A Simple Method for Classifier Accuracy Prediction under Prior Probability Shift

Lorenzo Volpi, Alejandro Moreo, and Fabrizio Sebastiani

Istituto di Scienza e Tecnologie dell'Informazione
Consiglio Nazionale delle Ricerche
56124 Pisa, Italy
firstname.lastname@isti.cnr.it

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. In this paper we propose a CAP method specifically designed to address *prior probability shift* (PPS), an instance of dataset shift in which the training and test distributions are characterized by different class priors. By solving a system of n^2 independent linear equations, with n the number of classes, our method estimates the n^2 entries of the contingency table of the test data, and thus allows estimating any specific evaluation measure. Since a key step in this method involves predicting the class priors of the test data, we further observe a connection between our method and the field of "learning to quantify". Our experiments show that, when combined with state-of-the-art quantification techniques, under PPS our method tends to outperform existing CAP methods.

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

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 classifiers we deploy are effective. The accuracy of a classifier (where "accuracy" is broadly understood as any effectiveness metric – e.g., vanilla accuracy, or F_1) 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 expected to follow the same distribution. Such an assumption is often violated in many real 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, or

misleading its optimization process. CAP is thus an open problem when the IID assumption is not verified, i.e., when *dataset shift* [21] is present.

One specific type of dataset shift is *prior probability shift* (PPS), which is defined as the type of dataset shift in which (a) the class priors can change between the training distribution and the test distribution and (b) the class-conditional distribution of the covariates is stationary.

In this paper we propose a simple yet effective method for CAP which is specifically tailored to address PPS; we dub this method LEAP (for *Linear Equation -based Accuracy Prediction*). Given that the vast majority of evaluation measures are computed on a contingency table that relates the true labels to the predicted labels, we approach CAP by treating the entries of this table as unknowns in a system of linear equations. In a classification problem with n classes, there are n^2 such unknowns; our method involves a set of n^2 independent linear equations, whose solution results in the prediction of the n^2 contingency table entries. From these, the accuracy of the classifier on the test data can be estimated, according to one or multiple accuracy measures at the same time. This is in contrast to previously proposed methods, that estimate accuracy according to a single measure and need to be retrained if the estimate of a different measure is needed.

We further observe 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* [5, 8, 11], the supervised learning task of predicting the distribution of the class labels in the test data. Through several experiments we show that, when equipped with a state-of-the-art quantification technique, our CAP method often outperforms other competing methods in scenarios characterized by PPS.

The rest of the paper is structured as follows. Section 2 reviews the related literature on classifier accuracy prediction. In Section 3 we define the notation and the concepts we use in the rest of the paper. Section 4 is devoted to explaining our novel method, the set of equations on which it is based, and its connections with quantification learning. We present our experimental results in Section 5, while Section 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. *Average Thresholded Confidence* (ATC) [10] estimates model accuracy by evaluating the expected proportion of test items for which the confidence score of the model exceeds a threshold learned on the training set. The authors propose two variants for computing such scores, ATC-MC (that relies on maximum confidence) and ATC-NE (that instead relies on negative entropy). *Difference of Confidence* (DoC) [12] also leverages the confidence a model shows in its predictions for training a regressor that estimates the accuracy of the original model on the test set. *Mandoline* [4] belongs instead to the family of "importance-weighting methods" [22], and relies on user-defined so-called "slicing" functions to create

common representations and compare training data with test data. Chen et al. [3] 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) [13] 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.

A common trait of all these methods, and an important difference between them and our proposed method, is that the former are trained to estimate one specific evaluation measure (typically: vanilla accuracy). In cases where more than one evaluation measure is needed, these methods should be retrained from scratch.

3 Background

3.1 Notation

Throughout this paper we use the following notation. By $\mathcal{X} = \mathbb{R}^d$ and $\mathcal{Y} = \{1, \ldots, n\}$ we denote the input space (vectors of covariates) and the output space (classes), respectively; in binary problems we write instead $\mathcal{Y} = \{0, 1\}$. By $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^{m}$ we denote a generic labelled set consisting of pairs $\mathbf{x}_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$, that we use for training or testing our models.

A (crisp) classifier $h : \mathcal{X} \to \mathcal{Y}$ is a function mapping vectors of covariates into classes. A probabilistic classifier has instead the form $h : \mathcal{X} \to \Delta^{n-1}$, i.e., it maps vectors of covariates into vectors of posterior probabilities lying in the probability simplex $\Delta^{n-1} = \{(p_1, \ldots, p_n) : p_i \geq 0, \sum_{i=1}^{n} p_i = 1\}$. From a probabilistic classifier one can easily obtain a crisp classifier by returning the class corresponding to the largest posterior probability.

3.2 Prior Probability Shift

Dataset shift is defined as the situation in which the training set is drawn from a distribution P and the test set is drawn from another distribution Q such that

$$P(X, Y) \neq Q(X, Y) \tag{1}$$

Prior probability shift is a type of dataset shift (to be found in *anti-causal learning* [19], i.e., the class of learning problems in which $\mathcal{Y}$ represents the phenomenon to be predicted and $\mathcal{X}$ represents *symptoms* of this phenomenon) defined as the case in which

$$\begin{aligned} P(Y) &\neq Q(Y) \\ P(X|Y) &= Q(X|Y) \end{aligned} \tag{2}$$

i.e., where the prevalence of the classes can change between the training distribution and the test distribution while the class-conditional distribution of the

covariates does not change. PPS is sometimes called *target shift* [24] or *label shift* [15]. For all X-measurable functions $f : \mathcal{X} \to \mathbb{R}$, it also follows that from $P(X|Y) = Q(X|Y)$ it also follows that $P(Z|Y) = Q(Z|Y)$ with $Z = f(X)$ [23]. In particular, if we take $f = h$, it follows that the distribution of the class-conditional predictions $\hat{Y}$ issued by the classifier is stationary between the training distribution P and the test distribution Q, i.e., $P(\hat{Y}|Y) = Q(\hat{Y}|Y)$.

3.3 Problem Setting

We assume a classifier h trained on a set L of labelled items which may no longer be available. We also assume a validation set V of labelled items on which we train classifier accuracy predictor. 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. We assume L and V are drawn from the same distribution P, while U is instead drawn from a different distribution Q, such that P and Q are related by PPS. Note that h is assumed already trained, and 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 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) with those of the random variable $\hat{Y}$ (which denotes the predicted labels). A contingency table is a matrix where entry (i, j) denotes the number of items whose true class is i and whose predicted class is 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 items, i.e., we will consider entry (i, j) of a contingency table as the *fraction* of items whose true class is i and whose predicted class is j; in other words, 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 $V = \{(\mathbf{x}_i, y_i)\}_{i=1}^{m}$ drawn from the training distribution P. Applying h to V brings about a transformation $V' = \{(\hat{y}_i, y_i)\}_{i=1}^{m}$, with $\hat{y}_i = h(\mathbf{x}_i)$, from which we can obtain a contingency table $\mathbf{V}$ with entries

$$v_{ij} = \frac{|\{(\hat{y}_k, y_k) \in V' \ : \ y_k = i \ \wedge \ \hat{y}_k = j\}|}{|V'|} \tag{3}$$

where symbol v reminds us that the classified items are taken from the validation set V. Note that $v_{ij} \approx P(\hat{Y} = j, Y = i)$ when $|V|$ is large enough.

As noted in the introduction, the contingency table $\mathbf{V}$ cannot be used to reliably estimate the accuracy of h on the test data U if V and U are drawn from different distributions (P and Q) related by PPS. Ideally, to estimate the accuracy of h on U we would like to directly estimate the entries u_{ij} of the

contingency table $\mathbf{U}$ that derives from U, but this is not trivial, since on U we can observe the predicted labels (by simply applying h to the items 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

$$\mathrm{Acc}(h,U) = \frac{\sum_{i=1}^{n} u_{ii}}{\sum_{i=1}^{n}\sum_{j=1}^{n} u_{ij}} = \sum_{i=1}^{n} u_{ii} \tag{4}$$

In the binary case we will indicate fractions v_{11} and u_{11} as TP_V and TP_U, respectively, where TP stands for "true positives" and the V and U subscripts indicate from which set of items (the validation set V or the test set U) these fractions originate;[1] analogously, FP, FN, TN, will stand for "false positives", "false negatives", "true negatives", respectively. We will also use the shorthands

$$\mathrm{tpr} = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}} \quad \mathrm{fpr} = \frac{\mathrm{FP}}{\mathrm{FP}+\mathrm{TN}} \tag{5}$$

to refer to the "true positive rate" and the "false positive rate" of classifier h, and we will write tpr_V, tpr_U, fpr_V, fpr_U to indicate whether these ratios are computed on the contingency tables for sets V or U.

4.1 LEAP: A System of n^2 Linear Equations for CAP

We approach CAP by treating the entries of contingency table $\mathbf{U}$ as unknowns in a system of linear equations. In a classification problem with n classes, there are n^2 such unknowns; LEAP involves a set of n^2 independent linear equations, whose solution results in the prediction of the n^2 entries of the contingency table. From these, the accuracy of classifier h on the test data U can be estimated, according to one or more accuracy measures. In the following we derive LEAP for the general multiclass case ($n > 2$), and discuss how it looks like in the binary case ($n = 2$), in which our 4 unknowns are TP_U, FP_U, FN_U, and TN_U.

The first equation is trivial: the entries of the contingency table are probabilities of disjoint events that cover the entire event space, and they thus must sum to one, i.e.,

$$\sum_{i=1}^{n}\sum_{j=1}^{n} u_{ij} = 1 \tag{6}$$

In the binary case, this corresponds to equation

$$\mathrm{TP}_U + \mathrm{FP}_U + \mathrm{FN}_U + \mathrm{TN}_U = 1 \tag{7}$$

[1] The term "true positives" is typically used to refer to the *number* (and not the *prevalence*) of items which have correctly been predicted to belong to class 1; however, in this paper it is useful for us to always think in terms of prevalence values.

The second batch of equations relates the sum of the counts by columns to the fraction of predicted positives for each class, that is $\sum_{i=1}^{n} Q(\hat{Y}, Y = i) = Q(\hat{Y})$. Note that $Q(\hat{Y})$ is observed and can be obtained by simply classifying and counting the fraction of instances attributed to each class by h. Let us define a vector of observed normalized counts (fractions) $\mathbf{c} = (c_1, \ldots, c_n) \in \Delta^{n-1}$ where

$$c_i = \frac{1}{|U|} \sum_{\mathbf{x} \in U} \mathbb{1}[h(\mathbf{x}) = i] \tag{8}$$

This adds $(n-1)$ independent equations (note that the n-th one is constrained) which correspond to adding

$$\sum_{i=1}^{n} u_{ij} = c_j, \quad \text{for } j \in \{1, \ldots, n-1\} \tag{9}$$

In the binary case, this amounts to adding one equation imposing that the fraction of the instances classified as positive (c_1) must coincide with the sum of FP_U and TP_U, that is,

$$\mathrm{FP}_U + \mathrm{TP}_U = c_1 \tag{10}$$

The third batch of equations imposes that the class-conditional rates should remain stationary across the training and test distributions. This follows from the PPS assumptions (Equation 2) and from the fact (noted at the end of Section 3.2) that $P(X|Y) = Q(X|Y)$ implies $P(\hat{Y}|Y) = Q(\hat{Y}|Y)$. Let us define a matrix of class-conditional rates $\mathbf{R}$ with entries $r_{ij} = \frac{v_{ij}}{\sum_{k=1}^{n} v_{ik}}$ that we compute on the validation set V. This allows us to add the $(n-1)^2$ independent equations

$$\frac{u_{ij}}{\sum_{k=1}^{n} u_{ik}} = r_{ij}, \quad \text{for } i, j \in \{1, \ldots, n-1\} \tag{11}$$

which can be more conveniently written as

$$u_{i1} r_{ij} + \cdots + u_{ij}\,(r_{ij} - 1) + \cdots + u_{in} r_{ij} = 0, \quad \text{for } i, j \in \{1, \ldots, n-1\}$$

In the binary case, this comes down to taking $r_{11} \equiv \mathrm{tpr}_V$ (as computed on the validation set V) as an estimate of tpr_U. That is, we add equations

$$\frac{\mathrm{TP}_U}{\mathrm{TP}_U + \mathrm{FN}_U} = \mathrm{tpr}_V \tag{12}$$

which can be more conveniently written as

$$\mathrm{FN}_U \cdot \mathrm{tpr}_V + \mathrm{TP}_U \cdot (\mathrm{tpr}_V - 1) = 0 \tag{13}$$

The fourth batch of equations is more complicated since it relates the sum of the entry values by rows to the class prevalence values in the test set, which we do not know; that is, $\sum_{j=1}^{n} Q(\hat{Y} = j, Y) = Q(Y)$. However, it turns out that

the class prevalence values can be estimated from the training counts with the aid of the PPS assumptions (Equation 2). This is better shown in the binary case by noting that

$$\begin{aligned} Q(\hat{Y}=1) &= Q(\hat{Y}=1|Y=1)Q(Y=1) + Q(\hat{Y}=1|Y=0)Q(Y=0) \\ &= P(\hat{Y}=1|Y=1)Q(Y=1) + P(\hat{Y}=1|Y=0)Q(Y=0) \end{aligned} \tag{14}$$

where the first step derives by the law of total probability and the second step derives by the already observed fact that, by the PPS assumptions, $P(\hat{Y}|Y) = Q(\hat{Y}|Y)$. By taking into account the fact that $Q(Y=0) = 1 - Q(Y=1)$, and $P(\hat{Y}=1|Y=1) \approx \text{tpr}_V$ and $P(\hat{Y}=1|Y=0) \approx \text{fpr}_V$, and the fact that $Q(\hat{Y}=1)$ is observed (we called its estimate c_1), it then follows that

$$Q(Y=1) \approx \frac{c_1 - \text{fpr}_V}{\text{tpr}_V - \text{fpr}_V} \tag{15}$$

The above solution is well-known in the literature, but we will come back to this in the next section. Since $c_1 = \text{FP}_U + \text{TP}_U$ and $Q(Y=1) \approx \text{FN}_U + \text{TP}_U$, Equation 15 leads to the following equation in the binary case

$$\text{FP}_U + \text{FN}_U \cdot (\text{fpr}_V - \text{tpr}_V) + \text{TP}_U \cdot (1 + \text{fpr}_V - \text{tpr}_V) = \text{fpr}_V \tag{16}$$

In the multiclass case, there are $(n-1)$ such independent equations. The derivation is slightly more complicated, since it entails writing Equation 14 as a system of linear equations, one per class, that involves the class-conditional rates $P(\hat{Y}=j|Y=i)$. This problem can be written in matrix form as $\mathbf{c} = \mathbf{R}^\top \mathbf{q}$, where $\mathbf{c} \in \Delta^{n-1}$ is our vector of (normalized) counts, $\mathbf{R}$ is our matrix of class-conditional rates, as before, and $\mathbf{q} = (q_1, \ldots, q_n) \in \Delta^{n-1}$ is the sought class distribution on the test data. The solution thus comes down to solving the system as $\mathbf{q} = (\mathbf{R}^\top)^{-1}\mathbf{c}$, then taking $(n-1)$ test prevalence values from $\mathbf{q}$ and adding the corresponding equations

$$q_i = \sum_{j=1}^{n} u_{ij}, \quad \text{for } i \in \{1, \ldots, n-1\} \tag{17}$$

This batch of equations adds $(n-1)$ new independent equations. Adding the first equation (Equation 6), the $(n-1)$ equations from the second batch (Equation 9), the $(n-1)^2$ equations from the third batch (Equation 11), and the $(n-1)$ equations from the second batch (Equation 17), we obtain a system of exactly $(1+(n-1)+(n-1)^2+(n-1)) = n^2$ independent equations with n^2 unknowns.

Finally, note that the system can be written in matrix form as $\mathbf{AX} = \mathbf{b}$, with $\mathbf{X}$ our unknowns. In degenerate cases in which the PPS assumptions do not hold, or in which the estimated class prevalence values deviate much from the true value, the system of equations might not be solvable via matrix inversion ($\mathbf{X} = \mathbf{A}^{-1}\mathbf{b}$), or the solution might be unfeasible (e.g., some of the $\hat{u}_{ij}$ might not be in $[0,1]$). In cases like this, we instead resort to solving the constrained

problem $\mathbf{X}^* = \arg\min_{\mathbf{X}\in\mathcal{A}^{n-1}} ||\mathbf{AX}-\mathbf{b}||_2$. Either way, the solution to the system is something modern software packages solve very quickly.

Recap for the binary case: If we write $q_1 = \frac{c_1 - \text{fpr}_V}{\text{tpr}_V - \text{fpr}_V}$ (Equation 15), in the binary case the system of four equations is:

$$\begin{cases} \text{TN}_U + \text{FN}_U + \text{FP}_U + \text{TP}_U = 1 \\ \text{FP}_U + \text{TP}_U = c_1 \\ \text{TP}_U \cdot (\text{tpr}_V - 1) + \text{FN}_U \cdot \text{tpr}_V = 0 \\ \text{FN}_U + \text{TP}_U = q_1 \end{cases}$$

which has the solution

$$\begin{aligned} \text{TN}_U &= 1 - c_1 + q_1(\text{tpr}_V - 1) \\ \text{FN}_U &= q_1(1 - \text{tpr}_V) \\ \text{FP}_U &= c_1 - q_1 \cdot \text{tpr}_V \\ \text{TP}_U &= q_1 \cdot \text{tpr}_V \end{aligned} \tag{18}$$

We can now derive the closed form of many well-known evaluation measures under PPS. For example, vanilla accuracy and F_1 as computed on the test contingency table $\mathbf{U}$ (Equations 19 and 22) can be expressed in compact form with the aid of $\mathbf{c}$ and $\mathbf{q}$ (Equations 20 and 23), or expanded as functions that use only $\mathbf{c}$ along with counts from the validation contingency table $\mathbf{V}$ (Equations 21 and 24), as

$$\text{Acc}(h, U) = \frac{\text{TP}_U + \text{TN}_U}{\text{TP}_U + \text{TN}_U + \text{FN}_U + \text{FP}_U} \tag{19}$$

$$= 1 + q_1(2\text{tpr}_V - 1) - c_1 \tag{20}$$

$$= \frac{((\text{TN}_V + \text{FP}_V) \cdot c_1 - \text{FP}_V)(\text{TP}_V - \text{FN}_V)}{\text{TN}_V\text{TP}_V - \text{FP}_V\text{FN}_V} + 1 - c_1 \tag{21}$$

$$F_1(h, U) = \frac{2 \cdot \text{TP}_U}{2 \cdot \text{TP}_U + \text{FP}_U + \text{FN}_U} \tag{22}$$

$$= \frac{2q_1 \cdot \text{tpr}_V}{c_1 + q_1} \tag{23}$$

$$= \frac{2 \cdot \text{TP}_V(c_1 \cdot \text{TN}_V - c_0 \cdot \text{FP}_V)}{c_1 \cdot \text{TN}_V(2 \cdot \text{TP}_V + \text{FN}_V) - \text{FP}_V(\text{FN}_V + \text{TP}_V \cdot c_0)} \tag{24}$$

Note that the compact formulas above (Equations 20 and 23) are generic, and can be used also in cases in which the class prevalence $\mathbf{q}$ is obtained by means of other techniques. We discuss this topic in the following section.

4.2 Enhancing LEAP via Quantification

The fourth batch of equations from the previous section requires estimating the class prevalence values q_i of the test set. The solution we have shown in

Equations 14 and 15 are actually borrowed from a well-known method in the quantification literature called Adjusted Classify & Count (ACC) [1, 8, 9]; the method is also dubbed Black-Box Shift Estimator (BBSE) in [15]. This method is of particular interest for our derivation since it only involves components (as the tpr and fpr) which can be derived from the validation contingency table. ACC was originally devised for binary quantification but it was later extended to the multiclass case by Firat [7] and later improved by Fernandes Vaz et al. [6] and Bunse [2]. The implementation we use in this paper does indeed adhere to the improvements brought about by [2], which comes down to better handling the cases in which the matrix of the class-conditional rates $\mathbf{R}^\top$ is not invertible.

ACC is by no means the only method available for estimating class prevalence, nor the most sophisticated one. There is an entire body of literature devoted to devising better ways for predicting class prevalence [5, 11]. In this section, we propose an enhanced version of our LEAP method that defers the fourth batch of equations to more sophisticated quantification algorithms.

More formally, a quantifier is a function $\lambda : \mathbb{N}^{\mathcal{X}} \to \Delta^{n-1}$ mapping bags (or multi-sets) of instances from the input space $\mathcal{X}$ to the probability simplex. We focus our attention to the so-called distribution matching approaches, that take a permutation-invariant function Φ to represent samples, and solve for $\mathbf{q}$ (the sought test prevalence vector) the equation

$$\mathbf{t} = \mathbf{z}^\top \mathbf{q} \tag{25}$$

where $\mathbf{t} = \Phi(U)$ is the representation of test sample U and $\mathbf{z} = [\Phi(V_1), \ldots, \Phi(V_n)]$ contains the class-wise representations of validation sets $V_i = \{\mathbf{x}_k : (\mathbf{x}_k, y_k) \in V, y_k = i\}$. Specifically, we adopt the KDEy-ML variant proposed in [17] (hereafter simply called KDEy) which relies on kernel density estimation for representing the samples as density functions in the probability simplex. KDEy then solves Equation 25 via maximum likelihood, as the minimization problem

$$\mathbf{q}^* = \underset{\mathbf{q} \in \Delta^{n-1}}{\arg\min}\, \mathcal{D}_{\mathrm{KL}}(\mathbf{t} || \mathbf{z}^\top \mathbf{q}) \tag{26}$$

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 $\mathbf{q}$. We call this extension $\mathrm{LEAP}_{\mathrm{KDEy}}$.

5 Experiments

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

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 experiments 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 Equation 4). Aside from the fact that it is one of the most important measures of classifier effectiveness, the reason why we choose Acc is that some of our baselines only work for this evaluation measure. Therefore, for the sake of fair experimental comparison, we adopt Acc as our only target measure.

The **experimental protocol** we adopt is described as follows. Given a labelled collection $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^{m}$, 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, using stratification. We train and test all competing methods on the same partitions. We then extract, from U, 1000 test samples $U_1, U_2, \ldots, U_{1000}$ of $|U_i| = 100$ elements each. For each such extracted test sample, we first draw, uniformly at random, a vector of prevalence values from the unit simplex Δ^{n-1} (using the Kraemer sampling algorithm [20]), and then draw bags of instances with replacement from the corresponding classes in U so as to satisfy the required prevalence distribution. 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 all 1000 tests. Note that L, V (as well as the original U) come from the same distribution P, while the test samples U_i extracted from U instead come from a different distribution Q which is related to P via PPS (since the extraction protocol indeed simulates PPS).

Although our methods are natively multiclass, in this paper we mainly concentrate on **binary classification**, and defer a more in-depth experimentation on multiclass problems to future work. As our **datasets**, we use the 29 datasets from the UCI machine learning repository [14] used in [17]. The number of instances vary from a minimum of 150 instances (iris.2, iris.3) to a maximum of 5,473 (pageblocks.5), while the number of features vary 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 generating the **classifiers** we consider four learning algorithms. In particular, we report results for the case in which h is learned via logistic regression (LR), kNN with $k = 10$, support vector machines (SVM) with the radial basis function kernel, and multilayer perceptron (MLP) with hidden sizes of 100 and 15 neurons and the ReLU activation function. In all cases, we rely on the implementations provided in scikit-learn [18], leaving the rest of the hyperparameters at their default values.

As the **baseline methods** against which we test the performance of our LEAP methods, we use DoC [12] and ATC-MC (hereafter ATC) [10] (see Sec-

tion 2). We do not report results for Mandoline [4] since the results we have obtained for it were not competitive with the rest of the methods.[2] We also report the results of Naive, a method that assumes that the training and test data are IID, and report the vanilla accuracy score obtained in the contingency table generated in the validation set V by h. For ATC we borrow the implementation provided by the authors, while the rest of the methods are implemented by us.

In order to assess the extent to which the quality of the surrogate quantifier impacts on the estimated accuracy, we include two additional baselines. The first one acts as a lower-bound baseline, in which the quantifier is taken to be the simplest possible method: the Classify and Count (CC); we denote this variant LEAP_{CC}. The second one, instead, acts as an upper-bound baseline, in which the quantifier is replaced by an oracle that always returns, as the prevalence values of the test set, the true class proportions; we denote this variant LEAP_{Φ}. For the sake of consistency, we will hereafter denote LEAP_{ACC} to our "vanilla" LEAP method. As recalled from Section 4.2, we denote $\text{LEAP}_{\text{KDEy}}$ to our enhanced variant in which we employ KDEy as our quantifier.

For the implementation of the quantification algorithms (ACC and KDEy) we rely on the QuaPy[3] package [16]. The code to reproduce all our experiments is available on GitHub.[4]

5.2 Results

Table 1 shows the results we have obtained in our experiments in terms of AE (lower is better). Overall, our results show that $\text{LEAP}_{\text{KDEy}}$ obtains the best results (averaged across all datasets) for all four classifiers. $\text{LEAP}_{\text{KDEy}}$ obtains 14 best results for LR, 7 best results for kNN, 14 best results for SVM, and 7 best results for MLP, out of 29 datasets. Conversely, the vanilla variant LEAP fares better than ATC on average for LR and kNN, and better than DoC on average for SVM, but it only manages to beat both methods simultaneously for MLP. The superiority of $\text{LEAP}_{\text{KDEy}}$ over LEAP_{ACC} speaks in favor of replacing the basic quantifier ACC with a more sophisticated quantification method like KDEy for predicting the test class prevalence values.

However, these results also show that all methods (including our proposed variants) sometimes fail loudly in certain cases where other methods would instead fare reasonably well. This is the case of, e.g., (sonar, LR) in which DoC yields the smallest error (0.070) while $\text{LEAP}_{\text{KDEy}}$ obtains the worst score (0.113). Figure 1 shows a selection of prototypical diagonal plots for LR, including sonar (the full set of plots for all datasets and all classifiers can be consulted online[5]).

[2] We ran experiments using the code provided by the authors. The likely reason why the performance of Mandoline is not competitive with other baselines in our experiments 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.

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

[4] `https://github.com/lorenzovolpi/LEAP`

[5] `https://github.com/lorenzovolpi/LEAP/tree/main/plots`

Table 1. Values of AE obtained in our experiments for different CAP methods. **Boldface** indicates the best method for a given (dataset, classifier) pair. Superscripts † and ‡ denote the methods (if any) whose scores are not statistically significantly different from the best one according to a Wilcoxon signed-rank test at different confidence values: symbol † indicates $0.001 < p\text{-value} < 0.01$ while symbol ‡ indicates $0.01 \leq p\text{-value}$. Cells are colour-coded so as to facilitate readability and allow for quick comparisons across results. Intense green highlights the best result while intense red highlights the worst one; milder colours are used in the obvious way.

	LR					k-NN					SVM					MLP				
	Naive	ATC	DoC	LEAP_{ACC}	$\text{LEAP}_{\text{KDEy}}$	Naive	ATC	DoC	LEAP_{ACC}	$\text{LEAP}_{\text{KDEy}}$	Naive	ATC	DoC	LEAP_{ACC}	$\text{LEAP}_{\text{KDEy}}$	Naive	ATC	DoC	LEAP_{ACC}	$\text{LEAP}_{\text{KDEy}}$
balance.1	**.020**	.034	.026	.023	.025	.033	.032	.056	**.018**	.020	**.015**	.023	.016	.017	.026	.021	**.012**	.016	.020	.021
balance.3	.024	.048	.024	.016‡	**.015**	.019	.076	.019	**.015**	.029	.037	.039	.036‡	**.035**	.050	.033	.019†	.029	.031	**.018**
breast-cancer	.030	.026	.016	.017	**.013**	.019	.035	**.012**	.013	.014	.014	.012‡	.015	.016	**.011**	.030	.034	.020	**.016**	.019
cmc.1	.072‡	.080	.106	.069	**.067**	.130	.236	.106	.071	**.061**	.160	.128	.181	.105	**.086**	.077	.094	.081	**.036**	.036‡
cmc.2	.267	.183	.102	.124	**.074**	.267	.336	**.072**	.188	.074‡	.320	.235	**.142**	.252	.157	.226	.134	**.043**	.078	.080
cmc.3	.184	.167	.098	.174	**.086**	.157	.260	**.073**	.098	.078‡	.270	.192	**.172**	.252	.205	.104	.061	.059	.065	**.051**
ctg.1	.045	.041	.030	.033	**.019**	.081	.039	.035	.032	**.026**	.155	.070	.085	.050	**.026**	.041	.038	.024	**.016**	.019
ctg.2	.087	.037	**.021**	.039	.061	.165	.194	**.063**	.132	.116	.376	.372	.383	.252	**.019**	.074	.022‡	**.021**	.022	.026
ctg.3	.091	.036	.072	**.033**	.049	.200	.192	**.027**	.057	.038	.315	.239	.166	.144	**.050**	.078	.032	**.024**	.026	.041
german	.160	.124	.050	.128	**.039**	.226	.368	**.122**	.240	.171	.279	.211	.248	.200	**.113**	.099	.060	**.038**	.042	.053
haberman	.224	.162	.154	.183	**.120**	.253	.175	.363	.386	**.101**	.298	.158	**.090**	.252	.236	.174	**.089**	.379	.271	.210
ionosphere	**.063**	.064‡	.109	.084	.096	.118	**.037**	.074	.095	.092	.046	**.022**	.028	.076	.094	**.045**	.059	.198	.099	.119
iris.2	.203	**.065**	.102	.135	.076	.030	**.018**	.024	.037	.036	.056	.051	.084	.058	**.041**	.022	.018	**.016**	.026	.037
iris.3	**.030**	.051	.065	.075	.118	.113	.049	**.034**	.082	.046	.037	.083	**.021**	.027	.032	.035	.035	.035	.035	**.034**
mammographic	.064	.087	.074	.062	**.046**	.068	.148	.052	.070	**.043**	.060	.035‡	.062	.059	**.034**	.054	.069	.082	**.040**	.057
pageblocks.5	.368	.293	.114	.102	**.068**	.477	.378	**.101**	.496	.177	.449	.223	**.137**	.170	.160	.367	.266	**.089**	.097‡	.092‡
semeion	.125	.067	.037	.036	**.031**	.170	.060	**.024**	.073	.037	.160	.113	.073	.045	**.029**	.102	**.023**	.043	.040	.030
sonar	.113	.109	**.070**	.108	.113	.172	**.112**	.298	.266	.234	.155	.108	**.082**	.210	.219	.083	.112	.104	**.060**	.073
spambase	.022	.026	**.020**	.021†	.023	.083	.054	**.038**	.043	.042†	.139	.086	.110	.062	**.054**	.048	.040	.034	.030‡	**.029**
spectf	.157	**.064**	.139	.145	.102	.142‡	**.135**	.174	.236	.157	.340	.280	.219	.496	**.107**	.124	.175	.084	.055	**.046**
tictactoe	.018	.018	.013	**.008**	.008†	.079	.074	.061	**.021**	.024	.022	.023	.024	**.010**	.011‡	.026	.016	.024	.016	**.013**
transfusion	.269	.194	.081	.122	**.072**	.286	.357	**.091**	.201	.099‡	.289	.241†	**.232**	.286	.296	.290	.176	.128	.197	**.087**
wdbc	**.021**	.040	.037	.024‡	.027	.045	.021	.024	.021	**.019**	.058	.048	.055	.023	**.021**	.135	.088	.101	**.040**	.065
wine.1	.039	.056	**.022**	.061	.084	.041	.069	**.036**	.078	.115	.030	.069	**.025**	.027‡	.042	**.027**	.032†	.034	.052	.059
wine.2	.023	**.020**	.023	.027	.033	.105	.311	.234	**.051**	.056	.082	.093	.160	.065	**.056**	.046	.035‡	**.035**	.039	.044
wine.3	.021	.028	.013	.012	**.005**	.093	.095	.083	.064‡	**.058**	.304	.320	**.116**	.252	.251	.013	.013	.013	**.013**	.013
wine-q-red	.049	.048	.045‡	.049	**.043**	.045	.068	.046	**.034**	.043	.135	.111	.116	**.078**	.098	.033	.040	.034	**.030**	.033
wine-q-white	.130	.084	**.051**	.058	.053‡	.100	.269	.055	.057	**.050**	.282	.221	**.156**	.426	.367	.035	.059	.037	**.032**	.035
yeast	.220	.131	**.085**	.209	.123	.123	.095	**.062**	.123	.097	.168	.107	.111	.106	**.084**	.089	.059	**.036**	.039‡	.040†
Average	.108	.082	.062	.075	**.058**	.132	.148	.085	.114	**.074**	.174	.135	.115	.140	**.103**	.087	.066	.064	.054	**.051**

These plots display the estimated accuracy as a function of the true accuracy; a perfect CAP method would only contain values lying on the diagonal from (0,0) to (1,1). For the sake of clarity, we have omitted the Naive method, which always predicts accuracy values lying on a horizontal line and is thus uninteresting. Note that ATC often overestimates accuracy (haberman, cmc.2), while DoC tends to be less biased in this respect. DoC slightly overestimates accuracy in cmc.2 and underestimates it in iris.2, but generally produces values close to the diagonal (e.g., german). $\text{LEAP}_{\text{KDEy}}$ is generally closer to the diagonal but also produces high errors at the extremes in some datasets (see haberman). Notably, LEAP_{ACC} often shows a marked pattern in which the predicted values appear to lie on parallel horizontal and equidistant lines (haberman, cmc.2). The explanation for this pattern is that the estimation for the class prevalence in LEAP_{ACC} relies on the crisp counts of the classifier predictions (Equation 15), which are

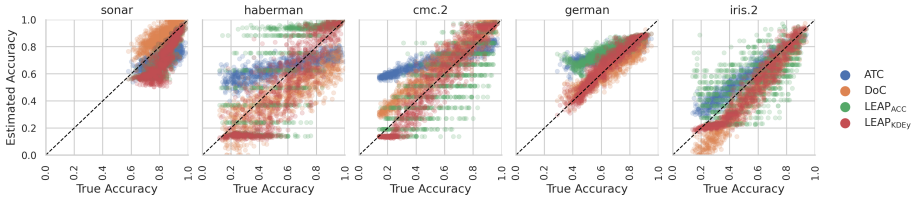

Fig. 1. Diagonal plots illustrating the correlation between true accuracy (x-axis) and estimated accuracy (y-axis) of a classifier trained via LR, for different CAP methods.

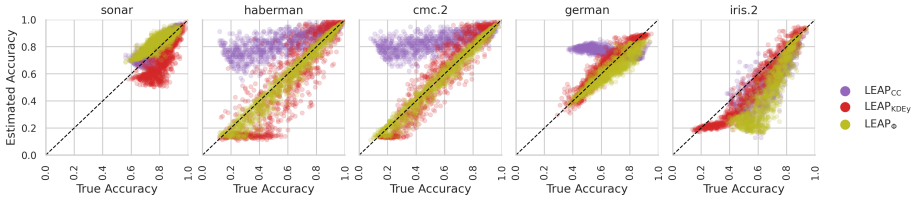

Fig. 2. Diagonal plots showing the same experimental setup as 1, but confronting $\text{LEAP}_{\text{KDEy}}$ (lower-bound) with LEAP_{CC} and LEAP_{Φ} (upper-bound).

naturally discrete. This effect is corrected in the enhanced variant $\text{LEAP}_{\text{KDEy}}$, which instead allows for continuous predictions via KDEy. Turning back to the results of Table 1, it is also noteworthy that there are some cases (few though) in which the Naive method beats all other competitors. All in all, this suggests that further research is still needed to better determine which method to apply in which scenario.

Concerning the baselines ATC and DoC, our experiments seem to indicate that ATC is weaker than DoC when facing PPS. On average, DoC beats ATC consistently in all cases. ATC shows erratic behavior for kNN, obtaining 4 best results and 14 worst results, thus performing, on average, even worse than the Naive baseline for this classifier. DoC obtains 6 best results for LR, 13 for kNN, 10 for SVM, and 8 for MLP, out of 29 datasets. Still, $\text{LEAP}_{\text{KDEy}}$ shows a relative error reduction, with respect to DoC, of 5.96% in LR, 12.7% in kNN, 11.1% in SVM, and 20.3% in MLP.

Table 2. Pearson correlation between the random variables "classifier accuracy" (in terms of Acc) and "CAP performance" (in terms of AE).

	Naive	ATC	DoC	LEAP	$\text{LEAP}_{\text{KDEy}}$
LR	-0.8340	-0.7241	-0.5050	-0.5560	**-0.3392**
k-NN	-0.7724	-0.8221	**-0.3130**	-0.6519	-0.3987
SVM	-0.8640	-0.8228	-0.5975	-0.5338	**-0.3785**
MLP	-0.7222	-0.6807	-0.3955	**-0.3686**	-0.3825
Average	-0.7981	-0.7624	-0.4527	-0.5276	**-0.3747**

To better understand the performance leeway that depends on the choice of the quantifier, we confront $\text{LEAP}_{\text{KDEy}}$ against LEAP_{CC} (the worst quantifier) and against LEAP_{Φ} (the perfect quantifier). As shown in Figure 2, LEAP_{CC} performs markedly worse than $\text{LEAP}_{\text{KDEy}}$, thus confirming that KDEy plays a key role in the effectiveness of the method. On the other side, LEAP_{Φ}, represents an ideal upper bound for LEAP. Nonetheless, the results show that the gap between LEAP_{Φ} and $\text{LEAP}_{\text{KDEy}}$ is relatively narrow, which indicates our proposed variant is a good method in practice.

Regarding the classifiers under study, the accuracy of MLP seems to be the easiest one to determine, as witnessed by the fact that the averaged CAP errors are the smallest across the four classifiers. While CAP accuracy should not be confused with classifier accuracy, we observed a strong negative Pearson correlation between these two random variables, approximately $r = -0.6$ for all classifiers and CAP methods (p-value $\ll 0.0001$). This indicates a general trend where better classifier performance is associated with lower CAP error. This observation is consistent with the average CAP results obtained for our classifiers, which achieved the following average accuracy scores across all datasets: $\text{Acc}(\text{LR}, U_i) = 0.7843 \pm 0.2052$, $\text{Acc}(k\text{NN}, U_i) = 0.7275 \pm 0.1963$, $\text{Acc}(\text{SVM}, U_i) = 0.7089 \pm 0.2622$, and $\text{Acc}(\text{MLP}, U_i) = 0.8148 \pm 0.1739$. This strong correlation is undesirable because we want our CAP methods to perform well regardless of classifier accuracy. Table 2 shows the correlation values for each pair of (CAP method, classifier). Although still biased, $\text{LEAP}_{\text{KDEy}}$ exhibits the smallest correlation on average. This, along with the fact that $\text{LEAP}_{\text{KDEy}}$ achieves the lowest CAP errors on average across all classifiers, suggests that $\text{LEAP}_{\text{KDEy}}$ is a noteworthy new contender in the CAP arena.

In preliminary experiments we have carried out (here omitted for reasons of space), we did not find any method (including ours) that stands out in terms of performance in the multiclass case. For this experiment, we considered 21 multiclass datasets from the UCI machine learning repository [14]; our results seem to indicate there is no statistically significant difference among the methods tested (including Naive). The number of classes surely makes the problem so difficult that it ends up hindering the relative merits of the different systems. We plan to investigate this issue more thoroughly in the near future.

Concerning running times, all CAP methods exhibit fast performance, never exceeding 3.5 seconds during training and 23 milliseconds during test. DoC tends to be the slowest method during training, with LEAP_{ACC} and $\text{LEAP}_{\text{KDEy}}$ requiring times up to two orders of magnitude lower. However, $\text{LEAP}_{\text{KDEy}}$ appears to be the slowest method of the lot at test time, performing up to ten times slower than the baselines. Nevertheless, test times range from milliseconds to tens of milliseconds for both LEAP_{ACC} and $\text{LEAP}_{\text{KDEy}}$, indicating that both variants are well-suited for real-world applications.

6 Conclusions

In this paper we have proposed a new method for classifier accuracy prediction (CAP) specifically designed to address situations affected by prior probability shift (PPS). Our method consists of 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 methods from the literature, LEAP is not limited to any specific evaluation measure. We have also drawn a connection with the field of quantification, and through many experiments we have shown that, when our method is endowed with state-of-the-art quantifiers, it tends to outperform existing CAP methods under PPS.

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. We also plan to conduct a systematic examination of the multiclass setting, aiming to enhance the accuracy estimation of our methods in such scenario. We intend to extend our method to other types of dataset shift, thus accommodating different assumptions on the data distributions, such as covariate shift. Additionally, we plan to investigate the ability of our method to estimate the accuracy of classifiers based on deep learning.

Acknowledgments

Founded by the QuaDaSh project (P2022TB5JF) "Finanziato dall'Unione europea-Next Generation EU, Missione 4 Componente 2 CUP B53D23026250001".

References

1. Bella, A., Ferri, C., Hernández-Orallo, J., Ramírez-Quintana, M.J.: Quantification via probability estimators. In: Proceedings of the 11th IEEE International Conference on Data Mining (ICDM 2010). pp. 737–742. Sydney, AU (2010)
2. Bunse, M.: On multi-class extensions of Adjusted Classify and Count. In: Proceedings of the 2nd International Workshop on Learning to Quantify (LQ 2022). pp. 43–50. Grenoble, IT (2022)
3. Chen, J., Liu, F., Avci, B., Wu, X., Liang, Y., Jha, S.: 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). pp. 14980–14992. Virtual Event (2021)
4. Chen, M.F., Goel, K., Sohoni, N.S., Poms, F., Fatahalian, K., Ré, C.: Mandoline: Model evaluation under distribution shift. In: Proceedings of the 38th International Conference on Machine Learning (ICML 2021). pp. 1617–1629. Virtual Event (2021)
5. Esuli, A., Fabris, A., Moreo, A., Sebastiani, F.: Learning to quantify. Springer Nature, Cham, CH (2023). https://doi.org/10.1007/978-3-031-20467-8
6. Fernandes Vaz, A., Izbicki, R., Bassi Stern, R.: Prior shift using the ratio estimator. In: Proc. of the International Workshop on Bayesian Inference and Maximum Entropy Methods in Science and Engineering. pp. 25–35. Jarinu, BR (2017)

7. Firat, A.: Unified framework for quantification. arXiv:1606.00868v1 [cs.LG] (2016)
8. Forman, G.: Counting positives accurately despite inaccurate classification. In: Proceedings of the 16th European Conference on Machine Learning (ECML 2005). pp. 564–575. Porto, PT (2005). https://doi.org/10.1007/11564096_55
9. Forman, G.: Quantifying counts and costs via classification. Data Mining and Knowledge Discovery **17**(2), 164–206 (2008)
10. Garg, S., Balakrishnan, S., Lipton, Z.C., Neyshabur, B., Sedghi, H.: Leveraging unlabeled data to predict out-of-distribution performance. In: Proceedings of the 10th International Conference on Learning Representations (ICLR 2022). Virtual Event (2022)
11. González, P., Castaño, A., Chawla, N.V., del Coz, J.J.: A review on quantification learning. ACM Computing Surveys **50**(5), 74:1–74:40 (2017)
12. Guillory, D., Shankar, V., Ebrahimi, S., Darrell, T., Schmidt, L.: Predicting with confidence on unseen distributions. In: Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV 2021). pp. 1134–1144. Montreal, CA (2021)
13. Jiang, Y., Nagarajan, V., Baek, C., Kolter, J.Z.: Assessing generalization of SGD via disagreement. In: Proceedings of the International Conference on Learning Representations (ICLR 2022). Virtual Event (2022)
14. Kelly, M., Longjohn, R., Nottingham, K.: The UCI machine learning repository, https://archive.ics.uci.edu
15. Lipton, Z.C., Wang, Y., Smola, A.J.: Detecting and correcting for label shift with black-box predictors. In: Proceedings of the 35th International Conference on Machine Learning (ICML 2018). pp. 3128–3136. Stockholm, SE (2018)
16. Moreo, A., Esuli, A., Sebastiani, F.: QuaPy: A Python-based framework for quantification. In: Proceedings of the 30th ACM International Conference on Knowledge Management (CIKM 2021). pp. 4534–4543. Gold Coast, AU (2021)
17. Moreo, A., González, P., del Coz, J.J.: Kernel density estimation for multiclass quantification. arXiv:2401.00490 [cs.LG] (2024)
18. 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.: Scikit-learn: Machine learning in Python. Journal of Machine Learning Research **12**, 2825–2830 (2011)
19. Schölkopf, B., Janzing, D., Peters, J., Sgouritsa, E., Zhang, K., Mooij, J.M.: On causal and anticausal learning. In: Proceedings of the 29th International Conference on Machine Learning (ICML 2012). Edinburgh, UK (2012)
20. Smith, N.A., Tromble, R.W.: Sampling uniformly from the unit simplex. Tech. rep., Johns Hopkins University (2004)
21. Storkey, A.: When training and test sets are different: Characterizing learning transfer. In: Quiñonero-Candela, J., Sugiyama, M., Schwaighofer, A., Lawrence, N.D. (eds.) Dataset shift in machine learning, pp. 3–28. The MIT Press, Cambridge, US (2009)
22. Sugiyama, M., Nakajima, S., Kashima, H., Buenau, P., Kawanabe, M.: 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). pp. 1433–1440. Vancouver, CA (2007)
23. Tasche, D.: Comments on Friedman's method for class distribution estimation. arXiv:2405.16666 [cs.LG] (2024)
24. Zhang, K., Schölkopf, B., Muandet, K., Wang, Z.: Domain adaptation under target and conditional shift. In: Proceedings of the 30th International Conference on Machine Learning (ICML 2013). pp. 819–827. Atlanta, US (2013)