 ± .043	.049 ± .043	.056 ± .038	.032† ± .027	.034 ± .023	.049 ± .038	.038 ± .029	.047 ± .036	.041 ± .031
wine-q-red	.039 ± .027	.044 ± .036	.043 ± .032	.041 ± .029	.042 ± .033	.056 ± .040	.065 ± .044	**.035 ± .027**	.041 ± .031	.036† ± .027	.035† ± .026
semeion	.204 ± .141	.048 ± .043	.029 ± .026	.122 ± .083	.085 ± .071	.105 ± .066	.033 ± .023	.066 ± .059	.032 ± .026	.042 ± .035	**.026 ± .021**
yeast	.108 ± .074	.069 ± .049	.072 ± .047	.103 ± .063	.062 ± .048	.105 ± .071	.051 ± .039	.052 ± .044	.047 ± .039	.043† ± .033	**.043 ± .033**
cmc.1	.072 ± .048	.068 ± .050	.073 ± .049	.069 ± .047	.052 ± .043	.057 ± .041	.048 ± .038	.045 ± .038	.050 ± .040	.042† ± .033	**.042 ± .033**
cmc.2	.237 ± .171	.151 ± .113	.089 ± .067	.188 ± .134	.174 ± .136	.168 ± .112	**.075 ± .056**	.152 ± .119	.079† ± .063	.090 ± .065	.086 ± .064
cmc.3	.105 ± .074	**.069 ± .053**	.107 ± .074	.097 ± .060	.078 ± .054	.078 ± .052	.079 ± .059	.105 ± .069	.069† ± .054	.070† ± .050	.072† ± .050
german	.160 ± .107	.109 ± .082	**.057 ± .044**	.116 ± .077	.077 ± .063	.118 ± .107	.084 ± .056	.075 ± .069	.074 ± .060	.074 ± .057	.061† ± .048
tictactoe	.046 ± .034	.032 ± .029	.022 ± .019	.041 ± .028	.037 ± .029	.047 ± .034	.144 ± .026	.022 ± .020	.024 ± .024	.020 ± .017	**.017 ± .015**
mammographic	.049 ± .036	.050† ± .037	**.047 ± .034**	.047† ± .035	.166 ± .082	.051 ± .036	.088 ± .043	.050 ± .035	.052 ± .042	.050 ± .035	.049 ± .035
transfusion	.188 ± .137	.188 ± .125	.122 ± .074	.149 ± .091	.124 ± .100	.181 ± .126	.131 ± .092	.100 ± .084	.076† ± .061	**.073 ± .056**	.077 ± .060
breast-cancer	.023 ± .015	**.017 ± .015**	.025 ± .014	.025 ± .016	.028 ± .023	.024 ± .016	.028 ± .019	.019 ± .013	.018† ± .013	.019 ± .012	.019 ± .012
balance.1	.050 ± .037	.079 ± .037	.033 ± .024	.048 ± .036	.099 ± .049	.049 ± .032	.033 ± .025	.050 ± .038	**.028 ± .024**	.051 ± .038	.048 ± .034
balance.3	.045 ± .028	.045 ± .032	.041 ± .024	.044 ± .028	.042 ± .031	.038† ± .027	.070 ± .027	.039† ± .023	**.037 ± .031**	.042 ± .024	.041 ± .023
wdbc	.020 ± .016	.027 ± .021	.048 ± .022	.021 ± .015	.042 ± .023	**.018 ± .015**	.024 ± .016	.022 ± .017	.037 ± .032	.022 ± .017	.022 ± .017
ionosphere	.028 ± .019	**.020 ± .016**	.066 ± .039	.035 ± .021	.037 ± .028	.029 ± .021	.160 ± .057	.032 ± .025	.040 ± .031	.037 ± .027	.043 ± .032
haberman	.194 ± .142	.106 ± .070	.258 ± .175	.177 ± .108	.189 ± .137	**.073 ± .054**	.206 ± .129	.345 ± .181	.117 ± .086	.118 ± .083	.116 ± .076
spectf	.287 ± .210	.236 ± .168	.076† ± .048	.201 ± .138	.202 ± .145	.102 ± .072	**.071 ± .050**	.213 ± .157	.098 ± .077	.241 ± .127	.073† ± .058
sonar	.054 ± .036	.044 ± .034	.064 ± .045	.055 ± .035	.084 ± .058	.137 ± .058	.096 ± .039	.054 ± .042	.104 ± .082	**.036 ± .028**	.039 ± .030
wine.1	.027 ± .015	.056 ± .039	.038 ± .024	.047 ± .021	**.021 ± .024**	.025 ± .014	.094 ± .024	.052 ± .032	.113 ± .075	.041 ± .027	.044 ± .028
wine.2	.033 ± .020	.028 ± .022	**.023 ± .020**	.044 ± .024	.044 ± .032	.032 ± .018	.131 ± .017	.045 ± .031	.063 ± .045	.040 ± .028	.041 ± .028
wine.3	.040 ± .019	.034 ± .024	.046 ± .026	.025 ± .014	.035 ± .024	.026 ± .022	.088 ± .023	.021 ± .017	.020† ± .017	**.020 ± .016**	.020 ± .016
iris.2	**.021 ± .015**	.034 ± .026	.034 ± .025	.022 ± .015	.027 ± .022	.029 ± .018	.036 ± .022	.027 ± .017	.061 ± .031	.027 ± .018	.026 ± .017
iris.3	.062 ± .035	**.021 ± .019**	.100 ± .036	.049 ± .033	.059 ± .043	.049 ± .034	.025† ± .018	.045 ± .028	.030 ± .022	.042 ± .026	.033 ± .024
Average	.095 ± .121	.068 ± .082	.061 ± .068	.076 ± .080	.076 ± .084	.070 ± .075	.075 ± .063	.068 ± .092	.055 ± .055	.052 ± .060	**.045 ± .043**

Table 13 CAP results, in terms of AE, for multiclass classifiers trained via RFs

	Baselines							*Our Methods*			
	Naïve	ATC	DoC	DS	L-CBPE	NN	Q-COT	LEAP_{ACC}	$\text{LEAP}_{\text{KDEy}}$	$\text{S-LEAP}_{\text{KDEy}}$	$\text{O-LEAP}_{\text{KDEy}}$
poker-hand	.434 ± .092	.358 ± .097	.076 ± .051	.080 ± .056	.179 ± .103	**.073 ± .055**	.076† ± .056	.075† ± .056	.101 ± .070	.094 ± .065	.093 ± .068
connect-4	.249 ± .174	.140 ± .111	.088 ± .062	.155 ± .106	.123 ± .093	.108 ± .082	.070† ± .049	.129 ± .106	**.066 ± .051**	.072† ± .056	.069† ± .056
shuttle	.096 ± .079	.045 ± .044	.044 ± .041	.071 ± .042	.080 ± .055	.071 ± .043	.033† ± .031	.061 ± .054	.066 ± .057	**.033 ± .032**	.044 ± .042
chess	.048 ± .037	.049 ± .038	**.038 ± .029**	.045 ± .035	.172 ± .171	.042 ± .033	.149 ± .047	.042 ± .033	.071 ± .047	.040† ± .031	.043 ± .033
letter	.021 ± .015	.021† ± .016	**.019 ± .015**	.021† ± .015	.449 ± .228	.021† ± .015	.227 ± .029	.022 ± .016	.064 ± .027	.020† ± .015	.028 ± .018
dry-bean	.026 ± .018	.023 ± .018	**.020 ± .015**	.022 ± .017	.055 ± .044	.024 ± .017	.032 ± .020	.021 ± .017	.030 ± .020	.021† ± .016	.022 ± .016
nursery	.104 ± .091	.041 ± .039	.038 ± .029	.063 ± .063	.079 ± .065	.083 ± .062	.097 ± .035	.039 ± .036	.029 ± .022	.025 ± .022	**.018 ± .016**
hand-digits	.009† ± .008	.010 ± .010	.009 ± .007	.009 ± .008	.073 ± .116	.010 ± .009	.100 ± .016	.009† ± .007	.013 ± .008	.009† ± .007	**.007 ± .005**
isolet	.022 ± .016	.022† ± .017	**.021 ± .015**	.023 ± .016	.577 ± .296	.024 ± .016	.331 ± .029	.026 ± .018	.067 ± .027	.022 ± .015	.034 ± .020
wine-quality	.208 ± .103	.154 ± .084	.090 ± .065	.089 ± .061	.157 ± .114	.088 ± .062	**.080 ± .057**	.115 ± .087	.085† ± .063	.102 ± .066	.093 ± .065
satellite	.052 ± .046	.034 ± .028	.027† ± .022	.045 ± .037	.099 ± .096	.041 ± .033	.070 ± .031	.031 ± .026	.037 ± .027	**.027 ± .021**	.027† ± .020
digits	.014 ± .011	.013† ± .011	.012† ± .010	.014 ± .012	.122 ± .205	**.012 ± .009**	.202 ± .026	.013† ± .010	.032 ± .017	.012† ± .010	.014 ± .010
page-block	.230 ± .148	.145 ± .114	.085 ± .067	.122 ± .095	.146 ± .114	.117 ± .084	**.053 ± .042**	.112 ± .099	.065 ± .051	.079 ± .066	.058 ± .046
waveform-v1	.035 ± .026	.042 ± .033	.033 ± .024	.034 ± .026	.070 ± .064	.036 ± .026	.131 ± .035	**.029 ± .022**	.044 ± .032	.029† ± .022	.029 ± .022
academic-success	.137 ± .116	.068 ± .057	.077 ± .054	.113 ± .096	.104 ± .077	.107 ± .083	.074 ± .050	.080 ± .065	.055 ± .045	.051† ± .039	**.049 ± .039**
abalone	.056 ± .041	.076 ± .051	.044† ± .032	.046 ± .037	.070 ± .052	.045 ± .034	.046 ± .033	.056 ± .042	.047 ± .037	.041 ± .031	**.039 ± .030**
molecular	.031 ± .026	.027 ± .023	.021† ± .016	.028 ± .022	.062 ± .055	.029 ± .021	.232 ± .027	.023 ± .018	.026 ± .021	.021† ± .017	**.021 ± .017**
image-seg	.023 ± .023	.029 ± .026	.020 ± .019	.024 ± .023	.050 ± .064	.028 ± .024	.059 ± .021	.019 ± .019	.021 ± .015	.019 ± .019	**.017 ± .016**
obesity	.023 ± .017	.023 ± .019	**.019 ± .015**	.023 ± .017	.086 ± .139	.023 ± .018	.225 ± .044	.022 ± .016	.054 ± .033	.021 ± .015	.028 ± .020
cmc	.064 ± .046	.065 ± .048	.053 ± .038	.064 ± .046	.080 ± .059	.068 ± .048	.063 ± .045	.046† ± .036	.056 ± .049	.046† ± .033	**.046 ± .034**
hcv	.044 ± .035	.058 ± .045	.045 ± .036	.044 ± .035	.060 ± .044	**.037 ± .029**	.072 ± .042	.059 ± .046	.051 ± .041	.046 ± .037	.048 ± .037
phishing	.100 ± .088	.069 ± .052	.050 ± .040	.089 ± .067	.101 ± .071	.079 ± .054	.035† ± .026	.048 ± .040	.041 ± .031	.040 ± .034	**.034 ± .027**
yeast	.068 ± .052	.070 ± .055	.069 ± .054	.068 ± .050	.121 ± .089	.071 ± .053	.060 ± .043	.045 ± .036	.053 ± .040	.043† ± .035	**.042 ± .034**
mhr	.062 ± .041	**.048 ± .035**	.103 ± .035	.063 ± .041	.100 ± .080	.064 ± .044	.102 ± .048	.057 ± .036	.069 ± .053	.061 ± .038	.064 ± .038
Average	.090 ± .121	.068 ± .090	.046 ± .046	.056 ± .062	.134 ± .171	.054 ± .055	.109 ± .086	.049 ± .057	.052 ± .045	.041 ± .043	**.040 ± .041**

15 classes), only a few seconds to estimate the classifier's accuracy. Moreover, one of our methods, S-LEAP, scales linearly with the number of classes, requiring only a tenth of a second in the worst-case scenario. Nevertheless, it is fair to mention that our best-performing method, O-LEAP, scales quadratically with the number of cases, meaning that it is more advisable to use S-LEAP when the number of classes is very large.

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/s10994-025-06878-y.

Acknowledgements Lorenzo Volpi's work was supported by project "Italian Strengthening of ESFRI RI RESILIENCE" (ITSERR), funded by the European Union under the NextGenerationEU funding scheme (CUP B53C22001770006). Alejandro Moreo's and Fabrizio Sebastiani's work was partially supported by project "Future Artificial Intelligence Research" (FAIR), project "Quantification under Dataset Shift" (QuaDaSh), and project "Strengthening the Italian RI for Social Mining and Big Data Analytics" (SoBigData.it), all funded by the European Union under the NextGenerationEU funding scheme (CUP B53D22000980006, CUP B53D23026250001, CUP B53C22001760006, respectively).

Author contributions All authors contributed to the main idea. L.V. conducted the experiments. A.M. and F.S. wrote the first draft of the manuscript. All authors contributed to the final writing and reviewed the manuscript.

Funding Open access funding provided by Consiglio Nazionale Delle Ricerche (CNR) within the CRUI-CARE Agreement.

Data availability No datasets were generated or analysed during the current study.

Declarations

Conflict of interest The authors have no relevant financial or non-financial interests to disclose.

Ethical approval Not applicable.

Consent for publication Not applicable.

Code availability The code to reproduce all the experiments is available at https://github.com/lorenzovolpi/LEAP.

Conflict of interest The authors declare no Conflict of interest.

References

Storkey, A. (2009). When training and test sets are different: Characterizing learning transfer. In Qui onero-Candela, J., Sugiyama, M., Schwaighofer, A., & Lawrence, N. D. (Eds.) *Dataset shift in machine learning* (pp. 3–28). The MIT Press, Cambridge.

Moreno-Torres, J. G., Raeder, T., Alaíz-Rodríguez, R., Chawla, N. V., & Herrera, F. (2012). A unifying view on dataset shift in classification. *Pattern Recognition*, *45*(1), 521–530. https://doi.org/10.1016/j.patcog.2011.06.019

Schölkopf, B., Janzing, D., Peters, J., Sgouritsa, E., Zhang, K., & Mooij, J.M. (2012). On causal and anticausal learning. In *Proceedings of the 29th international conference on machine learning (ICML 2012)*, Edinburgh, UK.

Patrone, P. N., & Kearsley, A. J. (2024). Minimizing uncertainty in prevalence estimates. *Statistics & Probability Letters, 205*, Article 109946. https://doi.org/10.1016/j.spl.2023.109946

Godau, P., Kalinowski, P., Christodoulou, E., Reinke, A., Tizabi, M., Ferrer, L., Jäger, P., & Maier-Hein, L. (2025). Navigating prevalence shifts in image analysis algorithm deployment. *Medical Image Analysis, 102*, Article 103504. https://doi.org/10.1016/J.MEDIA.2025.103504

Wolpert, D. H., & Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation, 1*(1), 67–82.

Garg, S., Balakrishnan, S., Lipton, Z.C., Neyshabur, B., & Sedghi, H. (2022). Leveraging unlabeled data to predict out-of-distribution performance. In *Proceedings of the 10th international conference on learning representations (ICLR 2022)*, Virtual event.

Forman, G. (2005). Counting positives accurately despite inaccurate classification. In *Proceedings of the 16th European conference on machine learning (ECML 2005)* (pp. 564–575). Porto, PT. https://doi.org/10.1007/11564096_55

González, P., Castaño, A., Chawla, N. V., & Coz, J. J. (2017). A review on quantification learning. *ACM Computing Surveys, 50*(5), 74–17440. https://doi.org/10.1145/3117807

Esuli, A., Fabris, A., Moreo, A., & Sebastiani, F. (2023). *Learning to quantify*. Springer, Cham. https://doi.org/10.1007/978-3-031-20467-8

Garg, S., Balakrishnan, S., Lipton, Z.C., Neyshabur, B., & Sedghi, H. (2022). Leveraging unlabeled data to predict out-of-distribution performance. arXiv:2201.04234 [cs.LG].

Guillory, D., Shankar, V., Ebrahimi, S., Darrell, T., & Schmidt, L. (2021). Predicting with confidence on unseen distributions. In *Proceedings of the IEEE/CVF international conference on computer vision (ICCV 2021)*, Montreal, CA (pp. 1134–1144).

Volpi, L., Moreo, A., & Sebastiani, F. (2024). A simple method for classifier accuracy prediction under prior probability shift. In *Proceedings of the 27th international conference on discovery science (DS 2024)*, Pisa, IT (pp. 267–283). https://doi.org/10.1007/978-3-031-78980-9_17

Xie, R., Wei, H., Feng, L., Cao, Y., & An, B. (2023). On the importance of feature separability in predicting out-of-distribution error. In *Proceedings of the 36th annual conference on neural information processing systems (NeurIPS 2023)*, New Orleans, US (pp. 27783–27800).

Deng, W., Suh, Y., Gould, S., & Zheng, L. (2023). Confidence and dispersity speak: Characterizing prediction matrix for unsupervised accuracy estimation. In *Proceedings of the 40th international conference on machine learning (ICML 2023)*, Honolulu, US (pp. 7658–7674).

Lu, Y., Qin, Y., Zhai, R., Shen, A., Chen, K., Wang, Z., Kolouri, S., Stepputtis, S., Campbell, J., & Sycara, K. P. (2023). Characterizing out-of-distribution error via optimal transport. In *Proceedings of the 36th annual conference on neural information processing systems (NeurIPS 2023)*, New Orleans, US.

Kivimäki, J., Bialek, J., Kuberski, W., & Nurminen, J. K. (2025). Performance estimation in binary classification using calibrated confidence. arXiv:2505.05295 [cs.LG]

Elsahar, H., & Gall , M. (2019). To annotate or not? Predicting performance drop under domain shift. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP 2019)*, Hong Kong, CN (pp. 2163–2173). https://doi.org/10.18653/v1/D19-1222

Chen, M.F., Goel, K., Sohoni, N.S., Poms, F., Fatahalian, K., & Ré, C. (2021). Mandoline: Model evaluation under distribution shift. In *Proceedings of the 38th international conference on machine learning (ICML 2021)*, virtual event (pp. 1617–1629).

Chen, J., Liu, F., Avci, B., Wu, X., Liang, Y., & Jha, S. (2021). Detecting errors and estimating accuracy on unlabeled data with self-training ensembles. In *Proceedings of the 35th conference on neural information processing systems (NeurIPS 2021)*, virtual event (pp. 14980–14992).

Jiang, Y., Nagarajan, V., Baek, C., & Kolter, J.Z. (2022). Assessing generalization of SGD via disagreement. In *Proceedings of the international conference on learning representations (ICLR 2022)*, virtual event.

Sugiyama, M., Nakajima, S., Kashima, H., Buenau, P., & Kawanabe, M. (2007). Direct importance estimation with model selection and its application to covariate shift adaptation. In *Proceedings of the 21st conference on advances in neural information processing systems (NIPS 2007)*, Vancouver, CA (pp. 1433–1440).

Ovadia, Y., Fertig, E., Ren, J., Nado, Z., Sculley, D., Nowozin, S., Dillon, J., Lakshminarayanan, B., & Snoek, J. (2019). Can you trust your model's uncertainty? Evaluating predictive uncertainty under dataset shift. In *Proceedings of the 32nd annual conference on neural information processing systems (NeurIPS 2019)*, Vancouver, CA (pp. 13969–13980).

Zhang, K., Sch lkopf, B., Muandet, K., & Wang, Z. (2013). Domain adaptation under target and conditional shift. In *Proceedings of the 30th international conference on machine learning (ICML 2013)* (pp. 819–827). Atlanta, US.

Lipton, Z.C., Wang, Y., & Smola, A.J. (2018). Detecting and correcting for label shift with black box predictors. In *Proceedings of the 35th international conference on machine learning (ICML 2018)*, Stockholm, SE (pp. 3128–3136)

Tasche, D. (2024). Comments on Friedman's method for class distribution estimation. arXiv:2405.16666 [cs.LG]

Forman, G. (2008). Quantifying counts and costs via classification. *Data Mining and Knowledge Discovery, 17*(2), 164–206. https://doi.org/10.1007/s10618-008-0097-y

Bella, A., Ferri, C., Hern ndez-Orallo, J., & Ram rez-Quintana, M.J. (2010). Quantification via probability estimators. In *Proceedings of the 11th IEEE international conference on data mining (ICDM 2010)*, Sydney, AU (pp. 737–742). https://doi.org/10.1109/icdm.2010.75

Bunse, M. (2022). On multi-class extensions of adjusted classify and count. In *Proceedings of the 2nd international workshop on learning to quantify (LQ 2022)*, Grenoble, IT (pp. 43–50).

Moreo, A., Gonz lez, P., del & Coz, J.J. (2025). Kernel density estimation for multiclass quantification. *Machine Learning* **114**(4). https://doi.org/10.1007/s10994-024-06726-5

Smith, N.A., & Tromble, R.W. (2004). *Sampling uniformly from the unit simplex* [Technical report], Johns Hopkins University. https://www.cs.cmu.edu/~nasmith/papers/smith+tromble.tr04.pdf

Kelly, M., Longjohn, R., & Nottingham, K. The UCI machine learning repository. https://archive.ics.uci.edu

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research, 12*, 2825–2830.

Popordanoska, T., Radevski, G., Tuytelaars, T., & Blaschko, M. (2024) Lascal: Label-shift calibration without target labels. In *Proceedings of the 37th annual conference on neural information processing systems (NeurIPS 2024)*, Vancouver, CA (pp. 65386–65414).

Moreo, A., Esuli, A., & Sebastiani, F. (2021). QuaPy: A Python-based framework for quantification. In *Proceedings of the 30th ACM international conference on knowledge management (CIKM 2021)*, Gold Coast, AU (pp. 4534–4543). https://doi.org/10.1145/3459637.3482015

Diamond, S., & Boyd, S. (2016). CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research, 17*(83), 1–5.