UNIVERSITÀ DI PISA

Corso di Laurea in Informatica

TESI DI LAUREA MAGISTRALE

Predicting Classifier Accuracy under Prior Probability Shift

Relatori
Andrea Esuli
Alejandro Moreo
Fabrizio Sebastiani

Candidato
Lorenzo Volpi

ANNO ACCADEMICO 2022/2023

Abstract

Predicting the accuracy that a classifier will have on unseen data (i.e., on unlabelled data that were not available at training time) can be done via k-fold cross-validation (kFCV). However, using kFCV returns reliable predictions only when the training data and the unseen data are *identically and independently distributed* (IID), i.e., were randomly sampled from the same distribution. Unfortunately, in real-world applications it is often the case that the training data and the unseen data are not IID, i.e., that we want to deploy the trained model on unseen data that exhibit some kind of *dataset shift* with respect to the training data. In this work we deal with the problem of predicting classifier accuracy on unseen data characterised by *prior probability shift* (PPS), an important type of dataset shift. We propose a class of methods built on top of *quantification* algorithms robust to PPS, i.e., algorithms devised for estimating the prevalence values of the classes in unseen data characterised by PPS. The methods we propose are based on the idea of viewing the cells of the contingency table (on which classifier accuracy is computed) as classes. We perform systematic experiments in which we test the prediction accuracy of our methods against state-of-the-art classifier accuracy prediction methods from the machine learning literature.

Acknowledgments

This thesis was the end of a long journey, filled with lots of experiences. There are many people who, in a way or another, were close down the road, and thanking all of them one by one would be impossible. Nonetheless, some special mentions are due. Many thanks go to Andrea, Alejandro, and Fabrizio, who walked me through this last step, which would not have been possible without them; to Roberto, the one from whom this thesis started; to my mother, who believed in me and in this journey; to Rosanna, who always sides with me, and bears with me, no matter what stands in the way; and to all my friends, far and near, those who are here today and those who could not, as without them all of this would not have been the same.

Contents

Chapter 1

Introduction

Classification is one of the most important tasks in machine learning (if not *the* most important), and consists of assigning, given a predefined and finite set of classes (the *codeframe*), each data item from a domain $\mathcal{X}$ to the class (or classes) where it belongs. Classification is usually tackled by means of *supervised* learning, whereby a learning algorithm trains a *classifier* (i.e., a function that assigns data items to classes) by exposing it to a number of ("labelled") data items whose classes are known (the *training examples*). The classifier thus learns, by observing the characteristics of the data items (the *features*, or *covariates*) and how these characteristics are related to the class(es) assigned to the data items themselves, the characteristics that a new data item should have in order to be assigned to a given class.

Classification is a nondeterministic endeavour; in other words, any trained classifier may make mistakes, i.e., misclassify into class y' data items that should have instead been assigned to class y''. (Any similar deterministic endeavour, such as assigning natural numbers to classes Prime and NonPrime, is *not* considered classification). As a result, a key indicator of the quality of a trained classifier is its *accuracy*, i.e., a quantitative measure of the tendency of the classifier to assign to each data item its correct class. Mathematical measures of accuracy have been defined (one of them is "vanilla accuracy", i.e., the fraction of correct class assignments that the classifier makes), and are routinely measured, in a laboratory setting, by taking a new set of labelled items (the *test examples*), hiding their class labels, asking the classifier to classify them, and checking the classifier's assignments against the true class labels.

However, once we exit the laboratory setting and deploy the classifier onto a real-world application, we may face the problem of predicting the accuracy that the classifier will have *on the data to which we are going to apply it* (the "unseen (unlabelled) data"), whose true class labels are unknown. This task is called *classifier accuracy prediction* (CAP).

1.1 Predicting the Accuracy of a Classifier

The standard way of predicting the accuracy of a classifier on unseen data U is using *k-fold cross-validation* (kFCV). In a nutshell, k-FCV is performed by (i) partitioning the training set T into k equally-sized subsets ("folds"), (ii) classifying the data items in the i-th fold by means of a classifier trained on the other $(k-1)$ folds, (iii) repeating the previous step for all the k folds, so that all the training items have been classified, and (iv) measuring accuracy on the entire training set, by checking the automatically assigned class labels against the true class labels. For a sufficiently high value of k (typical values are 5 or 10), the accuracy value returned is a good approximation of the accuracy that a classifier trained on the entire training set would display.

However, the accuracy estimates that kFCV returns prove accurate only when the training data T and the unseen data U are *independently and identically distributed* (IID), i.e., when no *dataset shift* (Quiñonero-Candela et al., 2009) is present in the data. Unfortunately, dataset shift is ubiquitous in real-world applications, for a variety of reasons. One such reason is the possible non-stationarity of the environment across time and/or space and/or other variables, in which case the deployment conditions are irreproducible at training time. Another reason is the possible presence of sample selection bias in the training data, as when the process of labelling these data has introduced bias in them intentionally (e.g., when oversampling the minority class) or unintentionally. In all these cases, kFCV is a poor estimator of prediction accuracy for a classifier, and predicting the accuracy of a classifier on unseen data becomes a non-trivial task.

1.2 Subject and Outline of this Thesis

In this thesis we tackle the problem of predicting the accuracy of a classifier on unseen data affected by *prior probability shift* (PPS), an important type of dataset shift ((Ziegler and Czyż, 2023) even call it "the paradigmatic case" of dataset shift) in which $P(X|Y)$, the class-conditional distribution of the covariates, is invariant across the training data and the test data, but $P(Y)$, the distribution of the class labels, is not.

We propose a novel classifier accuracy prediction (CAP) method built on top of *quantification* algorithms (Esuli et al., 2023) robust to PPS, i.e., algorithms whose task is predicting the prevalence values of the classes (i.e., the class prior probabilities) in samples of unseen data affected by PPS; we call this method QuAcc ("Quantification for Accuracy Prediction"). The key idea that underlies QuAcc is that of viewing the cells of the contingency table (a.k.a. "confusion table"), on which classifier accuracy is computed, as classes, and estimating the prevalence of these classes via quantification algorithms. QuAcc can thus deal with (a) any classifier accuracy measure defined on a contingency table, and

(b) classifiers trained by any supervised learning method. QuAcc is a radically novel CAP method, since no previously proposed such method (a) is based on viewing the cells of the contingency table as classes, and (b) is built on top of quantification methods.

We present experiments in which we test QuAcc against state-of-the-art CAP methods on data generated by a robust experimental protocol, i.e., on settings characterised by many different amounts of training data imbalance, test data imbalance, and prior probability shift. In order to ensure the reproducibility of our experiments we make available the code and the data on which they are based[1]. The main focus of this thesis, for what concerns both the presentation of the method and the experimentation, will be on accuracy prediction for *binary* classifiers; however, the techniques behind QuAcc extend straightforwardly to the multiclass case.

The rest of the thesis is structured as follows. We first discuss preliminaries and the problem setting in Chapter 2. This discussion is followed, in Chapter 3, by a thorough presentation of previous works that address the problem of predicting classifier accuracy under dataset shift. We then move to describing our proposed methods in Chapter 4, and the results of our experiments in Chapter 5. In Chapter 6 we will some conclusions about our work and outline possible avenues for future research.

The work carried out in this thesis has originated a paper that we have submitted to the 41st International Conference on Machine Learning, and which is currently under revision[2].

[1] https://github.com/lorenzovolpi/QuAcc

[2] Lorenzo Volpi, Andrea Esuli, Alejandro Moreo, Fabrizio Sebastiani: Using quantification to predict classifier accuracy under prior probability shift. Submitted to the 41st International Conference on Machine Learning, Vilnius, LT. https://icml.cc/Conferences/2024

Chapter 2

Preliminaries

2.1 Notation

In this thesis we adopt the following notation. By $\mathbf{x}$ we indicate a datapoint drawn from a domain $\mathcal{X}$. By y we indicate a class drawn from a set of classes $\mathcal{Y} = \{y_1, ..., y_n\}$, which we call the *codeframe*; when $n = 2$ we write $\mathcal{Y} = \{\oplus, \ominus\}$ to indicate that $\oplus$ is the "positive" class, which is usually the minority class and represents the concept we aim to characterise, and $\ominus$ is the "negative" class. We write $(\mathbf{x}, y)$ to indicate that y is the class label of $\mathbf{x}$. By T (the training set), V (the validation set), and U (the unlabelled set) we denote three sets of labelled datapoints, where the labels of U are assumed unknown.

We use symbol σ to denote a *sample*, i.e., a non-empty set of labelled datapoints $(\mathbf{x}, y)$. We use $p_\sigma(y)$ to denote the (true) prevalence of class y in sample σ (i.e., the fraction of items in σ that belong to y); note that $p_\sigma(y)$ is just a shorthand of $\Pr(Y = y \mid \mathbf{x} \in \sigma)$, where Pr indicates probability and Y is a random variable that ranges on $\mathcal{Y}$. Note that $(p_\sigma(y_1), ..., p_\sigma(y_n)) \in \Delta^{(n-1)}$ is a *probability distribution* over $\mathcal{Y}$, where $\Delta^{(n-1)}$ denotes the probability simplex, i.e., the set of all probability distributions over n classes.

We define a *quantifier* as a function $q : 2^{\mathcal{X}} \to \Delta^{(n-1)}$, i.e., a function that maps a sample σ into a probability distribution $(\hat{p}^q_\sigma(y_1), ..., \hat{p}^q_\sigma(y_n)) \in \Delta^{(n-1)}$, where $\hat{p}^q_\sigma(y_i)$ denotes the estimate of $p_\sigma(y_i)$ returned by q. A quantifier is thus an estimator of class prevalence values, and is trained by an inductive learning algorithm on a set of labelled datapoints.

By $h : \mathcal{X} \to \mathcal{Y}$ we denote a classifier trained on T. We assume h to return labels in $\mathcal{Y}$ by the usual rule

$$h(\mathbf{x}) = \arg\max_{y_i \in \mathcal{Y}} \Pr(y_i|\mathbf{x})$$

where the $\Pr(y_i|\mathbf{x})$ (the *posterior probabilities*, which we assume to be well-calibrated) each represent the probability that the classifier subjectively attributes to the fact that $\mathbf{x}$ belongs to class y_i. This assumption brings about no loss of

generality, since a classifier that returns generic confidence scores (instead of posterior probabilities) can be made to return well-calibrated posterior probabilities by means of a calibration function.

By $A(h, U)$ we denote a function that measures the accuracy of classifier h on an unlabelled set U, while by $E(A(h, U), \hat{A}(h, U))$ we denote a function that measures the CAP error incurred by estimating $A(h, U)$ as $\hat{A}(h, U)$.

2.2 Dataset Shift and its Subtypes

Standard supervised learning algorithms are based on the assumption that the training distribution, on which a model is trained, and the unlabelled distribution, on which the model is supposed to predict, are *independently and identically distributed* (IID). In other words, since labelled datapoints are represented by pairs of type $(\mathbf{x}, y)$, the distribution of pairs in the labelled set is assumed to be the same as that in the set of unlabelled items, i.e.,

$$p_T(\mathbf{x}, y) = p_U(\mathbf{x}, y)$$

Of particular interest to quantification tasks is the fact that the distribution of labels is assumed to stay constant, i.e., $p_T(y) = p_U(y)$.

Data provided by the world we live in, though, are constantly evolving, and the scenarios in which we would like to deploy trained models may differ substantially. In many cases, this variety results in a violation of the IID assumption and $p_T(\mathbf{x}, y) \neq p_U(\mathbf{x}, y)$. This phenomenon is usually referred to as *dataset shift*; when dataset shift is present, the data from $p_U(\mathbf{x}, y)$ are often called *out-of-distribution (OOD) data*.

It is interesting to note that $p(y)$ may be written as

$$p(y) = \sum_{\mathbf{x}} p(y|\mathbf{x})p(\mathbf{x})$$

This formulation of $p(y)$ can be useful to assess the impact of distribution shift on quantification tasks and their applications.

When any of $p(y|\mathbf{x})$ and $p(\mathbf{x})$ vary when switching from training data to unlabelled data, distribution shift occurs.

The scenario in which $p(\mathbf{x})$ varies happens when certain regions of the feature space are more densely populated in U than in T while, conversely, other regions are more densely populated in T than in U. This phenomenon is called *covariate shift*. It is important to note that, under covariate shift, while $p(\mathbf{x})$ varies, $p(y|\mathbf{x})$ stays the same (see Figure 2.1 for an example).

The case in which $p(y|\mathbf{x})$ varies, instead, occurs when the meaning of class y has changed and the same item $\mathbf{x}$ is labelled differently in T and U. This is known as *concept shift*. Also in this case, when concept shift takes place, $p(\mathbf{x})$ stays put while $p(y|\mathbf{x})$ varies.

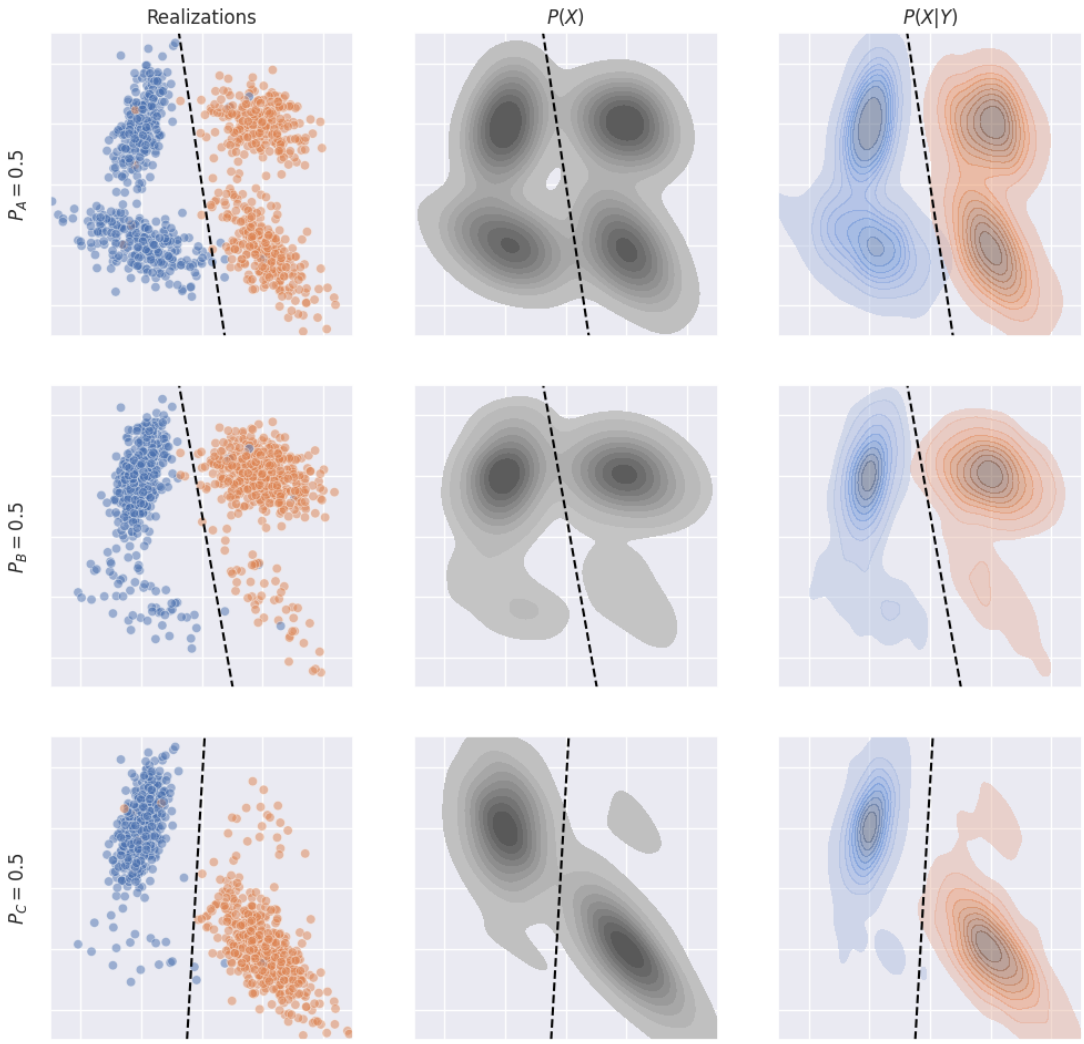

Figure 2.1: Example of covariate shift generated with synthetic data using a normal distribution for each cluster. Situation (a) (1st row): original data distribution. Each class consists on two clusters of data (for example positive or negative opinions of two different categories: ELECTRONICS and BOOKS). Situation (b) (2nd row): there is a shift in the number of opinions of one category, that affects both classes. $P(X)$ changes (see 2nd column) but $P(Y|X)$ remains invariant. Situation C (3rd row), $P(X)$ changes abruptly, affecting the posterior probabilities $s(\mathbf{x})$ that a soft classifier, trained via induction on this scenario, would issue. Image taken from (González et al., 2023).

Covariate shift, concept shift and, in general, distribution shift, fall under the so-called category of $\mathcal{X} \rightarrow \mathcal{Y}$ *problems* (Fawcett and Flach, 2005), i.e., problems in which it is the values of the features in $\mathbf{x}$ that probabilistically determines the label y of $\mathbf{x}$.

Similarly to what we have seen above, we can rewrite $p(\mathbf{x})$ using the following

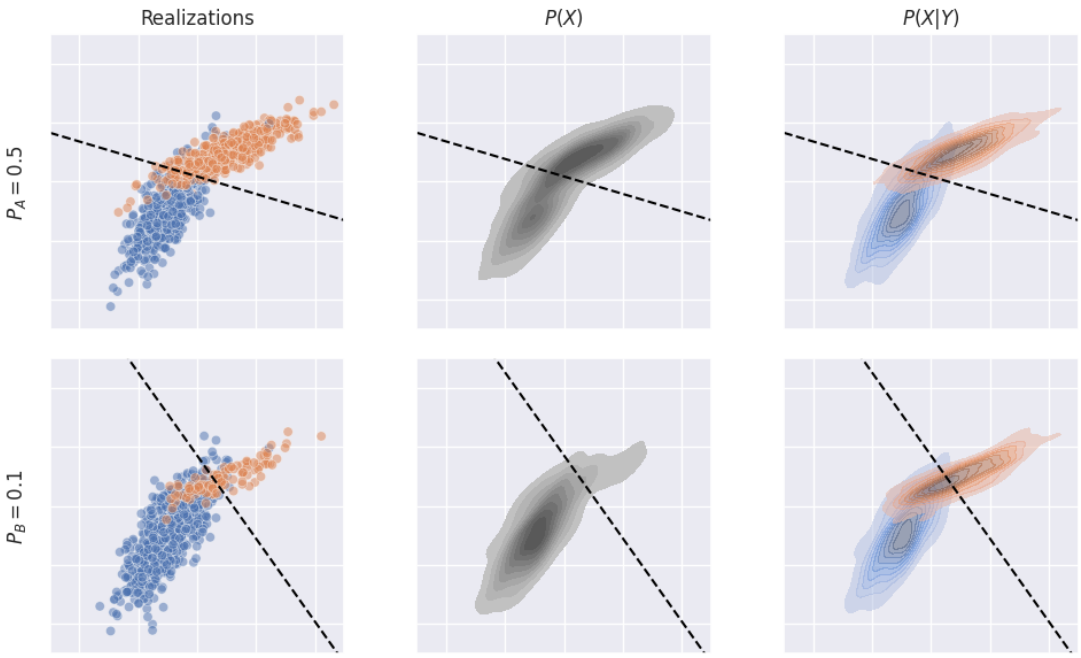

Figure 2.2: Example of prior probability shift generated with synthetic data using a normal distribution for each class. Scenario A (1st row): original data distribution, in which the positive class (orange) and the negative class (blue) have the same prevalence, i.e., $p_A = 0.5$. Scenario B (2nd row): with respect to Scenario A there is a shift in the prevalence such that $p_B = 0.1$. Dashed lines represent linear hypotheses learnt from the corresponding empirical distributions. Note that, although the positive class and the negative class may have not changed in meaning between A and B, i.e., $P_A(Y|X) = P_B(Y|X)$, the posteriors we would obtain by calibrating two soft classifiers trained from the two empirical distributions would likely differ. Note also that $P_A(X) \neq P_B(X)$ (2nd column) but $P_A(X|Y) = P_B(X|Y)$ (3rd column). Image taken from (González et al., 2023).

formulation:

$$p(\mathbf{x}) = \sum_{y} p(\mathbf{x}|y)p(y)$$

This formulation is relevant in what Fawcett and Flach (2005) defined $\mathcal{Y} \rightarrow \mathcal{X}$ *problems*, i.e., problems in which the class to which a datapoint $\mathbf{x}$ belongs probabilistically determines the values of the features in vector $\mathbf{x}$.

In $\mathcal{Y} \rightarrow \mathcal{X}$ *problems*, when $p(y)$ varies between T and U, it does so independently, as y is a cause and not an effect. When this happens we speak about *prior probability shift* (Storkey, 2009) or *label shift* (Alexandari et al., 2020) (see Figure 2.2 for an example).

In our work we will only focus on *prior probability shift* (PPS), leaving the research on other kinds of shift to future work.

2.3 Classification

Classification is a supervised machine learning task where a model tries to predict the correct label of an input datapoint. Being a *supervised learning* method, the model used is trained on a dataset, a collection of datapoints, for which the correct labels are known.

The aspect of classification we are most interested in is *accuracy*. A perfect model that tries to predict the correct labels of an unlabelled dataset will make no errors. If the model is not perfect, it will misclassify some data points of the unlabelled dataset. The *accuracy* of the model while predicting labels over an unlabelled dataset measures how close the prediction is to ground-truth information.

2.3.1 Evaluation Measures for Classification

Many measures can be used to assess classifier accuracy. The ones we will mainly use and consider are *vanilla accuracy* and F_1.

2.3.1.1 Vanilla Accuracy

Vanilla accuracy measures the fraction of correct predictions output by a classifier. If $\mathcal{X}$ is the source dataset for the classifier h, $\hat{y}_i$ the class predicted by h for the i-th datapoint and y_i its true class, then the fraction of correct predictions over $\mathcal{X}$ is

$$\text{Acc}(\hat{y}, y) = \frac{1}{|\mathcal{X}|} \sum_{i=1}^{|\mathcal{X}|} \mathbb{1}(\hat{y}_i = y_i) \tag{2.1}$$

where Acc stands for *vanilla accuracy* and $\mathbb{1}(x)$ is the indicator function.

2.3.1.2 F_1

F_1 is a special case of F_β, which can be seen as a weighted harmonic mean of precision and recall. F_1 gives equal importance to precision and recall. It is an error measure especially useful in cases of class imbalance. In these cases, e.g., when the number of negative datapoints is substantially higher than the number of positives, a *trivial rejector* (a classifier that always predicts the negative label for a datapoint) would obtain great results if vanilla accuracy is used. This is because a measure like vanilla accuracy tends to favour trivial classifiers (models that are not driven by the contents of training datasets) in cases of class imbalance. F_1 is an error measure that tries to discourage trivial classifiers by imposing a non-linear relation between the correctly and incorrectly classified datapoints. As an example, with a large disproportion in favour of negative datapoints in a given distribution, while a trivial rejector would give a prediction resulting in a

high vanilla accuracy, its prediction would result in an F_1 value equal to 0, given how it is computed in Equation 2.2.

In the binary case, we can see precision as the ability of a classifier not to label as positive a datapoint that is negative and recall as the ability to find all positive datapoints. Suppose to have the following contingency table for a binary classifier:

TN	FP
FN	TP

then, we can compute precision and recall as follows:

$$P = \frac{\text{TP}}{\text{TP} + \text{FP}} \qquad R = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

where p stands for precision and r for recall. The value of F_1 is then computed as:

$$F_1 = 2\,\frac{P \times R}{P + R}$$

which, for the binary case, resolves to:

$$F_1 = \frac{2\,\text{TP}}{2\,\text{TP} + \text{FP} + \text{FN}} \tag{2.2}$$

This is what is known as the *binary* formulation for F_1, which is fully computed around the class labelled as positive, ignoring the other class.

2.4 Quantification

As observed by Forman (2005), in several applications involving classification, the final goal is not determining which class (or classes) single unlabelled datapoints belong to, but estimating the *prevalence* (also called, "relative frequency", "prior probability" or simply "prior") of each class in the unlabelled data.

The task of training class prevalence estimators via supervised learning has come to be known as *quantification* (Forman, 2005); the term *learning to quantify* is also used and widely accepted.

Quantification appears to be relevant in all those scenarios in which the interest is not at the individual level and the aggregate level is all that matters. Indeed, there are plenty of fields of human inquiry which are devoted to studying phenomena only at a collective level, such as market research, social sciences, epidemiology, to name a few.

Without methods for estimating class prevalence values more directly, the obvious method for doing so is *Classify and Count*, i.e., classifying each unlabelled

datapoint and estimating class prevalence values by counting the number of items assigned to each class.

This approach, however, appears to be suboptimal: while a perfect classifier is also a perfect quantifier, a good classifier can be a bad quantifier. To explain this, let's assume we have a couple of classifiers, h_1 and h_2. If h_1 mispredicts m_1 datapoints and h_2 makes m_2 errors in its predictions with $m_1 > m_2$ we can say that h_2 is a better classifier. Nonetheless, if the number of *false positives* and *false negatives* predicted by h_1 matches, h_1 results to be a perfect quantifier, since the predicted class prevalence values perfectly match the true one. Even if h_2 is a better classifier than h_1, it can be a worse quantifier if the number of *false positives* and *false negatives* differs.

This means that even a good classifier can be *biased*, as long as it tries to keep low the number of false positives at the expense of a higher number of false negatives. This phenomenon is not infrequent, especially in the presence of imbalanced data.

Generally, literature discerns two large classes of quantification methods:

- *aggregative methods* are methods that require the classification of all the individual data items as an intermediate step; this class includes both methods relying on general-purpose learners and methods using special-purpose learning methods devised with quantification in mind; *Classify and Count* falls inside this class;

- *non-aggregative methods* solve the quantification problem as a whole, i.e., without classifying individual items.

Our focus will be on *aggregative methods*. Sections 2.4.1 and 2.4.2 will be about the two main quantification we focused on in our work. Section 2.4.3 will include a brief discussion about other quantification algorithms we took into account in our preliminary studies. Lastly, Section 2.4.4 will focus on possible evaluation measures used with quantification.

2.4.1 The SLD Quantification Method

While many quantification methods have an *inductive* nature, since the quantification model is trained exclusively on the training set, other methods have a *transductive* nature, i.e., they are trained by also looking at certain characteristics of the unlabelled data they are going to predict upon (although, not at their labels).

The *Saerens-Latinne-Decaestecker* (SLD) algorithm, proposed by Saerens et al. (2002), has a transductive component, since it applies a transductive correction to the test predictions issued by an inductive classifier.

SLD is an instance of *Expectation Maximisation* (Dempster et al., 1977), a well-known iterative method for finding maximum-likelihood estimates of param-

Input : Class prevalence values $p_T(y)$ on T, for all $y \in \mathcal{Y}$;
Posterior probabilities $p(y|\mathbf{x})$, for all $y \in \mathcal{Y}$ and for all $\mathbf{x} \in U$;
Output : Estimates $\hat{p}_U(y)$ of class prevalence values on U;

/* Initialisation */
$s \leftarrow 0$;
for $y \in \mathcal{Y}$ **do**
  $\hat{p}_U^{(s)}(y) \leftarrow p_T(y)$;
  for $\mathbf{x} \in U$ **do**
    $p^{(s)}(y|\mathbf{x}) \leftarrow p(y|\mathbf{x})$;
  end
end
/* Main Iteration Cycle */
while *stopping condition* = *false* **do**
  $s \leftarrow s + 1$;
  for $y \in \mathcal{Y}$ **do**
    for $\mathbf{x} \in U$ **do**

$$p^{(s)}(y|\mathbf{x}) \leftarrow \frac{\frac{\hat{p}_U^{(s-1)}(y)}{\hat{p}_U^{(0)}(y)} \cdot p^{(0)}(y|\mathbf{x})}{\sum_{y \in \mathcal{Y}} \frac{\hat{p}_U^{(s-1)}(y)}{\hat{p}_U^{(0)}(y)} \cdot p^{(0)}(y|\mathbf{x})};$$

    end
    $\hat{p}_U^{(s)}(y) \leftarrow \frac{1}{|U|} \sum_{\mathbf{x} \in U} p^{(s)}(y|\mathbf{x})$;
  end
end
/* Generate output */
for $y \in \mathcal{Y}$ **do**
  $\hat{p}_U(y) \leftarrow \hat{p}_U^{(s)}(y)$;
end

Algorithm 1: The SLD algorithm (Saerens et al., 2002).

eters (the class prevalence values) for models that depend on unobserved values (the class labels).

SLD incrementally updates the posterior probabilities by using the class prevalence values computed in the last step of the iteration and updates the class prevalence values by using the posterior probabilities computed in the last step of the iteration in a mutually recursive fashion.

More in depth, the SLD algorithm updates, at each iteration s, the posterior probability $p^{(s)}(y|\mathbf{x})$ with the fraction between the posterior of y at the previous iteration $(s-1)$ and the training prevalence of y (since $\hat{p}_U^{(0)}(y) = p_T(y)$), all weighted with the posterior probability obtained by the inductive classifier $p^{(0)}(y|\mathbf{x})$. This value is then normalised with the sum of the same value computed on all $y \in \mathcal{Y}$. After this step, the posterior probability for y at iteration s

are computed as the mean of all posteriors $p^{(s)}(y|\mathbf{x})$.

It must be noted that the transductive step in the SLD algorithm is there to try and improve the accuracy of posterior probabilities predicted by the underlying inductive classifier, i.e., it represents a strategy to adjust the posteriors we would obtain using the *Classify and Count* method using the said classifier.

2.4.2 The KDEy Quantification Method

KDEy is a quantification algorithm proposed by Moreo et al. (2023) that relies on the assumption that the source and target distributions are related to each other by means of prior probability shift. KDEy builds on top of *distribution-matching* (DM) methods and goes one step further by improving the application to the multiclass case, a scenario in which traditional DM methods perform suboptimally.

Before delving deeper into the particularities of KDEy, it may be convenient to succinctly review the rationale behind general DM approaches.

DM approaches aim at reconstructing the distribution of the test datapoints by looking for the closest mixture of the class-conditional distributions of the training datapoints. Modelling distributions is known to be difficult, and especially so when the covariates live in a very large-dimensional space. In order to deal with this limitation, most DM approaches prefer to transform the datapoints into posterior probabilities by means of a probabilistic classifier; the rationale is that the space of posterior probabilities is much easier to handle than the original space of covariates. Current DM methods model the distribution of posteriors by means of class-specific *histograms*. In the binary case, this comes down to generating one histogram for the distribution of posteriors from the positive training instances, and another for the distribution of posteriors of negative training instances.

The first distribution matching method for quantification was proposed by Forman (2005) and was originally called Mixture Model (MM). MM was devised for binary quantification only and is based on the observation that

$$\mathbb{Q} = \alpha Q_1 + (1 - \alpha) Q_0$$

i.e., on the fact that the distribution of test instances ($\mathbb{Q}$) is a mixture of the class-conditional distribution of positives (Q_1) and of the negatives (Q_0), where α is the mixture parameter, which corresponds to the proportion of positives (i.e., the sought prior probability). Here, distributions $\mathbb{Q}$, Q_1 and Q_0 are represented by means of discrete cumulative distribution function of classifier scores $\tilde{Q}$, $\tilde{Q}_1$, $\tilde{Q}_0$. Of course, $\tilde{Q}_1$ and $\tilde{Q}_0$ are unknown since labels for test distribution are not observed. However, under PPS conditions the class-conditional probability distributions are assumed stationary, so one can safely assume that $\tilde{Q}_1 \approx \tilde{P}_1$ and $\tilde{Q}_0 \approx \tilde{P}_0$. MM tries to reconstruct the test distribution as a linear combination

of the class-conditional training distributions, with respect to any given specific loss function $\mathcal{L}$:

$$\alpha^* = \arg\min_{0\leq\alpha\leq 1} \mathcal{L}(\alpha\tilde{P}_1 + (1-\alpha)\tilde{P}_0, \tilde{Q})$$

Later on, many variants of MM were proposed, varying on the loss function $\mathcal{L}$, the way the discrete distributions were presented and also replacing cumulative distributions with histograms. HDy is a renowned example that uses the Hellinger Distance (HD) as a loss function, a method that has become highly influential in the field. Many adaptations to the multiclass scenario were then attempted for DM methods (Bunse and Morik, 2022; Firat, 2016). All these attempts, however, suffer from three main limitations:

1. A native extension of histograms to the multiclass case scales combinatorially, in terms of computational cost, with the number of classes and bins in the histogram.

2. Binning posterior probabilities is akin to performing hard assignments to the bin centres. These bin centres are data-agnostic, and the hard assignment incurs a loss of information.

3. Modelling samples as a set of class-specific histograms is efficient, but fails to capture inter-class information.

This is the point at which KDEy made its appearance. KDEy was proposed as a means to overcome the above-mentioned limitations in the multiclass regime. The main idea behind KDEy is to switch the problem representation from discrete, univariate PDFs (histograms) of the posterior probabilities to continuous, multivariate PDFs on the $(n-1)$-simplex. The new PDFs are represented via Gaussian Mixture Models (GMMs) obtained via kernel density estimation (hence its name). At training time, the quantifier generates a representation (a KDE function) for each of the class-conditional distributions of posterior probabilities $s(\mathbf{x})$ returned by or soft classifier. Given a set of reference points for each class $(\tilde{T}_1, \ldots, \tilde{T}_n)$, where $\tilde{T}_i = \{s(\mathbf{x}) : \mathbf{x} \in T_i\}$ and $T_i = \{\mathbf{x} : (\mathbf{x}, y) \in T \wedge y = i\}$, and a mixture vector $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n) \in \boldsymbol{\Delta}^{(n-1)}$, where $\boldsymbol{\Delta}^{(n-1)}$ is the $(n-1)$-simplex (or probability simplex) defined as $\boldsymbol{\Delta}^{(n-1)} = \{(p_1, \ldots, p_n) : p_i \geq 0, \sum_{1\leq i\leq n} p_i = 1\}$, the density of a datapoint $\mathbf{x} \in \mathcal{X}$ is estimated by plugin-in the posterior probability $s(\mathbf{x})$ into the KDE mixture $\boldsymbol{p}_{\boldsymbol{\alpha}} : \boldsymbol{\Delta}^{(n-1)} \to \mathbb{R}_{\geq 0}$, which is defined as:

$$\boldsymbol{p}_{\boldsymbol{\alpha}}(\tilde{\mathbf{x}}) = \sum_{i=1}^{n} \alpha_i p_{\tilde{T}_i}(\tilde{\mathbf{x}})$$

in which $p_R : \boldsymbol{\Delta}^{(n-1)} \to \mathbb{R}_{\geq 0}$ is the GMM given by:

$$p_R(\mathbf{x}) = \frac{1}{|R|} \sum_{\mathbf{x}_i \in R} K\left(\frac{\mathbf{x} - \mathbf{x}_i}{h}\right)$$

that uses the datapoints in R (called "the reference set") as kernel centres and where h is the bandwidth parameter of the Gaussian kernel K.

The distribution matching problem is framed in terms of the following optimisation problem:

$$\hat{p}_U^{\text{KDEy}} = \underset{\boldsymbol{\alpha} \in \boldsymbol{\Delta}^{(n-1)}}{\arg\min}\, \mathcal{D}(\boldsymbol{p}_{\boldsymbol{\alpha}} || q_{\tilde{U}})$$

where $q_{\tilde{U}}$ is the KDE function modelled on the reference set $\tilde{U} = \{s(\mathbf{x}) : \mathbf{x} \in U\}$ that represents the unlabelled test set U, and where $\mathcal{D}$ is any divergence function measuring the degree of discrepancy between the mixture distribution and the test distribution.

Different such divergences were tested in Moreo et al. (2023), including the (squared) Hellinger Distance, the Cauchy-Schwarz divergence, and the Kullback-Leibler divergence. In this work, we rely on the latter (a configuration dubbed KDEy-ML as the problem becomes a maximum likelihood estimate) since this is the best-performing variant among the ones explored in the original paper.

Note that, while we mainly focus on binary *classification* problems, we are instead particularly interested in *multiclass* quantification since many of the approaches we define recast the binary problem in terms of a multiclass one.

2.4.3 Other Quantification Methods

We have explored other quantification algorithms in our research and considered the possibility to include them as a basis to our method. We have, eventually, decided not to include them because the methods described above appeared to be more advanced and to give higher performance. Nonetheless, we will briefly describe them in the following sections.

2.4.3.1 Classify and Count

Classify and Count (CC) quantification algorithm is the most obvious method for this machine learning task. It consists of training a classifier h on a labelled dataset T via a standard learning algorithm, classifying the items in an unlabelled dataset U and estimating the fraction of items in U belonging to class y $p_U(y)$ by simply counting the number of items in U predicted to belong to y by h and dividing this number by the cardinality of U. This corresponds to computing:

$$\begin{aligned}\hat{p}_U^{\text{CC}}(y) &= p_U^h(\hat{y}) \\ &= \frac{|\{\mathbf{x} \in U | h(\mathbf{x}) = y\}|}{|U|}\end{aligned}$$

The name of this method comes from the definition given by Forman (2008).

CC is suboptimal because standard classifiers might be biased, i.e., generate unbalanced numbers of false negatives and false positives, and because they

are trained to minimise classification error, which can be deeply distant from quantification error.

However, CC represents a reasonable baseline for other quantification method and was one of the quantification methods we firstly tried in our tests.

2.4.3.2 Probabilistic Adjusted Classify and Count

Probabilistic Adjusted Classify and Count (PACC) is a probabilistic variant of *Adjusted Classify and Count* (ACC).

ACC, much like CC, works by training a classifier h over a labelled set T, classifying the items in an unlabelled set U, and then observing that it holds that:

$$p_U^h(\hat{y}_j) = \sum_{y_i \in \mathcal{Y}} p_U^h(\hat{y}_j|y_i) \cdot p_U(y_i) \tag{2.3}$$

Here, $p_U^h(\hat{y}_j)$ represents the fraction of items in U that have been assigned to y_j by h and $p_U^h(\hat{y}_j|y_i)$ is the fraction of items in U whose true class is y_i and have been assigned to class y_j by h. While these two values can be observed (the former) or estimated from T via k-fold cross-validation (the latter), the value $p_U(y_i)$ is unknown and is what we are interested in. Since Equation 2.3 defines $|\mathcal{Y}|$ equations with $|\mathcal{Y}|$ variables of type $p_U(y_i)$ we have a system of $|\mathcal{Y}|$ linear equations. This system can then be solved to estimate the $\hat{p}_U^{\text{ACC}}(y_i)$ values.

PACC works in a similar way to ACC, with the underlying idea of replacing both sides of Equation 2.3 with their expected values. Equation 2.3 then becomes:

$$\begin{aligned} E\big[p_U^h(\hat{y}_j)\big] &= E\left[\sum_{y_i \in \mathcal{Y}} p_U^h(\hat{y}_j|y_i) \cdot p_U(y_i)\right] \\ &= \sum_{y_i \in \mathcal{Y}} E\Big[p_U^h(\hat{y}_j|y_i) \cdot p_U(y_i)\Big] \\ &= \sum_{y_i \in \mathcal{Y}} E\big[p_U^h(\hat{y}_j|y_i)\big] \cdot p_U(y_i) \end{aligned} \tag{2.4}$$

where

$$\begin{aligned} E\big[p_U^h(\hat{y}_j)\big] &= \frac{1}{|U|} \sum_{\mathbf{x} \in U} p(y_j|\mathbf{x}) = \hat{p}_U^{\text{PACC}}(y_i) \\ E\big[p_U^h(\hat{y}_j|y_i)\big] &= \frac{1}{|U_i|} \sum_{\mathbf{x} \in U_i} p(y_j|\mathbf{x}) \end{aligned}$$

in a similar way to ACC, $E\big[p_U^h(\hat{y}_j)\big]$ can be observed and $E\big[p_U^h(\hat{y}_j|y_i)\big]$ can be estimated via k-fold cross-validation. So we are again in front of $|\mathcal{Y}|$ linear equations with $|\mathcal{Y}|$ variables which can be solved in a linear system.

PACC was first proposed by Bella et al. (2010) while ACC is actually very old, since its binary version was most likely proposed by Gart and Buck (1966).

Both ACC and PACC can return values for $\hat{p}_U(y_i)$ that fall off the $[0, 1]$ range. To address this issue, Esuli et al. (2023) propose to apply a softmax as an improvement with respect to methods (i.e., "clipping") proposed by other authors such as Forman (2008).

2.4.4 Evaluation Measures for Quantification

We will describe, in this section, some relevant evaluation measures used for the quantification task. These measure are aimed to compute the distance between the predicted prevalence of labels and the true one. Even though we did not use these measures directly in our experiments, we consider them a good starting point for the actual evaluation measures used in our work.

2.4.4.1 Absolute Error

Absolute Error (AE) (Saerens et al., 2002) is defined as:

$$\mathrm{AE}(p, \hat{p}) = \frac{1}{|\mathcal{Y}|} \sum_{y \in \mathcal{Y}} |\hat{p}(y) - p(y)|$$

This measure enforces the fact that both negative and positive *bias* (the distance between true and predicted prevalence for a class) are equally undesirable. Moreover, this measure is crafted in such a way that an error should be penalised independently of the value of the true class prevalence. AE ranges between 0 (best) and z_{AE} (worst), where z_{AE} is defined as:

$$z_{\mathrm{AE}} = \frac{2(1 - \min_{y \in \mathcal{Y}} p(y))}{|\mathcal{Y}|}$$

i.e., its range depends on the true distribution p and the cardinality of $\mathcal{Y}$.

Normalised Absolute Error (NAE) (Esuli and Sebastiani, 2014) is a variant of AE that always ranges between 0 and 1 and is defined as follows:

$$\mathrm{NAE}(p, \hat{p}) = \frac{\mathrm{AE}(p, \hat{p})}{z_{\mathrm{AE}}} = \frac{\sum_{y \in \mathcal{Y}} |\hat{p}(y) - p(y)|}{2(1 - \min_{y \in \mathcal{Y}} p(y))}$$

2.4.4.2 Relative Absolute Error

Relative Absolute Error (RAE) (González-Castro et al., 2010) is defined as

$$\mathrm{RAE}(p, \hat{p}) = \frac{1}{|\mathcal{Y}|} \sum_{y \in \mathcal{Y}} \frac{|\hat{p}(y) - p(y)|}{p(y)}$$

and is a refinement of AE that enforces the fact that the error is more serious when the true class prevalence is smaller. Also in this case, as for AE, both underestimation and overestimation of the true class prevalence are equally penalised. RAE ranges between 0 (best) and z_{RAE} (worst), where z_{RAE} is defined as:

$$z_{\text{RAE}} = \frac{|\mathcal{Y}| - 1 + \frac{1 - \min_{y \in \mathcal{Y}} p(y)}{\min_{y \in \mathcal{Y}} p(y)}}{|\mathcal{Y}|}$$

i.e., its range depends on the true distribution p and the cardinality of $\mathcal{Y}$.

Normalised Relative Absolute Error (NRAE) (Esuli and Sebastiani, 2014) is a version of RAE that ranges between 0 and 1 and can be obtained as:

$$\text{NRAE}(p, \hat{p}) = \frac{\text{RAE}(p, \hat{p})}{z_{\text{RAE}}} = \frac{\sum_{y \in \mathcal{Y}} \frac{|\hat{p}(y) - p(y)|}{p(y)}}{|\mathcal{Y}| - 1 + \frac{1 - \min_{y \in \mathcal{Y}} p(y)}{\min_{y \in \mathcal{Y}} p(y)}}$$

It must be noted that, even though NRAE is considered a *relative* error, it does not enforce either the fact that the seriousness of the error depends on the true class prevalence, or that the error is independent of this value.

2.4.4.3 Kullback-Leibler Divergence

Kullback-Leibler Divergence (KLD) (Forman, 2005), also known as *normalised cross-entropy*, is defined as:

$$\text{KLD}(p, \hat{p}) = \sum_{y \in \mathcal{Y}} p(y) \log \frac{p(y)}{\hat{p}(y)}$$

KLD ranges between 0 (best) and $+\infty$ (worst). This means that KLD is not upper-bounded.

Normalised Kullback-Leibler Divergence (NKLD) (Esuli and Sebastiani, 2014) is an upper-bounded variant of KLD which varies between 0 (best) and 1 (worst). It is defined as:

$$\text{NKLD}(p, \hat{p}) = 2 \frac{e^{\text{KLD}(p, \hat{p})}}{e^{\text{KLD}(p, \hat{p})} + 1} - 1$$

Unfortunately, Sebastiani (2020) shows how both KLD and NKLD do not satisfy any of the properties described for the other measures, i.e., there is no relation between the seriousness of the error and the value of the true class prevalence, the error is not necessarily penalised independently of the value of the true class prevalence, and overestimation and underestimation of the true class prevalence is not necessarily equally penalised. As stated by Esuli et al. (2023), these deficiencies make the use of KLD and NKLD as evaluation measures for quantification questionable.

Chapter 3

Previous Approaches to Predicting Classifier Accuracy on Out-of-Distribution Data

Although the problem of estimating a pre-trained classifier accuracy on an unlabelled test set is practically very useful and can have many real-world applications, it seems to receive less attention than other classification-related problems.

In the following sections, we disclose some of the main works of the past years, focusing first on those that we consider of greater relevance and then, in Section 3.6, briefly reporting those related to the subject that we deemed of a more relative interest.

It must be noted that, while treating previous works, we describe them in the context of the binary classification problem, which is the main focus of our work and is often the path chosen also in many of the works we describe. Nonetheless, most the these approaches, such as ours, are able to perform also in the multiclass formulation of the problem.

3.1 *k*-Fold Cross-Validation

Many methods have been proposed to estimate and predict the accuracy of a classification model on unlabelled datasets. The best-known and most well established baseline is easily *k-fold cross-validation* (kFCV).

The basic idea under this approach involves holding out a test set for final evaluation from a labelled training set T and splitting the remaining data into k smaller sets, also called *folds*. A model is then trained using $(k-1)$ of the folds and the remaining part of the data, the test portion, is used as a test set to compute a performance measure of the trained model. This procedure is usually repeated k times, each time leaving out a different fold, and the resulting performance measure obtained by k-fold cross-validation is the average of the

measures computed on the test data in the k loops. kFCV and cross-validation in general have many variants, but the one described is one of the most widely used.

kFCV can be a powerful tool, even though it has its own downsides:

- First, it tries to predict the performance of a classifier only taking into account the data the classifier is trained on. This means that the predicted performance is the same for any unlabelled dataset the model is checked against and that no information about any kind of shift on test data is taken into account. For this reason, the estimation given by kFCV is flat and stays the same at any time and in every condition.

- Moreover, kFCV estimates performance by retraining the same model each time on a different portion of the training set, trying to catch different aspects and characteristics of this set. This can be costly, depending on the size of the folds and their number, and can fail to consider a wide enough range of possible shifts and variations to make the computed measure actually relevant, especially on test sets which are consistently distant from the training set.

For this reason we look at other approaches that, by overcoming these downsides, tend to result as more interesting methods.

3.2 Average Thresholded Confidence (ATC)

Average Thresholded Confidence (ATC) is a practical method proposed by Garg et al. (2022) for predicting the target domain accuracy in real-world classification settings characterised by distribution shift between source (training) and target (test) distributions. It works by using only labelled source data and unlabelled target data. It learns a *threshold* on the model's confidence, predicting accuracy as the fraction of unlabelled examples for which model's confidence exceeds that threshold.

ATC method defines, at first, a score function $s : [0, 1]^n \to \mathbb{R}$, where $n = |\mathcal{Y}|$, that takes as input the posterior probabilities output by the classifier h and outputs a scalar. The score functions explored by Garg et al. (2022) are:

- *maximum confidence* (*MaxConf*): $s(\mathbf{p}) = \max_{j \in \mathcal{Y}} p_j$

- *negative entropy* (*NegEnt*): $s(\mathbf{p}) = \sum_{j \in \mathcal{Y}} p_j \log p_j$

where $\mathbf{p} = \{p_1, \ldots, p_n\}$ are the posterior probabilities obtained from h for a datapoint $\mathbf{x}$.

The method finds a threshold t on the source dataset T such that the expected number of datapoints that obtain a score less than t match the error of h on T:

$$\mathbb{E}_{\mathbf{x}\sim T}\big[\mathbb{I}[s(\mathbf{p}) < t]\big] = \mathbb{E}_{(\mathbf{x},y)\sim T}\big[\mathbb{I}[\arg\max_{j\in\mathcal{Y}} p_j \neq y]\big] \tag{3.1}$$

where y is the ground truth label for a datapoint $\mathbf{x}$.

The vanilla accuracy estimate $A_{\text{ATC}_s}(U)$ given by ATC on a score function s on the target dataset U is then given by the expected number of target datapoints that obtain a score greater than t:

$$A^s_{\text{ATC}}(h, U) = \mathbb{E}_{\mathbf{x}\sim U}\big[\mathbb{I}[s(\mathbf{p}) > t]\big]$$

In other words, this method selects a threshold on the score function such that the error in the source dataset matches the fraction of data points that receive a score below t and then predict the error on the target domain as the fraction of unlabelled datapoints that obtain a score below t.

While Garg et al. (2022) focus only on vanilla accuracy as a measure to estimate the performance of a classifier h, other measures can be used, such as F_1. Computing F_1 is straightforward once we have the threshold t and the scores for the target domain U, at least for the binary case: all datapoints above the threshold are considered to be *true positives* and *true negatives*, based on their predicted class, and all datapoints below t are considered *false positives* and *false negatives*.

It must be noted, also, that Garg et al. (2022) focus on many kinds of *distribution shift*, not only prior probability shift, even though the description of what kind of shift taken into account in every experiment reported in their work in not always specified thoroughly.

3.3 Difference of Confidences (DoC)

Difference of Confidences (DoC) is a method proposed by Guillory et al. (2021) to predict model performance on previously unseen distributions without access to labelled data. The work relies on the fact that a substantial amount of information about various kinds of distribution shifts is encoded in the *difference of confidences* of the model's predictions between the training distribution and the previously unseen test distribution. DoC tries to estimate the accuracy of prediction on an unlabelled test set U of a model h trained over a dataset T using a regression model R trained over data computed on a validation set V for which it holds the IID assumption with respect to T and a set of sample validation datasets $\bar{V}$ expressing various kinds of dataset shift.

Given $A(h, X)$ the accuracy of predictions from h on a dataset X, the method tries to predict either the accuracy $A(h, U)$ directly or the accuracy gap

$$\Delta A(h, V, U) = A(h, U) - A(h, V)$$

since getting this last value allows to recover the other given that $A(h, V)$ is known.

The regressor used by DoC is trained on the distance between the datasets in $\bar{V}$ and the validation set V using as a target the $\Delta A(h, V, V_i)$ for $V_i \in \bar{V}$. To this end, DoC needs a measure of distance S_{V,V_i}. The measure mainly explored for DoC is *difference of confidence*, from which the method takes its name. Guillory et al. (2021) compute difference of confidence as the difference between the *average confidences* of two datasets, i.e.,

$$\text{DoC}(V, U) = F(h, V) - F(h, U)$$

where $F(h, X)$ is the average *MaxConf* of model h on dataset X and is defined as

$$F(h, X) = \frac{1}{|X|} \sum_{\mathbf{x} \in X} \max_{i \in \mathcal{Y}} p_i$$

where $\mathbf{p} = \{p_1, \ldots, p_n\}$ are the posterior probabilities obtained from h for a datapoint $\mathbf{x}$. In practice, the method computes the average confidence $F(h, V)$ for V and the one for each dataset $V_i \in \bar{V}$. Then, the source datapoints for the regressor are the DoCs computed between $F(h, V)$ and $F(h, V_i)$ for each $V_i \in \bar{V}$:

$$\text{DoC}(V, V_i) = F(h, V) - F(h, V_i)$$

The targets the regressor is trained against are the accuracy deltas computed between the same datasets, so, for each $V_i \in \bar{V}$ we compute:

$$\Delta A(h, V, V_i) = A(h, V_i) - A(h, V)$$

Once the regressor is trained, we can use it on an unlabelled dataset U to predict $\Delta A(h, V, U)$ by passing $\text{DoC}(V, U) = F(h, V) - F(h, U)$ as input to the model. The predicted accuracy of h on U can be retrieved as:

$$\hat{A}(h, U) = \Delta A(h, V, U) + A(h, V)$$

The work by Guillory et al. (2021) also explores a variant of this method which uses *NegEnt* as a score function instead of *MaxConf*, resulting in a method called *Difference of Entropy* (DoE). While this still being an interesting and valid alternative, we chose to pick only the first variant as a baseline.

The method can also be improved by using non-linear regression or by calibrating the model, as discussed by Guillory et al. (2021).

The main focus of this work is to use *vanilla accuracy* as an accuracy score, but also other measures can be used, such as F_1, by simply training the regressor using the different measure as a target.

3.4 Mandoline

Mandoline is a method to predict how well a deployed model will perform on a target unlabelled dataset. It is based on the original idea of *Importance Weighting* (IW) and Chen et al. (2021b) advocate how Mandoline should represent an improvement over IW.

Importance Weighting is a general approach for estimating properties of a random variable X drawn from a target distribution U given samples $\{x_i\}_{i=1}^n$ from a different source distribution T. Source and target distributions T and U have densities p_T and p_U while $\mathbb{E}_T$ and $\mathbb{E}_U$ are expectations with respect to T and U. Since $\mathbb{E}_T\left[\frac{p_U(X)}{p_T(X)}h(X)\right] = \mathbb{E}_U[h(X)]$, one can estimate $\mathbb{E}_U[h(X)]$ with the empirical average $\frac{1}{n}\sum_{i=1}^n \frac{p_U(x_i)}{p_T(x_i)}f(x_i)$. Typically, also the *density ratio* $r(X) = \frac{p_U(X)}{p_T(X)}$ is unknown and must be estimated. This is actually why the method is also known as *density ratio estimation*.

There are many approaches to estimate density ratio $r(x)$. One of the best kown is the *Kullback-Leibler Importance Estimation Procedure* (KLIEP) described by Sugiyama et al. (2007). This procedure models the density ratio $r(x)$ by the linear model:

$$\hat{r}(\mathbf{x}) = \sum_{l=1}^{b} \alpha_l \varphi_l(\mathbf{x})$$

where $\{\alpha_l\}_{l=1}^b$ are parameters to be learned from data samples and $\{\varphi_l\}_{l=1}^b$ are basis functions. As discussed above, test input density can be estimated as:

$$\hat{p}_U(\mathbf{x}) = \hat{r}(\mathbf{x})p_T(\mathbf{x})$$

since we know $p_T(\mathbf{x})$ beforehand, but we do not know $p_U(\mathbf{x})$.

Parameters $\{\alpha_l\}_{l=1}^b$ are determined so that the Kullback-Leibler divergence from $p_U(\mathbf{x})$ to $\hat{p}_U(\mathbf{x})$ is minimised. To do this, we can maximise the following term, which is independent of $p_U(\mathbf{x})$, which we do not know:

$$\sum_{j=1}^{n_t} \log \sum_{l=1}^{b} \alpha_l \varphi_l(\mathbf{x}_j^t)$$

This is a convex optimisation problem and the global solution can be obtained by simply performing gradient ascent.

IW works well when the source and target distributions overlap significantly, but performs poorly when the distributions' supports have little overlap, which is common under distribution shift. Moreover, IW works well in low dimensions but struggles with large variances of estimated density ratios in high-dimensional settings (Chen et al., 2021b).

Mandoline tries to overcome these limitations leveraging on the information the user can convey to the method: users can construct *slicing functions* to identify the axes along which the distributions may have shifted. These slices create a common representation on which the methods can project and compare source and target data, reducing dimensionality and thus mitigating the problems proper of IW. Other than that, Mandoline can be instantiated with many standard density ratio estimation methods at its core.

Chen et al. (2021b), however, present an extension of the LL-KLIEP method that can denoise the density ratio. In fact, this is needed because slices are user-provided and can be noisy, correlated and incomplete. This core is then able to represent slices in a graphical model, polish them and then use the denoised density ratio to generate importance weights for evaluating models.

The need to provide information to the method in the form of slices may pose a limitation since, in many cases, the structure of the datasets used both as source and as target is not known and can be difficult, in few dimensions, to capture important and significant properties of these distributions.

Fortunately, Chen et al. (2021b) themselves give a general idea of what can be used as a slicing function for Mandoline, which comes very close to what we've already seen in other works and also to what we will see in our methods. The proposed idea uses:

- 6 fixed slices computed using negative entropy on posteriors output by model h with $H(\mathbf{p}) = -\sum_{i\in[1..3]} p_i \log p_i$. The slices g_j for $j \in [1..6]$ are then computed as follows:

$$g_j(\mathbf{x}) = \mathbb{1}[H(\mathbf{p}) \in [0.2 \cdot (j-1),\ 0.2 \cdot j]]$$

- n slices, where $n = |\mathcal{Y}|$ for the original classification problem; these slices are computed as follows:

$$g_j(\mathbf{x}) = \mathbb{1}[\arg\max_{i\in\{1,\dots,n\}} p_i = j] \quad \text{for} \quad j \in \{1,\dots,n\}$$

Here, $\mathbf{p} = \{p_1,\dots,p_n\}$ are the posterior probabilities obtained from h for a datapoint $\mathbf{x}$.

The focus of (Chen et al., 2021b), for what concerns accuracy measures, is on vanilla accuracy. This score is computed directly by the method and appears non-trivial to extend it to use other measures, such as F_1.

3.5 Reverse Classification Accuracy

Reverse Classification Accuracy (RCA) is a method proposed by Elsahar and Gallé (2019) to estimate the performance of a classifier h. It uses h to pseudo-label

the target distribution and then train a new classifier h' on the newly obtained dataset; the accuracy of h' is measured on a validation dataset obtained as a held-out set from the training distribution and is used to estimate the accuracy of h on the target distribution.

More in detail, the classifier h is trained on a labelled distribution T and is then used to perform predictions on an unlabelled target distribution U. The labels obtained by this prediction, or *pseudo-labels*, are then used as training data for a *reverse classifier* h'; this is trained using the same architecture and the same training algorithm used to train h. The performances of both classifiers are then compared on a validation set V, obtained as a held-out set from T, and used to define the RCA score:

$$\mathrm{RCA}(h, V) = \frac{1}{|V|} \sum_{(\mathbf{x}_i, y_i) \in V} \mathbb{1}\Big[y_i = h(\mathbf{x}_i)\Big] - \mathbb{1}\Big[y_i = h'(\mathbf{x}_i)\Big]$$

Elsahar and Gallé (2019) advocate that RCA score could be low for two main reasons:

- the domain shift which the measure tries to capture, as a very different distribution could have a major impact on the training data for h' generated on U
- the accumulation of error generated by the "back-and-forth" training

To mitigate these effects they propose a variant of RCA, called RCA*, which works in a similar way to RCA; in this case the newly fit classifier h' is compared to another newly fit model h^* which is trained on the pseudo-labelled set V, with pseudo-labels obtained using the predictions of h on V. It must be noted that V used to train h^* is the same dataset used, both in RCA and in RCA*, to evaluate the two classifiers, but with different labels in the two contexts. RCA* score can then be defined as:

$$\mathrm{RCA}^*(h, V) = \frac{1}{|V|} \sum_{(\mathbf{x}_i, y_i) \in V} \mathbb{1}\Big[y_i = h^*(\mathbf{x}_i)\Big] - \mathbb{1}\Big[y_i = h'(\mathbf{x}_i)\Big]$$

The RCA and RCA* measures are finally used to train a linear regressor which in turn is able to predict the accuracy of h on an unlabelled dataset U. To do this, the method assumes the existence of a small fixed number of validation datasets $\tilde{V}$. For each of these validation sets the chosen measure is computed together with the accuracy on that dataset. These values are used to train the linear regressor, which, in turn, is then able to predict the accuracy of h on an unlabelled dataset once the proper measure, RCA or RCA*, is computed.

Elsahar and Gallé (2019) also propose other measures in their work, but we chose to focus on the two described above because they seem to be the most

interesting ones and those that most distinguish themselves from other works we describe in this chapter.

This method, while interesting, has one major downside in the computational overhead. The need to train at least one new classification model for each dataset encountered, both while training the regressor and in testing, represents a significant cost.

3.6 Other Works

In this section we briefly present some of the works of the past years related to the CAP problem that we consider of a more moderate interest with respect to those presented above.

3.6.1 Trust Score

A work that moves towards the prediction of a classifier accuracy is proposed by Jiang et al. (2018), which proposes to compute the so called *trust score*, a measure of the agreement between the classifier and a modified nearest-neighbour classifier on an unlabelled test set. This work does not try to estimate classification accuracy directly, but to measure a confidence score for the original classifier accuracy on a test set, which, in turn, can be used to assess the performance of the model on unlabelled data and if the prediction can be trusted or not.

The approach proceeds in two steps:

1. The training set is pre-processed to find the α-*high-density-set* of each class, which is defined as the training samples within that class after filtering out α-fraction of the samples with lowest density, where α is a threshold parameter.

2. Given a test dataset, its *trust score* is defined to be the ratio between the distance from the testing set to the α-high-density-set of the nearest class different from the predicted class, and the distance from the test set to the α-high-density-set of the class predicted by the original method h. The intuition, here, is that if h predicts a label that is considerably farther than the closest label, then this is a warning that the classifier may be making a mistake.

This procedure can thus be viewed as a comparison to a modified nearest-neighbour classifier. Jiang et al. (2018) remarks how many distance measures can be used in this approach, such as nearest-neighbour distance or k-nearest-neighbour distance.

Even though this method cannot be applied directly to predict the accuracy of a classifier, it poses an interesting alternative in its approach and we considered it worth of a mention.

3.6.2 Reverse Testing

Bhaskaruni et al. (2018) propose an idea for classifier accuracy prediction which is actually based on the *reverse testing framework*, an approach originally proposed by Fan and Davidson (2006).

The idea of this approach is, given a labelled training set and an unlabelled test set, to first train a model on the training set and to use it to perform predictions on the test set obtaining pseudo-labels for said test set. Then a new equivalent model is trained on the pseudo-labelled test set and used to perform predictions on the original training set. These predictions are used to obtain an estimation of the accuracy of the original model on the test set. Bhaskaruni et al. (2018) calls the prediction qualities of the second model on the training set *reversed testing qualities* and they hypothesise these qualities to be more accurate estimates of the testing qualities than traditional training qualities.

While simple, this method appears to be an interesting approach to accuracy estimation that tries to differentiate itself from classic approaches such as k-fold cross-validation.

3.6.3 Performance Predictor

Redyuk et al. (2019) propose an approach where a *performance predictor* tries to predict the classification accuracy of a black-box model h leveraging on several *shift generators* provided by the user of the framework, i.e., functions that are able to generate new datasets characterised by some kind of shift starting from a known and labelled dataset.

The aim of the method is to compute the estimation $\hat{l}_U$ of the actual accuracy score l_U that h achieves on the target test distribution U without access to ground truth labels y_U. First, the method computes a series of validation labelled sets with user-provided shift generators applied on an original labelled validation set. For each validation set V generated in this way, the model h is applied on V and the predictions for this dataset are indicated as $\hat{y}_V$. A score l_V is computed comparing the predicted and true labels for V using a loss function $\mathcal{L}$. Then, an univariate non-parametric estimate of the distributions of each output dimension of h, indicated as ϕ_V, is computed. Redyuk et al. (2019) call these values *percentiles*. These percentiles and the l_V score previously computed are stored for each validation set V and are finally used to train a regression model. The estimated accuracy score $\hat{l}_U$ for an unlabelled test set U can then be computed obtaining the value ϕ_U for this set and then using the regressor on it.

Although interesting, this method appears similar to other approaches reviewed above. Moreover, this description appears less precise, given the fact that neither the kind of accuracy score l is explicitly specified, nor it is clearly described how to obtain percentile values ϕ.

3.6.4 Self-Training Ensembles

Chen et al. (2021a) propose a framework that uses a self-training technique on ensembles of models for accuracy estimation and error detection of a pre-trained model on unlabelled test data.

The framework's approach uses a *check model* k and the disagreement between k and the pre-trained model h in order to estimate accuracy. This disagreement approach succeeds if k agrees with h on points where h is correct and disagrees with h on points where h is incorrect. To ensure that this requirement is satisfied in as many cases as possible, Chen et al. (2021a) propose to use an *ensemble* of models as a check model. A datapoint $\mathbf{x}$ is, then, identified as misclassified if a large fraction of the models in the ensemble disagree with h on $\mathbf{x}$. Since the ensemble may only be able to identify a subset of the misclassified points, Chen et al. (2021a) propose to iteratively identify more and more misclassified datapoints using *self-training*. For each misclassified datapoint $\mathbf{x}$ identified by the ensemble, a pseudo-label is assigned to it that is different from $h(\mathbf{x})$. Then, a new ensemble is trained in such a way that the disagreement of the models in the new ensemble with h on the pseudo-labelled data R is encouraged. Pseudo-labels may not all be correct, but the only requirement here is that the new ensemble mostly disagrees with h on R so that the ensemble's models still identify R as misclassified points. This process is repeated N times, where N is an hyperparameter of the framework.

This approach appears to be interesting and general enough to be considered as a viable and broad method to estimate classification accuracy.

3.6.5 AutoEval

Deng and Zheng (2021) formulate the accuracy prediction problem of a pre-trained classifier on unlabelled test sets as a dataset-level regression problem.

The proposed method involves a training set on which the original model is classified and a set of validation sets (also called *meta-dataset*), which is constructed through data synthesis via various transformation of the original training set. Since the problem was tackled under the visual classification domain, these transformations include rotation, background substitution, foreground scaling, etc. Given N validation sets, Deng and Zheng (2021) represent the j-th set $\mathcal{D}_j$ with $(\tilde{\mathcal{D}}_j, A_j)$, where $\tilde{\mathcal{D}}_j$ is the representations for $\mathcal{D}_j$ used to train the regression algorithm and A_j is the corresponding target which equals to the accuracy of the original model on $\mathcal{D}_j$. The method aims to use a regression model to be trained on the representations of the validation sets which will then be able to predict the accuracy of the original model on unseen and unlabelled test sets, for which the same representation used in training is extracted and provided to the regressor.

The regression model is indicated as:

$$A_j = R(\tilde{\mathcal{D}}_j)$$

Deng and Zheng (2021) consider the possibility of using mainly two types of regression model, i.e., linear regression and neural network regression. For each of these models a different dataset representation is used, in both cases the main proposed idea is based on using Fréchet distance (Dowson and Landau, 1982) computed on the gap between the dataset for which the representation is being computed and the original training set.

One of the main downsides of this method is the restricted domain for which the method was conceived, which implies that trying to make the method suitable for a more general use requires finding other ways to generate synthetic validation sets when they are not available in nature.

3.6.6 Generalisation Disagreement Equality (GDE)

Jiang et al. (2022) propose a method to estimate a model's accuracy by training the same architecture on the same training set twice and then measuring the disagreement rate between the two models on an unlabelled test distribution.

More specifically, the two models are trained on two training sets of the same size independently drawn from the same distribution. These two models are then used to predict over an unlabelled test set and the rate of disagreement between those two predictions is used as an estimate of the original model on the test set.

This method was originally devised for *Stochastic Gradient Descent* (SDG), so it is not clear if it is suited to a more general application in the field of classifier accuracy prediction. Nonetheless, it appears to be an interesting and simple approach that could potentially give good results.

3.6.7 Majority Voting ensemble

You et al. (2022) propose another approach based on an ensemble of models to estimate the accuracy of a pre-trained classification model on unlabelled test sets. This approach exploits the concepts of majority voting, which is achieved by training, first, an ensemble of multiple homogeneous models. You et al. (2022) outline the proposed method as follows:

1. Use the training dataset to train $M = 2N + 1$ models, where N is a sufficiently large positive number.

2. Let $b_i \leftarrow 0$ for $1 \leq i \leq N + 1$.

3. Use the trained models to predict the label of one data point in the unlabelled dataset. Suppose that m models predict the datapoint as *positive* and the rest as *negative*.

4. If $m \leq N$, then $b_{m+1} \leftarrow b_{m+1} + 1$; otherwise $b_{2N+2-m} \leftarrow b_{2N+2-m} + 1$.

5. Repeat Steps 3 and 4 for all the datapoints in the dataset.

6. Compute $R(k)$ over the cumulated bin number k as

$$R(k) = \frac{\sum_{i=1}^{k} b_i}{|U|}$$

   where $|U|$ is the number of datapoints in the test distribution and k is in the range of 1 to $(N + 1)$.

7. Use $R(k)$ to estimate the prediction accuracy of the test dataset with a linear model.

In other words, this approach puts each datapoint in a bin indexed with the number corresponding to the number of discordant models in the ensemble. $R(k)$ is then the fraction of datapoints for which the number of discordant models was less or equal to k.

The linear model suggested by You et al. (2022) is a linear regressor to be trained on a set of labelled validation datasets and featuring a linear equation like the following:

$$\hat{A} = a_0 R(k) + b_0$$

This method, although interesting, shows some downsides. First, it needs to train and use a relatively large number of models to build the ensemble, which can result in poor performance. Moreover, the need to choose a proper value for k and N to correctly estimate the original classifier accuracy can pose a challenge, also given the fact that the value of N itself can impact on performance, since the number of models in the ensemble is proportional to N.

3.6.8 SHAP-Eval

Silva and Veloso (2022) propose SHAP-Eval, a method for classifier accuracy estimation where SHAP values, i.e., local feature importance information, are used to provide patterns of correct and incorrect predictions on the training and test sets, which in turn can be used to estimate the efficacy of a model h on a particular test set.

SHAP values (Lundberg and Lee, 2017) represent a way to interpret predictions of a model. The SHAP framework assigns each feature an importance value for a particular prediction in a weighted fashion by learning how the model behaves in the vicinity of a datapoint.

The SHAP-Eval method explores SHAP values to discover correct and incorrect classification patterns. First, SHAP-Eval trains a model h on the training

set T. Then, an explainer G is used to generate SHAP values of h on T and the test set U, i.e., $\mathcal{X}_T^G$ and $\mathcal{X}_U^G$. Next, a new dataset $\mathcal{X}_{\text{tci}}$ is created concatenating the values from $\mathcal{X}_T^G$ and from $\mathbf{y}_n$, which is a vector of randomly generated labels. Also $\mathbf{y}_{\text{tci}}$ is created, checking, for each datapoint, if $y_n^i = y_T^i$ and assigning 1 to y_{tci}^i if they match, 0 otherwise. Finally, a model h_{ci} is trained on $(\mathcal{X}_{\text{tci}}, \mathbf{y}_{\text{tci}})$. To estimate the efficacy of h on U also a test set $\mathcal{X}_{\text{teci}}$ is created based on SHAP values $\mathcal{X}_U^G$ and predictions $h(U)$, which are concatenated together. The prediction of h_{ci} on $\mathcal{X}_{\text{teci}}$ is then used to estimate the accuracy of h on U. Since SHAP-Eval involves randomness, the process of generating SHAP values is repeated v times and the final estimated accuracy is the mean of all estimations.

Chapter 4

A Quantification-Based Approach for Predicting Classifier Accuracy on Data Characterised by Prior Probability Shift

Suppose we have a classifier $h : \mathcal{X} \rightarrow \mathcal{Y}$ trained on a set T of labelled items $(\mathbf{x}, y)$, where data items in $\mathcal{X}$ are represented as vectors $\mathbf{x} = (x_1, ..., x_m)$ and labels y range on $\mathcal{Y} = (y_1, ..., y_n)$. We will outline, in this chapter, the characteristics of our method, QuAcc, describing in detail what machine learning tools it leverages and how they are used to tackle the CAP problem.

4.1 Single Multiclass Quantifier (1×4)

We focused our investigation and our experiments mainly on the binary instance of the problem, even though our method can easily be extended to the multiclass case. In the general multiclass case, our method for estimating $A(h, U)$ is based on the following idea:

1. View the contingency table $C = \{c_{11}, ..., c_{nn}\}$ of classifier h as a codeframe, i.e., consider each cell $c_{ij} \in C$ as a class.

2. Consider the values c_{ij}^U of the cells that would result by applying classifier h (trained on set T) to the datapoints in U and checking, for each unlabelled datapoint $(\mathbf{x}, y) \in U$, the assigned label $h(\mathbf{x})$ against the corresponding true label y. Note that the values c_{ij}^U are unknown, since the true labels of the datapoints in U are unknown.

3. Train, on a separate labelled set V, a model that estimates the values c_{ij}^U. We use a separate labelled set V since we here make the assumption (which

is likely satisfied in many real-world applications) that the set T on which the classifier has been trained is no longer available. We will also make this assumption in our experiments of Chapter 5. If T were still available, though, nothing would prevent us from using T in place of V.

4. Use the estimates $\hat{c}^U_{ij}$ to compute $\hat{A}(h, U)$.

The contingency table C we refer to in Step 1 stores, in each cell c_{ij}, the number of datapoints that actually belong to class i and were predicted to fall in class j by the classifier h. As mentioned in Step 2, these values are only known when we refer to a contingency table for a labelled set T, but are unknown when we refer to an unlabelled set U.

Step 3 can obviously be recast as training on V a model that estimates the *prevalence values* $p_U(c_{ij})$ of classes c_{ij} in U. This is a crucial aspect to our method, since we assess that its strength comes from the fact that we use, for tackling this step, techniques from quantification, since this task is indeed concerned with training estimators of the class prevalence values and can perform particularly well under prior probability shift (Esuli et al., 2023). The quantifier of Step 3 is trained on a separate labelled set V since here we make the assumption (which is likely satisfied in many real-world applications) that the set T on which the classifier has been trained is no longer available. The only assumption we make on the relation between T and V is that they are IID. Our method is supposed to run well also lifting this assumption, even though we have not explored this scenario in our experiments.

Step 4 can also be recast to produce estimates in the form of *prevalence values* such as $\hat{p}_U(c_{ij})$. The way the estimated accuracy measure $\hat{A}(h, U)$ can be obtained from the prevalence values will not be explored further in this chapter as it will be the main focus of Section 5.3. We will mainly deal with how to obtain prevalence estimations.

In order to carry out Steps 3 and 4, we represent the datapoints as pairs $(\ddot{\mathbf{x}}, \ddot{y})$. Here $\ddot{\mathbf{x}}$ is a vector

$$\ddot{\mathbf{x}} = (\mathbf{x}, \Pr(y_1|\mathbf{x}), ..., \Pr(y_m|\mathbf{x}))$$

which incorporates (i) the original representation $\mathbf{x}$ that classifier h has used, and (ii) the posterior probabilities that h has returned for $\mathbf{x}$. In other words, we train a quantifier q by providing it with all the information we have available about $\mathbf{x}$ and that q might need to figure out which cell c^U_{ij} datapoint $\mathbf{x}$ is likely to belong to. Of course, the quantifier is not interested in individual datapoints *per se*, but is interested in them only insofar as they contribute to the distribution of U across the classes. If $\ddot{\mathbf{x}}$ is the representation of a datapoint (either in V or in U) the posterior probabilities and the prediction are those returned by the trained classifier h.

In our pairs $(\ddot{\mathbf{x}}, \ddot{y})$, $\ddot{y}$ is instead a label that ranges not on $\mathcal{Y}$ but on $\ddot{\mathcal{Y}} \equiv C$. When $\ddot{y}$ is the label of a datapoint in V, this indicates that $\ddot{y}$ is its true label.

When $\ddot{y}$ is the label of a datapoint in U, instead, $\ddot{y}$ is unknown. While it is not our goal to guess $\ddot{y}$ for this datapoint in particular, it is our goal to estimate, for each $y_i, y_j \in \mathcal{Y}$, the *prevalence* $p_U(c_{ij}) \equiv c_{ij}^U/|U|$ of datapoints $(\mathbf{x}, y_i) \in U$ such that $h(\mathbf{x}) = y_j$. It is for reaching this goal that we use a quantifier. Once we have estimated the prevalence in U of all class pairs $(y_i, y_j) \in \mathcal{Y} \times \mathcal{Y}$, by multiplying all these estimates by $|U|$ we obtain estimates of all counts c_{ij}^U, and we thus can compute the estimate of $A(h, U)$.

An important aspect of this method is that it works for any classifier accuracy measure A defined in terms of a contingency table, since it does not estimate measure A directly but estimates the values of the cells of the contingency table C on which measure A is based. Another important aspect of this method is that it is learner-independent, since the learner used to train the classifier whose accuracy needs to be estimated plays no role in the method. Our method is also quantifier-independent, i.e., it does not make any hypothesis on which method has been used to train quantifier q.

In the binary case (to which we restrict our analysis in this thesis) in which $\mathcal{Y} = \{\oplus, \ominus\}$, the contingency table is $C = \{\text{TP}, \text{TN}, \text{FP}, \text{FN}\}$, and we need to train one multiclass quantifier that operates on these four classes; we thus call this the *1×4 method*.

4.2 Multiple Binary Quantifiers (2×2)

A viable alternative of the 1×4 method can be obtained by observing that, once we have applied classifier h to U, we already know the value of $|\text{TP} \cup \text{FP}|$ (resp., the value of $|\text{TN} \cup \text{FN}|$), since this is the number of datapoints in U which h has assigned to class $\oplus$ (resp., to class $\ominus$). This means that we can leverage this information and solve our prediction problem by training, instead of one 4-class quantifier, two binary quantifiers, i.e., one, $q_\oplus$ that estimates how the datapoints in $\{\mathbf{x}|h(\mathbf{x}) = \oplus, \mathbf{x} \in U\}$ are distributed across TP and FP, and one, $q_\ominus$,that estimates how the datapoints in $\{\mathbf{x}|h(\mathbf{x}) = \ominus, \mathbf{x} \in U\}$ are distributed across FN and TN. Note that the prevalence values predicted by $q_\oplus$ and by $q_\ominus$ will both sum up to 1, i.e.,

$$\hat{p}_U^\oplus(\text{TP}) + \hat{p}_U^\oplus(\text{FP}) = 1$$
$$\hat{p}_U^\ominus(\text{TN}) + \hat{p}_U^\ominus(\text{FN}) = 1$$

This means that, to retrieve the estimated contingency table, we have to multiply, in this case, each estimated prevalence for the size of the corresponding subset for which it was computed, respectively $|U^\oplus|$ and $|U^\ominus|$. Since this method involves two binary quantifiers, we call it the *2×2 method*.

The rationale behind this method is that, by having two separate quantification models focusing on two separate and distinct subsets, we are able to simplify

the task, down to binary quantification problems in the case of two original classification classes, and this can potentially provide more accurate results.

In the multiclass case, switching from the analogue of the 1×4 method to the analogue of the 2×2 method means switching from a single quantifier that operates on n^2 classes to n quantifiers that operate on n classes each.

4.3 Single Multiclass Quantifier with Collapsed Classes (1×3)

A further variant of the 1×4 method can be obtained by observing that class FP is likely to be largely composed by negative (resp., positive) examples that, while lying in the region that the classifier has assigned to class $\oplus$ (resp., class $\ominus$), lie just across the separating surface. In other words, most of the false positives and most of the false negatives are likely to lie in two regions of space that both flank (from two different sides) the separating surface, i.e., are likely to lie in two *contiguous* regions of space. It may thus make sense to merge FP and FN into a single class FP ∪ FN. This solution would thus involve a single three-class quantifier that needs to estimate how the datapoints are distributed across classes

$$\ddot{\mathcal{Y}} = \{\text{TP}, \text{TN}, \text{FP} \cup \text{FN}\}$$

and not across classes defined by C. Since this method involves a quantifier working on three classes (in the binary case) we call it the *1×3 method*.

In this case, the contingency table $\hat{C}_U$ cannot be fully reconstructed from the prevalence values predicted by q. This poses a potential disadvantage for this solution with respect to the two previous ones, in that it does *not* work for all classifier accuracy measures; in particular, it does not work for measures A such that |FP| and |FN| contribute differently to A (one example are cost-sensitive accuracy measures, in which the cost of a false positive may be stipulated to be different from the cost of a false negative). However, this solution does work for important classifier accuracy measures such as, e.g., vanilla accuracy and F_1, whose mathematical forms (see Equations 2.1 and 2.2) are such that knowing the value of |FP ∪ FN| suffices, and knowing the individual values of |FP| and |FN| is not required.

This method can be potentially extended to the multiclass case by collapsing false classes in various ways. This poses the same problems as in the binary case and possibly more, since any variation of this method in the multiclass case could make the computation of F_1 unfeasible. Moreover, collapsing classes in multiclass could pose some theoretical issues, that, however, we did not explore thoroughly.

4.4 Variants and Extensions

When we extend the covariates of a datapoint $\mathbf{x}$ with posterior probabilities obtained from h prediction to build extended datapoint $\ddot{\mathbf{x}}$, we are giving q additional pieces of information to determine how well h classified datapoints in V and to predict its performance on any unlabelled dataset U. We can choose to give additional information to q to better estimate h performance.

The ideal kind of information we would like to include in $\ddot{\mathbf{x}}$ would be relative to the ability of h to predict, hence a measure of h's confidence in its prediction.

We devise three interesting functions able to detect the confidence score of h on each datapoint, plus an additional one based on the core idea proposed by Garg et al. (2022) in their method. These confidence measures can be added all together or in part. The best selection of covariates to add to $\ddot{\mathbf{x}}$ can vary and can be chosen case by case, e.g., employing proper methods of model selection, as we will discuss in Section 5.6.

4.4.1 MaxConf

Given $\mathbf{x}$ the feature vector for a datapoint and $\mathbf{p} = (p_1, ..., p_n)$ its posterior probabilities obtained from h, the *MaxConf* covariate (previously used by Garg et al. (2022) as a score) is defined as:

$$\mathrm{MC}(\mathbf{p}) = \max_{i \in \{1,...,n\}} p_i$$

The value computed by *MaxConf* ranges from a maximum of 1 to a minimum of $1/|\mathcal{Y}|$ and gives a measure of how confident is h of its prediction of the class label of $\mathbf{x}$: the higher the value the higher the confidence in the prediction. The confidence level is thus decoupled from the actual class y_i it is associated with. This gives q an additional information on how much it can trust the prediction obtained from h and included in $\ddot{\mathbf{x}}$. It must be noted that this information is already present in $\ddot{\mathbf{x}}$. The additional covariate reinforces this information making it potentially easier for the quantifier to exploit it.

4.4.2 Negative Entropy

In general information theory, entropy of a random variable is a measure of the level of uncertainty proper of the variable possible outcomes. In machine learning *negative entropy* (*NegEnt*) is usually used as a measure to indicate the level of confidence of a prediction emitted by a model h. This is due to the fact that the opposite of a measure, *entropy*, which can measure the level of uncertainty, is viable as a confidence measure.

This measure is usually computed on the posterior probabilities $\mathbf{p}$ emitted by h. We use *NegEnt* as an additional covariate by drawing inspiration from (Garg

et al., 2022) (who use it as a score in their computation) and we compute it as:

$$\mathrm{NE}(\mathbf{p}) = \sum_{i=1}^{n} p_i \log p_i$$

NE, just like MC reaches its maximum value for cases in which $p_i = 1$ for some i, and attain its minimum value for the uniform distribution. However, NE contributes a different type of information than MC: while the latter only accounts for the maximum value in a distribution, NE processes all values. This gives a quantifier q a potentially more interesting information about the confidence of h.

4.4.3 Inverse Softmax

The third type of additional covariate that we explore is based on the well known *softmax* function $s : \mathbb{R}^n \to (0,1)^n$, also known as *normalised exponential function*. It transforms any real-values vector $\mathbf{z} = (z_1, \ldots, z_n)$ into a probability distribution $s(\mathbf{z}) \equiv \mathbf{p} = (p_1, \ldots, p_n)$, where each distribution value is obtained as:

$$s_i(\mathbf{z}) = p_i = \frac{e^{z_i}}{\sum_{j=1}^{n} e_j} \tag{4.1}$$

In other words, it applies the exponential function to each element of $\mathbf{z}$ and normalises these values by dividing them by the sum of all these exponentials. The normalisation ensures that the sum of the components of $\mathbf{p}$ is equal to 1, making it a probability distribution.

The canonical base used in this function is e. In general, for any base $b > 1$, larger input values will result in lager output probabilities and for values of $\mathbf{z}$ greater than 1 the *softmax* function tends to amplify the maximum due to the use of the exponential function.

If we try to apply *softmax* function to a vector of probabilities, e.g., a vector of posteriors as in our case, we tend to obtain the opposite effect as we are applying the exponentials on an array of values which are already normalised and ranging in $[0, 1]$, so we tend to flatten values instead of amplifying them.

To amplify the values of a probability distribution we can *invert* the *softmax* function, thus obtainining the *inverse softmax* function $s^{-1}(\mathbf{p}) = (z'_1, \ldots, z'_n)$. To invert *softmax* we can apply log function to both sides of Equation 4.1 and get:

$$\log p_i = z'_i - \log\Big(\sum_{i=1}^{n} e^{z_i}\Big)$$

Rearranging the function we get:

$$z'_i = \log p_i + \log\Big(\sum_{i=1}^{n} e^{z_i}\Big)$$

Factor $c = \log\left(\sum_{i=1}^{n} e^{z_i}\right)$ is a constant for all components of $\mathbf{p}$. Since we do not know *a priori* the actual values z_i, we cannot know of c. We can, though, choose c in such a way that the sum $\sum_{j=1}^{n} z'_j = 0$ and that all values z'_i are centred around 0:

$$c = -\frac{1}{n}\sum_{j=1}^{n} \log p_j$$

The resulting equation is then:

$$z'_i = \log p_i - \frac{1}{n}\sum_{j=1}^{n} \log p_j \tag{4.2}$$

The rationale behind inverse softmax is that of amplifying, in a non-linear way, the difference between low-confidence and high-confidence values.

As with MC, we focus on the maximum value. The resulting *max inverse softmax* (MIS) covariate is then given by:

$$\text{MIS} = \max_{i\in\{1,\dots,n\}} \left(\log p_i - \frac{1}{n}\sum_{j=1}^{n} \log p_j \right)$$

Using MIS as a covariate can potentially result in an improvement to other additional covariates such as MC, since the amplified values can add important information for the quantifier q.

4.4.4 Threshold

Taking direct inspiration from the work of (Garg et al., 2022), we devised an additional covariate called *threshold* (TS). Since the threshold value t proposed by (Garg et al., 2022) and reported in Equation 3.1 is supposed to separate well-classified datapoints from misclassified ones, we can use this approach to craft a covariate. The separation of datapoints is determined by the value of a score computed for each datapoint with respect to t: if the score is higher than t the datapoint is estimated as well-classified, otherwise it is estimated as misclassified. The score, here, is either *MaxConf* or *NegEnt* (Garg et al., 2022). TS can, then, be computed on a distribution of posterior probabilities as:

$$\text{TS}(\mathbf{p}) = t_s - s(\mathbf{p})$$

where s is the chosen score measure for the evaluation.

Although interesting, we have tested this covariate in various settings exploiting well suited validation sets, but it did not show any improvement with respect to the results obtained employing other covariates. For this reason, we decided to not use it further in our experiments and it will not be featured in our results.

Chapter 5

Experiments

In this section we describe the experiments we have carried out and discuss the results we have obtained.

5.1 Evaluation Protocols

A dataset for testing quantification systems is usually the result of the application of an *extraction protocol* (Esuli et al., 2023, §3.4) to a dataset $\Omega = (\mathcal{T}, \mathcal{V}, \mathcal{U})$ otherwise used to test *classification* systems, where:

- $\mathcal{T}$ is a labelled training pool from which we extract training samples to train the classifier for which we want to estimate accuracy;
- $\mathcal{V}$ is a labelled validation pool from which we extract validation samples to train quantification algorithms;
- $\mathcal{U}$ is an unlabelled[1] test pool from which we extract test samples to be used to evaluate methods performance.

To extract training and validation samples we first resample training and validation pools $\mathcal{T}$ and $\mathcal{V}$ with random sampling (without replacement) in such a way that prevalence values for positive and negative classes are equal ($p_{\mathcal{T}}(\oplus) = p_{\mathcal{T}}(\ominus)$ and $p_{\mathcal{V}}(\oplus) = p_{\mathcal{V}}(\ominus)$). This inevitably reduces the original size of the two pools, but allows us to apply the extraction protocol starting always from the same conditions, despite the original characteristics of the given dataset.

After this, we randomly extract (with replacement) from $\mathcal{T}$ a training sample T_i for each value of $p(\oplus)$ that lies on the 9-point grid $\{0.1, \ldots, 0.9\}$. These 9 samples are equally sized, and their size is

$$|T_i| = \frac{10}{9} \cdot \min\{p(\oplus), p(\ominus)\} \cdot |\mathcal{T}|$$

[1] All test sets at our disposal are actually labelled, but in out experiments we treat them as if we did not have ground truth labels in order for them to simulate real-world scenarios.

which is the largest possible number that can result in 9 equally-sized samples. Similarly, we randomly extract (with replacement) from $\mathcal{V}$ a validation sample V_i for each value of $p(\oplus)$ that lies on $\{0.1, \ldots, 0.9\}$. In all our experiments we keep $p_{T_i}(\oplus) = p_{V_i}(\oplus)$. Using training samples T_i characterised by different $p_{T_i}(\oplus)$ values is meant to test our methods under different conditions of class imbalance, while stipulating that $p_{T_i}(\oplus) = p_{V_i}(\oplus)$ is meant to represent the fact that T_i and V_i are sampled from the same distribution.

For test samples extraction, we here use the widely adopted *artificial prevalence protocol* (APP – (Esuli et al., 2023, §3.4.2)), originally proposed by Forman (2005), which, in the binary case, consists of randomly extracting from $\mathcal{U}$ a number of test samples $U_1, \ldots, U_k$ characterised by values of $p(\oplus)$ that lie on a predefined grid of values. In this case, we use a 21-point grid: $\{0.00, 0.05, \ldots, 0.95, 1.00\}$. In our experiments, for each value of the grid we randomly extract (with replacement) 100 test samples consisting of 1000 datapoints each, for a total of k=2100 test samples. The implementation of APP in our experiments was realised using the QuaPy open-source Python library (Moreo et al., 2021).

Each result we report is thus the average value of $E(A(h, U), \hat{A}(h, U))$ (see Equation 5.1) across some of the axes of all combinations of 9 values of $p_{T_i}(\oplus)$ (and $p_{V_i}(\oplus)$), 21 values of $p_{U_j}(\oplus)$, and 100 samples for each value of $p_{U_j}(\oplus)$. Final results, that we shown in Table 5.2, will interest the average across all possible combinations, i.e., across 9×21×100=18,900 combinations. Therefore, the result of using this protocol is that of "stress-testing" the CAP methods, i.e., testing their ability to correctly estimate classifier accuracy in a wide range of situations, characterised by different values of training class imbalance, test class imbalance, and PPS .

Concerning this latter, note that this experimental protocol clearly simulates PPS, since the distribution of the covariates X conditional on the distribution of the labels Y is the same for the T_i's and the U_j's, but the distribution of the labels Y is not the same for the T_i's and the U_j's.

5.2 Datasets

The data we used in our experiments were mainly taken from two datasets:

- The well-known IMDB movie reviews dataset (IMDB – (Maas et al., 2011)). IMDB is a binary dataset in which labelled data are balanced with 50% of positives and in which both training and test sets include 25000 documents. The dataset was imported through the QuaPy library (Moreo et al., 2021) and imposing a minimum *document frequency* of 3 per term.

- The well-known Reuters Corpus Volume I (RCV1), an archive of manually categorised newswire stories available by Reuters, Ltd. for research purposes (Lewis et al., 2004). RCV1 is comprised of 804414 samples of which

	$\|\mathcal{T}\|$	$\|\mathcal{U}\|$	$p_{\mathcal{T}}(\oplus)$	$\|T_i\|$
IMDB	12,500	25,000	0.500	6,944
CCAT	11,575	781,265	0.474	5,992
GCAT	11,575	781,265	0.297	3,872
MCAT	11,575	781,265	0.255	3,267

Table 5.1: Main features of the datasets used in this thesis; $|\mathcal{T}|$ is the size of the training pool from which the training samples T_i are extracted (which is equal to $|\mathcal{V}|$, the size of the validation pool), $|\mathcal{U}|$ is the size of the test pool, from which the test samples U_j are extracted, $p_{\mathcal{T}}(\oplus)$ is the prevalence value of $\oplus$ in the training pool (which is the same as the prevalence value $p_{\mathcal{V}}(\oplus)$ of $\oplus$ in the validation pool), while $|T_i|$ is the size of the samples used for *classifier* training (which is the same as the size $|V_i|$ of the samples used for *quantifier* training).

the first 23149 correspond to the training set, while the others are to be considered part of the test set. Datapoints have 47236 features, while labels have 103 categories, making the dataset multiclass in its original form. Since we focused our experimentation mainly on the binary case, we have used RCV1 as a binary dataset selecting one category at a time. The categories we mainly used were CCAT, GCAT and MCAT. In fact, these resulted to be the categories more balanced both in training and test sets. We will consider, hereafter, CCAT, GCAT and MCAT as standalone datasets, rather than categories selected from RCV1, and we will treat them as such. These datasets were imported directly through "scikit-learn" Python library (Pedregosa et al., 2011) and adapted to our codebase.

Table 5.1 shows more detailed characteristics about datasets presented above. More specifically, we can see how dataset IMDB is completely balanced in the training (and validation) set, with exactly half datapoints in class $\oplus$ and half in $\ominus$. Other datasets do not show the same balance in training (and validation) set. Dataset sizes reported in column $|T_i|$ are relative to the size of training (and validation) samples obtained via the extraction protocol described in Section 5.1.

5.3 Evaluation Measures for Classifier Accuracy Prediction

The goal of our method when targeting the CAP problem is to output a contingency table $\hat{C}_U$ that estimates the ground-truth contingency table C_U proper of the predictions of a classifier h with respect to the true (and unknown) labels of an unlabelled set U. To evaluate the performance of the prediction given by our method, instead of comparing $\hat{C}$ and C directly, we compute an accuracy value $A(h, U)$ of classifier h on the unlabelled set U. The two accuracy measures used to compute $A(h, U)$ in our experiments are vanilla accuracy and F_1, described in more details in the following two sections.

As the measure of the error we incur into when predicting the accuracy $A(h, U)$, we use the absolute difference between the true accuracy value and its estimate, i.e.,

$$E(A(h,U), \hat{A}(h,U)) = |A(h,U) - \hat{A}(h,U)| \tag{5.1}$$

5.3.1 Vanilla Accuracy

The formulation we saw in Equation 2.1 for *vanilla accuracy* is only available when we have at hand predictions and true labels of each single datapoint from a dataset. When our method outputs a contingency table we only have prevalence values of classes defined considering C as a codeframe, namely (in the binary case) p_{TP}, p_{TN}, p_{FP} and p_{FN}. To compute vanilla accuracy from these values, we can observe that the prevalence values p_{TP} (resp. p_{TN}) represent the fraction of datapoints that were correctly predicted to belong to $\oplus$ (resp. $\ominus$) by h. This means that we can compute vanilla accuracy by summing the prevalence values of classes that refer to correctly classified datapoints, i.e.,

$$\text{Acc}(h,U) = p_{\text{TP}} + p_{\text{TN}}$$

5.3.2 F_1

Since F_1 is obtained from a contingency table, the computation of this measure from the output of our method is straightforward. The only difference stands in the fact that, while in Equation 2.2 the values indicate the amounts of datapoints belonging to a certain class, in our output they indicate the prevalence values of those classes. This, though, is not relevant for the purpose of the computation of F_1, which can still be computed similarly as in Equation 2.2, i.e.,

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

5.4 Baselines

We compared the results to our work to a variety of baselines taken from previous works. The approaches we decided to adopt as baselines are:

- ATC (Garg et al., 2022) (see Section 3.2). The original work did not tackle classifier accuracy prediction using F_1, but the method itself allows the use of this measure, so we decided to include it alongside with vanilla accuracy as a baseline. Moreover, we decided to compare our work with only one version of the two provided by Garg et al. (2022), i.e., the one based on the use of MaxConf. This was done because, by preliminary tests on some validation sets, we noticed how the two alternative approaches gave very similar results.

- DoC (Guillory et al., 2021) (see Section 3.3). Also in this case the approach allowed the use of F_1 measure even though the original work did not take it into account so we decided to include the measure in our baselines for DoC. Moreover, also in this case Guillory et al. (2021) provided more variants for the same approach and for similar reasons as for ATC we decided to include the alternative that appeared to be the main one. Although Guillory et al. (2021) took into account the possibility of using many regression methods inside their method, we chose, for sake of simplicity, to use DoC as a baseline based on a linear regressor.

- Mandoline (Chen et al., 2021b) (see Section 3.4). As discussed above, it is not easy to extend Mandoline to measures other than vanilla accuracy, for this reason we limited our use of this approach as a baseline to this measure. Moreover, Mandoline allows many possible configurations, and the results can vary considerably depending on the slicing functions used. The slicing we decided to use as baseline is the one proposed by Chen et al. (2021b) and discussed in Section 3.4 since it seems reasonable and in line with our work and other methods we consider.

- RCA and RCA* (Elsahar and Gallé, 2019) (see Section 3.5). The approach proposed by Elsahar and Gallé (2019) uses RCA and RCA* scores to train a linear regressor. Other scores were proposed in the same work, but we considered these two the most interesting and innovative and used only them as baselines to our work. Also in this case it is possible to extend the capabilities of the method to use F_1 as an error measure and is what we have done in using RCA and RCA* as baselines.

- Naïve. In this case the baseline is a simple approach which estimates the prediction error as the Absolute Error between the accuracy obtained on the test samples and the one obtained on a validation set.

For various reasons, we did not take into account other previous works. For some methods (Deng and Zheng, 2021; Redyuk et al., 2019; Silva and Veloso, 2022) the code was not available. For other methods (Chen et al., 2021a; You et al., 2022), while the code was available, we could not import it in such a way that we could use it in our codebase as a baseline. We could not use some other works (Jiang et al., 2018) directly as baselines since the accuracy prediction was not the direct scope of the works. We, also, previously tested some of the remaining works (Bhaskaruni et al., 2018; Jiang et al., 2022) on separate validation sets and we acknowledged how they were not high-performing enough to be used as effective baselines.

5.5 QuAcc Instances and Optimisations

QuAcc methods we used in our experiments employ one of SLD (see Section 2.4.1) and KDEy (see Section 2.4.2) quantification algorithms, with one of logistic regression and C-support vector classification (SVC) with radial basis function (RBF) kernel as underlying classification algorithm.

We obtained optimised QuAcc methods via model selection. More specifically, we executed a *grid search* to explore a set of available hyperparameters, that depend on the quantification and underlying classification algorithms on which the method is instantiated. This search aims to minimise the loss function $E(A(h,U),\hat{A}(h,U))$ on a validation sample. In particular, model selection:

- chooses the best regularisation factor C in the range $\{10^i\}_{i=-3}^{i=3}$ for all underlying classification algorithms;
- decides whether to rebalance, for both underlying classification algorithms, the instances in order to counter the possible class imbalance of the training sample;
- chooses the best γ hyperparameter for SVC algorithm with RBF kernel in the range $s \times \{10^i\}_{i=-2}^{i=2}$, where s is a scale factor defined as $s = \frac{1}{|V|\cdot\mathrm{Var}(V)}$ for a validation sample V;
- decides whether to recalibrate via *bias corrected temperature scaling* (BCTS) the classifier underlying SLD quantification algorithm;
- chooses the best bandwidth for KDEy in the range $(0.01, 0.02, \ldots, 0.2)$
- checks whether the additional covariates of Section 4.4 are worth including.

This model selection is able to generate optimised versions of the three methods described in Sections 4.1, 4.2 and 4.3. We also applied a further model selection that decides which of these three variants should be used for each training (and validation) sample, i.e., in each scenario characterised by a given training (and validation) prevalence.

5.6 Results

We conducted several experiments that interested multiple directions of investigation. We can summarise the explored experimentation paths as follows:

- We compared the Single Multiclass Quantifier (1×4) method with all the selected baselines.
- We compared all base versions of our methods (1×4, 2×2 and 1×3) among themselves.

- We tested variants of our methods featuring many combinations of additional covariates among themselves.
- We tested optimised versions of our methods obtained through *grid search* on quantifier and classifier hyperparameters against the base versions.
- We tested two different quantification algorithms against each other.
- We tested quantification algorithms with two different classification algorithms as their core.
- We tested our methods on both datapoints with original covariates enriched with posterior probabilities and possible additional features and datapoints with only posteriors and additional features.
- We applied model selection to our optimised methods and tested the resulting methods against the best baselines.
- We tested our methods on different accuracy prediction measures, showing different plot and table representations for each of our tests and repeated our experimentation over all the four datasets presented in Section 5.2.

In the following sections we will discuss in depth about all the tests we conducted, showing relevant results for each direction of investigation. The results we present will be mainly relative to the IMDB dataset and vanilla accuracy as a prediction error measure, reporting data mainly with the amount of PPS on the x-axis. This will be true for all the following sections, except for Section 5.6.9, where we will show final results, also varying on these aspects.

5.6.1 Single Multiclass Quantifier against Baselines

The Single Multiclass Quantifier (1×4) was the first formulation of our solution to the classifier accuracy estimation problem. For this reason, we conducted our first tests comparing this method against all the baselines. In Figure 5.1 we show a plot representing 1×4 method based on SLD algorithm which is using a logistic regressor as its core (`(1x4)_SLD_LR` in the plot) compared with all the baselines on vanilla accuracy (Acc).

We obtained this result averaging results given by experiments conducted on multiple training sets with various class prevalence values. Data are shown by amount of *prior probability shift*, i.e., values represented at a given point on the x-axis are the average of all the results for a given method whose absolute difference between the training and test prevalence values result to that x-axis value. In practice, this means that the more a value is encountered to the right in the plot the higher was the prior probability shift affecting training and test distributions when we obtained that specific result. The prior probability shift measure is

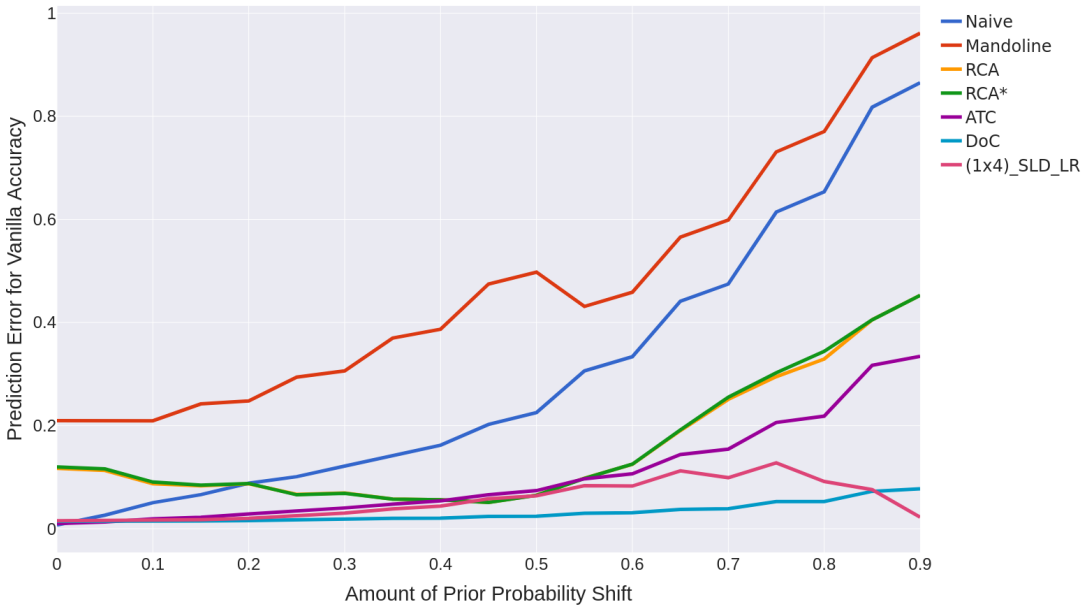

Figure 5.1: Results of tests conducted comparing 1×4 method based on SLD algorithm and logistic regression (`(1x4)_SLD_LR`) against all baselines in IMDB. The plot shows the amount of PPS on x-axis and Acc error on y-axis (lower is better). The amount of PPS is computed as the absolute difference between the prevalence value of positive datapoints measured in training and test samples. The resulting plot is the average of values obtained over 9 training prevalence values.

obtained as the absolute distance between the prevalence of the positive class in training and test sets. The y-axis shows *accuracy error* as described in previous sections.

As can be seen in Figure 5.1, our first formulation of 1×4 shows promising results in comparison with other baselines, putting already itself almost on par with those that show the best results and strictly outperforming some of them, such as Mandoline (`Mandoline` in the plot) and Naïve (`Naive`). Another interesting aspect to note is that our method shows a tendency to lower, or at least steady, accuracy errors when put in conditions of extreme shift, in contrast to other methods that tend to show always higher errors while the shift increases.

Due to the poor performance of some baselines (e.g., Mandoline and Naïve) and to the overlapping results of others (e.g., RCA and RCA*), obtained not only in this test but throughout all our experiments, we will present only a subset of these in future plots, both for sake of clarity in our presentation and to allow ourselves to show results on a finer scale for what concerns the y-axis. This can be confirmed by comparing Figure 5.1 with Figure 5.2, which shows the same results, but presents a smaller set of baselines.

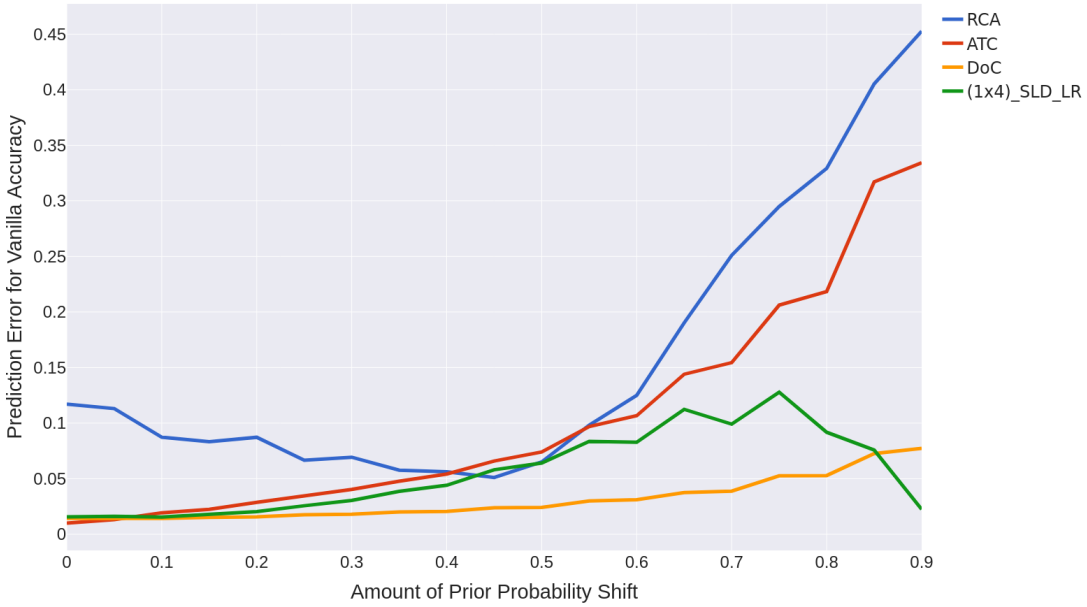

Figure 5.2: As in Figure 5.1 but showing only the baselines RCA. ATC and DoC since other baselines perform substantially worse than all the methods that we here plot.

5.6.2 Base Versions of our Method

The next focus in our research was to improve our method by finding new ways of exploiting quantification to solve the accuracy estimation problem. We devised two other methods still based on the same idea, but following a slightly different approach: Multiple Binary Quantifiers (2×2), using two different quantifiers on different portions of the same set (see Section 4.2), and Single Multiclass Quantifier with Collapsed Classes (1×3), featuring a single quantifier that predicts on a reduced number of quantification classes (see Section 4.3). We show in this section results of tests comparing SMQ with the two new methods and the best-performing baselines.

Similarly to what can be seen in the previous Section, plot in Figure 5.3 shows the accuracy error (y-axis) with relation to the variation of prior probability shift measure (x-axis). Moreover, all methods were based on a SLD quantifier relying on a logistic regressor as its core classifier.

As we can see, 2×2 and 1×3 methods show a promising improvement with respect to 1×4, giving results that narrow the distance from the best-performing baseline, DoC (Guillory et al., 2021). Even though the two methods showed an improvement over 1×4, we decided to keep testing all three methods in further scenarios, both because the improvement is not steep enough to justify the exclusion of the original method from our research, and because of the limited cases in which these three methods were compared at this point.

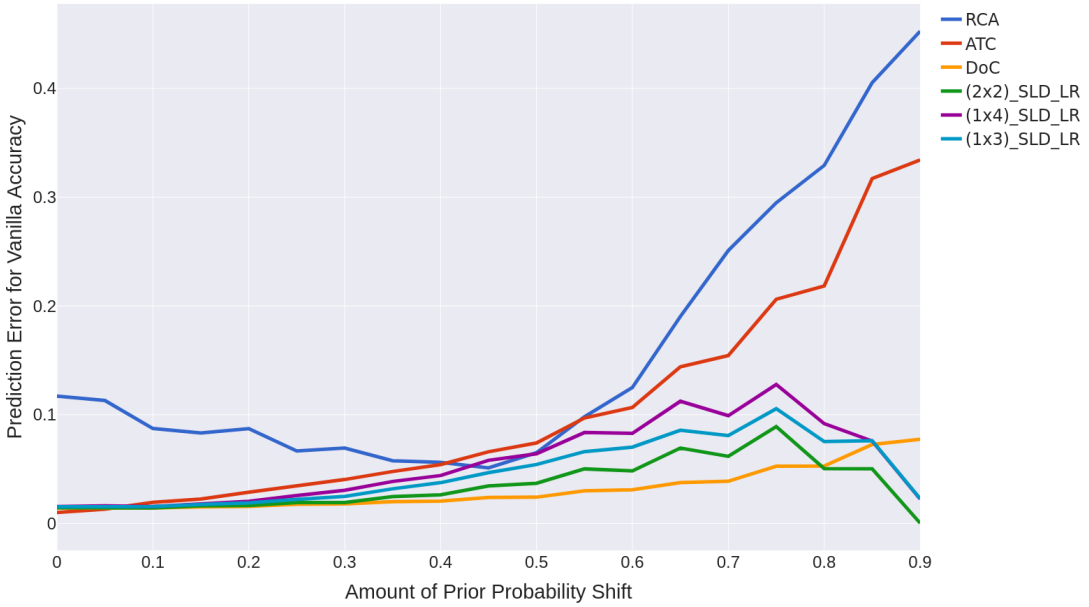

Figure 5.3: Results comparing 2×2, 1×4 and 1×3 methods (`(2x2)_SLD_LR`, `(1x4)_SLD_LR`, and `(1x3)_SLD_LR` respectively) all featuring a SLD quantifier with a logistic regressor. The methods are compared against ATC, DoC and RCA. We here do not plot other baselines since, in this experiment, they perform substantially worse than all the methods that we here plot. Results are shown for the IMDB dataset. The plot shows Acc error on the y-axis and amount of PPS on the x-axis.

As a side note, the RCA baseline (Elsahar and Gallé, 2019) keeps showing itself as the one that, overall, performs most poorly among those shown. Since we noticed this behaviour in all scenarios, we will not keep reporting its results in our results from now on, for sake of clarity.

5.6.3 Additional Covariates

As a further improvement, we included additional covariates to the datapoints on which our quantifiers are trained and tested in order to improve the quality of the information they have access to (see Section 4.4).

The plot in Figure 5.4 shows a comparison of all the three additional covariates applied one at a time to the 2×2 method, the one that showed the best performance in the previous section. Also the base version of 2×2 method and ATC and DoC baselines are shown for reference. It appears clear how the three additional covariates show a progressive improvement on the method, with the last two giving results that beat all the baselines over the whole shift spectrum.

We also tested whether various combinations of additional covariates could, when added to $\ddot{\mathbf{x}}$, improve the performance of our methods. The rationale behind

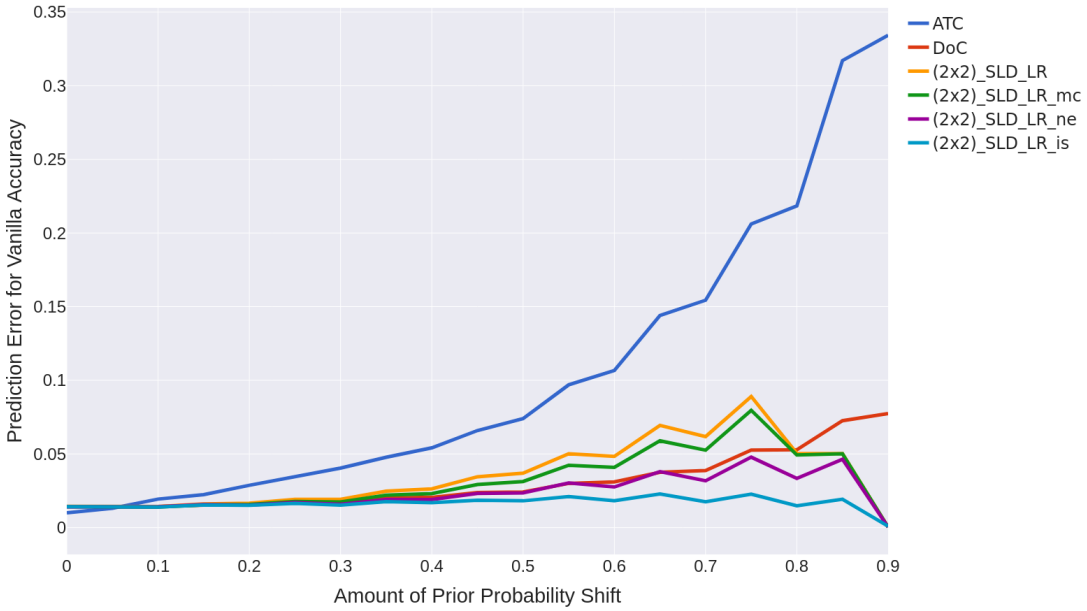

Figure 5.4: Results comparing variants of the 2×2 method enriched with *MaxConf*, *NegEnt* and *max inverse softmax* (`(2x2)_SLD_LR_mc`, `(2x2)_SLD_LR_ne`, and `(2x2)_SLD_LR_is` respectively) all featuring a SLD quantifier with a logistic regressor. The methods are compared against ATC and DoC. We here do not plot other baselines since, in this experiment, they perform substantially worse than all the methods that we here plot. Base version of the 2×2 method (`(2x2)_SLD_LR`) is also shown for reference. Results are shown for the IMDB dataset. The plot shows Acc error on the y-axis and amount of PPS on the x-axis.

this idea is that the information provided to a quantifier by multiple covariates could improve its ability to better predict classifier prediction accuracy. Results shown in Figures 5.5 and 5.6 show how this is only partially true. In fact, covariates that, when added on their own to $\ddot{\mathbf{x}}$, give better performance, tend to dominate the outcomes when joined by other less performing covariates. This is true both when we add both *MaxConf* and *NegEnt* as covariates, obtaining method `(2x2)_SLD_LR_c` in the plot of Figure 5.5, and when we add all covariates, getting method `(2x2)_SLD_LR_a` in the plot of Figure 5.6. In both cases, the method resulting from the union of covariates performs almost indistinguishably from the method featuring the best covariate by itself. To highlight this, we show `(2x2)_SLD_LR_c` along with `(2x2)_SLD_LR_mc` and `(2x2)_SLD_LR_ne` in Figure 5.5, and `(2x2)_SLD_LR_a` along with all methods featuring a single additional covariate in Figure 5.6. It appears that adding multiple covariates does not provide substantially valuable information to the quantifier in addition to those already provided by the best-performing covariate on its own. Nonetheless, we will consider these combinations in future plots and refinements of our methods presented below as they still appear as a well-performing configuration.

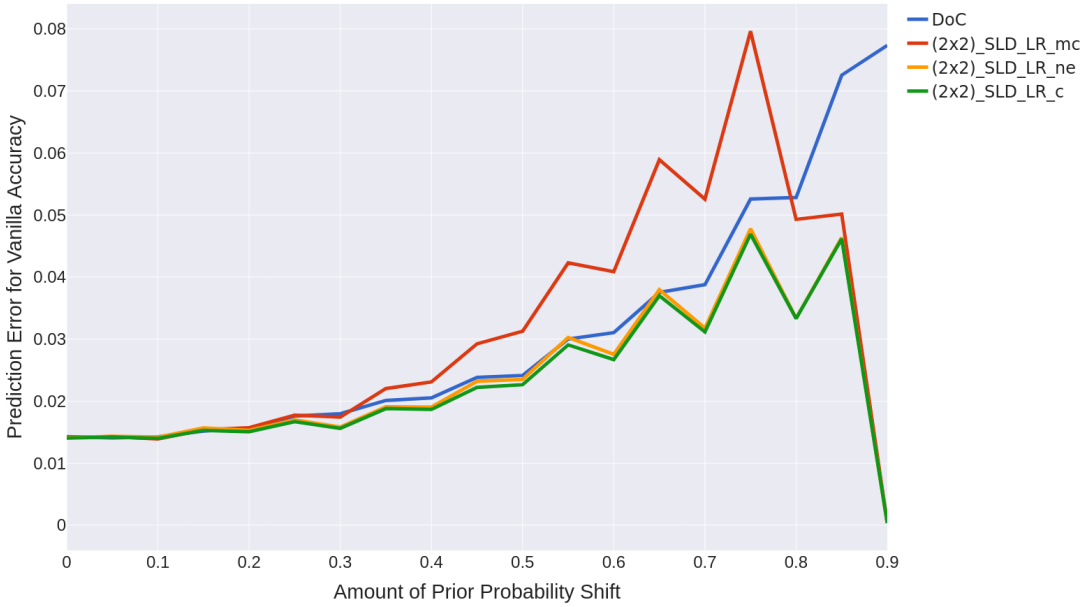

Figure 5.5: As in Figure 5.4, but comparing the variants of method 2×2 using one of covariates MC, NE and MIS (`(2x2)_SLD_LR_mc`, `(2x2)_SLD_LR_ne`, and `(2x2)_SLD_LR_is` respectively) with the variant that combines them (`(2x2)_SLD_LR_a`). Only DoC is shown for reference, in this experiment, since other baselines perform substantially worse than all the methods that we here plot.

Note that, in Figures 5.5 and 5.6, we only show DoC among all baselines as a reference to better focus the comparison on how our methods behave.

5.6.4 Optimised Versions of our Methods

In order to get better results, we performed a model selection on the hyperparameters regarding our methods, as described in Section 5.5. Note that, in this context additional covariates become hyperparameters for our methods and the model selection protocol decides, for each training prevalence, if and which covariates to add to obtain the best performance based on accuracy prediction error.

The plot in Figure 5.7 shows a comparison among the 1×3 method optimised over accuracy error (`(1x3)_SLD_LR_gs`), the same method with default hyperparameters and all additional covariates (`(1x3)_SLD_LR_a`), and the method with default hyperparameters and no additional covariates (`(1x3)_SLD_LR`). All methods shown are based on SLD and logistic regression. DoC is also shown as a reference. As we can see, while `(1x3)_SLD_LR_a` showed already a consistent improvement over the original version of the method, `(1x3)_SLD_LR_gs` pushes the performance a step further, managing to stay on par or below the showed

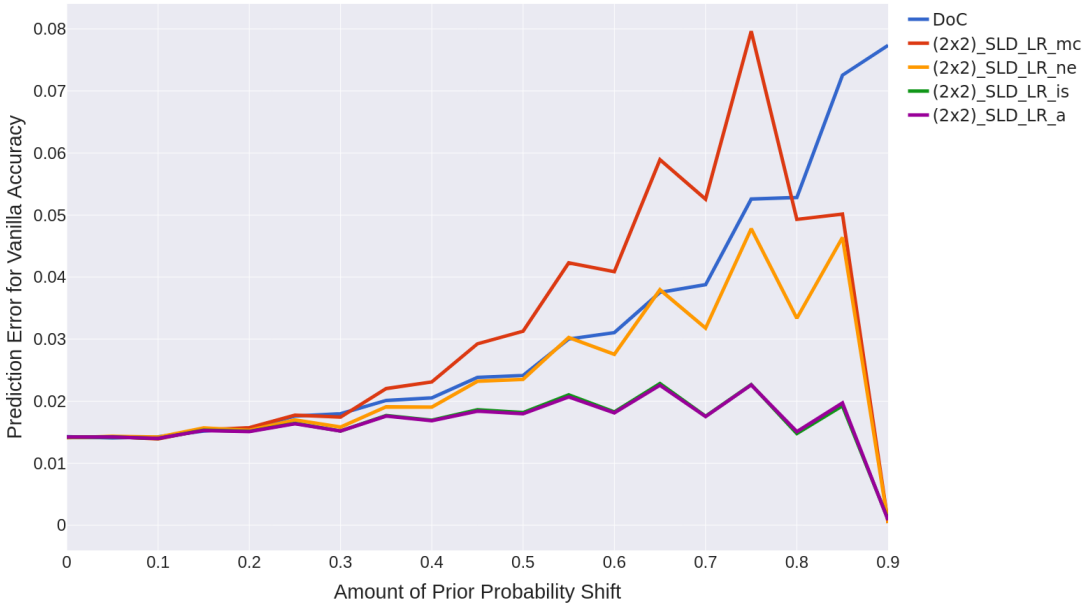

Figure 5.6: As in Figure 5.4, but comparing the variants of method 2×2 using one of covariates MC and NE (`(2x2)_SLD_LR_mc` and `(2x2)_SLD_LR_ne` respectively) with the variant that combines them (`(2x2)_SLD_LR_c`). Only DoC is shown for reference, in this experiment, since other baselines perform substantially worse than all the methods that we here plot.

baseline consistently (here DoC is the best performing baseline).

Figure 5.8 shows all optimised versions of our methods using SLD and logistic regression. They are compared against DoC and ATC, the baselines that get the best results in this setting. We can observe how our methods all beat consistently the best baselines. Other than that, they also show a trend to get low and steady values of prediction error for high values of PPS, while other methods tend to lose performance in these circumstances. This denotes a desirable resilience to harsh scenarios in which training and test prevalence values differ considerably, a real-world case that can present itself more often than not.

5.6.5 Varying the Quantification Algorithm

Given the flexibility our methods offer in terms of instantiation with underlying quantification algorithms, we tested the same configuration seen in Figure 5.8, but using KDEy quantification algorithm (see Section 2.4.2) for our methods.

In Figure 5.9 we show our optimised methods instantiated with KDEy and logistic regression (`(2x2)_KDEy_LR_gs`, `(1x3)_KDEy_LR_gs` and `(1x4)_KDEy_LR_gs`). We also show `(1x3)_SLD_LR_gs` (best of our methods in Figure 5.8) and DoC and ATC baselines as a reference.

First we note how all our optimised methods, instantiated with KDEy, per-

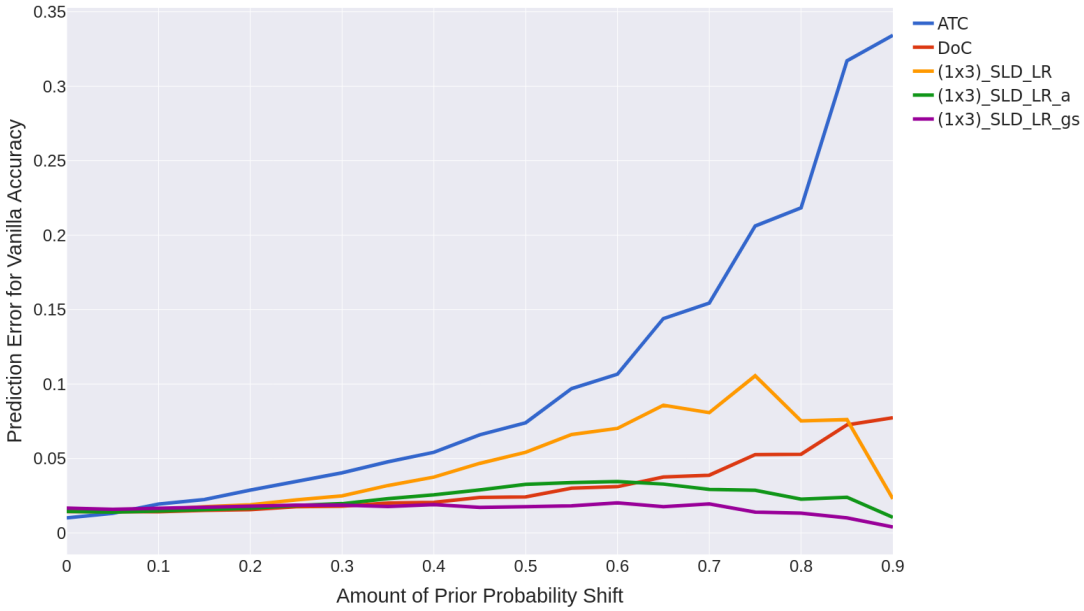

Figure 5.7: Results comparing the optimised version of 1×3 method (`(1x3)_SLD_LR_gs`) with its base version (`(1x3)_SLD_LR`) and the variant featuring all additional covariates at the same time (`(1x3)_SLD_LR_a`). All our methods use SLD and logistic regression as quantification and classification algorithms. ATC and DoC are also shown for comparison. We here do not plot other baselines since, in this experiment, they perform substantially worse than all the methods that we here plot. The prediction error measure considered is vanilla accuracy. These results are relative to the IMDB dataset.

form really well against both the two baselines and the best method, in this setting, instantiated with SLD. This is not necessarily true for KDEy vs. SLD in all scenarios, but they tend to show similar results, with KDEy outperforming SLD in some cases. This is justified by the fact that KDEy is a quantification algorithm tailored to work best under PPS in the multiclass case, and, in our methods in many cases we need to solve a multiclass quantification problem to predict accuracy.

In Figure 5.9, note that we have no data for the `(2x2)_KDEy_LR_gs` method when the amount of PPS exceeds 0.8. This is due to the fact that some instantiation of our methods might fail to train a quantifier on most unbalanced training samples in some specific datasets. This failure is caused by the fact that some of the classes considered by the quantifier might remain empty (with no datapoints) due to the lack of examples falling in one of the cells of the contingency table C (defined in Section 4.1). This represents surely a limitation of our method, even though we will describe later how we can try to address this issue with a further model selection on optimised methods.

Figure 5.10 shows a comparison among optimised versions of our methods

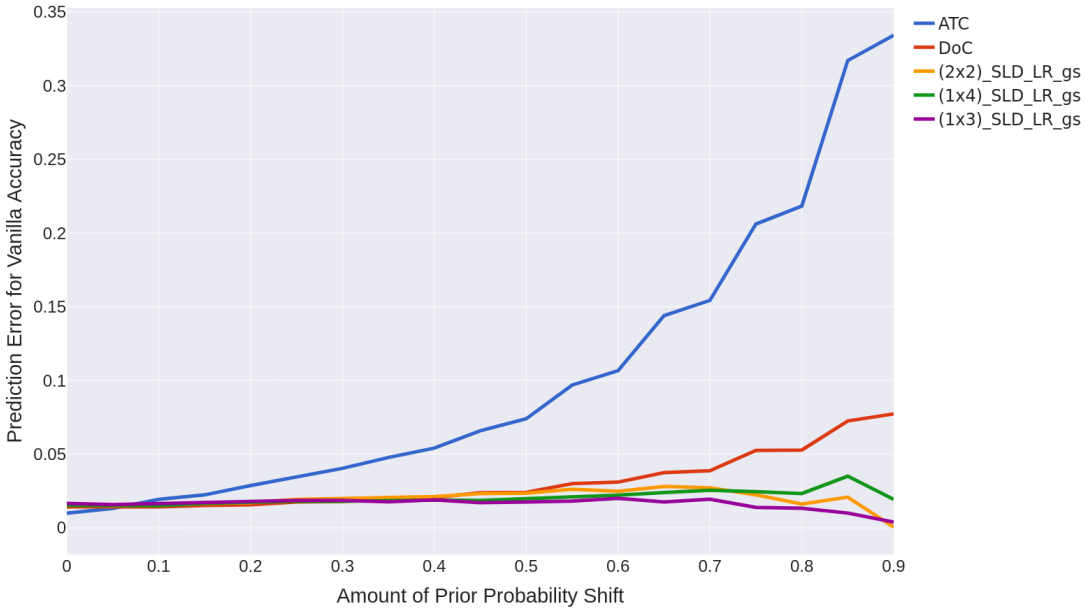

Figure 5.8: Results comparing the optimised version of all our methods (`(2x2)_SLD_LR_gs`, `(1x4)_SLD_LR_gs` and `(1x3)_SLD_LR_gs`) with the two best baselines, ATC and DoC. We here do not plot other baselines since, in this experiment, they perform substantially worse than all the methods that we here plot. All our methods use SLD and logistic regression as quantification and classification algorithms. The prediction error measure considered is vanilla accuracy. These results are relative to the IMDB dataset.

1×4 and 1×3 instantiated with both KDEy and SLD in the same setting of Figure 5.9. DoC is also shown for reference. Here we can better see, with a finer grained scale on y-axis, how KDEy methods tend to outperform some SLD methods in this scenario (`(1x4)_SLD_LR_gs`) while with others they remain on par (`(1x3)_SLD_LR_gs`).

5.6.6 Varying the Classification Algorithm Underlying the Quantification Method

Other than the quantification algorithm, another flexible aspect of our methods is the classification algorithm used as core for our quantifiers. We executed all experiments shown so far on models using a logistic regressor as quantification core. A suitable alternative is to use a different kind of classifier, e.g., a *Support Vector Machine* (SVM).

In this section we explore the results obtained using a *C-support vector classification* (SVC) algorithm, using a *radial basis function* (RBF) kernel, as a quantification core.

Note that SVM methods generally perform best when applied to datapoints

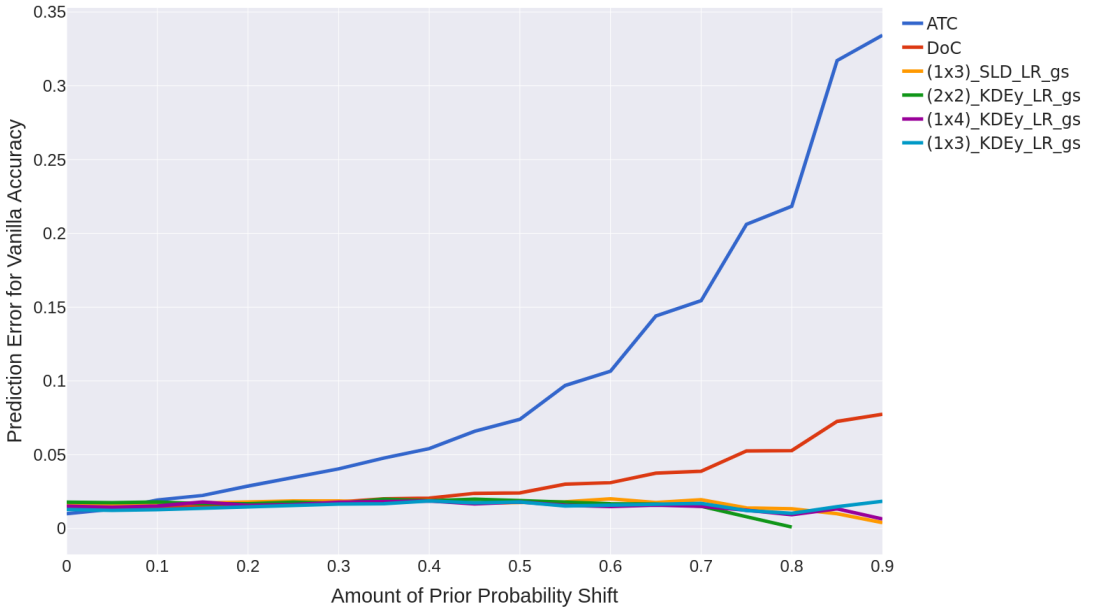

Figure 5.9: Results comparing the optimised version of all our methods instantiated with KDEy (`(2x2)_KDEy_LR_gs`, `(1x4)_KDEy_LR_gs` and `(1x3)_KDEy_LR_gs`) with the optimised version of 1×3 method instantiated with SLD (`(1x3)_SLD_LR_gs`) and the two best baselines, ATC and DoC. We here do not plot other baselines since, in this experiment, they perform substantially worse than all the methods that we here plot. All our methods use logistic regression as a classification algorithm. The prediction error measure considered is vanilla accuracy. These results are relative to the IMDB dataset.

characterised by dense covariates, while the datapoints on which this classifier is supposed to work on, in our case composed like $\ddot{\mathbf{x}} = (\mathbf{x}, \hat{p}_{\oplus}, \hat{p}_{\ominus}, F_1, \ldots, F_k)$, are sparse in the first part ($\mathbf{x}$) and dense in the second. For this reason, when applying quantification algorithms featuring SVM models, we strip out the original sparse covariates $\mathbf{x}$ from $\ddot{\mathbf{x}}$ in order to have a methods that performs in its ideal conditions.

Figure 5.11 shows the results of optimised versions of our methods instantiated with SLD using this classifier (methods ending with "`RBF_gs`" in the plot) and compares them with our optimised method featuring SLD and a logistic regressor that performs best in this setting. Also DoC is shown for reference, as it is the best-performing baseline.

The results show how, in this specific setting, RBF methods do not add anything in terms of performance with respect to LR ones, as they never manage to beat the best LR method. While this is surely correlated to the domain in which these experiments were conducted (dataset IMDB belongs to the *sentiment analysis* domain), also the absence of the original covariates might play a significant role in the obtained results, showing how original covariates $\mathbf{x}$ carry

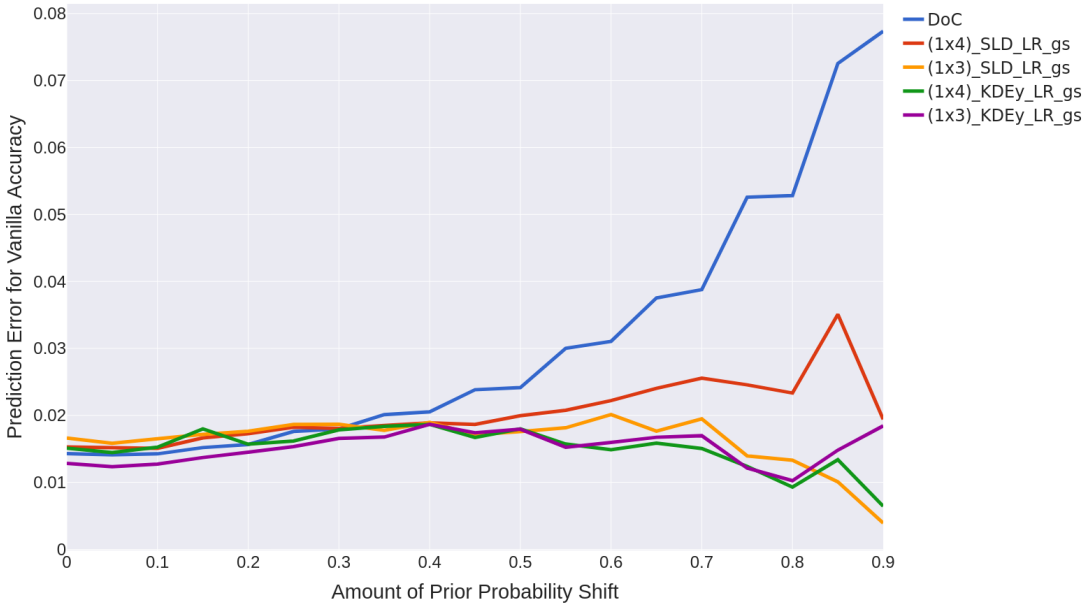

Figure 5.10: Results comparing the optimised version of methods 1×4 and 1×3 instantiated with both SLD (`(1x4)_SLD_LR_gs` and `(1x3)_SLD_LR_gs`) and KDEy (`(1x4)_KDEy_LR_gs` and `(1x3)_KDEy_LR_gs`) quantification algorithms. The DoC baseline is also shown for reference. We here do not plot other baselines since, in this experiment, they perform substantially worse than all the methods that we here plot. All our methods use logistic regression as a classifier training algorithm. The prediction error measure considered is vanilla accuracy. These results are relative to dataset IMDB.

non-redundant information for our quantifiers.

5.6.7 Removing the Original Covariates

To elaborate on the statements of Section 5.6.6 about the influence of removing sparse covariates in our methods, we conducted some tests comparing optimised versions of our methods against the same methods applied on datapoints $\ddot{\mathbf{x}}$ from which the original covariates $\mathbf{x}$ (the sparse component) were removed.

The results shown in Figure 5.12 confirm how the original covariates $\mathbf{x}$ carry valuable information that cannot be easily removed when trying to predict accuracy. We can see, in fact, how our optimised methods working on only dense covariates (those starting with "`d`" in the plot) tend to generally perform worse than other methods that work with full covariates (here `(1x3)_SLD_LR_gs` is shown for reference together with DoC). This appears to be true, not only for this experimental setting, but was evident also in other scenarios.

We can conclude that our methods involving all covariates are the most promising and we will focus on them in further analysis.

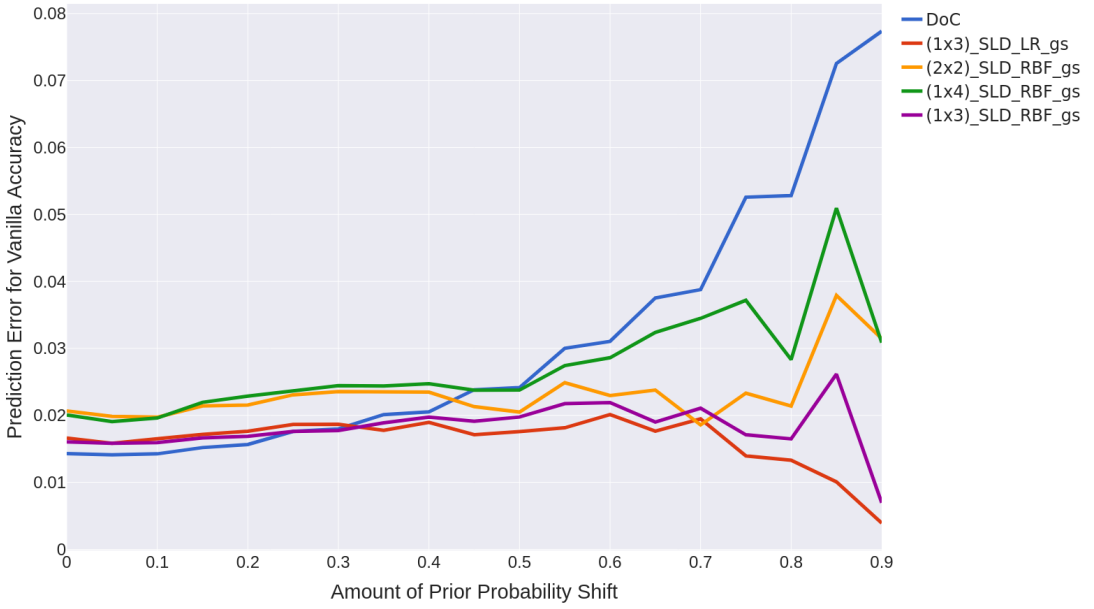

Figure 5.11: Results comparing the optimised version of all our methods instantiated with SVC classification algorithm with RBF kernel (`(2x2)_SLD_RBF_gs`, `(1x4)_SLD_RBF_gs` and `(1x3)_SLD_RBF_gs`) with the optimised version of 1×3 method instantiated with logistic regression (`(1x3)_SLD_LR_gs`) and the best baseline DoC. We here do not plot other baselines since, in this experiment, they perform substantially worse than all the methods that we here plot. All our methods use SLD as a quantification algorithm. The prediction error measure considered is vanilla accuracy. These results are relative to the IMDB dataset.

5.6.8 Model Selection on Optimised Methods

Our optimised methods perform well against all baselines in all datasets they were tested in. There is not, however, a method among the three that outperforms the others clearly, as some methods perform best in some situations while others perform best in others. For this reason we devised a model selection approach that selects, for each training and validation prevalence, the best-performing method among the three, and uses the selected method in that particular setting.

The results shown in Figures 5.13 and 5.14 present the effects of these model selections applied over our methods using SLD and KDEy quantification algorithms (respectively `SLD_LR_gs` and `KDEy_LR_gs` in the plot). As we can see, the selected method tends to perform, in both cases, as the best method among all, giving a unique resulting method that matches our expectations.

It must be noted that, in cases in which one of our methods fails, the method obtained through model selection chooses among the available ones, as it can be seen in Figure 5.14. This partially overcomes the problem described in Section 5.6.5, since we only need one successful method among 1×4, 1×3 and 2×2 for each training prevalence to get one single method that is able to produce valid

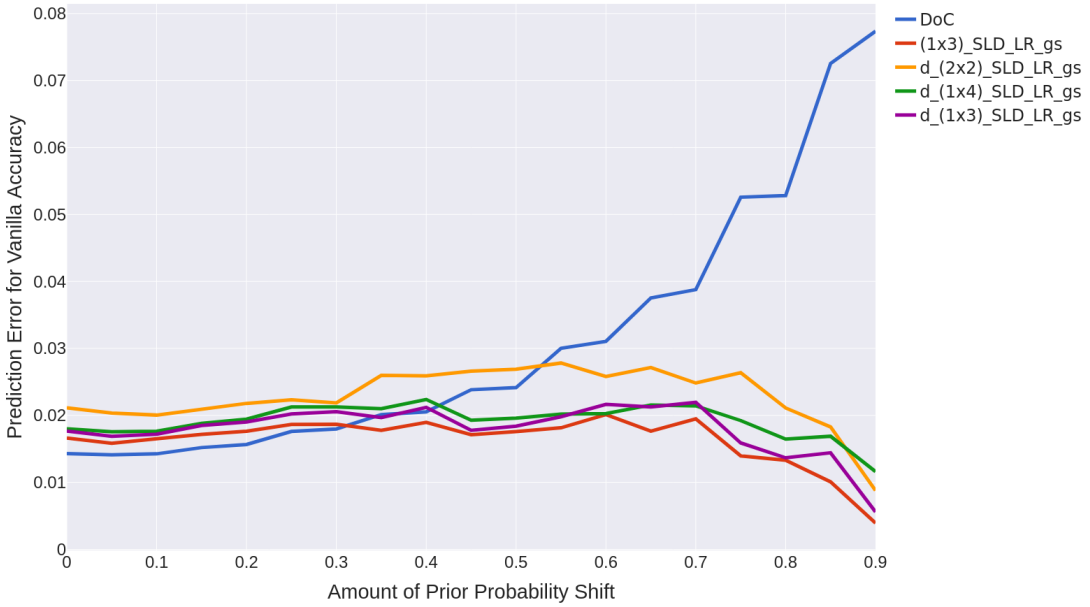

Figure 5.12: Results comparing the optimised version of all our methods working only on dense covariates (`d_(2x2)_SLD_LR_gs`, `d_(1x4)_SLD_LR_gs` and `d_(1x3)_SLD_LR_gs`) with the optimised version of 1×3 method working on all covariates (`(1x3)_SLD_LR_gs`) and the best baseline DoC. We here do not plot other baselines since, in this experiment, they perform substantially worse than all the methods that we here plot. All our methods use SLD and logistic regression as quantification and classification algorithms. The prediction error measure considered is vanilla accuracy. These results are relative to the IMDB dataset.

results across all prevalence values.

5.6.9 Final Results

The results presented so far always shared the same characteristics for what concerns

1. the dataset chosen (IMDB);
2. the CAP error measure (vanilla accuracy);
3. the kind of data visualisation (amount of PPS on the x-axis).

We have also conducted experiments on different datasets and using different error measures, and all data were visualised using multiple representations. We will present in this section a representative set of results for all these variants.

Figures 5.15, 5.16, 5.17 and 5.18 all show the best optimised version of our methods instantiated with SLD and KDEy (both using a logistic regression core)

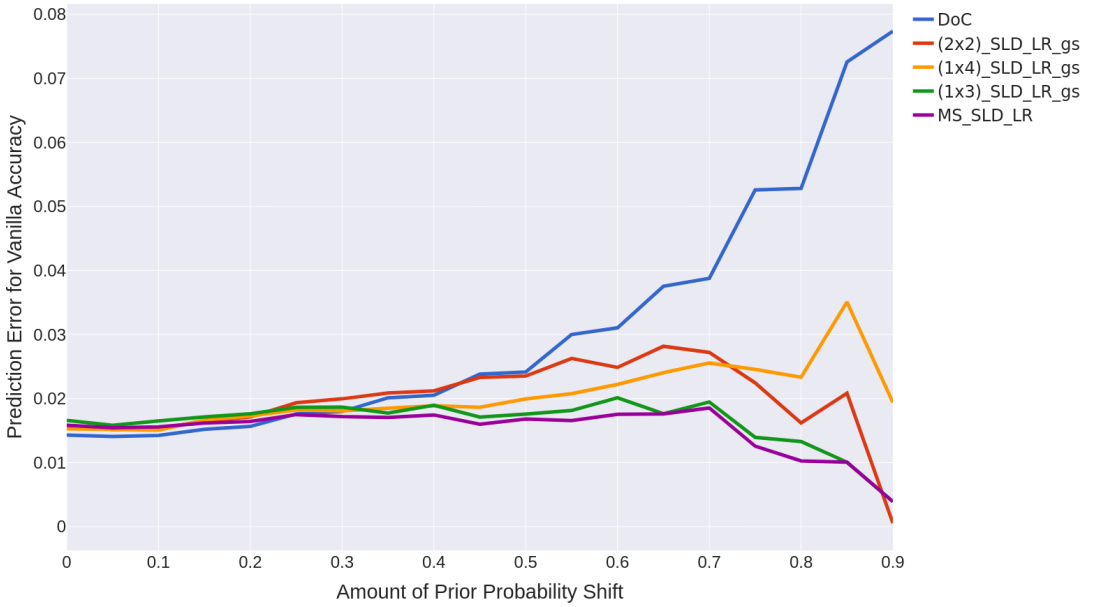

Figure 5.13: Results comparing the optimised version of all our methods (`(2x2)_SLD_LR_gs`, `(1x4)_SLD_LR_gs` and `(1x3)_SLD_LR_gs`) with the version obtained through model selection applied to all of them (`MS_SLD_LR`). Also DoC baselines is shown for reference. We here do not plot other baselines since, in this experiment, they perform substantially worse than all the methods that we here plot. All our methods use SLD and logistic regression as quantification and classification algorithms. The prediction error measure considered is vanilla accuracy. These results are relative to dataset IMDB.

compared with the two best baselines ATC and DoC on all the datasets, respectively IMDB, CCAT, GCAT and MCAT. These plots show how our methods consistently beat the baselines for mid and high amounts of PPS, while providing similar values of prediction error on low values of PPS. Note that, in real-world settings, mid and high values of shift are more likely than low ones. We can also see how, in this collection of datasets, QuAcc instantiated with KDEy tends to perform better than QuAcc instantiated with SLD, except for CCAT where results are mixed, while still being good.

We used so widely the data visualisation representing the amount of PPS on the x-axis, since we consider it the most interesting one, and the one that conveys the higher amount of information in our problem setting, given our specific focus on this kind of shift. However, it is also interesting to see the results of our experiments under a different perspective. Figure 5.19 shows the same data represented in Figure 5.15, but using a different representation. Here, we show on the x-axis the prevalence values $(p_{\ominus}, p_{\oplus})$ of the test samples on which each method was evaluated. Each point in the plot is the average of results obtained over test samples with the corresponding class prevalence and over all training

Figure 5.14: As in Figure 5.13, but instantiating our methods with KDEy quantification algorithm.

prevalence values. In our setting, there are 100 test samples per prevalence value for every training sample, i.e., each point in the plot is the average of $9\times100=900$ results. We can see here how, despite changing the way data are represented, the results given by of our methods still outperform all baselines, giving great average results on the whole spectrum of test prevalence values.

Vanilla accuracy was not the only measure we used to measure prediction error. Figures 5.20 and 5.21 show the same results presented respectively by Figures 5.15 and 5.19, but using F_1 as accuracy prediction error measure. Also in this case we can see how QuAcc performs better that the baselines, despite showing a different outline in the plot given by the different nature of the measure. Overall, under F_1 our methods show even a higher increase of performance with respect to the baselines. It must be said that none of the previous methods to which we compare were originally devised to be measured using F_1. Nonetheless, given that their extension to use this measure is actually straightforward, we have decided to compare with them also in this context.

In order to give a more complete view of how our method performs across all datasets, we show in Table 5.2 the results values for the best optimisation of our methods instantiated with SLD and KDEy with a logistic regression core (indicated with QuAcc(SLD) and QuAcc(KDEy), respectively) compared with all the baselines. We also show the results of QuAcc(CC), i.e., the method QuAcc instantiated with the trivial quantification algorithm CC. This method is shown

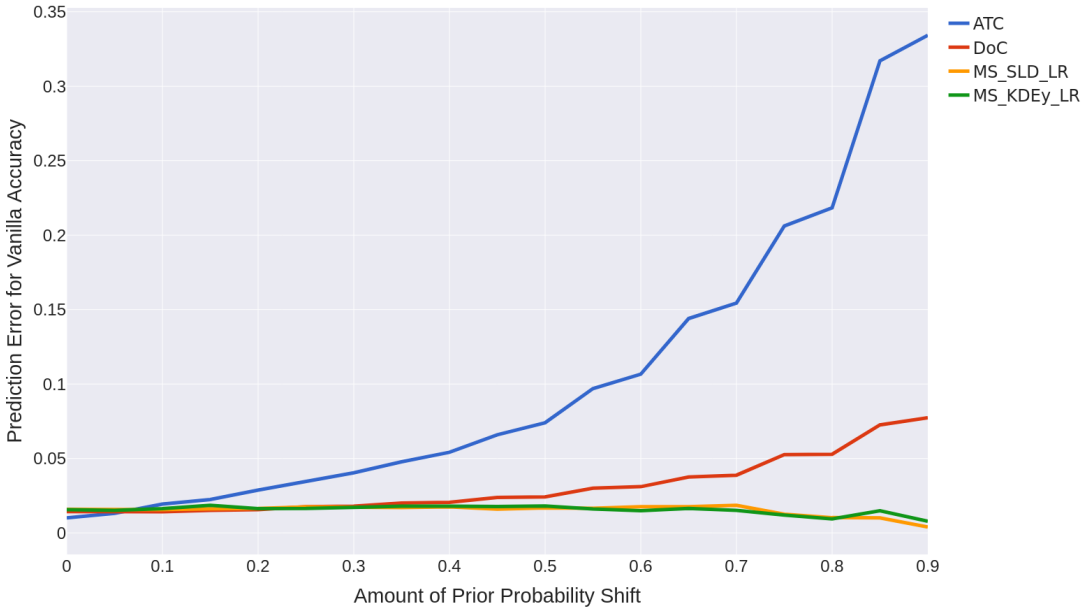

Figure 5.15: Results comparing the optimised and model selected versions of our methods, instantiated with both SLD (`MS_SLD_LR`) and KDEy (`MS_KDEy_LR`) quantification algorithms, with the two best baselines, ATC and DoC. We here do not plot other baselines since, in this experiment, they perform substantially worse than all the methods that we here plot. All our methods use logistic regression as a classification algorithm. The prediction error measure considered is vanilla accuracy. These results are relative to the IMDB dataset.

as a reference for the other two methods in order to give an idea of how much the choice of the right quantification algorithm can make a difference. All results are shown for all datasets averaging the values obtained for each method over all $9\times21\times100$=18,900 test samples. Table 5.2a shows the results using vanilla accuracy as a prediction accuracy error measure, while Table 5.2b shows the results using F_1. It is evident how our methods show a significant performance improvements with respect to all baselines over all datasets and over both measures. This is made explicit by the values reported in the last rows of both Tables 5.2a and 5.2b. We can see how the improvement given by our best method with respect to the best baseline ranges from 5.52% to 31.66% for vanilla accuracy and from 67.99% to 75.93% for F_1, which confirms how for F_1 we obtain even better results, as anticipated before.

In Table 5.3 we show data describing how our best optimised methods are selected in model selection when instantiated with SLD and KDEy. We can see that additional covariates are selected at least 62.97% of the times for SLD and at least 71.03% of the times for KDEy, meaning that they represent a valuable addition to the base versions of QuAcc. Also recalibration (for SLD) and class balancing appear to be important hyperparameters that, when properly tuned,

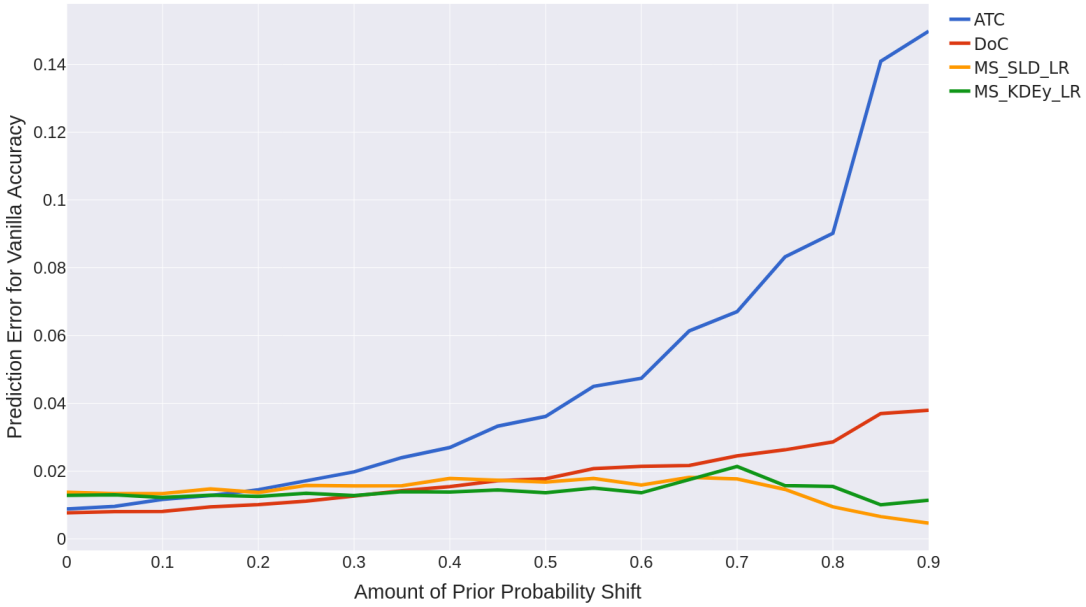

Figure 5.16: The same results reported in Figure 5.15 but for dataset CCAT.

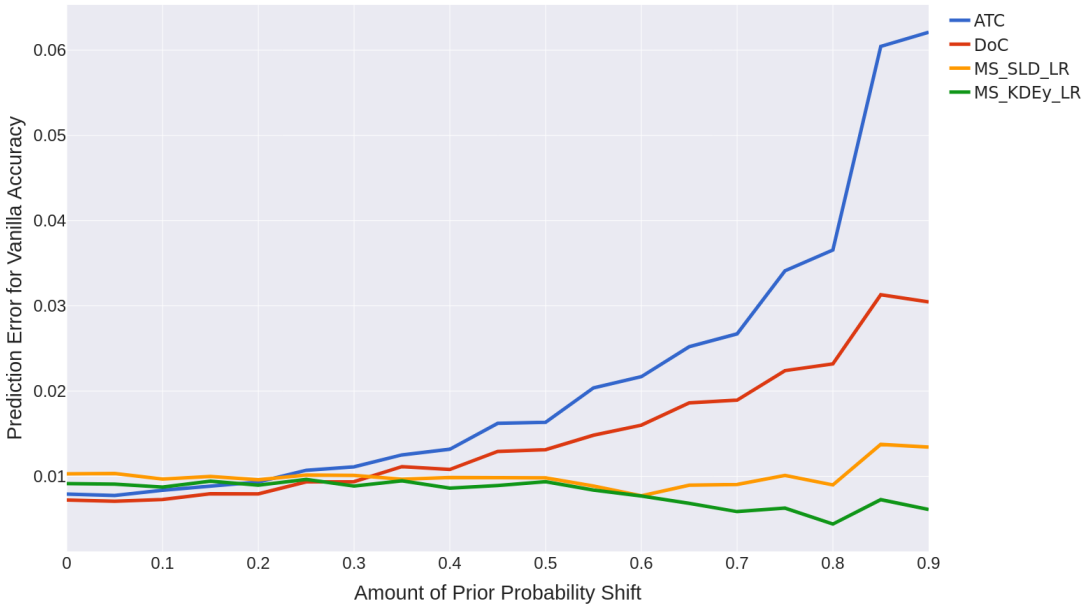

Figure 5.17: The same results reported in Figure 5.15 but for dataset GCAT.

improve the performance of QuAcc. We also show how many times the three methods are selected while generating the final one. Here, there is no clear

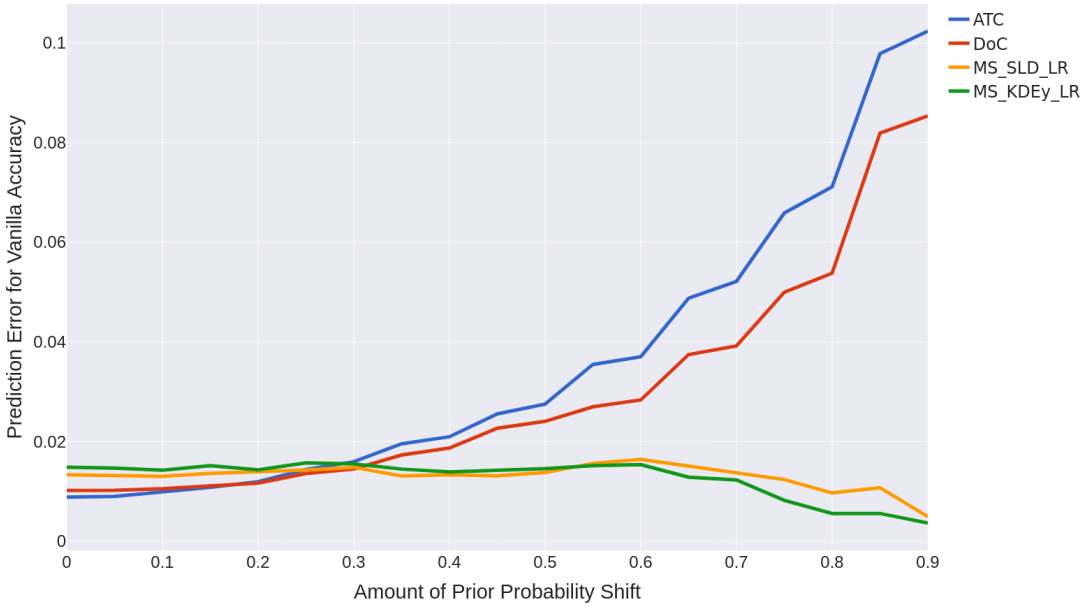

Figure 5.18: The same results reported in Figure 5.15 but for dataset MCAT.

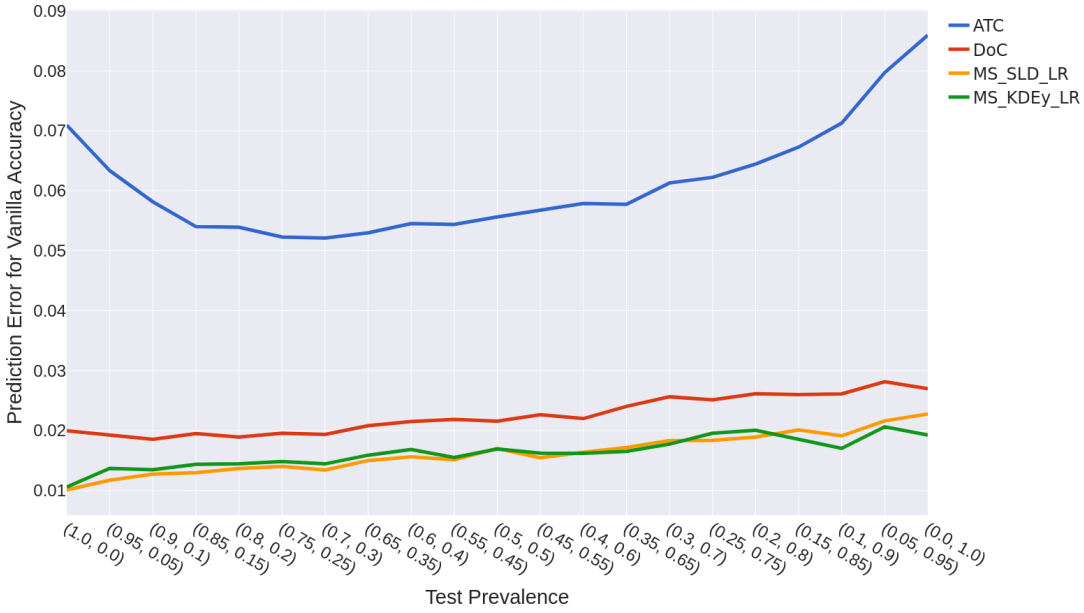

Figure 5.19: As in Figure 5.15 for dataset IMDB, but showing the test prevalence as the x-axis. Every point on the plot is the average of results obtained on the test samples for that test prevalence value over all training prevalence values, i.e., the average over $9 \times 100 = 900$ test samples.

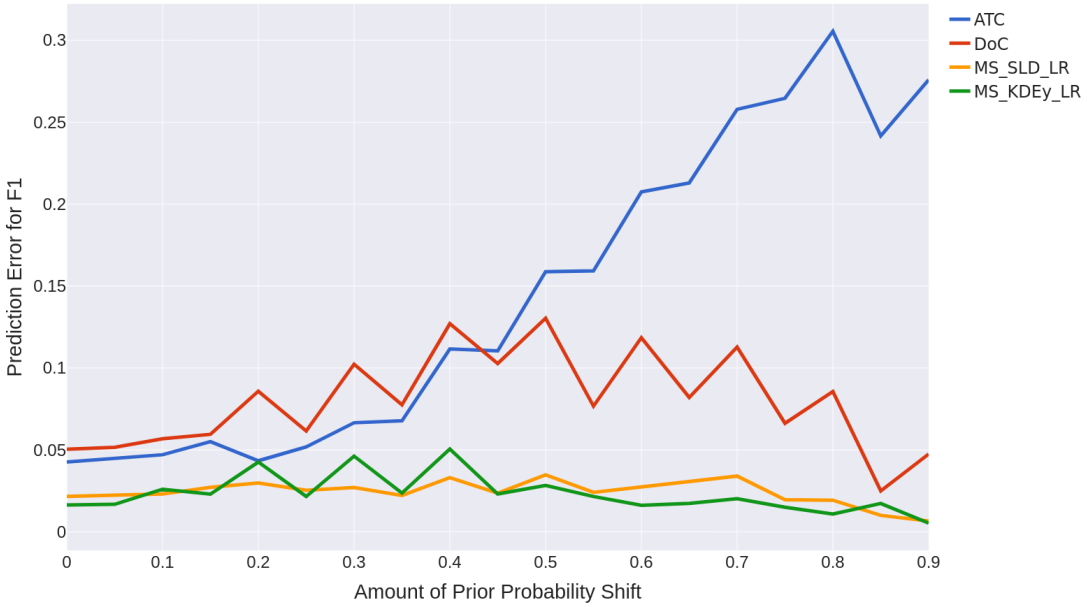

Figure 5.20: As in Figure 5.15 for dataset IMDB, but reporting F_1 as the prediction error on the y-axis.

prevalence of one method over the others, as the distribution of selected methods change significantly from one dataset to another.

We also conducted tests of statistical significance employing the Wilcoxon signed-rank test (Wilcoxon, 1945). We conducted this test on pairs of methods, obtaining a p-value for each test. This p-value indicates how much the averages obtained by two methods compared to each other are statistically significant: the lower the value the more statistically distinct are the two methods. In Table 5.4 we show the Wilcoxon p-values for all methods reported in Table 5.2 taken in pairs on the IMDB dataset. Methods bound by a p-value $p \geq 0.05$ are marked with $\cdot^{\ddagger}$, indicating a low decoupling between them; methods bound by a p-value $0.001 \leq p \leq 0.05$ are marked with $\cdot^{\dagger}$, indicating an intermediate level of decoupling; unmarked methods are considered statistically distinct by the Wilcoxon test. We can see how we get good statistical results overall, even though some methods show relatively high p-values, e.g., QuAcc(KDEy) and DoC.

These results are also reported in Table 5.2 with the same notation for methods compared to the best-performing approach.

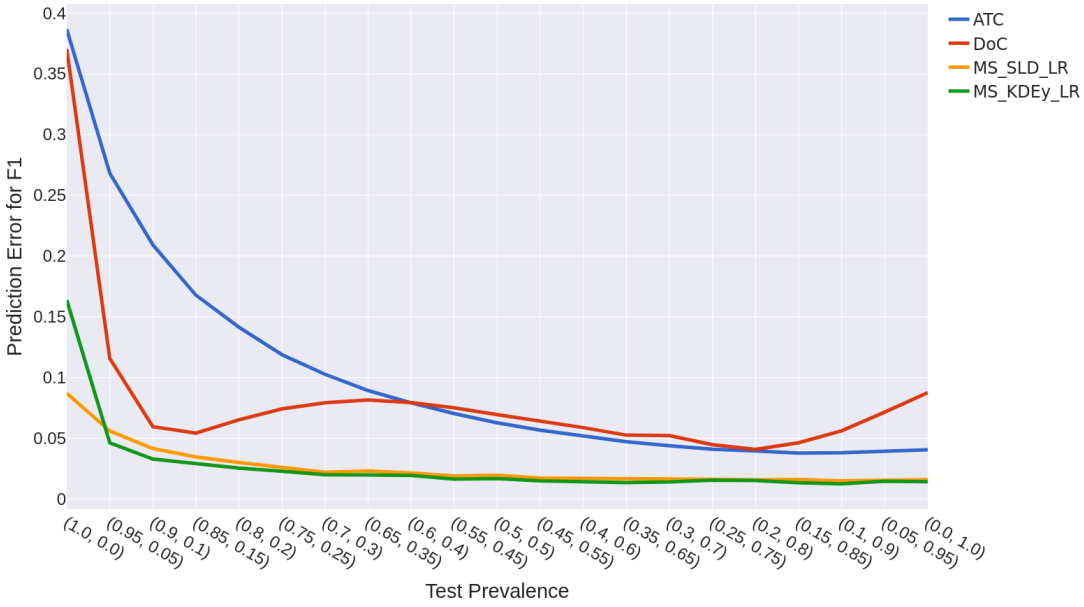

Figure 5.21: As in Figure 5.20 for dataset IMDB and error measure F_1, but showing the test prevalence as the x-axis. Every point on the plot is the average of results obtained on the test samples for that test prevalence value over all training prevalence values, i.e., the average over 9×100=900 test samples.

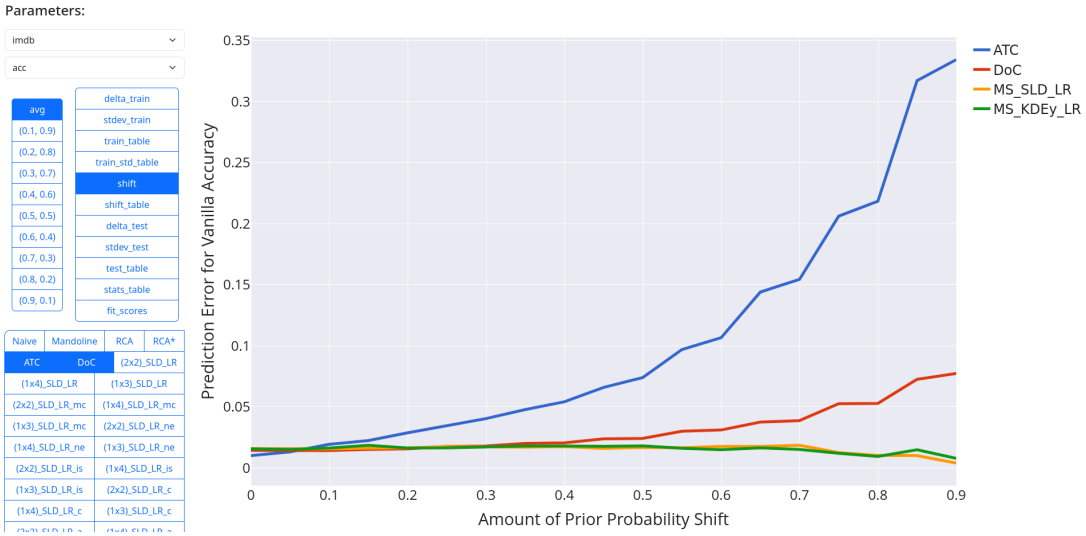

Figure 5.22: A screenshot of the web application we realised to visualise the results of our experiments.

		IMDB	CCAT	GCAT	MCAT
Baselines	Naïve	.1799 ± .208	.1274 ± .165	.1071 ± .150	.1183 ± .164
	RCA Elsahar and Gallé (2019)	.1085 ± .116	.0493 ± .056	.0519 ± .051	.0618 ± .076
	RCA* Elsahar and Gallé (2019)	.1099 ± .118	.0564 ± .073	.0504 ± .049	.0610 ± .076
	Mandoline Chen et al. (2021b)	.3616 ± .280	.1920 ± .176	.1581 ± .161	.3599 ± .346
	DoC Guillory et al. (2021)	.0226 ± .020	.0145 ± .013‡	.0112 ± .010	.0199 ± .022†
	ATC Garg et al. (2022)	.0613 ± .078	.0292 ± .036	.0151 ± .017	.0230 ± .030
Ours	QuAcc(CC)	.0474 ± .038	.0297 ± .024	.0201 ± .015	.0313 ± .042
	QuAcc(SLD)	**.0162 ± .013**	.0151 ± .014	.0097 ± .007	**.0136 ± .010**
	QuAcc(KDEy)	.0167 ± .018	**.0137 ± .012**	**.0090 ± .008**	.0143 ± .013‡
	Error reduction	28.32%	5.52%	19.64%	31.66%

(a) Vanilla Accuracy as the classifier accuracy measure.

		IMDB	CCAT	GCAT	MCAT
Baselines	Naïve	.1512 ± .226	.1327 ± .229	.1288 ± .225	.1300 ± .223
	RCA Elsahar and Gallé (2019)	.1204 ± .148	.1008 ± .134	.1058 ± .147	.1147 ± .151
	RCA* Elsahar and Gallé (2019)	.2085 ± .237	.2593 ± .284	.2347 ± .238	.2486 ± .250
	Mandoline Chen et al. (2021b)	—	—	—	—
	DoC Guillory et al. (2021)	.0809 ± .096	.0951 ± .109	.0947 ± .126	.0911 ± .123
	ATC Garg et al. (2022)	.1015 ± .133	.0798 ± .116	.0918 ± .141	.0765 ± .124
Ours	QuAcc(CC)	.0640 ± .091	.0499 ± .082	.0470 ± .094	.0492 ± .092
	QuAcc(SLD)	**.0259 ± .038**	**.0199 ± .032**	**.0221 ± .051**	**.0227 ± .055**
	QuAcc(KDEy)	.0292 ± .066	.0201 ± .051	.0240 ± .067	.0302 ± .080
	Error reduction	67.99%	75.06%	75.93%	75.08%

(b) F_1 as the classifier accuracy measure.

Table 5.2: Results of our experiments (expressed in terms of $E(A(h,T), \hat{A}(h,T))$) on the IMDB, CCAT, GCAT, MCAT datasets with vanilla accuracy in Table 5.2a and F_1 in Table 5.2b as the classifier accuracy measure A. **Boldface** indicates the best method, while underlining indicates the best baseline. Each figure is an average across all 9×21×100=18,900 combinations of a training sample and a test sample for the given dataset. The last row indicates the reduction in error obtained by the best method over the best baseline. No results are available for Mandoline with F_1 since (i) its proposers (Chen et al., 2021b) use vanilla accuracy only as the classifier accuracy measure, and since (ii) it is not obvious how to modify the Mandoline code for use with a measure different from Acc. Also the other baselines were tested by their proposers with vanilla accuracy only, but in their case it is immediate to understand how to modify the code for tackling F_1. An indication of Wilcoxon p-values are reported for methods comparing to the best performer. Values $p \geq 0.05$ are indicated with $\cdot^{\ddagger}$ while values $0.001 \leq p \leq 0.05$ are indicated with $\cdot^{\dagger}$.

5.6.10 A Web Application for Visualising Experimental Results

It is worth mentioning that we have also developed a web application focused on data visualisation that shows all results from our experiments. This application, which we make publicly available,[2] was realised using the "Dash" Python

[2] http://ilona.isti.cnr.it:33421/?root=output%2Fmain

Dataset	Quantifier	Additional Covariates	Recalibration	Class Balancing	The 1×4 variant	The 2×2 variant	The 1×3 variant
IMDB	SLD	81.49%	77.78%	59.26%	66.67%	11.11%	22.22%
	KDEy	71.43%	—	88.18%	55.56%	11.11%	33.33%
CCAT	SLD	62.97%	70.38%	77.78%	22.22%	66.67%	11.11%
	KDEy	78.06%	—	84.82%	22.22%	55.56%	22.22%
GCAT	SLD	76.93%	61.54%	65.39%	0.00%	55.56%	44.44%
	KDEy	71.36%	—	78.65%	11.11%	44.44%	44.44%
MCAT	SLD	62.97%	48.15%	74.08%	33.33%	66.67%	0.00%
	KDEy	71.03%	—	68.70%	22.22%	33.33%	44.44%

Table 5.3: Frequencies with which the model selection phase chooses (i) options Additional Covariates (see Section 4.4), Recalibration (for the SLD quantifier, as KDEy does not recalibrate the classifier), and Class Balancing (for logistic regression), and (ii) the three variants of our method (1×4, 2×2, 1×3). Concerning the values of the last three columns, note that they are all multiples of 11.11% (i.e., $\frac{1}{9}$); this is because, for each combination of a dataset and a quantifier (SLD or KDEy), 9 model selections (i.e., one for each pair (T_i, V_i)) are performed.

	RCA	RCA*	Mandoline	DoC	ATC	QuAcc(CC)	QuAcc(SLD)	QuAcc(KDEy)
RCA	1.0000e+0							
RCA*	5.8428e-185	1.0000e+0						
Mandoline	0.0000e+0	0.0000e+0	1.0000e+0					
DoC	0.0000e+0	0.0000e+0	0.0000e+0	1.0000e+0				
ATC	0.0000e+0	0.0000e+0	0.0000e+0	0.0000e+0	1.0000e+0			
QuAcc(CC)	0.0000e+0	0.0000e+0	0.0000e+0	0.0000e+0	1.7853e-15	1.0000e+0		
QuAcc(SLD)	0.0000e+0	0.0000e+0	0.0000e+0	6.2143e-3†	4.6722e-283	0.0000e+0	1.0000e+0	
QuAcc(KDEy)	0.0000e+0	0.0000e+0	0.0000e+0	4.0410e-1‡	0.0000e+0	0.0000e+0	2.2710e-7	1.0000e+0

Table 5.4: Wilcoxon p-values computed on the shown methods taken in pairs on the CCAT dataset. The p-values are computed across all 9×21×100=18,900 test samples for each pair. The table is represented as a triangular inferior matrix as results above the main diagonal repeat. Values $p \geq 0.05$ are indicated with $\cdot^{\ddagger}$ while values $0.001 \leq p \leq 0.05$ are indicated with $\cdot^{\dagger}$.

framework,[3]. A screenshot of the application can be seen in Figure 5.22. Via the application it is possible to select a dataset, a prediction accuracy measure and multiple methods at the same time. The plot updates dynamically when the selection changes. It is also possible to select different kinds of visualisation, both plots and tables. Each visualisation changes the way results are aggregated

[3] https://dash.plotly.com/

and averaged. Some representations are also available for each single training prevalence (selectable on the left). Each plot can be dynamically manipulated (e.g., selecting a specific area or zooming in and out) and can be downloaded as a PNG image.

5.6.11 Wrap-up

To summarise, the main observations we can derive from the results obtained from our experiments, as displayed in Tables 5.2 and 5.3, are the following:

1. The best performance, on all 8 combinations of a dataset and a classifier accuracy measure, is obtained by either QuAcc(SLD) or QuAcc(KDEy); this confirms the validity of the intuitions underlying QuAcc.

2. While the best performer is always one of QuAcc(SLD) and QuAcc(KDEy), it is not always the same. (This is true also for the baselines, since the best baseline is not always the same method.) QuAcc(SLD) is the best on 6 combinations of a dataset and a classifier accuracy measure, and QuAcc(KDEy) on 2. It is noteworthy, though, that only in 1 combination (CCAT+Acc) the best baseline (slightly) beats our 2nd best method; in the other 7 combinations both QuAcc(SLD) and QuAcc(KDEy) beat all baselines.

3. In terms of the amounts of error reduction that our best method obtains with respect to the best baseline (see last rows of Tables 5.2a and 5.2b), while these amounts are very respectable when Acc is the classifier accuracy measure (from +5.52% to +31.66%), they are dramatic when F_1 is the measure (from +67.99% to +75.93%). A detailed inspection of Table 5.2 shows that, while all methods perform better with Acc than with F_1 (no doubt due to the instability of F_1 (Sebastiani, 2015)), the difference in performance is small for QuAcc and very large for all the baselines. QuAcc thus also proves robust to "difficult" classifier accuracy measures.

4. Table 5.3 shows that all the three methods discussed in Sections 4.1 to 4.3 are chosen in the model selection phase a substantive number of times. Out of 72 invocations of model selection (4 datasets $\times$ 2 quantification methods $\times$ 9 (T_i, V_i) pairs), Method 1×4 is selected 21 times, Method 2×2 is selected 32 times, and Method 1×3 is selected 20 times. This shows that none of them is a strawman, while also confirming the intuition that the 2nd is superior to the 1st, given that it additionally exploits the knowledge of $\text{TP} \cup \text{FP}$ and $\text{TN} \cup \text{FN}$.

5. QuAcc(CC), a variant of QuAcc that uses the trivial "Classify and Count" (CC) quantifier, beats (see 3rd-to-last rows of Tables 5.2a and 5.2b) all baselines in 4 cases out of 8. Since CC is not robust to PPS (Esuli et al.,

2023, §1.2), this result shows that the idea of estimating the prevalence values of the cells of the contingency table is a good idea in itself. However, QuAcc(CC) always dramatically underperforms QuAcc(SLD) and QuAcc(KDEy), which shows that using "true" quantification algorithms robust to PPS for performing this estimation is also a good idea.

6. The best-performing baseline is always either DoC (Guillory et al., 2021) or ATC (Garg et al., 2022). RCA and RCA* (Elsahar and Gallé, 2019) are always worse than both DoC and ATC, and Mandoline (Chen et al., 2021b) always follows at a distance.

Chapter 6

Conclusion

In this thesis we have explored the problem of Classification Accuracy Prediction (CAP), a problem with many potential real-world applications that has not been extensively investigated in the literature.

We have presented QuAcc, a new method for CAP devised for scenarios in which the data are affected by prior probability shift (PPS), an important type of dataset shift. QuAcc is built on top of *quantification* methods robust to PPS, i.e., methods devised for estimating the class prevalence values in samples of unlabelled datapoints affected by PPS. QuAcc is based on the key intuition of viewing the cells of the contingency table, which is used for computing classifier accuracy, as classes, and of training a quantifier that estimates the values of these cells. The experiments we run on four large datasets simulate a wide variety (a) of amounts of training and/or test data imbalance, and (b) of amounts of PPS.

6.1 The Takeaway Message

The results of our experiments show that QuAcc systematically outperforms (on all combinations of a dataset and a classifier accuracy measure) all four state-of-the-art baseline methods we have tested, exhibiting average levels of error reduction, with respect to the best baseline method for the tested (dataset, classifier accuracy measure) combination used, ranging from +5.52% to 75.93%. These results pave the way for potential new CAP techniques built on top of quantification techniques, that we plan to investigate in the near future.

QuAcc is independent of the algorithm used for training the classifier, of the algorithm used for training the quantifier, of the function used for measuring classifier accuracy, and of the function used for measuring classifier accuracy prediction error. QuAcc can thus be instantiated with methods and evaluation measures different from those used in this thesis.

While our experiments have concentrated on the binary case, we note that two of the QuAcc variants we have presented (the 1×4 and 2×2 methods) extend

straightforwardly to the multiclass case.

6.2 Future Work

This research can be extended on several fronts, of which we here mention the most interesting ones.

6.2.1 Predicting the Accuracy of Multiclass Classification

This research can be extended by carrying out systematic experiments in the multiclass case too. While the extension to the multiclass case is straightforward for the 1×4 method, which would become a $1\times n^2$ method (n being the number of classes on which the original model issues predictions), and for the 2×2 method, which would become a $n \times n$, the extension of method 1×3 is non-trivial. This latter method, in fact, relies on an assumption (that of spatial contiguity between the FN region and the FP region) that proves fruitful in the binary case, as confirmed by our results, but does not have obvious equivalents in the multiclass case. A possible extension of the 1×3 method to the multiclass case thus requires further research.

6.2.2 Prior Probability Shift, Again

A further direction that we would like to explore consists of testing QuAcc also on the more challenging scenario in which the data V on which the quantifier is to be trained, and the data T on which the classifier was trained, are not IID (as in this thesis) but are themselves affected by prior probability shift. This experimental protocol would mirror a scenario that is likely to occur in practice, since we often cannot guarantee that the data on which we would like to train our quantifier is representative of the data that was used for training the classifier.

6.2.3 Types of Dataset Shift other than Prior Probability Shift

Additional future research that can be carried out concerns the problem of classifier accuracy prediction under types of dataset shift different from PPS, such as e.g.,covariate shift. This might not necessarily mean devising methods alternative to QuAcc, but might mean instantiating QuAcc with quantification methods which have proven robust to types of dataset shift different from PPS. This may prove nontrivial, though, since most of the quantification literature has focused on PPS (González et al., 2023), somehow neglecting other types of dataset shift.

Index

Bibliography

Alexandari, A., Kundaje, A., and Shrikumar, A. (2020). Maximum likelihood with bias-corrected calibration is hard-to-beat at label shift adaptation. In *Proceedings of the 37th International Conference on Machine Learning (ICML 2020)*, pages 222–232, Virtual Event.

Bella, A., Ferri, C., Hernández-Orallo, J., and Ramírez-Quintana, M. J. (2010). Quantification via probability estimators. In *Proceedings of the 11th IEEE International Conference on Data Mining (ICDM 2010)*, pages 737–742, Sydney, AU.

Bhaskaruni, D., Moss, F. P., and Lan, C. (2018). Estimating prediction qualities without ground truth: A revisit of the reverse testing framework. In *Proceedings of the 24th International Conference on Pattern Recognition (ICPR 2018)*, pages 49–54, Beijing, CN.

Bunse, M. and Morik, K. (2022). Unification of algorithms for quantification and unfolding. In *Proceedings of the 2nd International Workshop on Learning to Quantify (LQ 2022)*, pages 1–10, Grenoble, IT.

Chen, J., Liu, F., Avci, B., Wu, X., Liang, Y., and Jha, S. (2021a). Detecting errors and estimating accuracy on unlabeled data with self-training ensembles. In *Proceedings of the 35th Conference on Neural Information Processing Systems (NeurIPS 2021)*, pages 14980–14992, Virtual Event.

Chen, M., Goel, K., Sohoni, N. S., Poms, F., Fatahalian, K., and Ré, C. (2021b). Mandoline: Model evaluation under distribution shift. In *Proceedings of the 38th International Conference on Machine Learning (ICML 2021)*, pages 1617–1629, Virtual Event.

Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, B*, 39(1):1–38.

Deng, W. and Zheng, L. (2021). Are labels always necessary for classifier accuracy evaluation? In *Proceedings of the 2021 IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2021)*, pages 15069–15078, Virtual Event.

Dowson, D. C. and Landau, B. V. (1982). The Fréchet distance between multivariate normal distributions. *Journal of Multivariate Analysis*, 12(3):450–455.

Elsahar, H. and Gallé, M. (2019). To annotate or not? Predicting performance drop under domain shift. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2163–2173, Hong Kong, CN.

Esuli, A., Fabris, A., Moreo, A., and Sebastiani, F. (2023). *Learning to quantify*. Springer Nature, Cham, CH.

Esuli, A. and Sebastiani, F. (2014). Explicit loss minimization in quantification applications (preliminary draft). In *Proceedings of the 8th International Workshop on Information Filtering and Retrieval (DART 2014)*, pages 1–11, Pisa, IT.

Fan, W. and Davidson, I. (2006). Reverse testing: An efficient framework to select amongst classifiers under sample selection bias. In *Proceedings of the 12th ACM International Conference on Knowledge Discovery and Data Mining (KDD 2006)*, pages 147–156, Philadelphia, US.

Fawcett, T. and Flach, P. (2005). A response to Webb and Ting's 'On the application of ROC analysis to predict classification performance under varying class distributions'. *Machine Learning*, 58(1):33–38.

Firat, A. (2016). Unified framework for quantification. arXiv:1606.00868v1 [cs.LG] 2 Jun 2016.

Forman, G. (2005). Counting positives accurately despite inaccurate classification. In *Proceedings of the 16th European Conference on Machine Learning (ECML 2005)*, pages 564–575, Porto, PT.

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

Garg, S., Balakrishnan, S., Lipton, Z. C., Neyshabur, B., and Sedghi, H. (2022). Leveraging unlabeled data to predict out-of-distribution performance. In *Proceedings of the 10th International Conference on Learning Representations (ICLR 2022)*, Virtual Event.

Gart, J. J. and Buck, A. A. (1966). Comparison of a screening test and a reference test in epidemiologic studies: II. A probabilistic model for the comparison of diagnostic tests. *American Journal of Epidemiology*, 83(3):593–602.

González, P., Moreo, A., and Sebastiani, F. (2023). Binary quantification and dataset shift: An experimental investigation. arXiv:2310.04565 [cs.LG].

González-Castro, V., Alaiz-Rodríguez, R., Fernández-Robles, L., Guzmán-Martínez, R., and Alegre, E. (2010). Estimating class proportions in boar semen analysis using the Hellinger distance. In *Proceedings of the 23rd International Conference on Industrial Engineering and other Applications of Applied Intelligent Systems (IEA/AIE 2010)*, pages 284–293, Cordoba, ES.

Guillory, D., Shankar, V., Ebrahimi, S., Darrell, T., and Schmidt, L. (2021). Predicting with confidence on unseen distributions. In *Proceedings of the 2021 IEEE/CVF International Conference on Computer Vision (ICCV 2021)*, pages 1114–1124, Montreal, CA.

Jiang, H., Kim, B., Guan, M., and Gupta, M. (2018). To trust or not to trust a classifier. In *Proceedings of the 32nd Conference on Neural Information Processing Systems (NeurIPS 2018)*, pages 5546–5557, Montréal, CA.

Jiang, Y., Nagarajan, V., Baek, C., and Kolter, J. Z. (2022). Assessing generalization of SGD via disagreement. In *Proceedings of the International Conference on Learning Representations (ICLR 2022)*, Virtual Event.

Lewis, D. D., Yang, Y., Rose, T. G., and Li, F. (2004). RCV1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research*, 5:361–397.

Lundberg, S. M. and Lee, S. (2017). A unified approach to interpreting model predictions. In *Proceedings of the 31st Conference on Neural Information Processing Systems (NIPS 2017)*, pages 4765–4774, Long Beach, US.

Maas, A. L., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y., and Potts, C. (2011). Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics (ACL 2011)*, pages 142–150, Portland, US.

Moreo, A., Esuli, A., and Sebastiani, F. (2021). QuaPy: A Python-based framework for quantification. In *Proceedings of the 30th ACM International Conference on Knowledge Management (CIKM 2021)*, pages 4534–4543, Gold Coast, AU.

Moreo, A., González, P., and del Coz, J. J. (2023). Kernel density estimation for multiclass quantification. arXiv:2401.00490 [cs.LG].

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

Quiñonero-Candela, J., Sugiyama, M., Schwaighofer, A., and Lawrence, N. D., editors (2009). *Dataset shift in machine learning*. The MIT Press, Cambridge, US.

Redyuk, S., Schelter, S., Rukat, T., Markl, V., and Bießmann, F. (2019). Learning to validate the predictions of black box machine learning models on unseen data. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics (HILDA@SIGMOD 2019)*, pages 4:1–4:4, Amsterdam, NL.

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

Sebastiani, F. (2015). An axiomatically derived measure for the evaluation of classification algorithms. In *Proceedings of the 5th ACM International Conference on the Theory of Information Retrieval (ICTIR 2015)*, pages 11–20, Northampton, US.

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

Silva, I. S. and Veloso, A. (2022). Automatic model evaluation using feature importance patterns on unlabeled data. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN 2022)*, pages 1–8, Padova, IT.

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

Sugiyama, M., Nakajima, S., Kashima, H., Buenau, P., and Kawanabe, M. (2007). Direct importance estimation with model selection and its application to covariate shift adaptation. In *Proceedings of the 21st Conference on Advances in Neural Information Processing Systems (NIPS 2007)*, pages 1433–1440, Vancouver, CA.

Wilcoxon, F. (1945). Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6):80–83.

You, S. D., Liu, H., and Liu, C. (2022). Predicting classification accuracy of unlabeled datasets using multiple deep neural networks. *IEEE Access*, 10:44627–44637.

Ziegler, A. and Czyż, P. (2023). Bayesian quantification with black-box estimators. arXiv:2302.09159 [stat.ML].