Journal of Machine Learning Research 26 (2025) 1-54 Submitted 3/21; Revised 2/25; Published 3/25

A Comparative Evaluation of Quantification Methods

Tobias Schumacher TOBIAS.SCHUMACHER@UNI-MANNHEIM.DE
University of Mannheim, Germany
RWTH Aachen University, Germany
Markus Strohmaier MARKUS.STROHMAIER@UNI-MANNHEIM.DE
University of Mannheim, Germany
GESIS - Leibniz Institute for the Social Sciences, Germany
Complexity Science Hub, Austria
Florian Lemmerich FLORIAN.LEMMERICH@UNI-PASSAU.DE
University of Passau, Germany

Editor: Ingo Steinwart

Abstract

Quantification represents the problem of estimating the distribution of class labels on unseen data. It also represents a growing research field in supervised machine learning, for which a large variety of different algorithms has been proposed in recent years. However, a comprehensive empirical comparison of quantification methods that supports algorithm selection is not available yet. In this work, we close this research gap by conducting a thorough empirical performance comparison of 24 different quantification methods on in total more than 40 datasets, considering binary as well as multiclass quantification settings. We observe that no single algorithm generally outperforms all competitors, but identify a group of methods that perform best in the binary setting, including the threshold selection-based *median sweep* and *TSMax* methods, the *DyS* framework including the *HDy* method, *Forman's mixture model*, and *Friedman's method*. For the multiclass setting, we observe that a different, broad group of algorithms yields good performance, including the *HDx* method, the *generalized probabilistic adjusted count*, the *readme* method, the *energy distance minimization* method, the *EM algorithm for quantification*, and *Friedman's method*. We also find that tuning the underlying classifiers has in most cases only a limited impact on the quantification performance. More generally, we find that the performance on multiclass quantification is inferior to the results obtained in the binary setting. Our results can guide practitioners who intend to apply quantification algorithms and help researchers identify opportunities for future research.

Keywords: quantification, supervised machine learning, comparative evaluation, class distribution estimation, prevalence estimation

1. Introduction

Quantification is the problem of estimating the distribution of class labels on unseen (test) data. That is, after being trained on a dataset with known class labels, we want to estimate the number of instances of each class in a dataset with unknown class labels. In contrast to traditional classification tasks, we are not interested in individual predictions, but only in aggregated values on a group level. For this problem setting, previous research has

established that training a classification algorithm and counting instance-wise predictions generally does not yield accurate estimates (Forman, 2008; Tasche, 2016). This has given rise to a relatively young but vivid research field within the machine learning community. As an increasing number of researchers are becoming aware of this issue, a growing number of novel methods have been proposed. Although a first review of existing quantification methods has been provided by González et al. (2017), and recent publications also provide broader frameworks for quantification learning (Maletzke et al., 2019, 2020), a thorough, empirical, and independent comparison of quantification methods has not yet been presented.

With this work, we aim to fill this research gap by providing a comparison of 24 different quantification algorithms over 40 datasets. Apart from assessing approaches for the binary-class setting, we also include experiments for the multiclass quantification setting, which has received limited attention in quantification research so far. For each dataset and algorithm, we evaluate several degrees of distribution shifts between training data and test data with varying training set sizes. Furthermore, we evaluate whether applying more accurate base classifiers will also yield a better performance of the quantifiers using these. Altogether, these experiments encompass more than 5 million algorithm runs. To further validate our findings, we conduct a case study using the external competitive benchmark of the *LeQua 2022 challenge* (Esuli et al., 2022a,b).

Our experiments with binary class labels show that there is not a single algorithm that outperforms all others—but we identify a group of algorithms that on average perform significantly better than the rest, including the threshold selection-based *median sweep* and *TSMax* methods (Forman, 2008), *Friedman's method* (Friedman, 2014), *Forman's mixture model*, (Forman, 2005) and the *DyS* framework (Maletzke et al., 2019) including the *HDy* method (González-Castro et al., 2013). We also find that algorithms which optimize a classifier for the quantification problem yield on average worse performance, implying that their benefits in practice might be restricted to particular scenarios.

In the multiclass setting, we find a broader group of algorithms which show significantly better average performance than the rest, with the *HDx* method (González-Castro et al., 2013), *generalized probabilistic adjusted count* (Bella et al., 2010; Firat, 2016), *readme* (Hopkins and King, 2010), *energy distance minimization* (Kawakubo et al., 2016), the *EM algorithm for quantification* (Saerens et al., 2002), and *Friedman's method* (Friedman, 2014) leading in averaged rankings. These algorithms share the characteristic that they naturally allow for multiclass quantification. By contrast, extending predictions from binary quantifiers to the multiclass case in a one-vs.-rest fashion does not appear to yield competitive results, even when using strong base quantifiers such as the *median sweep* or the *DyS* framework. More generally, we observe significantly weaker performance for the multiclass case, corroborating that multiclass quantification constitutes a harder research problem and might need more research attention in the future.

In addition, across both settings, we observe that classifiers that were tuned for classification accuracy do not, in general, improve the predictions of the quantifiers applying them.

Overall, our results guide practitioners toward the most propitious quantification approaches for certain applications and help researchers identify promising future research avenues.

In the following, we first briefly introduce the quantification problem and describe how it conceptually differs from the classification problem. Afterward, Section 3 gives an overview of the algorithms included in our experimental comparison, providing a summary of the

state-of-the-art in quantification. Next, in Section 4, we provide a thorough description of the experimental setup of our comparison, before giving an in-depth presentation of the experimental results in Section 5. In Section 6, we present the results of the case study on the dataset from the LeQua 2022 challenge. Finally, in Section 7, we discuss the results of our experiments, before closing with our conclusions in Section 8.

2. The Quantification Problem

Quantification is a supervised machine learning problem that aims to estimate the distribution of class labels in a test set instead of predicting the class of individual instances. Throughout this paper, we use the following notation. For training, we are given a dataset of instances D_{train}, for which we know the values of multiple (categorical or continuous) features X and the corresponding class label Y. Letting L denote the number of possible values for the class label, we distinguish between the *binary* case, that is, there are exactly $L = 2$ possible values for the class label, and the *multiclass* case, in which there are $L > 2$ options for the class label. Using the training data, the goal is then to train a model that predicts the distribution of the class label in some test data D_{test}, for which only the values of the features X are known. In the following, we will often use the term *prevalence* for the relative frequency of single labels in training or test data. We formally denote the distributions of X and Y in the training set by $P_{\text{train}}(X)$ and $P_{\text{train}}(Y)$, and their distribution in the test set by $P_{\text{test}}(X)$ and $P_{\text{test}}(Y)$. Since in the binary case, the full distribution is already specified by the share of one class, we will denote for shorter notation the instances of one arbitrary class as *positives*, and label their prevalence in training and test data as pos_{train} and pos_{test}, respectively.

In contrast to traditional classification, a *shift* of the distribution of the class label Y, that is, a difference between the class probabilities $P_{\text{train}}(Y)$ in the training set and the class probabilities $P_{\text{test}}(Y)$ in the test set, is expected. However, it is assumed that the conditional distributions $P(X|Y)$ are stable between training and test sets—this kind of distribution shift is also known as *prior probability shift* in machine learning literature (Storkey, 2008). Furthermore, compared to classification, it is also more common to expect the occurrence of instances with the exact same feature values but different labels.

A trivial approach to quantification, known as the *classify and count (CC)* method, applies an arbitrary classification method trained on the training data to the test data and predicts the distribution of the predicted labels. However, this has been theoretically and empirically shown to achieve insufficient results in many scenarios (Forman, 2008; Tasche, 2016).

3. Algorithms for Quantification

We first outline the quantification algorithms under consideration. Following a previous categorization (González et al., 2017), we distinguish between (i) adaptations of the *adjusted count*, (ii) distribution matching methods, and (iii) adaptations of traditional classification algorithms. An overview of the algorithms considered in our evaluation is given in Table 1.

Quantification Algorithm	Abbreviation	Reference	Multiclass	Continuous
Adjusted Count	AC	Forman (2005)	OVR	Yes
Probabilistic Adjusted Count	PAC	Bella et al. (2010)	OVR	Yes
Threshold Selection Policy X	TSX	Forman (2008)	OVR	Yes
Threshold Selection Policy T50	TS50	Forman (2008)	OVR	Yes
Threshold Selection Policy Max	TSMax	Forman (2008)	OVR	Yes
Median Sweep	MS	Forman (2008)	OVR	Yes
Generalized Adjusted Count	GAC	Firat (2016)	Yes	Yes
Generalized Prob. Adjusted Count	GPAC	Firat (2016)	Yes	Yes
DyS Framework (Topsøe Distance)	DyS	Maletzke et al. (2019)	OVR	Yes
Forman's Mixture Model	FMM	Forman (2008)	OVR	Yes
readme	readme	Hopkins and King (2010)	Yes	No
HDx	HDx	González-Castro et al. (2013)	Yes	No
HDy	HDy	González-Castro et al. (2013)	OVR	Yes
Friedman's Method	FM	Friedman (2014)	Yes	Yes
Energy Distance Minimization	ED	Kawakubo et al. (2016)	Yes	Yes
EM-Algorithm for Quantification	EM	Saerens et al. (2002)	Yes	Yes
CDE Iteration	CDE	Tasche (2017)	No	Yes
Classify and Count	CC	Forman (2008)	Yes	Yes
Probabilistic Classify and Count	PCC	Bella et al. (2010)	Yes	Yes
SVM^{perf} using *KLD* loss	SVM-K	Esuli et al. (2010)	No	Yes
SVM^{perf} using Q-measure loss	SVM-Q	Barranquero et al. (2015)	No	Yes
Nearest Neighbor Quantification	PWK	Barranquero et al. (2013)	Yes	No
Quantification Forest	QF	Milli et al. (2013)	Yes	No
AC-corrected Quantification Forest	QF-AC	Milli et al. (2013)	No	No

Table 1: Overview of considered quantification algorithms. *Multiclass* indicates whether an algorithm can naturally handle this setting (Yes), requires the one-vs.-rest approach (OVR), or is not considered in our multiclass experiments (No). *Continuous* indicates whether an algorithm can handle continuous features.

3.1 Adaptations of the Adjusted Count

The trivial *classify and count (CC)* method just applies an arbitrary classifier c on the test data and counts the number of respective predictions. The core idea behind the *adjusted count (AC)* approach is to adjust these results post hoc for potential biases. This is done by exploiting the assumption that the likelihood $P(X|Y)$ of the features X given the class label Y does not vary between training and test data. Assuming binary labels, the true positive rate (*tpr*) and false positive rate (*fpr*) of a classifier, which correspond to the probabilities $P(c(X) = 1|Y = 1)$ and $P(c(X) = 1|Y = 0)$, respectively, can be expected to be identical between training and test data—see also Appendix A, Equation 5 for formal definitions of these rates. Letting $\widehat{pos}_{\text{test}}$ denote the predicted prevalence of positives by the *CC* method, we can express this quantity in terms of the true prevalence of positives pos_{test} and the (mis)classification rates *tpr* and *fpr* via

$$\widehat{pos}_{\text{test}} = pos_{\text{test}} \cdot tpr + (1 - pos_{\text{test}}) \cdot fpr,$$

which we can solve for pos_{test} to obtain the *AC* estimation

$$pos_{\text{test}} = \frac{\widehat{pos}_{\text{test}} - fpr}{tpr - fpr}. \tag{1}$$

In practice, it can occur that the estimate falls outside the feasible interval $[0, 1]$. In such cases, the outcome has to be clipped to the boundary values.

Based on this idea, in the literature a few variations of the AC method have been introduced, and the following methods are included in our experiments.

1. **Adjusted Count (AC).** As described above, we estimate the true positive and false positive rates from the training data and use them to adjust the output of the CC method (Forman, 2005).

2. **Probabilistic Adjusted Count (PAC).** This method adapts the AC approach by using average class-conditional confidences from a probabilistic classifier instead of true positive and false positive rates (Bella et al., 2010).

3. **Threshold Selection Policies (TSX, TS50, TSMax, MS).** The core idea of these variations is to shift the decision boundary (e.g., classify an instance as positive if the original estimate $c(x)$ is larger than 0.7) of the underlying classifier in order to make the AC estimation in Equation 1 more numerically stable. Different strategies involve using the threshold that maximizes the denominator $tpr - fpr$ ($TSMax$), a threshold for which we have $fpr = 1 - tpr$ (TSX), a threshold at which $tpr \approx 0.5$ holds ($TS50$), or, as in the *median sweep* (MS) method, using an ensemble of such threshold-based methods and taking the median prediction (Forman, 2008).

3.2 Distribution Matching Methods

The majority of existing quantification methods can be categorized as *distribution matching algorithms*. These algorithms are implicitly based on the assumption that the distribution of the features X conditioned on the distribution of the class labels Y does not change between training data and test data. Under that assumption, with $\ell_j, j \in \{1, \ldots, L\}$, denoting the possible values of the labels Y, the law of total probability yields that

$$P_{\text{test}}(X) = \sum_{j=1}^{L} P_{\text{train}}(X|Y = \ell_j) P_{\text{test}}(Y = \ell_j). \tag{2}$$

As in this equation, both the left-hand distribution $P_{\text{test}}(X)$ and the conditional distributions $P_{\text{train}}(X|Y = \ell_j)$ on the right-hand side can be seen as represented by given training and test data, only the sought-for probabilities $P_{\text{test}}(Y = \ell_j)$ are left as unknowns. To estimate these class probabilities, there are two main issues to be worked out from a methodological point of view. First, estimating or modeling the distributions $P_{\text{train}}(X|Y = \ell_j)$ and $P_{\text{test}}(X)$ is not at all trivial. There can be an arbitrary amount of features X, and the training data usually does not provide nearly enough samples to accurately represent the distribution of the feature space, even more when conditioning on the class labels Y. Second, once the distributions $P_{\text{test}}(X)$ and $P_{\text{train}}(X|Y = \ell_j)$ have been estimated, there are also various ways to predict the class probabilities $P_{\text{test}}(Y = \ell_j)$ from these estimations.
The methods discussed in this chapter tackle these issues in various ways. One basic approach to tackle the first issue has, for instance, already been introduced when discussing

the *adjusted count*. In the *adjusted count* approach, information on the distribution of the features X was derived by applying a classifier c and considering the distribution of their outputs $P(c(X))$ instead of $P(X)$. That way, Equation 2 would be transformed to the set of linear equations

$$P_{\text{test}}(c(X) = \ell_i) = \sum_{j=1}^{L} P_{\text{train}}(c(X) = \ell_i | Y = \ell_j) P_{\text{test}}(Y = \ell_j), \quad i \in \{1, \ldots, L\}. \tag{3}$$

However, there are also methods that do not apply classifiers, and instead, for instance, estimate $P(X)$ based on the distributions of single features, or in terms of distances between individual instances in the data.
Regarding the second issue, most of the presented methods translate Equation 2 into a set of linear equations, and then minimize some distance function between the left- and right-hand side expressions, subject to the constraints that $\sum_{j=1}^{L} P_{\text{test}}(Y = \ell_j) = 1$ and $P_{\text{test}}(Y = \ell_j) \geq 0$ for all $j \in \{1, \ldots, L\}$ have to hold. This common pattern has already been noted by Firat (2016).
Among all the methods of this category, we compare the following methods:

1. **Generalized Adjusted Count Models (GAC, GPAC).** As described above, the most simple work-around to avoid estimating $P(X)$ is to apply a classifier to build a system of linear questions as in Equation 3, and solve it via constrained least-squares regression (Firat, 2016). That approach can be considered as a *generalized adjusted count (*GAC*)* method, which also naturally includes the multiclass case. Similarly, one can obtain the *generalized probabilistic adjusted count* (*GPAC*) method, by making use of the posterior probabilities from probabilistic classifiers as in the *PAC* method.

2. **The DyS Framework (DyS, HDy).** More recently, Maletzke et al. (2019) proposed the *DyS* framework, in which the main idea is to use confidence scores resulting from the decision functions of a binary classifier. More precisely, the confidence scores obtained on the training data are divided into bins, and then the probability that the confidence score of an instance ends up in that bin is estimated from the training set. Thus, in our context, the number of linear equations we obtain from Equation 2 equals the chosen number of bins, which, next to the distance function that this set of equations is optimized on, can be seen as a parameter of this framework. A main drawback of this framework is that it only works for the binary case, and that many of the distance functions that were proposed and evaluated for this framework are not convex, requiring methods such as ternary search to estimate the optimal solution. Since using the Topsøe distance (Deza and Deza, 2009) has proven to yield consistently good results (Maletzke et al., 2019), we are applying this setup as *DyS* method in our experiments. Furthermore, it is noteworthy that this framework was motivated as a generalization of the *HDy* method (González-Castro et al., 2013), which uses the Hellinger distance to match distributions.

3. **Forman's Mixture Model (FMM).** Like the *DyS* framework, this method is based on matching distributions of classifier scores. Yet, instead of matching probability density functions which are estimated from binned classifier scores, Forman (2005)

proposed to match the cumulative distributions of classifier scores to avoid sparsity issues. To match these distributions, Forman proposed minimizing their *PP-area*, which practically corresponds to minimizing the Manhattan distance (Firat, 2016).

4. **Friedman's Method (FM).** Similar to the *GPAC* method, Friedman (2014) proposed to use the confidence scores from probabilistic classifiers. However, rather than averaging class-conditional confidence scores, his approach uses the fraction of class-conditional confidence scores that are above and below the observed class prevalences in the training data.

5. **Feature Distribution Matching (readme, HDx).** Instead of applying a classifier, one can also directly model the distribution of features by counting co-occurences of multiple features as in the *readme* method (Hopkins and King, 2010), or by counting occurrences of individual features as in the *HDx* method (González-Castro et al., 2013). This requires that all features are categorical, or preprocessed accordingly via binning. In the *readme* method, one then matches the distributions via constrained least-squares regression. Due to sparsity issues, this is, however, only done by considering a random subset of all features. Yet, multiple of such subsets are drawn, and the resulting predictions are averaged to obtain the final estimate of the true class distribution. In the *HDx* method (González-Castro et al., 2013), by contrast, distributions of single features are aggregated and matched via the Hellinger distance.

6. **Energy Distance Minimization (ED).** As the name of this method suggests, its core idea is to minimize the energy distance between the left-hand and right-hand side distribution in Equation 2. In that context, the distribution of the feature space is intrinsically modeled by the Euclidean distances between individual instances, and therefore no classifiers or additional parameters are required (Kawakubo et al., 2016).

7. **The EM Algorithm for Quantification (EM).** This method applies the classic *EM algorithm* (Dempster et al., 1977) on the outputs of probabilistic classifiers to adjust them for potential distribution shift between the class distributions in training and test data. While quantification was not the main focus in the original proposal of the algorithm (Saerens et al., 2002), the sought-for class prevalences are obtained as a side-product.

8. **CDE Iteration (CDE).** The *class distribution estimation (CDE) iterator* (Xue and Weiss, 2009) applies principles from cost-sensitive classification to account for changes in class distributions between training and test data. For that purpose, the misclassification costs are updated iteratively, and in the original proposition of the algorithm, the underlying classifier is retrained in every iteration step. In our experiments, we use the more efficient variant proposed by Tasche (2017), in which each iteration rather updates the decision threshold of an underlying probabilistic classifier. For this variant of the algorithm, Tasche has also proven that the iteration will eventually converge.

3.3 Classifiers for Quantification

Classifiers for quantification apply established classification methods in the setting of quantification. The main approach behind most of these methods is to optimize such established

classifiers based on a loss function that minimizes the quantification error, and then estimate the class distributions based on the predictions of the individual instances. Thus, these approaches are all, in some sense, variants of the *CC* method. In our experiments, the following methods are included:

1. **Classify and Count (CC).** This trivial approach applies a classifier and counts the number of times that each class is predicted (Forman, 2008).

2. **Probabilistic Classify and Count (PCC).** This approach takes probabilistic predictions, i.e., continuous values between zero and one, and averages the predictions of all instances to estimate the class prevalences (Forman, 2008; Bella et al., 2010).

3. SVM^{perf} **optimization (SVM-Q, SVM-K).** This pair of methods applies the so-called SVM^{perf} classifier, which is an adaptation of traditional support vector machines that can be optimized for multivariate loss functions (Joachims, 2005). Based on this algorithm, multiple classifiers with different quantification-oriented loss functions have been proposed. For instance, Esuli et al. (2010) have proposed using the Kullback-Leibler divergence (*SVM-K*), while Barranquero et al. (2015) have developed *Q-measure* for this purpose (*SVM-Q*).

4. **Nearest Neighbor Quantification (PWK).** Barranquero et al. (2013) adapted the k-nearest neighbors algorithm for classification to the setting of quantification. In their k-NN approach, they apply a weighting scheme which applies less weight on neighbors from the majority class.

5. **Quantification Forests (QF, QF-AC).** The decision tree and random forest classifiers have been adapted for quantification by Milli et al. (2013). Other than in the traditional approach, the authors propose that the split in each decision tree is made based on a quantification-oriented loss function. Since in their original proposition, applying the *AC* method to the predictions of these random forests yielded particularly strong results, we include both the *quantification forests* and the *AC* adaptation of them in our experiments.

3.4 Multiclass Quantification

In the literature on quantification, the multiclass setting has received relatively little attention so far, despite Forman (2008) pointing out that this problem is much harder than binary quantification. In our comparative evaluation, we also take a closer look into this scenario. Approaches for multiclass quantification can be broadly separated into two categories:

1. **Natural Multiclass Quantifiers.** Like in classification, some quantification methods can also naturally handle the multiclass setting. This is the case for most distribution matching methods, as by Equation 2, there is no constraint on the number of classes that are summated. Further, quantification-oriented classifiers such as *PWK* can handle the multiclass setting as well, since the underlying classifier allows for it.

2. **One-vs.-Rest Quantifiers.** Traditional quantification methods such as *adjusted count* and its adaptations have been specifically designed for the binary setting. To

extend such methods to the multiclass setting, one can estimate the prevalence of each individual class in a one-vs.-rest fashion, and then normalize the resulting estimations afterward so that they sum to 1 (Forman, 2008). Next to all *adjusted count* adaptations, we also applied this strategy for the distribution matching methods from the *DyS* framework, and Forman's mixture model, as these do not naturally generalize to the multiclass setting.

An overview regarding which multiclass strategy is used for each quantification algorithm is also provided in Table 1. For the *SVM-K*, *SVM-Q*, and *QF-AC* methods, we did not conduct any multiclass experiments, as the underlying implementations do not provide a multiclass feature. Furthermore, for the *CDE iterator* we did not run multiclass experiments, since the individual one-vs.-rest predictions yielded extreme predictions of either 0 or 1 regularly.

4. Experimental Setup

In total, we compare 24 algorithms on 40 datasets. In the following, we provide details on the datasets, sampling protocols, algorithmic parameters, and evaluation measures. The implementation of the algorithms and experiments can be found on GitHub[1].

4.1 Datasets

We applied all algorithms on a broad range of 40 datasets collected from the UCI machine learning repository[2] and from Kaggle[3]. An overview of these datasets, along with their characteristics and abbreviations that we use when describing our results, is given in Table 2. Of the 40 datasets, 17 had a non-binary set of class labels or were even regression datasets. The regression datasets were converted to both multiclass and binary datasets by binning the values of the class variable. This was usually done with the abstract goal of achieving groups of similar size with respect to the number of instances to allow for a more robust basis for potential shifts in the following steps. The cutoff points for the bins were determined manually after looking at the distribution of the classes. Furthermore, the real multiclass datasets were also converted to binary datasets. In these cases, we kept the most populated class as is, and merged the other classes into a single class, like in a one-vs.-rest classification problem. By doing so, we preserved meaningful class semantics that classifiers and quantifiers could recognize. All datasets have been preprocessed the same way as for standard classification, including dummy coding their non-ordinal features, rescaling their continuous features, and removing missing values. Furthermore, to enable the application of algorithms that require a finite feature space, we created a variation of each dataset in which all non-categorical features were binned. All algorithms that could handle a non-finite feature space were run on the unbinned datasets. While one may argue that due to these alterations in the datasets the results would be less comparable, the binning procedure ultimately simulates the loss of information that one would have to accept when applying such restricted algorithms in the first place.

1. https://github.com/tobiasschumacher/quantification_paper
2. https://archive.ics.uci.edu/ml/index.php
3. https://www.kaggle.com/datasets

Dataset	Abbr.	D	Non-Categorical	N	L	Source
Internet Advertisements	ads	1560	Yes	2359	2	UCI
Adult	adult	89	Yes	45222	2	UCI
Student Alcohol Consumption	alco	57	Yes	1044	2	Kaggle
Avila	avila	10	Yes	20867	2	UCI
Breast Cancer Wisconsin (Diagnostic)	bc-cat	31	Yes	569	2	UCI
Breast Cancer Wisconsin (Original)	bc-cont	10	Yes	683	2	UCI
Bike Sharing Dataset	bike	59	Yes	17379	4	UCI
BlogFeedback	blog	280	Yes	52397	4	UCI
MiniBooNE Particle Identification	boone	50	Yes	129569	2	UCI
Credit Approval	cappl	44	Yes	653	2	UCI
Car Evaluation	cars	22	No	1728	2	UCI
Default of Credit Card Clients	ccard	34	Yes	30000	2	UCI
Concrete Compressive Strength	conc	8	Yes	1030	3	UCI
Superconductivity Data	cond	89	Yes	21263	4	UCI
Contraceptive Method Choice	contra	13	Yes	1473	3	UCI
SkillCraft1 Master Table	craft	18	Yes	3338	3	UCI
Diamonds	diam	22	Yes	53940	3	Kaggle
Dota2 Games Results	dota	116	No	102944	2	UCI
Drug Consumption	drugs	136	Yes	1885	3	UCI
Appliances Energy Prediction	ener	25	Yes	19735	3	UCI
FIFA 19 Complete Player Dataset	fifa	117	Yes	14751	4	Kaggle
Solar Flare	flare	28	No	1066	2	UCI
Electrical Grid Stability Simulated Data	grid	11	Yes	10000	2	UCI
MAGIC Gamma Telescope	magic	10	Yes	19020	2	UCI
Mushroom	mush	111	No	8124	2	UCI
Geographical Original of Music	music	116	Yes	1059	2	UCI
Musk (Version 2)	musk	166	Yes	6598	2	UCI
News Popularity in Multiple Social Media Platforms	news	60	Yes	39644	4	UCI
Nursery	nurse	27	No	12960	3	UCI
Occupancy Detection	occup	5	Yes	20560	2	UCI
Phishing Websites	phish	31	No	11055	2	UCI
Spambase	spam	58	Yes	4601	2	UCI
Students Performance in Exams	study	19	Yes	1000	2	Kaggle
Telco Customer Churn	telco	45	Yes	7032	2	Kaggle
First-order Theorem Proving	thrm	51	Yes	6117	3	UCI
Turkiye Student Evaluation	turk	31	No	5820	3	UCI
Video Game Sales	vgame	133	Yes	6825	4	Kaggle
Gender Recognition by Voice	voice	20	Yes	3168	2	Kaggle
Wine Quality	wine	14	Yes	6497	4	UCI
Yeast	yeast	9	Yes	1484	5	UCI

Table 2: Datasets used in our experiments. *Abbr.* indicates abbreviations of their names that we use when describing our experimental results, D indicates the number of features, L indicates the number of classes, N corresponds to the number of instances in the data, and *Non-Categorical* indicates whether a dataset contains features that required binning. Note that this latter aspect is relevant for quantification algorithms such as *readme* that require a finite feature space.

Overall, these datasets represent a wide range of domains, and are shaped differently in terms of their number of instances as well as in the design of their feature spaces.

4.2 Sampling Strategy

As we aimed to evaluate quantifiers under a large set of diverse conditions, we chose a sampling approach in which we varied (i) the training distribution, (ii) the test distribution, and (iii) the (relative) sizes of training and test datasets. Regarding training and test distributions, in the binary case, we considered different prevalences of training positives pos_{train} and test positives pos_{test} in the respective sets

$$pos_{\text{train}} \in \{0.05, 0.1, 0.3, 0.5, 0.7, 0.9\} \quad \text{and}$$
$$pos_{\text{test}} \in \{0, 0.01, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\},$$

following the protocol introduced by Forman (2008). For both distributions, we sampled broadly across the interval $[0, 1]$, also including very unbalanced, and thus presumably difficult settings with only very few (or, for the test set, even no) positive labels.
Concerning the multiclass case, we considered datasets with a varying number of $L \in \{3, 4, 5\}$ different classes. For each of these values of L, we fixed a set of three training and five test class distributions, representing relatively uniform as well as polarized class distributions, which can be seen in Table 3.
In both binary and multiclass settings, we considered splits with relative amounts of training versus test data samples in

$$\{(0.1, 0.9), (0.3, 0.7), (0.5, 0.5), (0.7, 0.3)\},$$

thereby simulating scenarios in which we have little as well as relatively much data at hand to train our models. We omitted splits with 90% training data to save computational resources, since the computational complexity of most algorithms in our experiments is determined by the size of the training data rather than the test data. Even without this particular split, in the binary setting we obtained 288 combinations of training distributions, test distributions, and training/test-splits, and in the multiclass setting we obtained 60 of such combinations for each dataset.
To collect experimental data from each dataset that satisfy these constraints, we used undersampling, i.e., we sampled from a given dataset as many data instances as possible without replacement. We illustrate this sampling strategy with an example. Assume a dataset with 1000 instances and a binary class attribute, consisting of 700 positive and 300 negative instances. As an example evaluation scenario, we aim to sample data with an 80/20 split in training and test sets and with a 60/40 distribution of positive and negative instances in both training and test sets. Splitting the 300 negative instances randomly 80/20, we have $0.8 \cdot 300 = 240$ negative instances available for training and $0.2 \cdot 300 = 60$ instances available for testing. To obtain a 60/40 distribution of positives and negatives in the training data, we therefore have to choose $240 : \frac{40}{60} = 360$ positive instances to include in the training data, which we randomly sample from the full set of positive instances. The positives for the test data are sampled analogously. Note that the instance count for each label imposes a constraint on the number of sampled instances with other labels. In general, we used the maximum number of instances for each label that satisfied all constraints.

L	Training Distributions $P_{\text{train}}(Y)$	Test Distributions $P_{\text{test}}(Y)$
3	$(0.2, 0.5, 0.3)$, $(0.05, 0.8, 0.15)$, $(0.35, 0.3, 0.35)$	$(0.1, 0.7, 0.2)$, $(0.55, 0.1, 0.35)$, $(0.35, 0.55, 0.1)$, $(0.4, 0.25, 0.35)$, $(0., 0.05, 0.95)$
4	$(0.5, 0.3, 0.1, 0.1)$, $(0.7, 0.2, 0.1, 0.1)$, $(0.25, 0.25, 0.25, 0.25)$	$(0.65, 0.25, 0.05, 0.05)$, $(0.2, 0.25, 0.3, 0.25)$, $(0.45, 0.15, 0.2, 0.2)$, $(0.2, 0, 0, 0.8)$, $(0.3, 0.25, 0.35, 0.1)$
5	$(0.05, 0.2, 0.1, 0.2, 0.45)$, $(0.05, 0.1, 0.7, 0.1, 0.05)$, $(0.2, 0.2, 0.2, 0.2, 0.2)$	$(0.15, 0.1, 0.65, 0.1, 0)$, $(0.45, 0.1, 0.3, 0.05, 0.1)$, $(0.2, 0.25, 0.25, 0.1, 0.2)$, $(0.35, 0.05, 0.05, 0.05, 0.5)$, $(0.05, 0.25, 0.15, 0.15, 0.4)$

Table 3: List of training distributions $P_{\text{train}}(Y)$ and test distributions $P_{\text{test}}(Y)$ considered for experiments in the multiclass setting, ordered by number of classes L. In both the columns for training and test distribution, each row represents a distribution of instances that was sampled from the corresponding data. For instance, assuming that for a dataset with $L = 3$ classes, the corresponding labels are given by $Y \in \{1, 2, 3\}$, the first row among the column of training distribution indicates that in our experiments, we have sampled training sets where Label 1 had a prevalence of 0.2, Label 2 had a prevalence of 0.5, and Label 3 had a prevalence of 0.3. For each combination of training and test distributions, we generated ten test scenarios by taking different samples.

In cases where the class distributions we aimed to sample strongly deviated from the natural class distributions in the given dataset, this sampling procedure led to a relatively small subset compared to the whole corpus. This made the quantification task comparatively more challenging in these settings.

To address possible variances in the drawn samples, we made ten independent draws for each combination of distributions that could occur within our protocol and ran all algorithms under study on each of these draws. To ensure the reproducibility of all these draws, we used a set of ten fixed seeds for the random number generators. For the binary setting, we therefore performed in total 2880 draws per dataset, which, considering that we applied 24 algorithms on 40 datasets, yielded 2,764,800 experiments for that setting. Adding 204,000 additional experiments in the multiclass case and 2,666,520 more experiments on tuned alternative base classifiers (cf. Section 5.3), we conducted a total number of more than 5 million experiments in our evaluation.

4.3 Algorithms and Parameter Settings

In our experiments, we compared all algorithms that are described in Section 3 and listed in Table 1. Except for the SVM$^{\text{perf}}$-based quantifiers and *quantification forests*, all algorithms were implemented from scratch in Python 3, using `scikit-learn` as base implementation for the underlying classifiers and the package `cvxpy` (Diamond and Boyd, 2016) to solve constrained optimization problems. For the SVM$^{\text{perf}}$ algorithm, we used the corresponding open source software package by Joachims (2005), and adapted the code that Esuli et al. (2018) have used as a baseline for their *QuaNet* method to connect Joachim's C++ implementation to Python. Regarding *quantification forests*, we used the original implementation that was kindly provided by the authors (Milli et al., 2013).
We further compared our code against the `QuaPy` package (Moreo et al., 2021), which implements a subset of the methods considered in this evaluation, and has been released after the initial publication of our preprint. The results are presented in Appendix B.
As the focus of this work is on a general comparison of quantification algorithms, for all algorithms, we initially fixed a set of default parameters based on which the main experiments were conducted. When choosing the hyperparameters of each model, we followed recommendations from the original papers where possible. For all quantification methods that required a base classifier, we used the same logistic regression classifier for each dataset split. The logistic regression model was chosen because it is one of the most established and popular base classifiers and also actively models its outputs as class probability scores that are required for quantification methods such as the *PAC*, *EM*, or *FM* methods. In this way, the results of different quantifiers could not be biased by differences in the underlying classification performances. We acknowledge that fine-tuning the hyperparameters of each quantifier for each dataset could overall improve the performance, but argue that fixing parameters once allows for a fairer comparison of individual approaches and makes larger numbers of algorithm runs computationally feasible. However, since one could suspect a strong dependence of the quantification performances on the performance of the underlying classifiers, we further conducted a series of experiments in which we used stronger classifiers with tuned parameters; see Sections 4.3.2 and Section 5.3. In addition, we also explored the impact of parameter tuning within our case study on the dataset from the LeQua challenge (cf. Section 6.3). In the following, we first outline the parameter settings for the main experiments before giving details on the experiments in which we used tuned classifiers.

4.3.1 Parameter Settings in the Main Experiments

In our main experiments, we chose the following hyperparameters for the quantifiers:

- As mentioned above, for all methods that use a classifier to perform quantification, we used the logistic regression classifier with the default `L-BFGS` solver along with its built-in probability estimator provided by `scikit-learn` and set the number of maximum iterations at 1000. We always used stratified 10-fold cross-validation on the training set when estimating the misclassification rates or computing the set of scores and thresholds that the quantifiers needed.

- In all adaptations of the *adjusted count* that apply threshold selection policies, namely the *TSX*, *TS50*, *TSMax* and *MS* methods, we reduced the sets of scores and thresholds

obtained from cross-validation by rounding to three decimals. Additionally, in the *MS* algorithm, we followed Forman's recommendation to only use models that yield a value of at least 0.25 in the denominator of Equation 1.

- For the *DyS* framework, including the *HDy* method, we chose to divide its confidence scores into 10 bins, as this number of bins appeared to produce consistently strong results in the study by Maletzke et al. (2019).

- For the *EM* algorithm and the *CDE* iterators, we chose $\varepsilon = 10^{-6}$ as the convergence parameter and limited the number of iterations to a maximum of $m = 1000$ iterations, which was reached only very rarely.

- For the *readme* algorithm, we set the size of each feature subset to $\lfloor \log_2(D) + 1 \rfloor$, with D denoting the number of features in X. We considered an ensemble of 50 subsets that were all drawn uniformly.

- In the *QF* and *QF-AC* algorithms, we used the `weka`-based implementation that has kindly been provided by the authors. We left all parameters at their default values, including the size of the forest, which was set to 100 trees.

- For both the *SVM-Q* and the *SVM-K* method, we chose $C = 1$ as the regularization coefficient, which was, however, decreased to $C = 0.1$ when there were more than 10,000 training samples. This adaptation was chosen because, in our experiments, we observed that when large amounts of training data were present, a higher regularization parameter would significantly slow the convergence of the optimization.

- For the *PWK* algorithm, we chose a neighborhood size of $k = 10$, and a weighting parameter of $\alpha = 1$, as different weight values did not yield significantly better results in the study by Barranquero et al. (2015).

- In the rare case that in one-vs.-rest quantification, all individual class prevalences were predicted as 0, we returned the uniform distribution as prevalence estimation.

4.3.2 Parameter Settings in the Experiments on Tuned Classifiers

Many quantification methods rely on the predictions of an underlying base classifier to form their class prevalence estimations. Since the quality of these underlying classification models could have a strong impact on the performance of the quantifier, we evaluated the impact of applying more advanced classification methods with tuned hyperparameters in our second set of experiments. For that purpose, we conducted experiments with four classification models, namely random forests, AdaBoost, RBF kernel support vector machines, and logistic regression models. For each of these classifiers, we conducted a grid search to optimize the hyperparameters on every single dataset split in our experiments. Due to scalability issues, we however restricted ourselves to the 24 datasets which have not more than 10,000 instances in total. After having determined their optimal parameter configuration for each dataset split, we used each of the four classification models with their optimal parameterization as base classifiers for the quantification methods.

For the *CC*, *AC*, *GAC*, and *HDy* methods, all four classification models could be applied, as these only require pure (mis)classification rates from the training data for their estimations. For all quantifiers which require scores from a classifier's decision function, namely the threshold selection policies *TS50*, *TSX*, *TSMax*, and *MS*, as well as the *DyS* and *FMM* method, we only used the support vector classifier and the logistic regression model, since AdaBoost and random forests do not actively model such decision functions. Furthermore, for all quantifiers that require probability scores, we only applied the tuned logistic regressor, because it is the only method for which the outputs are modeled to represent probabilities.
Regarding the grid search protocol, we applied standard 5-fold cross-validation on the training data—test data was not considered for tuning—when tuning classifiers both in the binary and the multiclass setting, and determined the optimal parameterization based on the accuracy of the resulting classifiers. Given that in the multiclass case, many quantification methods apply the one-vs.-rest approach to generalize to this setting, and thus use L different binary quantifiers that each build on a binary classifier, we further applied a second protocol to accommodate this setting. Specifically, for each parameter configuration in the given grid, we trained L binary classifiers—one one-vs.-rest classifier for each class. For each class-wise classifier, we computed the *balanced accuracy*, i.e., the average of the true positive rate and the true negative rate in the given binary prediction settings—see also Appendix A, Equations 4 and 6 for formal definitions. For the one-vs.-rest quantification, we then used L differently parameterized base classifiers, always applying the parameters which yielded the best balanced accuracy in the corresponding one-vs.-rest classification.
Regarding the parameterization of all quantifiers and base classifiers in this experiment, we made the following choices:

- All parameters of the quantification methods that do not regard the underlying classifiers were kept as described in Section 4.3.1.

- In the grid search for the logistic regression classifier, we varied the regularization weight C within the set $\{2^i : i \in \{-15, -13, -11, \ldots, 13, 15\}\}$. Furthermore, for all values of C, we varied the weighting strategy for the instances, either setting the weights of all instances to 1, or weighting the instances inversely proportional to the prevalence of their corresponding class. Like in previous experiments, we applied the `L-BFGS` solver to efficiently learn the corresponding models and set the number of maximum iterations to 1000.

- For the random forest, we varied the maximum number of features considered per tree among the values $\{2^i : i \in \{1, 2, \ldots, 11\}\}$, and the minimum number of samples per leaf, which we considered as the main parameter to control the tree size, within the set $\{2^i : i \in \{1, 2, \ldots, 7\}\}$. Regarding the forest size, we kept a fixed high number of 1000 trees, since it is well-established that choosing a high number of trees yields more reliable results than any lower number of trees.

- In the support vector classifier, we varied the regularization weight C and the kernel parameter γ. We varied the first in the range $C \in \{2^i : i \in \{-5, -3, -1, \ldots, 13, 15\}\}$ and the latter in $\gamma \in \{2^i : i \in \{-17, -15, -13, \ldots, 3, 5\}\}$.

- Finally, for the AdaBoost classifier, there is a well-established trade-off between the number of classifiers and the learning rate. Therefore, we only varied the learning rate $\alpha \in \{2^i : i \in \{-19, -17, -13, \ldots, 1, 3\}\}$ and set the number of weak classifiers to a medium amount of 100.

In addition to these experiments with tuned base classifiers, we also performed experiments on the same datasets with variants of the *SVM-K* and *SVM-Q* methods, which applied an RBF kernel instead of the default linear kernel. Since these methods are designed to optimize for quantification-oriented loss functions, we did not perform any classification-oriented parameter tuning on these, and thus these methods in principle would not fit into this set of experiments. Yet, given that these RBF kernel-based variants are very computationally expensive, we were unable to incorporate these in our main experiments where the size of the datasets was not restricted to 10,000 instances. For these variants, we chose $C = 1$ as the regularization coefficient and $\gamma = 1$ as the kernel parameter.

4.4 Evaluation

Next, we describe the error measures that we used in our evaluation, as well as the procedure used to rank the quantification algorithms and determine statistically significant differences in the performance of the algorithms we have compared.

4.4.1 Error Measures for Quantification

The choice of performance measures for quantification is in itself not a trivial issue, and for a thorough review and discussion of existing quantification measures, we point to a recent survey by Sebastiani (2020). To evaluate the quantification performances in our experiments, we decided to use the *absolute error (AE)* and the *normalized Kullback-Leibler divergence (NKLD)*. In the following, we let $p \in \Delta^{L-1}$ denote the true distribution of labels Y in an unseen test set, and $\hat{p} \in \Delta^{L-1}$ denote the distribution of labels Y that has been predicted from a given quantifier on the test set, with Δ^{L-1} denoting the probability simplex. The absolute error between the true distribution p and an estimated distribution $\hat{p}$ is then given by

$$e_{AE}(p, \hat{p}) := \sum_{i=1}^{L} |p_i - \hat{p}_i| \ ,$$

whereas the normalized Kullback-Leibler divergence between p and $\hat{p}$ is defined as

$$e_{\mathrm{NKLD}}(p, \hat{p}) := 2 \cdot \frac{\exp\{e_{\mathrm{KLD}}(p, \hat{p})\}}{1 + \exp\{e_{\mathrm{KLD}}(p, \hat{p})\}} - 1 \ ,$$

with

$$e_{\mathrm{KLD}}(p, \hat{p}) := \sum_{i=1}^{L} p_i \log\left(\frac{p_i}{\hat{p}_i}\right)$$

denoting the Kullback-Leibler divergence. Since the Kullback-Leibler divergence is not defined when $\hat{p}_i = 0$ and $p_i \neq 0$ for some $i \leq L$, we smoothed the distributions by a small value $\varepsilon = 10^{-8}$ to avoid this problem.

We chose the AE measure because of its interpretability and its robustness against outliers. In contrast to related studies as conducted by González-Castro et al. (2013), we do not use the *Mean Absolute Error*, i.e., we do not divide by the number L of predicted classes. This avoids having different upper bounds for the error depending on L, which may make the resulting values harder to interpret, specifically when the number of classes is high, such as in the LeQua case study where $L = 28$. In addition, we selected NKLD because, in contrast to AE, it particularly punishes quantifiers which marginalize the minority class. Both measures are bounded to the same interval in both binary and multiclass quantification, with both values obtaining their minimum (and optimal value) at 0, and the maximum AE value being 2, while the maximum NKLD value is 1.

4.4.2 Statistical Evaluation of Performance Rankings

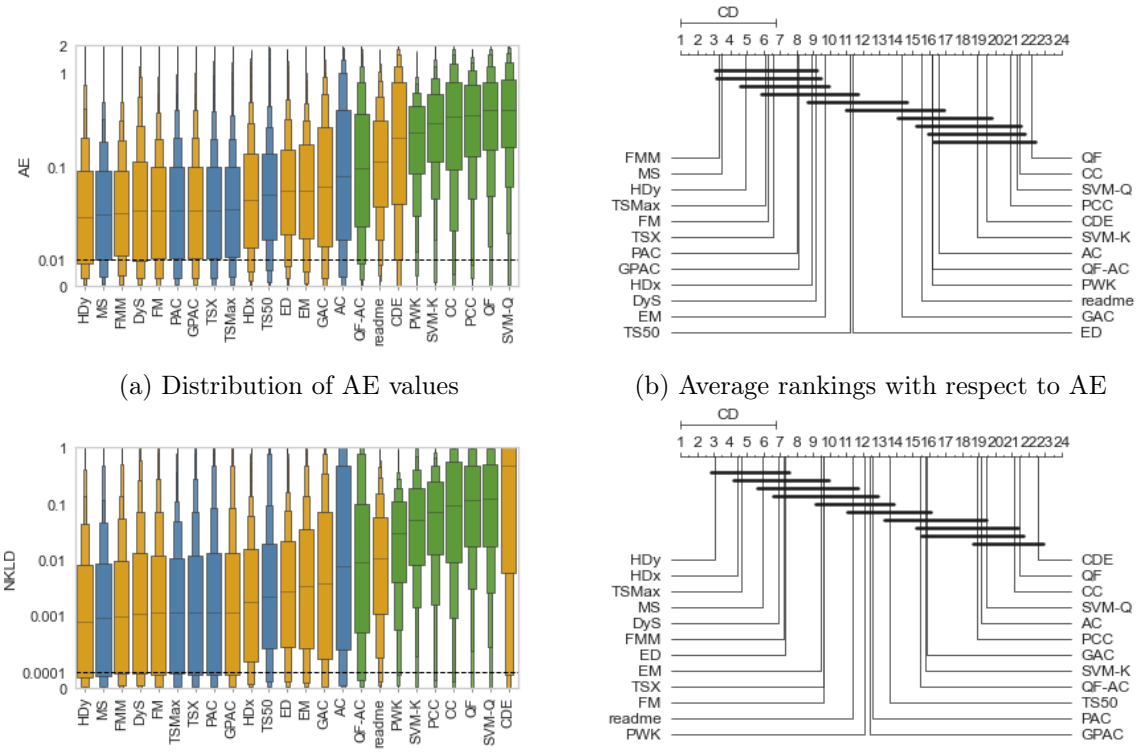
Regarding the actual comparison of the given quantifiers, we adapted a statistical procedure established by Demšar (2006), who, in the context of classification, suggested to conduct comparisons of multiple algorithms by statistical tests in a two-step approach that is based on the performance rankings of all algorithms considered with respect to a number of datasets they were applied on.

Within that two-step approach, at first a *Friedman test* (Friedman, 1940) is conducted on the null hypothesis that all algorithms perform equally well over a given set of datasets with respect to a chosen error measure. If that null hypothesis is rejected, one may follow up with the *Nemenyi post-hoc test* (Nemenyi, 1963) to compare the performance rankings of each algorithm per dataset with each other and determine which algorithms differ from each other in a statistically significant way. The margin of statistical significance is modeled by the critical distance value, which is determined by both the number of algorithms and datasets that are considered as well as the chosen significance level α.

While in classification, the underlying rankings would usually be obtained based on a cross-validated accuracy score, in our context, we averaged the quantification errors obtained from all the settings in our protocol over each dataset. Based on these average errors, for each dataset, we then determined a ranking of our algorithms for this dataset. To account for outliers, we also averaged the resulting scores via the mean and not the median value, which, by design of this measure, became more noticeable for NKLD.

5. Results

This section presents the results of our extensive experimental evaluation for binary quantification (i.e., labels with exactly two values) and multiclass quantification (i.e., labels with more than two values). For both types, we start by showing the main results that aggregate the performance of each algorithm across all datasets and settings. Then, we present detailed results for more distinct scenarios, namely different shifts (differences between training and test distributions) and varying amounts of training data. Finally, we compare the performance of all algorithms under study in the multiclass case, which is a setting that has not received much attention yet.



(a) Distribution of AE values

(b) Average rankings with respect to AE

(c) Distribution of NKLD values

(d) Average rankings with respect to NKLD

Figure 1: Visual representation of the main results for binary quantification. The top row shows results with respect to absolute error (AE), the bottom row for normalized Kullback-Leibler divergence (NKLD) values. On the left, letter-value plots for the distribution of error score across all scenarios per algorithm are shown. Colors indicate the category of the algorithm, with count adaptation-based algorithms shown in blue, distribution matching methods in orange, and adaptations of traditional classification algorithms in green. Plots are scaled logarithmically above the dotted vertical threshold, and linearly below. On the right, we plot the distributions of rankings with a Nemenyi post-hoc test at 5% significance. For each algorithm, we depict the average performance rank across all datasets. Horizontal bars indicate which average rankings do not differ to a degree that is statistically significant. The critical difference (CD) was 5.6973. Overall, the *HDy*, *MS*, *FMM*, and *DyS* methods appear to work best in general.

5.1 Binary Quantification

We first describe our results for binary quantification, that is, quantification with binary class labels.

5.1.1 Overall Results

We show the general performance results of all quantification algorithms across all datasets in Figure 1 and Table 4. The letter-value plots in Figures 1a) and 1c) represent the respective distributions of absolute error (AE) and normalized Kullback-Leibler divergence (NKLD) scores resulting from all experiments. The colors in the graph indicate the categories of the algorithms, i.e., *adjusted count* adaptation-based algorithms are shown in blue, distribution matching methods in orange, and adaptations of traditional classification algorithms are shown in green. The plots in 1b) and 1d) depict the average performance ranks of all algorithms per dataset along with the critical differences between the average ranks, which indicate whether the difference in the average ranks is statistically significant according to the Nemenyi post-hoc test (Demšar, 2006). Here, horizontal bars show which average rankings do not differ to a degree that is statistically significant. Tables 4a) and 4b) complement these graphs by providing average absolute errors (AE) and normalized Kullback-Leibler divergences (NKLD) for all scenarios per algorithm and dataset. Based on these averages, the rankings for the plots 1b) and 1d) have been compiled. Further, for each algorithm, a total average error score across all datasets is provided.

Overall, under both NKLD and AE, we observe substantial differences between the algorithms. While there is no single best algorithm for all cases, the results suggest that there is a group of algorithms that perform particularly well compared to the rest. First and foremost, the *HDy*, *MS*, *FMM*, *DyS*, and *FM* methods, in that order, appear to yield the best performances when considering the overall distributions of error scores with respect to both AE and NKLD.
When considering the aggregated rankings, these methods also tend to perform well, with the *FMM* and *MS* methods performing the strongest with respect to AE, and *HDy* performing strongest for NKLD. However, except for the *FM* method that falls off in the NKLD-based rankings, there is no statistically significant difference between these methods with respect to the Nemenyi post-hoc test.

Considering the overall distribution of error scores, the *PAC* and *GPAC* methods also appear to yield relatively robust performance over all datasets, but with respect to NKLD, these methods are significantly worse in their average rankings than the top-ranking *HDy* method. In addition, the *TSMax* method also appears among the top performing methods in the aggregated rankings, and the *HDx* method appears particularly strong in the NKLD-based rankings, although it does not stand out in the overall error distributions.

These general impressions are confirmed by Tables 4a) and 4b), where we see that the *FMM* and *HDy* algorithms take the top rank on most datasets with respect to AE, whereas for NKLD, the *HDy* method is most dominant. Considering the overall means in these tables, it is further notable that the *MS* method has the overall lowest average error with respect to AE, and *HDx* the lowest mean error with respect to NKLD, indicating a relatively high robustness against outliers.

	AC	PAC	TSX	TS50	TSMax	MS	GAC	GPAC	DyS	FMM	readme	HDx	HDy	FM	ED	EM	CDE	CC	PCC	PWK	SVM-K	SVM-Q	QF	QF-AC
adult	0.042	0.022	0.018	0.029	0.018	0.018	0.041	0.022	0.017	**0.013**	0.032	0.02	0.014	0.018	0.024	0.017	0.225	0.467	0.443	0.272	0.447	0.528	0.570	0.118
avila	0.56	0.086	0.079	0.081	0.071	0.069	0.459	0.086	0.186	0.066	0.086	**0.045**	0.074	0.096	0.075	0.214	0.899	0.852	0.682	0.286	0.765	0.849	0.678	0.327
bike	0.036	0.023	0.021	0.033	0.022	0.018	0.036	0.023	0.017	0.015	0.079	0.034	**0.014**	0.021	0.073	0.044	0.096	0.29	0.309	0.281	0.209	0.363	0.498	0.065
blog	0.072	0.036	0.034	0.034	0.031	0.03	0.072	0.036	0.030	0.029	0.042	**0.024**	0.029	0.033	0.055	0.042	0.569	0.643	0.575	0.394	0.387	0.654	0.625	0.213
bc-cat	0.23	0.112	0.077	0.137	0.079	**0.055**	0.193	0.112	0.121	0.056	0.276	0.109	0.083	0.062	0.093	0.207	0.315	0.38	0.390	0.091	0.304	0.753	0.315	0.151
bc-cont	0.133	0.072	0.051	0.130	0.049	0.042	0.117	0.072	0.106	0.048	0.121	0.103	0.056	**0.039**	0.052	0.125	0.123	0.172	0.245	0.058	0.167	0.838	0.249	0.133
cars	0.13	0.080	0.063	0.110	0.06	**0.049**	0.113	0.080	0.078	0.051	0.229	0.101	0.059	0.059	0.154	0.087	0.180	0.299	0.306	0.229	0.228	0.227	0.485	0.296
conc	0.533	0.171	0.154	0.190	0.144	**0.121**	0.369	0.171	0.175	0.125	0.258	0.172	0.178	0.155	0.184	0.336	0.745	0.699	0.608	0.300	0.304	0.601	0.627	0.275
contra	0.613	0.332	0.351	0.371	0.326	0.307	0.472	0.331	0.434	0.297	0.366	0.284	0.4	0.351	0.408	**0.249**	0.881	0.814	0.672	0.526	0.565	0.802	0.664	0.47
cappl	0.323	0.155	0.127	0.200	0.128	0.104	0.289	0.156	0.205	0.109	0.374	0.233	0.172	0.115	0.222	**0.087**	0.302	0.473	0.465	0.257	0.330	0.322	0.514	0.292
ccard	0.312	0.061	0.066	0.054	0.054	**0.044**	0.28	0.061	0.055	0.05	0.283	0.061	0.048	0.064	0.090	0.062	0.847	0.753	0.641	0.412	0.496	0.69	0.615	0.33
diam	0.315	0.037	0.032	0.045	0.032	0.031	0.27	0.038	0.037	0.027	0.063	**0.021**	0.027	0.031	0.085	0.217	0.746	0.709	0.609	0.323	0.627	0.853	0.497	0.27
dota	1.074	0.048	0.054	0.054	0.056	0.056	0.397	0.048	0.063	**0.047**	0.360	0.211	0.048	0.053	0.189	0.13	0.864	0.835	0.680	0.557	0.587	0.806	0.886	0.69
drugs	0.168	0.118	0.102	0.115	0.106	0.088	0.174	0.119	0.144	**0.080**	0.163	0.124	0.101	0.104	0.114	0.134	0.134	0.421	0.428	0.275	0.318	0.337	0.504	0.181
ener	0.271	0.040	0.040	0.048	0.041	0.037	0.224	0.040	0.041	**0.032**	0.207	0.073	0.034	0.041	0.067	0.12	0.699	0.672	0.596	0.270	0.399	0.742	0.741	0.5
fifa	0.76	0.035	0.030	0.036	0.029	0.028	0.055	0.035	0.027	0.023	0.137	0.025	**0.022**	0.03	0.040	0.031	0.204	0.461	0.447	0.329	1.240	1.188	0.418	0.056
flare	0.584	0.344	0.353	0.345	0.306	0.269	0.482	0.342	0.454	0.291	0.316	0.267	0.416	0.346	0.314	**0.256**	0.675	0.694	0.629	0.405	0.480	0.614	0.668	0.347
grid	0.09	0.046	0.046	0.052	0.052	0.038	0.086	0.046	0.042	0.035	0.149	0.07	**0.033**	0.044	0.058	0.048	0.258	0.492	0.468	0.225	0.749	0.668	0.782	0.51
ads	0.175	0.103	0.075	0.113	0.067	**0.054**	0.138	0.102	0.106	0.06	0.225	0.144	0.077	0.082	0.195	0.087	0.199	0.352	0.352	0.389	0.255	0.341	0.434	0.317
magic	0.271	0.043	0.042	0.052	0.045	0.041	0.236	0.043	0.043	0.039	0.103	0.056	**0.038**	0.044	0.044	0.057	0.469	0.606	0.542	0.314	0.607	0.587	0.661	0.477
boone	0.013	0.009	0.007	0.015	0.007	0.009	0.013	0.009	0.007	0.007	0.087	0.008	**0.006**	0.007	0.011	0.024	0.069	0.282	0.307	0.133	0.257	0.693	0.505	0.27
mush	0.014	0.011	0.008	0.048	0.009	**0.007**	0.014	0.011	0.014	0.016	0.033	0.018	**0.007**	0.008	0.027	0.017	0.009	0.027	0.054	0.009	0.098	0.054	0.215	0.021
music	0.547	0.324	0.327	0.346	0.299	0.272	0.462	0.324	0.429	0.283	0.682	0.479	0.371	0.328	0.404	**0.257**	0.840	0.748	0.651	0.449	0.465	0.572	0.741	0.609
musk	0.11	0.070	0.067	0.080	0.068	0.058	0.096	0.069	0.073	**0.053**	0.434	0.117	0.058	0.068	0.126	0.065	0.188	0.367	0.379	0.276	0.248	0.321	0.489	0.177
news	0.346	0.052	0.053	0.053	0.057	0.048	0.243	0.052	0.057	**0.046**	0.433	0.087	0.05	0.053	0.089	0.058	0.866	0.772	0.651	0.470	0.475	0.842	0.842	0.57
nurse	**0.000**	0.002	0.007	0.351	0.007	0.061	**0.000**	0.002	0.004	0.123	0.448	0.008	0.001	**0.000**	0.024	0.02	0.002	**0.000**	0.024	0.128	0.001	**0.000**	0.065	0.005
occup	0.04	0.017	0.006	0.057	**0.005**	0.006	0.034	0.017	0.007	0.021	0.020	0.01	**0.005**	0.006	0.012	0.103	0.098	0.125	0.192	0.015	0.241	0.531	0.113	0.012
phish	0.821	0.023	0.020	0.037	0.021	0.018	0.029	0.023	0.020	0.016	0.058	0.021	0.015	0.019	0.033	**0.014**	0.026	0.188	0.212	0.137	0.188	0.153	0.364	0.042
craft	0.248	0.084	0.065	0.088	0.075	0.058	0.219	0.084	0.082	**0.053**	0.211	0.069	0.058	0.067	0.070	0.144	0.528	0.602	0.543	0.319	0.344	0.684	0.568	0.222
spam	0.274	0.069	0.047	0.071	0.05	0.043	0.236	0.069	0.072	**0.041**	0.177	0.168	0.042	0.047	0.082	0.265	0.603	0.595	0.537	0.204	0.261	0.638	0.667	0.298
alco	0.48	0.328	0.341	0.366	0.3	**0.277**	0.451	0.337	0.431	0.282	0.584	0.415	0.36	0.342	0.468	0.296	0.695	0.693	0.625	0.491	0.495	0.608	0.653	0.495
study	0.347	0.187	0.201	0.215	0.194	0.161	0.301	0.187	0.233	0.162	0.287	0.151	0.194	0.192	0.308	0.175	0.533	0.589	0.538	0.330	0.610	0.696	0.460	**0.145**
cond	0.04	0.018	0.017	0.034	0.017	0.015	0.04	0.018	0.015	0.014	0.090	0.017	**0.013**	0.017	0.019	0.022	0.097	0.319	0.317	0.124	0.206	0.287	0.399	0.069
telco	0.224	0.075	0.071	0.080	0.069	0.06	0.211	0.075	0.075	**0.056**	0.097	**0.056**	0.059	0.07	0.065	0.059	0.401	0.571	0.525	0.387	0.373	0.476	0.609	0.304
thrm	0.612	0.318	0.320	0.355	0.298	0.272	0.462	0.318	0.423	0.291	0.534	0.348	0.358	0.309	0.355	**0.266**	0.861	0.773	0.655	0.444	0.491	0.629	0.698	0.495
turk	0.619	0.248	0.282	0.283	0.24	0.239	0.477	0.246	0.303	0.219	0.351	0.258	0.281	0.28	0.211	**0.164**	0.881	0.847	0.684	0.529	0.558	0.64	0.844	0.702
vgame	0.209	0.085	0.088	0.086	0.091	0.076	0.209	0.085	0.090	0.075	0.201	0.13	0.084	0.089	0.266	**0.066**	0.586	0.631	0.570	0.400	0.407	0.594	0.654	0.302
voice	0.15	0.048	0.035	0.060	0.032	0.034	0.134	0.047	0.037	0.038	0.210	0.045	**0.030**	0.036	0.060	0.178	0.289	0.346	0.378	0.076	0.166	0.417	0.246	0.051
wine	0.479	0.095	0.091	0.093	0.096	0.081	0.372	0.095	0.140	**0.079**	0.198	0.123	0.096	0.102	0.162	0.233	0.815	0.75	0.637	0.350	0.662	0.905	0.649	0.319
yeast	0.681	0.238	0.276	0.306	0.234	**0.212**	0.471	0.241	0.338	0.221	0.343	0.386	0.273	0.261	0.246	0.38	0.873	0.839	0.680	0.428	0.569	0.881	0.672	0.45
Mean	0.324	0.107	0.104	0.131	0.097	**0.088**	0.224	0.107	0.131	0.09	0.234	0.127	0.107	0.102	0.139	0.134	0.467	0.529	0.481	0.297	0.414	0.585	0.547	0.29

(a) Absolute error values

	AC	PAC	TSX	TS50	TSMax	MS	GAC	GPAC	DyS	FMM	readme	HDx	HDy	FM	ED	EM	CDE	CC	PCC	PWK	SVM-K	SVM-Q	QF	QF-AC
adult	0.018	0.005	0.003	0.005	0.002	0.002	0.017	0.005	**0.001**	**0.001**	0.003	0.002	**0.001**	0.002	0.003	**0.001**	0.200	0.175	0.139	0.061	0.129	0.184	0.294	0.052
avila	0.513	0.061	0.037	0.031	0.022	0.026	0.256	0.06	0.097	0.026	0.013	**0.005**	0.015	0.056	0.016	0.156	0.850	0.642	0.292	0.063	0.335	0.593	0.395	0.238
bike	0.013	0.007	0.005	0.007	0.003	0.003	0.013	0.007	0.002	0.003	0.012	0.004	**0.001**	0.005	0.012	0.007	0.100	0.073	0.077	0.063	0.044	0.116	0.215	0.013
blog	0.035	0.01	0.009	0.008	0.005	0.006	0.035	0.009	0.004	0.006	0.004	**0.002**	**0.002**	0.009	0.007	0.004	0.575	0.305	0.214	0.103	0.102	0.333	0.375	0.074
bc-cat	0.161	0.065	0.024	0.088	0.017	0.015	0.089	0.065	0.038	0.016	0.075	0.022	0.018	0.023	0.017	0.16	0.409	0.182	0.123	**0.013**	0.08	0.316	0.083	0.033
bc-cont	0.084	0.04	0.013	0.081	0.01	0.019	0.052	0.04	0.022	0.024	0.022	0.017	0.007	0.015	0.008	0.087	0.184	0.067	0.060	**0.006**	0.035	0.447	0.061	0.028
cars	0.074	0.051	0.028	0.057	0.016	0.019	0.051	0.049	0.021	0.019	0.053	0.018	**0.013**	0.03	0.032	0.034	0.212	0.099	0.083	0.046	0.051	0.045	0.317	0.265
conc	0.459	0.13	0.089	0.125	0.052	0.060	0.156	0.13	0.077	0.067	0.066	**0.039**	0.07	0.091	0.045	0.325	0.799	0.495	0.245	0.072	0.074	0.306	0.324	0.132
contra	0.537	0.247	0.258	0.271	0.175	0.172	0.242	0.247	0.245	0.199	0.107	**0.085**	0.203	0.26	0.16	0.125	0.843	0.581	0.286	0.166	0.197	0.382	0.392	0.286
cappl	0.238	0.093	0.061	0.128	**0.036**	0.040	0.156	0.095	0.075	0.045	0.110	0.057	0.057	0.054	0.053	0.037	0.415	0.244	0.159	0.056	0.093	0.086	0.192	0.082
ccard	0.206	0.029	0.026	0.021	0.01	0.012	0.164	0.029	0.008	0.015	0.074	0.008	**0.006**	0.023	0.015	0.015	0.835	0.447	0.260	0.109	0.16	0.386	0.267	0.093
diam	0.285	0.015	0.011	0.012	0.007	0.009	0.139	0.015	0.008	0.009	0.008	**0.002**	0.003	0.011	0.013	0.185	0.811	0.474	0.244	0.082	0.309	0.665	0.322	0.261
dota	0.877	0.015	0.018	0.019	0.013	0.013	0.2	0.015	0.013	0.015	0.104	0.049	**0.009**	0.017	0.042	0.024	0.843	0.583	0.289	0.189	0.225	0.499	0.685	0.475
drugs	0.093	0.057	0.041	0.059	0.025	0.031	0.093	0.057	0.037	0.028	0.032	0.022	**0.019**	0.044	0.02	0.022	0.094	0.144	0.134	0.059	0.078	0.088	0.178	0.043
ener	0.22	0.017	0.012	0.013	0.007	0.011	0.125	0.017	0.009	0.009	0.047	0.011	**0.004**	0.013	0.013	0.066	0.803	0.409	0.232	0.06	0.112	0.376	0.520	0.429
fifa	0.779	0.013	0.007	0.010	0.005	0.008	0.028	0.013	0.004	0.005	0.025	0.003	**0.002**	0.008	0.005	**0.002**	0.256	0.164	0.141	0.081	0.981	0.953	0.132	0.007
flare	0.436	0.247	0.251	0.259	0.152	0.151	0.296	0.244	0.217	0.178	0.087	0.082	0.192	0.234	0.097	**0.081**	0.711	0.42	0.256	0.11	0.159	0.243	0.324	0.138
grid	0.041	0.015	0.009	0.010	0.007	0.007	0.034	0.015	0.005	0.005	0.028	0.009	**0.002**	0.009	0.008	0.014	0.414	0.188	0.151	0.045	0.596	0.425	0.554	0.432
ads	0.112	0.074	0.035	0.070	**0.016**	0.021	0.078	0.074	0.033	0.024	0.052	0.027	0.018	0.039	0.044	0.027	0.187	0.134	0.108	0.134	0.071	0.107	0.152	0.101
magic	0.252	0.009	0.007	0.009	0.006	0.007	0.101	0.009	0.006	0.006	0.016	0.007	**0.005**	0.008	0.006	0.014	0.528	0.378	0.197	0.077	0.371	0.359	0.470	0.408
boone	0.002	0.001	**0.000**	0.001	**0.000**	0.001	0.002	0.001	**0.000**	0.001	0.013	**0.000**	**0.000**	**0.000**	0.001	0.001	0.027	0.07	0.075	0.023	0.058	0.463	0.325	0.261
mush	0.002	0.001	0.001	0.012	0.001	0.001	0.002	0.001	0.001	0.004	0.004	0.001	**0.000**	0.001	0.004	0.001	0.004	0.003	0.006	0.001	0.016	0.007	0.041	0.002
music	0.435	0.248	0.223	0.242	0.147	0.142	0.258	0.248	0.207	0.172	0.291	0.174	0.168	0.224	0.134	**0.082**	0.829	0.474	0.270	0.131	0.136	0.204	0.474	0.390
musk	0.057	0.029	0.029	0.036	0.016	0.019	0.045	0.028	0.017	0.016	0.143	0.02	**0.007**	0.028	0.023	0.011	0.198	0.116	0.109	0.062	0.049	0.087	0.187	0.042
news	0.254	0.016	0.015	0.016	0.011	0.009	0.107	0.015	0.01	0.011	0.139	0.013	**0.006**	0.015	0.013	**0.006**	0.843	0.484	0.268	0.134	0.144	0.466	0.625	0.480
nurse	**0.000**	**0.000**	0.006	0.325	0.006	0.044	**0.000**	**0.000**	**0.000**	0.107	0.129	**0.000**	**0.000**	**0.000**	0.002	0.005	0.010	**0.000**	0.002	0.017	**0.000**	**0.000**	0.017	0.001
occup	0.022	0.005	**0.000**	0.025	**0.000**	0.001	0.016	0.005	**0.000**	0.006	0.001	**0.000**	**0.000**	0.001	0.001	0.054	0.128	0.034	0.041	0.001	0.066	0.3	0.017	0.001
phish	0.451	0.004	0.004	0.007	0.003	0.003	0.007	0.004	0.002	0.003	0.007	0.001	**0.000**	0.004	0.004	**0.000**	0.002	0.034	0.041	0.022	0.032	0.027	0.108	0.013
craft	0.179	0.049	0.02	0.042	0.014	0.020	0.106	0.049	0.021	0.018	0.048	0.01	**0.008**	0.027	0.013	0.089	0.733	0.318	0.199	0.076	0.09	0.306	0.311	0.151
spam	0.22	0.036	0.011	0.031	0.009	0.012	0.121	0.036	0.025	0.011	0.036	0.032	**0.004**	0.013	0.012	0.218	0.718	0.351	0.200	0.04	0.061	0.298	0.299	0.082
alco	0.365	0.254	0.259	0.280	0.155	0.159	0.279	0.26	0.207	0.192	0.225	0.137	0.176	0.262	0.164	**0.102**	0.783	0.392	0.254	0.147	0.167	0.238	0.319	0.200
study	0.264	0.115	0.106	0.129	0.071	0.069	0.145	0.115	0.095	0.078	0.072	**0.030**	0.075	0.103	0.088	0.084	0.689	0.337	0.202	0.082	0.213	0.283	0.166	0.047
cond	0.017	0.003	0.002	0.008	**0.001**	0.002	0.017	0.003	0.002	0.003	0.014	0.002	**0.001**	0.003	0.003	0.003	0.148	0.101	0.082	0.021	0.043	0.069	0.170	0.030
telco	0.152	0.038	0.032	0.035	0.015	0.024	0.12	0.04	0.016	0.021	0.015	0.011	0.011	0.032	0.012	**0.007**	0.532	0.284	0.186	0.104	0.099	0.151	0.379	0.274
thrm	0.505	0.252	0.235	0.267	0.169	0.174	0.224	0.251	0.222	0.2	0.200	**0.110**	0.183	0.221	0.129	0.191	0.837	0.534	0.275	0.129	0.164	0.295	0.401	0.269
turk	0.527	0.194	0.197	0.206	0.113	0.112	0.247	0.192	0.133	0.138	0.104	0.081	0.109	0.207	0.082	**0.048**	0.843	0.613	0.292	0.163	0.215	0.294	0.633	0.538
vgame	0.152	0.045	0.038	0.036	0.026	0.028	0.131	0.045	0.02	0.03	0.043	0.023	0.019	0.04	0.07	**0.013**	0.763	0.323	0.215	0.106	0.114	0.267	0.350	0.099
voice	0.107	0.025	0.013	0.021	0.006	0.010	0.067	0.024	0.006	0.014	0.048	0.004	**0.002**	0.014	0.01	0.121	0.467	0.153	0.113	0.009	0.032	0.183	0.052	0.006
wine	0.419	0.05	0.039	0.036	0.024	0.026	0.164	0.049	0.057	0.032	0.041	0.021	**0.020**	0.048	0.059	0.211	0.831	0.524	0.262	0.089	0.248	0.513	0.350	0.174
yeast	0.595	0.18	0.198	0.225	0.111	0.115	0.2	0.183	0.179	0.133	0.096	0.12	0.115	0.19	**0.079**	0.373	0.842	0.652	0.291	0.121	0.228	0.636	0.393	0.263
Mean	0.254	0.069	0.059	0.082	0.037	0.040	0.115	0.069	0.054	0.047	0.065	**0.032**	0.039	0.06	0.038	0.075	0.507	0.3	0.177	0.077	0.159	0.3	0.297	0.173

(b) Normalized Kullback-Leibler divergence values

Table 4: Main results for binary quantification. We show error scores averaged across all scenarios per algorithm and dataset, along with the total means per algorithm (last row). For absolute error, *MS* performs best. For normalized Kullback-Leibler divergence, *HDx* and *HDy* achieve the best results on the plurality of datasets.

When considering the performance of basic algorithms such as *(probabilistic) classify and count* and *adjusted count*, we observe that these baselines are clearly outperformed by the top algorithms. Moreover, all algorithms that we have categorized as *classifiers for quantification*, and also the *CDE iterator* consistently show the worst performances with respect to both measures.

5.1.2 Influence of Distribution Shift

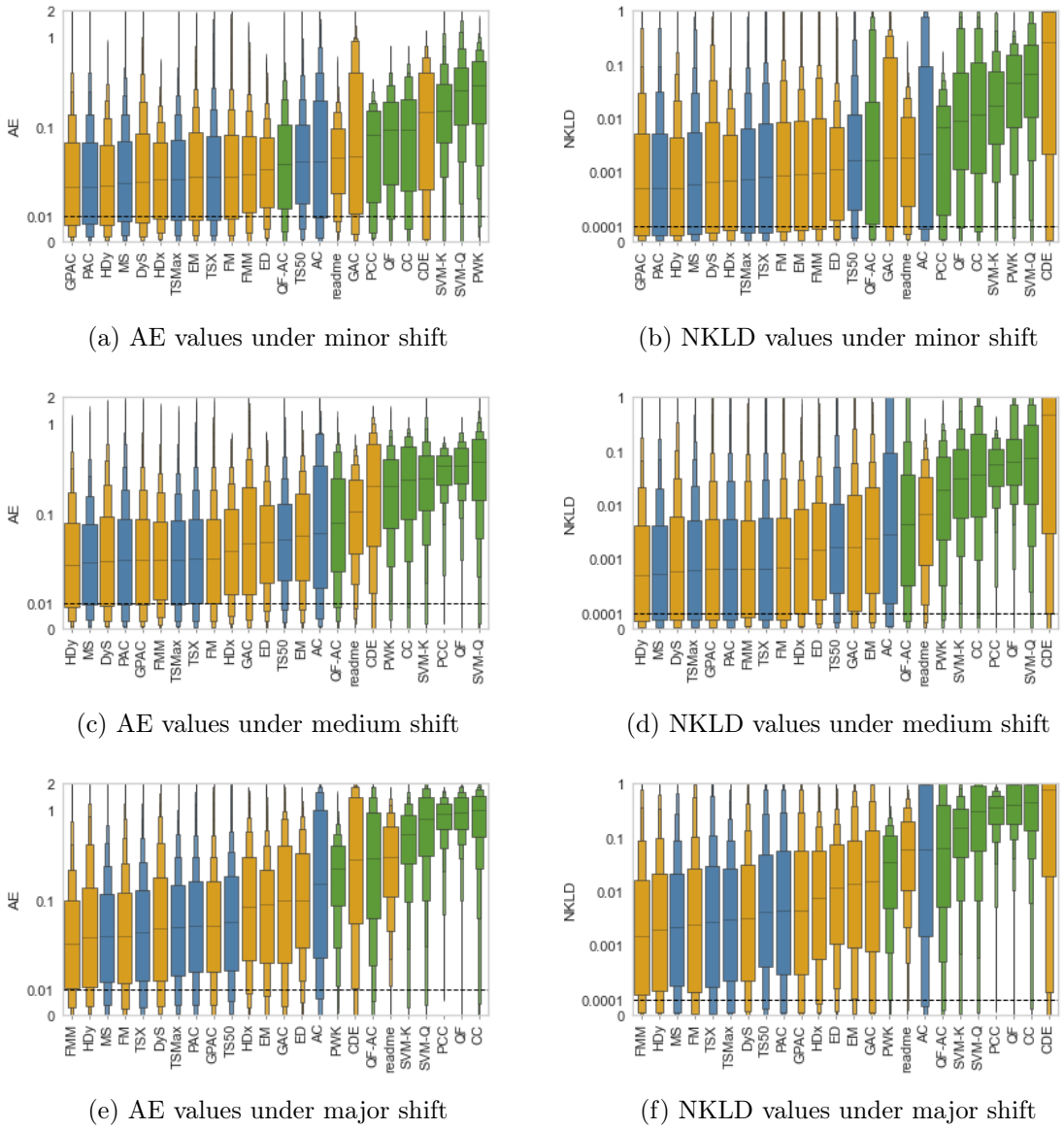
In the context of quantification, a shift in the distribution of the class labels Y between the training and the test set is assumed. It could be expected that the severity of the distribution shift affects the difficulty of the quantification task, as we assume that stronger shifts make accurate quantification more challenging. For that reason, we now take a closer look at the impact of this distribution shift to find out which methods are more or less sensitive to the severity of a distribution shift. In that context, we categorize all settings into three scenarios, namely a minor shift, a medium shift, and a major shift in these distributions. More precisely, we consider the shift to be

- minor, if the distribution shift is lower than 0.4 in L_1 distance,
- medium, if the distribution shift is bigger or equal to 0.4 and lower than 0.8 in L_1 distance,
- major, if the distribution shift is bigger or equal to 0.8 in L_1 distance.

We show the aggregated performance of the quantification algorithms under these three kinds of shifts in Figure 2. Unsurprisingly, we can observe that the performance of all quantification algorithms generally deteriorates with increasing shifts in class distributions. In that regard, the effect appears to be the strongest for classification-based approaches, in particular for the *quantification forests* and the *PCC* method. The only exception to this principle appears to be the *PWK* quantifier, which with respect to NKLD appears relatively robust toward distribution shift. Furthermore, the *readme*, *PAC* and *GPAC* methods also appear strongly affected by the increasing distribution shift, which is exemplified by the drop in their average rankings per dataset (cf. Appendix C.1, Figure 13). By contrast, the *HDy* and *FMM* methods appear the most robust to larger shifts.
For all other algorithms, except for the relatively robust *PWK* method, the decrease in performance appears to be between the aforementioned robust algorithms and the *classify and count*-based quantifiers, with their overall rankings appearing mostly unaffected from a distribution shift. That implies that even though the overall performance deteriorates, the same methods perform well, regardless of the amount of shift.

5.1.3 Influence of Training Set Size

Next, we consider the performance of quantification algorithms when relatively few training samples are given. For that purpose, we restrict the experimental data to only those cases in which the given data was split into 10% training samples and 90% test samples. The overall distribution of error scores with respect to AE and NKLD values can be found in Figure 3. We observe that, in general, the performance of all algorithms seems to be worse compared to the results when not being restricted to a small amount of training data, which is also to



(a) AE values under minor shift

(b) NKLD values under minor shift

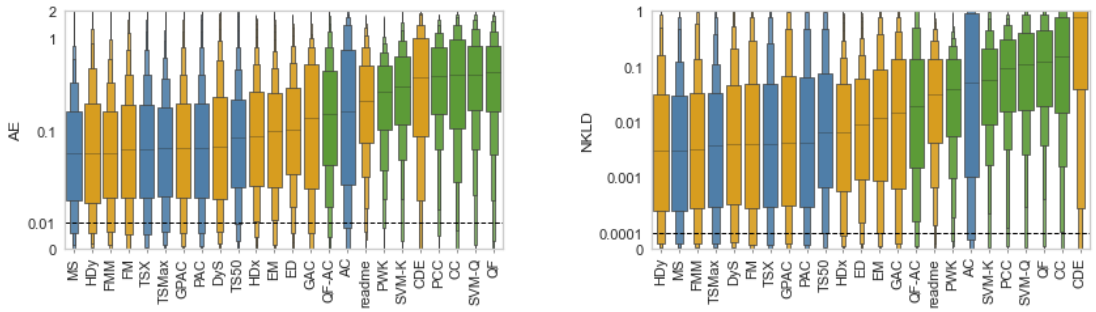
(c) AE values under medium shift

(d) NKLD values under medium shift

(e) AE values under major shift

(f) NKLD values under major shift

Figure 2: Impact of distribution shift in binary quantification. We show the distribution of error scores, split by severity of shift in the evaluation scenario. The left column shows results according to the absolute error (AE), the right one according to normalized Kullback-Leibler divergence (NKLD). Colors indicate the category of the algorithm. Plots are scaled logarithmically above the dotted vertical threshold, and linearly below. *GPAC* appears to perform best under minor shifts, *FMM* under major shifts.



(a) Distribution of AE values

(b) Distribution of NKLD values

Figure 3: Performance under small amounts of training data in binary quantification. Plot (a) shows results according to the absolute error (AE), plot (b) according to normalized Kullback-Leibler divergence (NKLD). Plots are scaled logarithmically above the dotted vertical threshold, and linearly below. Colors indicate the category of the algorithm. We observe similar trends compared to the general setting, with *MS*, *HDy*, and *FMM* being among the best-performing algorithms.

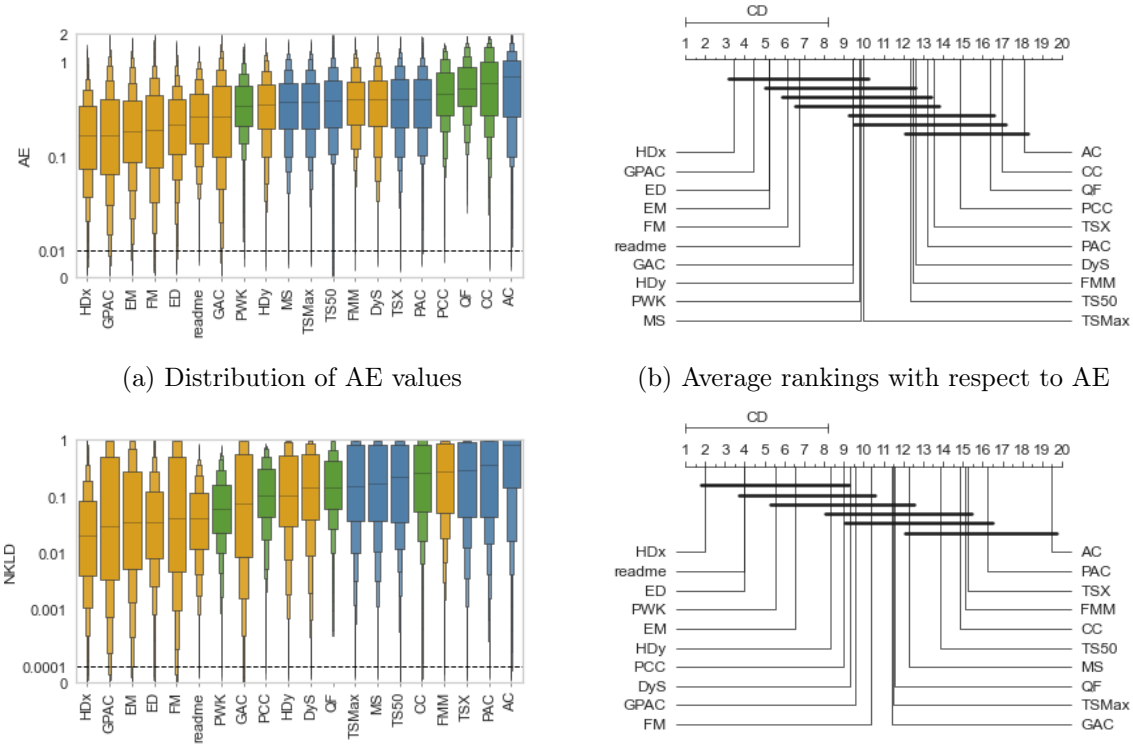
be expected. However, again the methods which yield the overall best performances, such as *MS*, *HDy*, and *FMM*, also appear to be the most robust toward this scenario. The average performance rankings of all algorithms per dataset (cf. Appendix C.1, Figure 14) are mostly in line with the general setting.

5.2 Multiclass Quantification

Next, we present results for multiclass quantification, that is, quantification for labels with more than two values.

5.2.1 Overall Results

Tables 5a) and 5b) as well as Figure 4 present the main results for multiclass quantification. Compared to the binary case, we obtain substantially different results. First of all, the overall prediction performance is much worse, as both AE values and NKLD values appear to be multiple times higher on average. For instance, AE values below 0.1 and NKLD values below 0.01 were widespread in the binary case, whereas in the multiclass case, such scores are only rarely achieved. Instead, the average AE values of each algorithm across all experiments are mostly around the interval [0.3,0.4], which is three to four times higher than the average AE values of the best algorithms in the binary case. The second main difference regards the algorithms that appear to work best: algorithms such as the *DyS* framework, the *median sweep (MS)*, and the other threshold selection policies, which have worked very well for binary quantification, appear comparatively weak in their performance. By contrast, the best performances seem to be achieved by distribution matching algorithms which also naturally extend to the multiclass setting, namely the *GPAC*, *ED*, *FM*, *EM*, *readme*, and *HDx* methods. In that context, the *HDx* method stands out. Furthermore, the *GPAC*,



(a) Distribution of AE values

(b) Average rankings with respect to AE

(c) Distribution of NKLD values

(d) Average rankings with respect to NKLD

Figure 4: Visual representation of the main results for multiclass quantification. The top row shows results for the absolute error (AE), the bottom row for normalized Kullback-Leibler divergence (NKLD) values. On the left, letter-value plots for the distribution of error score across all scenarios per algorithm are shown, colors indicate the category of the algorithm. Plots are scaled logarithmically above the dotted vertical threshold, and linearly below. On the right, we plot the distributions of rankings with a Nemenyi post-hoc test at 5% significance. For each algorithm, we depict the average performance rank across all datasets. Horizontal bars indicate which average rankings do not differ to a degree that is statistically significant. The critical difference (CD) was 7.0045. Overall, performance scores are much worse than in the binary setting. Best performances are generally achieved by distribution matching methods that naturally extend to the multiclass setting, with the *HDx* method standing out.

	AC	PAC	TSX	TS50	TSMax	MS	GAC	GPAC	DyS	FMM	readme	HDx	HDy	FM	ED	EM	CC	PCC	PWK	QF
bike	0.675	0.469	0.426	0.397	0.455	0.465	0.113	**0.073**	0.465	0.461	0.201	0.126	0.454	0.102	0.176	0.082	0.368	0.364	0.315	0.638
blog	0.795	0.671	0.594	0.585	0.565	0.557	0.360	0.236	0.533	0.580	0.180	**0.148**	0.541	0.285	0.29	0.196	0.588	0.500	0.422	0.547
conc	0.864	0.574	0.615	0.591	0.502	0.508	0.486	0.473	0.562	0.564	0.432	**0.380**	0.536	0.51	0.457	0.498	0.915	0.692	0.480	0.662
contra	0.829	0.483	0.496	0.508	0.466	0.462	0.600	0.515	0.538	0.467	0.424	**0.338**	0.481	0.512	0.434	0.396	0.833	0.699	0.572	0.675
diam	0.399	0.232	0.272	0.251	0.244	0.241	0.197	0.098	0.251	0.254	0.117	**0.044**	0.207	0.118	0.209	0.214	0.784	0.645	0.404	0.501
drugs	0.228	0.166	0.170	0.177	0.171	**0.147**	0.256	0.199	0.213	0.160	0.338	0.203	0.180	0.181	0.238	0.218	0.465	0.482	0.407	0.600
ener	0.634	0.354	0.354	0.322	0.351	0.346	0.273	**0.115**	0.337	0.366	0.331	0.178	0.347	0.129	0.169	0.131	0.879	0.699	0.439	0.925
fifa	0.838	0.656	0.616	0.615	0.567	0.564	0.313	0.181	0.581	0.599	0.221	**0.126**	0.525	0.216	0.278	0.127	0.481	0.441	0.384	0.432
news	0.825	0.581	0.548	0.541	0.522	0.523	0.498	0.335	0.535	0.545	0.446	0.237	0.522	0.376	0.245	**0.221**	0.827	0.614	0.471	0.917
nurse	0.077	0.104	0.064	0.159	0.068	0.082	0.023	**0.019**	0.047	0.203	0.263	0.034	0.047	0.02	0.049	0.022	0.138	0.173	0.213	0.399
craft	0.560	0.525	0.515	0.488	0.474	0.464	0.296	**0.190**	0.494	0.531	0.412	0.228	0.475	**0.190**	0.274	0.191	0.752	0.654	0.442	0.763
cond	0.541	0.442	0.479	0.353	0.500	0.485	0.155	0.066	0.456	0.516	0.129	0.077	0.469	0.088	0.093	**0.059**	0.343	0.362	0.213	0.431
thrm	1.297	0.633	0.726	0.684	0.593	0.587	0.780	0.629	0.694	0.619	0.471	**0.441**	0.634	0.663	0.47	0.494	1.042	0.769	0.511	0.827
turk	0.651	0.326	0.375	0.392	0.349	0.348	0.525	0.342	0.455	0.324	0.489	0.421	0.372	0.392	0.356	**0.277**	0.976	0.727	0.622	0.834
vgame	0.741	0.640	0.630	0.626	0.574	0.575	0.520	0.46	0.557	0.600	0.364	0.334	0.521	0.474	0.424	**0.322**	0.590	0.520	0.418	0.589
wine	1.061	0.706	0.700	0.693	0.595	0.607	0.656	0.575	0.719	0.637	0.428	**0.416**	0.546	0.605	0.44	0.757	0.965	0.636	0.496	0.613
yeast	1.015	0.541	0.518	0.487	0.446	0.464	0.567	0.408	0.527	0.505	0.474	0.342	0.412	0.413	**0.289**	0.613	0.878	0.612	0.295	0.526
Mean	0.708	0.477	0.476	0.463	0.438	0.437	0.389	0.289	0.468	0.466	0.336	**0.240**	0.428	0.31	0.288	0.284	0.696	0.564	0.418	0.640

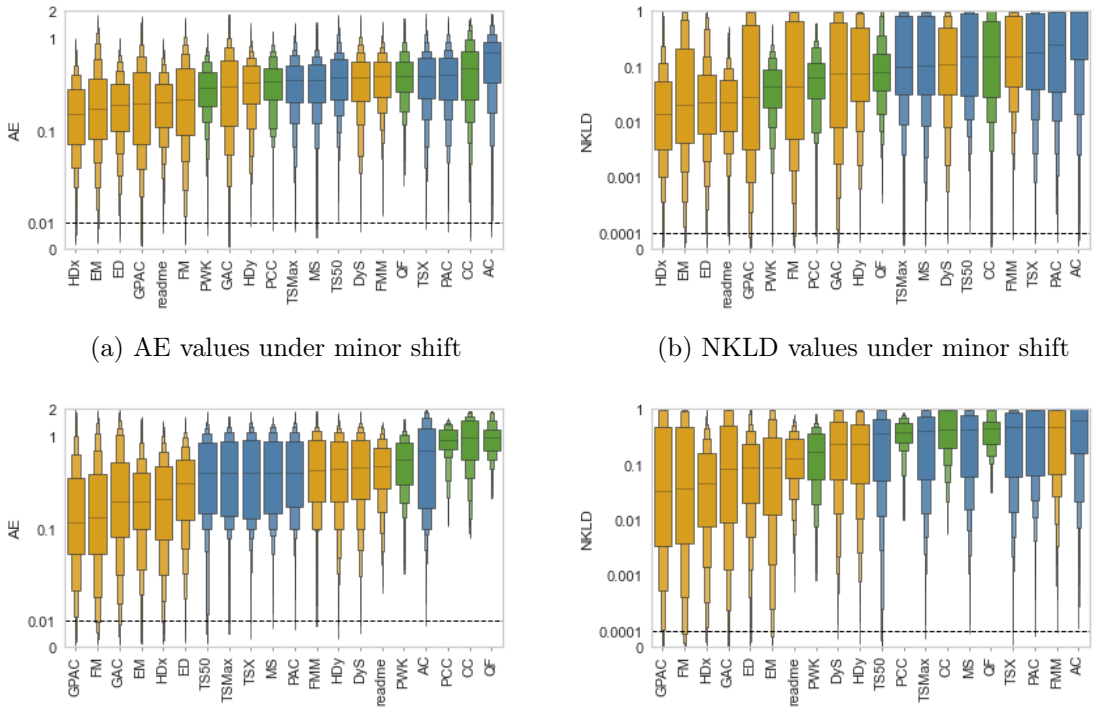
(a) Absolute error values

	AC	PAC	TSX	TS50	TSMax	MS	GAC	GPAC	DyS	FMM	readme	HDx	HDy	FM	ED	EM	CC	PCC	PWK	QF
bike	0.657	0.378	0.296	0.331	0.305	0.303	0.045	**0.016**	0.266	0.351	0.05	0.026	0.282	0.032	0.045	**0.016**	0.116	0.105	0.092	0.244
blog	0.707	0.822	0.658	0.656	0.642	0.648	0.402	0.201	0.463	0.673	0.04	**0.031**	0.565	0.243	0.113	0.044	0.315	0.155	0.135	0.206
conc	0.841	0.443	0.439	0.410	0.362	0.393	0.310	0.467	0.304	0.407	**0.126**	0.129	0.275	0.455	0.211	0.46	0.640	0.276	0.137	0.254
contra	0.662	0.425	0.412	0.433	0.333	0.350	0.448	0.469	0.312	0.395	0.131	**0.123**	0.275	0.445	0.214	0.237	0.464	0.280	0.179	0.258
diam	0.472	0.161	0.186	0.176	0.161	0.159	0.103	0.062	0.160	0.167	0.016	**0.003**	0.143	0.092	0.091	0.17	0.531	0.254	0.096	0.225
drugs	0.164	0.100	0.125	0.091	0.074	0.087	0.180	0.15	0.069	0.108	0.085	**0.039**	0.046	0.126	0.053	0.049	0.151	0.147	0.112	0.204
ener	0.598	0.383	0.366	0.327	0.330	0.337	0.137	0.085	0.222	0.390	0.086	**0.041**	0.264	0.084	0.050	0.087	0.491	0.270	0.12	0.527
fifa	0.761	0.790	0.660	0.594	0.621	0.623	0.316	0.115	0.476	0.652	0.049	**0.024**	0.489	0.152	0.099	0.029	0.254	0.126	0.115	0.129
news	0.751	0.456	0.398	0.396	0.358	0.363	0.539	0.318	0.316	0.389	0.143	0.068	0.337	0.400	0.076	**0.059**	0.524	0.227	0.16	0.608
nurse	0.060	0.063	0.008	0.018	0.007	0.038	0.011	0.005	0.003	0.189	0.055	0.002	0.002	0.007	0.005	**0.001**	0.025	0.033	0.049	0.115
craft	0.502	0.457	0.423	0.377	0.420	0.416	0.172	0.15	0.222	0.438	0.113	**0.052**	0.218	0.117	0.080	0.159	0.398	0.242	0.113	0.403
cond	0.652	0.525	0.515	0.382	0.496	0.493	0.089	0.011	0.301	0.524	0.022	0.009	0.330	0.027	0.018	**0.004**	0.166	0.098	0.044	0.130
thrm	0.969	0.608	0.729	0.706	0.530	0.533	0.605	0.648	0.517	0.641	**0.145**	0.214	0.502	0.723	0.248	0.442	0.692	0.340	0.151	0.382
turk	0.580	0.320	0.377	0.396	0.260	0.259	0.412	0.347	0.274	0.295	0.176	0.254	0.193	0.372	0.177	**0.105**	0.585	0.296	0.216	0.435
vgame	0.717	0.620	0.555	0.515	0.485	0.492	0.584	0.522	0.364	0.548	0.102	**0.098**	0.385	0.509	0.134	0.133	0.238	0.170	0.134	0.205
wine	0.810	0.714	0.690	0.665	0.521	0.552	0.434	0.62	0.537	0.620	**0.129**	0.185	0.410	0.617	0.278	0.781	0.714	0.240	0.157	0.221
yeast	0.817	0.598	0.580	0.502	0.485	0.519	0.358	0.431	0.479	0.593	0.143	0.105	0.342	0.401	0.115	0.702	0.585	0.224	**0.075**	0.173
Mean	0.631	0.463	0.436	0.410	0.376	0.386	0.303	0.272	0.311	0.434	0.095	**0.083**	0.298	0.283	0.118	0.205	0.405	0.205	0.123	0.278

(b) Normalized Kullback-Leibler divergence values

Table 5: Main results for multiclass quantification. We show error scores averaged across all scenarios per algorithm and dataset, along with the total means per algorithm (last row). Overall, distribution matching methods that naturally generalize to the multiclass setting appear to perform better than one-vs.-rest or *classify and count*-based approaches, with the *HDx* method appearing to stand out.

ED, *EM*, and *FM* methods show strong performance with respect to AE, whereas the *ED*, *readme*, and *EM*, but also the classification-based *PWK* method obtains high average rankings with respect to NKLD. These general trends are also confirmed in Tables 5a) and 5b), where the *HDx* method stands out with regard to both AE and NKLD. In addition, from the overall distributions of errors in Figures 4a) and 4c) it becomes apparent that these algorithms also have strong differences in the variance of their performance. In particular, the *GPAC* method appears to have a much higher variance in its error scores compared to the rest, while the *ED* and *readme* methods display the lowest variance in their performance. However, the given results also have one big commonality with the results from the binary setting, that is, all algorithms that are based on the *classify and count* principle display subpar performances, even when optimizing quantification-based loss functions.



(a) AE values under minor shift

(b) NKLD values under minor shift

(c) AE values under major shift

(d) NKLD values under major shift

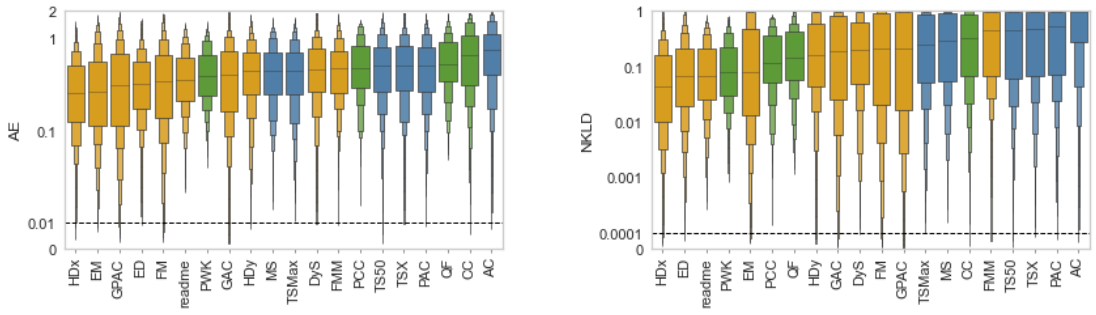
Figure 5: Impact of distribution shift in multiclass quantification. We show the distribution of error scores, split by severity of shift in the evaluation scenario. The left column shows results according to absolute errors (AE), the right one according to normalized Kullback-Leibler divergence (NKLD). Colors indicate the category of the algorithm. Plots are scaled logarithmically above the dotted vertical threshold, and linearly below. *GPAC* and *FM* appear most robust toward major shifts.

5.2.2 Impact of Distribution Shift

As in the binary case, we also investigate the effect that the shift of the distribution of the class labels Y between training and test sets has on the resulting quantification performance. Since we have less experimental data than in the binary case, here we distinguish only a minor shift and a major shift. We consider the shift to be

- minor, if the distribution shift is lower than 0.5 in L_1 distance,
- major, if the distribution shift is bigger or equal to 0.5 in L_1 distance.

The results of multiclass quantification under these scenarios are shown in Figure 5. Similarly to the binary case, we observe that the algorithms which appeared to work best in general also appear to be the most robust with respect to high distribution shifts. In particular, the *GPAC* method appears almost unaffected by a high shift in its average performance—it consistently achieves higher performance ranks with increasing shifts, although significant



(a) Distribution of AE values

(b) Distribution of NKLD values

Figure 6: Performance under small amounts of training data in the multiclass setting. Plot (a) shows results according to the absolute error (AE), plot (b) according to normalized Kullback-Leibler divergence (NKLD). Plots are scaled logarithmically above the dotted vertical threshold, and linearly below. Colors indicate the category of the algorithm. Overall trends are similar to the general setting, although in particular the *GPAC* method deteriorates with respect to NKLD.

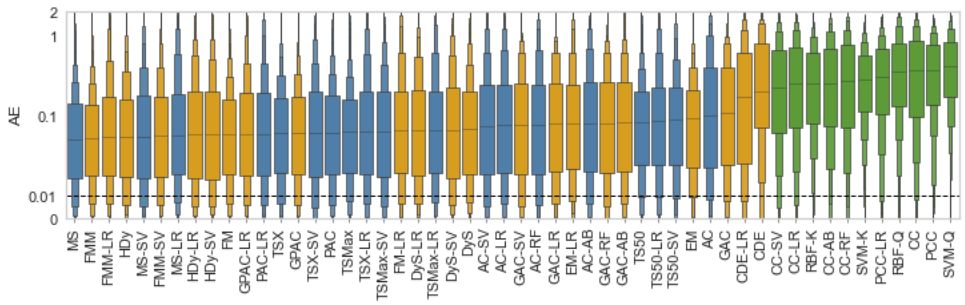
variance can be observed in its performance. By contrast, all methods which apply the *classify and count* principle are again the most susceptible to higher error rates when applied in scenarios with higher shifts between training and test distribution.

5.2.3 Influence of Training Set Size

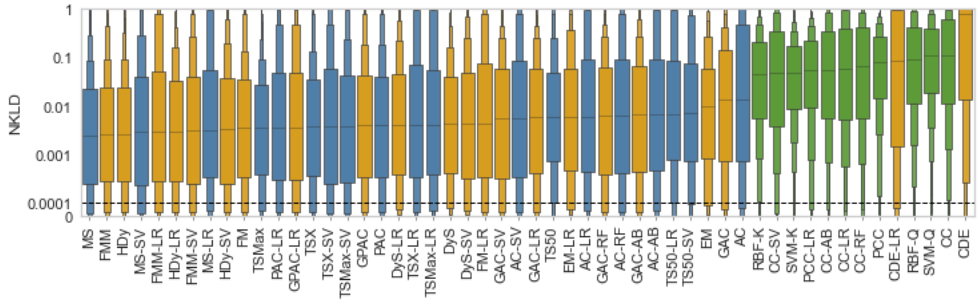
Finally, we consider the performance of the given algorithms when the given data was split into 10% training samples and 90% test samples. As before, this serves to investigate the impact of having a relatively small set of training data. The distributions of error scores with respect to the AE and NKLD measures can be found in Figure 6. Compared to the distribution of error scores in the main experiment, the performance deteriorates when only small training sets are given. In particular, we observe that the *GPAC* is much less competitive than in the general scenario, particularly with respect to NKLD. Conversely, the *HDx*, *EM* and *ED* algorithms, and, with respect to NKLD, also the *readme* method appear to be most robust toward this setting—this latter result may be due to *readme* returning an average prediction of an ensemble, which makes it less likely to falsely predict class prevalences of 0 and obtain a high NKLD value in consequence. This implies that those algorithms could be recommended if only limited training data is available.

5.3 Impact of Alternative Classifiers and Tuning

We close this chapter by presenting the results of our experiments with quantifiers that applied tuned base classifiers. We begin with the results on binary data, before finishing with the results from the multiclass setting.



(a) Distribution of absolute error (AE) values

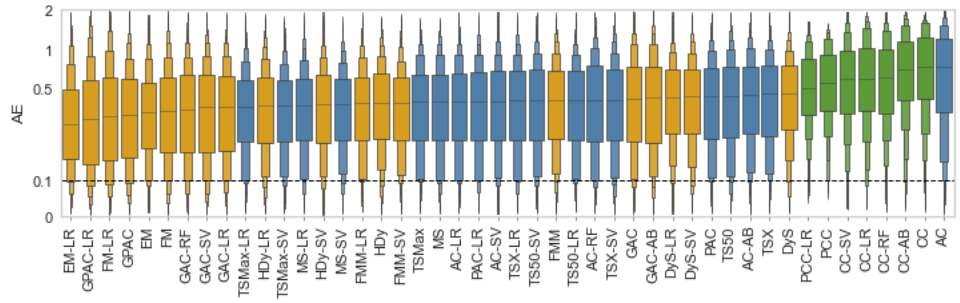


(b) Distribution of normalized Kullback-Leibler divergence (NKLD) values

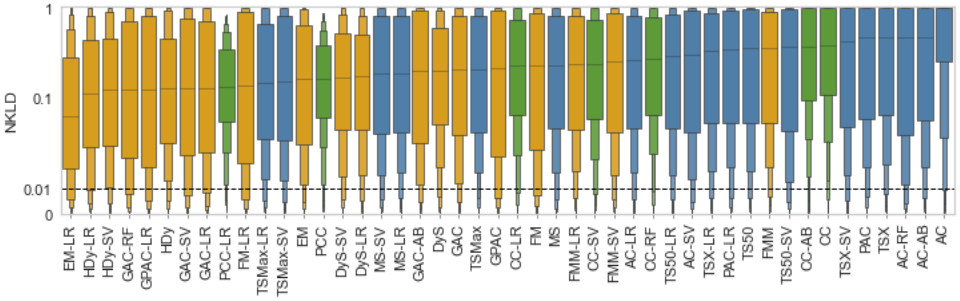
Figure 7: Results of our experiments in the binary setting, where base classifiers were tuned with respect to their accuracy. Plots are scaled logarithmically above the dotted vertical threshold, and linearly below. Colors indicate the category of the algorithm. Algorithms based on untuned logistic regression classifiers are denoted as before (no suffix), alternative tuned base classifiers are marked with respective suffixes: logistic regressors (LR), support vector machines (SV), random forests (RF) and AdaBoost (AB). We also show results of the *RBF-K* and *RBF-Q* methods, which are variants of the *SVM-K* and *SVM-Q* methods that use an RBF kernel instead of a linear one. Except for the *CC*, *PCC*, *GAC* and *CDE* methods, tuning base classifiers does not seem to have a consistently positive effect.

5.3.1 Experiments on Binary Data

In Figure 7, we show the scores of all quantifiers using different tuned base classifiers aggregated over all considered datasets, cf. Section 4.3.2. As a baseline, we also include the results from the quantifiers that apply the default logistic regressor. These results yield a few key findings. First, for most algorithms, tuning the base classifier does not seem to have a significant positive effect. Instead, for the best-performing algorithms *MS*, *TSX*, *FM*, and *TSMax*, the performance even appears to deteriorate. The few exceptions where tuned base classifiers appear to strongly benefit the predictions include the *CC*, *PCC*, *CDE*, *GAC* methods. While the first two directly apply the *classify and count* principle, where it can



(a) Distribution of absolute error (AE) values



(b) Distribution of normalized Kullback-Leibler divergence (NKLD) values

Figure 8: Results of our experiments with quantifiers that apply tuned classifiers in the multiclass setting. For natural multiclass quantifiers, base classifiers were tuned with respect to their accuracy. For one-vs.-rest-based quantifiers, the binary base classifiers were tuned with respect to their *balanced* accuracy. Plots are scaled logarithmically above the dotted vertical threshold, and linearly below. Colors indicate the category of the algorithm. Algorithms based on untuned logistic regression classifiers are denoted as before (no suffix), alternative tuned base classifiers are marked with respective suffixes: logistic regressors (LR), support vector machines (SV), random forests (RF) and AdaBoost (AB). We observe mostly positive effects from applying tuned base classifiers.

be expected that more accurate classification will yield more accurate quantification, the results for the *CDE* and *GAC* methods pose as outliers. It is also notable that the effects of parameter tuning often vary strongly over the given datasets, as can be seen in Tables 6, 7 and 8 in Appendix C.2. Specifically for the *PAC* and *GPAC* methods, strong fluctuations in performance across datasets can be observed, while their overall distribution of error scores, as depicted in Figure 7, appears quite robust. Regarding the *SVM-K* and *SVM-Q* methods, we observe that the application of the alternative RBF kernel appears to have a slight positive effect, but these *RBF-K* and *RBF-Q* variants still show inferior performance compared

to most other quantifiers, while at the same time coming at very high computational costs. In general, the given results also appear to be consistent across both AE and NKLD.

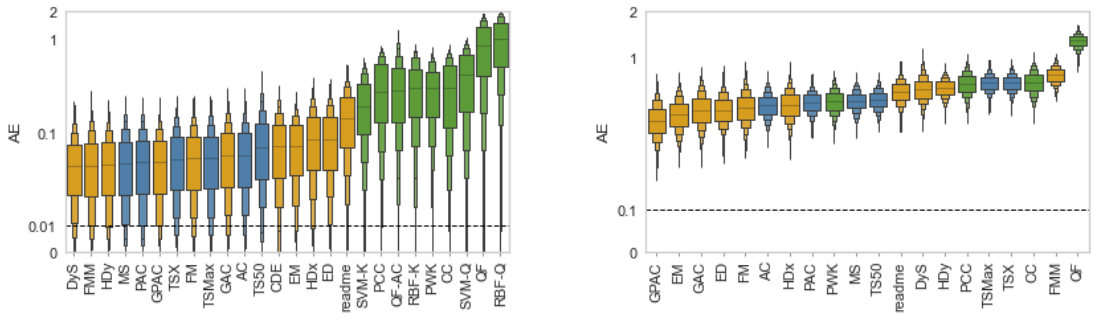
5.3.2 Experiments on Multiclass Data

The results of our experiments on quantification with tuned base classifiers in the multiclass setting can be found in Figure 8. In contrast to the binary setting, we observe that tuning the base classifiers appears to have a strong positive effect for almost every pair of quantifier and base classifier—the only base classifier for which the effect of tuning appears less consistent is the AdaBoost classifier. However, when also considering the average error scores per dataset in Table 9, this effect is not consistent across all datasets, but still yields a substantial improvement on aggregate. Further, only the probability-based *EM*, *GPAC*, and *FM* methods, in which the logistic base classifiers have been tuned, appear to outperform all default variants of the given quantifiers with respect to both AE and NKLD. The *EM algorithm* with a tuned logistic base classifier also appears to stand out overall with respect to both error scores.

6. A Case Study on the LeQua 2022 Challenge Data

To validate our findings in an external benchmark framework, we further conduct a case study on the datasets from the *LeQua 2022 challenge* (Esuli et al., 2022a,b). In this challenge, Esuli et al. (2022a) provided the participants with two textual datasets, one with binary labels and one with multiclass labels. Each was given both in raw document format and in a preprocessed numerical vector format—the preprocessed features were derived from the average *GloVe* (Pennington et al., 2014) embedding vectors of the words in each document, which were standardized to zero mean and unit variance. The data was collected from a large crawl of Amazon product reviews, where the binary labels were derived from the sentiment of the reviews, and the 28 labels in the multiclass task correspond to product categories. The challenge then consisted of two main tasks, where the first task was to perform quantification on the preprocessed datasets, and the second task was to evaluate the raw documents in an end-to-end fashion that could occur in practical scenarios. Both tasks were split into two subtasks in which (i) the binary and (ii) the multiclass versions of the dataset were to be analyzed.

In our case study, we only consider the preprocessed data from the first task, since preprocessing techniques for textual datasets are out of scope for this work, and differences in preprocessing may further hinder comparability of results. The binary and multiclass datasets are both split into training, validation, and test data. The class labels for each document are provided only for the training data, which consists of 5,000 documents in the binary setting and 20,000 documents in the multiclass setting. The validation sets consist of 1,000 samples of 250 (binary) and 1,000 (multiclass) documents each, where no class labels are given for any document, but the label distribution of each sample is known and can be used for model tuning. Finally, the test sets in the binary and the multiclass dataset contain 5,000 data samples, each consisting of 250 documents in the binary and 1,000 documents in the multiclass case. We note that the setting in this challenge specifically differs from the experimental settings in this work with the availability of large amounts of validation data, which has been separated from the relatively small amount of training data. In addition,



(a) AE values on the binary LeQua data

(b) AE values on the multiclass LeQua data

Figure 9: Results of our experiments with untuned quantifiers on the LeQua test sets. We present distributions of absolute error (AE) values across all test samples. Plots are scaled logarithmically above the dotted vertical threshold, and linearly below. Colors indicate the category of the algorithm. In addition to all quantifiers used in our main experiments, we present results of the *RBF-K* and *RBF-Q* methods, which are variants of the *SVM-K* and *SVM-Q* that use an RBF kernel instead of a linear kernel. Overall results are in line with our findings from the main experiments. On the binary data, the *DyS* and *FMM* methods appear to work best, on the multiclass data, the *GPAC* and *EM* methods appear to stand out.

the number of labels in the multiclass part of the challenge ($L = 28$) is significantly higher than the number of classes used in our experiments ($L = 5$).

On the LeQua dataset, we conducted three experiments. First, as in our main experiments, we applied all quantifiers using their default parameters. In the second experiment, we again considered all quantifiers that use a base classifier, and tuned the parameters of these classifiers with respect to their accuracy on the training data before applying the quantifiers with tuned base classifiers on the test data. In the third and final experiment, we explored the effects of tuning the parameters, including base classifiers, for quantification, making use of the given validation samples.

In the following, we describe the results from these experiments, focusing in particular on the results with respect to AE. Additional results with respect to NKLD are presented in Appendix E, where it can be seen that in the binary case, the results were mostly very similar. For the multiclass setting on this dataset, where the results differed more strongly from the AE-based results, we do not consider NKLD to be very suitable. This is due to NKLD specifically punishing cases where prevalences of classes are falsely estimated to be zero. Given that the multiclass dataset has $L = 28$ classes, very low prevalences of individual classes are, however, very frequent by nature and thus less of a concern.

6.1 Comparison of Quantifiers With Default Parameters

We begin with presenting the results from using quantifiers with their default parameters on the LeQua dataset—we used the same parameterization as in our main experiments, which

has been outlined in Section 4.3.1. All quantifiers have been trained on the given training data and directly applied on the test data without considering the validation samples. The only optimization performed was for the *HDx*, *readme*, and *QF* methods, which require binned input data. For these methods, we optimized the binning strategy by varying the number of bins that would be used for all features between 2 and 8, and by testing equidistant as well as quantile-based binning. The results that we report are based on the binning strategy that yielded the best average AE value on the validation sets.

The results of these experiments can be found in Figure 9, where we depict the distribution of AE values on the test datasets. Overall, these results appear to be in line with the findings of our main experiments. On the binary dataset, *DyS* and *MS* appear to work best, with methods such as *PAC*, *GPAC*, *TSX*, *FM*, and *TSMax* appearing relatively competitive, and *classify and count*-based methods, even when optimized for quantification, appearing to fall behind. On the multiclass datasets, specifically the *GPAC* and *EM* methods appear to stand out, and, overall, natural multiclass quantifiers seem to outperform one-vs.-rest approaches. As a notable difference to our main experiments, the *HDx* and *readme* methods appear to perform relatively weak overall. We suppose that this is due to these methods requiring binned inputs, for which we may not have found an optimal binning strategy. Although, as noted before, we have performed some optimization of the binning, more fine-grained optimization of bins, which could also include different strategies for different features, might be required.

6.2 Comparison of Quantifiers with Tuned Base Classifiers

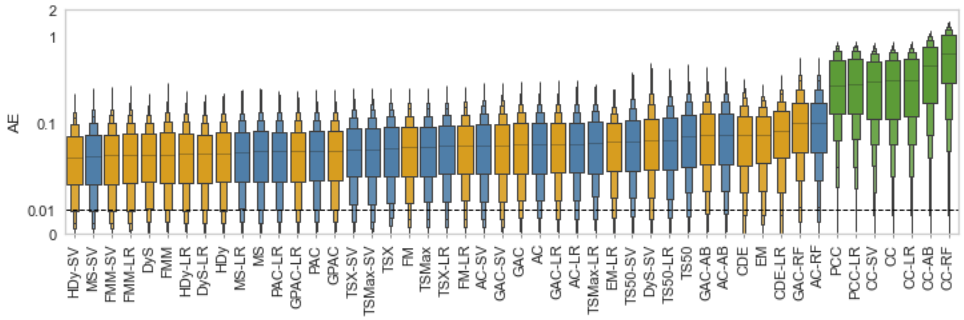
Next, we present the results from applying quantification methods for which the base classifiers have been tuned. We applied the same parameter grid as in previous experiments (cf. Section 4.3.2), and tuned the parameters on the training set via cross-validation to optimize their accuracy—since the validation data does not provide labels for individual documents, this data could not be used for tuning.

The AE values that we obtain from these experiments are depicted in Figure 10. On both binary and multiclass data, we generally see a mixed picture regarding the benefits of tuning the base classifier. Some methods, such as the *EM* and *GPAC* approaches, seem to improve particularly in the multiclass case, while other methods, such as the *classify and count*-based approaches, seem to deteriorate. However, there are no general trends for any group of algorithms, which is overall in line with the results from our main experiments.

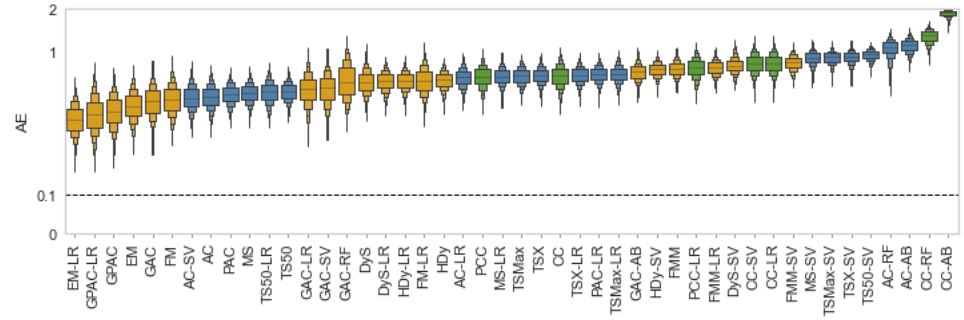
6.3 Comparison of Tuned Quantifiers

Finally, we discuss our results from the experiments in which we tuned the parameters of all quantification methods using the extensive validation data available within the LeQua dataset. Parameters were tuned with respect to AE on the validation data, and the optimization also considered parameters of the logistic regressor that was chosen as the base classifier for all quantifiers requiring a base classifier to form their predictions. A detailed overview of the parameter grids that we used can be found in Appendix D.

The distribution of the resulting AE values is shown in Figure 11, where we can see that tuning parameters appears to have a significant positive effect on the outcomes.



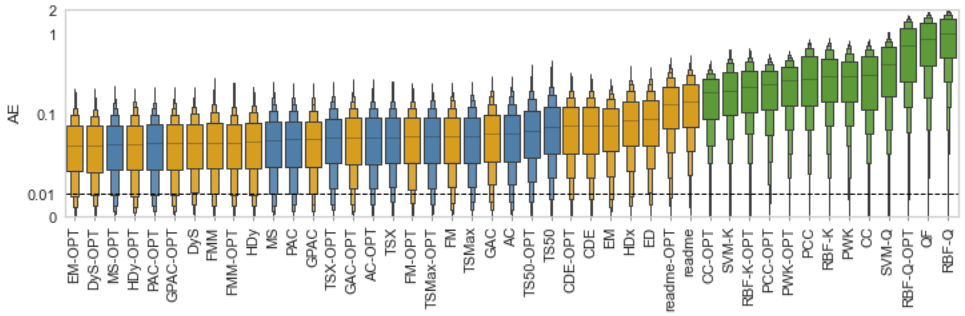
(a) Distribution of absolute error (AE) values on the binary LeQua test data



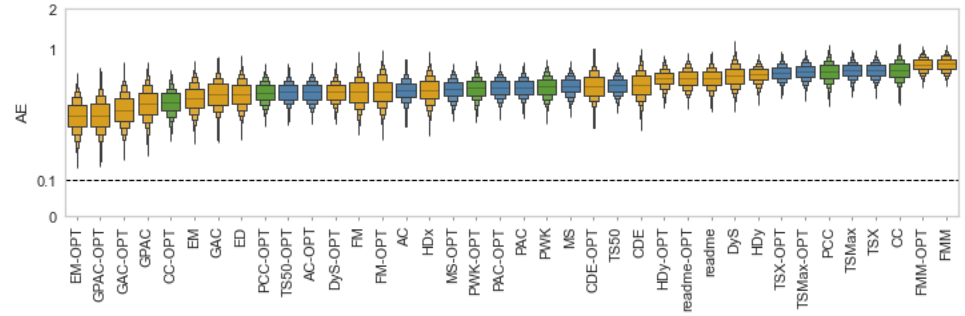
(b) Distribution of absolute error (AE) values on the multiclass LeQua test data

Figure 10: Results from applying quantifiers with tuned base classifiers on the LeQua data. In the binary setting and for natural multiclass quantifiers, base classifiers were optimized with respect to their accuracy. For quantifiers that apply the one-vs.-rest approach in the multiclass setting, the binary base classifiers were tuned with respect to *balanced* accuracy. Plots are scaled logarithmically above the dotted vertical threshold, and linearly below. Colors indicate the category of the algorithm. Algorithms based on untuned logistic regression classifiers are denoted as before (no suffix), alternative tuned base classifiers are marked with respective suffixes: logistic regressors (LR), support vector machines (SV), random forests (RF) and AdaBoost (AB). Overall, there appears to be no consistent positive effect from tuning base classifiers.

In the binary setting, the tuned *EM* and *DyS* methods perform best, with the tuned *MS*, *HDy*, *PAC* and *GPAC* methods only marginally behind. Interestingly, the untuned *DyS*, *MS*, *PAC*, and *GPAC* methods still appear to outperform the tuned variants of almost every other algorithm we considered. Further, it is notable that specifically the *EM* algorithm appears to benefit greatly from the parameter tuning. A strong positive impact can also be observed for all *classify and count*-based approaches, but, even after tuning, these methods perform worse than almost any other method with default parameters.



(a) Distribution of absolute error (AE) values on the binary LeQua test data



(b) Distribution of absolute error (AE) values on the multiclass LeQua test data

Figure 11: Results of our experiments on the LeQua test data using quantifiers that were tuned on the LeQua validation data. Plots are scaled logarithmically above the dotted vertical threshold, and linearly below. Colors indicate the category of the algorithm. In the binary setting, we include results of the *RBF-K* and *RBF-Q* methods, which are variants of the *SVM-K* and *SVM-Q* that use an RBF kernel instead of a linear kernel. Algorithms using their default parameters are denoted as before (no suffix), their tuned variants are marked with a short suffix (OPT). In both the binary and the multiclass task, the tuned *EM* algorithm appears to perform best.

In the multiclass case, we also observe significant improvements in the resulting error scores. In particular, the *EM* and *GPAC* methods appear to perform better than the rest, with *GAC* also showing strong results after tuning. The untuned versions of these algorithms further appear to outperform almost all other methods, even after tuning, with respect to AE. The only exception is the *CC* method, which performs surprisingly well on this dataset.

7. Discussion

Next, we discuss the main results and potential limitations of our study.

7.1 Discussion of Results

Our experiments yielded substantially different results for the binary case compared to the multiclass case, both in terms of overall quality of performance, and in terms of which algorithms performed best.

In the binary case, we identified a group of algorithms that appeared to work particularly well with respect to both AE and NKLD, namely *HDy*, *FMM*, *MS*, *TSMax*, *Friedman's method*, and the *DyS* framework. These methods stood out both in terms of their ranks and in terms of their overall error distribution (although *HDy* appears to have a slight edge over the rest in these distributions). Next to these algorithms, other methods have shown similarly strong performances, at least with respect to one of the two measures that were considered. In this regard, *TSX* has shown very strong performances with respect to AE, while the *ED* method appears to work particularly well with respect to NKLD. The strong performance of the *MS* and *TSMax* methods indicates that the simple idea behind the *adjusted count* approach, even when using a rather unsophisticated baseline classifier, can still yield very decent results, as long as numerical stability, i.e., a big denominator in Equation 1, is ensured. In that regard, the *MS* method also benefits from the policy that all thresholds, for which the denominator is below 0.25, are excluded. A similar argument can be made for the superiority of the *DyS* framework, which includes the *HDy* method, and Forman's mixture model (*FMM*) compared to other distribution matching methods that use predictions from classifiers. Specifically, the approach of binning confidence scores into more than just two classes, which ultimately adds more equations to the system in Equation 2, also appears to yield more robust results. By contrast, classifiers that optimize quantification-oriented loss functions also tended to show worse performance than the majority of other quantifiers. This is another strong indicator that pure classification without adjustments for potential distribution shifts does not perform well for quantification. The reason for this is that, under a shift in the class distribution, predictions are strongly biased toward the training distribution, as exemplified in our experiments. This practical outcome is also clearly in line with Forman's Theorem (Forman, 2008), which states that when a distribution shift is given, a bias in the *CC* estimates toward the training distribution is to be expected. This finding stands in contrast to a recent discussion of this kind of approach by Moreo and Sebastiani (2021), who have reassessed the performance of the *classify and count* approach and found that when doing careful optimization of hyperparameters, such quantification-oriented classification approaches would deliver near-state-of-the-art performance, although still inferior to methods such as *EM* or *HDy*. Our experimental results suggest that this type of approach should be used only carefully for quantification, as a vulnerability toward distribution shifts in theory as well as in experimental results can be clearly observed. Finally, the overall subpar performance of the *CDE iterator* is also in line with theoretical results that emphasized its lack of consistency (Tasche, 2017).

Considering the multiclass case, results are qualitatively different. Most notably, error scores were considerably higher than in the binary setting. Another key difference is that methods such as *HDy*, *DyS*, *MS*, or *TSMax*, which have excelled in binary quantification, only

showed mediocre performance in the multiclass case. By contrast, distribution matching methods that naturally extend to the multiclass setting appeared to work best, with the *HDx* method appearing to stand out. These results indicate that generalizing quantification methods to the multiclass case via a one-vs.-rest approach is not an optimal strategy for multiclass quantification. This finding has recently been taken up and analyzed more deeply by Donyavi et al. (2023, 2024), who pointed out that this is due to a shift in the distributions $P(X|Y)$, which is introduced when binarizing multiclass labels for the one-vs.-rest settings.

From our experiments with tuned base classifiers, we can further infer that in general, more accurate base classifiers do not yield more accurate estimations of class prevalences when used by quantifiers. Particularly in the binary case, we hardly observed any positive effect from using tuned base classifiers. For quantifiers that use misclassification rates, an explanation of this outcome might be that having somewhat higher misclassification rates may actually yield more numerical stability in the predictions. The only exception to this pattern was given by the *classify and count*-based methods *CC* and *PCC*, for which it could also be expected that optimizing the base classifiers would be beneficial. Yet, these methods still did not appear on par with the best-performing even methods after this kind of tuning. This overall result appears to contradict the findings of a simulation study by Tasche (2019), who concluded that more accurate base classifiers led to shorter confidence intervals in class prevalence estimations. However, Tasche only considered normally distributed synthetic data, which likely does not accurately represent the nature of real-world data. In the multiclass setting, tuned base classifiers appeared to have a more positive effect on aggregate over all datasets, specifically for the *EM* and *GPAC* methods, for which their tuned variants also appeared strong in the LeQua case study. Yet, when looking at the average error scores over the individual datasets, one can observe that this is not at all a consistent trend, and the strong aggregate performance appears to result from outstanding performances on a few of the only nine multiclass datasets on which we performed this hyperparameter tuning. In conclusion, if, in practice, resources for parameter tuning are available, we recommend that they should not be used to train more accurate base classifiers. Instead, one should consider parameters of base classifiers as parameters of the quantifier applying it, and directly optimize for quantification performance.

Considering our case study on the LeQua data, the results obtained from applying quantifiers with default parameterization and quantifiers with tuned base classifiers were mostly in line with the main results. Smaller variations, such as slightly weaker performance of the *TSMax* and *FM* methods, have also been observed on individual datasets in the main experiments, and the relatively weak performance of the *HDx* and *readme* methods is probably due to non-optimal binning of the given data.

However, novel insights were gained from the final part of the case study, in which the hyperparameters of all quantifiers, including those of base classifiers, were tuned for quantification performance. In these experiments, we observed that the methods that already performed best with their default parameters were also among the best methods after tuning. Specifically, the tuned *DyS* and *MS* methods were among the best methods in the binary setting, while the tuned *EM* and *GPAC* methods overall yielded the best performance in the multiclass setting. In addition, the untuned variants of these methods also performed better than the tuned versions of most other methods, with only a few exceptions. In particular, in the binary setting, the best performing algorithm was given by the *EM* method, which

appeared rather mediocre with default parameterization. Given that also in the multiclass setting, this method did strongly improve its performance with respect to AE, this indicates that this algorithm strongly relies on proper calibration of its probabilistic base classifier, as has also been found by Esuli et al. (2021). The results on the binary LeQua data also provide further evidence that *classify and count*-based approaches are not reliable quantifiers, given that, even after tuning, these methods yielded worse performances than the untuned variants of all other methods in the binary setting. However, somewhat surprisingly, the tuned *CC* and *PCC* methods appeared to perform relatively well in the multiclass setting, although clearly being behind the strongest algorithms. Given that these methods can be considered natural multiclass quantifiers, this could be attributed to the overall observation that one-vs.-rest approaches are not suitable for the multiclass setting. By contrast, the only natural multiclass quantifiers that performed worse than these methods after tuning are *readme* and *FM*, which generally did not appear to work well on the LeQua dataset.

7.2 Limitations

This paper presents an extensive empirical comparison of state-of-the-art quantification methods. As such, our results are necessarily affected by some experimental design choices.

First, in our main experiments, we relied on default parameters for the individual algorithms and did not perform extensive hyperparameter optimization for the quantification algorithms on each dataset. While, on the one hand, this is due to computational considerations—we have performed more than 295,000 experiments with 10 sampling iterations each, making extra hyperparameter optimization steps infeasible—this also reflects the performance that these methods would achieve when being used off-the-shelf. Further, there is surprisingly little research on tuning protocols for quantification (see Esuli et al., 2023, chap. 3.5). Standard model selection approaches such as k-fold cross-validation may, for instance, not necessarily work well for quantification, as these are unlikely to yield strong shifts between training and test distributions. Big validation sets, by contrast, are, in general, neither available nor trivial to construct, and thus, non-optimal optimization schemes may also bias the given results. However, we tested hyperparameter tuning on the dataset from the LeQua challenge, where a huge set of validation samples had been provided.

Similarly, properly designing sampling protocols for evaluation is not trivial either, and design choices in our approach may have yielded unintended biases. We aimed to cover a wide range of training set sizes, training/test distributions, and distribution shifts, but, for instance, our grids for training and test distributions in the multiclass experiments are much coarser than in the binary case and thus might not completely represent all possible scenarios. In addition, while we tried to broadly sample from diverse distributions, there may be imbalances in the representation of individual classes given that, in our undersampling approach, instances from less populated classes are more likely to be used than instances from more populated classes. However, such imbalances in given datasets are generally hard to work around, and different approaches such as oversampling, i.e., sampling with replacement large amounts of instances from a very limited pool, may also come with different caveats. Although in the literature it is agreed that training and test distributions (Hassan et al., 2021; Esuli et al., 2023) and test set sizes (Maletzke et al., 2020) should be artificially varied when evaluating quantification methods, there has also been limited discussion on

how to effectively sample such distributions from a given dataset in a representative fashion, specifically when it is limited in size or unbalanced in its class distribution.

Furthermore, despite the broad range of datasets considered, an analysis as we have just conducted cannot realistically cover all possible application scenarios. In that regard, we would like to note that this study does not include algorithms from the authors or collaborators, such that the authors do not have stakes in any particular outcome.

Finally, the field of quantification research is very dynamic, and more recently published methods such as novel ensemble approaches (Donyavi et al., 2024) or the *Continuous Sweep* (Kloos et al., 2023) have not been included in our evaluation. Similarly, related problems such as ordinal quantification (Sakai, 2021; Castaño et al., 2024; Bunse et al., 2024) or multi-label quantification (Moreo et al., 2023), which have gained some research interest recently, are out of scope for this study, and systematic analyses of methods for these problems could pose an interesting avenue for future research.

8. Conclusions

In this study, we have conducted a thorough experimental comparison of 24 quantification methods over 40 datasets, involving more than 5 million algorithm runs. In our experiments, we have considered both the binary and the multiclass case in quantification and have also specifically considered the impact of shifting class label distributions between training and test data, as well as the impact of having relatively small training sets. In the binary case, we have identified a group of methods that generally appear to work best, namely the threshold selection-based *median sweep* and *TSMax* methods (Forman, 2008), the distribution matching approaches from the *DyS* framework (Maletzke et al., 2019) including *HDy* (González-Castro et al., 2013), *Forman's mixture model* (Forman, 2005), and *Friedman's method* (Friedman, 2014). Regarding the multiclass case, a group of distribution matching methods, which naturally extend to multiclass quantification, appeared to be generally superior to the other evaluated algorithms. We provide further evidence that the multiclass setting in general is much harder to solve for established quantification methods, as the error scores obtained were consistently multiple times higher than in the binary case. This indicates a certain potential for future research in this specific setting. Further, our experiments demonstrate that more accurate base classifiers generally do not yield more accurate quantification. In addition, our results demonstrate that algorithms that are based on the *classify and count* principle, even when the underlying classifier is optimized for quantification, exhibit on average worse performance compared to other specialized solutions. Overall, we hope our findings provide guidance to practitioners in choosing the right quantification algorithm for a given application and aid researchers in identifying promising directions for future research.

Acknowledgements

The authors acknowledge support by the state of Baden-Württemberg through the bwHPC and the German Research Foundation (DFG) through grant INST 35/1597-1 FUGG. The authors thank Fabrizio Sebastiani and Letizia Milli for their help and for providing the code for the quantification forests.

Appendix A. Performance Measures for Base Classifiers

Several quantifiers that are analyzed in this study apply base classifiers and consider performance measures for these classifiers to form their predictions. Similarly, we also consider such performance measures in our experiments on tuned base classifiers. In the following, we briefly provide definitions for the performance measures that are used in this work.
We assume that we are given a dataset $D = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$ of N instances, where $\boldsymbol{x}_i \in \mathbb{R}^k$ denotes the feature vector of each instance, and $y_i \in \{\ell_1, \ldots, \ell_L\}$ the corresponding ground truth label. In addition, we assume that we are given a classifier $c : \mathbb{R}^k \longrightarrow \{\ell_1, \ldots, \ell_L\}$, which we apply on the given data to obtain the instance-wise predictions $\hat{y}_i = c(\boldsymbol{x}_i)$. Then, the *accuracy* of the classifier c on this dataset is given by

$$e_{\text{acc}}(y, \hat{y}) = \frac{1}{N} \cdot \sum_{i=1}^{N} \mathbb{1}(y_i = \hat{y}_i),$$

where $\mathbb{1}(\cdot)$ denotes the indicator function. When the distribution of class labels is unbalanced, the predictions of instances from minority classes carry little weight with respect to the resulting accuracy score. In such cases, one may consider the *balanced accuracy*, which is defined as

$$e_{\text{bal-acc}}(y, \hat{y}) = \frac{1}{L} \cdot \sum_{j=1}^{L} \frac{\sum_{i=1}^{N} \mathbb{1}(y_i = \hat{y}_i = \ell_j)}{\sum_{i=1}^{N} \mathbb{1}(y_i = \ell_j)}. \tag{4}$$

In the binary setting, we distinguish more specifically between *positive* and *negative* instances, for which the ground-truth labels are given by $y_i = 1$ and $y_i = 0$, respectively. Adjusted count-based quantifiers then specifically consider the ratio of predicted positives $\widehat{pos} = \frac{1}{N} \sum_{i=1}^{N} \mathbb{1}(\hat{y}_i = 1)$, and adjust these for true positive rate (tpr) and false positive rate fpr of their base classifiers, which are defined as

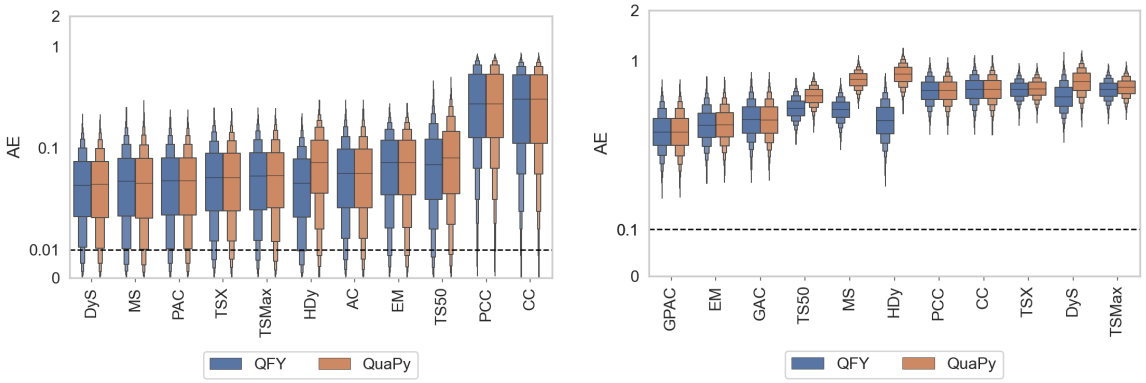
$$tpr := tpr(y, \hat{y}) = \frac{\sum_{i=1}^{N} \mathbb{1}(y_i = \hat{y}_i = 1)}{\sum_{i=1}^{N} \mathbb{1}(y_i = 1)} \text{ and } fpr := fpr(y, \hat{y}) = \frac{\sum_{i=1}^{N} \mathbb{1}(y_i = 0 \wedge \hat{y}_i = 1)}{\sum_{i=1}^{N} \mathbb{1}(y_i = 0)}. \tag{5}$$

Similarly, one may consider the true negative rate (tnr) and false negative rate (fnr), which are defined as

$$tnr := tnr(y, \hat{y}) = \frac{\sum_{i=1}^{N} \mathbb{1}(y_i = \hat{y}_i = 0)}{\sum_{i=1}^{N} \mathbb{1}(y_i = 0)} \text{ and } fnr := fnr(y, \hat{y}) = \frac{\sum_{i=1}^{N} \mathbb{1}(y_i = 1 \wedge \hat{y}_i = 0)}{\sum_{i=1}^{N} \mathbb{1}(y_i = 1)}.$$

In the binary setting, the balanced accuracy corresponds to the average of true positive rate and true negative rate of the given classifier, i.e., in this setting it holds that

$$e_{\text{bal-acc}}(y, \hat{y}) = \tfrac{1}{2}(tpr + tnr). \tag{6}$$



(a) AE values on the binary LeQua test data

(b) AE values on the multiclass LeQua test data

Figure 12: Comparison of our implementation (`QFY`) with the `QuaPy` package. We plot the distribution of absolute error (AE) values of all algorithms that are implemented in both codebases after applying them with the same parameterization on the LeQua test data. Overall, results from these packages appear either almost identical, or the results from the `QuaPy` implementation have higher AE values than those resulting from our implementation.

Appendix B. Comparison to the QuaPy Package

After the publication of our initial preprint, the `QuaPy` (Moreo et al., 2021) package has been published, which also implements a number of methods that are analyzed in this paper. To further validate the correctness of our implementation, we conduct a comparison of the `QuaPy` and our `QFY` implementation. To that end, we use the dataset from the LeQua challenge (cf. Section 6). In this experiment, we used `QuaPy` version 0.1.9, which is to date the latest version of this package.

The methods included in both implementations are the *CC*, *PCC*, *AC*, *PAC*, *TSX*, *TSMax*, *TS50*, *MS*, *EM*, and *HDy* methods. We leave out the SVM$^{\text{perf}}$-based methods, as, in our implementation, these have been adapted from an earlier implementation by the same research group that developed the `QuaPy` package (Esuli et al., 2022a). We note that in the multiclass case, the `QuaPy` implementation of *AC* and *PAC* corresponds rather to what we denoted as *GAC* and *GPAC* since no one-vs.-rest approach is applied there, but rather a direct least-squares-based solution of the system outlined in Equation 2. In addition, a notable difference lies in the `QuaPy` implementation of the *HDy* method, which uses an ensemble approach, matching distributions based on varying numbers of bins in $\{10, 20, \ldots, 110\}$, and then returning the average prediction, as originally proposed by González-Castro et al. (2013). By contrast, in our implementation we only match distributions once, using 10 bins as default value.

In the comparison, we used the same experimental setting as in Section 6.1. We tried to keep the parameterization of the algorithms as consistent as possible across both implementations, including the use of the same logistic regression base classifier.

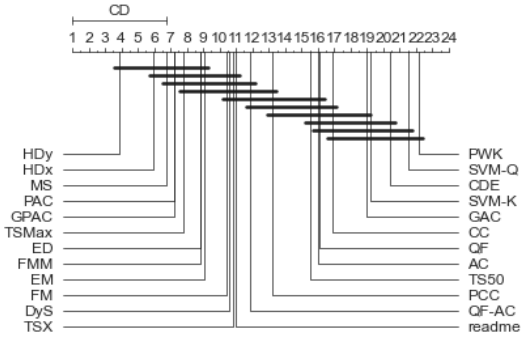
In Figure 12, we present the distribution of AE values on test samples from the LeQua data, both for the binary and multiclass versions of this challenge. Overall, we observe that the results from our `QFY` implementation are either (close to) identical or better than the results from the `QuaPy` package with respect to AE.
In the binary case, the only notable difference in performance can be seen for the *HDy* method, where we also identified a difference in implementation that we discussed above. The subpar performance of the `QuaPy` implementation can likely be explained by the finding that, when using more than 10 bins, the performance of this method tends to deteriorate (Maletzke et al., 2019).
In the multiclass case, there are some differences in the performances of one-vs.-rest quantifiers, specifically for the *TS50*, *MS*, *DyS*, and *HDy* methods. We suppose that these result from minor differences in the implementations for the binary case that could get amplified when normalizing binary one-vs.-rest predictions over 28 classes.
Overall, we find that our `QFY` implementation provides similar results to the `QuaPy` implementation and—where results differ—our implementation generally tends to yield lower error scores.

Appendix C. Additional Plots and Tables for the Main Experiments

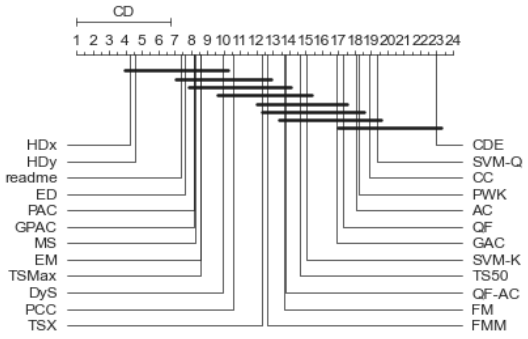
Complementing the results of Section 5, we show additional plots and tables regarding our main experiments.

C.1 Aggregated Ranking Plots

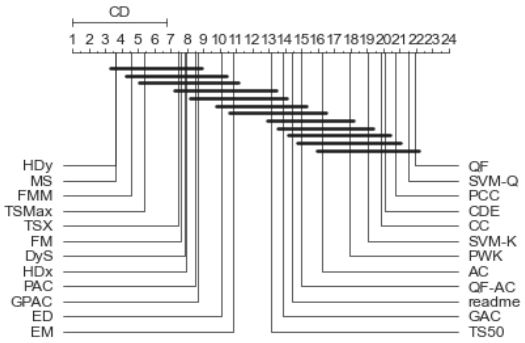
In the following, we present additional analytical results regarding the ranking of algorithms. We compute the average ranks of all algorithms aggregated per dataset, filtered by several conditions. Then, we apply a Nemenyi post-hoc test at 5% significance. In the individual plots, we then show the average performance rank for each algorithm. Horizontal bars indicate which algorithms' average rankings do not differ to a degree that is statistically significant, cf. Demšar (2006).
Complementing the results of Section 5.1, Figure 13 shows the distributions of rankings under varying shifts between training and test data, and Figure 14 displays the rankings of the quantification methods when only a few training samples are given. In both figures, we observe that the rankings are very similar to the general cases. However, we observe a stronger distinction in the average ranks for high shifts and few training data.
Figure 15 and Figure 16 complement the results of Section 5.2 by presenting additional rankings in the multiclass settings. Figure 15 displays the distributions of rankings of quantification algorithms under minor and major shifts between training and test data. We only observe bigger changes in the rankings with respect to AE, with *GPAC* appearing most robust toward major shifts. Figure 16 displays the rankings of multiclass quantifiers when only settings with few training samples are considered. Rankings generally appear to align with the general setting.



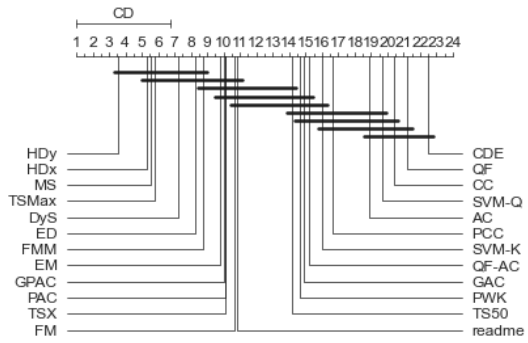
(a) AE-based rankings under minor shift



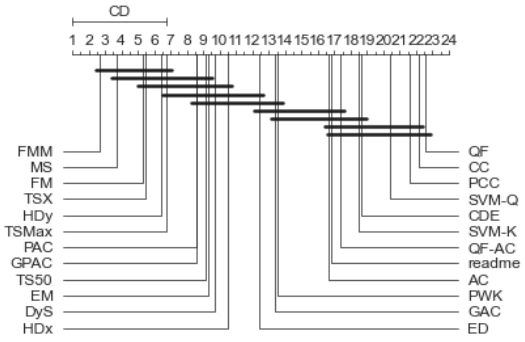
(b) NKLD-based rankings under minor shift



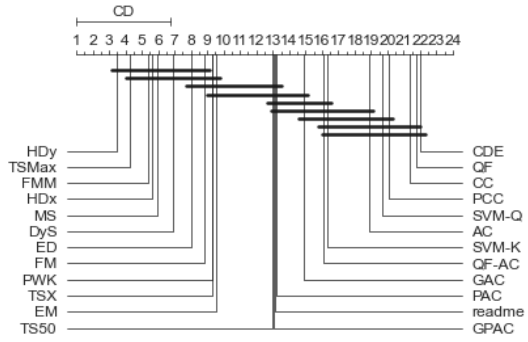
(c) AE-based rankings under medium shift



(d) NKLD-based rankings under medium shift

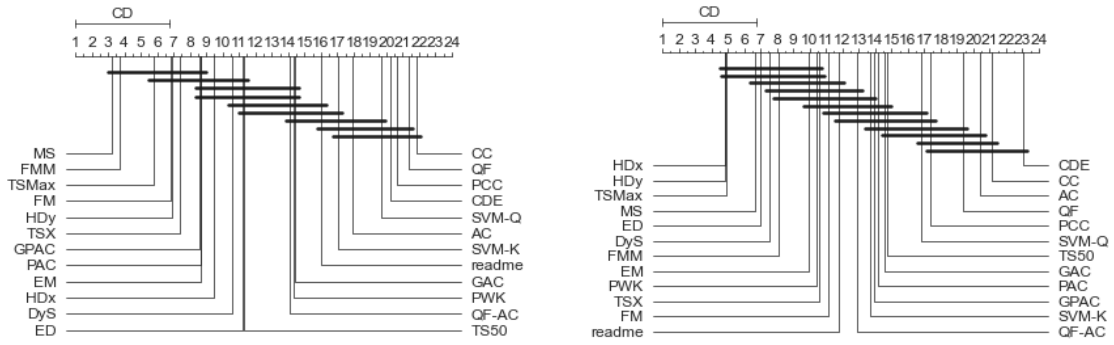


(e) AE-based rankings under major shift



(f) NKLD-based rankings under major shift

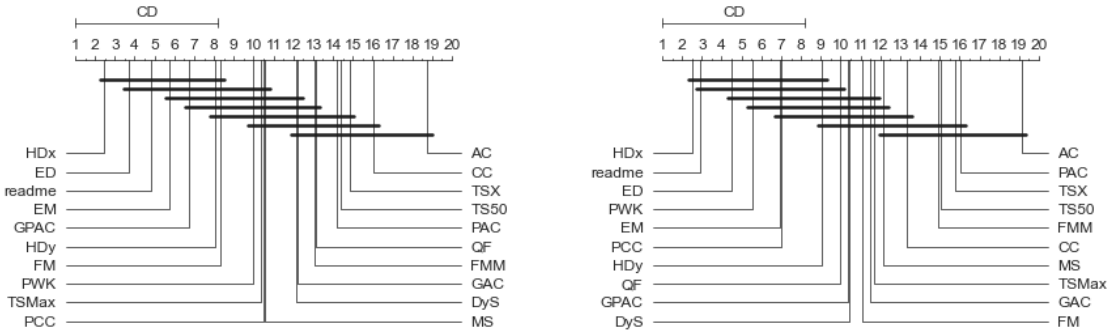
Figure 13: Impact of distribution shifts on algorithm rankings in the binary setting. We plot distributions of rankings with respect to absolute error (AE) and normalized Kullback-Leibler divergence (NKLD), separated by minor, medium, and major shifts.



(a) Average rankings with respect to AE

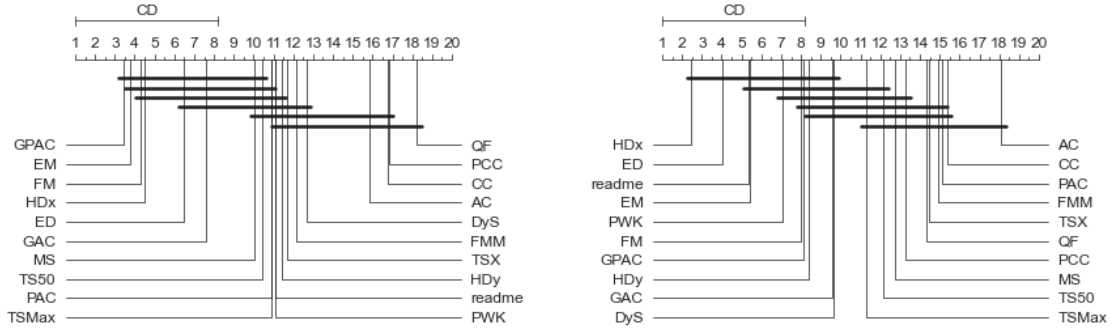
(b) Average rankings with respect to NKLD

Figure 14: Performance rankings under small amounts of training data in the binary setting. We plot the distributions of rankings with respect to absolute error (AE) and normalized Kullback-Leibler divergence (NKLD), obtained by 10/90 training/test splits.



(a) AE-based rankings under minor shift

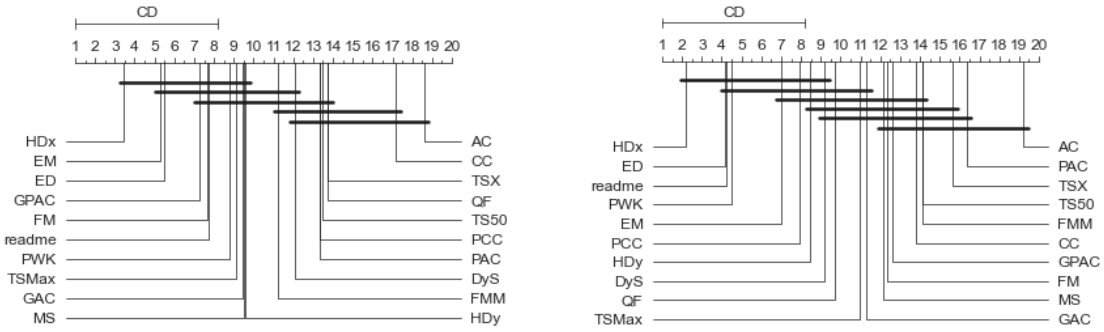
(b) NKLD-based rankings under minor shift



(c) AE-based rankings under major shift

(d) NKLD-based rankings under major shift

Figure 15: Impact of distribution shifts on algorithm rankings in the multiclass setting. We plot distributions of rankings with respect to absolute error (AE) and normalized Kullback-Leibler divergence (NKLD), separated by minor and major shifts.



(a) Average rankings with respect to AE

(b) Average rankings with respect to NKLD

Figure 16: Performance rankings under small amounts of training data in the multiclass setting. We plot distributions of rankings with respect to absolute error (AE) and normalized Kullback-Leibler divergence (NKLD), obtained by 10/90 training/test splits.

C.2 Detailed Error Scores for Quantifiers With Tuned Base Classifiers

Finally, we present additional results from our experiments with quantifiers that apply tuned base classifiers.

Tables 6, 7, and 8 display the average error scores of all algorithms per dataset in the binary setting, where it can be seen that only for *classify and count*-based methods there is a trend that tuned base classifiers improve quantification performance.

Table 9 shows the corresponding results in the multiclass setting. It can be seen that tuned base classifiers appear to improve the average error scores of the quantifiers applying them when aggregating over all datasets. However, this trend is not consistent across all individual datasets, with tuned base classifiers often times leading to worse results.

	AC	AC -LR	AC -RF	AC -AB	AC -SV	PAC	PAC -LR	TSX	TSX -LR	TSX -SV	TS50	TS50 -LR	TS50 -SV	TSMax	TSMax -LR	TSMax -SV	MS	MS -LR	MS -SV
bc-cat	0.230	0.087	0.125	0.128	0.114	0.112	0.076	0.077	0.088	0.08	0.137	0.153	0.129	0.079	0.081	0.079	**0.055**	0.074	0.071
bc-cont	0.133	0.071	0.085	0.102	0.089	0.072	0.066	0.051	0.065	0.061	0.130	0.128	0.141	0.049	0.06	0.061	**0.042**	0.061	0.057
cars	0.130	0.091	0.106	0.087	0.086	0.080	0.074	0.063	0.078	0.061	0.110	0.115	0.107	0.060	0.075	0.061	**0.049**	0.068	0.055
conc	0.533	0.201	0.224	0.210	0.177	0.171	0.195	0.154	0.203	0.143	0.190	0.223	0.164	0.144	0.214	0.153	**0.121**	0.185	0.131
contra	0.613	0.532	0.430	0.439	0.549	0.332	0.48	0.351	0.539	0.5	0.371	0.535	0.505	0.326	0.563	0.479	**0.307**	0.545	0.464
cappl	0.323	0.296	0.272	0.240	0.297	0.155	0.238	0.127	0.28	0.183	0.200	0.322	0.235	0.128	0.291	0.182	**0.104**	0.273	0.162
drugs	0.168	0.213	0.316	0.250	0.236	0.118	0.138	0.102	0.19	0.119	0.115	0.196	0.137	0.106	0.221	0.128	**0.088**	0.193	0.108
flare	0.584	0.601	0.617	0.630	0.621	0.344	0.483	0.353	0.577	0.527	0.345	0.559	0.531	0.306	0.555	0.509	**0.269**	0.535	0.496
grid	0.090	0.075	0.083	0.062	0.029	0.046	0.05	0.046	0.048	0.02	0.052	0.056	0.040	0.052	0.055	0.021	0.038	0.038	**0.017**
ads	0.175	0.114	0.156	0.119	0.139	0.103	0.095	0.075	0.094	0.084	0.113	0.126	0.121	0.067	0.09	0.084	**0.054**	0.082	0.074
mush	0.014	0.008	0.010	0.009	0.018	0.011	**0.007**	0.008	**0.007**	0.012	0.048	0.044	0.053	0.009	**0.007**	0.012	**0.007**	0.018	0.016
music	0.547	0.577	0.592	0.606	0.460	0.324	0.532	0.327	0.535	0.405	0.346	0.549	0.411	0.299	0.557	0.409	**0.272**	0.542	0.386
musk	0.110	0.088	0.129	0.093	0.066	0.070	0.074	0.067	0.076	0.047	0.080	0.093	0.087	0.068	0.072	0.048	0.058	0.064	**0.044**
craft	0.248	0.146	0.146	0.183	0.169	0.084	0.112	0.065	0.11	0.108	0.088	0.120	0.110	0.075	0.127	0.105	**0.058**	0.110	0.09
spam	0.274	0.060	0.071	0.056	0.061	0.069	0.05	0.047	0.043	0.041	0.071	0.066	0.067	0.050	0.049	0.044	0.043	0.048	**0.040**
alco	0.480	0.548	0.506	0.581	0.490	0.328	0.504	0.341	0.58	0.427	0.366	0.550	0.432	0.300	0.588	0.416	**0.277**	0.568	0.397
study	0.347	0.197	0.190	0.174	0.199	0.187	0.189	0.201	0.192	0.176	0.215	0.197	0.186	0.194	0.214	0.183	0.161	0.182	**0.147**
telco	0.224	0.226	0.206	0.217	0.276	0.075	0.115	0.071	0.141	0.232	0.080	0.159	0.244	0.069	0.149	0.216	**0.060**	0.137	0.21
thrm	0.612	0.486	0.461	0.496	0.454	0.318	0.451	0.320	0.445	0.418	0.355	0.468	0.418	0.298	0.46	0.405	**0.272**	0.438	0.376
turk	0.619	0.653	0.652	0.693	0.575	0.248	0.36	0.282	0.523	0.581	0.283	0.542	0.579	0.240	0.531	0.517	**0.239**	0.531	0.517
vgame	0.209	0.192	0.256	0.233	0.201	0.085	0.156	0.088	0.186	0.135	0.086	0.186	0.134	0.091	0.198	0.133	**0.076**	0.190	0.122
voice	0.150	0.029	0.033	0.031	0.031	0.048	0.026	0.035	0.023	**0.022**	0.060	0.063	0.069	0.032	0.024	0.025	0.034	0.033	0.025
wine	0.479	0.286	0.185	0.338	0.232	0.095	0.198	0.091	0.228	0.172	0.093	0.229	0.170	0.096	0.259	0.168	**0.081**	0.239	0.155
yeast	0.681	0.513	0.365	0.421	0.425	0.238	0.449	0.276	0.475	0.378	0.306	0.501	0.397	0.234	0.526	0.372	**0.212**	0.517	0.353
Mean	0.332	0.262	0.259	0.267	0.250	0.155	0.213	0.151	0.239	0.205	0.177	0.258	0.228	0.140	0.249	0.200	**0.124**	0.236	0.188

(a) Absolute error values

	AC	AC -LR	AC -RF	AC -AB	AC -SV	PAC	PAC -LR	TSX	TSX -LR	TSX -SV	TS50	TS50 -LR	TS50 -SV	TSMax	TSMax -LR	TSMax -SV	MS	MS -LR	MS -SV
bc-cat	0.161	0.031	0.066	0.071	0.052	0.065	0.038	0.024	0.039	0.028	0.088	0.092	0.071	0.017	0.025	0.02	**0.015**	0.030	0.027
bc-cont	0.084	0.024	0.036	0.044	0.034	0.04	0.04	0.013	0.022	0.017	0.081	0.077	0.087	**0.010**	0.018	0.014	0.019	0.032	0.030
cars	0.074	0.043	0.059	0.040	0.038	0.051	0.038	0.028	0.039	0.027	0.057	0.059	0.051	**0.016**	0.03	0.019	0.019	0.034	0.023
conc	0.459	0.137	0.151	0.131	0.100	0.13	0.116	0.089	0.126	0.077	0.125	0.151	0.090	**0.052**	0.116	0.061	0.06	0.114	0.065
contra	0.537	0.449	0.321	0.356	0.422	0.247	0.303	0.258	0.436	0.375	0.271	0.436	0.388	0.175	0.412	0.326	**0.172**	0.411	0.325
cappl	0.238	0.201	0.187	0.152	0.205	0.093	0.158	0.061	0.192	0.107	0.128	0.237	0.155	**0.036**	0.184	0.09	0.04	0.192	0.091
drugs	0.093	0.145	0.254	0.171	0.165	0.057	0.085	0.041	0.118	0.058	0.059	0.128	0.069	**0.025**	0.121	0.051	0.031	0.119	0.052
flare	0.436	0.482	0.483	0.488	0.476	0.247	0.338	0.251	0.467	0.387	0.259	0.462	0.400	0.152	0.442	0.339	**0.151**	0.440	0.341
grid	0.041	0.027	0.051	0.021	0.007	0.015	0.017	0.009	0.011	0.004	0.010	0.013	0.006	0.007	0.009	**0.003**	0.007	0.008	0.004
ads	0.112	0.062	0.095	0.059	0.081	0.074	0.063	0.035	0.053	0.041	0.070	0.079	0.069	**0.016**	0.042	0.032	0.021	0.049	0.035
mush	0.002	**0.001**	0.002	0.002	0.008	**0.001**	**0.001**	**0.001**	**0.001**	0.007	0.012	0.012	0.019	**0.001**	**0.001**	0.005	**0.001**	0.004	0.005
music	0.435	0.472	0.475	0.460	0.349	0.248	0.343	0.223	0.443	0.293	0.242	0.450	0.306	0.147	0.433	0.26	**0.142**	0.433	0.262
musk	0.057	0.039	0.076	0.042	0.028	0.029	0.03	0.029	0.036	0.016	0.036	0.051	0.046	0.016	0.023	**0.010**	0.019	0.025	0.013
craft	0.179	0.087	0.077	0.121	0.091	0.049	0.057	0.02	0.059	0.051	0.042	0.066	0.051	**0.014**	0.053	0.033	0.02	0.055	0.039
spam	0.220	0.022	0.033	0.020	0.018	0.036	0.018	0.011	0.016	0.013	0.031	0.029	0.026	**0.009**	0.015	**0.009**	0.012	0.016	0.010
alco	0.365	0.452	0.407	0.457	0.374	0.254	0.337	0.259	0.456	0.323	0.280	0.452	0.329	**0.155**	0.428	0.276	0.159	0.432	0.276
study	0.264	0.121	0.103	0.088	0.114	0.115	0.104	0.106	0.11	0.092	0.129	0.122	0.106	0.071	0.103	0.079	**0.069**	0.102	0.075
telco	0.152	0.172	0.148	0.160	0.216	0.038	0.075	0.032	0.1	0.167	0.035	0.110	0.177	**0.015**	0.09	0.146	0.024	0.092	0.149
thrm	0.505	0.366	0.310	0.342	0.301	0.252	0.296	0.235	0.363	0.300	0.267	0.377	0.300	**0.169**	0.335	0.235	0.174	0.341	0.232
turk	0.527	0.546	0.557	0.586	0.438	0.194	0.278	0.197	0.418	0.456	0.206	0.439	0.459	0.113	0.404	0.375	**0.112**	0.404	0.375
vgame	0.152	0.132	0.172	0.154	0.129	0.045	0.096	0.038	0.131	0.076	0.036	0.126	0.077	**0.026**	0.124	0.064	0.028	0.128	0.066
voice	0.107	0.008	0.010	0.008	0.007	0.025	0.007	0.013	0.004	0.007	0.021	0.021	0.026	0.006	**0.003**	0.004	0.01	0.009	0.009
wine	0.419	0.201	0.096	0.245	0.136	0.05	0.118	0.039	0.172	0.099	0.036	0.173	0.097	**0.024**	0.175	0.081	0.026	0.174	0.081
yeast	0.595	0.425	0.258	0.304	0.319	0.18	0.297	0.198	0.418	0.266	0.225	0.444	0.289	**0.111**	0.393	0.229	0.115	0.406	0.230
Mean	0.259	0.194	0.184	0.188	0.171	0.106	0.136	0.092	0.176	0.137	0.114	0.192	0.154	**0.058**	0.166	0.115	0.06	0.169	0.117

(b) Normalized Kullback-Leibler divergence values

Table 6: Results of *adjusted count*-based quantifiers with tuned base classifiers in the binary setting, where the base classifiers were tuned with respect to their accuracy. We show error scores averaged across all scenarios per algorithm and dataset, along with the total means per algorithm (last row). Algorithms based on untuned logistic regression classifiers are denoted as before (no suffix), alternative tuned base classifiers are marked with respective suffixes: logistic regressors (LR), support vector machines (SV), random forests (RF) and AdaBoost (AB).

	GAC	GAC -LR	GAC -RF	GAC -AB	GAC -SV	GPAC	GPAC -LR	DyS	DyS -LR	DyS -SV	FMM	FMM -LR	FMM -SV	HDy	HDy -LR	HDy -SV	FM	FM -LR	EM	EM -LR	CDE	CDE -LR
bc-cat	0.193	0.084	0.124	0.127	0.107	0.112	0.076	0.121	0.118	0.103	**0.056**	0.064	0.065	0.083	0.084	0.096	0.062	0.106	0.207	0.195	0.315	0.145
bc-cont	0.117	0.072	0.090	0.108	0.080	0.072	0.066	0.106	0.070	0.099	0.048	0.060	0.064	0.056	0.058	0.087	**0.039**	0.132	0.125	0.262	0.123	0.176
cars	0.113	0.083	0.093	0.077	0.079	0.080	0.074	0.078	0.069	0.071	**0.051**	0.063	**0.051**	0.059	0.059	0.061	0.059	0.078	0.087	0.086	0.180	0.098
conc	0.369	0.194	0.216	0.206	0.177	0.171	0.193	0.175	0.172	0.172	**0.125**	0.174	**0.125**	0.178	0.156	0.147	0.155	0.187	0.336	0.216	0.745	0.284
contra	0.472	0.438	0.369	0.370	0.445	0.331	0.479	0.434	0.448	0.538	0.297	0.505	0.455	0.4	0.416	0.489	0.351	0.505	**0.249**	0.422	0.881	0.830
cappl	0.289	0.247	0.237	0.229	0.258	0.156	0.238	0.205	0.250	0.26	0.109	0.240	0.155	0.172	0.23	0.228	0.115	0.333	**0.087**	0.413	0.302	0.416
drugs	0.174	0.185	0.261	0.227	0.208	0.119	0.138	0.144	0.174	0.142	**0.080**	0.194	0.107	0.101	0.174	0.131	0.104	0.219	0.134	0.187	0.134	0.312
flare	0.482	0.437	0.464	0.526	0.462	0.342	0.483	0.454	0.476	0.64	0.291	0.510	0.46	0.416	0.428	0.54	0.346	0.643	**0.256**	0.547	0.675	0.721
grid	0.086	0.075	0.080	0.059	0.028	0.046	0.05	0.042	0.034	**0.015**	0.035	0.038	0.016	0.033	0.035	**0.015**	0.044	0.051	0.048	0.068	0.258	0.213
ads	0.138	0.090	0.122	0.116	0.112	0.102	0.095	0.106	0.092	0.095	**0.060**	0.082	0.08	0.077	0.08	0.091	0.082	0.162	0.087	0.383	0.199	0.282
mush	0.014	0.008	0.010	0.009	0.018	0.011	**0.007**	0.014	0.009	0.025	0.016	0.015	0.021	**0.007**	**0.007**	0.015	0.008	**0.007**	0.017	0.015	0.009	0.008
music	0.462	0.429	0.440	0.497	0.387	0.324	0.532	0.429	0.471	0.416	0.283	0.542	0.366	0.371	0.436	0.4	0.328	0.555	**0.257**	0.479	0.840	0.809
musk	0.096	0.078	0.105	0.087	0.062	0.069	0.074	0.073	0.067	0.051	0.053	0.062	**0.044**	0.058	0.061	0.05	0.068	0.074	0.065	0.102	0.188	0.130
craft	0.219	0.144	0.142	0.176	0.164	0.084	0.112	0.082	0.196	0.11	**0.053**	0.086	0.098	0.058	0.189	0.099	0.067	0.113	0.144	0.096	0.528	0.276
spam	0.236	0.059	0.069	0.057	0.060	0.069	0.05	0.072	0.065	0.052	0.041	0.045	**0.039**	0.042	0.054	0.046	0.047	0.046	0.265	0.067	0.603	0.074
alco	0.451	0.425	0.407	0.477	0.415	0.337	0.504	0.431	0.437	0.42	**0.282**	0.547	0.369	0.36	0.433	0.395	0.342	0.566	0.296	0.457	0.695	0.720
study	0.301	0.188	0.182	0.176	0.189	0.187	0.188	0.233	0.156	0.161	0.162	0.162	**0.140**	0.194	0.153	0.154	0.192	0.191	0.175	0.167	0.533	0.205
telco	0.211	0.188	0.174	0.186	0.227	0.075	0.115	0.075	0.142	0.225	**0.056**	0.135	0.205	0.059	0.112	0.199	0.07	0.155	0.059	0.122	0.401	0.428
thrm	0.462	0.369	0.372	0.449	0.389	0.318	0.451	0.423	0.409	0.456	0.291	0.440	0.372	0.358	0.361	0.423	0.309	0.479	**0.266**	0.419	0.861	0.688
turk	0.477	0.451	0.455	0.477	0.438	0.246	0.359	0.303	0.492	0.548	0.219	0.488	0.484	0.281	0.483	0.502	0.28	0.558	**0.164**	0.460	0.881	0.878
vgame	0.209	0.163	0.215	0.192	0.175	0.085	0.156	0.090	0.147	0.137	0.075	0.177	0.123	0.084	0.145	0.129	0.089	0.182	**0.066**	0.188	0.586	0.348
voice	0.134	0.030	0.033	0.031	0.031	0.047	0.026	0.037	0.024	0.022	0.038	0.028	0.028	0.03	**0.021**	0.022	0.036	0.025	0.178	0.067	0.289	0.050
wine	0.372	0.230	0.176	0.260	0.207	0.095	0.198	0.140	0.186	0.183	**0.079**	0.236	0.158	0.096	0.183	0.17	0.102	0.241	0.233	0.201	0.815	0.650
yeast	0.471	0.414	0.328	0.380	0.374	0.241	0.445	0.338	0.435	0.422	**0.221**	0.512	0.339	0.273	0.387	0.377	0.261	0.469	0.38	0.404	0.873	0.743
Mean	0.273	0.212	0.215	0.229	0.212	0.155	0.213	0.192	0.214	0.224	**0.126**	0.225	0.182	0.16	0.198	0.203	0.148	0.253	0.174	0.251	0.496	0.395

(a) Absolute error values

	GAC	GAC -LR	GAC -RF	GAC -AB	GAC -SV	GPAC	GPAC -LR	DyS	DyS -LR	DyS -SV	FMM	FMM -LR	FMM -SV	HDy	HDy -LR	HDy -SV	FM	FM -LR	EM	EM -LR	CDE	CDE -LR
bc-cat	0.089	0.026	0.062	0.072	0.030	0.065	0.037	0.038	0.028	0.026	**0.016**	0.026	0.026	0.018	0.018	0.028	0.023	0.046	0.16	0.120	0.409	0.135
bc-cont	0.052	0.023	0.037	0.048	0.023	0.040	0.040	0.022	0.013	0.019	0.024	0.037	0.037	**0.007**	0.009	0.018	0.015	0.045	0.087	0.188	0.184	0.231
cars	0.051	0.030	0.037	0.027	0.027	0.049	0.037	0.021	0.014	0.017	0.019	0.030	0.018	0.013	**0.011**	0.013	0.030	0.039	0.034	0.030	0.212	0.058
conc	0.156	0.082	0.114	0.105	0.067	0.130	0.109	0.077	0.057	0.058	0.067	0.103	0.059	0.07	**0.044**	0.048	0.091	0.111	0.325	0.097	0.799	0.295
contra	0.242	0.204	0.177	0.152	0.191	0.247	0.297	0.245	0.211	0.291	0.199	0.404	0.321	0.203	0.166	0.264	0.260	0.410	**0.125**	0.181	0.843	0.813
cappl	0.156	0.093	0.102	0.109	0.103	0.095	0.151	0.075	0.098	0.104	0.045	0.169	0.084	0.057	0.078	0.088	0.054	0.202	**0.037**	0.280	0.415	0.489
drugs	0.093	0.068	0.104	0.099	0.073	0.057	0.083	0.037	0.077	0.042	0.028	0.122	0.051	**0.019**	0.071	0.036	0.044	0.126	0.022	0.088	0.094	0.429
flare	0.296	0.180	0.200	0.294	0.188	0.244	0.330	0.217	0.237	0.368	0.178	0.418	0.303	0.192	0.192	0.297	0.234	0.438	**0.081**	0.304	0.711	0.737
grid	0.034	0.027	0.043	0.018	0.006	0.015	0.017	0.005	0.003	**0.001**	0.005	0.007	0.004	0.002	0.003	**0.001**	0.009	0.012	0.014	0.038	0.414	0.326
ads	0.078	0.034	0.063	0.056	0.043	0.074	0.062	0.033	0.024	0.027	0.024	0.046	0.036	0.018	**0.017**	0.025	0.039	0.079	0.027	0.262	0.187	0.268
mush	0.002	0.001	0.003	0.002	0.008	0.001	0.001	0.001	**0.000**	0.005	0.004	0.003	0.007	**0.000**	**0.000**	0.003	0.001	0.001	0.001	0.004	0.004	0.005
music	0.258	0.187	0.192	0.276	0.179	0.248	0.339	0.207	0.235	0.213	0.172	0.433	0.249	0.168	0.187	0.201	0.224	0.421	**0.082**	0.216	0.829	0.780
musk	0.045	0.030	0.051	0.036	0.024	0.028	0.029	0.017	0.013	0.008	0.016	0.024	0.012	**0.007**	0.01	**0.007**	0.028	0.033	0.011	0.023	0.198	0.063
craft	0.106	0.066	0.066	0.087	0.068	0.049	0.055	0.021	0.137	0.029	0.018	0.049	0.041	**0.008**	0.13	0.023	0.027	0.055	0.089	0.036	0.733	0.339
spam	0.121	0.017	0.030	0.020	0.016	0.036	0.018	0.025	0.01	0.007	0.011	0.016	0.013	**0.004**	0.008	0.007	0.013	0.015	0.218	0.013	0.718	0.023
alco	0.279	0.183	0.182	0.241	0.177	0.260	0.332	0.207	0.212	0.211	0.192	0.428	0.249	0.176	0.184	0.199	0.262	0.429	**0.102**	0.200	0.783	0.723
study	0.145	0.077	0.078	0.085	0.073	0.115	0.101	0.095	0.05	0.055	0.078	0.087	0.068	0.075	**0.048**	0.055	0.103	0.101	0.084	0.053	0.689	0.121
telco	0.120	0.074	0.075	0.070	0.087	0.040	0.073	0.016	0.076	0.114	0.021	0.092	0.142	0.011	0.051	0.102	0.032	0.094	**0.007**	0.044	0.532	0.561
thrm	0.224	0.160	0.182	0.258	0.171	0.251	0.292	0.222	0.188	0.231	0.2	0.343	0.241	0.183	**0.143**	0.21	0.221	0.354	0.191	0.195	0.837	0.641
turk	0.247	0.182	0.184	0.192	0.178	0.192	0.267	0.133	0.306	0.321	0.138	0.389	0.352	0.109	0.28	0.291	0.207	0.416	**0.048**	0.195	0.843	0.840
vgame	0.131	0.056	0.099	0.075	0.063	0.045	0.094	0.020	0.062	0.053	0.03	0.124	0.066	0.019	0.056	0.05	0.040	0.126	**0.013**	0.062	0.763	0.296
voice	0.067	0.008	0.009	0.008	0.007	0.024	0.007	0.006	**0.002**	**0.002**	0.014	0.008	0.010	**0.002**	**0.002**	**0.002**	0.014	0.007	0.121	0.023	0.467	0.032
wine	0.164	0.080	0.072	0.102	0.069	0.049	0.115	0.057	0.076	0.073	0.032	0.170	0.085	**0.020**	0.069	0.064	0.048	0.168	0.211	0.093	0.831	0.786
yeast	0.200	0.180	0.153	0.188	0.163	0.183	0.285	0.179	0.201	0.215	0.133	0.402	0.231	**0.115**	0.155	0.186	0.190	0.386	0.373	0.188	0.842	0.778
Mean	0.140	0.086	0.097	0.109	0.085	0.106	0.132	0.082	0.097	0.104	0.069	0.164	0.113	**0.062**	0.08	0.092	0.092	0.171	0.103	0.122	0.556	0.407

(b) Normalized Kullback-Leibler divergence values

Table 7: Results of *distribution matching methods* in the binary setting, where the base classifiers were tuned with respect to their accuracy. We show error scores averaged across all scenarios per algorithm and dataset, along with the total means per algorithm (last row). Algorithms based on untuned logistic regression classifiers are denoted as before (no suffix), alternative tuned base classifiers are marked with respective suffixes: logistic regressors (LR), support vector machines (SV), random forests (RF) and AdaBoost (AB).

	CC	CC -LR	CC -RF	CC -AB	CC -SV	PCC	PCC -LR	SVM -K	SVM -Q	RBF -K	RBF -Q
bc-cat	0.380	**0.127**	0.207	0.174	0.166	0.390	0.202	0.304	0.753	0.146	0.202
bc-cont	0.172	0.084	0.116	0.14	0.107	0.245	0.251	0.167	0.838	0.08	**0.066**
cars	0.299	0.181	0.195	0.181	**0.140**	0.306	0.195	0.228	0.227	0.499	0.54
conc	0.699	0.434	0.454	0.421	0.37	0.608	0.446	0.304	0.601	**0.279**	0.507
contra	0.814	0.777	0.716	0.718	0.771	0.672	0.662	**0.565**	0.802	0.579	0.719
cappl	0.473	0.426	0.422	0.383	0.431	0.465	0.485	0.33	**0.322**	0.454	0.496
drugs	0.421	0.463	0.536	0.476	0.474	0.428	0.488	**0.318**	0.337	0.52	0.62
flare	0.694	0.712	0.735	0.727	0.731	0.629	0.653	**0.480**	0.614	0.616	0.655
grid	0.492	0.458	0.448	0.391	**0.158**	0.468	0.468	0.749	0.668	0.194	0.52
ads	0.352	0.234	0.322	**0.218**	0.283	0.352	0.287	0.255	0.341	0.416	0.479
mush	0.027	0.011	0.018	**0.010**	0.012	0.054	0.017	0.098	0.054	0.022	0.364
music	0.748	0.77	0.792	0.751	0.711	0.651	0.666	**0.465**	0.572	0.614	0.684
musk	0.367	0.277	0.359	0.277	**0.180**	0.379	0.289	0.248	0.321	0.313	0.509
craft	0.602	0.515	0.509	0.509	0.549	0.543	0.492	0.344	0.684	**0.324**	0.52
spam	0.595	0.246	0.263	**0.216**	0.236	0.537	0.264	0.261	0.638	0.217	0.519
alco	0.693	0.731	0.741	0.746	0.695	0.625	0.647	**0.495**	0.608	0.692	0.658
study	0.589	**0.382**	0.428	0.385	0.386	0.538	**0.382**	0.61	0.696	0.567	0.641
telco	0.571	0.582	0.600	0.583	0.603	0.525	0.544	**0.373**	0.476	0.541	0.648
thrm	0.773	0.694	0.679	0.677	0.675	0.655	0.627	**0.491**	0.629	0.494	0.604
turk	0.847	0.851	0.845	0.848	0.836	0.684	0.692	**0.558**	0.64	0.562	0.734
vgame	0.631	0.571	0.659	0.608	0.601	0.570	0.533	**0.407**	0.594	0.749	0.699
voice	0.346	0.081	0.089	**0.077**	0.08	0.378	0.126	0.166	0.417	0.103	0.323
wine	0.750	0.655	0.604	0.656	0.622	0.637	0.596	0.662	0.905	**0.408**	0.661
yeast	0.839	0.759	0.672	0.717	0.697	0.680	0.653	0.569	0.881	**0.516**	0.78
Mean	0.549	0.459	0.475	0.454	0.438	0.501	0.444	**0.394**	0.567	0.413	0.548

(a) AE values

	CC	CC -LR	CC -RF	CC -AB	CC -SV	PCC	PCC -LR	SVM -K	SVM -Q	RBF -K	RBF -Q
bc-cat	0.182	**0.027**	0.060	0.038	0.055	0.123	0.046	0.08	0.316	0.038	0.05
bc-cont	0.067	0.013	0.025	0.026	0.026	0.060	0.063	0.035	0.447	0.033	**0.009**
cars	0.099	0.048	0.065	0.046	**0.038**	0.083	0.045	0.051	0.045	0.241	0.288
conc	0.495	0.206	0.196	0.168	0.146	0.245	0.151	0.074	0.306	**0.067**	0.253
contra	0.581	0.541	0.430	0.461	0.514	0.286	0.28	**0.197**	0.382	0.213	0.352
cappl	0.244	0.232	0.218	0.177	0.238	0.159	0.173	0.093	**0.086**	0.188	0.227
drugs	0.144	0.223	0.322	0.242	0.244	0.134	0.171	**0.078**	0.088	0.239	0.269
flare	0.420	0.494	0.503	0.48	0.498	0.256	0.275	**0.159**	0.243	0.259	0.295
grid	0.188	0.152	0.176	0.124	**0.030**	0.151	0.145	0.596	0.425	0.037	0.23
ads	0.134	0.075	0.120	**0.056**	0.108	0.108	0.078	0.071	0.107	0.156	0.173
mush	0.003	**0.001**	0.002	**0.001**	**0.001**	0.006	0.002	0.016	0.007	0.002	0.202
music	0.474	0.537	0.545	0.477	0.451	0.270	0.284	**0.136**	0.204	0.29	0.369
musk	0.116	0.074	0.127	0.078	**0.041**	0.109	0.073	0.049	0.087	0.088	0.283
craft	0.318	0.216	0.208	0.231	0.25	0.199	0.167	0.09	0.306	**0.079**	0.211
spam	0.351	0.063	0.071	0.047	0.057	0.200	0.062	0.061	0.298	**0.045**	0.265
alco	0.392	0.501	0.498	0.501	0.458	0.254	0.273	**0.167**	0.238	0.363	0.308
study	0.337	0.153	0.162	0.124	0.153	0.202	**0.118**	0.213	0.283	0.233	0.306
telco	0.284	0.316	0.311	0.31	0.352	0.186	0.205	**0.099**	0.151	0.224	0.299
thrm	0.534	0.433	0.387	0.38	0.386	0.275	0.259	**0.164**	0.295	0.176	0.282
turk	0.613	0.642	0.637	0.652	0.593	0.292	0.299	0.215	0.294	**0.195**	0.391
vgame	0.323	0.287	0.373	0.325	0.312	0.215	0.196	**0.114**	0.267	0.443	0.397
voice	0.153	0.011	0.013	**0.010**	0.011	0.113	0.021	0.032	0.183	0.013	0.15
wine	0.524	0.379	0.296	0.389	0.328	0.262	0.237	0.248	0.513	**0.115**	0.392
yeast	0.652	0.532	0.370	0.424	0.434	0.291	0.273	0.228	0.636	**0.174**	0.501
Mean	0.318	0.256	0.255	0.24	0.239	0.187	0.162	**0.136**	0.259	0.163	0.271

(b) NKLD values

Table 8: Results of *classify and count*-based quantifiers in the binary setting, where the base classifiers were tuned with respect to their accuracy. We show the averaged error scores for all scenarios per algorithm and dataset with respect to absolute error (AE) and normalized Kullback-Leibler divergence (NKLD). We further provide the total mean error scores per algorithm (last row). Algorithms based on untuned logistic regression classifiers are denoted as before (no suffix), alternative tuned base classifiers are marked with respective suffixes: logistic regressors (LR), support vector machines (SV), random forests (RF) and AdaBoost (AB). In addition, we present results for the *SVM-K* and *SVM-Q* methods and their adaptations that use an RBF kernel (*RBF-K* and *RBF-Q*).

	GAC	GAC -LR	GAC -RF	GAC -AB	GAC -SV	GPAC	GPAC -LR	FMM	FMM -LR	FMM -SV	FM	FM -LR	EM	EM -LR	CC	CC -LR	CC -RF	CC -AB	CC -SV	PCC	PCC -LR
conc	0.486	0.313	0.299	0.423	**0.259**	0.473	0.298	0.564	0.494	0.294	0.51	0.305	0.498	0.283	0.915	0.555	0.563	0.733	0.459	0.692	0.527
contra	0.600	0.495	0.490	0.517	0.534	0.515	0.579	0.467	0.463	0.551	0.512	0.620	**0.396**	0.531	0.833	0.825	0.808	0.829	0.835	0.699	0.705
drugs	0.256	0.252	0.284	0.391	0.247	0.199	0.206	0.160	**0.157**	0.185	0.181	0.203	0.218	0.278	0.465	0.516	0.623	0.648	0.518	0.482	0.554
craft	0.296	0.238	0.250	0.337	0.28	**0.190**	0.194	0.531	0.43	0.341	**0.190**	0.199	0.191	0.235	0.752	0.666	0.673	0.716	0.707	0.654	0.622
thrm	0.780	0.694	0.565	0.760	0.645	0.629	0.658	0.619	0.56	0.582	0.663	0.751	**0.494**	0.533	1.042	1.026	0.893	1.115	0.928	0.769	0.759
turk	0.525	0.498	0.572	0.562	0.518	0.342	0.385	0.324	0.324	0.472	0.392	0.451	**0.277**	0.441	0.976	0.984	1.003	0.987	1.028	0.727	0.732
vgame	0.520	0.517	0.529	0.567	0.536	0.46	0.463	0.600	0.568	0.572	0.474	0.465	**0.322**	0.339	0.590	0.574	0.658	0.694	0.614	0.520	0.493
wine	0.656	0.553	0.572	0.699	0.567	0.575	0.647	0.637	0.566	0.557	0.605	0.693	0.757	**0.460**	0.965	0.777	0.708	0.843	0.647	0.636	0.586
yeast	0.567	0.425	0.386	0.497	0.415	0.408	0.399	0.505	0.491	0.466	0.413	0.387	0.613	**0.336**	0.878	0.514	0.478	0.611	0.512	0.612	0.468
Mean	0.521	0.443	0.438	0.528	0.445	0.421	0.425	0.490	0.45	0.447	0.438	0.453	0.419	**0.382**	0.824	0.715	0.712	0.797	0.694	0.643	0.605

(a) Absolute error values for *natural multiclass quantifiers*

	GAC	GAC -LR	GAC -RF	GAC -AB	GAC -SV	GPAC	GPAC -LR	FMM	FMM -LR	FMM -SV	FM	FM -LR	EM	EM -LR	CC	CC -LR	CC -RF	CC -AB	CC -SV	PCC	PCC -LR
conc	0.310	0.212	0.192	0.312	0.162	0.467	0.226	0.407	0.335	0.252	0.455	0.234	0.46	**0.142**	0.640	0.248	0.263	0.361	0.172	0.276	0.173
contra	0.448	0.340	0.335	0.359	0.368	0.469	0.470	0.395	0.373	0.485	0.445	0.480	**0.237**	0.256	0.464	0.458	0.451	0.469	0.455	0.280	0.284
drugs	0.180	0.141	0.177	0.246	0.132	0.150	0.156	0.108	0.127	0.109	0.126	0.132	**0.049**	0.189	0.151	0.225	0.311	0.353	0.236	0.147	0.184
craft	0.172	0.133	0.125	0.185	0.146	0.150	0.133	0.438	0.401	0.298	0.117	0.123	0.159	**0.099**	0.398	0.304	0.309	0.361	0.360	0.242	0.22
thrm	0.605	0.531	0.481	0.575	0.510	0.648	0.619	0.641	0.528	0.610	0.723	0.726	0.442	**0.303**	0.692	0.648	0.496	0.711	0.539	0.340	0.334
turk	0.412	0.348	0.378	0.405	0.384	0.347	0.335	0.295	0.270	0.392	0.372	0.420	**0.105**	0.22	0.585	0.606	0.691	0.636	0.639	0.296	0.299
vgame	0.584	0.524	0.561	0.569	0.529	0.522	0.501	0.548	0.498	0.597	0.509	0.472	**0.133**	0.136	0.238	0.247	0.363	0.412	0.312	0.170	0.157
wine	0.434	0.466	0.520	0.580	0.534	0.620	0.594	0.620	0.546	0.586	0.617	0.621	0.781	0.247	0.714	0.492	0.446	0.606	0.372	0.240	**0.209**
yeast	0.358	0.380	0.298	0.407	0.328	0.431	0.362	0.593	0.595	0.534	0.401	0.340	0.702	0.213	0.585	0.234	0.296	0.501	0.325	0.224	**0.143**
Mean	0.389	0.342	0.341	0.404	0.343	0.423	0.377	0.449	0.408	0.429	0.418	0.394	0.341	**0.201**	0.497	0.385	0.403	0.490	0.379	0.246	0.222

(b) Normalized Kullback-Leibler divergence values for *natural multiclass quantifiers*

	AC	AC -LR	AC -RF	AC -AB	AC -SV	PAC	PAC -LR	TSX	TSX -LR	TSX -SV	TS50	TS50 -LR	TS50 -SV	TSMax	TSMax -LR	TSMax -SV	MS	MS -LR	MS -SV	DyS	DyS -LR	DyS -SV	FMM	FMM -LR	FMM -SV	HDy	HDy -LR	HDy -SV
conc	0.864	0.490	0.328	0.405	0.292	0.574	0.521	0.615	0.511	0.281	0.591	0.513	0.279	0.502	0.434	**0.274**	0.508	0.452	0.299	0.562	0.518	0.372	0.564	0.494	0.294	0.536	0.459	0.34
contra	0.829	0.490	0.54	0.543	0.583	0.483	0.468	0.496	0.494	0.616	0.508	0.496	0.615	0.466	0.459	0.525	0.462	**0.453**	0.519	0.538	0.575	0.675	0.467	0.463	0.551	0.481	0.487	0.569
drugs	0.228	0.157	0.351	0.270	0.211	0.166	0.165	0.170	0.158	0.185	0.177	0.165	0.177	0.171	0.168	0.193	**0.147**	0.16	0.184	0.213	0.171	0.209	0.16	0.157	0.185	0.180	0.17	0.205
craft	0.560	0.399	**0.290**	0.344	0.379	0.525	0.467	0.515	0.409	0.338	0.488	0.377	0.312	0.474	0.395	0.327	0.464	0.422	0.330	0.494	0.539	0.380	0.531	0.43	0.341	0.475	0.41	0.395
thrm	1.297	0.578	0.575	0.642	0.566	0.633	0.579	0.726	0.643	0.692	0.684	0.626	0.662	0.593	0.524	0.536	0.587	**0.521**	0.537	0.694	0.702	0.669	0.619	0.56	0.582	0.634	0.636	0.549
turk	0.651	0.382	0.691	0.643	0.577	0.326	0.338	0.375	0.378	0.614	0.392	0.401	0.632	0.349	0.361	0.49	0.348	0.359	0.490	0.455	0.432	0.568	**0.324**	**0.324**	0.472	0.372	0.382	0.492
vgame	0.741	0.591	0.699	0.707	0.640	0.640	0.586	0.630	0.604	0.613	0.626	0.598	0.611	0.574	0.543	0.54	0.575	0.548	0.547	0.557	0.557	0.567	0.6	0.568	0.572	0.521	**0.518**	0.555
wine	1.061	0.618	0.708	0.720	0.632	0.706	0.613	0.700	0.618	0.611	0.693	0.641	0.625	0.595	0.522	0.515	0.607	0.538	0.524	0.719	0.591	0.616	0.637	0.566	0.557	0.546	0.511	**0.509**
yeast	1.015	0.481	0.494	0.500	0.476	0.541	0.533	0.518	0.498	0.492	0.487	0.478	0.501	0.446	0.436	0.422	0.464	0.463	0.449	0.527	0.438	0.491	0.505	0.491	0.466	0.412	**0.398**	0.442
Mean	0.805	0.465	0.519	0.530	0.484	0.510	0.474	0.527	0.479	0.494	0.516	0.477	0.491	0.463	0.427	**0.425**	0.462	0.435	0.431	0.529	0.503	0.505	0.49	0.45	0.447	0.462	0.441	0.451

(c) Absolute error values for *one-vs.-rest-based quantifiers*

	AC	AC -LR	AC -RF	AC -AB	AC -SV	PAC	PAC -LR	TSX	TSX -LR	TSX -SV	TS50	TS50 -LR	TS50 -SV	TSMax	TSMax -LR	TSMax -SV	MS	MS -LR	MS -SV	DyS	DyS -LR	DyS -SV	FMM	FMM -LR	FMM -SV	HDy	HDy -LR	HDy -SV
conc	0.841	0.360	0.280	0.359	0.236	0.443	0.406	0.439	0.371	0.250	0.410	0.381	0.229	0.362	0.303	0.175	0.393	0.341	0.243	0.304	0.246	**0.165**	0.407	0.335	0.252	0.275	0.209	0.173
contra	0.662	0.383	0.483	0.463	0.489	0.425	0.392	0.412	0.386	0.537	0.433	0.425	0.543	0.333	0.297	0.377	0.350	0.324	0.399	0.312	0.351	0.402	0.395	0.373	0.485	**0.275**	0.28	0.337
drugs	0.164	0.104	0.288	0.191	0.134	0.100	0.145	0.125	0.123	0.110	0.091	0.093	0.095	0.074	0.089	0.080	0.087	0.120	0.099	0.069	0.054	0.079	0.108	0.127	0.109	**0.046**	0.048	0.07
craft	0.502	0.353	0.199	0.283	0.344	0.457	0.459	0.423	0.370	0.318	0.377	0.327	0.302	0.420	0.331	0.231	0.416	0.372	0.254	0.222	0.299	**0.181**	0.438	0.401	0.298	0.218	0.203	0.192
thrm	0.969	0.574	0.525	0.652	0.548	0.608	0.523	0.729	0.635	0.713	0.706	0.643	0.694	0.530	0.444	0.510	0.533	0.460	0.537	0.517	0.49	0.507	0.641	0.528	0.610	0.502	0.49	**0.418**
turk	0.580	0.353	0.636	0.592	0.494	0.320	0.317	0.377	0.361	0.548	0.396	0.379	0.560	0.260	0.245	0.389	0.259	0.243	0.391	0.274	0.277	0.331	0.295	0.270	0.392	**0.193**	0.213	0.286
vgame	0.717	0.519	0.758	0.711	0.630	0.620	0.536	0.555	0.515	0.598	0.515	0.482	0.549	0.485	0.460	0.532	0.492	0.480	0.559	0.364	**0.342**	0.389	0.548	0.498	0.597	0.385	0.375	0.427
wine	0.810	0.598	0.700	0.728	0.630	0.714	0.617	0.690	0.596	0.604	0.665	0.610	0.608	0.521	0.444	0.471	0.552	0.496	0.522	0.537	0.371	0.422	0.620	0.546	0.586	0.41	**0.334**	0.353
yeast	0.817	0.534	0.497	0.494	0.493	0.598	0.605	0.580	0.588	0.543	0.502	0.532	0.541	0.485	0.484	0.452	0.519	0.543	0.501	0.479	0.344	**0.324**	0.593	0.595	0.534	0.342	0.34	0.358
Mean	0.674	0.420	0.485	0.497	0.444	0.476	0.445	0.481	0.438	0.469	0.455	0.430	0.458	0.385	0.344	0.357	0.400	0.375	0.389	0.342	0.308	0.311	0.449	0.408	0.429	0.294	**0.277**	0.291

(d) Normalized Kullback-Leibler divergence values for *one-vs.-rest-based quantifiers*

Table 9: Results of quantifiers that use tuned base classifiers in the multiclass setting. For natural multiclass quantifiers, base classifiers were tuned with respect to their accuracy. For quantifiers that use the one-vs.-rest approach, the binary base classifiers were tuned with respect to balanced accuracy. We show error scores averaged across all scenarios per algorithm and dataset, along with the total means per algorithm (last row). Algorithms based on untuned logistic regression classifiers are denoted as before (no suffix), alternative tuned base classifiers are marked with respective suffixes: logistic regressors (LR), support vector machines (SV), random forests (RF) and AdaBoost (AB).

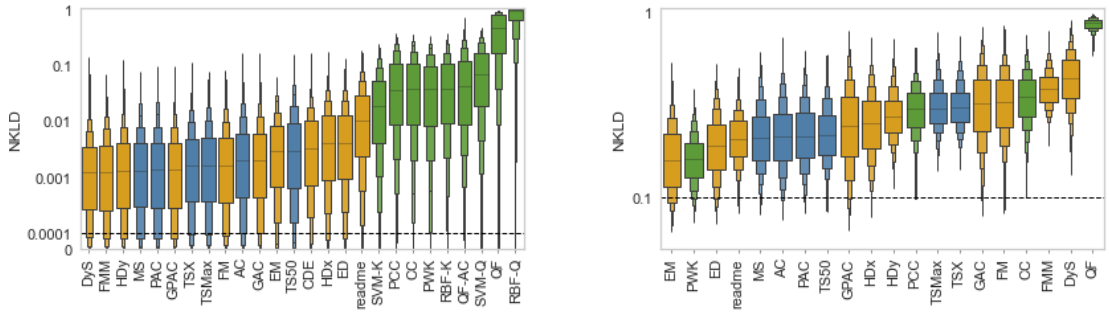
Appendix D. Parameter Settings in the LeQua Case Study

As noted in the main text, in the case study on the LeQua dataset, we used the same parameters as described in Section 4.3.1 for the experiments using untuned quantifiers, and the same parameters as described in Section 4.3.2 for the experiments with tuned base classifiers. In the same case study, we further explored the effects of tuning quantifiers with respect to AE on the given validation data. In this experiment, we chose the following parameter grids to optimize on:

- For all quantification methods that require a base classifier, a logistic regression classifier was chosen as base classifier. The parameters of this classifier were individually tuned for each quantifier, and in the corresponding grid search we varied the regularization weight C within the set $\{2^i : i \in \{-15, -13, -11, \ldots, 13, 15\}\}$. Furthermore, for all values of C, we varied the weighting strategy for the instances, either setting the weights of all instances to 1, or weighting the instances inversely proportional to the prevalence of their corresponding class. Like in all previous experiments, we applied the `L-BFGS` solver to efficiently learn the corresponding models and set the number of maximum iterations to 1000.
- For the *DyS* method, we varied the number of bins in which the confidence scores of the base classifiers were placed among the values $\{2, 4, 6, 8, 10, 15, 20\}$.
- For the *readme* method, we varied the number of features that were sampled for each subset among the values $\{2, 4, 6, 8, 10, 15, 20\}$.
- For the *PWK* method, we used the same parameter grid that was used in the experiments by Barranquero et al. (2013) when they proposed this method. Thus, we varied the number of neighbors to consider among the set $\{1, 3, 5, 7, 11, 15, 25, 35, 45\}$, and the weight factor α was varied in the set $\{1, 2, 3, 4, 5\}$.
- For the SVM$^{\text{perf}}$-based quantifiers, we tested tuning the variants of the *SVM-K* and *SVM-Q* methods which applied an RBF kernel function. Toward that end, we varied the kernel parameter γ among the values $\{2^i : i \in \{-17, -15, -13, \ldots, 3, 5\}\}$.

Appendix E. Additional Plots for the LeQua Case Study

Finally, in Figures 17 and 18, we present additional plots regarding the case study on the LeQua dataset, in which we present results with respect to NKLD. In binary data, results generally align with the results with respect to AE. By contrast, in the multiclass case results appear quite different from those with respect to AE, or related results from the main experiments, as can be seen in Figure 17(b). As discussed in the main text, we attribute this to NKLD not being particularly suitable for this setting. Thus, we omit further plots of results with respect to NKLD in the multiclass setting. In addition, we omit the plots of the NKLD values from the experiments in Section 6.3, as we argue that these are not really meaningful, given that in these experiments, methods were optimized with respect to AE.



(a) NKLD values on the binary LeQua data (b) NKLD values on the multiclass LeQua data

Figure 17: Results of our experiments with untuned quantifiers on the LeQua test sets. We present distributions of normalized Kullback-Leibler divergence (NKLD) values across all test samples. Plots are scaled logarithmically above the dotted vertical threshold, and linearly below. Colors indicate the category of the algorithm. On the binary data, overall results are mostly in line with our findings from the main experiments and results with respect to the absolute error (AE) values.

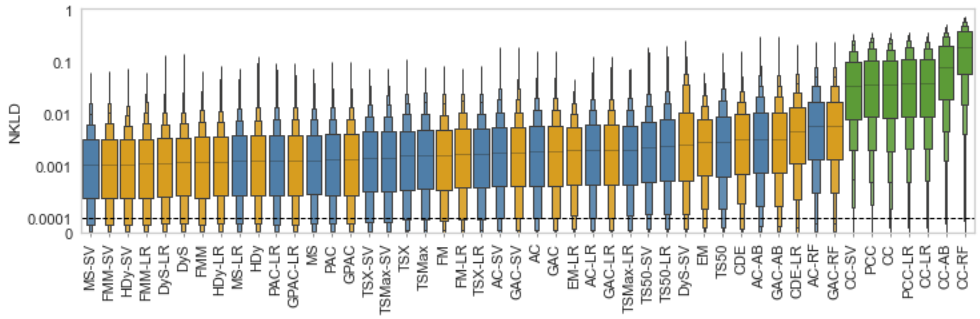


Figure 18: Results of our experiments with quantifiers that apply tuned classifiers on the binary LeQua data. We present distributions of normalized Kullback-Leibler divergence (NKLD) values across all test samples. Plots are scaled logarithmically above the dotted vertical threshold, and linearly below. Colors indicate the category of the algorithm. Algorithms based on untuned logistic regression classifiers are denoted as before (no suffix), alternative tuned base classifiers are marked with respective suffixes: logistic regressors (LR), support vector machines (SV), random forests (RF) and AdaBoost (AB).

References

Jose Barranquero, Pablo González, Jorge Díez, and Juan José del Coz. On the study of nearest neighbor algorithms for prevalence estimation in binary problems. *Pattern Recognition*, 46(2):472–482, 2013.

Jose Barranquero, Jorge Díez, and Juan José del Coz. Quantification-oriented learning based on reliable classifiers. *Pattern Recognition*, 48(2):591–604, 2015.

Antonio Bella, Cesar Ferri, José Hernández-Orallo, and María José Ramírez-Quintana. Quantification via probability estimators. In *2010 IEEE International Conference on Data Mining*, pages 737–742, Sydney, Australia, 2010.

Mirko Bunse, Alejandro Moreo, Fabrizio Sebastiani, and Martin Senz. Regularization-based methods for ordinal quantification. *Data Mining and Knowledge Discovery*, 38(6):4076–4121, 2024.

Alberto Castaño, Pablo González, Jaime Alonso González, and Juan José del Coz. Matching distributions algorithms based on the earth mover's distance for ordinal quantification. *IEEE Transactions on Neural Networks and Learning Systems*, 35(1):1050–1061, 2024.

A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 1977.

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

Michel Marie Deza and Elena Deza. *Encyclopedia of Distances*. Springer Berlin Heidelberg, Berlin & Heidelberg, Germany, 2009.

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

Zahra Donyavi, Adriane B. S. Serapião, and Gustavo Batista. MC-SQ: A highly accurate ensemble for multi-class quantification. In *Proceedings of the 2023 SIAM International Conference on Data Mining (SDM)*, pages 622–630, Minneapolis, Minnesota, 2023.

Zahra Donyavi, Adriane B. S. Serapião, and Gustavo Batista. MC-SQ and MC-MQ: Ensembles for multi-class quantification. *IEEE Transactions on Knowledge and Data Engineering*, 36(8):4007–4019, 2024.

Andrea Esuli, Fabrizio Sebastiani, and Ahmed Abasi. AI and opinion mining, part 2. *IEEE Intelligent Systems*, 25(4):72–79, 2010.

Andrea Esuli, Alejandro Moreo Fernández, and Fabrizio Sebastiani. A recurrent neural network for sentiment quantification. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, pages 1775–1778, Torino, Italy, 2018.

Andrea Esuli, Alessio Molinari, and Fabrizio Sebastiani. A critical reassessment of the Saerens-Latinne-Decaestecker algorithm for posterior probability adjustment. *ACM Transactions on Information Systems*, 39(2):1–34, 2021.

Andrea Esuli, Alejandro Moreo, and Fabrizio Sebastiani. LeQua@CLEF 2022: Learning to quantify. In *Advances in Information Retrieval: 44th European Conference on IR Research, Part II*, pages 374–381, Stavanger, Norway, 2022a.

Andrea Esuli, Alejandro Moreo, Fabrizio Sebastiani, and Gianluca Sperduti. A concise overview of LeQua@CLEF 2022: Learning to quantify. In *Experimental IR Meets Multilinguality, Multimodality, and Interaction: 13th International Conference of the CLEF Association*, pages 362–381, Bologna, Italy, 2022b. Springer.

Andrea Esuli, Alessandro Fabris, Alejandro Moreo, and Fabrizio Sebastiani. *Learning to Quantify*. Springer International Publishing, Cham, Switzerland, 2023.

Aykut Firat. Unified framework for quantification. *arXiv preprint arXiv:1606.00868*, 2016.

George Forman. Counting positives accurately despite inaccurate classification. In *Proceedings of the 16th European Conference on Machine Learning*, pages 564–575, Porto, Portugal, 2005.

George Forman. Quantifying counts and costs via classification. *Data Mining and Knowledge Discovery*, 17(2):164–206, 2008.

Jerome H. Friedman. Class counts in future unlabeled samples, 2014. Presentation at MIT CSAIL Big Data Event.

Milton Friedman. A comparison of alternative tests of significance for the problem of m rankings. *The Annals of Mathematical Statistics*, 11(1):86–92, 1940.

Pablo González, Alberto Castaño, Nitesh V. Chawla, and Juan José del Coz. A review on quantification learning. *ACM Computing Surveys*, 50(5):1–40, 2017.

Víctor González-Castro, Rocío Alaiz-Rodríguez, and Enrique Alegre. Class distribution estimation based on the Hellinger distance. *Information Sciences*, 218(1):146–164, 2013.

Waqar Hassan, André Gustavo Maletzke, and Gustavo Enrique de Almeida Prado Alves Batista. Pitfalls in quantification assessment. In *First International Workshop on Learning to Quantify: Methods and Applications (LQ 2021)*, pages 1–10, Virtual Event, Gold Coast, Australia, 2021.

Daniel J. Hopkins and Gary King. A method of automated nonparametric content analysis for social science. *American Journal of Political Science*, 54(1):229–247, 2010.

Thorsten Joachims. A support vector method for multivariate performance measures. In *Proceedings of the 22nd International Conference on Machine Learning*, pages 377–384, Bonn, Germany, 2005.

Hideko Kawakubo, Marthinus Christoffel du Plessis, and Masashi Sugiyama. Computationally efficient class-prior estimation under class balance change using energy distance. *IEICE Transactions on Information and Systems*, 99(1):176–186, 2016.

Kevin Kloos, Julian D Karch, Quinten A Meertens, and Mark de Rooij. Continuous sweep: An improved, binary quantifier. *arXiv preprint arXiv:2308.08387*, 2023.

André Maletzke, Denis dos Reis, Everton Cherman, and Gustavo Batista. DyS: A framework for mixture models in quantification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 4552–4560, Honolulu, Hawaii, 2019.

André Maletzke, Waqar Hassan, Denis dos Reis, and Gustavo Batista. The importance of the test set size in quantification assessment. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*, pages 2640–2646, Yokohama, Japan, 2020.

Letizia Milli, Anna Monreale, Giulio Rossetti, Fosca Giannotti, Dino Pedreschi, and Fabrizio Sebastiani. Quantification trees. In *2013 IEEE 13th International Conference on Data Mining*, pages 528–536, Dallas, Texas, 2013.

Alejandro Moreo and Fabrizio Sebastiani. Re-assessing the "classify and count" quantification method. In *Advances in Information Retrieval: 43rd European Conference on IR Research, Part II*, pages 75–91, 2021.

Alejandro Moreo, Andrea Esuli, and Fabrizio Sebastiani. QuaPy: A python-based framework for quantification. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, page 4534–4543, 2021.

Alejandro Moreo, Manuel Francisco, and Fabrizio Sebastiani. Multi-label quantification. *ACM Transactions on Knowledge Discovery from Data*, 18(1):1–36, 2023.

Peter B. Nemenyi. *Distribution-free multiple comparisons.* PhD thesis, Princeton University, Princeton, New Jersey, 1963.

Jeffrey Pennington, Richard Socher, and Christopher Manning. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pages 1532–1543, Doha, Qatar, 2014.

Marco Saerens, Patrice Latinne, and Christine Decaestecker. Adjusting the outputs of a classifier to new a priori probabilities: A simple procedure. *Neural Computation*, 14(1): 21–41, 2002.

Tetsuya Sakai. Evaluating evaluation measures for ordinal classification and ordinal quantification. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 2759–2769, Virtual Event, 2021.

Fabrizio Sebastiani. Evaluation measures for quantification: An axiomatic approach. *Information Retrieval Journal*, 23(3):255–288, 2020.

Amos Storkey. When training and test sets are different: Characterizing learning transfer. In Joaquin Quiñonero-Candela, Masashi Sugiyama, Anton Schwaighofer, and Neil D. Lawrence, editors, *Dataset Shift in Machine Learning*. The MIT Press, Cambridge, Massachusetts, 2008.

Dirk Tasche. Does quantification without adjustments work? *arXiv preprint arXiv:1602.08780*, 2016.

Dirk Tasche. Fisher consistency for prior probability shift. *Journal of Machine Learning Research*, 18(95):1–32, 2017.

Dirk Tasche. Confidence intervals for class prevalences under prior probability shift. *Machine Learning and Knowledge Extraction*, 1(3):805–831, 2019.

Jack Chongjie Xue and Gary M. Weiss. Quantification and semi-supervised classification methods for handling changes in class distribution. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 897–906, Paris, France, 2009.