Université catholique de Louvain
Institut de Statistique, Biostatistique et Sciences Actuarielles

Master thesis

Quantification and learning algorithms to manage prior probability shift

Olivier Caelen
UCL noma: 7700-11-00

Supervisors:
Prof. Marco Saerens
Prof. Johan Segers

Reader:
Prof. Bernadette Govaerts

2017-2018

Acknowledgements

First of all, I would like to express my gratitude to my two supervisors, Prof. Marco Saerens and Prof. Johan Segers for accepting to be my guide during this year and for their inspiration, patience and encouragements. The meetings that I had with them during my master thesis were for me real moments of discussion and they helped me to have new points of views. I would like to thank Prof. Marco Saerens specifically for introducing me to the topic of quantification learning.

My acknowledgements are also going to Prof. Bernadette Govaerts for accepting to be member of my jury and for the time spent in the lecture of this master thesis.

I would like to thank all the members of the Louvain School of Statistics, Biostatistics and Actuarial Science for introducing me to the amazing and interesting world of Statistics.

A special thanks to Sylvain Courtain who helped me a lot during our common work this year on quantification learning problems.

Finally, I would especially express my very profound gratitude to my parents and my wife Alexandra for the love, support, and constant encouragement I have got over the years.

And Alexandra, I know that these last four years were difficult for you. Thank you for everything. I know that you also had to do a lot of sacrifices and I promise you that this has been my last master thesis ;-)

Contents

Chapter 1

Introduction

In this work, we consider a *transduction* learning scenario [17] with *dataset shift* [37] in the case where all the training and testing data are available in advance (transduction learning) and where it is assumed possible that the distribution generating the training and testing data changes between the two situations (dataset shift). The purpose of this work is to study techniques able to use the training data and the inputs of the testing data in order to adapt the predictive models to the dataset shift and consequently have better prediction accuracies on the testing points.

This work studies more specifically the problem of dataset shifts in a supervised learning context when the marginal probability of the output variable changes between the training and the testing environments and when this new marginal probability in the testing environment is unknown. This problem is called in the scientific literature *prior probability shift*[1] but we can find it under other names like *global shift* [24].

Besides the problem of adapting the classifier to the new distribution, it can also be interesting to estimate only the new marginal probability of the output variable in the testing set. This problem is called *quantification learning* [21, 48, 18, 20, 42]. The key difference between the quantification and the classification tasks is that for quantification, the ultimate goal is to estimate the prevalence of each class in the testing set, whereas in the other task this is only an intermediate step, the ultimate goal being to improve classification performance on data drawn from a new testing distribution.

Classical machine learning algorithms are solving two nested optimization problems: the *structural identification* (i.e. search of the optimal hyper-parameters) and the *parametric identification* (i.e. given hyper-parameters, search of the optimal parameters). In this work, we investigate how the testing sets can be used to adapt the parametric identification phase of the learning problem to the dataset shift.

1.1 Motivation

Shifts in the data distribution frequently occur in real-world classification problems. As examples of changes in the data distributions, let us consider the following scenarios:

[1] Note that the expression '*prior probability shift*' can be confusing. If $P(Y = y|X = x)$ is the learning task that we try to estimate where X and Y are respectively the input and the output variables, then, in this context, $P(Y = y)$ is called the prior probability (which is ultimately nothing else than the marginal probability of the output variable). In *prior probability shift*, it is assumed that this marginal probability of the output Y can change between the training and the testing environments.

- In several binary classification problems, the two classes are not equally represented in the dataset. For example, in fraud detection, fraudulent transactions are normally outnumbered by genuine ones [8]. When one class is underrepresented in a dataset, the data is said to be *unbalanced*. A common strategy for dealing with unbalanced classification tasks is undersampling the majority class in the training set before learning a classifier [10, 9]. The assumption behind this strategy is that in the majority class there are many redundant observations and randomly removing some of them does not change the estimation of the within-class distribution. If we make the assumption that training and testing sets come from the same distribution, then when the training set is unbalanced, the testing set has a skewed distribution as well. By removing majority class instances, the training set is artificially rebalanced. As a consequence, we obtain different distributions for the training and testing sets, violating the basic assumption in machine learning that the training and testing sets are drawn from the same underlying distribution.

- In a medical experimental setting with case control, it is common to fix in advance the number of patients suffering from an illness and the number of healthy individuals. All data collected from this study could be used to make a predictive model able to detect newly infected patients. In this case, the prevalence of the two classes in the training set does not reflect reality and once we apply the model in real new case situations, we have no idea of the true a priori probability of the disease [38].

- Consider a *geographic information system* (GIS) platform able to automatically classify areas of geographical maps based on remote sensing informations like satellite images. Each area of a map under examination has to be labeled automatically by the GIS platform according to its ground type (i.e. forest, house, farm, water, street, etc.). According to the region where the satellite image is taken (agricultural zone, urban zone, etc.), the prevalence of each type of soil can strongly change. As the a priori probabilities are unknown in advance and considerably vary from one map to another, the marginal probability of the output variable in the training set could be very far from the one of the test set. Therefore, it is necessary to recalibrate the model on each new map such that the GIS platform can adapt to each new classification problem [27].

- This phenomenon of change in prevalence can also be observed in helpdesk callcenter services, where the occurrence of certain support issues can vary over time (e.g. there are more reports of cracked screens in the United States on July 4th, the U.S. Independence Day [48]). The cause of an issue is stable but the prevalence of the type of issue changes over time.

- In [18], the authors highlight the fact that most of the previous studies dealing with *tweet* sentiment classification use a suboptimal approach. Usually, the ultimate goal of most of these studies is not to estimate the label of an individual tweet but to study the distribution of a set of tweets across the classes of interest. In other words, the interest in such studies is not at the individual level, but at the aggregated level. Tweet sentiment classification studies that focus on the aggregate level are concerned with estimating the *prevalence* (or *relative frequency*) of each class of interest in the unlabelled dataset. This task is known as *quantification*. The authors of [18] demonstrate the importance of using optimal methods for each specific problem. The fact

that quantification learning is still sometimes unknown by the machine learning community can lead to suboptimal solutions [21].

1.2 First formalization of the problem

This section describes the problem studied in this work from a formal point of view. We have two populations from which observations can be generated:

- The first population is following the unknown distribution $F_{XY}(x, y)$ where $Y \in \mathcal{Y} = \{\omega_0, \omega_1, \dots \omega_{|\mathcal{Y}|-1}\}$ is a discrete random variable and the variable X takes its values in $\mathcal{X} \subset \mathbb{R}^d$. From $F_{XY}(x, y)$, we assume that a set of $N_{\mathfrak{d}}$ observations $(x^{\mathfrak{d}}, y^{\mathfrak{d}})$ is available. These $N_{\mathfrak{d}}$ observations are stored in a set $\mathfrak{d} = \{(x_i^{\mathfrak{d}}, y_i^{\mathfrak{d}})\}_{i=1}^{N_{\mathfrak{d}}}$ called the training set. It is common to say that these observations are coming from the training (source) *environment*.

- The second population is following the unknown distribution $F_{\widetilde{X}\widetilde{Y}}(x, y)$ with $\widetilde{X} \in \mathcal{X}$ and $\widetilde{Y} \in \mathcal{Y}$. This population has the particularity that the random variable Y is latent, meaning that no observations are available from this variable. From the second population, we assume that a set of $N_{\mathfrak{t}}$ observations $x^{\mathfrak{d}}$ of $\widetilde{X}$ is available. These $N_{\mathfrak{t}}$ observations are stored in a set $\mathfrak{t} = \{x_i^{\mathfrak{t}}\}_{i=1}^{N_{\mathfrak{t}}}$ called the testing set. It is common to say that these observations are coming from the testing (target) *environment*.

In the *learning problem* [3, 14, 23, 33, 44, 47] studied in this work, we assume the possibility that the distributions from the two populations are different, i.e. $F_{XY}(\cdot, \cdot) \neq F_{\widetilde{X}\widetilde{Y}}(\cdot, \cdot)$.

We assume a very specific type of change between the distributions of the two populations $F_{XY}(\cdot, \cdot)$ and $F_{\widetilde{X}\widetilde{Y}}(\cdot, \cdot)$. In this work, we assume that the marginal distributions $P(Y = \cdot)$ and $P(\widetilde{Y} = \cdot)$ are different (i.e. *prior probability shift* [21, 48, 18, 20, 42]).

Given a the two couples of random variables (X, Y) and $(\widetilde{X}, \widetilde{Y})$, in *prior probability shift*, it is assumed that:

- The within class probability density is conserved between the source and the target environments, i.e.

$$\forall x \in \mathcal{X} \text{ and } \forall y \in \mathcal{Y} : f_{X|Y}(x|y) = f_{\widetilde{X}|\widetilde{Y}}(x|y).$$

- The marginal distribution of the output variable changes[2], i.e.

$$\exists y \in \mathcal{Y} : P(Y = y) \neq P(\widetilde{Y} = y).$$

Thanks to the Bayes' theorem, we have

$$P(Y = y|X = x) = \frac{f_{X|Y}(x|y)P(Y = y)}{f_X(x)}$$

and

$$P(\widetilde{Y} = y|\widetilde{X} = x) = \frac{f_{\widetilde{X}|\widetilde{Y}}(x|y)P(\widetilde{Y} = y)}{f_{\widetilde{X}}(x)}.$$

[2]Note that due to the constraint that the probabilities must sum to one, there is actually at least *two* values for which the marginal distribution of the output variable changes.

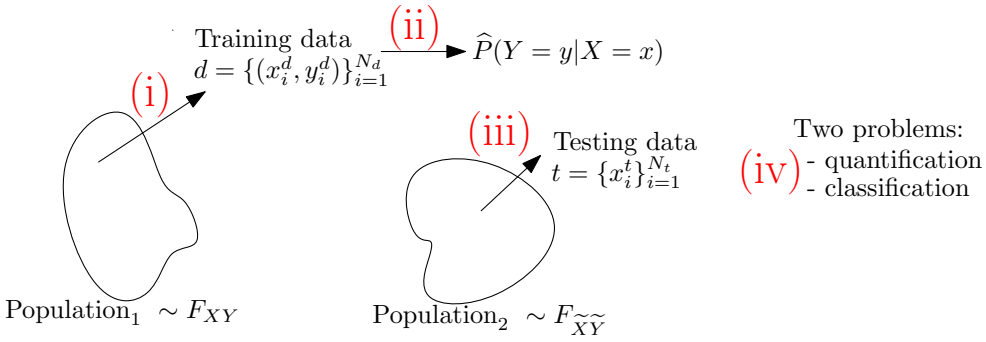

Figure 1.1: (i) A training set $\mathfrak{d}$ is generated from the first population. (ii) An estimation of the conditional probability to observe $Y = y$ knowing $X = x$ is extracted from $\mathfrak{d}$. (iii) A testing set $\mathfrak{t}$ is generated from the second population where $F_{XY}(\cdot, \cdot) \neq F_{\widetilde{X}\widetilde{Y}}(\cdot, \cdot)$. (iv) The two following problems can be considered. In quantification, the marginal $P(\widetilde{Y} = \cdot)$ is estimated. In classification, the conditional probability estimated in F_{XY} is recalibrated for $F_{\widetilde{X}\widetilde{Y}}$.

Although the within class probability density is conserved, from the Bayes' theorem, we can see that a change in the marginal distribution of the output variable can implied that $P(Y = \cdot|X = \cdot) \neq P(\widetilde{Y} = \cdot|\widetilde{X} = \cdot)$.

Note that as the within class probability density is conserved, this information about $f_{X|Y}(\cdot|\cdot)$ could be exchanged (*transferred*) between the training environment (i.e. first population) and the testing environment (i.e. second population). As we will see, the algorithms for quantification learning are often based on this idea of transferring this within class probability density.

Two types of problems are studied in this work:

- If we are interested in the estimation of $P(\widetilde{Y} = \cdot)$ in the testing environment then it is a *quantification* learning problem. Note that quantification goes under different names in different fields and different papers. It is variously called *prevalence estimation* [2], *class probability re-estimation* [1], *class prior estimation* [49], and *class distribution estimation* [22, 28].

- If $P(Y = y|X = x)$ has been learned (i.e. estimated) in the context of the first population and if we want to use this predictive model in the context of the testing environment then $\widehat{P}(Y = y|X = x)$ must be recalibrated. It is a *classification* learning problem.

Figure 1.1 summarizes the quantification and the classification problems studied in this work.

The learning problem studied in this work is called *prior probability shift* and is a type of *transfer learning* problem [35, 40, 30]. A transfer learning problem defines a wide general framework where knowledge obtained in one (or many) source environment(s) can be transferred to one (or many) target environment(s) in order to improve machine learning activities in the target environment(s). Chapter 2 introduces, as contribution of this work, a new general formalism and a new definition (page 11) for transfer learning. The prior probability shift that we will consider in this work is a rather simple transfer learning problem with only one source environment and one target environment.

- The source environment contains $F_{XY}(\cdot,\cdot)$ from which a training set $\mathfrak{d}$ is available.
- The target environment contains $F_{\widetilde{X}\widetilde{Y}}(\cdot,\cdot)$ from which a testing set $\mathfrak{t}$ is available.

Note that as the testing set is available during the learning phase, *prior probability shift* is a *transductive* learning problem [17].

There are two families of algorithms for quantification learning when a prior probability shift has been introduced.

- The first class of methods is the *adjusted classify and count* method or ACC (chapter 4). In ACC, a predictive model h learns $\mathfrak{d}$ from the first population. The model h is then applied (by cross-validation) on the same training set $\mathfrak{d}$ to compute an estimation $\widehat{P}(h(X) = \omega_k | Y = \omega_j)$. The same model is also applied on the testing set $\mathfrak{t}$ to compute an estimation $\widehat{P}(h(\widetilde{X}) = \omega)$. By using properties from transfer learning, these two estimations are then used to compute the probability $\widehat{P}(\widetilde{Y} = \omega)$, which is what quantification learning tries to estimate.

  To get $\widehat{P}(\widetilde{Y} = \omega)$, a system of equations must be solved, but it may happen that the solution proposed by the system is not a distribution (i.e. the solutions are not in $[0, 1]$ and/or do not sum to one). As a contribution of this work[3], we propose the use of a quadratic program with constraints to solve the system of linear equations (section 4.3 at page 37).

- The second method concerns the approaches based on the *expectation-maximization* (EM) method (chapter 6). It is to highlight that the *class distribution estimation* (CDE) method (chapter 5) is part of the family of the EM methods [42]. The EM method is an iterative method that tries, during each step of the loop, to estimate both the unknown marginal distribution $P(\widetilde{Y} = \cdot)$ and the unknown conditional distribution $P(\widetilde{Y} = y | \widetilde{X} = x)$.

  In section 6.3 at page 58, a method for adjusting, with the EM method, the bias term of a classifier in presence of prior probability shift is introduced[4]. As contribution of this work, in section 6.3, we have derived a new way to obtain the equation giving the adjustment of the bias term.

  We have also empirically shown that sometimes the EM algorithm can strongly diverge to a bad solution when $P(\widetilde{Y} = y | \widetilde{X} = x)$ is poorly estimated (sections 6.1.2 and 6.1.3 at page 51 and 53). As contribution of this work, we propose a modification of the original EM algorithm able to stop the iteration when it starts to diverge (section 6.2 at page 56).

1.3 Contributions

- A new definition of transfer learning is introduced (page 11).
- We propose, in section 6.3 at page 58, a new way to get the equation (3.20) at page 32 for adjusting, with the EM method, the bias term of a softmax classifier in presence

[3] Based on an idea introduced by Prof. Johan Segers.

[4] The method was first introduced in a still unpublished paper *Adjusting the Bias Term of Classifiers to Unknown Prior* written by Prof. Marco Saerens and Prof. Christine Decaestecker.

of prior probability shift. As expected, we have empirically shown that adjusting the probability $P(Y = y|X = x)$ or directly adjusting the bias term when a softmax classifier is used, leads to the exact same result (page 59).

- We also propose a new stopping criterion for the EM algorithm (section 6.2) and empirical experiments show that our version significantly outperforms the original EM algorithm in many cases.
- The ACC adjustment algorithm is adapted to use a quadratic program with constraints during the evaluation of the system of linear equations (section 4.3).
- An experimental design is applied on 25 datasets to evaluate the algorithms presented in this work (chapter 7). The full Python implementation of the experimental design is available on GitLab[5] for research reproducibility.

1.4 Structure of this master thesis

This work is structured as follows:

- **Chapter 1**: Introduction.
- **Chapter 2**: Literature review on the topic of transfer learning. As contribution, a new general definition for transfer learning is proposed. Note that a focus is given on prior probability shift which is the subtype of transfer learning problems that we will cover.
- **Chapter 3**: Some prerequisites needed for a good understanding of the rest of the document. This chapter covers three main topics. It starts with a general introduction about machine learning. Then the EM algorithm, needed for chapter 6, is introduced. The chapter ends with a section about the recalibration method in presence of prior probability shift when the new marginal probability of the output variable is known. This recalibration method is important because it makes the link between the quantification and classification problems already defined at page 4 in section 1.2.
- **Chapter 4**: The *adjusted classify and count* (ACC) algorithm [6] to solve the prior probability shift problem when the new prior is unknown. This method asks for the solution of a system of equations. As a contribution of this work, we propose the use of a constrained quadratic optimization method.
- **Chapter 5**: The *class distribution estimation* (CDE) algorithm to solve the prior probability shift problem when the new prior is unknown.
- **Chapter 6**: The *expectation-maximization* (EM) algorithm to solve the prior probability shift problem when the new prior is unknown. We also introduce a new stopping criterion for the EM algorithm.
- **Chapter 7**: The experimental evaluation of the algorithms introduced in chapter 4, 5 and 6.
- **Chapter 8**: Conclusion of this work.

[5] https://gitlab.com/ocaelen/exp-masterthesis-stat

[6] Also known as the *confusion matrix approach*

1.5 Table of symbols

Concepts and notation are introduced when needed. For an additional quick reference, here is a short list of the most important symbols:

$F_{\cdot}(\cdot)$	Cumulative distribution function
$f_{\cdot}(\cdot)$	Probability density function
$g_y(x)$	Estimation of the conditional probability: $g_y(x) = \widehat{P}(Y = y \mid X = x)$
$\mathfrak{d}$	Training dataset
$\mathfrak{t}$	Testing dataset
$\mathcal{X}$	Support of the input
$\mathcal{Y}$	Support of the output
$\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$	Support of an input/output observation
d	Input dimension
$\mathcal{I}^S$	Structural identification algorithm
$\mathcal{I}^P$	Parametric identification algorithm
Λ	Set of different model parameters
α	Vector with given model parameters
$\widehat{\alpha}$	Estimation of α by empirical risk minimization (ERM)
$L(\cdot, \cdot)$	Loss function
$R(\cdot)$	Function risk of a given model
$R_{emp}(\cdot)$	Empirical risk (an estimation of R)
$G(\cdot, \cdot)$	Generalization error
$m(\cdot)$	Target function (in regression)
$h(\cdot, \cdot)$	Predictive model (also called hypotheses)
ϵ	Noise
$\mathscr{L}$	Likelihood
ℓ	Log likelihood

Chapter 2

A general framework for transfer learning

2.1 Definition

In this work, we assume to be in a context where a shift in the data distribution is present between the training environment and the testing environment. This problem is closely related to another field of study known under various terms but the most common one tends to be *transfer learning*. Many surveys about transfer learning are available in the scientific literature [35, 40, 30]. Based on these articles, this section proposes a comprehensive overview of some types of problems that can be encountered in transfer learning and some of the most common methods to solve them.

Most of the machine learning methods assume that all the data involved in the learning and the prediction phases are generated independently and are coming from the same distribution. But in many real situations, data can come from different sources and transfer learning is one of the machine learning research fields that aims to transfer knowledge from one or more source environments to one or more target environments. The central idea of transfer learning is that the experience gained in a learning environment can help to improve learning performance in a related but different environment.

In this work, we consider the context of *offline* learning, meaning that we have static datasets. This is in contrast with *online* or *batch* learning where new fresh data are continually received and where the learning algorithm updates its parameters regularly after consuming these new data. *Online* learning and *batch* learning also have problems when the distribution that generates the data changes [13, 26] but we will not consider these cases.

More formally, for the rest of the explanation, let us suppose that we have two sets of random variable pairs.

- The first set of random variables $S_o = \{(X_i, Y_i)\}_{i=1}^{|S_o|}$ contains $|S_o|$ different input/output pairs (X_i, Y_i) with $i = 1, \ldots, |S_o|$, where the pair (X_i, Y_i) follows the distribution $F_{(X_i Y_i)}(x, y)$ and where $\mathcal{X}_i$ and $\mathcal{Y}_i$ are the two respective supports. The set S_o is characterized by the fact that in its pairs both variables X_i and Y_i are observable[1]. Let $\mathfrak{d}_i = \{(x_1^{\mathfrak{d}_i}, y_1^{\mathfrak{d}_i}), \ldots, (x_{N_{\mathfrak{d}_i}}^{\mathfrak{d}_i}, y_{N_{\mathfrak{d}_i}}^{\mathfrak{d}_i})\}$ be a set of $N_{\mathfrak{d}_i}$ observations extracted from (X_i, Y_i).

  To summarize the situation, we have $|S_o|$ distributions where both the input and the

[1] The suffix 'o' in S_o stands for 'observable'.

output variables are observable and from where $|S_o|$ sets of observations $\mathfrak{d}_i$ can be generated:

$$\begin{cases} F_{(X_1Y_1)}(x,y) & \underset{iid}{\longrightarrow} \mathfrak{d}_1 = \{(x_1^{\mathfrak{d}_1}, y_1^{\mathfrak{d}_1}), \ldots, (x_{N_{\mathfrak{d}_1}}^{\mathfrak{d}_1}, y_{N_{\mathfrak{d}_1}}^{\mathfrak{d}_1})\} \\ F_{(X_2Y_2)}(x,y) & \underset{iid}{\longrightarrow} \mathfrak{d}_2 = \{(x_1^{\mathfrak{d}_2}, y_1^{\mathfrak{d}_2}), \ldots, (x_{N_{\mathfrak{d}_2}}^{\mathfrak{d}_2}, y_{N_{\mathfrak{d}_2}}^{\mathfrak{d}_2})\} \\ & \vdots \\ F_{(X_{|S_o|}Y_{|S_o|})}(x,y) & \underset{iid}{\longrightarrow} \mathfrak{d}_{|S_o|} = \{(x_1^{\mathfrak{d}_{|S_o|}}, y_1^{\mathfrak{d}_{|S_o|}}), \ldots, (x_{N_{\mathfrak{d}_{|S_o|}}}^{\mathfrak{d}_{|S_o|}}, y_{N_{\mathfrak{d}_{|S_o|}}}^{\mathfrak{d}_{|S_o|}})\} \end{cases}$$

- The second set of random variables $S_l = \{(\widetilde{X}_j, \widetilde{Y}_j)\}_{j=1}^{|S_l|}$ contains $|S_l|$ different pairs following the distributions $F_{(\widetilde{X}_j\widetilde{Y}_j)}(x,y)$ with $j = 1, \ldots, |S_l|$ and where $\widetilde{\mathcal{X}}_j$ and $\widetilde{\mathcal{Y}}_j$ are the supports. The set S_l is characterized by the fact that the outputs $\widetilde{Y}_j$ are latent unobservable random variables[2]. Let $\widetilde{\mathfrak{d}}_j = \{x_1^{\widetilde{\mathfrak{d}}_j}, \ldots, x_{N_{\widetilde{\mathfrak{d}}_j}}^{\widetilde{\mathfrak{d}}_j}\}$ be a set of $N_{\widetilde{\mathfrak{d}}_j}$ observations extracted from $(\widetilde{X}_j, \widetilde{Y}_j)$ where $\widetilde{Y}_j$ is latent.

  To summarize the situation, we have $|S_l|$ distributions where only the input variable is observable and from where $|S_l|$ sets of observations $\widetilde{\mathfrak{d}}_j$ can be generated:

$$\begin{cases} F_{(\widetilde{X}_1\widetilde{Y}_1)}(x,y) & \underset{iid}{\longrightarrow} \widetilde{\mathfrak{d}}_1 = \{x_1^{\widetilde{\mathfrak{d}}_1}, \ldots, x_{N_{\widetilde{\mathfrak{d}}_1}}^{\widetilde{\mathfrak{d}}_1}\} \\ F_{(\widetilde{X}_2\widetilde{Y}_2)}(x,y) & \underset{iid}{\longrightarrow} \widetilde{\mathfrak{d}}_2 = \{x_1^{\widetilde{\mathfrak{d}}_2}, \ldots, x_{N_{\widetilde{\mathfrak{d}}_2}}^{\widetilde{\mathfrak{d}}_2}\} \\ & \vdots \\ F_{(\widetilde{X}_{|S_l|}\widetilde{Y}_{|S_l|})}(x,y) & \underset{iid}{\longrightarrow} \widetilde{\mathfrak{d}}_{|S_l|} = \{x_1^{\widetilde{\mathfrak{d}}_{|S_l|}}, \ldots, x_{N_{\widetilde{\mathfrak{d}}_{|S_l|}}}^{\widetilde{\mathfrak{d}}_{|S_l|}}\} \end{cases}$$

We will now assume that there are two types of environments: the source environment and the target environment. The source environment is close to the target one and could be used to help machine learning activities in the target environment. The central idea of transfer learning is that the experience gained in the source environment can help to improve machine learning performances in a related but different target environment. It should be emphasized that the goal is not to make predictions in the source environment. The only purpose of the data in the source environment is to make better predictions in the target environment.

Let

$$S_t \subseteq \{\mathfrak{d}_1, \ldots, \mathfrak{d}_{|S_o|}\} \cup \{\widetilde{\mathfrak{d}}_1, \ldots, \widetilde{\mathfrak{d}}_{|S_l|}\}$$

be the subset of all the observations in the target environment[3] and let

$$S_s = \left(\{\mathfrak{d}_1, \ldots, \mathfrak{d}_{|S_o|}\} \cup \{\widetilde{\mathfrak{d}}_1, \ldots, \widetilde{\mathfrak{d}}_{|S_l|}\}\right) \setminus S_t$$

be the subset with all the observations in the source environment[4]. The target environment is the environment where someone would like to infer models in order to do predictions or to do any other kind of machine learning activities like clustering. The goal is not to infer

[2]The suffix 'l' in S_l stands for 'latent'.
[3]The suffix 't' in S_t stands for 'target'.
[4]The suffix 's' in S_s stands for 'source'.

models directly in the source environment. The data in the source environment are only used to improve machine learning activities in the target environment.

Note that S_t and S_s can both contain supervised ($\mathfrak{d}$, with label) and unsupervised ($\widetilde{\mathfrak{d}}$, without labels) observations.

In order to be able to make the distinction between the unsupervised observations in target environment S_t on which a prediction is asked and the ones on which no prediction is asked, the unsupervised observations on which a prediction is asked are noted $\mathfrak{t}$ (rather than $\widetilde{\mathfrak{d}}$). Note that when S_t contains $\mathfrak{t}$ elements than we are in a transduction learning context because the testing observations $\mathfrak{t}$ are known in advance.

Based on our formalism, we propose the following general definition for transfer learning:

Definition 1. *Given $|S_l| + |S_o|$ distributions with at least one distribution that is different from the others, transfer learning studies ways to transfer knowledge from the data observed in the source environments S_s in order to do better machine leaning activities on the data extracted from the target environments S_t.*

Our definition includes a large number of different types of problems. The term '*machine learning activities*' includes any kind of tasks: building a predictive model, doing clustering, reinforcement learning,...

Notably, it is necessary that at least one of the $|S_l| + |S_o|$ distributions is different from the other ones because when all the input/output random variables in the source and target environments follow the same distribution, that is to say

$$(X_1 Y_1) \sim \ldots \sim (X_{|S_o|} Y_{|S_o|}) \sim (\widetilde{X}_1 \widetilde{Y}_1) \sim \ldots \sim (\widetilde{X}_{|S_l|} \widetilde{Y}_{|S_l|}) \sim F_{XY}(x, y)$$

then the learning problem becomes a traditional machine learning problem without changes in the data distributions.

Therefore, although very close, transfer learning cannot be considered as a *semi supervised* learning problem [50]. Semi supervised learning refers to a set of learning methods able to use both labeled and unlabeled data for the training of the predictive model. Although some labels are missing in the training set, semi supervised learning assumes that there is no change in the probability distributions that generated the data. In contrast, transfer learning assumes that some distributions are different. Based on our previously defined formalism, *semi supervised* learning problems can be defined as follows:

- As there is no transfer, the source environment is empty: $S_s = \varnothing$.
- There are two pairs of random variables in the target environment: (X, Y) and $(\widetilde{X}, \widetilde{Y})$. These two pairs of variables follow the same probability distribution $F_{XY}(x, y)$:

  $$(X, Y) \sim F_{XY}(x, y) \text{ and } (\widetilde{X}, \widetilde{Y}) \sim F_{XY}(x, y)$$

  but in $(\widetilde{X}, \widetilde{Y})$ the output variable $\widetilde{Y}$ is latent (not observable).
- From (X, Y) and $(\widetilde{X}, \widetilde{Y})$, two sets of observations are generated $\mathfrak{d}$ and $\widetilde{\mathfrak{d}}$. The training set is composed of these two sets of observations $\{\mathfrak{d}, \widetilde{\mathfrak{d}}\}$. In other words, the training set contains supervised and unsupervised observations but both are generated from the same unknown distribution (i.e. no dataset shift).
- The semi supervised learning problem consists of building a predictive model with this training dataset containing supervised and unsupervised observations.

In [35], another definition of transfer learning is given. We will now see that our definition is compatible with the definition of transfer learning given in [35]. Before discussing this definition, it is needed to define a *domain* and a *task*. Let $D = \{\mathcal{X}, F_X(\cdot)\}$ be a *domain* defined by two components: a feature space $\mathcal{X}$ (i.e. the support of the input random variable) and a probability distribution $F_X(\cdot)$, and let $\mathcal{T} = \{\mathcal{Y}, F_{Y|X}(\cdot|\cdot)\}$ be a *task* defined by two components: a label space (i.e. the support of the output random variable) and a conditional probability distribution $F_{Y|X}(\cdot|\cdot)$. In [35], transfer learning is defined as follows (definition 1 of [35]):

Definition 2. *Given a source domain D_S and learning task $\mathcal{T}_S$, a target domain D_T and learning task $\mathcal{T}_T$, transfer learning aims to help improving the learning of the target prediction function $F^T_{Y|X}(\cdot|\cdot)$ in $\mathcal{T}_T$ using the knowledge in D_S and $\mathcal{T}_S$, where $D_S \neq D_T$, or $\mathcal{T}_S \neq \mathcal{T}_T$.*

Concerning this definition 2 given in [35], we can make the following remarks[5]:

- A domain is a pair $D = \{\mathcal{X}, F_X(\cdot)\}$. Thus the condition $D_S \neq D_T$ implies that either the feature space changes $\mathcal{X}_S \neq \mathcal{X}_T$ or the distribution function changes $F^S_X(\cdot) \neq F^T_X(\cdot)$, or both. It should be noted that a modification in the support of the input (i.e. $\mathcal{X}_S \neq \mathcal{X}_T$) implies a modification of the distributions (i.e. $F^S_X(\cdot) \neq F^T_X(\cdot)$):

  $$\mathcal{X}_S \neq \mathcal{X}_T \Rightarrow F^S_X(\cdot) \neq F^T_X(\cdot).$$

  Therefore considering that there is transfer learning when there is a change in the distribution (i.e. $F^S_X(\cdot) \neq F^T_X(\cdot)$) automatically includes the case where the support of the variable can change.

  This distinction is not made in our definition 1. In our definition, there is transfer learning when there is a change in the distribution. Therefore our definition is compatible with definition 2 given in [35]. Note that the same remark can be made about $\mathcal{T}_S \neq \mathcal{T}_T$.

- As there are usually multiple relevant domains from where knowledge can be transferred, *multiple source transfer learning* (MSTL) has recently received much attention [19]. In definition 2 given in [35], only the case where there is one source and one target environment is taken in consideration. Therefore, this definition is not compatible with a MSTL problem. In our definition 1, we suppose to have $|S_o|$ source and $|S_l|$ target environments. Our definition is compatible with MSTL problems.

- Definition 2 given in [35] considers only the problem of estimating $F^T_{Y|X}(\cdot|\cdot)$ which excludes the other type of learning problems like unsupervised learning or reinforcement learning. Our definition includes these cases.

2.2 Main research topics in transfer learning

According to [35] (section 2.3), there are three main research topics in transfer learning: (i) what to transfer, (ii) how to transfer and (iii) when to transfer.

[5]Note: The last two points of this enumeration justify the fact to propose a new definition of transfer learning. Transfer learning covers a large number of different applications and, to the best of my knowledge, I haven't found a satisfying definition in the scientific literature able to consider all these cases.

- *What to transfer* searches what is the part of knowledge that can be transferred from the source to the target environments. The information that can be shared depends on the difference between distributions and supports. Usually we make assumptions about the type of change in the distribution and, given this type of change, some of the information may be transmitted.

  For instance, in section 3.3 and in chapters 4, 5 and 6, we will assume that the source environment contains only one element $S_s = \{\mathfrak{d}\}$ and the target contains also one element $S_t = \{\mathfrak{t}\}$. We will also assume to be in presence of *prior probability shift* meaning that the prior probability of the output changes $P(Y = \cdot) \neq P(\widetilde{Y} = \cdot)$. As we will see, in presence of prior probability shift, the within class probability density is conserved $f_{X|Y}(x|y) = f_{\widetilde{X}|\widetilde{Y}}(x|y)$ and it is this information that we will transfer between the source and the target environments.

  Note that in the simplest case when all the distributions in S_o and S_l are the same, we retrieve a traditional machine learning problem and a maximum of knowledge can be transferred from the source environment to the target environment.

- Each type of transfer has a specific method of transfer. *How to transfer* tries to identify the methods to perform this transfer of knowledge.

  For instance, in chapters 4, 5 and 6, we will propose some methods to perform the transfer of knowledge in presence of *prior probability shift* and next the section 2.3 will present other techniques to do this transfer.

- *When to transfer* aims to identify the situations when knowledge should *not* be transferred. In some situations when the source and target domains are too far, the transfer may reduce the performance of the predictive model. This situation is often referred to as *negative transfer* [19]. Negative transfer occurs when knowledge is transferred from highly irrelevant sources.

2.3 Categorization of transfer learning settings

The authors of [35] give a taxonomy of the most common transfer learning settings in the machine learning literature. This section proposes a categorization close to the one proposed in [35]: (i) inductive transfer learning, (ii) transductive transfer learning (iii) unsupervised transfer learning (iv) reinforcement transfer learning.

2.3.1 Inductive transfer learning

Inductive transfer learning concerns the supervised learning setting where output data Y from the target environment are available for the training. The most simple setting of inductive transfer learning is the following:

Let $(X_1, Y_1) \sim F_{X_1Y_1}(x, y)$ and $(X_2, Y_2) \sim F_{X_2Y_2}(x, y)$ be two input/output pairs of random variables. We suppose that (X_1, Y_1) and (X_2, Y_2) are respectively in the source and target environments and, to have a transfer learning setting, one distribution must be different from the other. Two sets of observations can be generated from these two distributions: $\mathfrak{d}_1$ and $\mathfrak{d}_2$ with $S_s = \{\mathfrak{d}_1\}$ and $S_t = \{\mathfrak{d}_2\}$. The difficulty of this inductive transfer learning is to find methods able to use the data in $\mathfrak{d}_1$ and $\mathfrak{d}_2$ to build a predictive model for the target environment.

Note that this situation is close to the *multi-task* learning setting [6] which tries to learn $F_{X_1Y_1}(x, y)$ and $F_{X_2Y_2}(x, y)$ simultaneously. Multi-task learning is in contrast with standard methodology in machine learning which consists in learning one task at a time. In multi-task learning, we have no element in the source environment (i.e. $S_s = \varnothing$) and both sets of observations are in the target environment (i.e. $S_t = \{\mathfrak{d}_1, \mathfrak{d}_2\}$). Following definition 1, multi-task learning is a kind of transfer learning problem because $\mathfrak{d}_1$ and $\mathfrak{d}_2$ are generated from two distinct unknown distributions and we try to build simultaneously two models from the two datasets by using all the information available in $\{\mathfrak{d}_1, \mathfrak{d}_2\}$.

2.3.2 Transductive transfer learning

Transductive transfer learning is a situation where *no* labeled data are available in the target situation. The most simple setting of transductive transfer learning is the following with two distributions.

Let $F_{XY}(x, y)$ be a distribution in the source context and let $F_{\widetilde{X}\widetilde{Y}}(x, y)$ be a distribution in the target context where $\widetilde{Y}$ is a latent variable meaning that no observations can be extracted from it. To have a transfer learning setting, $F_{XY}(x, y)$ must be different from $F_{\widetilde{X}\widetilde{Y}}(x, y)$. From the two previous distributions, we can generate the two following sets of observations: $\mathfrak{d}$ and $\mathfrak{t}$. Set $\mathfrak{d}$ is the training dataset from which $F_{Y|X}(y|x)$ is estimated and $\mathfrak{t}$ is the testing dataset from which the estimation of $F_{Y|X}(y|x)$ is adjusted to fit $F_{\widetilde{Y}|\widetilde{X}}(y|x)$. As the testing set is known during the learning step, we are in a transductive learning context.

As no output observations are available in the target environment and as the two distributions $F_{XY}(x, y)$ and $F_{\widetilde{X}\widetilde{Y}}(x, y)$ can be arbitrarily far from each other, in general, the estimation of $F_{\widetilde{Y}|\widetilde{X}}(y|x)$ in the target environment is unsolvable. Hence, it is needed to make simplifying assumptions about the type of drift in the probability distributions. In [40], a number of common forms of dataset shift are introduced. Each form has its own assumption about the type of shift. This section will list these different forms.

We suppose to have two random variables: an input variable $X \in \mathbb{R}^d$ and a discrete output variable Y taking values in $\{\omega_0, \omega_1, \omega_2, \ldots, \omega_{|\mathcal{Y}|-1}\}$. In dataset shift context, we assume a change between the source and target situation. To make the distinction between both situations, we assume that $\widetilde{X}$ and $\widetilde{Y}$ are respectively the input and output random variables in the targeting situation where $\widetilde{Y}$ is latent.

Covariate Shift : This situation happens when the training data are generated from $F_{XY}(x, y)$ and the testing data are coming from $F_{\widetilde{X}\widetilde{Y}}(x, y)$ and only the marginal distribution of the input variable changes (i.e. $f_X(\cdot) \neq f_{\widetilde{X}}(\cdot)$). In covariate shift, it is assumed that the condition probability between the input and the output variables is conserved (i.e., $P(Y = \cdot|X = \cdot) = P(\widetilde{Y} = \cdot|\widetilde{X} = \cdot)$). As the join distribution is function of the distribution of the input variable, it is not surprising that a change in the input variable can have an impact on the join distribution.

Example: Assume the problem where the risk of a future disease (Y) must be predicted based on current habits (X). Of course, there can be a causal effect of the current habits on the risk of having a given disease in the future (for example smoking habit and lung cancer in the future). Suppose now that changing circumstances (e.g. public smoking ban) affect the habits (the distribution of X becomes different from $\widetilde{X}$). The cause of the disease is stable but the prevalence of the disease changes over time. How does this affect the predictive model $P(\widetilde{Y} = y|\widetilde{X} = x)$?

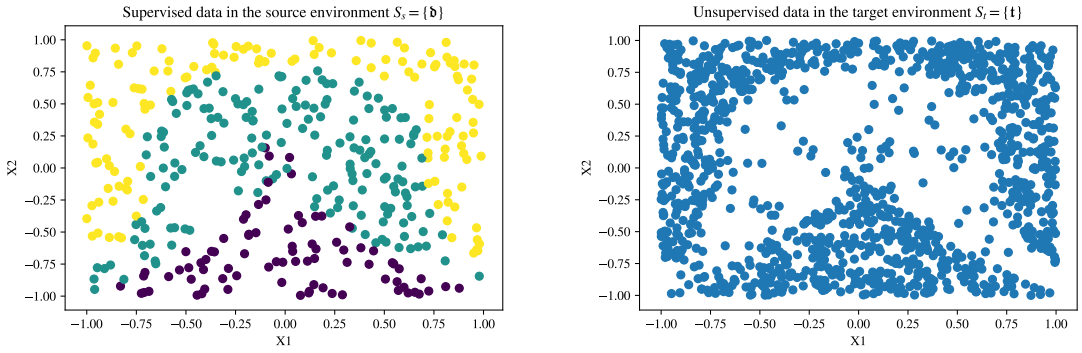

Figure 2.1: Left: Supervised training data before prior probability shift. Right: Unsupervised testing data after prior probability shift.

As, by definition, in a covariate shift only the input distribution changes ($f_X(\cdot) \neq f_{\widetilde{X}}(\cdot)$) and not the conditional one ($F_{Y|X}(\cdot|\cdot) = F_{\widetilde{Y}|\widetilde{X}}(\cdot|\cdot)$), covariate shift should have no effect on the predictive model. The knowledge about $F_{Y|X}(y|x)$ in the source environment can be directly transferred to the target environment. But there are a number of contributions like [39] that tend to show the potential computational advantages of adjusting for covariate shift. One of the advantage is that rather than worry about the overall fit of the model, the idea in [39] is to use weightings and simpler model classes to focus only on a local region relevant to the test cases.

Prior Probability Shift : This occurs when the output marginal distribution changes between the training and the testing (i.e. $P(Y = \cdot) \neq P(\widetilde{Y} = \cdot)$). Note that in prior probability shift, it is also assumed that the within class distribution of the input variable is not changing (i.e. $f_{X|Y}(\cdot|\cdot) = f_{\widetilde{X}|\widetilde{Y}}(\cdot|\cdot)$). Although the within class probability density is conserved, a change in the marginal distribution of the output variable can implied a change in the conditional probability to observe the output (i.e. $P(Y = \cdot|X = \cdot) \neq P(\widetilde{Y} = \cdot|\widetilde{X} = \cdot)$). As in classification, it is the conditional distribution of the output that we try to predict, it is not surprising that prior probability shift can have an effect on the predictive models.

In prior probability shift, two cases can be considered: either the new output distribution is known or not.

Example 1: Prior probability shift occurs for example in medical trials with control cases. If 50% of the individuals are in the control group and the other 50% are suffering a disease, then once the models are created and applied in real world situations, we have bad probability predictions because we have no idea about the true prior probability. This is an example where the new output distribution is unknown.

Example 2: When the dataset is highly unbalanced, it is common to reduce at random observations from the majority class [9]. By doing that, we introduce prior probability shift by design and it is an example where the shift in the a priori is known. Of course, in this second example, when the probability shift is known, doing quantification learning makes no sense.

Figure 2.1 shows a synthetic example of a prior probability shift problem. The left part

shows the supervised data observations available in the source environment and the right part of the figure shows the unsupervised data from the target environment after prior probability shift[6]. The purpose of the learning task can either be to estimate the new prior probability of the output variable or to recalibrate for the target environment a classifier learned on the supervised data available in the source environment.

For a *known* shift in $F_Y(\cdot)$ to $F_{\widetilde{Y}}(\cdot)$, the prior probability shift is easy to adjust for:

- As it is assumed that $f_{X|Y}(\cdot|\cdot) = f_{\widetilde{X}|\widetilde{Y}}(\cdot|\cdot)$, a generative model (like naive Bayes) can directly use the training set $\mathfrak{d}$ to estimate $f_{\widetilde{X}|\widetilde{Y}}(\cdot|\cdot)$.
- The new a priori $P(\widetilde{Y} = \cdot)$ is assumed to be known.
- We can estimate the conditional distribution of the output from $P(\widetilde{Y} = y|\widetilde{X} = x) \propto f_{\widetilde{X}|\widetilde{Y}}(x|y) \times P(\widetilde{Y} = y)$ where the normalization term $f_{\widetilde{X}}(x)$ can be obtained from the law of total probability (i.e. $f_{\widetilde{X}}(x) = \sum_y f_{\widetilde{X}|\widetilde{Y}}(x|y) \times P(\widetilde{Y} = y)$) or can be estimated from $\mathfrak{t}$.

In the case of non-generative models, a more generic method will be presented in section 3.3 at page 26.

When the shift in the output is *unknown*, making a prediction

$$P(\widetilde{Y} = y|\widetilde{X} = x) = \frac{f_{\widetilde{X}|\widetilde{Y}}(x|y)\ P(\widetilde{Y} = y)}{f_{\widetilde{X}}(x)},$$

is not possible. The distribution $f_{\widetilde{X}|\widetilde{Y}}(x|y)$ can be estimated from $\mathfrak{d}$ and $f_{\widetilde{X}}(x)$ can be estimated from $\mathfrak{t}$ but, to be able to estimate $P(\widetilde{Y} = y|\widetilde{X} = x)$, we need also an estimation of $P(\widetilde{Y} = y)$ (i.e. quantification).

To solve this problem, in [40], the authors propose to use generative classes of models to estimate $f_{\widetilde{X}|\widetilde{Y}}(x|y)$ from $\mathfrak{d}$ and to use Bayesian statistic to infer an a posteriori distribution of $\widetilde{Y}$ from an a priori distribution of $\widetilde{Y}$.

- An a priori distribution over $\widetilde{Y}$ is first specified by supposing that the distribution of $\widetilde{Y}$ is parameterized by a parameter θ. The a priori distribution over $\widetilde{Y}$ is then defined through an a priori distribution over Θ

  $$P(\widetilde{Y} = y) = \int P(\widetilde{Y} = y|\Theta = \theta)\ f_\Theta(\theta)\ d\theta.$$

  Note that the form of the parametric distribution and the a priori $f_\Theta(\cdot)$ must both be provided by the analyst.
- An a posteriori distribution over Θ is then computed from the unsupervised data in $\mathfrak{t}$ (see equation (13) in [40])

  $$f_\Theta(\theta|\mathfrak{t}) = \prod_j \sum_y f_{\widetilde{X}|\widetilde{Y}}(x_j|y)\ P(\widetilde{Y} = y|\Theta = \theta)\ f_\Theta(\theta)$$

  where $f_{\widetilde{X}|\widetilde{Y}}(x|y)$ can be estimated from $\mathfrak{d}$, where j counts over the testing data $\mathfrak{t}$, and where y goes through the values that the output can take.

[6] As we can guess, the proportion of points of the green category has decreased in the target environment.

- Then, the prediction taking into account the uncertainty of the parameters and the observed testing data is

$$P(\widetilde{Y} = y|\widetilde{X} = x, \mathfrak{t}) = \frac{f_{\widetilde{X}|\widetilde{Y}}(x|y) \int P(\widetilde{Y} = y|\Theta = \theta)\ f_{\Theta}(\theta|\mathfrak{t})d\theta}{f_{\widetilde{X}}(x)}.$$

It is prior probability shift that we will consider in the next chapters. The chapters 4, 5 and 6 will study other methods to manage prior probability shift when the new prior is unknown.

Sample Selection Bias : This problem occurs when learning data points do not accurately represent the distribution of the test case due to a selection process. This term includes all the biases that can lead to the fact that the training observations do not constitute a representative group of the population in the testing set.

Let v be a *binary selection* variable that decides if a data must be stored in the training set ($v = 1$) or not ($v = 0$). Let V be the associated Bernoulli random variable. In a sample selection bias, it is commonly assumed that the value taken by the *binary selection* variable depends on both the input and the output variable: $P(V = v|X = x, Y = y)$. It is important to depend on the two variables because we can see that the *sample selection bias* becomes a *covariate shift* if the binary selection variable is independent of the output (i.e. $P(V = v|X = x)$) or becomes a *prior probability shift* if the binary selection variable is independent of the input (i.e. $P(V = v|Y = y)$).

Domain Shift : This happens when there is a change in the measurement. In this case, we assume that X_0 is a latent unobservable random variable and the output variable Y is dependent on this latent X_0. The difficulty is that we only observe some map $X = M(X_0)$ into the observable space and $M(\cdot)$ can be different between the training and the testing. The issue comes from the fact that the map function is unknown.

Example: Such a latent variable X_0 could, for example, be a price of a product in dollar adjusted to a fixed euro / dollar index taken on a given date. The output variable Y is dependent on this latent variable X_0. The difficulty is that we never observe X_0. We only observe some map function $X = M(X_0)$ and that map can change between source and testing environment.

Source Component Shift : Source component shift can be the most common form of dataset shift. It indicates that the data come from a number of different sources, each with their own distribution and the proportions of these sources may vary between training and test scenarios.

Example: This occurs for example in production factories where a particular product can be made in many factories. The issue comes from the fact the proportion of product sourced by each factory can change between the source environment and the target environment. And the new proportions in the target environment are unknown to the analyst, (s)he only observes the products. Without knowledge about these proportions in the the target environment, the analyst has to transfer knowledge from the source environment to the target environment in order to estimate characteristics on the new products.

In source component shift, different data sources S are represented in the dataset and each data source has its own distribution $F_{XY|S}(x, y|s)$. The data source S is a latent variable from which no information is available and, as we are in a context of dataset shift, the distribution of S can change between the training and the testing environment.

2.3.3 Unsupervised transfer learning

Unsupervised transfer learning, also called *Self-taught clustering*, concerns situations where there are no labeled data available in both the source and the target environments [7]. It aims at clustering a small collection of target unlabeled data with the help of a large amount of non-labeled auxiliary data. The target and the auxiliary observations may be generated from different probability distributions.

Self-taught clustering uses the data from the source and the target environments to build a new representation of the input features. The rationale is that a good data representation can make the clustering task much easier. Once this new representation is built, the target observations are projected in this new space and clustering is performed in this space.

2.3.4 Reinforcement transfer learning

Reinforcement learning [41] refers to a class of machine learning problems where the purpose is to learn a policy from experiments. The policy says what to do in different situations in order to optimize a quantitative reward over time. Scientific works investigate the relation between transfer learning in reinforcement learning [43]. In a reinforcement learning context, the positive effects of transfer learning may help the agent to learn a new (but related) task more quickly. The idea is that generalization may occur not only within tasks, but also across reinforcement learning tasks. The experience gained in learning a policy in one task can help to improve the learning of a policy in a related but different task.

Chapter 3

Prerequisites

3.1 Formalization of the learning problem

This section aims to define the notations of the main machine learning concepts [3, 14, 23, 33, 44, 47] that we will use during the rest of this work. The formalization that we will introduce in this section is inspired by [4] who is itself reusing some notations from the work of [45]. Note that some notions have already been introduced previously in chapters 1 and 2. These notions will be reintroduced in this chapter into a machine learning perspective.

In this work, we consider the supervised setting. Let $\mathfrak{d} = \{(x_i, y_i)\}_{i=1}^{N_\mathfrak{d}} = \{z_i\}_{i=1}^{N_\mathfrak{d}}$ be an input/output training dataset with $N_\mathfrak{d}$ independent observations obtained from the same fixed but unknown joint distribution $Z = (X, Y) \sim F_Z(z) = F_{XY}(x, y) = F_{Y|X}(y|x)F_X(x)$ where $X \in \mathcal{X} \subset \mathbb{R}^d$, $Y \in \mathcal{Y}$ and $Z \in \mathcal{Z} = \mathcal{X} \times \mathcal{Y}$. The set $\mathfrak{d}$ is a realization of the random variable $\mathcal{D} \sim \prod_{i=1}^{N_\mathfrak{d}} F_Z(z_i) = F_\mathcal{D}(\mathfrak{d})$. For instance, if $N_\mathfrak{d} = 2$ then $\mathfrak{d} = (z_1, z_2)$ and $\mathcal{D} \sim F_{ZZ}(z_1, z_2) = F_Z(z_1)\ F_Z(z_2) = \prod_{i=1}^{2} F_Z(z_i)$.

In this work, we will consider the case $\mathcal{Y} \subseteq \mathbb{R}$ (i.e. regression) and the case $\mathcal{Y} = \{\omega_0, \omega_1, \omega_2, \ldots, \omega_{|\mathcal{Y}|-1}\}$ (i.e. classification). Note that the previous formalization is compatible with the classical definition of the regression problem

$$Y \mid x = m(x) + \epsilon, \text{ with } \epsilon \sim F_\epsilon(e) \text{ and } E(\epsilon) = 0 \tag{3.1}$$

where $E[Y|x] = m(x) \in \mathcal{Y} \subseteq \mathbb{R}$ is the unknown *target function* for which $h(x, \alpha_s)$ with $\alpha_s \in \Lambda_s$ is an estimation[1]. It is common to do the following *normal* assumption about the noise:

$$Y \mid x = m(x) + \epsilon, \text{ with } \epsilon \sim N(0, \sigma_\epsilon^2) \ \Leftrightarrow \ f_{Y|X}(y|x) = N(m(x), \sigma_\epsilon^2). \tag{3.2}$$

In a classification context, *Bayes' decision rule*

$$\arg\max_{\omega \in \mathcal{Y}} P(Y = \omega | X = x) \tag{3.3}$$

plays the same role as the target function $m(\cdot)$ in regression. This is an unknown function that assigns to each input $x \in \mathcal{X}$ a label from $\mathcal{Y}$ in a way that minimizes the average classification error.

Let $\mathfrak{t} = \{x_j^\mathfrak{t}\}_{j=1}^{N_\mathfrak{t}}$ be a testing set with $N_\mathfrak{t}$ independent input samples[2] following the unknown distribution $F_{\widetilde{X}\widetilde{Y}}(x, y)$ where $\widetilde{Y}$ is a latent variable meaning that no observations can be generated from it. The set $\mathfrak{t}$ is a realization of the random variable $\mathcal{T} \sim \prod_{j=1}^{N_\mathfrak{t}} F_{\widetilde{X}}(x_j^\mathfrak{t})$.

[1] The $h(x, \alpha_s)$ function will be defined later.

[2] The *exponent* $\mathfrak{t}$ in $x_j^\mathfrak{t}$ stands for $x_j^\mathfrak{t} \in \mathfrak{t}$.

Note that the basic assumption in machine learning is that the training and testing sets are drawn from the same distribution. But, as in this work we assume the possibility that there is a shift in the dataset, to be able to distinguish both situations, we put a *tilde* on the random variables coming from the target environment.

Following the definition 1 given at page 11:

- We have $|S_l| = 1$ distribution where the output random variable $\widetilde{Y}$ is latent and $|S_o| = 1$ distribution where all the variables are observable.
- We make the assumption that the distribution in the source and the target environments are different.
- Both the source and the target environments contain only one element $S_s = \{\mathfrak{d}\}$ and $S_t = \{\mathfrak{t}\}$.
- As no output observations are available in the target environments, it is a transductive transfer learning problem (see page 14).

Of course, if we assume that there is no shift in the distributions then

$$\forall x \in \mathcal{X}, \forall y \in \mathcal{Y} : F_{X,Y}(x,y) = F_{\widetilde{X},\widetilde{Y}}(x,y)$$

and the learning problem becomes a traditional machine learning problem without need for transfer learning.

The goal of our learning problem is to use $\mathfrak{d}$ to learn the parameters α of a *predictive model* (also called *hypotheses*) $h : \mathcal{X} \times \Lambda \to \mathcal{Y}$ where $\alpha \in \Lambda$ and transfer learning is needed to recalibrate the model to the target distribution $F_{\widetilde{X}\widetilde{Y}}(x,y)$.

The set Λ contains different possible values of the parameters and it is common to group these possible values in families or in *types of models*. Examples of families of models are linear models $\alpha_0 + \sum_{i=1}^{d} \alpha_i x_i$ where, in this case, $\alpha \in \mathbb{R}^{d+1}$ represents the coefficients of a given linear model and Λ is a subset[3] of $\mathbb{R}^{d+1}$. A feed-forward neural network is another example where α is the set of values taken by the synaptic weights.

A classifier $h : \mathcal{X} \times \Lambda \to \mathcal{Y} = \{\omega_0, \omega_1, \omega_2, \ldots, \omega_{|\mathcal{Y}|-1}\}$ with $\alpha \in \Lambda$ is a function that returns for each input x a prediction $\widehat{y} = h(x,\alpha) \in \mathcal{Y}$ in the discrete set of output values. This definition includes the *class probability estimation* (CPE) models [25] where

$$h(x,\alpha) = \arg\max_{\omega \in \mathcal{Y}} g_\omega(x,\alpha)$$

with $g_\omega : \mathcal{X} \times \Lambda \to [0,1] \subset \mathbb{R}$ and with $\sum_{\omega \in \mathcal{Y}} g_\omega(x,\alpha) = 1$. In the case of CPE models, the value of $g_\omega(x,\alpha)$ can be interpreted as an estimation of the a posteriori probability to observe any output value ω in $\mathcal{Y}$, for a given input x:

$$\begin{aligned} g_y(x,\alpha) &= \widehat{P}(Y = y | X = x, \alpha) \\ &\approx P(Y = y | X = x, \alpha). \end{aligned} \tag{3.4}$$

After learning, the predictive model can return predictions $\widehat{y} = h(x,\alpha) \in \mathcal{Y}$. Given a *loss* function $L(y, h(x,\alpha))$, in the classical *inductive* learning context, the *functional risk* R

[3] *Lasso* [25] and *ridge* [25] are two classical methods to reduce the size of Λ for linear models.

measures the capacity of a model to return good predictions for any new input value. It is defined as the expected value of the loss where $\widetilde{X}$ and $\widetilde{Y}$ are the random variables:

$$\begin{aligned} R(\alpha) &= E\left[L(\widetilde{Y}, h(\widetilde{X}, \alpha))\right] \\ &= \int_{\mathcal{X},\mathcal{Y}} L(y, h(x, \alpha))\ dF_{\widetilde{X}\widetilde{Y}}(x, y). \end{aligned} \tag{3.5}$$

In this work, *regression* and *classification* are the two main learning problems that we consider. In regression, the support of the output is in the set of real values and it is common to use the *mean squared error* (MSE)

$$L(y, h(x, \alpha)) = (y - h(x, \alpha))^2$$

as loss function. Note that, in regression, we can see that if the function h fits perfectly the target function (i.e. $\forall x \in \mathcal{X},\ m(x) = h(x, \alpha)$) then the functional risk R is equal to the variance of the noise σ^2.

$$\begin{aligned} R(\alpha) &= E\left[L(\widetilde{Y}, h(\widetilde{X}, \alpha))\right] \\ &= E\left[(\widetilde{Y} - h(\widetilde{X}, \alpha))^2\right] \\ &= E\left[(m(\widetilde{X}) + \tilde{\epsilon} - h(\widetilde{X}, \alpha))^2\right] \\ &= E\left[(m(\widetilde{X}) + \tilde{\epsilon} - m(\widetilde{X}))^2\right] \\ &= E\left[\tilde{\epsilon}^2\right] \\ &= \tilde{\sigma}^2 \end{aligned}$$

In classification, the output takes its values in a discrete set $\{\omega_0, \omega_1, \ldots, \omega_{|\mathcal{Y}|-1}\}$. The confusion matrix [11, 36] is typically used to evaluate or to visualize the behavior of models in supervised classification contexts [25]. It is a square matrix in which the rows represent the actual class of the instances and the columns their predicted class. If we are handling a binary classification task, then the confusion matrix [5] is a 2×2 matrix that reports the number of *true positives* ($\#TP$), *true negatives* ($\#TN$), *false positives* ($\#FP$), and *false negatives* ($\#FN$) as follows:

$$\begin{bmatrix} \#TP & \#FN \\ \#FP & \#TN \end{bmatrix}.$$

This matrix contains all the raw information about the predictions provided by a classification model on a given dataset. To evaluate the generalization accuracy of a model, it is common to use a *testing dataset* which was not used during the learning process of said model. Many synthetic one-dimensional performance indicators can be extracted from a confusion matrix. The performance indicator can be, for example, the precision, the recall, the F-score ... When different kinds of errors are not assumed to be equal, in association with a 2×2 cost matrix, cost-sensitive performance indicators [15] can also be computed from the confusion matrix. The choice of the suitable performance indicator is directly linked to the objective of the learning problem. If the classical *accuracy* measure is taken into account, then

$$L(y, h(x, \alpha)) = \begin{cases} 0, & \text{if } y = h(x, \alpha) \\ 1, & \text{otherwise} \end{cases}.$$

3.1.1 Learning as a nested optimization problem

Of course, as $F_{\widetilde{X}\widetilde{Y}}(x, y)$ is unknown, the functional risk R is not directly available and must be estimated. In this context, the objective of classical inductive learning is to find the optimal parameters in Λ that minimize the unknown functional risk $R(\cdot)$ when the input/output samples in $\mathfrak{d}$ are the only available information. We propose to follow a common procedure to find these optimal parameters [4]. It is implemented at two nested levels: the *structural identification* (i.e. search of the optimal hyper-parameters) and the *parametric identification* (i.e. given hyper-parameters, search of the optimal parameters).

The *parametric identification* algorithm $\mathcal{I}^P$ solves the inner level of the optimization problem. It searches, in a given set of parameters Λ, the optimal function $h(\cdot, \widehat{\alpha})$ minimizing an *empirical risk* R_{emp} computed on the training set $\mathfrak{d}$:

$$\widehat{\alpha} = \arg\min_{\alpha \in \Lambda} R_{emp}(\alpha) \tag{3.6}$$

where

$$R_{emp}(\alpha) = N_{\mathfrak{d}}^{-1} \sum_{i=1}^{N_{\mathfrak{d}}} L(y_i, h(x_i, \alpha)).$$

Note that equation (3.6) minimizes the *empirical risk* R_{emp} rather than the *functional risk* R. This is called the *empirical risk minimization* (ERM) inductive principle [45]. Examples of parametric identification procedures in regression are least-squares for linear models or back-propagated gradient-descent for feed-forward neural networks. In classification, maximum likelihood is commonly used for logistic regression models.

The parametric identification can be done in different sets of parameters Λ_s with $s = 1, \ldots, S$. It is common to consider that $\{\Lambda_s\}_{s=1}^S$ contains only one type of models where s measures the complexity (i.e. the size) of Λ_s with a nested sequence of classes of models $\Lambda_1 \subset \Lambda_2 \subset \ldots \subset \Lambda_S$. For example:

- in *linear models*, s could be the degree of the polynomial;
- in *neural networks*, s could be the number of hidden nodes;
- in *trees*, s could be the maximum depth.

Overfitting occurs when $\mathcal{I}^P$ is applied to a too complex (i.e. too big) set of models Λ_s and underfitting occurs when Λ_s is too small. The *structural identification* algorithm $\mathcal{I}^S$ solves the outer level of the optimization problem and ranges over different classes of models $\{\Lambda_s\}_{s=1}^S$ to select the best one.

Let $\widehat{\alpha}_s$ be the optimal vector of parameters obtained by ERM on the set Λ_s. The quantity $G(\Lambda_s)$ measures the average generalization error of the function $h(\cdot, \widehat{\alpha}_s)$ learned from a given Λ_s and from any possible training set $\mathfrak{d}$ following the distribution of $\mathcal{D} \sim F_{\mathcal{D}}(\mathfrak{d})$:

$$\begin{aligned} G(\Lambda_s) &= E\left[R(\widehat{\alpha}_s)\right] \\ &= E\left[R(\mathcal{I}^P(\Lambda_s, \mathcal{D}))\right] \\ &= \int_{\mathcal{Z}^{N_{\mathfrak{d}}}} R(\mathcal{I}^P(\Lambda_s, \mathfrak{d}))\, dF_{\mathcal{D}}(\mathfrak{d}). \end{aligned} \tag{3.7}$$

In equation (3.7), $\mathcal{D}$ is the random variable on which the expectation is computed (see figure 3.1). The structural identification algorithm $\mathcal{I}^S$ should select the set minimizing the

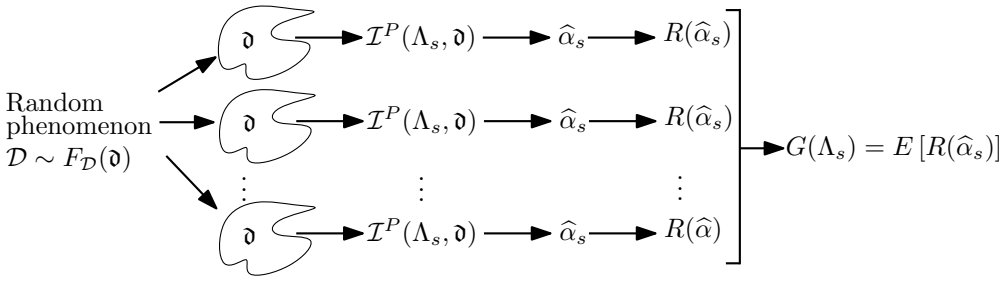

Figure 3.1: G is the mean of R computed on $h(\cdot, \widehat{\alpha})$ when an infinite number of training datasets is available.

Algorithm 1 K-fold cross validation

```
1:  function I_cv^S({Λ_s}_{s=1}^S, 𝔡, K)
2:      size_block ← ⌊N_𝔡 / K⌋
3:      err ← (0, 0, ..., 0)                          ▷ Vector of S elements
4:      for s ← 1 to S do
5:          for k ← 1 to K do
6:              i_1 ← (k − 1) × size_block + 1
7:              i_2 ← k × size_block
8:              𝔱̃ ← 𝔡[i_1, i_2]                       ▷ Take samples from index i_1 to index i_2
9:              𝔡̃ ← 𝔡 \ 𝔱̃                            ▷ Where "\" is the set minus operator
10:             α̂_s ← I^P(Λ_s, 𝔡̃)
11:             {ŷ_j}_{j=1}^{N_𝔱} ← h(x_j, α̂_s), for all x_j ∈ 𝔱̃
12:             err[s] ← err[s] + Σ_{j=1}^{N_𝔱} L(y_j, ŷ_j)
13:         end for
14:     end for
15:     ŝ^cv ← arg min_{s∈{1,...,S}} err[s]
16:     return ŝ^cv
17: end function
```

generalization error G but, as this quantity is not directly available, G must be estimated. In an inductive learning context, *cross validation* (see algorithm 1) is a well-known method, among many others, to estimate this generalization error.

The cross validation function $\mathcal{I}_{cv}^S$ has three input parameters: (i) a set of S classes of models, (ii) a training dataset $\mathfrak{d}$ and (iii) the number of sub samples K. At line 3, a vector of size S is created. It will contain the cross validation error estimates and it will be used at line 15 to select the best class of models. The $\mathcal{I}_{cv}^S$ function is composed of two nested loops. The outer loop (line 4 to 14) goes through all the S classes of models and will compute for each of them an estimation of $G(\cdot)$. This estimation is done by the inner loop (line 5 to 13). At line 6 to 9, the training set $\mathfrak{d}$ is split in two parts following a sliding window approach. Set $\widetilde{\mathfrak{t}}$ is a validation set containing *size_block* elements and set $\widetilde{\mathfrak{d}}$ is a new training set containing the $N_{\mathfrak{d}} - \mathit{size\_block}$ other samples. The new training set is used at line 10 to select $\widehat{\alpha}_s$ by parametric identification and the hypotheses $h(\cdot, \widehat{\alpha}_s)$ is evaluated on the validation set at line 11. At line 12, the error on the validation set is computed and added in the vector *err* at position s. After all the loops, the index of the best hyperparameter is selected at line 15 and returned at line 16.

3.2 The Expectation–Maximization (EM) algorithm

The method that we will discuss in chapter 6 uses an EM algorithm. This is the reason why we introduce this method in this section. This EM algorithm will be used in chapter 6 to improve the parametric identification procedure in a transduction learning context where the new input data are known and with presence of prior probability shift in the dataset.

We consider $(\mathcal{O}, \mathcal{L}) \sim F_{\mathcal{O}\mathcal{L}}(o, l|\theta)$, a couple of random variables following a parametric distribution where $\theta \in \Theta$ is unknown and must be estimated. The variable $\mathcal{O}$ is the *observable* random variable from which $N_{\mathfrak{o}}$ independent observed realizations are available in $\mathfrak{o} = \{o_j\}_{j=1}^{N_{\mathfrak{o}}}$. The variable $\mathcal{L}$ is called the *latent* random variable because no samples are available from it. The Expectation–Maximization (EM) algorithm [12, 32] is a common class of iterative methods in statistics to find the maximum likelihood estimator in parametric statistical models where the model depends on unobservable latent random variables.

The goal is to find

$$\widehat{\theta}_{mle} \in \arg\max_{\theta \in \Theta} \mathscr{L}(\theta, \mathfrak{o})$$

where

$$\mathscr{L}(\theta, \mathfrak{o}) = \prod_{j=1}^{N_{\mathfrak{o}}} f_{\mathcal{O}}(o_j|\theta)$$

is the likelihood and where

$$f_{\mathcal{O}}(o_j|\theta) = \int f_{\mathcal{O}\mathcal{L}}(o_j, l|\theta) dl$$

is a marginal distribution[4].

Due to the presence of the latent random variable, this maximum likelihood is difficult to maximize. The EM algorithm starts by initializing $\widehat{\theta}_1 \in \Theta$. Then for $s = 1, 2, 3, \ldots$, we have the following two steps:

$$\begin{array}{ll} \text{E-step:} & Q(\theta|\widehat{\theta}_s) = E_{\mathcal{L}|\widehat{\theta}_s, \mathfrak{o}}\left[\ell(\theta, (\mathfrak{o}, \mathcal{L}))\right] \\ \text{M-step:} & \widehat{\theta}_{s+1} = \arg\max_{\theta} Q(\theta|\widehat{\theta}_s) \end{array} \tag{3.8}$$

where ℓ is the log likelihood function. The E-step computes the expected value of the log likelihood ℓ with respect to the conditional distribution of the latent state $\mathcal{L}$ given $\mathfrak{o}$ and under the current estimate of the parameters $\widehat{\theta}_s$. The M-step searches the values that maximize the quantity computed at the E-step. These two steps are repeated until convergence of $\widehat{\theta}_s$. Note that it can be shown that the likelihood cannot decrease during the EM iterations ($\mathscr{L}(\widehat{\theta}_{s+1}, \mathfrak{o}) \geq \mathscr{L}(\widehat{\theta}_s, \mathfrak{o})$) but the algorithm can easily reach a local maximum and the convergence can also be slow.

We will now show that the likelihood cannot decrease during the EM iterations [29] and, for simplicity, let us consider that $N_{\mathfrak{o}} = 1$ and thus $\mathfrak{o} = \{o_1\}$. For any latent random variable $\mathcal{L}$ with distribution $f_{\mathcal{L}|\mathcal{O}}(l|o, \theta)$ where $\mathcal{O}$ is the observable random variable and where θ contains the parameters of the joint probability distribution of $(\mathcal{O}, \mathcal{L})$, we can write

$$\begin{aligned} \ell(\theta, o_1) &= \log f_{\mathcal{O}}(o_1|\theta) \\ &= \log \frac{f_{\mathcal{O}\mathcal{L}}(o_1, l|\theta)}{f_{\mathcal{L}|\mathcal{O}}(l|o_1, \theta)} \\ &= \log f_{\mathcal{O}\mathcal{L}}(o_1, l|\theta) - \log f_{\mathcal{L}|\mathcal{O}}(l|o_1, \theta) \\ &= \ell(\theta, (o_1, l)) - \log f_{\mathcal{L}|\mathcal{O}}(l|o_1, \theta) \end{aligned} \tag{3.9}$$

[4]The $\int dl$ must be replaced by a $\sum$ if $\mathcal{L}$ is a discrete random variable.

We will now take the expectation of the log likelihood function ℓ over possible values of the unknown variable $\mathcal{L}$ under the current estimation of the parameters $\widehat{\theta}_s$. To do that, we will multiply both sides of equation (3.9) by $f_{\mathcal{L}|\mathcal{O}}(l|o_1, \widehat{\theta}_s)$ and integrate over $\mathcal{L}$

$$\begin{aligned}\ell(\theta, o_1) \underbrace{\int f_{\mathcal{L}|\mathcal{O}}(l|o_1, \widehat{\theta}_s)\ dl}_{=1} &= \int f_{\mathcal{L}|\mathcal{O}}(l|o_1, \widehat{\theta}_s)\ \ell(\theta, (o_1, l))\ dl \\ &\quad - \int f_{\mathcal{L}|\mathcal{O}}(l|o_1, \widehat{\theta}_s)\ \log f_{\mathcal{L}|\mathcal{O}}(l|o_1, \theta)\ dl \\ &= E_{\mathcal{L}|\widehat{\theta}_s, o_1}\left[\ell(\theta, (o_1, \mathcal{L}))\right] \\ &\quad - \int f_{\mathcal{L}|\mathcal{O}}(l|o_1, \widehat{\theta}_s)\ \log f_{\mathcal{L}|\mathcal{O}}(l|o_1, \theta)\ dl\end{aligned}$$

and we can rewritten the previous equation as

$$\ell(\theta, o_1) = Q(\theta|\widehat{\theta}_s) + \mathcal{H}(\theta|\widehat{\theta}_s)$$

where Q equals the first expectation in the previous equation and is already defined in equation (3.8). Note that EM algorithm works by maximizing Q rather than directly improving $\mathscr{L}$. Function $\mathcal{H}$ is defined as the opposite of the integral in the previous equation.

$\ell(\theta, o_1)$ holds for any value of θ, including $\theta = \widehat{\theta}_s$ and $\theta = \widehat{\theta}_{s+1}$ where $\widehat{\theta}_{s+1}$ is defined in equation (3.8). Let us subtract $\ell(\widehat{\theta}_{s+1}, o_1)$ by $\ell(\widehat{\theta}_s, o_1)$

$$\ell(\widehat{\theta}_{s+1}, o_1) - \ell(\widehat{\theta}_s, o_1) = Q(\widehat{\theta}_{s+1}|\widehat{\theta}_s) - Q(\widehat{\theta}_s|\widehat{\theta}_s) + \mathcal{H}(\widehat{\theta}_{s+1}|\widehat{\theta}_s) - \mathcal{H}(\widehat{\theta}_s|\widehat{\theta}_s).$$

The *Gibbs' inequality* [31] tells us that if $\{p_1, \ldots, p_n\}$ is a probability distribution then for any other probability distribution $\{q_1, \ldots, q_n\}$ we have

$$-\sum_{i=1}^{n} p_i \log p_i \leq -\sum_{i=1}^{n} p_i \log q_i$$

with equality if and only if $p_i = q_i$ for all i. By *Gibbs' inequality*, we know that $\mathcal{H}(\theta_{s+1}|\widehat{\theta}_s) \geq \mathcal{H}(\widehat{\theta}_s|\widehat{\theta}_s)$ and therefore

$$\ell(\widehat{\theta}_{s+1}, o_1) - \ell(\widehat{\theta}_s, o_1) \geq Q(\widehat{\theta}_{s+1}|\widehat{\theta}_s) - Q(\widehat{\theta}_s|\widehat{\theta}_s).$$

Furthermore, from equation (3.8), we have $Q(\widehat{\theta}_{s+1}|\widehat{\theta}_s) = \max_\theta Q(\theta|\widehat{\theta}_s)$ and so

$$Q(\widehat{\theta}_{s+1}|\widehat{\theta}_s) - Q(\widehat{\theta}_s|\widehat{\theta}_s) \geq 0$$

then

$$\ell(\widehat{\theta}_{s+1}, o_1) - \ell(\widehat{\theta}_s, o_1) \geq 0.$$

The likelihood can therefore not decrease during the EM iterations.

3.2.1 The EM algorithm as a maximization-maximization procedure

In this section, we will present another way [34] to find $\widehat{\theta}_{mle}$ in case of incomplete information. Let $q(l) \in [0, 1]$ be an arbitrary density function defined on the support of the latent variable $\mathcal{L}$. If we multiply both sides of equation (3.9) by $q(l)$ and then integrate over l, we obtain

$$\ell(\theta, o_1) \underbrace{\int q(l)\ dl}_{=1} = \int q(l)\ \log f_{\mathcal{O}\mathcal{L}}(o_1, l|\theta)\ dl - \int q(l)\ \log f_{\mathcal{L}|\mathcal{O}}(l|o_1, \theta)\ dl.$$

If we add and remove $\log q(l)$ on the right hand side of the previous equation, we can write

$$\begin{aligned}\ell(\theta, o_1) &= \int q(l)\ \log \frac{f_{\mathcal{OL}}(o_1, l|\theta)}{q(l)}\ dl - \int q(l)\ \log \frac{f_{\mathcal{L}|\mathcal{O}}(l|o_1, \theta)}{q(l)}\ dl \\ &= \mathcal{F}(q(\cdot), \theta) + KL(q(\cdot)||f_{\mathcal{L}|\mathcal{O}}(\cdot|o_1, \theta))\end{aligned} \tag{3.10}$$

where KL is the Kullback–Leibler divergence.

Based on the equation (3.10), the EM algorithm can now be seen as a maximization-maximization procedure [34]. After initialization $\widehat{\theta}_1 \in \Theta$ and for $s = 1, 2, 3, \ldots$, we have

$$\begin{aligned}q \text{ maximization:} &\quad \widehat{q} = \arg\max_q \mathcal{F}(q(\cdot), \widehat{\theta}_s) \\ \theta \text{ maximization:} &\quad \widehat{\theta}_{s+1} = \arg\max_\theta \mathcal{F}(\widehat{q}(\cdot), \theta)\end{aligned}$$

The rationale of this procedure is the following. A maximum likelihood estimator $\widehat{\theta}_{mle}$ is the value in Θ maximizing the likelihood $f_{\mathcal{O}}(o_1|\theta)$ to observe the realization o_1. In the first step, we search the density function $\widehat{q}(\cdot)$ minimizing the KL divergence with the function[5] $f_{\mathcal{L}|\mathcal{O}}(\cdot|o_1, \widehat{\theta}_s)$. Note that $\mathcal{F}(q(\cdot), \theta)$ equals $\ell(\theta, o_1) - KL(q(\cdot)||f_{\mathcal{L}|\mathcal{O}}(\cdot|o_1, \theta))$ where the log likelihood ℓ is independent of q and so, maximizing $\mathcal{F}$ according to q is equivalent to minimizing the KL divergence. After the first step, we can now assume that $\ell(\theta, o_1) \approx \mathcal{F}(q(\cdot), \theta)$ and therefore ℓ is indirectly maximized by maximizing $\mathcal{F}$. This procedure is repeated until convergence.

3.3 Adjusting the output of a classifier to a new known a priori

Let $\mathfrak{d} = \{(x_i, y_i)\}_{i=1}^{N_\mathfrak{d}}$ be a training dataset with iid observations obtained from $(X, Y) \sim F_Z(z) = F_{XY}(x, y)$ where $Y \in \mathcal{Y} = \{\omega_0, \omega_1, \omega_2, \ldots, \omega_{|\mathcal{Y}|-1}\}$.

It sometimes happens that a classifier model is learned on a set $\mathfrak{d}$ that does not reflect the same a priori of the target classes in the testing set $\mathfrak{t}$. This mainly comes from shift in the data distribution where a bias is introduced during the selection of individuals. It happens for instance in case control studies, during automatic labeling of geographical maps based on remote sensing information [27] or when the dataset is originally highly unbalanced and rebalancing methods are used [9].

Many types of shift in the data are possible (section 2.3). In this section, we suppose to be in presence of prior probability shift where the new marginal probability mass function of the output variable (i.e. prior) is known. Note that, in chapters 4, 5 and 6, we will suppose that the new prior is unknown.

When the shift in the data distribution is introduced, we will assume that the prior probability is changed $P(Y = y) \neq P(\widetilde{Y} = y)$ but the within-class probability density is conserved $f_{X|Y}(x|y) = f_{\widetilde{X}|\widetilde{Y}}(x|y)$. By doing these assumptions, only the number of observations from each class has changed but not the input distribution within each class. This is a reasonable assumption when the sampling bias is introduced by stratifying the output variable.

Two learning problems are consider is this work: quantification and classification.

[5] Note that $f_{\mathcal{L}|\mathcal{O}}(\cdot|o_1, \widehat{\theta}_s)$ is known because the distribution is parametric and $\widehat{\theta}_s$ is assumed to be known.

In a quantification learning context, the goal is to use $\mathfrak{d}$ and $\mathfrak{t}$ to estimate the marginal probability to observe ω from $\mathcal{Y}$ in the target environment

$$\widehat{P}(\widetilde{Y} = y).$$

In a classification learning context, the goal is to use the same two datasets $\mathfrak{d}$ and $\mathfrak{t}$ to estimate the conditional probability $\widetilde{g}$ to observe any ω from $\mathcal{Y}$ in the target environment when the testing input $x^{\mathfrak{t}}$ is given

$$\widetilde{g}_y(x, \alpha) = \widehat{P}(\widetilde{Y} = y | \widetilde{X} = x^{\mathfrak{t}}). \tag{3.11}$$

Of course, when the new prior is known, it does not make sense to do quantification. That is why this this chapter only deals with the recalibration problem of the classifier in presence of prior probability shift.

Let $N_{\mathfrak{d}}^{\omega} = \sum_{i=1}^{N_{\mathfrak{d}}} I(y_i = \omega)$, with $I()$ the *indicator function*, be the number of observations in the training set $\mathfrak{d}$ for which the output value equals ω and of course, we have $\sum_{\omega \in \mathcal{Y}} N_{\mathfrak{d}}^{\omega} = N_{\mathfrak{d}}$. We propose the following estimator for $P(Y = y)$:

$$\widehat{P}(Y = y) = N_{\mathfrak{d}}^{y} / N_{\mathfrak{d}}. \tag{3.12}$$

The value $\widehat{P}(Y = \cdot)$ is an estimation of the output probability mass function in the source environment.

The Bayes' theorem tells us that [38]

$$\widehat{f}_{\widetilde{X}|\widetilde{Y}}(x|y) = \frac{\widehat{P}(\widetilde{Y} = y | \widetilde{X} = x)\ \widehat{f}_{\widetilde{X}}(x)}{\widehat{P}(\widetilde{Y} = y)}.$$

From equation (3.11) and by the fact that the a priori probability distribution of the testing set is assumed to be known, we have

$$\widehat{f}_{\widetilde{X}|\widetilde{Y}}(x|y) = \frac{\widetilde{g}_y(x)\ \widehat{f}_{\widetilde{X}}(x)}{P(\widetilde{Y} = y)}. \tag{3.13}$$

The Bayes' theorem tells us also that

$$\widehat{f}_{X|Y}(x|y) = \frac{\widehat{P}(Y = y | X = x)\ \widehat{f}_X(x)}{\widehat{P}(Y = y)}$$

and from equations (3.4) and (3.12), we have

$$\widehat{f}_{X|Y}(x|y) = \frac{g_y(x)\ \widehat{f}_X(x)}{N_{\mathfrak{d}}^{y} / N_{\mathfrak{d}}}. \tag{3.14}$$

By assuming that the within-class probability density is conserved, we can impose equality between both estimations in equations (3.13) and (3.14):

$$\begin{aligned} \frac{\widetilde{g}_y(x)\ \widehat{f}_{\widetilde{X}}(x)}{P(\widetilde{Y} = y)} &= \frac{g_y(x)\ \widehat{f}_X(x)}{N_{\mathfrak{d}}^{y} / N_{\mathfrak{d}}} \\ \Leftrightarrow\ \widetilde{g}_y(x) &= g_y(x) \frac{P(\widetilde{Y} = y)}{N_{\mathfrak{d}}^{y} / N_{\mathfrak{d}}} \frac{\widehat{f}_X(x)}{\widehat{f}_{\widetilde{X}}(x)}. \end{aligned}$$

Note that concretely it is in the previous step that the transfer of knowledge from the source environment to the target environment is performed. By assuming prior probability shift, the within-class probability density is conserved and, by imposing equality between the two within-class probabilities, we 'transfer' the within-class probability from the source environment to the target one.

Since $\sum_{y\in\mathcal{Y}} \widetilde{g}_y(x) = 1$, we have

$$\frac{\widehat{f}_X(x)}{\widehat{f}_{\widetilde{X}}(x)} \sum_{y\in\mathcal{Y}} g_y(x) \frac{P(\widetilde{Y}=y)}{N_{\mathfrak{d}}^y/N_{\mathfrak{d}}} = 1$$

$$\Leftrightarrow \quad \frac{\widehat{f}_X(x)}{\widehat{f}_{\widetilde{X}}(x)} = \left[\sum_{y\in\mathcal{Y}} g_y(x) \frac{P(\widetilde{Y}=y)}{N_{\mathfrak{d}}^y/N_{\mathfrak{d}}}\right]^{-1}$$

and so we get the following equation

$$\begin{aligned} \widetilde{g}_y(x) &= \frac{g_y(x)\frac{P(\widetilde{Y}=y)}{N_{\mathfrak{d}}^y/N_{\mathfrak{d}}}}{\sum_{y\in\mathcal{Y}} g_y(x)\frac{P(\widetilde{Y}=y)}{N_{\mathfrak{d}}^y/N_{\mathfrak{d}}}} \\ &= \frac{g_y(x)\widetilde{\rho}_y}{\sum_{y\in\mathcal{Y}} g_y(x)\widetilde{\rho}_y} \end{aligned} \tag{3.15}$$

with $\widetilde{\rho}_y = P(\widetilde{Y}=y)/\left(N_{\mathfrak{d}}^y/N_{\mathfrak{d}}\right)$ and where all the terms are known: $g_y(x)$ is the a posteriori probability of the predictive model learned with the training dataset $\mathfrak{d}$, $P(\widetilde{Y}=y)$ is the new a priori probability distribution of the testing set that is assumed to be known, $N_{\mathfrak{d}}^y$ is the number of samples in the training set where the output modality equals y and $N_{\mathfrak{d}}$ is the total number of training samples. This formula can be used in order to compute the a posteriori probability $\widetilde{g}_y(x)$ in the testing set. We observe that the correction of the a posteriori $\widetilde{g}_y(x)$ is simply the output provided by the model $g_y(x)$ weighted by the ratio $\frac{P(\widetilde{Y}=y)}{\widehat{P}(Y=y)}$ and the denominator is only there to ensure that $\sum_y \widetilde{g}_y(x) = 1$.

Example – Binary classification

As an illustration, let us consider $\mathfrak{d}$ a set containing 2000 realizations from the following simple synthetic classification problem where

$$X \sim Unif(-2, 2)$$

with

$$(Y|X=x) = \begin{cases} \omega_1, & \text{if } x+\epsilon > 0 \\ \omega_0, & \text{else} \end{cases}, \quad \text{and with } \epsilon \sim N(0, 1/4).$$

For the modeling purpose, let us consider the family of logistic models

$$g_{y=\omega_1}(x) = \frac{\exp(\alpha_0 + \alpha_1 x)}{1+\exp(\alpha_0+\alpha_1 x)}$$

where $\Lambda = \{(\alpha_0, \alpha_1)\} \in \mathbb{R}^2$ is the set of parameters.

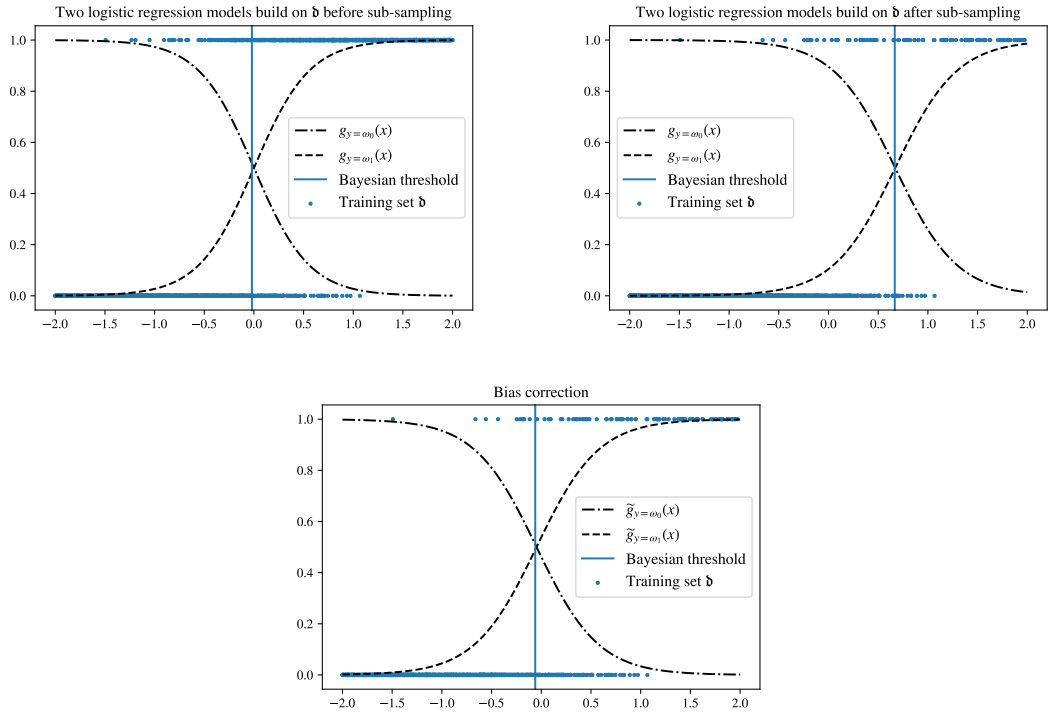

Figure 3.2: Example of an easy synthetic binary classification problem with only one input variable. Upper left: the predictive model obtained from the original training data before prior probability shift. Upper right: the predictive model obtained from the training data after prior probability shift and without recalibration. Lower: the predictive model obtained after recalibration.

If we apply the parametric identification algorithm $\mathcal{I}^P(\Lambda, \mathfrak{d})$, we obtain the predictive machine learning model shown in the upper left figure of 3.2 where the blue points are the training observations, the two black dashed lines are the model and the vertical blue lines correspond to the threshold obtained from the Bayes' decision rule (equation (3.3)). In our case, training set $\mathfrak{d}$ contains 993 observations where the target equals ω_1 and 1007 observations where the target equals ω_0. As expected from the definition of the synthetic classification problem, we can see that the threshold is close to zero.

We will now change the distribution by removing at random 90% of observations from $\mathfrak{d}$ where the target values equals ω_1. After removing these samples, the new training set contains only 99 observations with ω_1 and still 1007 observations with ω_0. After identification of the parameters with the new training set we obtain a new model shown in the upper right part of figure 3.2. As expected, the Bayesian decision threshold has moved (biased) to the right.

As we can see at the lower part of figure 3.2, if we apply equation (3.15) with $P(\widetilde{Y} = \omega_0) = 1007/2000$, $P(\widetilde{Y} = \omega_1) = 993/2000$, $N_{\mathfrak{d}}^{\omega_0} = 1007$, $N_{\mathfrak{d}}^{\omega_1} = 99$ and $N_{\mathfrak{d}} = 1106$, we correct the bias that we had introduced.

Note that the above transformation defined in equation (3.15) will not affect the ranking produced by $g_y(x)$ [10]. This is confirmed experimentally in this example because the ordering of testing samples sorted according to $g_y(x)$ and $\widetilde{g}_y(x)$ is the same in both cases.

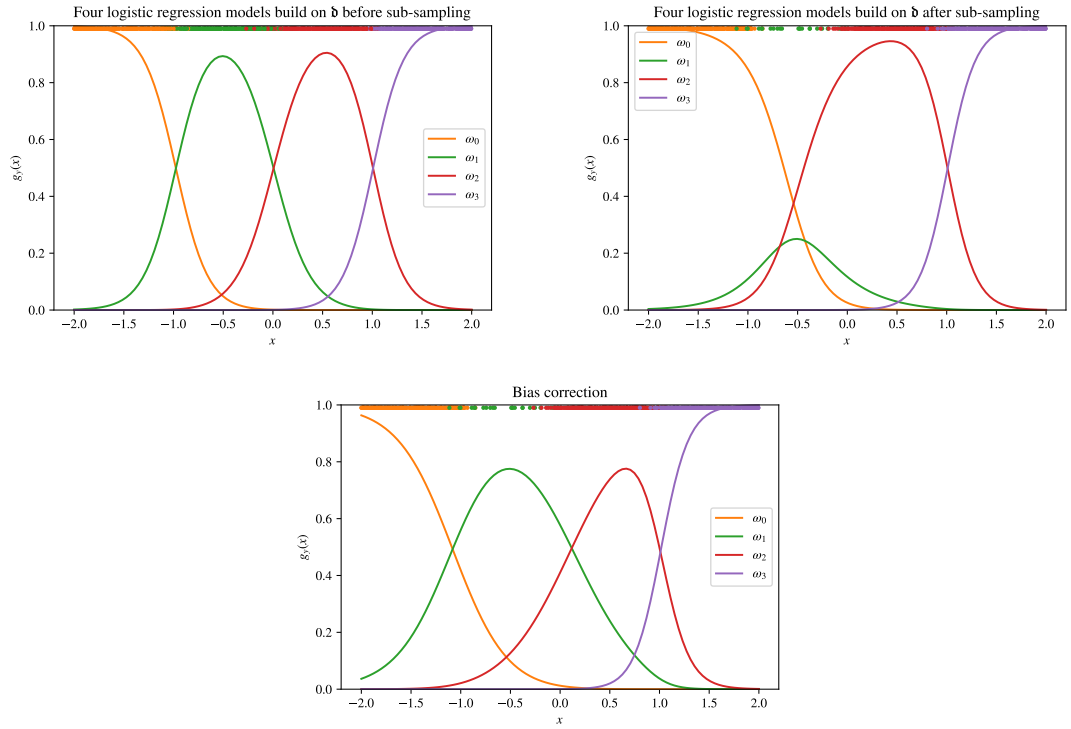

Figure 3.3: Example of an synthetic multi class classification problem with one input variable. Upper left: the predictive model obtained from the original training data before prior probability shift. Upper right: the predictive model obtained from the training data after prior probability shift and without recalibration. Lower: the predictive model obtained after recalibration.

Example – Multiclass classification

Let us now consider the following multiclass classification problem where $\mathcal{Y} = \{\omega_0, \omega_1, \omega_2, \omega_3\}$:

$$X \sim Unif(-2, 2) \tag{3.16}$$

with

$$(Y|X = x) = \begin{cases} \omega_0, & \text{if } x + \epsilon \in [-2, -1[\\ \omega_1, & \text{if } x + \epsilon \in [-1, 0[\\ \omega_2, & \text{if } x + \epsilon \in [0, 1[\\ \omega_3, & \text{if } x + \epsilon \in [1, 2] \end{cases}, \quad \text{and with } \epsilon \sim N(0, 0.01). \tag{3.17}$$

The top left of figure 3.3 shows four conditional probability distributions $\widehat{P}(Y = \cdot|X = x)$ obtained when the original training dataset from the source environment is used for parametric identification on the logistic regression models. The original training dataset contains the following numbers of observations of each class: 252 observations from ω_0, 248 observations from ω_1, 253 observations from ω_2 and 247 observations from ω_3, which gives us a total of 1000 observations in the training set.

To simulate shift in the data, the number of observations in ω_1 is reduced from 248 to 24 and the number of observations of the other output modalities remains the same. The

top right of figure 3.3 shows the four new conditional distributions. As we can see, the conditional probability to observe ω_1 (green curve) completely collapsed.

The algorithm (3.15) is then used on the four previous conditional distributions and the corrected distributions are displayed on the bottom of 3.3. If we compare this figure with the first one at the top left, we can observe that the algorithm corrected the bias rather well. The distribution in the middle has slightly inflated (i.e. larger variance) but it's remarkable to see how much it recovers the positions of the intersections between the curves. These are important information because they are used to make Bayesian decisions about the class membership of new observations.

Note that, on each figure of 3.3, the training set is displayed on the top. On the figure at the top left, it is the original set that is displayed. In the other two figures, we can see that, after subsampling, the number of observations of ω_1 (green points) is reduced.

3.3.1 Adjusting the bias term of a CPE classifier to a new known a priori

This section contains a contribution were the bias term of a CPE classifier is directly adjusted to the a new known a priori after a prior probability shift. This contribution is largely inspired a method introduced in a still unpublished paper *Adjusting the Bias Term of Classifiers to Unknown Prior* written by Prof. Marco Saerens and Prof. Christine Decaestecker.

We consider the case of adjusting the bias term of a CPE classifier with a softmax nonlinear function.

Let us assume that, after a parametric identification on Λ with $\mathfrak{d}$,

$$\begin{aligned} g_y(x) &= \widehat{P}(Y = y|X = x) \\ &= \frac{\exp\left[(\hat{\alpha}^y)^T\gamma(x) + \hat{\alpha}_0^y\right]}{\sum_{y\in\mathcal{Y}} \exp\left[(\hat{\alpha}^y)^T\gamma(x) + \hat{\alpha}_0^y\right]} \end{aligned}$$

returns an estimation of the probability that the input x belongs to class y where the vectors $(\hat{\alpha}^y)^T = (\hat{\alpha}_1^y, \hat{\alpha}_2^y, \ldots, \hat{\alpha}_{d'}^y)$ and the scalars $\hat{\alpha}_0^y$ are identified during the parametrization step and where $\hat{\alpha}_0^y$ is the bias term.

The function $\gamma : \mathbb{R}^d \to \mathbb{R}^{d'}$ is a generic function which transforms the original features. This is also identified during the learning phase. For instance, if $g_y(x)$ is estimated by a multilayer neural network perceptron then the γ transformation corresponds to the transformation done by the hidden layers of the neural network.

If the output contains $|\mathcal{Y}|$ modalities then the predictive model contains $|\mathcal{Y}|-1$ functions $g_y(x)$ in such a way that

$$\begin{cases} g_{\omega_0}(x) &= \widehat{P}(Y = \omega_0|X = x) \\ \vdots & \\ g_{\omega_{|\mathcal{Y}|-2}}(x) &= \widehat{P}(Y = \omega_{|\mathcal{Y}|-2}|X = x) \end{cases}$$

It should be noted that the last function $g_{\omega_{|\mathcal{Y}|-1}}$ must not be estimated since, for a given x, the sum of the conditional probabilities must be equal to one. Therefore the last function is defined as

$$g_{\omega_{|\mathcal{Y}|-1}}(x) = 1 - \sum_{k=0}^{|\mathcal{Y}|-2} g_{\omega_k}(x).$$

The predictive model is built with $\mathfrak{d}$ and is therefore calibrated to fit the training environment. But, as we assume the possibility to have a prior probability shift in the data, the testing environment changes. Therefore, the training and testing realizations are assumed to be generated from respectively X, Y and $\widetilde{X}, \widetilde{Y}$ where (of course) the output of the testing set is not observable.

We assume that the predictive model in the target environment is also of the softmax form

$$\begin{aligned} \widetilde{g}_y(x) &= \widehat{P}(\widetilde{Y} = y | \widetilde{X} = x) \\ &= \frac{\exp\left[(\tilde{\alpha}^y)^T \gamma(x) + \tilde{\alpha}_0^y\right]}{\sum_{y \in \mathcal{Y}} \exp\left[(\tilde{\alpha}^y)^T \gamma(x) + \tilde{\alpha}_0^y\right]}. \end{aligned} \tag{3.18}$$

The adjustment problem can be formulated as follows. Based on $\mathfrak{d}$ generated from X and Y, $\hat{\alpha}^y$ and $\hat{\alpha}_0^y$ are estimated via $\mathcal{I}^P$. Let $F_{\widetilde{X}\widetilde{Y}}(x, y)$ be the distribution in the testing environment where it is assumed that there was a prior probability shift between the training and testing. Let $\mathfrak{t}$ be the testing set generated from $\widetilde{X}$. The question is how to use $\mathfrak{t}$ to adjust the parameters $\hat{\alpha}^y$ and $\hat{\alpha}_0^y$ to the new testing environment?

From equation (3.15) at page 28,

$$\begin{aligned} \widetilde{g}_y(x) &= \frac{g_y(x)\widetilde{\rho}_y}{\sum_{y \in \mathcal{Y}} g_y(x)\widetilde{\rho}_y} \\ &= \frac{\exp\left[(\hat{\alpha}^y)^T \gamma(x) + \hat{\alpha}_0^y\right] \widetilde{\rho}_y}{\sum_{y \in \mathcal{Y}} \exp\left[(\hat{\alpha}^y)^T \gamma(x) + \hat{\alpha}_0^y\right] \widetilde{\rho}_y} \\ &= \frac{\exp\left[(\hat{\alpha}^y)^T \gamma(x) + \hat{\alpha}_0^y + \ln(\widetilde{\rho}_y)\right]}{\sum_{y \in \mathcal{Y}} \exp\left[(\hat{\alpha}^y)^T \gamma(x) + \hat{\alpha}_0^y + \ln(\widetilde{\rho}_y)\right]}. \end{aligned} \tag{3.19}$$

If we compare the previous result with equation (3.18), we get the following adjustment rule:

$$\left\{ \begin{array}{lcl} \tilde{\alpha}^y & = & \hat{\alpha}^y \\ \tilde{\alpha}_0^y & = & \hat{\alpha}_0^y + \ln\left(P(\widetilde{Y} = y)\right) - \ln\left(\widehat{P}(Y = y)\right) \end{array} \right. \tag{3.20}$$

where $\tilde{\alpha}^y$ and $\tilde{\alpha}_0^y$ are the softmax parameters in the testing environment, where $\hat{\alpha}^y$ and $\hat{\alpha}_0^y$ are the softmax parameters in the training environment and where $\widehat{P}(Y = y) = N_{\mathfrak{d}}^y / N_{\mathfrak{d}}$. This formula shows us how to adjust the parameters of a CPE model when a softmax is used at the output and when prior probability shift occurs in the data. As we can see, only the bias term must be adjusted. The other terms remain unchanged. The bias term adjustment is function of:

- the new prior probability in the testing set $P(\widetilde{Y} = y)$ that we assume to be known and
- the prior probability in the training set estimated by $\widehat{P}(Y = y) = N_{\mathfrak{d}}^y / N_{\mathfrak{d}}$.

Example

In this example, a logistic function is considered and the equation (3.20) will be used to adjust the bias term of the predictive model. We will reuse the same synthetic problem described in equations (3.16) and (3.17) at page 30. This synthetic problem is used to generate the same observations as at page 30: 252 observations from ω_0, 248 observations

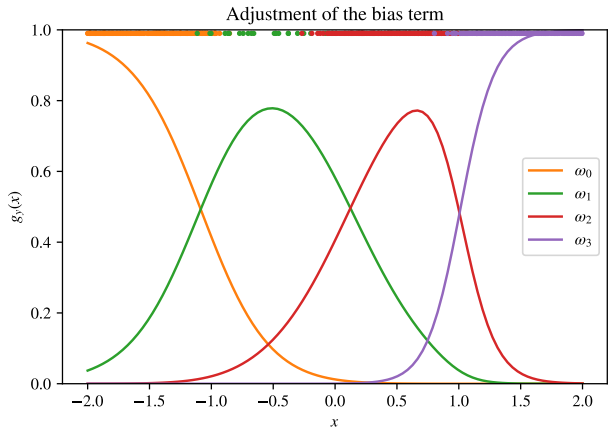

Figure 3.4: The predictive model obtained after recalibration of the bias term.

from ω_1, 253 observations from ω_2 and 247 observations from ω_3, which gives us a total of 1000 observations in the training set. To simulate shift in the data, the number of observations in ω_1 is reduced from 248 to 24; and the number of observations of the other output modalities remains the same (figure 3.3 at page 30).

After parametric identification, we get the following coefficients (for $y \in \{\omega_0 \dots \omega_3\}$):

$$\hat{\alpha}_0^y = (-0.35057895,\ 1.1672324,\ 3.1596501,\ -3.97630355)^T$$

and

$$\hat{\alpha}_1^y = (-5.90421903,\ -2.34167341,\ 0.58800961,\ 7.65788282)^T.$$

As, for $y \in \{\omega_0 \dots \omega_3\}$, we have $P(\widetilde{Y} = y) = (0.25,\ 0.25,\ 0.25,\ 0.25)^T$ and

$$\widehat{P}(Y = y) = (0.32474227,\ 0.03092784,\ 0.32603093,\ 0.31829897)^T,$$

the four adjustment values $\ln\left(P(\widetilde{Y} = y)\right) - \ln\left(\widehat{P}(Y = y)\right)$ are

$$(-0.26157093,\ 2.08980433,\ -0.26553133,\ -0.24153018)^T.$$

If we use these adjustment values in equation (3.20), we get the following new coefficients

$$\tilde{\alpha}_0^y = (-0.61214988,\ 3.25703673,\ 2.89411877,\ -4.21783373)^T$$

and

$$\tilde{\alpha}_1^y = (-5.90421903,\ -2.34167341,\ 0.58800961,\ 7.65788282)^T.$$

Note that only the bias terms are adjusted. Figure 3.4 shows the four new adjusted functions $\widetilde{g}_y(x)$. These curves are exactly the same as in the bottom of figure 3.3 at page 30, that shows that the algorithm (3.20) is a valid way to correct the distributions after a prior probability shift when the new prior is known.

Chapter 4

The adjusted classify and count (ACC) approach for prior probability shift

Let $\mathfrak{d} = \{(x_i, y_i)\}_{i=1}^{N_\mathfrak{d}}$ be a training dataset with iid observations obtained from $(X, Y) \sim F_Z(z) = F_{XY}(x, y)$ where $Y \in \mathcal{Y} = \{\omega_0, \omega_1, \omega_2, \ldots, \omega_{|\mathcal{Y}|-1}\}$.

We assume to be in a transductive transfer learning context where input testing data are known in advance. In order to distinguish between the training and testing populations, we will now say that the testing observations in $\mathfrak{t}$ are coming from $F_{\widetilde{X}\widetilde{Y}}(x, y)$ where $\widetilde{Y}$ is latent.

In this work, we address the problem where the class distribution changes and only unlabeled data are available from the new testing population. We will now study ways to correct the conditional probability predictions for this target environment when we assume to be in state of complete ignorance about the new a priori probability. Therefore we must simultaneously both estimate $P(\widetilde{Y} = y)$ (quantification problem) and correct the conditional probability (classification problem).

Different quantification methods have been proposed in the scientific literature. In this section, we study the *adjusted classify and count (ACC) approach* [16], also known as the *confusion matrix approach* [38].

4.1 The confusion matrix

The confusion matrix [11, 36] is typically used in machine learning to evaluate the performance of a predictive model in a supervised classification learning context. Let

$$\upsilon(\omega_j, \omega_k) = \sum_{i=1}^{N} I\left[h(x_i, \alpha) = \omega_k \ \wedge \ y_i = \omega_j\right]$$

be a function counting the number of times that the model predicts ω_k when the true class is ω_j where I is the indicator function which equals 1 if the proposition inside the bracket is true otherwise I returns 0.

The confusion matrix is a square matrix reporting all these υ values in such a way that

the rows represent the actual class of the instances and the columns their predicted class:

$$\begin{bmatrix} \upsilon(\omega_0,\omega_0) & \upsilon(\omega_0,\omega_1) & \cdots & \upsilon(\omega_0,\omega_{|\mathcal{Y}|-1}) \\ \upsilon(\omega_1,\omega_0) & \upsilon(\omega_1,\omega_1) & \cdots & \upsilon(\omega_1,\omega_{|\mathcal{Y}|-1}) \\ \vdots & \vdots & \ddots & \vdots \\ \upsilon(\omega_{|\mathcal{Y}|-1},\omega_0) & \upsilon(\omega_{|\mathcal{Y}|-1},\omega_1) & \cdots & \upsilon(\omega_{|\mathcal{Y}|-1},\omega_{|\mathcal{Y}|-1}) \end{bmatrix}.$$

We define

$$P(h(X|\alpha)=\omega_k|Y=\omega_j)$$

as the probability that the model $h(\cdot|\alpha)$ predicts ω_k when the true class is ω_j. Based on the values in the confusion matrix, this probability can be estimated as follows:

$$\begin{aligned} \widehat{P}(h(X|\alpha)=\omega_k|Y=\omega_j) &= \frac{\widehat{P}(h(X|\alpha)=\omega_k, Y=\omega_j)}{\widehat{P}(Y=\omega_j)} \\ &= \frac{\upsilon(\omega_j,\omega_k)}{\sum_{\omega_k\in|\mathcal{Y}|}\upsilon(\omega_j,\omega_k)}. \end{aligned} \tag{4.1}$$

Note that if $|\mathcal{Y}|=2$ and if ω_0 is the positive class then

$$\begin{bmatrix} \#TP & \#FN \\ \#FP & \#TN \end{bmatrix} = \begin{bmatrix} \upsilon(\omega_0,\omega_0) & \upsilon(\omega_0,\omega_1) \\ \upsilon(\omega_1,\omega_0) & \upsilon(\omega_1,\omega_1) \end{bmatrix}.$$

and

$$\widehat{P}(h(X|\alpha)=\omega_0|Y=\omega_0) = \frac{\#TP}{\#TP+\#FN} = \text{true positive rate (recall)}$$

$$\widehat{P}(h(X|\alpha)=\omega_1|Y=\omega_1) = \frac{\#TN}{\#FP+\#TN} = \text{true negative rate}$$

$$\widehat{P}(h(X|\alpha)=\omega_0|Y=\omega_1) = \frac{\#FP}{\#FP+\#TN} = \text{false positive rate}$$

$$\widehat{P}(h(X|\alpha)=\omega_1|Y=\omega_0) = \frac{\#FN}{\#TP+\#FN} = \text{false negative rate.}$$

4.2 The ACC approach

Let $P(h(\widetilde{X}|\alpha)=\omega_k)$ be the marginal of classifying an observation from the testing target environment in class ω_k. Note that this quantity can be easily estimated by applying the predictive model h on the testing set $\mathfrak{t}$ and then by counting the number of occurrences of each modality.

$$\widehat{P}(h(\widetilde{X}|\alpha)=\omega_k) = \frac{\sum_{i=1}^{N_\mathfrak{t}} I\left[h(x_i^\mathfrak{t},\alpha)=\omega_k\right]}{N_\mathfrak{t}}.$$

According to the law of total probability, we have

$$P(h(\widetilde{X}|\alpha)=\omega_k) = \sum_{j=1}^{|\mathcal{Y}|} P(h(\widetilde{X}|\alpha)=\omega_k|\widetilde{Y}=\omega_j)\ P(\widetilde{Y}=\omega_j) \tag{4.2}$$

In the previous equation, $P(\widetilde{Y}=\omega_j)$ is the prevalence of the output variable in the testing environment. It is this probability that we try to estimate in quantification learning. As

we already mentioned, the probability $P(h(\widetilde{X}|\alpha) = \omega_k)$ can easily be estimated. The only missing term in (4.2) is the conditional probability $P(h(\widetilde{X}|\alpha) = \omega_k|\widetilde{Y} = \omega_j)$.

As we suppose prior probability shift between (X, Y) and $(\widetilde{X}, \widetilde{Y})$, we assume that the within-class probability density is conserved

$$f_{X|Y}(x|\omega_j) = f_{\widetilde{X}|\widetilde{Y}}(x|\omega_j) \tag{4.3}$$

and, if the classifier h is applied on the input random variables X in the previous equation, we obtain

$$P(h(X|\alpha) = \omega_k|Y = \omega_j) = P(h(\widetilde{X}|\alpha) = \omega_k|\widetilde{Y} = \omega_j).$$

By using the previous equality with equation (4.2), we have

$$P(h(\widetilde{X}|\alpha) = \omega_k) = \sum_{j=1}^{|\mathcal{Y}|} P(h(X|\alpha) = \omega_k|Y = \omega_j)\ P(\widetilde{Y} = \omega_j) \tag{4.4}$$

where the conditional probability can now be estimated via the equation (4.1). As this estimate must done on an other dataset than the training set, cross validation (algorithm 1 at page 23) is used. Note that equation (4.4) is about solving a system of $|\mathcal{Y}|$ equations where $P(\widetilde{Y} = \omega_j)$ are the unknown variables.

The main issue with this method is that in practice the result of the probabilities $P(\widetilde{Y} = \omega_j)$ can turn out to be negative or greater than one [42]. This can happen for one or more of the following reasons:

- The shift in the dataset is in fact not prior probability shift, i.e. (4.3) does not hold.
- The cross-validation estimates of the confusion matrix is inaccurate. Note that it is known that the cross-validation estimate is a bias estimator for the prediction accuracy [46].
- The estimation of the scoring function $h(\cdot, \alpha)$ may be inaccurate.

In the next section, we will propose as contribution a method that solve the ACC method by a quadratic programming method with constraints that will require a solution that meets the definition of distribution, i.e. all positive values and sum equals to one.

We also notice that, for the resolution of the system, we often obtain a singular matrix and thus the inverse of this matrix does not exist. This is especially the case when, after the data shift, the prevalence of Y in the testing environment is very low. In this case, it easily happens that a column of the confusion matrix is null and therefore the matrix is singular (e.g. confusion matrix at equation (4.6) page 40).

4.3 Solving the ACC method by a quadratic program with constraints

We will reformulate the equation (4.4) as a quadratic program with constraints. The constraints ensure to have a solution that meets the definition of distribution (i.e. all the values must be positive and the sum must equal one).

To simplify the notation, we will rewrite the equation (4.4) in a matrix form. If $M \in \mathbb{R}^{|\mathcal{Y}|\times|\mathcal{Y}|}$, $p \in \mathbb{R}^{|\mathcal{Y}|}$ and $v \in \mathbb{R}^{|\mathcal{Y}|}$ then (4.4) can be rewritten as

$$v = M\ p$$

where $p = (P(\widetilde{Y} = \omega_0), \ldots, P(\widetilde{Y} = \omega_{|\mathcal{Y}|}))^T$ is the vector that must be estimated. We search a vector $\widehat{p}$ that respects the constraints such that $v \approx M\ \widehat{p} \ \Leftrightarrow\ (M\ \widehat{p} - v) \approx 0$. This can be represented as the following quadratic minimization problem with constraints:

$$\widehat{p} = \arg\min_{p\in\mathbb{R}^{|\mathcal{Y}|}} q(p)$$
$$\text{subject to: } \widehat{P}(\widetilde{Y} = \omega) \geq 0, \quad \forall \omega \in \mathcal{Y}$$
$$\sum_{\omega\in\mathcal{Y}} \widehat{P}(\widetilde{Y} = \omega) = 1$$

where

$$\begin{aligned}
q(p) &= \frac{1}{2}\|Mp - v\|^2 \\
&= \frac{1}{2}(Mp - v)^T(Mp - v) \\
&= \frac{1}{2}(p^T M^T - v^T)(Mp - v) \\
&= \frac{1}{2}(p^T M^T Mp - p^T M^T v - v^T Mp + v^T v) \\
&= \frac{1}{2}p^T \underbrace{M^T M}_{\mathscr{D}} p - \underbrace{v^T M}_{\mathscr{B}^T} p \\
&= \frac{1}{2}p^T \mathscr{D}\ p - \mathscr{B}^T p.
\end{aligned}$$

In the previous equation, the scalar $v^T v$ is removed from the optimization problem because it does not depend on p. Note that the constraints can also be rewritten in matrix form

$$\widehat{P}(\widetilde{Y} = \omega) \geq 0, \quad \forall \omega \in \mathcal{Y} \quad \Leftrightarrow \quad \underbrace{\mathbf{I}}_{\mathscr{A}_{\geq}}\ \widehat{p} \geq \underbrace{\vec{\mathbf{0}}}_{\mathscr{C}_{\geq}}$$

and

$$\sum_{\omega\in\mathcal{Y}} \widehat{P}(\widetilde{Y} = \omega) = 1 \quad \Leftrightarrow \quad \underbrace{\vec{\mathbf{1}}^T}_{\mathscr{A}_{=}} \widehat{p} = \underbrace{1}_{\mathscr{C}_{=}}$$

where $\mathbf{I}$ is the $|\mathcal{Y}| \times |\mathcal{Y}|$ identity matrix (i.e. diagonal matrix with ones on the diagonal) and, $\vec{\mathbf{0}} = (0, \ldots, 0)^T$ and $\vec{\mathbf{1}} = (1, \ldots, 1)^T$ are two vectors of size $|\mathcal{Y}|$ with respective zeros and ones.

In our experiments (chapter 7), to solve this quadratic program with constraints, we used the *solve_qp* function from the Python module *quadprog*. This function takes as input ($\mathscr{D}$, $\mathscr{B}$, $\mathscr{A}_{\geq}$, $\mathscr{C}_{\geq}$, $\mathscr{A}_{=}$, $\mathscr{C}_{=}$) and returns $\widehat{p}$.

4.4 Example – Binary classification

As an illustration, we will reuse the same binary classification problem that we have already defined at page 28. The training set $\mathfrak{d}$ contains 999 observations from ω_0 and 1001 observations from ω_1. The dataset $\mathfrak{d}$ is used to learn a logistic predictive model (like in the top left

of figure 3.2). If we evaluate the predictive model by $K = 5$ cross validation, we obtain the following confusion matrix:

$$\begin{bmatrix} 889 & 110 \\ 115 & 886 \end{bmatrix}$$

and with equation (4.1), we have

$$\begin{bmatrix} 0.88988989 & 0.11011011 \\ 0.11488511 & 0.88511489 \end{bmatrix}.$$

This gives us estimations for $P(h(X|\alpha) = \omega_k | Y = \omega_j)$.

The testing set is generated with the same random process as for the training set but, to simulate prior probability shift, observations from ω_1 are randomly removed. The testing dataset $\mathfrak{t}$ is composed of 1010 observations from ω_0 and only 99 observations from ω_1, that is to say $P(\widetilde{Y} = \omega_0) \approx 0.91073$ and $P(\widetilde{Y} = \omega_0) \approx 0.08927$. If we apply the predictive model learned with $\mathfrak{d}$ on the testing set $\mathfrak{t}$, we obtain the following estimation of $P(h(\widetilde{X}|\alpha) = \omega_k)$: 912/1109 and 197/1109 for respectively ω_0 and ω_1.

This gives us the following system of two linear equations:

$$\begin{cases} \widehat{P}(h(\widetilde{X}|\alpha) = \omega_0) &= \widehat{P}(h(X|\alpha) = \omega_0 | Y = \omega_0)\widehat{P}(\widetilde{Y} = \omega_0) + \widehat{P}(h(X|\alpha) = \omega_0 | Y = \omega_1)\widehat{P}(\widetilde{Y} = \omega_1) \\ \widehat{P}(h(\widetilde{X}|\alpha) = \omega_1) &= \widehat{P}(h(X|\alpha) = \omega_1 | Y = \omega_0)\widehat{P}(\widetilde{Y} = \omega_0) + \widehat{P}(h(X|\alpha) = \omega_1 | Y = \omega_1)\widehat{P}(\widetilde{Y} = \omega_1) \end{cases}$$

$$\Updownarrow$$

$$\begin{cases} 912/1109 &= 0.88988989\ \widehat{P}(\widetilde{Y} = \omega_0) + 0.11488511\ \widehat{P}(\widetilde{Y} = \omega_1) \\ 197/1109 &= 0.11011011\ \widehat{P}(\widetilde{Y} = \omega_0) + 0.88511489\ \widehat{P}(\widetilde{Y} = \omega_1) \end{cases}$$

$$\Updownarrow$$

$$\begin{bmatrix} 0.822 \\ 0.178 \end{bmatrix} = \begin{bmatrix} 0.88988989 & 0.11488511 \\ 0.11011011 & 0.88511489 \end{bmatrix} \begin{bmatrix} \widehat{P}(\widetilde{Y} = \omega_0) \\ \widehat{P}(\widetilde{Y} = \omega_1) \end{bmatrix}$$

$$\Updownarrow$$

$$\begin{bmatrix} 1.1420767 & -0.1482379 \\ -0.1420767 & 1.1482379 \end{bmatrix} \begin{bmatrix} 0.822 \\ 0.178 \end{bmatrix} = \begin{bmatrix} 0.91287 \\ 0.08713 \end{bmatrix} = \begin{bmatrix} \widehat{P}(\widetilde{Y} = \omega_0) \\ \widehat{P}(\widetilde{Y} = \omega_1) \end{bmatrix}$$

The estimation of the prevalence of the output in the testing environment (0.91287, 0.08713) is very close to real prevalence (0.91073 and 0.08927). Note that in this case, the solution of the linear equation provides a valid solution (i.e. $0.91287 > 0$, $0.08713 > 0$ and $0.91287 + 0.08713 = 1$), therefore it was not necessary to use the constraints.

4.5 Example – Multiclass classification

Let us now consider the same multiclass classification problem already defined at page 30 but where the output prevalence in the testing $P(\widetilde{Y} = \cdot)$ is now assumed to be unknown.

The training dataset contains 776 observations: 252 observations from ω_0, 24 observations from ω_1, 253 observations from ω_2 and 247 observations from ω_3.

$$\begin{bmatrix} \widehat{P}(Y=\omega_0) \\ \widehat{P}(Y=\omega_1) \\ \widehat{P}(Y=\omega_2) \\ \widehat{P}(Y=\omega_3) \end{bmatrix} = \begin{bmatrix} 0.3247 \\ 0.0309 \\ 0.3260 \\ 0.3183 \end{bmatrix} \tag{4.5}$$

The dataset $\mathfrak{d}$ is used to learn a logistic predictive model. A testing set with 1000 observations is also available: 249 observations from ω_0, 255 observations from ω_1, 242 observations from ω_2 and 252 observations from ω_3. The unknown four prevalences of Y in the testing set equal almost 0.25. It is these values that the algorithm ACC will have to estimate.

If we evaluate the logistic function by $K = 5$ cross validation, we obtain the following confusion matrix:

$$\begin{bmatrix} 252 & 0 & 0 & 0 \\ 15 & 0 & 9 & 0 \\ 0 & 0 & 230 & 23 \\ 0 & 0 & 2 & 245 \end{bmatrix} \tag{4.6}$$

and with equation (4.1), we have

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0.625 & 0 & 0.375 & 0 \\ 0 & 0 & 0.90909091 & 0.09090909 \\ 0 & 0 & 0.00809717 & 0.99190283 \end{bmatrix}.$$

As the determinant of the last matrix is null, this matrix cannot be inversed and therefore the system of equations defined in (4.4) cannot be solved. Meaning that in some cases, the ACC algorithm can just not be applied.

We tried to replace, on the same problem, the logistic model by a random forest and, with a $K = 5$ cross validation, we obtained the following confusion matrix:

$$\begin{bmatrix} 248 & 4 & 0 & 0 \\ 4 & 14 & 7 & 0 \\ 0 & 4 & 241 & 8 \\ 0 & 0 & 12 & 235 \end{bmatrix}$$

and with equation (4.1), we had

$$\begin{bmatrix} 0.98412698 & 0.01587302 & 0 & 0 \\ 0.125 & 0.58333333 & 0.29166667 & 0 \\ 0 & 0.01581028 & 0.95256917 & 0.03162055 \\ 0 & 0 & 0.048583 & 0.951417 \end{bmatrix}. \tag{4.7}$$

If we apply the predictive model learned with $\mathfrak{d}$ on the testing set $\mathfrak{t}$, we obtain:

$$\begin{bmatrix} \widehat{P}(h(\widetilde{X}|\alpha)=\omega_0) \\ \widehat{P}(h(\widetilde{X}|\alpha)=\omega_1) \\ \widehat{P}(h(\widetilde{X}|\alpha)=\omega_2) \\ \widehat{P}(h(\widetilde{X}|\alpha)=\omega_3) \end{bmatrix} = \begin{bmatrix} 0.28 \\ 0.18 \\ 0.289 \\ 0.251 \end{bmatrix}. \tag{4.8}$$

Note that equation (4.8) gives a naive estimation of the prevalence when the predictive model is used without adjustment[1].

By solving equation (4.4) with (4.7) and (4.8), we obtain the following estimation of the prevalence of Y in the testing set

$$\begin{bmatrix} \widehat{P}(\widetilde{Y} = \omega_0) \\ \widehat{P}(\widetilde{Y} = \omega_1) \\ \widehat{P}(\widetilde{Y} = \omega_2) \\ \widehat{P}(\widetilde{Y} = \omega_3) \end{bmatrix} = \begin{bmatrix} 0.28288341 \\ 0.1012286 \\ 0.2934499 \\ 0.24883235 \end{bmatrix}. \tag{4.9}$$

We know that the true prior in the testing set equals 0.25 for the four classes. Compared with (4.5), the previous equation corrected the distribution of Y in the target environment slightly but we are still far from the true distribution (i.e. $P(\widetilde{Y} = \omega_k) = 0.25$, for $k = 1, \ldots, 4$). Note even that the probabilities in (4.9) do not sum to one.

If we apply the method based on the resolution of a quadratic program, described in section 4.3, we obtain, for the same problem, the following estimation of the prevalence in the testing environment

$$\begin{bmatrix} \widehat{P}(\widetilde{Y} = \omega_0) \\ \widehat{P}(\widetilde{Y} = \omega_1) \\ \widehat{P}(\widetilde{Y} = \omega_2) \\ \widehat{P}(\widetilde{Y} = \omega_3) \end{bmatrix} = \begin{bmatrix} 0.2945241 \\ 0.1385011 \\ 0.3005869 \\ 0.2663878 \end{bmatrix}.$$

Compared with the original prior probability in (4.5), there is a correction of the prevalence in the target environment. As expected, the four probabilities are closer to their target 0.25 but the correction remains relatively weak. It should be noted that the naive approach (i.e. *base*) remain better

[1] In the experimental chapter 7, for the estimation of the prior in $\mathfrak{t}$, this method of using the predictive model learned on $\mathfrak{d}$ without any adjustment is named "*base*".

Chapter 5

The class distribution estimation (CDE-iterate) approach for prior probability shift

Let $\mathfrak{d}$ be a training dataset with $N_{\mathfrak{d}}$ iid observations drawn from $(X, Y) \sim F_Z(z) = F_{XY}(x, y)$ where Y is a discrete random variable and let $\mathfrak{t}$ be a testing dataset with $N_{\mathfrak{t}}$ iid observations coming from $F_{\widetilde{X}\widetilde{Y}}(x, y)$ where $\widetilde{Y}$ is a discrete latent random variable. We assume also the presence of a change in the marginal probability of the output where the new prevalence of the output variable in the testing environment is unknown.

In this section we will present the *class distribution estimation* (CDE) method [48] to do the quantification. As we will see, the algorithm proposed in [48], can only deal with binary classification problems. Let us consider a binary classification task (i.e. $|\mathcal{Y}| = 2$), where the confusion matrix is a 2×2 matrix:

$$\begin{bmatrix} \#TP & \#FN \\ \#FP & \#TN \end{bmatrix}.$$

The following terms can be defined from the confusion matrix:

- **Positive rate** $pr = (\#TP + \#FN)/(\#TP + \#FN + \#FP + \#TN)$
- **Negative rate** $nr = (\#FP + \#TN)/(\#TP + \#FN + \#FP + \#TN)$
- **Distribution mismatch ratio** $dmr = (pr/nr) : (\widetilde{pr}/\widetilde{nr})$
- **True positive rate** $tpr = \#TP/(\#TP + \#FN)$
- **False positive rate** $fpr = \#FP/(\#FP + \#TN)$

where the tilde symbol ($\widetilde{\ }$) is used to denote the values associated with the new testing environment.

Let us consider for example that the dataset from the original training distribution has 900 positive examples and 100 negative examples ($pr = 0.9$, $nr = 0.1$) and the new testing dataset has 200 positive and 800 negative examples ($\widetilde{pr} = 0.2$ and $\widetilde{nr} = 0.8$). In our example, we have $dmr = 9 : (1/4) = 36 : 1$. Based on the ratio of the positive rate to the negative rate, the positives are 36 times more frequent in the training environment than in the testing one.

Algorithm 2 CDE-iterate

function CDE-ITERATE($\mathfrak{d}, \mathfrak{t}, \Lambda, nb\_loops$)
  $\widehat{\alpha} \leftarrow \mathcal{I}^P(\Lambda, \mathfrak{d})$
  for $i \leftarrow 1$ **to** nb_loops **do**
    $\{\widehat{y}_j\}_{j=1}^{N_\mathfrak{t}} \leftarrow \{h(x_j, \widehat{\alpha}) \mid \forall x_j \in \mathfrak{t}\}$
    $\widehat{\widetilde{pr}/\widetilde{nr}} \leftarrow$ Use the predictions $\{\widehat{y}_j\}_{j=1}^{N_\mathfrak{t}}$ to estimate $\widetilde{pr}/\widetilde{nr}$
    $cost_{FP} \leftarrow \widehat{pr/nr} : \widehat{\widetilde{pr}/\widetilde{nr}}$ ▷ Note that if $\text{cost}_{FN} = 1$ then $\text{cost}_{FP} = dmr$.
    $\widehat{\alpha} \leftarrow \mathcal{I}^P(\Lambda, \mathfrak{d}, cost_{FP}, cost_{FN} = 1)$ ▷ Cost sensitive parametric identification [15].
  end for
  $\widehat{P}(\widetilde{Y} = \omega_1) \leftarrow \frac{\widehat{\widetilde{pr}/\widetilde{nr}}}{\widehat{\widetilde{pr}/\widetilde{nr}}+1}$
  return $\widehat{\alpha}, \widehat{P}(\widetilde{Y} = \omega_1)$
end function

As the output distribution changes between the training and the testing, it is needed to recalibrate the predictive model. It is possible to do this recalibration during the parametric identification step [15] in different ways:

- Resample (or reweight) the training examples to change the class distribution to match the new testing distribution.
- Adapt the thresholds used to decide the class label.
- Change the ratio of misclassification costs between false positive and false negative predictions.

In the original paper presented in [48], it is the third method that is used and the following equation determines the costs ratio between false positive and false negative predictions:

$$\frac{\text{cost}_{FP}}{\text{cost}_{FN}} = \frac{pr/nr}{\widetilde{pr}/\widetilde{nr}} = dmr. \tag{5.1}$$

Considering our example, the costs ratio:

$$\text{cost}_{FP} : \text{cost}_{FN} = 36 : 1 \;\Leftrightarrow\; \text{cost}_{FP} = 36 \times \text{cost}_{FN}.$$

Note that if cost_{FN} is fixed to 1, then $\text{cost}_{FP} = dmr$ (this property will be used in algorithm 2).

Let us assume that ω_1 is the positive class, where $P(\widetilde{Y} = \omega_1)$ is the new unknown prevalence in the testing set. In this case, we have the following ratio between the positive rate and the negative rate in the testing environment

$$\widetilde{pr}/\widetilde{nr} = \frac{P(\widetilde{Y} = \omega_1)}{1 - P(\widetilde{Y} = \omega_1)}.$$

Since the output values of the test observations are unknown, the probability $P(\widetilde{Y} = \omega_1)$ can not be directly estimated without quantification learning and therefore $\widetilde{pr}/\widetilde{nr}$ is also unknown.

The method *CDE-iterate* is an iterative algorithm and its pseudo-code is given in algorithm 2. The input of the CDE-iterate functions are (i) a training dataset, (ii) a testing

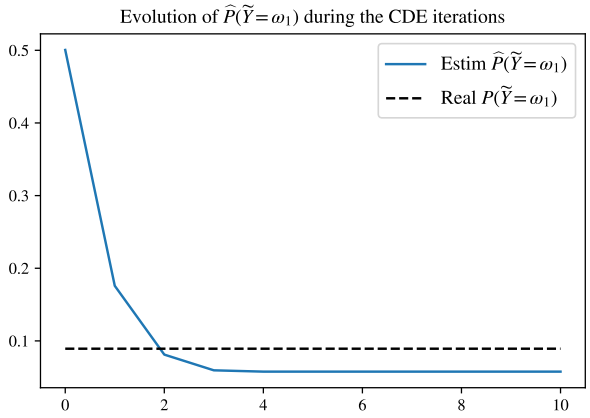

Figure 5.1: Evolution of the prior probabilities during the ten first iterations of the CDE algorithm.

dataset, (ii) a set of predictive models and (iv) the number of iterations of the main loop. At line 2, the training dataset is used to create an initial predictive model (equation (3.6) at page 22). The main loop is between lines 3 and 8. At line 4, the predictive model $h(\cdot, \widehat{\alpha})$ is applied on the testing set. Let[1] $N_{\mathfrak{t}}^{\omega_1}$ and $N_{\mathfrak{t}}^{\omega_0}$ be respectively the number of positive and negative predictions done on the testing observations $\{\widehat{y}_j\}_{j=1}^{N_{\mathfrak{t}}}$. At line 5, $\widehat{\widetilde{pr}/\widetilde{nr}}$ is estimated by $N_{\mathfrak{t}}^{\omega_1}/N_{\mathfrak{t}}^{\omega_0}$. Line 6 makes an estimation of the distribution mismatch ratio (drm) where $\widehat{pr/nr}$ is estimated from the training set $\mathfrak{d}$. Note that, as we assume $cost_{FN} = 1$, we have $drm = cost_{FN}$. A cost sensitive learning algorithm is used at line 7 where the false negative cost was computed at the previous line and the false positive cost is fixed at 1. Note that the experiences done in this work are implemented in Python and, unfortunately, no implementation of a cost sensitive algorithm is available in Python[2]. That is why, we had to adjust from scratch a classical learning algorithm for cost sensitive learning. At the end of iterations of the main loop, line 9 computes the estimation of the new prevalence of the positive class ω_1. Note that, as we assume to be in a binary classification setting, we have $\widehat{P}(\widetilde{Y} = \omega_0) = 1 - \widehat{P}(\widetilde{Y} = \omega_1)$. At the last line, the function returns the last generated model and the estimation of the prevalence.

It is interesting to note that this algorithm can be applied on non CPE models. It means that rather than correcting the conditional probability, this algorithm makes the correction at the level of the decision.

The CDE method proposed in [48] is applicable only for binary classification problems and this is due to the fact that equation (5.1) is valid only when $|\mathcal{Y}| = 2$. To adapt the CDE methods to multiclass problems, it is needed to adapt equation (5.1) when $|\mathcal{Y}| > 2$. We did not find this extension in the scientific literature.

5.1 Example – Binary classification

We reused the same simple binary classification problem that we already used at page 28 and in section 4.4 at page 38. Figure 5.1 shows the evolution of the estimation $\widehat{P}(\widetilde{Y} = \omega_1)$ during the 10 first iterations of the CDE algorithm where the horizontal dotted line is the target true distribution of the output variable in the testing environment.

As we can see, at the initialization step, $\widehat{P}(\widetilde{Y} = \omega_1)$ is close to 0.5 which is the prevalence of $Y = \omega_1$ in the training environment. Then the algorithm quickly reduces the value of the estimate and reached a stable value after step 3. As we can see, the algorithm has a little underrated the new probability in the testing environment.

5.2 The CDE-iterate approach is not a Fisher's consistency estimator

Fisher consistency is a desirable property for an estimator. Roughly speaking, it requires that if the *whole* population of a random variable is available then this estimator should give exactly the right answer.

The three estimators studied in this work (ACC, CDE and EM [3]) have been analyzed in [42]. The author found that ACC and EM are Fisher's consistent but a counter-example shows that CDE is not consistent.

Although Fisher's consistency gives no idea of the magnitude of the errors that an estimator will do when a given finite set of observations is available, the author of [42] argues that Fisher's consistency could be used as a criterion to reject estimators that are unlikely to provide good estimates in the test environment after prior probability shift. Being consistency is not a sufficient criterion for a quantification algorithm to be useful but, according to the author of [42], lack of consistency should be considered as a good reason to revoke an estimator because, even with a very huge dataset, lack of Fisher's consistency does not guarantee to have a good estimate. As CDE is not Fisher's consistent, it cannot be trusted to deliver reliable estimates of the prevalence in the testing environment.

It must be mentioned that even if the CDE estimator is not consistent, in the experimental chapter 7, we will see that empirically the CDE estimator is not worse than the other methods (when $|\mathcal{Y}| = 2$).

[1] Assuming that ω_1 stands for positive and ω_0 stands for negative.

[2] We only found "*COSTCLA: A Cost-Sensitive Classification Library in Python*" but the code is no longer maintained and is not compatible with the current version of Python 3.5.2.

[3] The EM algorithm will be presented in the next chapter.

Chapter 6

The expectation-maximization (EM) approach for prior probability shift

6.1 Adjusting the output of a CPE classifier in presence of a Prior Probability Shift

In this section, largely inspired by [38], we will assume to be in a classification context where $\mathfrak{d} = \{(x_i, y_i)\}_{i=1}^{N_\mathfrak{d}}$ is a training dataset with iid observations obtained from $(X, Y) \sim F_Z(z) = F_{XY}(x, y)$ where $Y \in \mathcal{Y} = \{\omega_0, \omega_1, \omega_2, \ldots, \omega_{|\mathcal{Y}|-1}\}$.

6.1.1 Adjusting the output of a classifier to a new unknown a priori

We assume that the new a priori probability distribution of the testing set $P(\widetilde{Y} = y)$ is unknown. In this case, equation (3.15) at page 28 cannot be directly applied. This section presents an EM iterative algorithm (section 3.2) for the a posteriori $\widetilde{g}_y(x)$ adjustment where $\widetilde{Y}$ is the latent variable and where $\widetilde{X}$ is the observable variable from which $N_\mathfrak{t}$ realizations are available in the testing set $\mathfrak{t}$.

The distribution of the new testing observations is

$$\begin{aligned} f_\mathcal{T}(\mathfrak{t}) &= \prod_{j=1}^{N_\mathfrak{t}} f_{\widetilde{X}}(x_j^\mathfrak{t}) \\ &= \prod_{j=1}^{N_\mathfrak{t}} \left[\sum_{y \in \mathcal{Y}} f_{\widetilde{X}|\widetilde{Y}}(x_j^\mathfrak{t}|y)\ P(\widetilde{Y} = y) \right] \\ &= \prod_{j=1}^{N_\mathfrak{t}} \left[\sum_{y \in \mathcal{Y}} f_{X|Y}(x_j^\mathfrak{t}|y)\ P(\widetilde{Y} = y) \right] \end{aligned}$$

where $\widetilde{Y}$ is the latent unknown random variable. The last line of the previous equation is obtained by assuming that the within-class probability density is conserved, since we assume that only the a priori probability changes from the training set to the new testing set. Roughly speaking, in the last line of previous equation, we have 'transferred' the

information about the within-class probability density from the source environment to the target environment. In the previous equation, $f_{X|Y}(x|y)$ can be estimated from the training dataset via density estimation techniques and generative machine learning models [25] but $P(\widetilde{Y}=\cdot)$ remains unknown. The goal of the iterative EM algorithm is to estimate all the a priori probabilities $P(\widetilde{Y}=\cdot)$ of the latent random variable (quantification).

As before, let $g_y(x)$ be the output of a *class probability estimate* (CPE) model (equation (3.4)) built on the training set $\mathfrak{d}$. For iteration number s of the EM procedure, let us define $\widehat{P}_s(\widetilde{Y}=y)$ and $\widetilde{g}_y^{(s)}(x)=\widehat{P}_s(\widetilde{Y}=y|\widetilde{X}=x)$ as respectively the estimations of the new a priori and the new a posteriori. If $\widehat{P}_s(\widetilde{Y}=y)$ for $s=1$ is initialized by the a priori of the training set (equation (3.12))

$$\widehat{P}_{s=1}(\widetilde{Y}=y)=N_{\mathfrak{d}}^y/N_{\mathfrak{d}}$$

then, according to [38], for $s=1,2,3,\ldots$ the EM method provides the two following steps for each testing observation $x_j^{\mathfrak{t}}$ and each class ω:

$$\begin{aligned}\widetilde{g}_y^{(s)}(x_j^{\mathfrak{t}})&=\frac{g_y(x_j^{\mathfrak{t}})\frac{\widehat{P}_s(\widetilde{Y}=y)}{N_{\mathfrak{d}}^y/N_{\mathfrak{d}}}}{\sum_{y\in\mathcal{Y}}g_y(x_j^{\mathfrak{t}})\frac{\widehat{P}_s(\widetilde{Y}=y)}{N_{\mathfrak{d}}^y/N_{\mathfrak{d}}}};\quad \forall x_j^{\mathfrak{t}}\in\mathfrak{t}\text{ and }\forall y\in\mathcal{Y}\\ \widehat{P}_{s+1}(\widetilde{Y}=y)&=(N_{\mathfrak{t}})^{-1}\sum_{j=1}^{N_{\mathfrak{t}}}\widetilde{g}_y^{(s)}(x_j^{\mathfrak{t}});\quad \forall y\in\mathcal{Y}.\end{aligned}\tag{6.1}$$

Note, from (6.1), we can see that if

$$\widehat{P}_{s=1}(\widetilde{Y}=y)=N_{\mathfrak{d}}^y/N_{\mathfrak{d}}$$

then

$$\widetilde{g}_y^{(s=1)}(x_j^{\mathfrak{t}})=g_y(x_j^{\mathfrak{t}})$$

and so,

$$\widehat{P}_{s=2}(\widetilde{Y}=y)=(N_{\mathfrak{t}})^{-1}\sum_{j=1}^{N_{\mathfrak{t}}}g_y(x_j^{\mathfrak{t}});\quad \forall y\in\mathcal{Y}.$$

In step $s=2$, we find an estimate of the prior that is using the output of the original model g_y applied on the test set $\mathfrak{t}$. It is possible to initialize the algorithm by the prior in the training $\mathfrak{d}$ or by the predictions of the initial model on the testing $\mathfrak{t}$. These two initializations are possible but at the end they are equivalent.

The only difference between $\widetilde{g}_{\omega_k}^{(s)}$ (equation (6.1)) and $\widetilde{g}_{\omega_k}$ (equation (3.15) at page 28) is that in $\widetilde{g}_{\omega_k}^{(s)}$ the a priori $P(\widetilde{Y}=\omega_k)$ is replaced by the estimation $\widehat{P}_s(\widetilde{Y}=\omega_k)$. At each iteration step s, both $\widetilde{g}_{\cdot}^{(s)}(\cdot)$ and $\widehat{P}_s(\widetilde{Y}=\cdot)$ are re-estimated for each testing example and for each class ω until convergence of the a priori probabilities to a (local) maximum. Note that, in order to obtain a good a priori probability estimation $\widehat{P}(\widetilde{Y}=y)$, it is necessary that the $g_y(x_j^{\mathfrak{t}})$ is reasonably well approximated by the model. A study of the robustness of the proposed EM procedure, with respect to an imperfect a posteriori probability estimation, is provided in [38].

We will now explain the rationale of this iterative EM method [38]. In order to pose the problem as a maximum likelihood problem with incomplete information, let us introduce an unobserved latent random variable $\mathcal{L}$ in such a way that we can assume $(\widetilde{X},\mathcal{L})\sim$

$F_{\widetilde{X}\mathcal{L}}(x,l)$ and we associate with the new testing input observations $\mathfrak{t} = \{x_1^{\mathfrak{t}}, \ldots, x_{N_{\mathfrak{t}}}^{\mathfrak{t}}\}$ a vector $\{\mathcal{L}_1, \ldots, \mathcal{L}_{N_{\mathfrak{t}}}\}$ of random variables. The support of the latent variable is a binary indicator vector of length $|\mathcal{Y}|$. Let l be a realization of $\mathcal{L}$. If l_{jk} is the component k of vector l_j, then $l_{jk} = 1$ and $l_{ji} = 0$ for each $k \neq i$, if and only if the output class label associated with observation $x_j^{\mathfrak{t}}$ is ω_k. For instance, if $x_j^{\mathfrak{t}}$ is assigned to ω_k then

$$l_j = [\underset{1}{0}, \ldots, 0, \underset{k}{1}, 0, \ldots, \underset{|\mathcal{Y}|}{0}]^T.$$

Let $\theta = [p(\omega_1), p(\omega_2), \ldots, p(\omega_{|\mathcal{Y}|})]^T$ be the parameters that must be optimized by EM where $p(\omega_k) = P(\widetilde{Y} = \omega_k)$ is the a priori to observe ω_k in the testing environment. The likelihood to observe the input observations of the testing set is

$$\mathscr{L}(\theta, (\mathfrak{t}, \mathcal{L})) = \prod_{j=1}^{N_{\mathfrak{t}}} \prod_{k=1}^{|\mathcal{Y}|} \left[f_{\widetilde{X}|\widetilde{Y}}(x_j^{\mathfrak{t}}|\omega_k)\ p(\omega_k) \right]^{\mathcal{L}_{jk}}$$

and the log likelihood is

$$\ell(\theta, (\mathfrak{t}, \mathcal{L})) = \sum_{j=1}^{N_{\mathfrak{t}}} \sum_{k=1}^{|\mathcal{Y}|} \mathcal{L}_{jk} \log [p(\omega_k)] + \sum_{j=1}^{N_{\mathfrak{t}}} \sum_{k=1}^{|\mathcal{Y}|} \mathcal{L}_{jk} \log \left[f_{\widetilde{X}|\widetilde{Y}}(x_j^{\mathfrak{t}}|\omega_k) \right].$$

Note that the log likelihood in the previous equation depends on a random variable $\mathcal{L}$. This is due to the fact that no observations are directly available from this latent variable. The EM iterative algorithm can be used in such situations where, during the E-step, the previous log likelihood is replaced by the conditional expectation of $\ell(\theta, (\mathfrak{t}, \mathcal{L}))$ over $P(\{\mathcal{L}_1, \ldots, \mathcal{L}_{N_{\mathfrak{t}}}\} = \{l_1, \ldots, l_{N_{\mathfrak{t}}}\}|\mathfrak{t}, \theta)$.

Let $\widehat{\theta}_1$ be initialized by the a priori computed on the training set and let $\widehat{\theta}_s = [\widehat{p}_s(\omega_1), \ldots, \widehat{p}_s(\omega_{|\mathcal{Y}|})]^T$ be the current estimation values of the parameters at step s. The E-step is

$$\begin{aligned} Q(\theta|\widehat{\theta}_s) &= E_{\mathcal{L}|\mathfrak{t},\widehat{\theta}_s} \left[\ell(\theta, (\mathfrak{t}, \mathcal{L})) \right] \\ &= \sum_{j=1}^{N_{\mathfrak{t}}} \sum_{k=1}^{|\mathcal{Y}|} E_{\mathcal{L}|x_j^{\mathfrak{t}},\widehat{\theta}_s} \left[\mathcal{L}_{jk} \right] \log \left[p(\omega_k) \right] \\ &+ \sum_{j=1}^{N_{\mathfrak{t}}} \sum_{k=1}^{|\mathcal{Y}|} E_{\mathcal{L}|x_j^{\mathfrak{t}},\widehat{\theta}_s} \left[\mathcal{L}_{jk} \right] \log \left[f_{\widetilde{X}|\widetilde{Y}}(x_j^{\mathfrak{t}}|\omega_k) \right] \end{aligned}$$

where we assume that the data $\{(x_j^{\mathfrak{t}}, l_j)\}_{j=1}^{N_{\mathfrak{t}}}$ are independent and where

$$\begin{aligned} E_{\mathcal{L}|x_j^{\mathfrak{t}},\widehat{\theta}_s} \left[\mathcal{L}_{jk} \right] &= 0 \cdot P(\mathcal{L}_{jk} = 0|x_j^{\mathfrak{t}}, \widehat{\theta}_s)\ +\ 1 \cdot P(\mathcal{L}_{jk} = 1|x_j^{\mathfrak{t}}, \widehat{\theta}_s) \\ &= P(\mathcal{L}_{jk} = 1|\widetilde{X} = x_j^{\mathfrak{t}}, \widehat{\theta}_s) \\ &= P(\widetilde{Y} = \omega_k|\widetilde{X} = x_j^{\mathfrak{t}}, \widehat{\theta}_s) \\ &= \widetilde{g}_{\omega_k}^{(s)}(x_j^{\mathfrak{t}}). \end{aligned}$$

The expected likelihood is therefore

$$Q(\theta|\widehat{\theta}_s) = \sum_{j=1}^{N_t}\sum_{k=1}^{|\mathcal{Y}|} \widetilde{g}_{\omega_k}^{(s)}(x_j^t)\ \log\left[p(\omega_k)\right] + \sum_{j=1}^{N_t}\sum_{k=1}^{|\mathcal{Y}|} \widetilde{g}_{\omega_k}^{(s)}(x_j^t)\ \log\left[f_{\widetilde{X}|\widetilde{Y}}(x_j^t|\omega_k)\right].$$

At the M-step of the EM algorithm, the new estimate at next iteration $s+1$ will be the value of the parameters θ maximizing $Q(\theta|\widehat{\theta}_s)$ (equation (3.8)). Given the constraint $\sum_k p(\omega_k) = 1$, we define the Lagrange function as follows:

$$\begin{aligned} L(\theta) &= Q(\theta|\widehat{\theta}_s) + \lambda\left[1 - \sum_{k=1}^{|\mathcal{Y}|} p(\omega_k)\right] \\ &= \sum_{j=1}^{N_t}\sum_{k=1}^{|\mathcal{Y}|} \widetilde{g}_{\omega_k}^{(s)}(x_j^t)\ \log\left[p(\omega_k)\right] + \sum_{j=1}^{N_t}\sum_{k=1}^{|\mathcal{Y}|} \widetilde{g}_{\omega_k}^{(s)}(x_j^t)\ \log\left[f_{\widetilde{X}|\widetilde{Y}}(x_j^t|\omega_k)\right] \\ &+ \lambda\left[1 - \sum_{k=1}^{|\mathcal{Y}|} p(\omega_k)\right]. \end{aligned}$$

The first and second derivatives of the Lagrange according to $p(\omega_k)$ are computed

$$\begin{aligned} \frac{\partial L(\theta)}{\partial p(\omega_k)} &= \left[p(\omega_k)\right]^{-1}\sum_{j=1}^{N_t} \widetilde{g}_{\omega_k}^{(s)}(x_j^t) - \lambda \\ \frac{\partial^2 L(\theta)}{\partial p(\omega_k)^2} &= -\underbrace{\left[p(\omega_k)\right]^{-2}}_{>0}\ \underbrace{\sum_{j=1}^{N_t} \widetilde{g}_{\omega_k}^{(s)}(x_j^t)}_{>0} < 0 \end{aligned}$$

As the second derivative is always negative, the value of the parameter $p(\omega_k)$ at next iteration canceling the first derivative is the one maximizing the Lagrange function:

$$\frac{\partial L(\theta)}{\partial p(\omega_k)} = 0 \implies \widehat{p}_{s+1}(\omega_k) = \lambda^{-1}\ \sum_{j=1}^{N_t} \widetilde{g}_{\omega_k}^{(s)}(x_j^t)$$

If we sum the previous equation over the values in $\mathcal{Y}$:

$$\begin{aligned} \sum_{k=1}^{|\mathcal{Y}|} \widehat{p}_{s+1}(\omega_k) = \lambda^{-1}\ \sum_{j=1}^{N_t}\underbrace{\sum_{k=1}^{|\mathcal{Y}|} \widetilde{g}_{\omega_k}^{(s)}(x_j^t)}_{=1} &= \lambda^{-1}\ N_t = 1 \\ \Leftrightarrow\ \lambda &= N_t \end{aligned}$$

and therefore we retrieve the M-step of equation (6.1) where $\widehat{p}_{s+1}(\omega_k) = \widehat{P}_{s+1}(\widetilde{Y} = \omega_k)$.

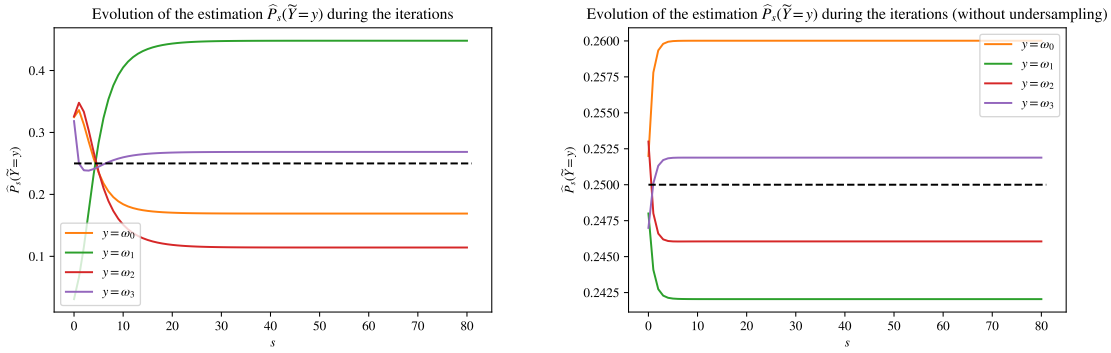

Figure 6.1: Evolution of the prior probability during the iterations of the EM algorithm when a logistic model is used to estimate $g_y(\cdot)$ and where the problem is defined in equations (3.16) and (3.17). Left: Prior probability shift is applied on the training data. Right: No prior probability shift is applied.

6.1.2 Example with $d = 1$

This adjustment method will now be tested on the same synthetic example described in equations (3.16) and (3.17) at page 30. The original training set contains $N_{\mathfrak{d}} = 1000$ observations with 252 observations from ω_0, 248 observations from ω_1, 253 observations from ω_2 and 247 observations from ω_3. The conditional probability distribution that we would like to have is shown in the top left of figure 3.3 at page 30. To simulate shift in the data, the number of observations in ω_1 is artificially reduced from 248 to 24 by selecting independently at random 24 observations from the 248 observations in the original training set. Reducing the number of examples in ω_1 alters the prior probabilities of each class. In the original training dataset, each class is represented by almost 1/4 of the observations. After reducing the number of examples in ω_1, classes ω_0, ω_2 and ω_3 each represents almost 32% of the observations in the new training set and ω_1 represents only almost 3% of the observations. The top right of figure 3.3 at page 30 shows the four new conditional distributions computed on the reduced training dataset.

Unlike the experience made on page 30, here we assume that the original proportions (i.e. 1/4 for each class) are unknown. The iterative method described in equation (6.1) is used to simultaneously estimate the four original proportions (i.e. quantification) and to adjust the a posteriori distributions.

The results of the evolution of the a priori estimations are given in the left part of figure 6.1. As we can see, the adjustment of the a priori does not converge to the expected value 0.25, annotated in the figure with a black dotted line. The two a priori distributions $\widehat{P}_s(\widetilde{Y} = \omega_0)$ and $\widehat{P}_s(\widetilde{Y} = \omega_2)$ are pushed to a too small value when s increases. It is interesting to see that both distributions ω_0 and ω_2 are on the edges (see top left of figure 3.3). It is as if, during the EM iterations, the distributions ω_0 and ω_2 were crushed by the distribution ω_1 in the middle.

Figure 6.2 shows the conditional distributions at six steps of the EM iterative algorithm ($s = 1$, $s = 2$, $s = 3$, $s = 4$, $s = 10$, $s = 20$). The initial step ($s = 0$) is at the top right of figure 3.3 (page 30). As we can see, the evolution of the conditional distributions is complex. Already at step $s = 10$, ω_2 (red) is pushed down. Figure 6.3 shows the four conditional

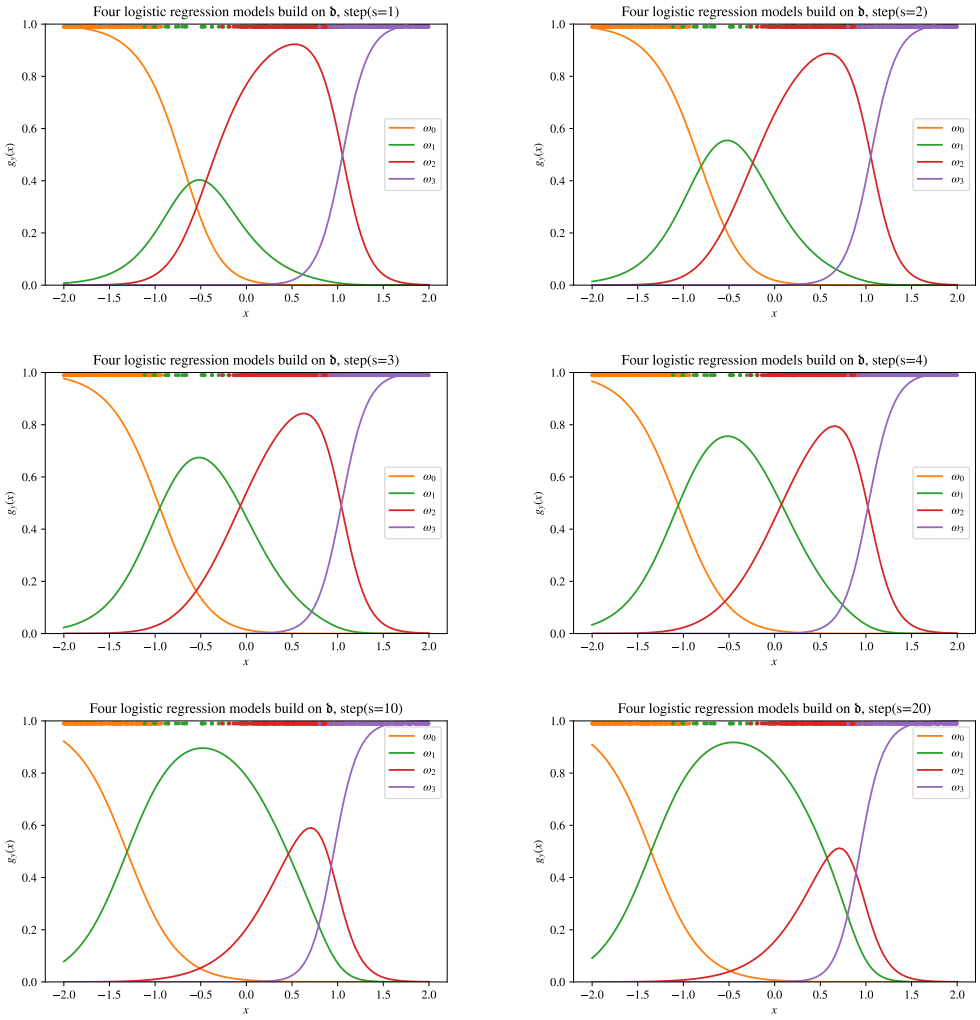

Figure 6.2: Six steps of the EM iterative algorithm where a logistic model is used to estimate $g_y(\cdot)$ and where the problem is defined in equations (3.16) and (3.17).

distributions after the convergence of the EM algorithm. As we can see $\widetilde{g}_{\omega_1}^{(s=79)}(x)$ crushes the distributions ω_2.

Logistic models were used to do the previous experiences. In order to evaluate the impact of the family of models on the quantification task, we did the same experience with random forest classifiers. The evolution of the prior probability during the EM iterations is given at the left part of figure 6.4. If we compare this result with the evolution given at the left part of figure 6.1, we can see that the distribution of ω_2 does not seem to be overestimated as it was previously when logistic classifiers were used. As expected, the family of model has an impact on the quantification learning process.

In order to understand what is going on, we also applied the EM algorithm on the same problem but without undersampling class ω_1, i.e. the training set is not modified. This means that, before starting the adjustment EM algorithm, the four probabilities are already at the beginning very close to their objectives (i.e. $\widehat{P}_{s=1}(\widetilde{Y} = y) = N_{\mathfrak{d}}^y/N_{\mathfrak{d}} \approx 1/4$). As the algorithm starts with values very close to the objective, we can expect it to converge very

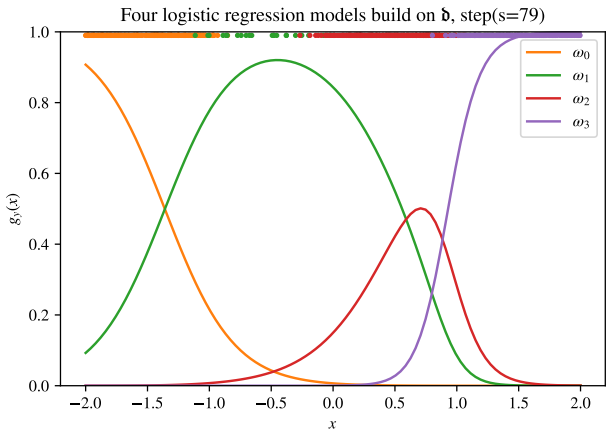

Figure 6.3: The estimation of the conditional probabilities $f_{\widetilde{x}|\widetilde{Y}}(x|y)$ after the convergence of the EM algorithm.

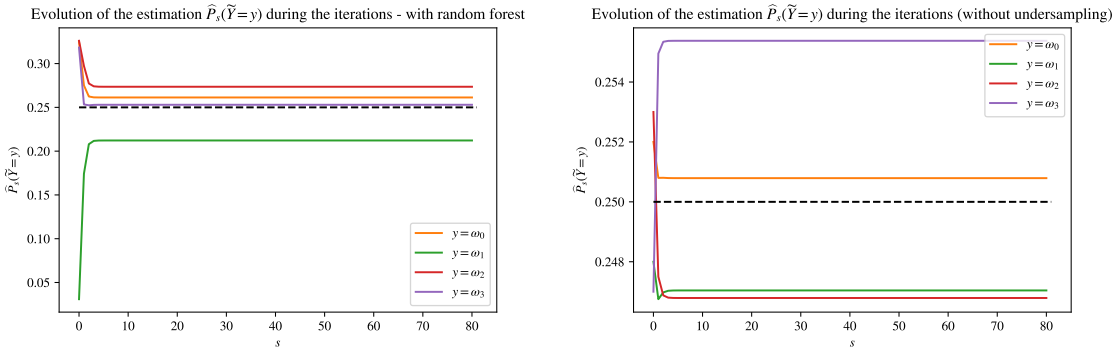

Figure 6.4: Evolution of the prior probability during the iterations of the EM algorithm when a random forest model is used. Left: Prior probability shift is applied on the training data. Right: No prior probability shift is applied.

quickly to the right values. As we can see on the right part of figures 6.1 and 6.4 , after starting with four prior probabilities close to $1/4$, the probabilities stabilize very quickly and do not diverge too much (the variation in the y-axis is relatively small (i.e. $[0.2425, 0.2600]$ in figure 6.1 and $[0.247, 0.255]$ in figure 6.4)). This corresponds to what was expected.

6.1.3 Example with $d = 2$

Let us now consider another multiclass classification problem where $\mathcal{Y} = \{\omega_0, \omega_1, \omega_2, \omega_3\}$ and where the input dimension d equals 2:

$$X_1 \sim Unif(-2, 2)$$
$$X_2 \sim Unif(-2, 2)$$

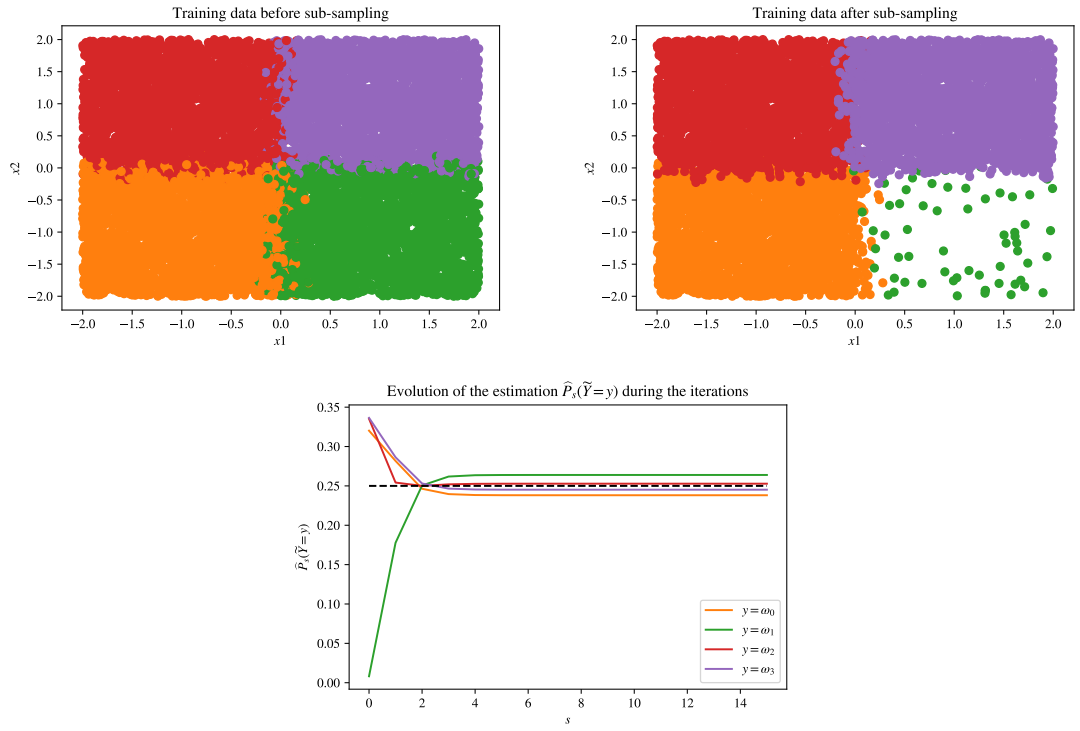

Figure 6.5: Upper left: Training data obtained from equation (6.2). Upper right: Training data after prior probability shift. Bottom: The evolution of the prior probability estimation during the EM iterations on the problem defined in equation (6.2) and when a logistic model is used.

with

$$(Y|X=x) = \begin{cases} \omega_0, & \text{if } x_1+\epsilon_1<0 \text{ and } x_2+\epsilon_1<0 \\ \omega_1, & \text{if } x_1+\epsilon_1>0 \text{ and } x_2+\epsilon_1<0 \\ \omega_2, & \text{if } x_1+\epsilon_1<0 \text{ and } x_2+\epsilon_1>0 \\ \omega_3, & \text{if } x_1+\epsilon_1>0 \text{ and } x_2+\epsilon_1>0 \end{cases}, \text{ where } \begin{bmatrix}\epsilon_1 \\ \epsilon_2\end{bmatrix} \sim N\left(\begin{bmatrix}0\\0\end{bmatrix}, \begin{bmatrix}0.01 & 0\\ 0 & 0.01\end{bmatrix}\right). \tag{6.2}$$

In the top left part of figure 6.5, we find the observations generated from the previous equation. The original training set contains $N_{\mathfrak{d}} = 10000$ observations with 2422 observations from ω_0, 2497 observations from ω_1, 2536 observations from ω_2 and 2545 observations from ω_3. To simulate shift in the data, the number of observations in ω_1 (green points) is artificially reduced from 2497 to 62 observations. The new sub sampled set is in the top right part of figure 6.5. On the bottom of the figure, we can see the evolution of the four a priori probabilities when the EM algorithm at equation (6.1) is used to adjust the priors and logistic models are used to estimate the conditional probability in the training environment. We can see that in this case, the four probabilities seem to converge towards the good objective represented by the black dashed line in the figure.

Figure 6.6 shows the predictions done by the model at three steps of the EM algorithm. The horizontal and vertical dotted lines indicate the true target function (equation (6.2)). The upper left of figure 6.6 shows the predictions before adjustment. We can see that ω_1

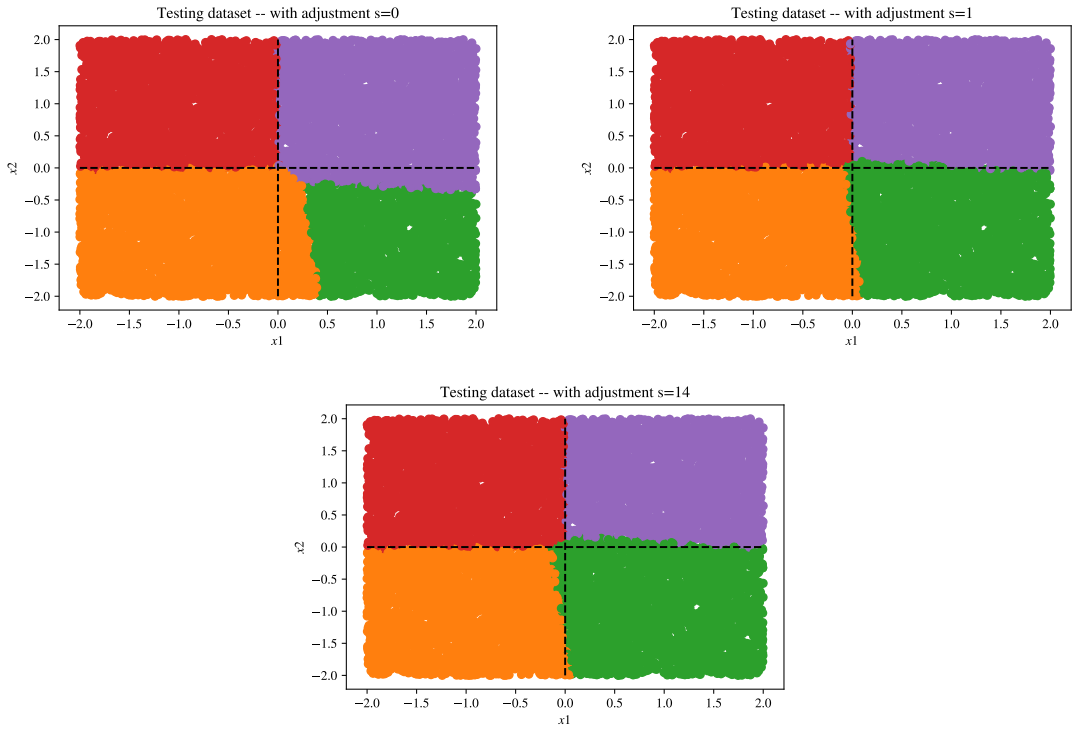

Figure 6.6: The predictive models obtained at three steps of the EM algorithm. Upper left: The predictive model learned on the training set without any recalibration. Upper right: The predictive model after the first step of the EM algorithm. Bottom: The predictive model after converge of the EM algorithm.

(green) is predicted only for few points in the lower right corner. Follows the confusion matrix of this initial model where the rows represent the actual classes of the instances and the columns their predicted classes:

$$\begin{bmatrix} 2364 & 0 & 54 & 4 \\ 407 & 1755 & 3 & 332 \\ 53 & 0 & 2441 & 42 \\ 2 & 0 & 53 & 2490 \end{bmatrix}.$$

We see that, on a total of 2497 observations from ω_1, only 1755 observations are well classified (i.e. the *recall* on ω_1 equals $1755/2497 \approx 0.7$) and it is also interesting to observe that, except on the diagonal, the second column of the confusion matrix has only zeros; meaning that no false predictions are done from the class ω_1 (i.e. the *precision* on ω_1 equals $1755/1755 = 1$). Without adjustment, the F_1-score on ω_1 equals $2 \times \frac{precision \times recall}{precision + recall} \approx 0.82$.

The upper right of figure 6.6 shows the predictions after only one iteration step of the EM algorithm. We see that after only one step ($s = 1$), the EM algorithm has already increased significantly the area covered by the predictions on ω_1. The new confusion matrix

is

$$\begin{bmatrix} 2310 & 65 & 47 & 0 \\ 72 & 2382 & 0 & 43 \\ 60 & 4 & 2427 & 45 \\ 0 & 69 & 45 & 2431 \end{bmatrix}.$$

The performance looks much better. We retrieve now 2382 of the 2497 observations from ω_1 (i.e. the *recall* on ω_1 equals $2382/2497 \approx 0.954$) but we are now doing some false predictions from the class ω_1 (i.e. the *precision* on ω_1 equals $2382/2520 \approx 0.945$). The F_1-score on ω_1 equals approximately 0.949.

The bottom of figure 6.6 shows the predictions after convergence of the EM algorithm (step $s = 14$). The confusion matrix after convergence is

$$\begin{bmatrix} 2270 & 98 & 54 & 0 \\ 46 & 2432 & 1 & 18 \\ 51 & 8 & 2438 & 39 \\ 0 & 109 & 55 & 2381 \end{bmatrix}.$$

The *recall* and the *precision* on ω_1 are now respectively approximately equal to 0.974 and 0.919. As expected, we get a much better recall but it is at the cost of a degradation of the precision. The F_1-score on ω_1 equals approximately 0.945.

6.2 A stopping criteria for the EM algorithm

We saw in the previous examples that, in some circumstances, it can happen that the evolution of the prior probability diverges during the iterations of the EM algorithm. To deal with this issue, it is common to decide in advance to do only a small number of iterations [18].

In this section, we will propose a method that tries to identify this divergence. This will help to identify when the algorithm starts to divergence in order to stop the iterations of the EM method (i.e. before the end of the convergence). Our method is therefore an adaptive method (i.e., it is not needed to decide in advance the number of iterations of the EM algorithm).

The rationale of our proposed method is the following.

- If we look at the upper right of figure 3.3 at page 30 and at the upper left of figure 6.6, we can see that *without adjustment*, the modalities overrepresented in the training environment $\mathfrak{d}$ compared to target environment $\mathfrak{t}$ crush the other modalities.

  For instance: in figure 3.3, without adjustment the distribution of ω_1 is crushed by ω_0 and ω_2, and in the upper left of figure 6.6, the distribution of ω_1 is blocked in the corner by ω_0 and ω_3.

- If we look at figure 6.3 at page 53 and at the bottom of figure 6.6, we can see that *after convergence*, the modality underrepresented in $\mathfrak{d}$ compared to $\mathfrak{t}$ exceed its limits after convergence. In figure 6.3, the two boundaries of the Bayes' decision rule of modality ω_1 were expected to be -1 and 0 (see equation (3.17) at page 30) but we see in the figure that the conditional distribution of ω_1 has a too high variance and the boundaries go well beyond their limits (i.e. approximately -1.5 and 0.5). The same remark can be made when we look at figure 6.6 where we can see that, after

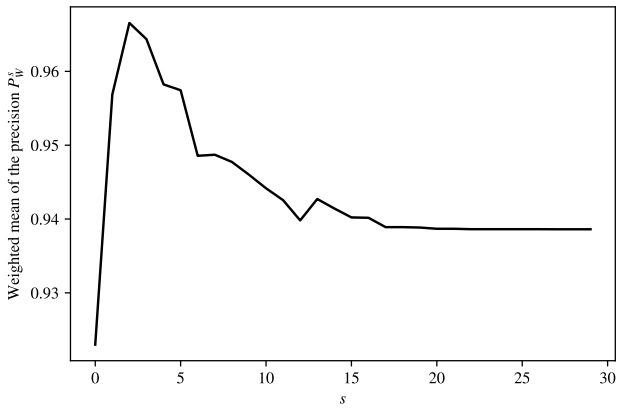

Figure 6.7: Evolution of P_W^s during the iterations of the EM algorithm on the quantification problem defined at page 51.

convergence, the conditional distribution on ω_1 tends to exceed its limits (although it is not as strong as in figure 6.3).

What we would like to avoid are situations where a previously minority distribution becomes majority after convergence to the point of dominating all others because it is a symptom that the EM algorithm has diverged. To detect these situations, we propose to use the weighted precision P_W score:

$$P_W^s = \sum_{\omega \in \mathcal{Y}} \widehat{P}(Y = \omega) \times \text{precision}(\omega)$$

where[1]

$$\widehat{P}(Y = \omega) = N_{\mathfrak{d}}^{\omega} / N_{\mathfrak{d}}$$

and

$$\text{precision}(\omega) = \frac{\sum_{i=1}^{N_{\mathfrak{d}}} I\left[h^s(x_i^{\mathfrak{d}}) = \omega \ \wedge \ y_i = \omega\right]}{\sum_{i=1}^{N_{\mathfrak{d}}} I\left[h^s(x_i^{\mathfrak{d}}) = \omega\right]}$$

with

$$h^s(x_i^{\mathfrak{d}}) = \arg\max_{\omega} \widetilde{g}_{\omega}^{(s)}(x_i^{\mathfrak{d}}).$$

The weighted precision score calculates precision metric for each label, and finds their average, weighted by the number of true instances for each label. It should be noticed that P_W^s is computed on the training set because no supervised observations are available in $\mathfrak{t}$.

Figure 6.7 shows the evolution of the weighted precision score during the 30 first steps of the EM algorithm applied on the example given at page 51 when $d = 1$ and when a logistic function is used to estimate $g_y(\cdot)$.

- At the two first steps, we observe an increase of P_W. This is because at $s = 0$ (top right of figure 3.3 at page 30) and $s = 1$ (top left of figure 6.2 at page 52), the precision on ω_1 is almost one (i.e. the maximum[2]) but the model does false predictions with ω_0 and

[1] It should be noted that we tried to replace $\widehat{P}(Y = \omega)$ by $\widehat{P}_s(\widetilde{Y} = \omega)$ but the performances were very bad.

[2] It is because the model predicted almost never ω_1 and the few of these predictions are true positives.

ω_2 and consequently it reduces the precision on these two modalities. At the two first steps, by increasing the prior probability of ω_1 and decreasing the prior probability on ω_0 and ω_2, the EM algorithm increases the precision of ω_0 and ω_2 while maintaining the precision of ω_1 close to one.

- The value of P_W starts to decrease after step 3 because it is from this step that the precision on ω_1 begins to reduce and, at the same time, the improvements on the precisions of ω_0 and ω_2 are less high.

The maximum of P_W is reached when $s = 2$. This corresponds to the solution obtained at the top right of figure 6.2. Compared to the initial situation at $s = 0$ (figure 3.3) and the final situation after convergence (figure 6.3 at page 53), the solution proposed by our strategy (top right of figure 6.2) is much better.

Note that there can be two reasons why, during the EM iterations, a previously minority modality becomes largely majority. The first one is that the EM algorithm diverges to a wrong solution and the second reason could be that it is the good solution (i.e. the previously minority modality in the training environment, after a strong prior probability shift, is really dominating the other modalities in the new testing environment). In this section, we make the assumption that the first reason is the right one. But it is an assumption and therefore, if our assumption is wrong, our stopping strategy can wrongly stop the EM iterations. This is what will happen in our experiments in table 7.3 at page 67 where, with a random forest, the classical EM method overperforms the EM method with the stopping criteria when β is small[3].

6.3 Adjusting the bias term of a CPE classifier in presence of Prior Probability Shift

This section investigates how it is possible to directly adjust the bias term of a classifier when the data are suffering from prior probability shift. Rather than just changing the output of the model, this new approach will have the advantage of being able to directly redesign the predictive model to the new sampling condition.

Equation (3.20) at page 32 gave us a procedure to adjust the bias term of a CPE classifier when the new a priori in the testing environment is known. But in many situations, this $P(\widetilde{Y} = y)$ is unknown and the procedure described in equation (3.20) can therefore not be applied. In this section, we will reuse the EM method presented in section 6.1.1 but here our goal will be to directly modify the bias term of the CPE model.

From equations (6.1), (3.19) and (3.20), if $\tilde{\alpha}_0^y(s)$ from $s = 1$ is initialized by $\hat{\alpha}_0^y$ computed in the training environment then, for $s = 1, 2, 3, \ldots$ the EM method provides the following

[3] A small β stands from a high probability shift.

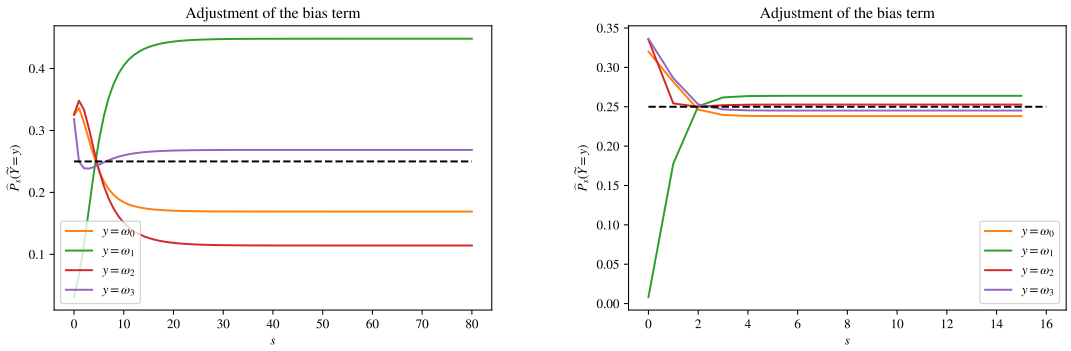

Figure 6.8: Evolution of the EM algorithm when the bias term is directly adjusted. Left: The quantification problem is defined in equations (3.16) and (3.17). Right: The quantification problem is defined in equation (6.2). In both cases, a logistic model is used to estimate $g_y(\cdot)$.

steps for each testing observation $x_j^{\mathfrak{t}}$ and each class ω:

$$\begin{aligned}
\widetilde{g}_y^{(s)}(x_j^{\mathfrak{t}}) &= \frac{g_y(x_j^{\mathfrak{t}})\frac{\widehat{P}_s(\widetilde{Y}=y)}{N_{\mathfrak{d}}^y/N_{\mathfrak{d}}}}{\sum_{y\in\mathcal{Y}} g_y(x_j^{\mathfrak{t}})\frac{\widehat{P}_s(\widetilde{Y}=y)}{N_{\mathfrak{d}}^y/N_{\mathfrak{d}}}}; \quad \forall x_j^{\mathfrak{t}} \in \mathfrak{t} \text{ and } \forall y \in \mathcal{Y} \\
&= \frac{\exp\left[(\hat{\alpha}^y)^T\gamma(x_j^{\mathfrak{t}}) + \tilde{\alpha}_0^y(s))\right]}{\sum_{y\in\mathcal{Y}}\exp\left[(\hat{\alpha}^y)^T\gamma(x_j^{\mathfrak{t}}) + \tilde{\alpha}_0^y(s)\right]}; \quad \forall x_j^{\mathfrak{t}} \in \mathfrak{t} \text{ and } \forall y \in \mathcal{Y} \qquad (6.3) \\
\widehat{P}_{s+1}(\widetilde{Y}=y) &= (N_{\mathfrak{t}})^{-1}\sum_{j=1}^{N_{\mathfrak{t}}}\widetilde{g}_y^{(s)}(x_j^{\mathfrak{t}}); \quad \forall y \in \mathcal{Y}
\end{aligned}$$

where $\tilde{\alpha}_0^y(s) = \hat{\alpha}_0^y + \ln\left(\widehat{P}_s(\widetilde{Y}=y)\right) - \ln\left(N_{\mathfrak{d}}^y/N_{\mathfrak{d}}\right)$. The justification of the rationale of this method follows the same way as in section 6.1.1.

6.3.1 Example

In this example, we will reproduce the two experiments done in section 6.1.2 at page 51.

First, the same synthetic example described in equations (3.16) and (3.17) is used. Like before, the training set contains $N_{\mathfrak{d}} = 1000$ observations with 252 observations from ω_0, 248 observations from ω_1, 253 observations from ω_2 and 247 observations from ω_3. To simulate a shift in the data, 24 observations from 248 are selected independently at random. We assume that the original proportions (i.e. 1/4 for each class) are unknown. The top right of figure 3.3 at page 30 shows the four conditional distributions computed on the new training dataset. The objective will be to recalibrate this conditional distribution. The evolution of the new prior probability estimation is given in the left part of the figure 6.8 when equation (6.3) is used to update the intercepts of the logistic model. As we can see, this curve is exactly the same as the one in the left of figure 6.1 at page 51.

We will now reuse the same 10000 observations generated from the equation (6.2) described at page 54. If we apply equation (6.3), the evolution of the prior of the new prior

estimation is given in the right part of the figure 6.8. Note that, this curve is exactly the same as the one in the bottom of figure 6.5.

This confirms that the equation (6.3) is equivalent to the equation (6.1) when the new prior is unknown except that (6.3) allows to directly modify the bias term in a CPE predictive model. As expected, it shows empirically that adjusting the probability $P(Y = y|X = x)$ or adjusting directly the bias term when a softmax classifier is used, leads exactly to the same result.

Chapter 7

Experiments

7.1 Experimental design

7.1.1 Datasets

In this section, we will use 25 well known real datasets (table 7.1) extracted from the *UCI Machine Learning Repository*[1] to assess the adaptive methods presented in the previous chapters. Note that, to the best of our knowledge, this is the first study that makes such a comprehensive empirically comparison of the 4 strategies for prior probability shift. For each dataset, we indicate in table 7.1, the number of observations (*nb obs.*), the input dimension (d) and the number of categories in the output variable ($|\mathcal{Y}|$). Note that for simplicity, we made the following modifications on the original datasets:

- To have datasets without missing values, rows with missing values are removed.
  - *Dermatology* had 8 observations with at least one missing value.
  - *Mammographic Mass* had 131 observations with at least one missing value.
  - *Mice Protein Expression* had 528 observations with at least one missing value.
- To have only real input variables, all the input categorical variables are transformed by one hot encoding[2].
- All the numerical input variables are normalized (i.e. centered and divided by the standard deviation).
- In *Letter Recognition*, we took only vowels (A, E, I, O, U, Y).

In the experiment, the datasets in table 7.1 will be randomly divided into two: a training set and testing set. The prior probability shift will be applied on the training set before the parametric identification and the test set will not be modified. Different levels of intensities of the prior probability shift will be studied.

[1] https://archive.ics.uci.edu/ml/

[2] It is a way to represent a categorical variable with, for instance, five levels as a binary vector of length four.

#	Dataset name	nb obs.	d	$\lvert\mathcal{Y}\rvert$
1	Abalone	4177	8	3
2	Balance Scale	625	4	3
3	Banknote authentication	1372	4	2
4	Blood Transfusion Service Center	748	4	2
5	Breast Tissue	106	9	6
6	Default of credit card clients	30000	23	2
7	Dermatology	366	34	6
8	Ecoli	336	8	8
9	Glass Identification	214	9	6
10	Haberman's Survival	306	3	2
11	HTRU2	17898	8	2
12	Image Segmentation	210	19	7
13	Iris	150	4	3
14	Letter Recognition	4664	16	6
15	Magic Gamma Telescope	19020	10	2
16	Mammographic Mass	830	5	2
17	Mice Protein Expression	552	80	8
18	Occupancy Detection	8143	5	2
19	Page Blocks Classification	5473	10	5
20	Pen-Based Recognition of Handwritten Digits	7494	16	10
21	Sonar - Connectionist Bench	208	60	2
22	Seismic-bumps	2584	18	2
23	Spambase	4601	57	2
24	Waveform Database Generator (Version 1)	5000	21	3
25	Wine	178	13	3

Table 7.1: List of the real datasets used during the experiments.

7.1.2 Algorithms

Two types of models were used during the experiments. We indicate, in the following table, the type of model, the Python function from *Scikit-Learn* used for the parametric identification and the hyper-parameters:

Type of model	Python function	Hyper-parameters
Logistic regression	*sklearn.linear_model.LogisticRegression()*	Default sklean's hyper-parameters
Random forest	*sklearn.ensemble.RandomForestClassifier()*	Number of trees set to 200

The following adaptive strategies for prior probability shift are compared:

- **acc**: The *adjusted classify and count* (ACC) adjustment strategy (also called the *confusion matrix* strategy) described in chapter 4 at page 35. As we have seen, the ACC adjustment method can issue problems when solving the system of equations because the matrix that we have to inverse can be singular. It is also possible to get probabilities that are not in $[0, 1]$ or that do not sum to one.

- **acc-qp**: This is the version of the ACC adjustment method, described in section 4.3 at page 37, where a quadratic program with constraints is used to resolve the system of linear equations.

- **cde**: The *class distribution estimation* (CDE-iterate) adjustment strategy described in chapter 5 at page 43. Note that this method cannot be applied when $|\mathcal{Y}| > 2$.

- **em**: The *expectation-maximization* adjustment strategy described in chapter 6 at page 47.

- **em-stop**: The *expectation-maximization* adjustment strategy with a stopping criterion described in section 6.2 at page 56.

For benchmarking purpose, we added the following other strategies:

- **oracle-bayes**: This method will cheat because it will already know the true prevalence of the output in the testing environment. Therefore, it is not going to do quantification. The *oracle-bayes* strategy will use its knowledge about the true prior probability in the target environment to apply the Bayes correction given in equation (3.15) at page 28. This is a strategy that will make a perfect adaptation of the predictive model $g(\cdot)$. It should be noted that in this case $g(\cdot)$ is still estimated with a biased training set $\mathfrak{d}$.

- **oracle** This method is also cheating because it uses for training the *original training data set* when no prior probability shift is applied. This means that in this situation, we will assume that all the samples in the training and testing sets are identically distributed. As we will see and as expected, this method will have the best performance during the experiments.

- **base** The two previous strategies are *oracle* based benchmarks meaning that they have information that is not accessible in real conditions. These two strategies are somehow superior strategies and the performance of conventional strategies should not reach the performance of these two strategies. They represent an upper bound of the best performance for the conventional strategies. We introduced a eighth and last strategy called *base*. This is the method that learns the predictive model on the training set on which priority probability shift was introduced and it does not make any adjustment to correct the model. It is a strategy that estimates the new prevalence by simply applying (without any adjustment) a classifier learned on data from the source environment on the testing date from the target environment and then the estimation of the new prevalence is obtained by counting the the prediction of each class. Although very naive, this method is very often used in practice by machine learners experts who are not always aware about the specificities of quantification learning [18]. The performance of this *base* method represents a lower bound that classical methods should beat. Note that, like in [20, 18], this method is often called *classify and count* (CC).

7.1.3 Experimental method

During the experiments, we will estimate the ability of the strategies in two contexts: quantification and classification. In quantification context, the strategies will have to calculate an estimate of the prior in the testing environment $\widehat{P}(\widetilde{Y} = y)$ and, in the classification context, the algorithm (3.15) is used to recalibrate the predictive model to the new estimated prior. Note that, as the methods oracle-bayes and oracle are not performing quantification, the quantification performances are compared between the other methods (acc, acc-qp, cde, em, em-stop, base).

The experimental design is described in algorithm 3. This function receives four parameters: (i) a generic type of model Λ which is logistic regression or random forest, (ii) a dataset $\mathfrak{D}$ from table 7.1, (iii) the rate of subsampling β that we will apply to simulate the prior probability shift and (iv) the number of times that the experience is repeated. The main loop (lines 2 to 17) repeats the whole experimental design "nb_loops" times. At line 3, the

Algorithm 3 Experimental design

1: **function** EXP-DESIGN($\Lambda, \mathfrak{D}, \beta$, nb_loops = 100)
2: **for** $l \leftarrow 1$ **to** nb_loops **do**
3: $\mathfrak{d}, \mathfrak{t} \leftarrow$ split_train_test($\mathfrak{D}$) ▷ Stratified random sampling
4: $\mathfrak{d}' \leftarrow$ modify_prior_proba($\mathfrak{d}, \beta$) ▷ See algorithm 4
5: $g \leftarrow \mathcal{I}^P(\Lambda, \mathfrak{d}')$
6: $\widetilde{g}_{\text{acc}} \leftarrow$ Adjustment-acc($g, \mathfrak{t}$)
7: $\widetilde{g}_{\text{cde}} \leftarrow$ Adjustment-cde($g, \mathfrak{t}$)
8: $\widetilde{g}_{\text{em}} \leftarrow$ Adjustment-em($g, \mathfrak{t}$)
9: $\widetilde{g}_{\text{em-stop}} \leftarrow$ Adjustment-em-stop($g, \mathfrak{t}$)
10: $\widetilde{g}_{\text{oracle-bayes}} \leftarrow$ Adjustment-oracle-bayes($g, \mathfrak{t}$)
11: $\widetilde{g}_{\text{oracle}} \leftarrow \mathcal{I}^P(\Lambda, \mathfrak{d})$
12: **for** $j \leftarrow 1$ **to** $N_{\mathfrak{t}}$ **do**
13: **for** $\mathfrak{g} \in \{g, \widetilde{g}_{\text{acc}}, \widetilde{g}_{\text{cde}}, \widetilde{g}_{\text{em}}, \widetilde{g}_{\text{em-stop}}, \widetilde{g}_{\text{oracle-bayes}}, \widetilde{g}_{\text{oracle}}\}$ **do**
14: $\widehat{y}_j^{\mathfrak{g}} \leftarrow \arg\max_{\omega \in \mathcal{Y}} \mathfrak{g}_\omega(x_j^{\mathfrak{t}})$
15: **end for**
16: **end for**
17: **end for**
18: **return** $\left(\{\widehat{y}_{jl}\}, \{\widetilde{y}_{jl}^{\widetilde{g}_{\text{acc}}}\}, \{\widetilde{y}_{jl}^{\widetilde{g}_{\text{cde}}}\}, \{\widetilde{y}_{jl}^{\widetilde{g}_{\text{em}}}\}, \{\widetilde{y}_{jl}^{\widetilde{g}_{\text{em-stop}}}\}\{\widetilde{y}_{jl}^{\widetilde{g}_{\text{oracle-bayes}}}\}, \{\widetilde{y}_{jl}^{\widetilde{g}_{\text{oracle}}}\}, \{y_{jl}^{\mathfrak{t}}\}\right)$
19: **end function**

dataset $\mathfrak{D}$ is divided at random (stratified random sampling) into the training dataset $\mathfrak{d}$ and the testing dataset $\mathfrak{t}$. Stratification sampling is used to ensure that $\mathfrak{d}$ and $\mathfrak{t}$ have exactly the same prior. But, as prior probability shift is assumed, it is needed to modify the training dataset $\mathfrak{d}$ and this is done at line 4 by the method "modify_prior_proba" (algorithm 4). At line 5, the parametric identification algorithm is applied on Λ with the new training dataset $\mathfrak{d}'$ and lines 6-10 adjust the predictive model[3] g. At line 11, the *oracle* model is learned with $\mathfrak{d}$ (rather than $\mathfrak{d}'$). The Bayes' decision rule is then applied on all testing observations (lines 12 to 16) with the original model g, the oracle model $\widetilde{g}_{\text{oracle}}$ and the adjusted models $\widetilde{g}$. The function returns eight matrices, each with dimensions "$N_{\mathfrak{t}} \times nb\_loops$":

- $\{\widehat{y}_{jl}\}$ contains all the predictions done by the model without adjustment (that corresponds to *base* method),
- $\{\widetilde{y}_{jl}^{\mathfrak{g}}\}$ contains all the predictions done by the model with adjustment where $\mathfrak{g} \in \{\widetilde{g}_{\text{acc}}, \widetilde{g}_{\text{acc-qp}}, \widetilde{g}_{\text{cde}}, \widetilde{g}_{\text{em}}, \widetilde{g}_{\text{em-stop}}, \widetilde{g}_{\text{oracle-bayes}}, \widetilde{g}_{\text{oracle}}\}$ and
- $\{y_{jl}^{\mathfrak{t}}\}$ contains the real output values (it is used to compute the classification errors).

At line 4 of algorithm 3, a method "modify_prior_proba" is used to transform the original dataset by introducing prior probability shift. The method "modify_prior_proba" is described in algorithm 4. It has two parameters: (i) a training dataset $\mathfrak{d}$ and (ii) the rate of prior probability shift β that we will introduce in the data. The prior probability shift is introduced by randomly removing observations from some modalities of the output variable. As $|\mathcal{Y}| > 1$, there are several modalities on which observations can be deleted. At line 3, a random number n between 1 and $|\mathcal{Y}| - 1$ is selected. n is the number of modalities on which

[3]The methods return also estimations of $P(\widetilde{Y} = \cdot)$ for quantification.

Algorithm 4 modification of the prior probability in $\mathfrak{d}$

```
1: function MODIFY_PRIOR_PROBA(𝔡, β)
2:     𝔡' ← {}
3:     n ← Random selection of an element from {1, 2, 3, ..., |𝒴| − 1}
4:     Ω ← Random selection of n elements from {ω_0, ω_1, ω_2, ..., ω_{|𝒴|−1}}
5:     for ω ∈ {ω_0, ω_1, ω_2, ..., ω_{|𝒴|−1}} do
6:         v ← {(x^𝔡, y^𝔡) | (x^𝔡, y^𝔡) ∈ 𝔡 ∧ y^𝔡 = ω}
7:         if ω ∈ Ω then
8:             v' ← Select randomly ⌈|v| × β⌉ elements from v.
9:             𝔡' ← 𝔡' ∪ v'
10:        else
11:            𝔡' ← 𝔡' ∪ v
12:        end if
13:    end for
14:    return 𝔡'
15: end function
```

the shift will be done. Line 4 selects randomly the n modalities Ω on which the algorithm will introduce prior probability shift. The loop from line 5 to line 13 iterates on the $|\mathcal{Y}|$ modalities of the output variable. v is a set containing all the observations from the original training set $\mathfrak{d}$ where the output equals ω. If ω is in Ω, a new set $v^{'}$ is created at line 8 where $\lceil |v| \times \beta \rceil$ elements from v are randomly selected without replacement. The parameter β is a real number in $[0, 1]$ and defines the intensity of the prior probability shift. A small value of β means a high intensity of prior probability shift and, on the other hand, if $\beta = 1$ then there is no transformation in the training dataset. The new set $v^{'}$ is added to $\mathfrak{d}^{'}$. In the second case, if ω is not in Ω, the original set v is added to $\mathfrak{d}^{'}$ at line 11. Finally, the function returns the new dataset at line 14.

7.1.4 Accuracy measures

The strategies are compared in two different contexts: classification and quantification. And different criteria for the accuracy evaluation are used according to the context.

1. In classification, the F_1 is used as accuracy criterion.

2. In quantification, the mean square error (mse) is used. For instance if

$$P(\widetilde{Y} = \cdot) = (0.1, 0.3, 0.6) \qquad \text{and} \qquad \widehat{P}(\widetilde{Y} = \cdot) = (0.4, 0.2, 0.4)$$

   then

$$\text{mse} = (0.1 - 0.4)^2 + (0.3 - 0.2)^2 + (0.6 - 0.4)^2 = 0.14.$$

As an indication, table 7.2 gives the mse quantification error that we obtain if we take

$$\widehat{P}(Y = \omega) = N_{\mathfrak{d}}^{\omega} / N_{\mathfrak{d}}$$

as estimation of $P(\widetilde{Y} = \omega)$. This represents a naive lower bound because we actually do nothing. We just take the empirical proportions $N_{\mathfrak{d}}^{\omega} / N_{\mathfrak{d}}$ in the training population and use

β	mse of $N_{\mathfrak{d}}^{\omega}/N_{\mathfrak{d}}$
0.1	0.22732
0.2	0.13337
0.3	0.07989
0.4	0.04752
0.5	0.02811
0.6	0.01561
0.7	0.00805
0.8	0.00311
0.9	0.00081

Table 7.2: Quantification context with mean square error (the lower the better) on 25 datasets. Mistake made when $N_{\mathfrak{d}}^{\omega}/N_{\mathfrak{d}}$ is used to estimate $P(\widehat{Y} = \omega)$.

these proportions in the testing environment as estimation of the prevalence of the output variable $P(\widetilde{Y} = \omega)$.

When β reduces, the intensity of the prior probability shift increases. As expected, when β is close to one, the error is very low. At the limit and by definition, when $\beta = 1$ then we know the the mean square error is null. The values in this table will be used as indication measures for the experimental results in the following section.

7.2 Results

This section discuss the results of the experiment described in the previous section. The appendix A contains all the figures with the detailed results of the different methods applied on the 25 datasets and this for quantification and classification problems.

In the previous section, we have seen that both *acc* and *cde* can not be applied in any cases. In certain circumstances, *acc* cannot compute a solution because it has to invert a singular matrix or, even when a solution is computed, we are not sure that the solution is a probability (e.g. values can be out of $[0, 1]$). For *cde*, solutions can be calculated only for binary classification problems $|\mathcal{Y}| = 2$.

As only *em* methods can be used in any cases, we will start by comparing the *em* and the *em-stop* algorithms with the 3 benchmark methods on the 25 datasets. Afterwards, the *acc* method will be consider and we will finish this section by assessing *cde* strategy on binary classification problems.

In the next subsections, tables will allows us to evaluate the relative performance of the adaptive strategies. For each value of β, for each adaptive strategy and for each learning algorithm (logistic or random forest), the value in a cell gives the average computed on 25 accuracy measures (mse from quantification or F1-score for classification). For each value of β and for each learning algorithm, the best adaptive strategy is underlined. A paired t-test, on 25 measures, is used to compare the performance of the best strategy against the performances of the other strategies. A cell is in bold if the p-value is bigger than .05.

7.2.1 The EM methods versus the benchmark methods

The methods *em* and *em-stop* are evaluated on the 25 datasets (table 7.1) with *oracle-bayes*, *oracle* and *base* as benchmark methods. The methods are evaluated in two contexts.

- First we evaluate the capacity of the methods to do quantification. As *oracle-bayes* and *oracle* know the prior probability in the testing set, these two methods are excluded

from these evaluation. The results are in table 7.3. Mean square error (lower better) is used in this context as accuracy measure. The values in the cells of the table are averages computed on 25 values.

- Once the quantification is done, the model trained on $\mathfrak{d}'$ is recalibrated[4] for the testing environment. The results are in table 7.4. The five methods are evaluated and the $F1$ score is used as accuracy measure (higher better). The values in the cells are averages computed on 25 values.

	Logistic			Random forest		
β	em	em-stop	base	em	em-stop	base
0.1	0.07731	**0.02451**	0.08904	**0.01344**	**0.02376**	0.08969
0.2	0.05242	**0.01553**	0.05177	**0.00866**	**0.01072**	0.05158
0.3	0.04309	**0.01225**	0.03146	**0.00588**	**0.00587**	0.03168
0.4	0.03720	**0.01136**	**0.01895**	**0.00441**	**0.00385**	0.01903
0.5	0.03056	**0.00904**	**0.01122**	**0.00351**	**0.00264**	0.01121
0.6	**0.02747**	**0.00704**	**0.00616**	0.00317	**0.00210**	0.00617
0.7	**0.02639**	**0.00684**	**0.00312**	0.00275	**0.00179**	0.00312
0.8	**0.02334**	**0.00571**	**0.00123**	**0.00235**	**0.00164**	**0.00122**
0.9	**0.02299**	**0.00563**	**0.00030**	0.00215	0.00156	**0.00030**

Table 7.3: Quantification context with mean square error (lower better) on 25 datasets. The best method is underlined and methods that are not significantly worse than the best (p-val > 0.05 with paired t-test) are in **bold**.

	Logistic					Random forest				
β	em	em-stop	oracle-bayes	oracle	base	em	em-stop	oracle-bayes	oracle	base
0.1	0.5484	**0.7302**	0.7899	0.8442	**0.7056**	**0.7656**	**0.7629**	0.8219	0.8680	0.7076
0.2	0.6346	**0.7748**	0.8215	0.8452	**0.7722**	**0.8006**	**0.8125**	0.8432	0.8676	0.7877
0.3	0.6710	**0.7897**	0.8311	0.8451	**0.8027**	**0.8201**	**0.8309**	0.8515	0.8672	**0.8200**
0.4	0.6931	**0.7945**	0.8356	0.8448	**0.8184**	**0.8308**	**0.8397**	0.8564	0.8682	**0.8381**
0.5	0.7118	**0.8028**	0.8391	0.8458	**0.8268**	**0.8392**	**0.8472**	0.8609	0.8678	**0.8496**
0.6	0.7248	**0.8147**	0.8409	0.8456	**0.8327**	0.8445	**0.8515**	0.8626	0.8676	**0.8564**
0.7	0.7324	**0.8153**	0.8421	0.8455	**0.8371**	0.8474	0.8533	0.8646	0.8679	**0.8615**
0.8	0.7389	0.8210	0.8431	0.8450	**0.8405**	0.8500	0.8548	0.8653	0.8669	**0.8633**
0.9	0.7428	0.8234	**0.8452**	0.8457	**0.8436**	0.8526	0.8572	**0.8668**	0.8673	**0.8657**

Table 7.4: Classification context with F1-score (higher better) on 25 datasets. The best method among *em*, *em-stop*, and *base* is underlined and methods that are not significantly different than the best (p-val > 0.05 with paired t-test) are in **bold**.

Concerning the quantification problem, in both cases logistic and random forest, when β is close to 1, the *base* method outperform the two other methods (table 7.3). This can be explained by the fact that when β is close to 1 then the a priori of the *base* method (i.e. $P(Y = \cdot)$) is close to the true value (i.e. $P(\widetilde{Y} = \cdot)$). The two other methods (*em* and *em-stop*) try to adjust the prior probability of the output from the training environment to the testing one. By doing this adjust, it can happen that the algorithms *em* and *em-stop* are moving away a bit from the true prior distribution $P(\widetilde{Y} = \cdot)$. We have already observed this issue in the right of the figure 6.1 at page 51 where the evolution of the prior probability during the iterations of the EM algorithm is showed when a logistic model is used. In the right of the figure 6.1, the EM algorithm is initialized with the true prior values computed on the testing and, instead of staying stable, the values of the four priors have a little diverged.

[4]Except *base* which is, by definition, not recalibrated.

This can probably be explained by the fact that the EM algorithm assume that the CPE model $g_y(x)$ estimates perfectly the unknown conditional probability $P(Y = y|X = x)$. As there is always an error in the estimate of $P(Y = y|X = x)$, this can explain why *em* and *em-stop* are less accurate than the *base* method when β is close to 1. We can even see that when logistic models are used, the *base* method outperform (not significantly) the two other methods already from $\beta = 0.6$. When random forest models are used, the *base* method outperform (not significantly) the two other methods at $\beta = 0.8$. It can be explained by the fact that random forest models tend to outperform in accuracy the logistic models on real dataset with lot of no linearity.

As we can see, if we compare the results in table 7.3 with the table 7.2 at page 66, except when β equals 0.9, all the methods are better than $N_{\mathfrak{d}}^{\omega}/N_{\mathfrak{d}}$. It was expected that when β equals 0.9, the mse of the estimation $N_{\mathfrak{d}}^{\omega}/N_{\mathfrak{d}}$ is very good because we know that at the limit when $\beta = 1$, the mse error is zero.

Still in the table 7.3, the *em-stop* adjustment method outperform the two other methods when logistic models are used and when β is small. The *em-stop* method has a mechanism that prevents the EM algorithm from diverging to inaccurate values. The EM algorithm assume that $g_y(x)$ is a good estimator for $P(Y = y|X = x)$ but as logistic models are intrinsically linear models followed by a logistic transformation and they separate the classes by hyperplanes (see the softmax equation (3.18) at page 32 where $\gamma(x) = x$), they are not able to capture the non-linear relationships between the input/output variables that we can have in real data sets like the ones in table 7.1. The logistic models tend to underfit (page 22) the training data set and therefore a predictive model generated from this family of models can be a bad predictor. The EM algorithm can then easily diverge with logistic models and that could be an explanation why *em-stop* has better quantification accuracy scores because it can stop this divergence.

On real data sets containing non-linear input/output relationships, a predictive model computed from the family of the random forest models is usually a better estimator than a predictive model extracted from the family of logistic models. We can see that the *em* method with random forest gives better results. When β equals 0.1 or 0.2, *em* with random forest is even better then *em-stop* (but not significantly). That can be explained by the fact that, by preventing the EM algorithm to diverge to an accurate prior value, the *em-stop* avoids the estimation $\widehat{P}(\widetilde{Y} = \cdot)$ to be far from $P(Y = \cdot)$. As the distance between the prior probabilities of the output variable in the training and the testing is maximal when β is small, it explains why *em* outperform *em-stop* when β equals 0.1 or 0.2.

We have also evaluated the three methods (*em*, *em-stop* and *base*) in the quantification context with non-parametric statistical tests. We studied the three methods in the cases when logistic models are used, when random forest models are used and when both types of models are used. The evaluations were done for all β values at once. This means that in the two first cases, the evaluations are done with 225 values (i.e., 25 datasets $\times$ 9 values for β). In the last case, when both types of models are used, the evaluations are done with 450 values (i.e., 2 types of models $\times$ 225). The results are given in table 7.5. As we can see, if we do an overall analysis over the β values, the Friedman test detects significant differences in the accuracies in the three contexts (*Logistic*, *Random forest* and *Both*). If we look at the averages and median, we can see that *em-stop* overperforms the other methods in all the cases except for the *random forest* scores aggregated by the mean where *em* overperforms. If we consider the p-values of the Nemenyi and the Wilcoxon tests, we can see that (i) in all the cases, the *em-stop* method significantly overperforms the *base*, (ii) the *em* method

	Logistic			Random forest			Both		
	em	em-stop	base	em	em-stop	base	em	em-stop	base
Mean	0.0379	0.0109	0.0237	0.0051	0.0060	0.0238	0.0215	0.0084	0.0237
Median	0.0036	0.0025	0.0066	0.0016	0.0014	0.0065	0.0019	0.0018	0.0065
Friedman test (p-val)	= 2.031e-09			< 2.2e-16			< 2.2e-16		

	Logistic		
	em - base	em-stop - base	em-stop - em
Nemenyi test (p-val)	0.0412	0.0002	0.2877
Wilcoxon test (p-val)	0.0369	2.1e-05	0.2401

	Random forest		
	em - base	em-stop - base	em-stop - em
Nemenyi test (p-val)	1.2e-13	7.6e-12	0.8344
Wilcoxon test (p-val)	5.2e-14	1.8e-12	0.5358

	Both		
	em - base	em-stop - base	em-stop - em
Nemenyi test (p-val)	2.2e-12	8.1e-14	0.8851
Wilcoxon test (p-val)	9.6e-12	1.2e-15	0.9124

Table 7.5: Comparisons of the 3 methods (*em*, *em-stop* and *base*) in the quantification context (mse) with non-parametric statistical tests.

significantly overperforms the *base* in all the cases except for *Logistic* where the two p-values stay close to 0.05, (iii) we never get significantly different performances between *em* and *em-stop*.

In the classification context (table 7.4), as expected the benchmark *oracle* method outperforms all the other methods. Rather than using $\mathfrak{d}'$ (line 4 of algorithm 3 at page 64) for the parametric identification, this method uses, the original training data set $\mathfrak{d}$ without prior probability shift (line 11 of algorithm 3). It means that for *oracle*, the observations in the training and in the testing sets are drawn from the same distribution. It is not the case for all the other methods where the parametric identification is done with $\mathfrak{d}'$.

If we compare *em* and *em-stop*, we see in table 7.4 that *em-stop* outperforms all the times *em* except from random forest with $\beta = 0.1$. We can do the same remark as for quantification learning: the *em-stop* method outperforms more significantly the *em* adjustment method when an inaccurate predictive model is used like logistic models. But the advantage of *em-stop* is reduced when an accurate predictive model is used with a small β.

The figure 7.1 shows the evolution of the number of loops done by the *em-stop* method. As we can see the *em-stop* method authorize more loops when random forest are used. It can be explained by the fact that, on real non-linear data sets, the predictive models generated from the family of random forest are often more accurate than the ones generated from the family of logistic models. When a predictive model is a bad estimator of the unknown target function, the EM method tends to diverge to a bad estimation of the probability $P(\widetilde{Y} = \cdot)$ and *em-stop* tries to detected this divergence and stops the iterations. That explain why *em-stop* authorize less loops when logistic models are used.

From figure 7.1, we can see that when β increases, the average of the number of loops in *em-stop* increase also. When β is close to one, the shift introduced in the prior probability is low. That mean that the distance between the source and target environment is relatively small and therefore *em-stop* authorize more iterations.

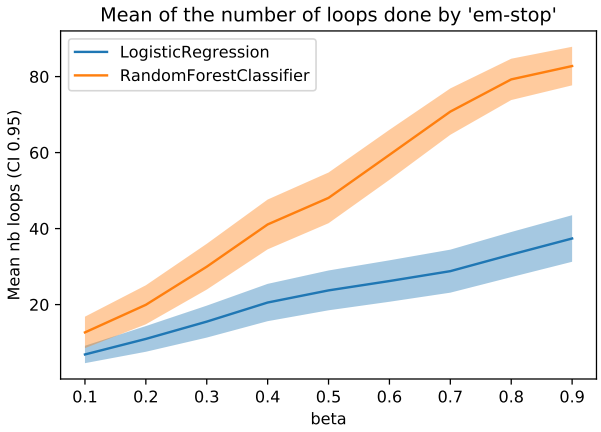

Figure 7.1: The evolution of average (with 0.95 confidence interval) of the number of loops done by the *em-stop* adjustment method in function of the intensity of the prior probability shift β and the type of model Λ. The average is computed across all the loops done on all the 25 data sets presented in table 7.1.

7.2.2 The ACC method and singular matrices

As we know that the ACC method may have trouble to estimate the new prior in the testing environment, we start by counting the number of times the the ACC method was not able to estimate $P(\widetilde{Y} = \cdot)$ The results are given in the table 7.6. This table gives the details for each dataset $\mathfrak{D}$, for each learning algorithm and for each value of β. ACC method can have two type of problems. Either the method can not invert the matrix or the method returns a probability vector with values smaller than zero or greater than one. For each combination for $\mathfrak{D}$, model and β , the first value in the table gives the number of times that the matrix was singular and the second value gives the number of times that the probability vector was not compliant.

As we can see, the method produces a very large number of errors. There was no mistake only with the dataset 3 (*Banknote authentication*). On this dataset, the ACC method was able to do the quantification in all situations (for any β and for any type of model (logistic or random forest)).

The ACC method and random forest model did no mistakes with three additional datasets: 14 (*Letter Recognition*), 18 (*Occupancy Detection*) and 20 (*Pen-Based Recognition of Handwritten Digits*). There are no additional case where the ACC method with the logistic model do no errors.

Concerning the value of β, we observe that there is more singular matrix issue when β is small. This can be explained as follows. When β is small, the method modify_prior_proba will remove a lot of observations from some modalities. When the model learned on $\mathfrak{d}'$ is tested on $\mathfrak{t}$, some of the modalities will never be predicted. Consequently, some column in the confusion matrix contain only zeros (e.g. equation (4.6) at page 40). This problem occurs more frequently when the training set is unbalanced and it is the case with small values of β.

The table 7.6 has shown that the classical version of the ACC has often problem on real data sets. We will now use the modified version of ACC that we proposed in the section 4.3.

$\mathfrak{D}$	Model	$\beta = 0.1$		$\beta = 0.2$		$\beta = 0.3$		$\beta = 0.4$		$\beta = 0.5$		$\beta = 0.6$		$\beta = 0.7$		$\beta = 0.8$		$\beta = 0.9$	
1	L	76	24	58	42	38	62	23	77	9	91	1	98	0	99	0	97	0	98
	RF	18	82	0	100	0	99	0	99	0	98	0	100	0	100	0	99	0	83
2	L	44	41	72	28	87	13	87	13	97	3	98	2	100	0	100	0	100	0
	RF	27	71	18	72	3	97	3	97	0	100	0	100	0	100	0	100	0	100
3	L	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	RF	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	L	30	70	14	58	6	47	4	58	0	90	0	100	0	100	0	100	0	100
	RF	8	89	0	59	0	55	0	88	0	99	0	100	0	100	0	100	0	100
5	L	80	20	82	18	87	12	88	12	74	26	59	39	42	56	25	73	8	83
	RF	71	25	77	20	61	37	44	49	21	69	4	81	3	82	0	88	1	71
6	L	0	100	0	89	0	68	0	100	0	100	0	100	0	100	0	100	0	100
	RF	4	96	0	58	0	89	0	100	0	100	0	100	0	100	0	100	0	100
7	L	71	7	10	30	4	10	0	0	0	1	0	0	0	0	0	0	0	0
	RF	84	4	48	31	16	32	3	27	0	24	0	15	0	7	0	2	0	0
8	L	95	5	94	6	86	14	81	19	80	20	75	25	80	20	76	24	78	22
	RF	100	0	97	3	95	5	95	5	97	3	94	6	98	2	96	4	97	3
9	L	93	7	87	13	84	15	79	21	86	14	85	15	91	9	85	15	83	17
	RF	85	15	71	26	68	32	72	28	54	46	57	43	53	47	41	59	29	71
10	L	46	54	22	74	6	69	2	92	2	98	1	99	6	94	0	100	0	100
	RF	32	68	8	78	0	55	2	65	0	90	0	97	5	95	3	97	0	100
11	L	0	44	0	100	0	100	0	100	0	100	0	100	0	100	0	100	0	100
	RF	0	44	0	56	0	94	0	100	0	100	0	100	0	100	0	100	0	100
12	L	83	16	69	20	43	44	14	52	6	68	0	54	1	40	0	34	0	16
	RF	81	18	59	28	26	46	3	44	1	28	0	7	0	10	0	0	0	2
13	L	70	12	27	46	13	67	5	62	0	44	0	28	0	24	0	2	0	1
	RF	43	22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	L	0	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	RF	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	L	0	100	0	43	0	47	0	52	0	47	0	52	0	43	0	43	0	100
	RF	0	100	0	50	0	54	0	51	0	40	0	0	0	0	0	0	0	0
16	L	0	100	0	74	0	6	0	0	0	0	0	0	0	0	0	0	0	0
	RF	0	100	0	71	0	19	0	1	0	0	0	0	0	0	0	0	0	0
17	L	69	23	8	30	0	8	0	0	0	1	0	0	0	0	0	0	0	0
	RF	94	5	25	47	6	38	0	32	0	2	0	1	0	0	0	0	0	0
18	L	0	31	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	RF	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	L	67	33	47	53	29	71	23	77	14	86	12	88	9	91	6	94	3	97
	RF	49	51	27	73	9	91	11	89	1	99	0	100	0	100	0	100	0	100
20	L	0	56	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	RF	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21	L	0	68	0	49	0	38	0	17	0	9	0	4	0	1	0	0	0	0
	RF	90	10	37	63	6	94	0	92	0	62	0	38	0	35	0	23	0	5
22	L	28	72	21	79	9	91	3	97	0	100	0	100	0	100	0	100	0	100
	RF	50	46	53	47	39	61	33	67	27	73	10	90	6	94	6	94	0	100
23	L	0	53	0	11	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	RF	0	49	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
24	L	0	62	0	48	0	9	0	0	0	0	0	0	0	0	0	0	0	0
	RF	0	99	0	48	0	21	0	0	0	0	0	0	0	0	0	0	0	0
25	L	12	16	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	RF	75	18	6	17	0	1	0	0	0	0	0	0	0	0	0	0	0	0

Table 7.6: Count the number of times that ACC was not able to produce a prediction.

The results are in tables 7.7 and 7.8. For these experiments, we tested only three methods (*em-stop*, *base* and *acc-qp*), the following three data sets where removed [5]: *Default of credit card clients*, *HTRU2* and *Magic Gamma Telescope* and nb_loops is set to 50 (rather than 100, like before). Compared to the two other methods, *acc-qp* is never the best and it is

	Logistic			Random forest		
β	em-stop	base	acc-qp	em-stop	base	acc-qp
0.1	**0.02308**	0.09042	**0.08148**	**0.02341**	0.08835	**0.05566**
0.2	**0.01574**	0.05137	0.03848	**0.01046**	0.05165	0.06136
0.3	**0.01277**	0.03169	0.04630	**0.00570**	0.03162	0.03639
0.4	**0.01185**	**0.01907**	0.04979	**0.00365**	0.01915	0.02310
0.5	**0.00917**	**0.01128**	0.04751	**0.00267**	0.01125	0.01428
0.6	**0.00782**	**0.00618**	**0.04917**	**0.00210**	0.00614	0.04607
0.7	**0.00605**	**0.00314**	**0.02037**	**0.00184**	0.00316	0.01650
0.8	**0.00674**	**0.00122**	**0.04805**	**0.00153**	**0.00122**	0.01631
0.9	**0.00622**	**0.00030**	**0.01960**	0.00145	**0.00030**	0.01532

Table 7.7: Quantification context with mean square error (lower better) on 22 datasets. *acc-qp* stand for the *adjusted classify and count* method based on a quadratic program with constraints (section 4.3 at page 37). The best method is underlined and methods that are not significantly worse than the best (p-val > 0.05 with paired t-test) are in **bold**.

	Logistic			Random forest		
β	em-stop	base	acc-qp	em-stop	base	acc-qp
0.1	**0.7293**	0.6956	0.6775	**0.7591**	0.7019	**0.7437**
0.2	**0.7715**	**0.7694**	0.7252	**0.8100**	0.7826	0.7877
0.3	**0.7881**	**0.8019**	0.7647	**0.8277**	**0.8170**	0.8130
0.4	**0.7910**	**0.8169**	0.7858	**0.8386**	0.8366	0.8257
0.5	**0.8042**	**0.8282**	0.7948	**0.8455**	**0.8479**	0.8328
0.6	**0.8092**	**0.8324**	0.8002	**0.8489**	**0.8531**	0.8397
0.7	0.8165	**0.8358**	0.8089	0.8511	**0.8586**	0.8452
0.8	0.8178	**0.8400**	0.8137	0.8539	**0.8614**	0.8490
0.9	0.8219	**0.8429**	0.8178	0.8557	**0.8635**	0.8506

Table 7.8: Classification context with F1-score (higher better) on 22 datasets. *acc-qp* stand for the *adjusted classify and count* method based on a quadratic program with constraints (section 4.3 at page 37). The best method is underlined and methods that are not significantly worse than the best (p-val > 0.05 with paired t-test) are in **bold**.

also often significantly worst. The gap between *acc-qp* and the two other methods is bigger when β is close to one.

If we compare these results with the table 7.2 at page 66. We can see that although the results in table 7.7 are not very good, they are still better than $N_{\mathfrak{d}}^{\omega}/N_{\mathfrak{d}}$ when β is small. When logistic models are used, the mse of *acc-qp* is better than table 7.7 when $\beta \leq 0.3$ and when random forest models are used, the mse is better when $\beta \leq 0.5$.

7.2.3 The CDE method when $|\mathcal{Y}| = 2$

In this section, we compare the CDE method, the EM methods and the benchmark methods on the 11 binary classification problems. As before, we consider two learning problems. First we consider the quantification learning problem where $P(\widetilde{Y} = \cdot)$ in the testing environment must be estimated. The results of the quantification learning problem are in table 7.9. In a second time, we consider the binary classification learning problem where the estimation of

[5] Because they are too big and require a lot of computing power to be evaluated

$P(\widetilde{Y} = \cdot)$ computed at the previous step is used with equation (3.15) at page 28 to recalibrate the predictive model. We recall that for the binary classification learning problem, the *oracle-bayes* method do not use an estimation $\widehat{P}(\widetilde{Y} = \cdot)$ but it uses directly the true value of $P(\widetilde{Y} = \cdot)$ and the *oracle* method do not need to do recalibration because it uses for $\mathcal{I}^P$ the original training set before prior probability shift is introduced. The results of the binary classification learning problem are in table 7.10.

	Logistic				Random forest			
β	cde	em	em-stop	base	cde	em	em-stop	base
0.1	**0.09565**	**0.06808**	**0.03461**	0.14764	0.08924	**0.01899**	0.04335	0.15027
0.2	**0.07492**	**0.03246**	**0.01929**	0.08572	0.04361	**0.01272**	**0.01881**	0.08533
0.3	**0.02956**	**0.02031**	**0.01220**	0.05176	0.02509	**0.00876**	**0.01025**	0.05209
0.4	0.02387	**0.01155**	**0.00798**	0.03136	0.01547	**0.00626**	**0.00647**	0.03122
0.5	0.02272	**0.00806**	**0.00605**	0.01840	**0.00993**	**0.00465**	**0.00412**	0.01847
0.6	0.02224	**0.00815**	**0.00515**	**0.01014**	0.00659	**0.00444**	**0.00317**	0.01016
0.7	0.02202	**0.00623**	**0.00389**	**0.00506**	0.00507	**0.00352**	**0.00255**	**0.00504**
0.8	0.02207	**0.00619**	0.00280	**0.00199**	**0.00405**	0.00296	**0.00227**	**0.00200**
0.9	0.02194	**0.00465**	**0.00239**	**0.00046**	0.00332	**0.00278**	**0.00220**	**0.00046**

Table 7.9: Quantification context with mean square error (lower better) on 11 datasets. *cde* stand for the *class distribution estimation* method (chapter 5 at page 43). The best method among *em*, *em-stop*, and *base* is underlined and methods that are not significantly worse than the best (p-val > 0.05 with paired t-test) are in **bold**.

	Logistic					
β	cde	em	em-stop	oracle-bayes	oracle	base
0.1	**0.68472**	**0.74949**	**0.77119**	0.83591	0.84518	**0.70858**
0.2	**0.73665**	**0.79947**	**0.80815**	**0.84121**	0.84537	**0.77829**
0.3	0.79353	**0.81792**	**0.82367**	**0.84306**	**0.84557**	**0.81599**
0.4	**0.80415**	**0.82937**	**0.83473**	**0.84380**	**0.84565**	**0.83296**
0.5	**0.80770**	**0.83513**	**0.83865**	0.84560	0.84630	**0.83952**
0.6	**0.80832**	**0.83232**	**0.83823**	0.84458	0.84538	**0.84156**
0.7	0.80921	**0.83935**	**0.84374**	**0.84571**	**0.84624**	**0.84537**
0.8	**0.80916**	**0.83759**	**0.84533**	**0.84578**	**0.84612**	**0.84540**
0.9	**0.81020**	**0.84251**	**0.84531**	**0.84627**	**0.84590**	**0.84590**

	Random forest					
β	cde	em	em-stop	oracle-bayes	oracle	base
0.1	0.74452	**0.81298**	0.75855	0.83828	0.86256	0.70959
0.2	0.79380	**0.82491**	**0.81587**	**0.84890**	**0.86184**	0.78116
0.3	0.81635	**0.83740**	**0.83730**	**0.85523**	**0.86241**	0.81532
0.4	**0.83134**	**0.84247**	**0.84512**	0.85664	0.86239	0.83497
0.5	**0.84168**	**0.84815**	**0.85007**	0.85924	0.86225	0.84587
0.6	**0.84792**	**0.84878**	**0.85209**	0.85830	0.86129	**0.85254**
0.7	0.85088	**0.85198**	**0.85392**	0.86015	0.86172	**0.85746**
0.8	0.85138	**0.85402**	**0.85511**	0.86095	0.86211	**0.85912**
0.9	**0.85377**	**0.85540**	**0.85665**	**0.86175**	**0.86243**	**0.86148**

Table 7.10: Classification context with F1-score (higher better) on 11 datasets. *cde* stand for the *class distribution estimation* method (chapter 5 at page 43). The best method is underlined and methods that are not significantly worse than the best (p-val > 0.05 with paired t-test) are in **bold**.

Concerning the quantification learning problem (table 7.9), although *cde* is often better than *base* when β is small, the *cde* adjustment method is never the best one among the four adjustment methods, neither with logistic or with random forest. It is even often also significantly worse than the best method. For quantification, the CDE method is a better adjustment method than the naive *base* strategy but it is beaten by the two EM methods.

It should be mentioned that concerning the mean square error measures computed for the methods *em*, *em-stop* and *base*, the only difference between table 7.3 at page 67 and table 7.9 is that the mse in table 7.3 is the average computed from 25 data sets and in table 7.3 the mse is the average computed from only the 11 binary data sets.

Concerning the binary classification problems (table 7.10), the models recalibrated with prior probability estimation computed by *cde* have bad F1-scores if we compare these F1 accuracies with the one that we obtain from the other strategies. When logistic models are used, *em-stop* outperform the other strategies when β is small and, when β is bigger, *base* becomes the best strategy. With random forest, we obtain the same result except that when $\beta \leq 0.3$ then *em* without stopping becomes better.

7.3 Discussion of the results

In this work, we have proposed two new algorithms for the quantification: *em-stop* and *acc-qp*. The first method tries to avoid situations where the EM method diverges to a wrong solution by stopping prematurely the iterations of the method. The second method proposes to solve the system of linear equations that the ACC method has to resolve by a quadratic program with constraints.

The two proposed methods were experimentally compared against other classical methods and three benchmark methods. These experiments were done on 25 well known real datasets (table 7.1 at page 62) extracted from the *UCI Machine Learning Repository*. To the best of our knowledge, this is the first comprehensive empirical comparison of the main strategies to adjust prior probability shift.

The experiments showed that, compared with *em-stop* and *base*, the *acc-qp* method underperforms significantly. But it should be noted that when β is small then *acc-qp* outperforms the naive method which uses the proportion $N_{\mathfrak{d}}^{\omega}/N_{\mathfrak{d}}$ counted in the source domain as an estimation of $P(\widetilde{Y} = \cdot)$. This bad precision can have several sources. One of them could come from the estimate of the error on the training set done by cross-validation. In this work we did 5-fold cross validations. It is known that this is a biased estimator and that could be one of the reasons explaining the relative poor performance of the *acc-qp* method.

The second method proposed in this work is the *em-stop* adjustment method. Compared to the other methods, it obtained very good results. Performances were relatively worse when an accurate random forest was used with a small value of β. As we already explained, *em-stop* tries to prevent the EM algorithm against instability. Sometimes it happens that it stops too early the EM algorithm when the prior probability shift between the training and the testing populations is high and when an accurate model is used (e.g. random forest).

We saw that for quantification, when β is close to one, it is better to use the simple *base* method. Of course, most of the time we do not know in advance if the level of the prior probability shift between the training and the testing environments is high. Therefore, it would be interesting to study the possibility to use a statistical test like in [38] and, based on p-value, decide to use the *base* method or a more advanced one.

Chapter 8

Conclusion

In realistic settings, the prevalence of the output class may change between the situation when the training set was generated and the situation when the predictive model is used for scoring (i.e. $P(Y = \cdot) \neq P(\widetilde{Y} = \cdot)$). If the within-class probability density is conserved (i.e. $f_{X|Y}(\cdot|\cdot) = f_{\widetilde{X}|\widetilde{Y}}(\cdot|\cdot)$) then we have *prior probability shift* and the task of estimating $P(\widetilde{Y} = \cdot)$ in the target environment is called *quantification* learning.

In chapter 2, we saw that *prior probability shift* can be considered as a particular type of *transfer problem* where the within-class probability density can be transferred from the source environment to the target environment. We did not find, in the current scientific literature, a sufficiently generic definition of transfer learning able to take into account all the types of transfer learning. Therefore, as contribution, we have proposed a new generic definition for transfer learning (page 11).

If the purpose is not to estimate the new prevalence of the output variable in the target environment but rather to adjust the classifier to the new environment than section 3.3 gives a method to recalibrate the models to the new distribution. In a still unpublished paper *Adjusting the Bias Term of Classifiers to Unknown Prior* written by Prof. Marco Saerens and Prof. Christine Decaestecker, a method to adjust directly the intercept term of a softmax classifier is proposed. As a contribution, we proposed a new (and more simple) way to adjust this term in presence of prior probability shift (equation (3.20) at page 32).

Three quantification algorithms were studied in this work:

- The adjusted classify and count (*acc*) approach (chapter 4).
- The class distribution estimation (*cde*) approach (chapter 5).
- The expectation-maximization (*em*) approach (chapter 6).

As we saw for instance in (4.9) at page 41, the *adjusted classify and count* method can produce inconsistent results with the definition of a probability. Based on an idea introduced by Prof. Johan Segers, as contribution of this work, we proposed a new version of the original *acc* method based on quadratic program with constraints and called *acc-qp* (section 4.3 at page 37).

As we saw empirically for instance in section 6.1.2 at page 51, the *em* algorithm is an iterative method that can diverge to a wrong solution. A classical method in the scientific literature is to deal with this problem by doing a predetermined small number of iterations of the *em* algorithm. As contribution, we proposed in this work an adaptive method able to stop the *em* iterations when *em* starts to diverge (section 6.2 at page 56) and called *em-stop*.

The three standard quantification algorithms (*acc*, *cde* and *em*) and the two proposed methods (*acc-qp* and *em-stop*) were compared against three benchmark strategies on 25 datasets. *Oracle-bayes* and *oracle* are two methods which are cheating in the sense that they are using information normally not available in real settings. The performances of these two methods are somehow an upper bound for the other methods. *Base* (also called *classify and counting* (*cc*)) is a strategy that estimates the new prevalence by applying (without any adjustment) a classifier learned on data from the source environment on the testing data from the target environment. The estimation of the new prevalence is then obtained by counting the prediction of each class. *Base* represents a lower bound for the five other methods. For reference, table 7.2 at page 7.2 gives the error that we would make if the prevalence in the training was directly used as estimation of the prevalence in the testing set.

The experimental results show that when β is close to one (low prior probability shift), the performances of the naive method *base* are relatively very good. The more complex methods become superior when the intensity of the prior probability shift increases.

Surprisingly, we were not able to obtain a good result with *acc* methods while these methods are often cited in the scientific literature. As future work, it would be interesting to study *acc* algorithms more deeply in order to identify the cases in which they are superior. We proposed in this work *acc-qp* as contribution to solve the issue that the *acc* algorithm can produce invalid solutions. To deal with the issue that acc does not guarantee to return a vector of values in $[0, 1]$ and summing to 1, [16] proposes to *clip* the probability estimations (i.e., equal to 1 every value higher than 1 and to 0 every value lower than 0), and afterwards rescale them so that they sum up to one. As contribution for a future work, it would be interesting to compare empirically this method with our method *acc-qp*. The accuracy of the *acc* method seems to be impacted by the precision of the estimation of the confusion matrix. In this work, $K = 5$ cross validation is used. It would be interesting to study the impact of another estimation method of the confusion matrix.

The *em* like methods are becoming superior when β is decreasing to zero (higher prior probability shift). Future works on *em* should focus on cases with high prior probability shift. When a logistic model is used, the proposed *em-stop* method is superior and when random forest is used, the standard *em* method becomes the best. It seems that when the accuracy of the model is bad (e.g. logistic model on no-linear real datasets), the *em-stop* is able to avoid the adjustment to diverge. When the accuracy of the model is better (e.g. random forest on no-linear real datasets), then the *em-stop* method stops too quickly the iterations of the adjustment algorithm and is therefore beaten by the standard *em*. *Em* seems to be impacted by the accuracy of the model. As a future work, it would be interesting to make a systematic empirical study on this topic.

Appendix A

Detailed experimental results

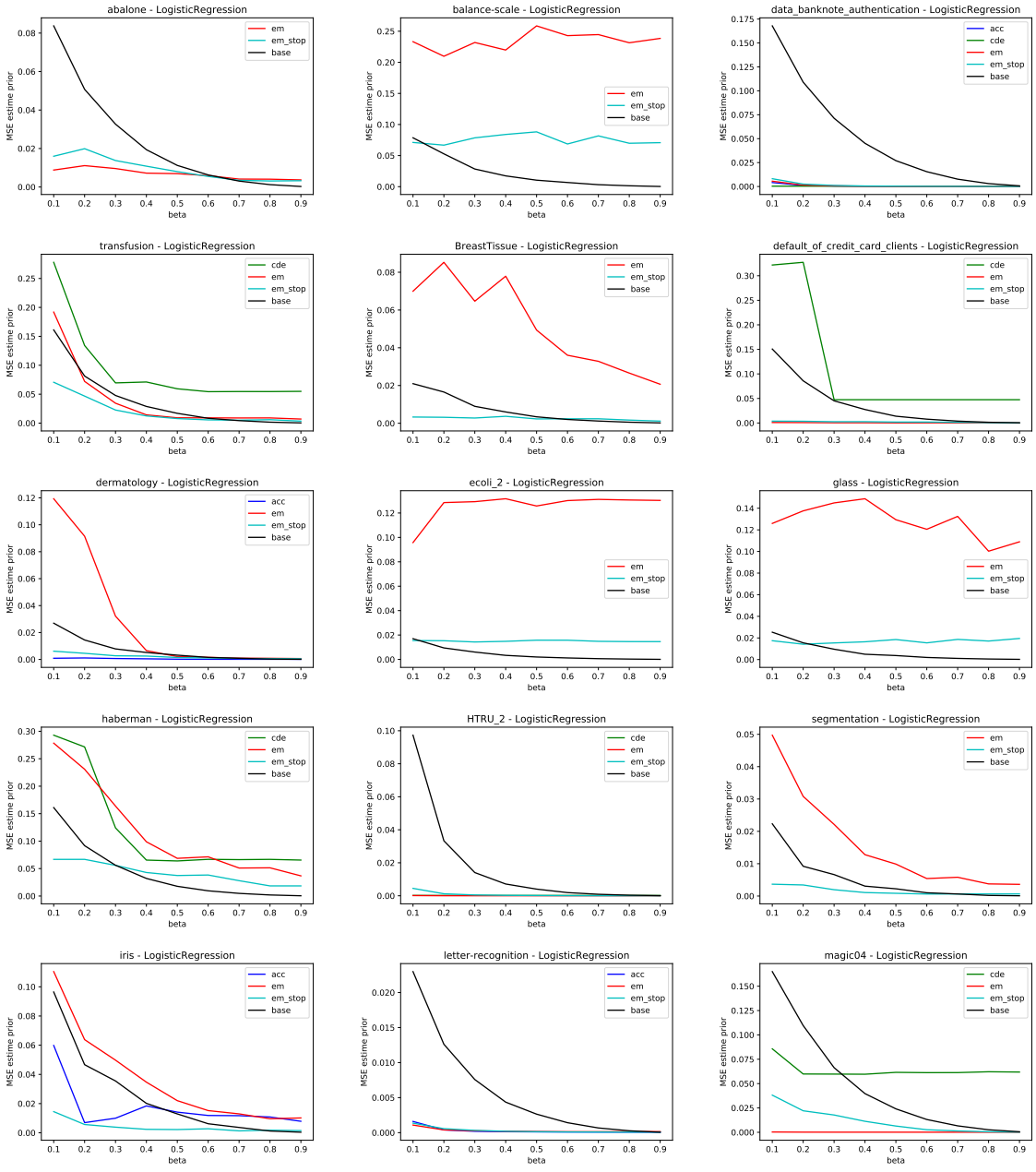

Figure A.1: Quantification error (mse) of **logistic regression** models (part 1)

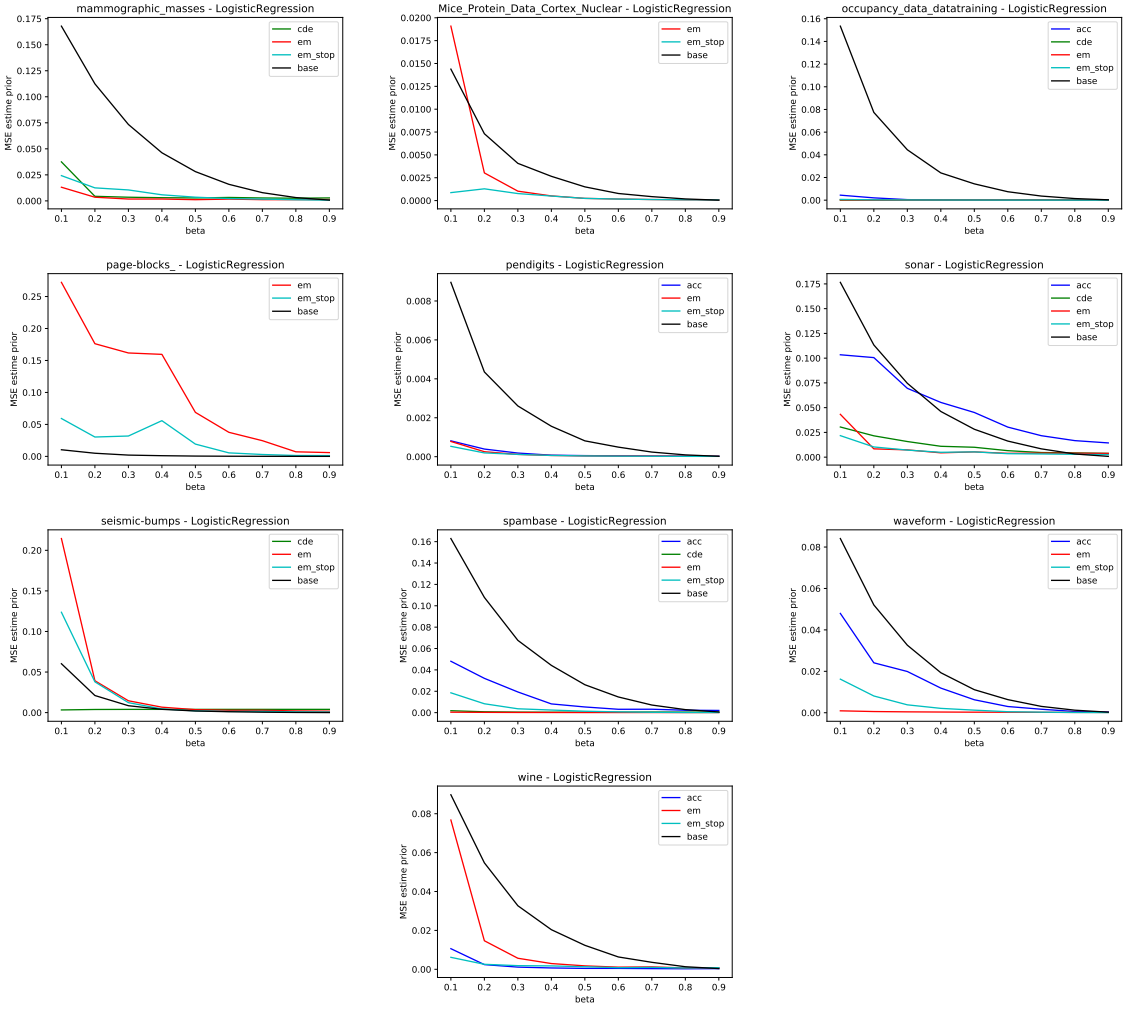

Figure A.2: Quantification error (mse) of **logistic regression** models (part 2)

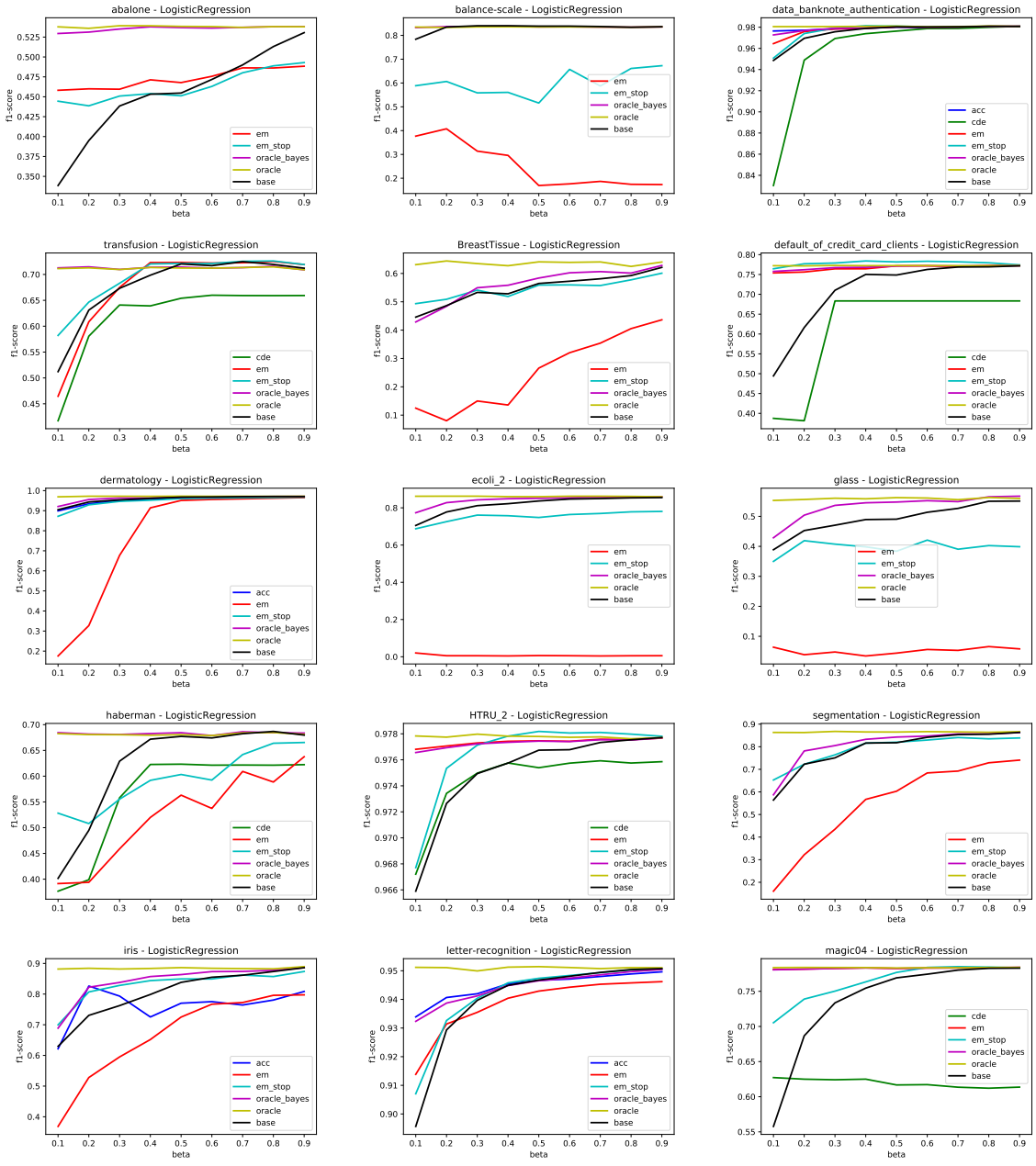

Figure A.3: Classification error (F1 score) of **logistic regression** models (part 1)

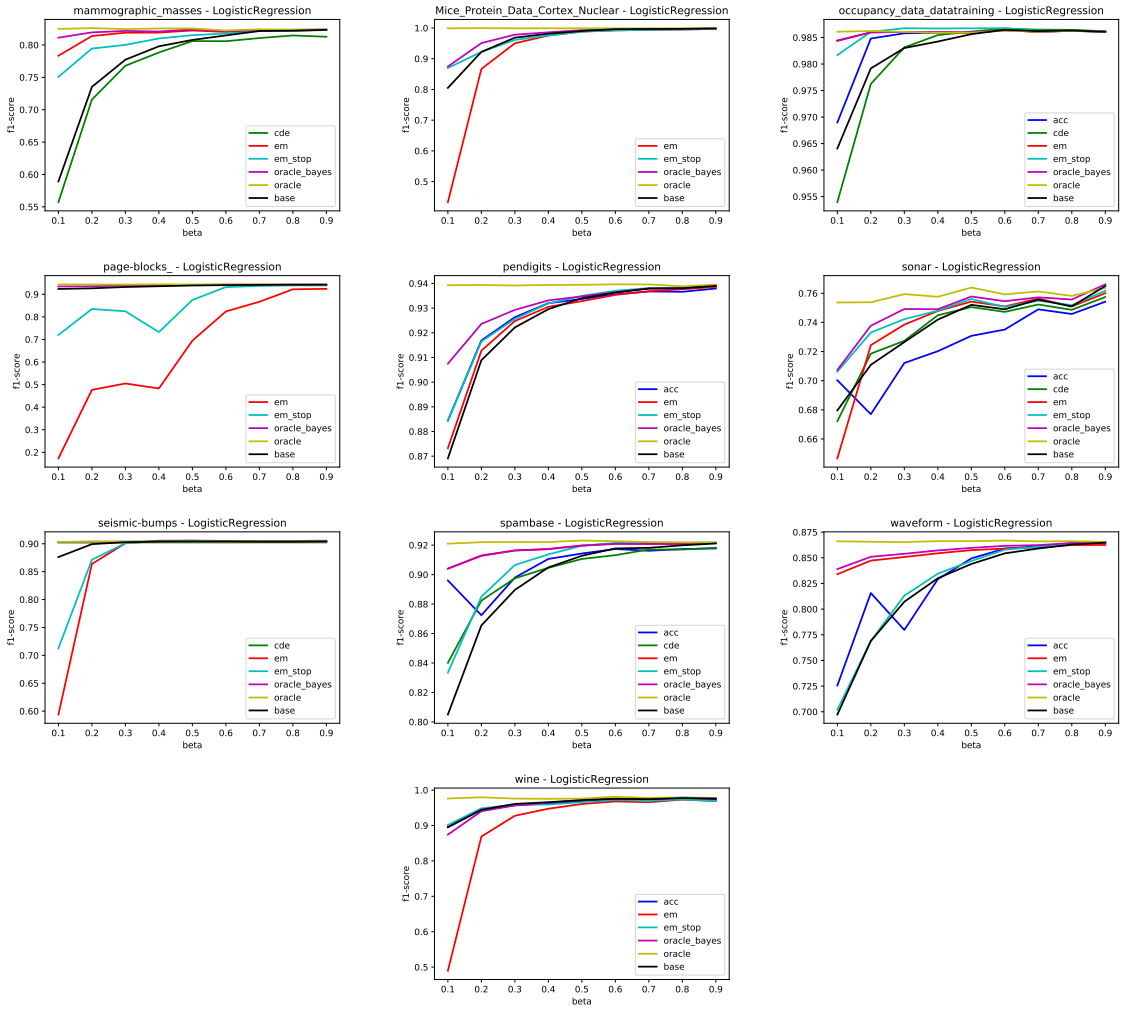

Figure A.4: Classification error (F1 score) of **logistic regression** models (part 2)

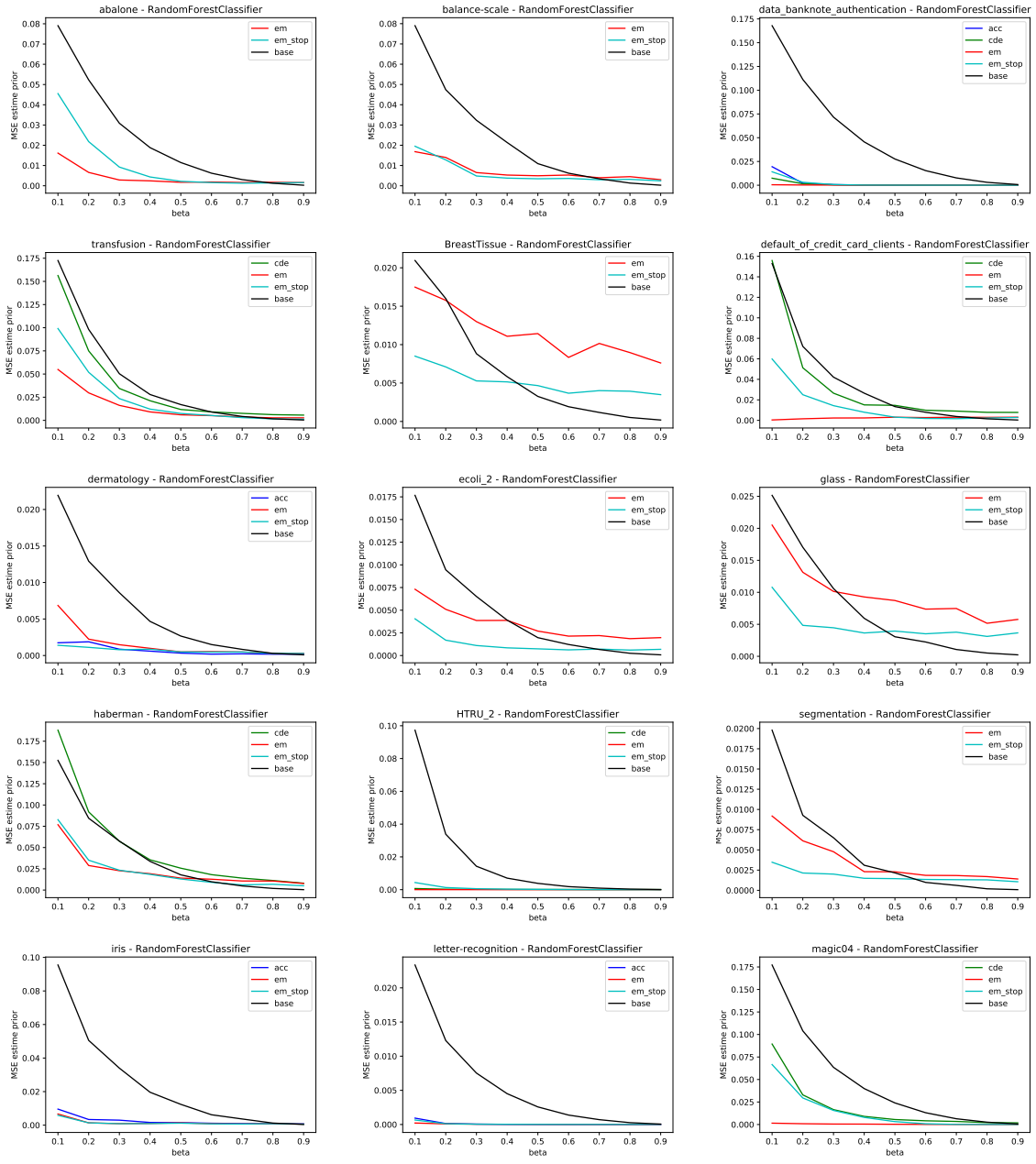

Figure A.5: Quantification error (mse) of **random forest** models (part 1)

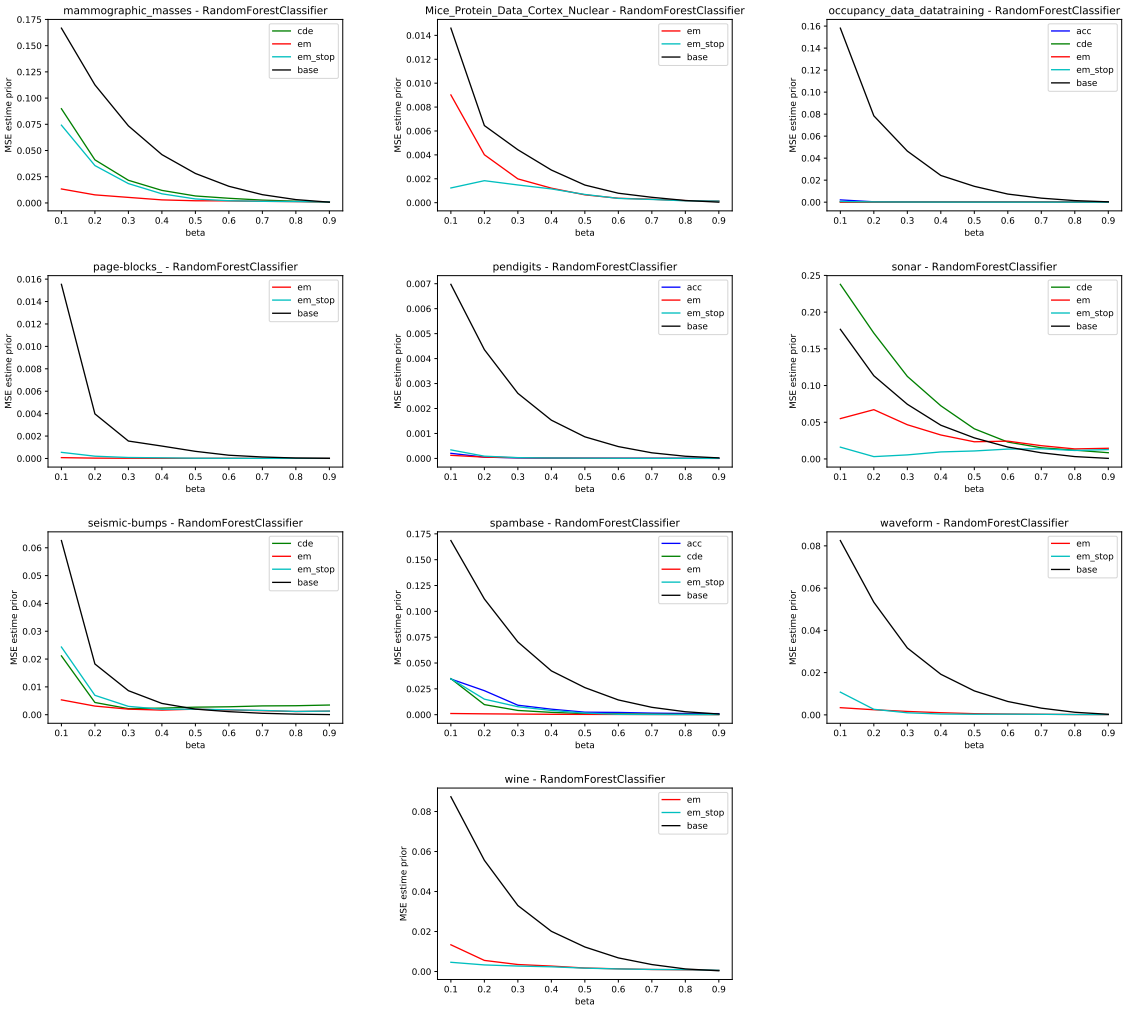

Figure A.6: Quantification error (mse) of **random forest** models (part 2)

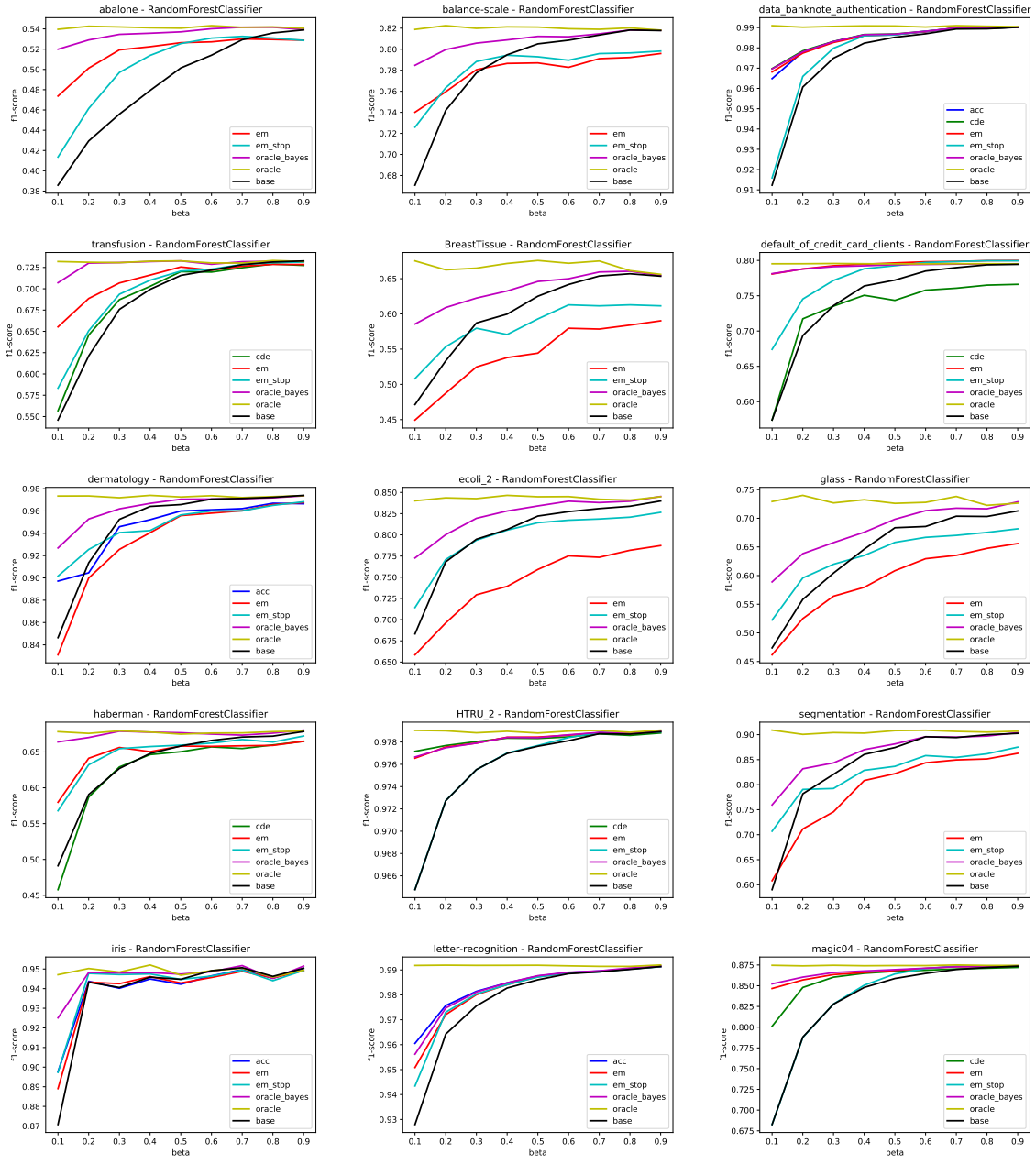

Figure A.7: Classification error (F1 score) of **random forest** models (part 1)

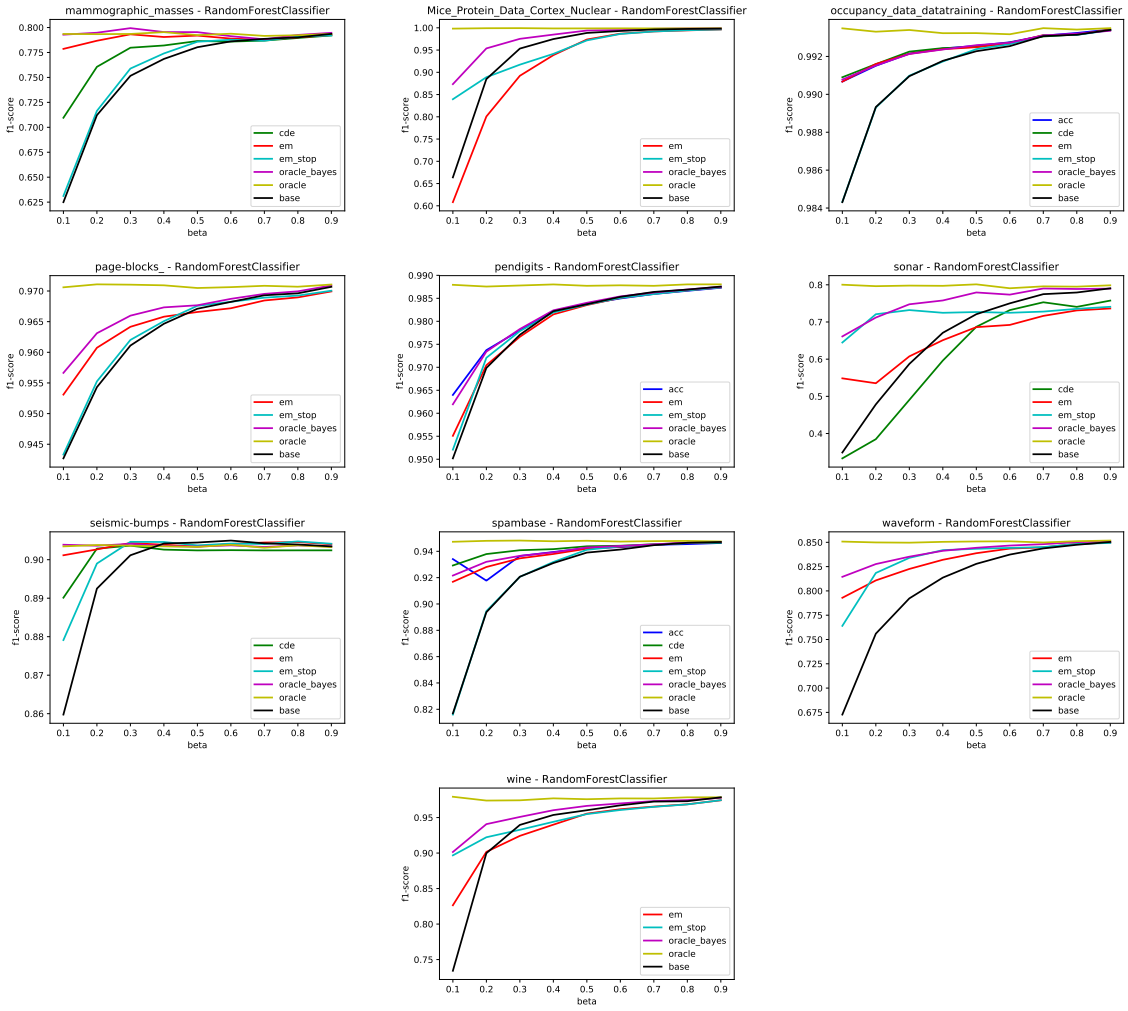

Figure A.8: Classification error (F1 score) of **random forest** models (part 2)

Bibliography

[1] Rocío Alaíz-Rodríguez, Alicia Guerrero-Curieses, and Jesús Cid-Sueiro. Class and subclass probability re-estimation to adapt a classifier in the presence of concept drift. *Neurocomputing*, 74(16):2614–2623, 2011.

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

[3] Christopher Bishop. *Pattern recognition and machine learning*. Springer, 2006.

[4] Gianluca Bontempi and Souhaib Ben Taieb. Statistical foundations of machine learning. *Université Libre de Bruxelles*, 2011.

[5] Olivier Caelen. A bayesian interpretation of the confusion matrix. *Annals of Mathematics and Artificial Intelligence*, 81(3-4):429–450, 2017.

[6] Rich Caruana. Multitask learning. In *Learning to learn*, pages 95–133. Springer, 1998.

[7] Wenyuan Dai, Qiang Yang, Gui-Rong Xue, and Yong Yu. Self-taught clustering. In *Proceedings of the 25th international conference on Machine learning*, pages 200–207. ACM, 2008.

[8] Andrea Dal Pozzolo, Giacomo Boracchi, Olivier Caelen, Cesare Alippi, and Gianluca Bontempi. Credit card fraud detection: a realistic modeling and a novel learning strategy. *IEEE transactions on neural networks and learning systems*, 2017.

[9] Andrea Dal Pozzolo, Olivier Caelen, and Gianluca Bontempi. When is undersampling effective in unbalanced classification tasks? In *Machine Learning and Knowledge Discovery in Databases*, pages 200–215. Springer, 2015.

[10] Andrea Dal Pozzolo, Olivier Caelen, Reid A Johnson, and Gianluca Bontempi. Calibrating probability with undersampling for unbalanced classification. In *Computational Intelligence, 2015 IEEE Symposium Series on*, pages 159–166. IEEE, 2015.

[11] Jesse Davis and Mark Goadrich. The relationship between precision-recall and roc curves. In *Proceedings of the 23rd international conference on Machine learning*, pages 233–240. ACM, 2006.

[12] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society. Series B (methodological)*, pages 1–38, 1977.

[13] Gregory Ditzler, Manuel Roveri, Cesare Alippi, and Robi Polikar. Learning in nonstationary environments: A survey. *IEEE Computational Intelligence Magazine*, 10(4):12–25, 2015.

[14] Richard O Duda, Peter E Hart, and David G Stork. *Pattern classification*. Wiley, 2nd edition, 2001.

[15] Charles Elkan. The foundations of cost-sensitive learning. In *International joint conference on artificial intelligence*, volume 17, pages 973–978, 2001.

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

[17] Alexander Gammerman, Volodya Vovk, and Vladimir Vapnik. Learning by transduction. In *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*, pages 148–155. Morgan Kaufmann Publishers Inc., 1998.

[18] Wei Gao and Fabrizio Sebastiani. From classification to quantification in tweet sentiment analysis. *Social Network Analysis and Mining*, 6(1):19, 2016.

[19] Liang Ge, Jing Gao, Hung Ngo, Kang Li, and Aidong Zhang. On handling negative transfer and imbalanced distributions in multiple source transfer learning. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 7(4):254–271, 2014.

[20] Pablo González, Alberto Castaño, Nitesh V Chawla, and Juan José Del Coz. A review on quantification learning. *ACM Computing Surveys (CSUR)*, 50(5):74, 2017.

[21] Pablo González, Jorge Díez, Nitesh Chawla, and Juan José del Coz. Why is quantification an interesting learning problem? *Progress in Artificial Intelligence*, 6(1):53–58, 2017.

[22] VíCtor GonzáLez-Castro, RocíO Alaiz-RodríGuez, and Enrique Alegre. Class distribution estimation based on the hellinger distance. *Information Sciences*, 218:146–164, 2013.

[23] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference, and prediction*. Springer, 2009.

[24] Vera Hofer and Georg Krempl. Drift mining in data: A framework for addressing drift in classification. *Computational Statistics & Data Analysis*, 57(1):377–391, 2013.

[25] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An introduction to statistical learning*. Springer, 2013.

[26] Imen Khamassi, Moamar Sayed-Mouchaweh, Moez Hammami, and Khaled Ghédira. Discussion and review on evolving data streams and concept drift adapting. *Evolving Systems*, 9(1):1–23, 2018.

[27] Patrice Latinne, Marco Saerens, and Christine Decaestecker. Adjusting the outputs of a classifier to new a priori probabilities may significantly improve classification accuracy: evidence from a multi-class problem in remote sensing. In *ICML*, pages 298–305, 2001.

[28] Nachai Limsetto and Kitsana Waiyamai. Handling concept drift via ensemble and class distribution estimation technique. In *International Conference on Advanced Data Mining and Applications*, pages 13–26. Springer, 2011.

[29] Roderick JA Little and Donald B Rubin. *Statistical analysis with missing data*, volume 333. John Wiley & Sons, 2014.

[30] Jie Lu, Vahid Behbood, Peng Hao, Hua Zuo, Shan Xue, and Guangquan Zhang. Transfer learning using computational intelligence: a survey. *Knowledge-Based Systems*, 80:14–23, 2015.

[31] David JC MacKay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

[32] Geoffrey McLachlan and Thriyambakam Krishnan. *The EM algorithm and extensions*, volume 382. John Wiley & Sons, 2007.

[33] Kevin Murphy. *Machine learning: A probabilistic perspective*. MIT Press, 2012.

[34] Radford M Neal and Geoffrey E Hinton. A view of the em algorithm that justifies incremental, sparse, and other variants. In *Learning in graphical models*, pages 355–368. Springer, 1998.

[35] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2010.

[36] David Martin Powers. Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation. 2011.

[37] Joaquin Quionero-Candela, Masashi Sugiyama, Anton Schwaighofer, and Neil D Lawrence. *Dataset shift in machine learning*. The MIT Press, 2009.

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

[39] Hidetoshi Shimodaira. Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of statistical planning and inference*, 90(2):227–244, 2000.

[40] Amos Storkey. When training and test sets are different: characterizing learning transfer. *Dataset shift in machine learning*, pages 3–28, 2009.

[41] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[42] Dirk Tasche. Fisher consistency for prior probability shift. *The Journal of Machine Learning Research*, 18(1):3338–3369, 2017.

[43] Matthew E Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(Jul):1633–1685, 2009.

[44] S. Theodoridis and K. Koutroumbas. *Pattern recognition*. Academic Press, 4th edition, 2009.

[45] Vladimir Vapnik. *The nature of statistical learning theory*. Springer science & business media, 2000.

[46] Sudhir Varma and Richard Simon. Bias in error estimation when using cross-validation for model selection. *BMC bioinformatics*, 7(1):91, 2006.

[47] Andrew Webb and Keith Copsey. *Statistical pattern recognition*. Wiley, 3rd edition, 2011.

[48] Jack Chongjie Xue and Gary M Weiss. Quantification and semi-supervised classification methods for handling changes in class distribution. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 897–906. ACM, 2009.

[49] Zhihao Zhang and Jie Zhou. Transfer estimation of evolving class priors in data stream classification. *Pattern Recognition*, 43(9):3151–3161, 2010.

[50] Xiaojin Zhu. Semi-supervised learning literature survey. 2005.