COST-SENSITIVE BOOSTING: A UNIFIED APPROACH

A THESIS SUBMITTED TO THE UNIVERSITY OF MANCHESTER
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY
IN THE FACULTY OF SCIENCE & ENGINEERING

2016

By
Nikolaos Nikolaou
School of Computer Science

Contents

Abstract **12**

Declaration **14**

Copyright **15**

Acknowledgements **16**

1 Introduction **18**

1.1 Motivation 18

1.1.1 A data-driven world 18

1.1.2 Making sense of data 19

1.1.3 Asymmetric learning 21

1.1.4 Ensemble learning 21

1.2 Research questions 22

1.3 Outline and contributions 23

1.4 Structure of this thesis 25

1.5 Publications 27

2 Asymmetric Learning **28**

2.1 Minimizing the risk 28

2.2 Cost-sensitive & imbalanced-class learning duality 33

2.3 Problem definition & optimal decision rule 34

2.4 Evaluation measures for asymmetric learning 35

2.4.1 Bad choices: accuracy & error rate 35

2.4.2 Good choices 36

Contigency table measures & ROC analysis 36

Cost curves & Brier curves 38

2.4.3 Solving a different problem: Neyman-Pearson detection . . . 40
2.5 Making learners asymmetric 41
2.5.1 Shifting the decision threshold 41
2.5.2 Manipulating the training data 41
2.5.3 Modifying the model 42
2.5.4 Using an asymmetric meta-technique 42
2.6 Potential complications 43

3 Boosting and its interpretations **45**
3.1 Hypothesis boosting 45
3.2 The AdaBoost algorithm 46
3.2.1 AdaBoost as an ensemble method 48
3.2.2 The successes of AdaBoost 49
3.2.3 Interpretations 50
3.2.4 Limitations 55
3.2.5 Variants of boosting 56
3.3 Cost-sensitive boosting algorithms 58
3.3.1 Modifying the training algorithm 59
CSB0, CSB1 & CSB2 59
Asymmetric-AdaBoost 60
AdaCost & Alternative AdaCost 61
AdaCost(β_1) & AdaCost(β_2) 62
AdaC1, AdaC2 & AdaC3 62
CS-Ada & AdaDB 63
3.3.2 Modifying the weight initialization 64
3.3.3 Modifying the prediction rule 65
3.4 Preview of the rest of the thesis 65

4 The functional gradient descent view **68**
4.1 Boosting as functional gradient descent 69
4.1.1 Example: AdaBoost 70
4.1.2 Why a margin loss? 70
4.2 Uncovering the hidden loss 71
4.2.1 Methods that do not minimize a margin loss 71
4.3 FGD-consistency 72
4.3.1 Checking for FGD-consistency 74

4.3.2 When ‘suboptimal’ steps are better than ‘optimal’ 75

5 The decision-theoretic view 78

5.1 Loss minimizers 79

5.1.1 Minimizer derivation 80

5.2 Cost-consistency 81

5.2.1 Checking for cost-consistency 82

5.3 A generalization of AdaMEC 84

5.3.1 Cost-consistency of AdaMEC and global loss 85

5.3.2 Loss at round t versus global loss 85

6 The margin-theoretic view 88

6.1 Asymmetry-preservation 89

6.1.1 Checking for asymmetry-preservation 90

6.2 Asymmetry swapping and poor generalization 93

7 The probabilistic view 96

7.1 AdaBoost as a Product of Experts 97

7.1.1 A systematic distortion 99

7.2 Cost-sensitive boosting as a Product of Experts 101

7.2.1 A simple example 101

7.2.2 Generalizing to other boosting variants 101

7.2.3 Prior-weighted vs. non-prior-weighted PoE 103

7.2.4 Variants not covered by Theorem 3 104

7.3 Calibration 104

7.3.1 Calibrated probability estimates versus raw scores 104

7.3.2 Calibrating boosting 106

7.4 Satisfying all desirable properties 107

8 Empirical comparison 111

8.1 Experimental setup 111

8.2 Analysis of experimental results 113

8.2.1 Overall trend 113

8.2.2 A deeper examination of AdaMEC, CGAda & AsymAda . . . 114

8.2.3 Examining individual Brier curves 117

8.2.4 Calibrating all variants 123

9 Conclusions and future directions **126**
9.1 Conclusions 126
9.2 Future work 128
9.3 Closing word: an argument for simplicity 134

A Theorem Proofs **136**

B Datasets Used **144**

Bibliography **145**

Word Count: 48579

List of Tables

2.1 General form of a fixed cost matrix in binary classification. 30

2.2 A cost matrix for a binary classification with $c_{TP} = c_{TN} = 0$. 30

2.3 An alternative cost matrix for a binary classification with $c_{TP} = c_{TN} = 0$, with only one degree of freedom. 32

3.1 A summary of cost-sensitive variants showing how they modify the weight updates, along with the weight initialization. The definition of α_t also varies across methods and can be looked up in Table 3.2. The original AdaBoost is included for reference. AdaMEC does not modify any of the training steps of AdaBoost, only its prediction rule. AdaCost(β_1) is the same algorithm as AdaC1. 66

3.2 A summary of cost-sensitive variants showing how they modify the base learner weights α_t. The weight initialization and weight update rule also vary across methods and can be looked up in Table 3.1. The original AdaBoost is included for reference. AdaMEC does not modify any of the training steps of AdaBoost, only its prediction rule. AdaCost(β_1) is the same algorithm as AdaC1. 67

4.1 Loss function $L(yF_t(\mathbf{x}))$ minimized by each cost-sensitive boosting variant. To derive it, we used Eq. (4.5) and the weight update rule and initial weight of each method given in Table 3.1. 73

5.1 The population minimizer of the loss function of each method. This is directly derived from the loss function of Table 5.2 and results to the decision rule shown on the same table (after replacing true probabilities with estimates as is the case in practice). We have made use of the fact that for AsymAda the final round of boosting will be $t = M$. For AdaMEC the decision rule was *defined* as in Table 5.2 which in turn specifies a loss whose minimizer is the one showed here. 79

5.2 A summary of the cost-sensitive variants showing the loss function minimized w.r.t. the ensemble constructed on round t and the final decision rule, written in terms of probability estimates and misclassification costs. In the second column, we have made use of the fact that for AsymAda the final round of boosting will be $t = M$. AdaMEC's decision rule was *defined* as shown here. AdaBoost is included for reference. 83

7.1 Properties of cost-sensitive boosting algorithms and their calibrated counterparts. AdaBoost added for completeness. 108

8.1 Average *area under the Brier curves* produced by each of the 15 methods examined for all 18 datasets. The area is equal to the average *Brier score*, so lower values are desirable. The lowest value per dataset is marked in bold. AsymAda, AdaMEC & CGAda outperform the other approaches and are in turn outperformed by their calibrated versions. The best performing method overall is *calibrated AsymAda*. 114

8.2 Average *area under the Brier curves* produced by the calibrated versions of AsymAda, AdaMEC & CGAda ensembles of equal ensemble size, for all 18 datasets. The area is equal to the average *Brier score*, so lower values are desirable. The lowest value per dataset is marked in bold. *By constraining AsymAda to use as many weak learners as AdaMEC & CGAda, it loses its advantage over the other two methods. In Figure 8.7 we provide evidence that the three methods do not significantly differ in performance.* 115

B.1 Characteristics of the datasets used in our experiments; number of instances used, number of features and number of classes. The class chosen as 'positive' was the minority class in the original file. In multiclass datasets, we followed a 1-vs-all approach, where the negative class consisted of uniformly sampled examples from the remaining classes. All datasets are taken from the UCI repository and can be found in the link: https://archive.ics.uci.edu/ml/datasets.html. 144

List of Figures

4.1 Blue: the ideal path towards the minimum of the loss function. Green: path followed by taking boosting steps of optimal size. Red: path followed by taking boosting steps in which shrinkage is applied. The red path more finely approximates the optimal one (blue), than the green one, but requires more steps to reach the minimum. Image taken from [116]. 77

6.1 LEFT: The loss function used in AdaBoost. This illustrates the reason why AdaBoost is seen as a margin-maximising method – the loss is non-zero even when an example has been correctly classified ($y_i F_t(\mathbf{x}_i) > 0$). RIGHT: The loss for CGAda in a $2:1$ cost ratio – note that the same margin maximising properties hold, and that examples of the positive (expensive) class always have a loss greater than that of examples of the negative (cheap) class, given an equal $y_i F_t(\mathbf{x}_i)$. 92

6.2 LEFT: The loss of CSAda & AdaDB does not preserve the class asymmetry leading to asymmetry swapping when $y_i F_t(\mathbf{x_i}) > 0$. Beyond that point, examples of the positive (expensive) class have a lower loss than that of examples of the negative (cheap) class, given an equal $y_i F_t(\mathbf{x}_i)$. RIGHT: The loss of AdaC3, plotted here for $t = 3$, also exhibits asymmetry swapping. More specifically it does so for margin values $y_i F_t(\mathbf{x_i}) > ((t-1)/(c_{FN} - c_{FP}))\log(\frac{c_{FN}}{c_{FP}})$, i.e. in this example, when $y_i F_t(\mathbf{x_i}) > 2\log 2$. 92

6.3 Cumulative (normalized) margin distributions for *AdaBoost*/AdaMEC, CGAda, AsymAda, CSAda & AdaC3 for degrees of imbalance $\frac{c_{FN}}{c_{FP}} = \{1,2,5,10\}$. When $\frac{c_{FN}}{c_{FP}} = 1$ all methods reduce to the original AdaBoost and produce similar distributions, indicative of margin maximization. But when $\frac{c_{FN}}{c_{FP}} > 1$, CSAda and AdaC3 produce margins closer to 0. This is a result of asymmetry swapping and margin theory suggests it has a negative effect on generalization. 94

7.1 The effect of a single outlier expert probability on the overall probability of the PoE, showing how a single high-confidence expert can dominate the PoE's prediction, leading it to the opposite direction of the rest of the ensemble. LEFT: The PoE probability for a product of 10 experts. The first 9 experts have probabilities $p_t(y=1|\mathbf{x}) \sim \mathcal{N}(0.53, 0.01)$, i.e predominantly predict the positive class. The probability of the 10-th expert varies in the range $p_{10}(y=1|\mathbf{x}) \in (0, 0.49)$, i.e. predicts the negative class. As $p_{10}(y=1|\mathbf{x})$ approaches 0 it tends to rapidly dominate the PoE probability. RIGHT: Same as before, but now the first 9 experts have probabilities $p_t(y=1|\mathbf{x}) \sim \mathcal{N}(0.47, 0.01)$, i.e predominantly predict the negative class. The probability of the 10-th expert varies in $p_{10}(y=1|\mathbf{x}) \in (0.51, 1)$, i.e. predicts the positive class. As $p_{10}(y=1|\mathbf{x})$ approaches 1 it tends to rapidly dominate the PoE probability. 100

7.2 The effect of the size M of the PoE on the overall probability. LEFT: The blue curve corresponds to a PoE whose expert probabilities $p_t(y=1|\mathbf{x})$ follow a Gaussian distribution with mean 0.53 and the green curve to a PoE whose expert probabilities $p_t(y=1|\mathbf{x})$ follow a Gaussian distribution with mean 0.47. In both cases the standard deviation is 0.1. We can see that for small numbers of experts M, the PoE probabilities remain close to 0.5. But as the number of experts M increases, the PoE probability of the former PoE tends to 1 and that of the latter to 0. RIGHT: The same experiment but the standard deviation of both Gaussians is now 0.01. The two PoE probabilities converge to 1 and 0, respectively, much faster. 100

8.1 All methods, across all datasets, ranked by their average area under the Brier curve (BS) – higher rank (higher bar) is better. This illustrates the resilience of each method to a spectrum of changing cost ratios. The highest ranked method is *AsymAda-Calibrated*. 116

8.2 Pareto plot showing the tradeoff between the size of the final ensemble and the rank by average area under the Brier curve (BS). The dashed curve demarcates the Pareto front. The plot shows that *calibrated* AdaMEC and *calibrated* AsymAda are pareto-optimal, with the former to be preferred for parsimonious models. 117

8.3 Loss Q under various degrees of skew z for some characteristic datasets included in the study. Lower values indicate better cost-sensitive classification performance. The area under each (Brier) curve corresponds to the *Brier Score* of the final model. The difference of the areas under the curve of AdaMEC and that of *calibrated AdaMEC* is due to the reduction of the *calibration loss* component of the BS. The $Q(z) = z$ diagonal, corresponds to the '*all-negatives*' classifier, while the $Q(z) = 1 - z$ diagonal to the '*all-positives*'. CSB2 is prone to saturating leading to high loss when the skew is high. AdaC1 is prone to ignoring the asymmetry, leading to high loss as z moves away from 0.5. *Calibrated AdaMEC*, adopts in each case an asymmetric behaviour that leads to low loss. As a result it consistently attains the lowest –or tied for lowest– loss. 118

8.4 Figure 8.3, continued 119

8.5 Figure 8.3, continued 120

8.6 Comparison of the average Brier Score ranks across all datasets attained by CSB2, AdaC1, AdaMEC & calibrated AdaMEC against each other under the Nemenyi test, for every other skew ratio examined. Groups of methods that are not significantly different at the 0.05 level are connected. We see that calibrated AdaMEC is consistently ranked best or tied for best, often significantly better than the second best. . . 121

8.7 Comparison of the average Brier Score ranks across all datasets attained by AdaMEC, CGAda & AsymAda against each other under the Nemenyi test, for every other skew ratio examined. All methods were calibrated and used the *same ensemble size on each run*. Groups of methods that are not significantly different at the 0.01 level are connected. We see no evidence that any method clearly dominates the others. 122

8.8 All boosting variants of Table 7.1, across all datasets, ranked by their average area under the Brier curve (BS) – higher rank (higher bar) is better. This illustrates the resilience of each method to a spectrum of changing cost ratios. Calibrated AsymAda, AdaMEC & CGAda outperform their uncalibrated counterparts and they in turn outrank all other variants. Interestingly, the methods that rank immediately after AsymAda, AdaMEC & CGAda are the original (i.e. cost-insensitive) AdaBoost and its calibrated version outrank all remaining variants. . . 124

8.9 Pareto plot showing the tradeoff between the size of the final ensemble and the rank by average area under the Brier curve (BS) for all boosting variants of Table 7.1. 125

Abstract

COST-SENSITIVE BOOSTING:
A UNIFIED APPROACH
Nikolaos Nikolaou
A thesis submitted to the University of Manchester
for the degree of Doctor of Philosophy, 2016

In this thesis we provide a unifying framework for two decades of work in an area of Machine Learning known as *cost-sensitive Boosting algorithms*. This area is concerned with the fact that most real-world prediction problems are *asymmetric*, in the sense that different types of errors incur different *costs*.

Adaptive Boosting (*AdaBoost*) is one of the most well-studied and utilised algorithms in the field of Machine Learning, with a rich theoretical depth as well as practical uptake across numerous industries. However, its inability to handle asymmetric tasks has been the subject of much criticism. As a result, numerous cost-sensitive modifications of the original algorithm have been proposed. Each of these has its own motivations, and its own claims to superiority.

With a thorough analysis of the literature 1997–2016, we find 15 distinct cost-sensitive Boosting variants – discounting minor variations. We critique the literature using *four* powerful theoretical frameworks: *Bayesian decision theory*, the *functional gradient descent* view, *margin theory*, and *probabilistic modelling*.

From each framework, we derive a set of properties which must be obeyed by boosting algorithms. We find that only 3 of the published Adaboost variants are consistent with the rules of *all* the frameworks – and even they require their outputs to be *calibrated* to achieve this.

Experiments on 18 datasets, across 21 degrees of cost asymmetry, all support the

hypothesis – showing that once calibrated, the three variants perform equivalently, outperforming all others.

Our final recommendation – based on theoretical soundness, simplicity, flexibility and performance – is to use the *original* Adaboost algorithm albeit with a shifted decision threshold and *calibrated* probability estimates. The conclusion is that novel cost-sensitive boosting algorithms are unnecessary if proper calibration is applied to the original.

Declaration

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Copyright

Acknowledgements

I would like to thank my advisor Dr. Gavin Brown for his patience –which I must have come very close to exhausting a couple of times– for all the opportunities he has given me and for guiding me through every aspect of academia from research and teaching, to writing and time-management. For providing me with the right guidance, while also giving me the freedom to pursue my own ideas.

I would also like to thank all members of staff at the University of Manchester for providing me with a perfect working environment all these years. Special thanks to the Centre for Doctoral Training (CDT) in Computer Science, which is funded by the Engineering and Physical Sciences Research Council (EPSRC) grant [EP/I028099/1] and to its organisers, Dr. Barry Cheetham, Dr. Jonathan Shapiro and Dr. Steve Furber. I thank Prof. Uli Sattler, Prof. Mikel Luján and Prof. Robert Stevens for helping me with my first steps of reintroduction to society after the Ph.D. I also thank the examiners of this thesis, Prof. Fabio Roli and Prof. Timothy Cootes, for their useful feedback and suggestions that improved the final version.

My deepest thanks to my colleagues and friends, Konstantinos Sechidis –who has been like a second advisor to me, Henry Reeve and Sarah Nogueira. Much of what is good in this thesis is due to their feedback and influence. Even more importantly, their friendship has made my time during my Ph.D. really fun and enjoyable. I also thank the other members of the team, Adam, Diego, Veronica, Laura, Emily and Georgiana, for all the things I learnt from them as well in the short time we spent together.

To my collaborators, Narayanan Edakunni, Peter Flach and Meelis Kull, thank you for the fruitful collaboration and the countless things I learned working with you. A big thanks is also due to all my labmates at the Machine Learning and Optimization group for all the interesting discussions we had, for their enthusiasm in sharing their own ideas and for the valuable feedback they provided me with when needed, especially Andrew Webb, Jon Parkinson and Joe Mellor.

To my fellow Ph.D. student Giorgos Kourtis, thank you for the support but also our deep conversations about mathematics, computer science, politics and life. An open mind with interests both broad and deep is always enjoyable to probe. To my housemates, Aitor, Dave, Ilia and Nico, thanks for the peaceful and enjoyable cohabitation. Special thanks to Ilia and Nico for their support and friendship.

A big thanks is also due to all the other friends I've made while staying in Manchester, Maria-Myrto, Idoia, Fei, Daniele, Spyros, Yujia, Paris, Laura, Smaragda, Mustafa, Yaman, Thanos, Ioanna, Simone, Oscar, Francis, Serhat and Paschalis, for all the fun times we had together. And thanks to all the other guys from the APT football team. Our games provided me with the perfect way to relax after a day of research. As did our visits in the pub afterwards.

A huge thank you to all my friends back in Greece, especially Kostas, Haris, Stratis B., Giorgos, Eleni, Niki, Christos, Zisis, Despoina, Panagiotis P., Panagiotis K., Nikos, Stratis K., Argyris, my cousin Dimitris and his family, for their love and for making me feel every time we meet that not a day has passed since I last left them.

Last but not least, I would like to thank from the bottom of my heart my family for their the endless love and support throughout these years, especially my parents, Dafni and Charalambos and my partner, Olga. Thank you for all your love, for believing in me, for all the strength you gave me, for being there for me all this time! Ευχαριστώ πολύ!

Chapter 1

Introduction

1.1 Motivation

1.1.1 A data-driven world

The last decades have witnessed a dramatic increase in the amount of data generated. At the time of writing this thesis, it is estimated[1] that “more than 90% of the data in the world have been generated in but the last two years”. This includes information of any kind: sensor measurements of all sorts, social media posts, web searches, transaction records, music, videos, images, GPS signals from mobile devices, results of all kinds of scientific experiments, news posts, to name but a few. We live in the era of *big data*.

The billions of stationary and portable devices connected to the internet constantly generate new data. In the years to come this trend will without doubt continue at an ever increasing rate, with smart devices entering every house and replacing conventional appliances (giving rise to the so-called *internet of things*) and every aspect of civic life, turning our cities into *smart cities*.

The large amount of available data has given rise to a multitude of *data-driven services* including smart search engines, automatic translation and voice recognition. The same algorithms used to power these applications are used in the domain of *Artificial Intelligence*, for instance to train self-driving cars and drones, soon to become a common sight in our streets and skies. And the same algorithms have dramatically accelerated the rate of new scientific discoveries, allowing us to sift through massive amounts of scientific data to test hypotheses and uncover patterns.

Every field of computer science has undoubtedly played a role in facilitating these

[1]Source: IBM, url: http://www-01.ibm.com/software/data/bigdata/what-is-big-data.html

data-driven breakthroughs. This includes advances in hardware, but also in algorithmic areas related to data collection, distribution and storage like signal processing, networks and databases. But arguably the most instrumental role is played by *machine learning*, *'the study of algorithms that make sense of data'*–to paraphrase the title of one of the most recent books in the field [38]. It is to this field that the present thesis contributes.

1.1.2 Making sense of data

This section is a brief introduction to the subject of machine learning. It provides a bird's-eye view of the field, abstracting the key notions that are most relevant to this work and establishing some terminology that will follow us in the rest of this thesis. We beg the reader's indulgence if all this terminology is overwhelming. All concepts and terms introduced in this chapter will become more concrete throughout the following chapters once related to specific problems, methods and models.

Machine learning studies the principles and methods for automating statistical inference. It allows us to *learn from data*, i.e. learn from past *observations* (also referred to as *examples* or *instances*) and make predictions regarding new ones. The past observations – the ones used by the learning algorithm as examples to *train a model* – constitute the *training set*, while the new ones –those used during *deployment*, to evaluate it – form the *test set*. *Training* translates to *finding a mathematical model that best describes the data*. Using this model we can detect *patterns* or *anomalies*, make *predictions* and ultimately make *decisions* based on said predictions.

As for the model[2], it can be of many sorts: a *tree* or a *graph*, an *algebraic equation*, a *logical rule* or a *probability distribution* – and more often than not it can be interpreted in many different ways, as we will see in the rest of this thesis.

Finding the *best* model (in a given family) is problem-dependent and usually involves solving an *optimization problem* of some *objective function* w.r.t. the *parameters* (the *free variables*) of the model. This means finding the values of the parameters that e.g. minimize the discrepancy between its predictions and the truth, or minimize the expected cost of the decisions, or maximize the likelihood of the data under the model. Without loss of generality[3], in this work we will be referring to the underlying

[2]Here it might be useful to disambiguate two different uses of the term *model*. It can be used to refer to a *family of models (hypothesis class)*, e.g. linear equations of the form $y = ax + b$, where a and b are free parameters, or to a specific *instance/realization (hypothesis)* of that family e.g. $y = -2x + 3$. Throughout the thesis, it will be clear from the context to which of the two meanings we refer each time.

[3]Any maximization problem can be easily transformed to a minimization one.

optimization problem as a *minimization problem*, and to the objective function as the *loss function*.

Ultimately, how well the model fits the training data is of secondary importance. A good model fits the training data well –if it doesn't, we say it *underfits the data*, meaning it fails to capture the underlying pattern in the data. But this is just part of the picture. What really matters is the model's ability to *generalize*, i.e. give good predictions on previously unseen examples. This generalization performance is thus estimated on the test set. To account for this, there usually exists some way of balancing the *complexity* of the model (e.g. as measured by the number of its free parameters or *degrees of freedom*) and how well it fits the data during training, which leads to good generalisation performance. The basic principle behind this is that a very complex model might capture the *noise* in the training data as part of the pattern –in which case it is said to *overfit the data*[4]. This means that the model is overly specific to the training data to the extend that it fails to generalize to new instances. On the other hand, a very simple model is prone to underfitting, again failing to give good predictions.

In this work we will be focusing on *classification* problems. Classification is perhaps the most common type of machine learning tasks. As the name suggests, it is concerned with assigning a new example to one of a discrete number of categories, known as *classes*. It is *supervised* in the sense that the model is trained on data of which the *label* (i.e. the *true class*, the true desired output) is known.

For ease of exposition and clarity, we will be focusing on *binary classification*, where examples can belong to one of two possible classes, the *positive* and the *negative* one. For example, a bank system that decides whether to grant a loan or not needs to solve a binary classification problem. Credit-worthy clients are the positive examples and those that defaulted form the negative class. Despite our focus on binary classification, our analysis can easily extend to the multi-class case e.g. by reducing the multiclass problem to multiple *binary classification* subproblems. Whenever applicable, we will be providing the reader with pointers on how this is achieved.

[4]The principle of *Occam's Razor*, ubiquitous in science, suggests that simpler models are to be preferred over more complicated ones that have the same explanatory power. Limiting model complexity to avoid overfitting can be viewed as an appeal to this principle. Machine learning is typically concerned with *inductive inference* –generalizing from a set of examples. This is in general an *ill-posed problem*, in the sense that more than one model can explain the examples. A general term to refer to any process of introducing additional information (constraining the space of possible models) in order to solve an ill-posed problem or to prevent overfitting, is '*regularization*'.

1.1.3 Asymmetric learning

Most classification problems are *asymmetric*, in the sense that either the *costs* of different types of misclassifications are unequal, or the classes have different *prior probabilities* – or both. Asymmetric prediction tasks are encountered everywhere, in real life applications and scientific data analysis alike.

An astrophysicist trying to detect a rare astronomical phenomenon, like a supernova, is faced with an *imbalanced class* problem, since most of the available observations will be of the negative class, i.e. not involving the phenomenon. A doctor testing a patient for a life-threatening disease, is faced with a *cost-sensitive* decision: a *false positive* will lead to further tests which will eventually reveal the misdiagnosis, while a *false negative* means that the disease is left undetected and thus untreated, with potentially lethal results. The same is true for the credit card fraud detection system of a bank: instances of illegal transactions are much rarer than legal ones, but a false negative incurs a far higher cost than a false positive. Mildly annoying customers when suspicious activity is detected in their accounts is preferable to them falling victims of fraud.

In all these cases, it is not the number of correct classifications that matters, but the *overall cost* our decisions incur. The goal is not to minimize the number of wrong decisions, but to make decisions in such a way that their *expected cost is minimized*. The primary aim of this work is to target this goal. And while our focus will be on cost-sensitive learning, as we will see, there is a duality in the two types of asymmetric learning problems that also allows imbalanced class tasks to be addressed using the same approaches as cost-sensitive tasks and vice-versa.

1.1.4 Ensemble learning

Ensemble learning algorithms are equivalently ubiquitous in the Machine Learning literature. The term encompasses techniques to combine multiple predictors –known as *base learners*– with the aim of producing a more powerful one. *Ensemble* algorithms have been shown to outperform other approaches in multiple extensive studies in the literature [13, 36]. Ensemble learning has also been a key principle in the winning entries of every major machine learning competition; the most celebrated example being the *Netflix Challenge* [119]. Its main branches –*bagging*, *boosting* and *random forests*– have been used in a number of successful applications, including face recognition for mobile phone cameras [123], the Microsoft Kinect motion sensing devices [50] and remote sensing [34, 90].

The *Adaboost* algorithm [42], a specific form of boosting, stands out in the field of ensemble learning – named in a community survey as one of the top ten algorithms in data mining [131], whilst also having a rich theoretical depth, winning the 2003 Gödel prize for the authors. One of the often-cited weaknesses of AdaBoost is its inability to handle cost-sensitivity [57, 112, 117]. It is no surprise therefore, that significant international research effort has been dedicated to adapting Adaboost for cost-sensitive tasks. At the time of writing this thesis at least 15 distinct variants –even more, if we were to count slight variations thereof as separate algorithms– have been proposed in a sequence of papers [33, 63, 64, 78, 79, 112, 113, 117, 123] published 1997-2016. Each of these variants is derived from different underlying perspectives, principles or assumptions –often heuristically– and each makes its own claim to superiority in some sense, either empirically, theoretically, or practically.

But what is each of these variants actually doing? Are they appropriate for solving our problem? Could it be that there is a simpler, more practical approach to minimize the expected cost of the predictions of a boosting ensemble? Ultimately, do we need all these variants or can the literature be simplified?

1.2 Research questions

This thesis aims to answer a series of questions regarding these cost-sensitive boosting variants.

- *What exactly does each variant achieve?* As many of the variants were proposed heuristically, the loss function they minimize is not explicit, but hidden within the steps of the algorithm. Identifying this loss is the first step towards understanding what the goal of each variant is.

- *What are the desirable properties of a cost-sensitive boosting variant?* Given the cost-sensitive nature of the task and the way boosting operates, what properties should a cost-sensitive boosting algorithm satisfy?

- *What properties does each variant satisfy?* Once its loss function is made explicit, we can start investigating whether it is a sensible choice under the properties we identified. We compile a list of all variants along with the properties they satisfy.

- *Are there algorithms that satisfy all desirable properties? If not, can we grant them a missing property?* As we will see, all variants assume that their outputs can be interpreted as probability estimates. We will see why this is not true and how we can correct for it.

- *Ultimately, do we need all these variants, or is there a simpler more principled approach?* To answer this question we will use arguments based on the theoretical analysis outlined in the following section, practical considerations and the empirical performance of the methods examined.

In the process of answering the above questions we will establish a unified treatment of these techniques, simplifying the existing literature and uncovering the underlying loss function each method minimizes, along with the properties it satisfies. The tools we use, along with the answers we will provide have the potential to offer insight to other learning algorithms beyond boosting and open multiple directions for further research.

1.3 Outline and contributions

In this work we analyse the literature, using tools from *four* theoretical frameworks: *decision theory*, *functional gradient descent*, *margin theory*, and *probabilistic modelling*. Each one of these will turn out to have its own advantages and perspective in the analysis, and the work could not be complete without all four.

The functional gradient descent view ensures the steps of the algorithm are consistent with greedy minimisation of a well defined loss function which is itself a function of the margin[5], thus ensuring an efficient path toward good generalisation properties. Analysis of the loss functions from the perspective of margin theory ensures the generalisation behaviour is in line with the defined cost-sensitive problem. The decision theoretic view ensures that the predictions generated by boosting are used in a way that is consistent with the goal of minimizing the expected cost of future classifications. A probabilistic analysis shows that the models generate *uncalibrated* outputs, which we proceed to remedy; key to this argument is rephrasing Adaboost as a *Product of Experts (PoE)*. Different cost-sensitive boosting variants translate to different PoE models but

[5]For now it suffices to think of the *margin* of the classifier on a given example as the *confidence of the classifier in its prediction regarding that example*. The concept of margin will be formally introduced in Chapter 4 and analyzed extensively in Chapter 6.

they all suffer from the same systematic distortion of the resulting probability estimates due to their PoE nature.

Our main contributions are summarized below:

- *We identify the loss function implicitly minimized by each variant. We simplify a large body of literature under a common framework.* (Chapter 4)
- *We provide a more general formulation of an existing AdaBoost variant [117] for assigning instances to the class with the minimum expected cost, which is independent of the specific way probabilities are estimated.* (Chapter 5)
- *We identify the desirable properties of a cost-sensitive boosting variant and establish when an algorithm fails to satisfy them and how this will affect its behaviour.* (Chapters 4 - 7)
- *We identify the properties each variant satisfies. This allows us to see which variants are closer to –theoretical– optimality than others.* (Chapters 4 - 7)
- *We determine that there are no algorithms that satisfy all desirable properties. More specifically, we show that all variants fail to produce outputs that can be interpreted as calibrated probability estimates.* (Chapters 4 - 7)
- *We propose the use of calibration to correct the algorithms for the missing property of poor probability estimation. We conclude that three variants, once properly calibrated now satisfy all desirable properties.* (Chapter 7)
- *We conduct the most extensive empirical comparison of cost-sensitive boosting algorithms in the literature.* (Chapter 8)
- *We conclude that existing cost-sensitive boosting algorithms are unnecessary if proper calibration is applied.* (Chapters 7 - 8)
- *We provide theoreticians with a set of tools to examine existing and future boosting variants, identifying directions for further exploration.* (Chapters 4 - 7)
- *We provide practitioners with a theoretically sound, fast, intuitive, simple and practical alternative to modifying the original AdaBoost algorithm and show that it matches or outperforms existing alternatives.* (Chapters 7 - 8)

To summarize our main result, we challenge the idea of modifying AdaBoost to account for asymmetries and propose instead handling such tasks by classic decision theoretic approaches after properly *calibrating* the scores of the original AdaBoost so that they correspond to probability estimates. This approach is grounded on decision theory, preserves the theoretical guarantees of AdaBoost, eschews the need of additional hyperparameters and can easily account for changes in asymmetry during deployment, without need for retraining. Extensive empirical comparisons of calibrated AdaBoost against the existing cost-sensitive AdaBoost variants show that it achieves superior performance. The conclusion is that existing cost-sensitive boosting algorithms are unnecessary if proper calibration is applied.

1.4 Structure of this thesis

The thesis is structured as follows.

In Chapter 2 we introduce the problem of *asymmetric* learning. We will discuss in depth the different flavours of asymmetric learning tasks, examine the duality between cost sensitive and class imbalanced classification and how they can be formulated and treated following the same principles. We will present different ways to evaluate asymmetric classification systems and mention the various factors that can complicate learning in the asymmetric setting.

In Chapter 3 we will present the concept of hypothesis boosting and its most popular representative, the basic Adaboost algorithm. We will discuss its importance both as a theoretical breakthrough and as a successful practical algorithm, but also its limitations. We will outline the different perspectives by which it can be understood, emphasizing the ones that we are going to be using throughout the rest of the thesis. Finally, we will analyse the multitude of cost sensitive boosting variants in the literature, and consolidate them into a common notation and terminology.

Chapters 4 - 7 constitute our main theoretical contribution. Each of them examines the cost-sensitive boosting algorithms from a different theoretical perspective, identifies a desirable property, explains how to examine if a given algorithm satisfies it and classifies the algorithms according to whether they do so or not.

More specifically, Chapter 4 uses the functional gradient descent view of boosting to infer the loss function minimized by each variant, thus making the objective of each algorithm explicit and allowing us to examine them all –even some previously regarded as heuristics– under the same framework. We will see which algorithms are efficiently optimizing their loss function and which take suboptimal steps. Most importantly, the loss function itself will play an important part for the three chapters to follow.

Chapter 5 examines the cost-sensitive boosting variants through the lens of decision theory. It relates their underlying loss functions to clear decision rules and allows us to determine whether these rules are the appropriate ones for making decisions in order to minimize the expected cost of the classifications. In the process, we also provide a more general formulation of an existing AdaBoost variant [117] aiming to do precisely that: assign a new instance to the class that will incur the lowest expected cost. Our new proposal, unlike the original [117] does not assume a specific way of estimating the probability of the example belonging to the positive class.

In Chapter 6 we make use of margin theory to determine whether the loss function is appropriate for training a boosting variant. This gives important insight into the generalization ability of each algorithm. We will see that certain families of loss functions will eventually force the algorithm to invert the relative importance of the two classes during training, leading to the classifier emphasizing the low-cost class more than the high-cost one on the subsequent round, ultimately hurting generalization.

Chapter 7 examines boosting from a probabilistic modelling angle, identifying a common issue in all boosting variants: their output (normalized to be a real number $\in [0,1]$) cannot be directly interpreted as a probability estimate. We show why this is the case by demonstrating that the probability estimates generated by boosting variants can be rephrased as Products of Experts, and as such are prone to systematic forms of distortions. To correct for this, we propose the use of logistic calibration (Platt scaling) [92] of probability estimates and provide a detailed pseudocode for its implementation. We summarize all variants and desirable properties in a complete table, the calibrated variants now satisfying all properties.

Chapter 8 provides a large scale experimental study to test our hypotheses – 18 datasets, 15 variants of boosting, across 21 different degrees of class imbalance. It constitutes the empirical part of our contributions and is supported by extensive analysis and tests of statistical significance.

Finally, Chapter 9 draws together the conclusions of the thesis, identifying multiple interesting directions for future work.

1.5 Publications

The work presented in this thesis has resulted into two publications:

[87]: Nikolaou, N., Brown, G., Calibrating AdaBoost for asymmetric learning, *In Proc. of Multiple Classifier Systems*, pages 112–124, 2015

[88]: Nikolaou, N., Edakunni, N., Kull, M., Flach, P., Brown, G., Cost-sensitive boosting algorithms: Do we really need them?, *Machine Learning*, 104(2):359–384, 2016.

Other published work not relevant to this thesis:

[109]: Sechidis, K., Nikolaou, N., Brown, G., Information theoretic feature selection in multi-label data through composite likelihood, *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*, pages 143–152, 2014

Chapter 2

Asymmetric Learning

In this chapter we will present the basic principles and premises of asymmetric learning. We will introduce the notation and terminology that will be used throughout the rest of the thesis. We will expose the duality between the two flavours of asymmetric learning, cost-sensitive learning and learning under class imbalance, thus justifying why we can focus on cost-sensitive problems without loss of generality. We will outline the established assumptions traditionally underlying the cost-sensitive learning literature and define the specific problem we will be addressing in the chapters to come. We will analyze the primary evaluation tool we are going to be using to assess cost-sensitive performance, namely the Brier curve [53]. Last but not least, we will present an overview of the basic families of techniques for handling cost-sensitive learning. Despite being beyond the scope of this work, for completeness, we will also present alternative asymmetric problem settings, formulations and evaluation measures as well as some interesting issues that may arise in the asymmetric learning setting and how they can be dealt with.

2.1 Minimizing the risk

Throughout this work we will be focusing on binary classification, for ease of exposition and clarity. Extension to the multiclass case is often handled by breaking down the problem into multiple binary ones, so our analysis and its results can carry over to the multiclass case[1]. In a binary classification task, an example can be either *positive*, denoted by a label $y = 1$ or *negative*, denoted by $y = -1$. The classifier is trained on a

[1]Common ways to do this include *one-vs-all* and *one-vs-one* approaches, as well as *error-correcting output codes (ECOC)*. See [23, 97, 115] for an overview.

dataset consisting of *labelled training examples* $(\mathbf{x}_i, y_i)$, where $\mathbf{x}_i \in \mathcal{X}$, $\mathcal{X}$ being some appropriate input space.

On a new *unlabelled test example* $\mathbf{x}_i$, the classifier makes *predictions* $h(\mathbf{x}_i) \in \{-1, 1\}$ and it incurs a *loss* when those predictions are wrong. In the most general case, each classification, is associated with a distinct cost c_i and the loss on the i-th example is thus a function of that cost. But in the most common setting, the loss on each example depends only on the combination of classifier output $h(\mathbf{x}_i)$ and true label y_i, so we denote the loss on the i-th example with $L(h(\mathbf{x}_i), y_i)$. In binary classification, we can have two types of correct classifications, *true positives (TPs)* –predicting a positive example as positive– and *true negatives (TNs)* –predicting a negative example as negative. We can also have two types of misclassifications, *false negatives (FNs)* also known as *type I errors* or *misses* –predicting a positive example as negative –and *false positives (FPs)* also known as *type II errors* or *false alarms*– predicting a negative example as positive. In our notation, the different outcomes of the classification of $\mathbf{x}_i$ are denoted as

- True Positive: $h(\mathbf{x}_i) = 1 \wedge y_i = 1$.
- True Negative: $h(\mathbf{x}_i) = -1 \wedge y_i = -1$.
- False Positive: $h(\mathbf{x}_i) = 1 \wedge y_i = -1$.
- False Negative: $h(\mathbf{x}_i) = -1 \wedge y_i = 1$.

The *cost matrix* of the problem shown in Table 2.1 is the structure that encodes the information about the cost associated with each combination of classifier output and true label. It is *fixed*, in the sense that for each example we use the *same* cost matrix to determine the cost of its classification.

A thorough treatment of cost matrices in particular and, more generally, the subject of *cost-sensitive classification* can be found in [32]. Elkan gives a set of conditions regarding the entries of a cost matrix in order for the cost matrix, thus our problem, to be *reasonable*. For the cost-matrix to be reasonable it must be the case that

$$\begin{cases} c_{FP} > c_{TN} \\ c_{FN} > c_{TP}, \end{cases} \tag{2.1}$$

which simply ensures that misclassifying any instance incurs a larger cost than correctly classifying it –indeed, a reasonable requirement.

Table 2.1: General form of a fixed cost matrix in binary classification.

		Predicted Label $h(\mathbf{x}_i)$	
		1	-1
True Label y_i	1	c_{TP}	c_{FN}
	-1	c_{FP}	c_{TN}

Here we will be examining the most commonly studied case [24, 32], under which, the cost of misclassifying the i-th example only depends on its class label y_i – what Landesa-Vázquez and Alba-Castro [65] refer to as *class-level asymmetry*. This requires that *correct classifications incur no cost*, i.e. $c_{TP} = c_{TN} = 0$. So the binary classification cost matrix is of the form shown in Table 2.2.

Therefore the conditions of Eq. (2.1) become

$$\begin{cases} c_{FP} > 0 \\ c_{FN} > 0. \end{cases} \tag{2.2}$$

Table 2.2: A cost matrix for a binary classification with $c_{TP} = c_{TN} = 0$.

		Predicted Label $h(\mathbf{x}_i)$	
		1	-1
True Label y_i	1	0	c_{FN}
	-1	c_{FP}	0

In all subsequent discussion, we will be assuming that the cost matrix is of the form of Table 2.2. The reader might get the impression that we are constraining the problem, however this is not the case. Any cost matrix of the form of Table 2.1 can be transformed to an equivalent cost matrix of the form of Table 2.2, by elementary matrix operations, more specifically, by subtracting c_{TP} from all entries of the first column and c_{TN} from all entries of the second column. For this reason, the entire cost-sensitive literature referenced in this chapter and all cost-sensitive boosting variants we will present in the next chapter, assume that the cost matrix is of the form of Table 2.2. Thus, without loss of generality, the cost of the *i*-th example is simply a function of its class label y_i, and more specifically,

$$c_i = c(y_i) = \begin{cases} c_{FN}, & \text{if } y_i = 1 \\ c_{FP}, & \text{if } y_i = -1. \end{cases} \tag{2.3}$$

Simply put, c_{FN}, the cost of misclassifying a positive as a negative captures the *importance of positive examples* and c_{FP}, the cost of misclassifying a negative as a positive captures the *importance of negative examples*.

Before we move on, note that *scaling* (multiplying) by a positive constant and *shifting* (adding a constant) the cost matrix does not affect the optimal solution of the underlying decision problem[2]. So, if we divide the entries of the cost matrix of Table 2.2 by c_{FP}, we get the *equivalent cost matrix* of Table 2.3, where

$$c_r = \frac{c_{FN}}{c_{FP}}, \tag{2.4}$$

is the *cost ratio*, which captures the *relative cost of positive examples over negatives*. Hence the cost matrix only has *one degree of freedom*.

The conditions of Eq. (2.2) in this case simplify to

$$0 < c_r < \infty.$$

We will be exploiting the fact that the cost ratio c_r uniquely defines the cost sensitive problem, but to make the exposition clearer, we will be using the equivalent form of the cost matrix of Table 2.2 in our subsequent analysis[3]. The special case where $c_{FP} = c_{FN}$ and thus $c_r = 1$, is –of course– the traditional, *cost-insensitive* setting.

[2]This will become more clear, once we properly *define* our problem in terms of *minimizing the expected loss of the classifications*. For now, the reader can think of such linear transformations of the cost matrix as merely modifying the *unit of measurement of the costs*, which should not affect our decisions.

[3]A common convention in the literature that goes hand in hand with the use of the cost matrix of Table 2.3, is to assume –without loss of generality– that the *important* class (*rare*, *high-cost*, *expensive*) is the positive one and that the *unimportant* one (*frequent*, *low-cost*, *cheap*) is the negative. Here we forgo this convention for the sake of clarity.

Table 2.3: An alternative cost matrix for a binary classification with $c_{TP} = c_{TN} = 0$, with only one degree of freedom.

		Predicted Label $h(\mathbf{x}_i)$	
		1	-1
True Label y_i	1	0	c_r
	-1	1	0

Cost-sensitivity is just one aspect of asymmetric learning. The ultimate goal of a classifier is to minimize its *risk* (i.e. its *expected cost* or *expected loss*). In the binary case, under the cost matrix of Table 2.2, the risk on an example $(\mathbf{x}_i, y_i)$ for a classifier h can be written as

$$\begin{aligned} R(h(\mathbf{x}_i), y_i) &= \mathbb{E}_{\mathbf{x}_i, y_i}\{L(h(\mathbf{x}_i), y_i)\} \\ &= c_{FN} \cdot p(h(\mathbf{x}_i) = -1, y_i = 1) + c_{FP} \cdot p(h(\mathbf{x}_i) = 1, y_i = -1), \end{aligned} \tag{2.5}$$

It is easy to see that Eq. (2.5) can be further analyzed into

$$R(h(\mathbf{x}_i), y_i) = c_{FN} \cdot \pi_+ \cdot p(h(\mathbf{x}_i) = -1 | y_i = 1) + c_{FP} \cdot \pi_- \cdot p(h(\mathbf{x}_i) = 1 | y_i = -1),$$

under the sensible assumption that $p(y_i = 1) = \pi_+$ and therefore also $p(y_i = -1) = \pi_-$, for all i, where π_+ and π_- are the *class priors*[4] of the positive and negative class, respectively. The class priors, defined as $\pi_- = N_-/N$ and $\pi_+ = N_+/N$, where N_+ is the number of positive training examples, N_- is the number of negative training examples and N the size of the training set, capture the class imbalance.

There are thus *two types of asymmetries* that can occur in a binary classification problem. If $c_{FP} \neq c_{FN}$, then we are facing a *cost-sensitive learning* problem [32]. If $\pi_+ \neq \pi_-$, then we are dealing with an *imbalanced class learning* problem [16]. When

[4]Also known as '*class ratios*' or '*class proportions*' [53].

at least one of these asymmetries is present in the problem, some authors [71] refer to it as a *non-standard case*. In this thesis we chose to refer to non-standard problems as *asymmetric* or as *skewed* following the terminology of [37] and to the methods that deal with them as *asymmetric* or *skew-sensitive*.

2.2 Cost-sensitive & imbalanced-class learning duality

The reason for grouping these asymmetries (*skewed class* and *skewed cost*) under a common term is that they can *be formulated and treated in similar ways* [24, 37, 74].

For example, suppose that the false positive cost c_{FP} associated with misclassifying negatives is twice the false negative cost c_{FN}. This can be simulated by an adjusted class prior which duplicates every negative, leading to $\pi'_- = \frac{2 \cdot N_-}{N_+ + 2 \cdot N_-} = \frac{(2/3) \cdot N_-}{(1/3) \cdot N_+ + (2/3) \cdot N_-}$ and $\pi'_+ = \frac{N_+}{N_+ + 2 \cdot N_-} = \frac{(1/3) \cdot N_+}{(1/3) \cdot N_+ + (2/3) \cdot N_-}$. More generally, if we define

$$c = \frac{c_{FP}}{c_{FP} + c_{FN}}, \tag{2.6}$$

then this can be simulated by an adjusted class distribution $\pi'_- = \frac{c \cdot N_-}{(1-c) \cdot N_+ + c \cdot N_-}$ and $\pi'_+ = \frac{(1-c) \cdot N_+}{(1-c) \cdot N_+ + c \cdot N_-}$.

Conversely, we can assign costs that are proportional to the rarity of the class of each example, $c'_{FP} \propto \frac{1}{\pi_-}$ and $c'_{FN} \propto \frac{1}{\pi_+}$, thus simulating an imbalanced class problem as a cost-sensitive one. For example, suppose that the positive class is twice as rare (i.e. half as frequent) as the negative one. This means that $\pi_- = 2\pi_+$. By assigning the examples class rarity-proportional costs, we get a cost sensitive-problem where $c'_{FN} = 2c'_{FP}$, i.e. positives are twice as important as negatives.

Finally, either approach can be used to address asymmetry due to both cost and class imbalance, if we take either the priors or the costs to also subsume the cost imbalance or the class imbalance respectively.

Hence we can focus on skewed *cost* problems in this thesis without loss of generality. Using the terminology of [28, 53], we shall refer to to c of Eq. (2.6) as the *cost skew* and to

$$z = \frac{\pi_- \cdot c_{FP}}{\pi_- \cdot c_{FP} + \pi_+ \cdot c_{FN}} \in [0, 1]. \tag{2.7}$$

as the *skew*[5] [28, 53]. The cost skew c accounts for the *asymmetry due to the different costs*, the skew z to the *combined asymmetry due to both different costs and different*

[5]Often referred to as the *probability cost function (pcf)* [24, 28].

class priors. If $c = 0.5$ then the problem is *cost-insensitive* and if $z = 0.5$ then the problem is *skew-insensitive* or *symmetric*. For most of the subsequent analysis, such a distinction between the different sources of asymmetry is not truly necessary[6], for the reasons we just explained. The reader can thus replace c in all equations found in the rest of the thesis with z (or equivalently replace c_{FP} with $\pi_- \cdot c_{FP}$ and c_{FN} with $\pi_+ \cdot c_{FN}$) and these would still be valid. We shall only keep the distinction for clarity, when appropriate, i.e. in Subsection 2.4.2 when discussing the evaluation measure we will use and in Chapter 8 when discussing our experiments.

2.3 Problem definition & optimal decision rule

The aim of this thesis will be to solve skewed *cost* problems with a fixed cost matrix of the form of Table 2.2, hence the cost of each example given by Eq. 3.3. Class priors are assumed to be equal[7]. Our goal, motivated by Bayesian decision theory [4, 7, 30, 32] is to minimize the expected cost (risk) of our classifications. Thus our desired solution is a decision rule that minimizes the risk.

There exists an optimal solution to our problem. Given the probability of a test example $\mathbf{x}$ belonging to the positive class, $p(y = 1|\mathbf{x})$, we should classify it as positive iff the risk of a positive prediction is lower than that of a negative prediction, i.e. iff

$$\begin{aligned}
&R(y, h(\mathbf{x}) = 1) < R(y, h(\mathbf{x}) = -1) \iff \\
&\mathbb{E}_{\mathbf{x},y}\{L(y, h(\mathbf{x}) = 1)\} < \mathbb{E}_{\mathbf{x},y}\{L(y, h(\mathbf{x}) = -1)\} \iff \\
&c_{TP} \cdot p(y = 1|\mathbf{x}) + c_{FP} \cdot p(y = -1|\mathbf{x}) < c_{TN} \cdot p(y = -1|\mathbf{x}) + c_{FN} \cdot p(y = 1|\mathbf{x}).
\end{aligned}$$

Under the cost matrix of Table 2.2, we have that $c_{TP} = c_{TN} = 0$, so we assign $\mathbf{x}$ to the positive class iff:

$$\begin{aligned}
&p(y = 1|\mathbf{x}) \cdot c_{FN} > p(y = -1|\mathbf{x}) \cdot c_{FP} \iff \\
&p(y = 1|\mathbf{x}) \cdot c_{FN} > (1 - p(y = 1|\mathbf{x})) \cdot c_{FP} \iff \\
&p(y = 1|\mathbf{x}) \cdot (c_{FN} + c_{FP}) > c_{FP} \iff \\
&p(y = 1|\mathbf{x}) > c,
\end{aligned} \tag{2.8}$$

where we made use of $p(y = -1|\mathbf{x}) = 1 - p(y = 1|\mathbf{x})$ and Eq. (2.6). Otherwise, $\mathbf{x}$ is

[6] We will thus often be referring to c as simply the *skew* and to the case of $c = 0.5$ as *skew-insensitive* or *symmetric*, assuming $c = z$, without loss of generality.

[7] This is only done for simplicity, as we said it does not preclude them being subsumed by the costs.

assigned to the negative class. When $c_{FP} = c_{FN}$, this reduces to the familiar threshold of 0.5. This all follows straightforwardly from Bayesian decision theory and simply translates to shifting the threshold from 0.5 to c to account for the imbalance in costs.

The output of most learning algorithms –including that of boosting ensembles which are the focus of the thesis– can be used to *estimate* the probability $p(y = 1|\mathbf{x})$, so the decision rule of Eq. (2.8) in practice uses estimates $\hat{p}(y = 1|\mathbf{x})$. A key point here, which will play a major role in this work, is that 'good' *probability estimates* are not always straightforward to obtain from a classifier, even if it has high classification accuracy – such estimates are referred to as *calibrated*, and will be discussed in more detail in Chapter 7.

2.4 Evaluation measures for asymmetric learning

Below we discuss the possible evaluation measures used in binary classification and especially in asymmetric scenarios. We will see why the most commonly used measure of evaluating classifiers, accuracy, is inappropriate, and discuss alternatives calculated from the *contigency table* (i.e. *confusion matrix*) of the classifier, explaining why neither these are the most informative choices for the problem we defined in Section 2.3. Finally, we will present the cost curve [28, 29] and the closely related Brier curve [53] which will be the evaluation measure used in this work. For completeness, we close the section with a brief discussion of an alternative question we could be posing to a cost-sensitive classifier, whose answer falls under the *Neyman-Pearson detection framework.*

2.4.1 Bad choices: accuracy & error rate

Perhaps the evaluation measures most associated with classification problems are

$$accuracy = \frac{\#\, correct\ classifications}{\#\, total\ classifications},$$

and its complementary *error rate* $= 1 -$ *accuracy*. But it is clear that these are bad choices for assessing the performance in asymmetric classification. They *do not differentiate between errors of the two different types*, so they assume that we are dealing with problems where $c_{FP} = c_{FN}$. In other words, the underlying misclassification loss they assume is the $0/1 - loss$,

$$L_{0/1}(h(\mathbf{x}_i), y_i) = \begin{cases} 1, & \text{if } h(\mathbf{x}_i) \neq y_i \\ 0, & \text{if } h(\mathbf{x}_i) = y_i. \end{cases}$$

rather than the cost-sensitive loss

$$L_{CS}(h(\mathbf{x}_i), y_i) = \begin{cases} c_{FP}, & \text{if } h(\mathbf{x}_i) = 1 \wedge y_i = -1 \\ c_{FN}, & \text{if } h(\mathbf{x}_i) = -1 \wedge y_i = 1 \\ 0, & \text{if } h(\mathbf{x}_i) = y_i. \end{cases}$$

defined by the cost-matrix of Table 2.2. As we will see in more depth in later chapters, the inability of the original AdaBoost algorithm to handle asymmetric classification, stems from the fact that it is geared towards minimizing a bound to $L_{0/1}$, rather than L_{CS}.

2.4.2 Good choices

Contigency table measures & ROC analysis

Numerous evaluation measures can be calculated from the four entries of the contingency table of the classification, namely the numbers of *true positives*, *true negatives*, *false positives* and *false negatives*, denoted with TP, TN, FP, FN respectively. Denoting the total numbers of positive and negative examples with $Pos = TP + FN$ and $Neg = TN + FP$, respectively, we can define, among other measures:

- The *recall* or *true positive rate (TPR)* or *sensitivity* or *detection rate* is the fraction of positives correctly classified
$$Rec = TPR = \frac{TP}{Pos}.$$
- The *precision* or positive predictive value (PPV) is the fraction of positive predictions that are correct
$$Prec = \frac{TP}{TP + FP}.$$

- The *true negative rate (TNR)* or *specificity* is the fraction of negatives correctly classified
$$TNR = \frac{TN}{Neg}.$$
- The *false positive rate (FPR)* or *false alarm rate* is the fraction of negatives incorrectly classified
$$FPR = \frac{FP}{Neg} = 1 - TNR.$$
- The *false negative rate (FNR)* is the fraction of positives incorrectly classified
$$FNR = \frac{FN}{Pos} = 1 - TPR.$$

If only the performance of the positive class is of interest, the two measures that are important are precision and recall. Still, precision or recall alone tell us little about the performance of a classifier. A classifier can have a perfect precision ($Prec = 1$) and a very poor recall and vice versa, just by assigning all examples to one of the classes. It is *the tradeoff between these two measures* that matters. A measure that combines both precision and recall into a single quantity is the *harmonic mean* of precision and recall, the F_1*-measure* [114] (often just referred to as *F-measure*)

$$F_1 - measure = \frac{2 \cdot Rec \cdot Prec}{Rec + Prec}.$$

The F_1-measure is high when both precision and recall are high. Being their harmonic mean, the F_1-measure is closest to the smallest among precision and recall.

Note that *the number of true negatives is ignored by both precision and recall*, thus also by the F_1-measure. When performance in both classes is of importance, Kubat [61] suggested the use of the *G-measure*

$$G - measure = \sqrt{TPR \cdot TNR},$$

which is the *geometric mean* of the true positive rate and the true negative rate. The G-measure is high when both TPR and TNR are high.

A generalization of the F_1-measure that can account for different weight placed on precision and recall is the F_β*-measure*, defined as

$$F_\beta - measure = \frac{(1+\beta^2) \cdot Rec \cdot Prec}{\beta^2 Rec + Prec},$$

where β corresponds to the relative importance of precision over recall. In the special case of $\beta = 1$, the F_β-measure reduces to the F_1-measure.

Ideally we would like to compare classifiers across a wider range of values of skew c. The *Receiver Operating Characteristic (ROC) curve* [35, 93], is a plot of the *TPR* against the *FPR*. Each (FPR, TPR) point of the curve represents a binary classifier that achieves a different tradeoff among the two measures. The ideal tradeoff depends on the skew c, hence different values of that parameter lead to different models being optimal for the task.

The ROC curves can be used to select the appropriate model (point on the curve) given a particular value of the skew c. If we want a more quantitative measure of the overall performance of different models across any value of c we can instead use the *area under the ROC curve* (*AUROC*) as an evaluation measure[8]. A detailed analysis of the interesting geometric properties of ROC curves and interpretations of the AUROC can be found in [37] and [35].

All these measures are useful to assess the performance of classification under class imbalance, but ultimately they address a different problem, be it ranking of examples (ROC, AUROC) or the balance between precision and recall (F_β-measure). In the cost-sensitive setting we defined in Section 2.3, the cost skew c given in Eq. (2.6) is a known problem characteristic and the goal of the classifier is to minimize the expected misclassification cost. As we will see next, there exist more intuitive visualizations for evaluating probabilistic classifiers with respect to the problem as we defined it in Section 2.3.

Cost curves & Brier curves

Given the nature of our problem, minimizing the expected loss under a cost matrix of Table 2.2, the expected loss itself –as estimated on the test set– is an obvious evaluation measure for a cost-sensitive classifier. More specifically we can use the *normalized cost-sensitive loss* $Q(\theta)$ [28, 53, 66], for a given decision threshold θ, averaged over the test set. The loss is given by

$$Q(\theta) = FNR(\theta) \cdot (1 - z) + FPR(\theta) \cdot z \in [0, 1],$$

[8]Properly normalized, the AUROC is equal to the probability that a classifier will rank a randomly chosen positive instance higher than a randomly chosen negative one, hence it is a good measure for assessing the *ranking* performance of a classifier across a wide range of degrees of skew.

where $FNR(\theta)$ is the false negative rate of the classifier at *decision threshold* θ, $FPR(\theta)$ its false positive rate at that threshold and z the skew of Eq. (2.7).

Lower values of Q are desirable, regardless of the asymmetry. When false positives and false negatives have equal costs, i.e. for $c_{FN} = c_{FP}$, the loss reduces to the expected error rate as measured on the test set. It is normalized in the sense that $Q \in [0,1]$, regardless of the values of c_{FN} and c_{FP}. A skew $z < 0.5$ signifies that negative examples are more important than positives, values $z > 0.5$ that positive examples are more important and $z = 0.5$ corresponds to the symmetric case of all examples being equally important.

As the expected loss is not straightforward to obtain from an ROC curve, Drummond and Holte [28] proposed cost curves for explicitly visualizing the risk of under varying degrees of skew z.

Since a decision threshold θ that is optimal in training may not necessarily be optimal on a test set, $Q(\theta)$ would constitute an optimistic assessment of the cost-sensitive performance. To avoid this, we can follow the '*probabilistic threshold choice*' practice suggested by [53] and instead of *cost curves* [28, 29] ($Q(\theta)$ vs. z for optimal θ on training set), produce *Brier curves* ($Q(z)$ vs. z). That is, we set the threshold θ for each classifier to be equal to the *expected skew*[9] In practice, the expectation is taken by generating the Brier curve under different skew values sampled over some range. The primary benefit w.r.t. cost curves in our experiments will be to account for the sampling effects of the different train/test splits on the class imbalance.

The *area below the Brier curve* is equal to the *Brier Score* (BS) [53]. The BS is a common measure for assessing the quality of probability estimates, first proposed by Brier [10]. It is defined as

$$BS = \frac{1}{N_{test}} \sum_{i=1}^{N_{test}} (\hat{p}(y=1|x) - y)^2 \in [0,1],$$

i.e. the mean squared difference between the probability estimates and the actual class labels of the test examples (if the negative class is denoted with '$y = 0$'), hence the lower it is, the better the estimates of the model.

[9]We can even take this a step further to account for potential changes in skew between training and deployment. We do so by setting the threshold to be equal to the *expected change in skew from training to deployment* z', which relates to the skew ratio during training z_{tr} and the skew ratio during deployment z_{dep} via $z_{dep} = \frac{z_{tr} \cdot z'}{z_{tr} \cdot z' + (1 - z_{tr}) \cdot (1 - z')}$. This change can be interpretted as an instance of *prior probability shift*, a special case of *dataset shift* which refers to changes in general between the joint distribution $p(\mathbf{x}, y)$ during and after training. See [83, 95] for more details on the types of datashet shift.

We thus chose to use Brier curves rather than ROC curves in our experiments in Chapter 8 as they are more intuitive in the cost-sensitive setting (the x-axis represents the degree of imbalance, the y-axis represents the (normalized) risk as estimated on the test set). Furthermore and perhaps more importantly, they are more appropriate for comparing *probabilistic classifiers* due to its aforementioned connection to the BS (on the other hand ROC curves are more appropriate for comparing *rankers*, as mentioned in Footnote 8). As we will see in later chapters, probability estimation will play an important role in our analysis and Brier curves will allow us to draw important conclusions on the quality of probability estimates of different methods, both qualitatively and quantitatively, via their corresponding BS.

2.4.3 Solving a different problem: Neyman-Pearson detection

Another basic measure of performance is the *number of important class misclassifications*. This translates into the *number of rare class misclassifications* when only dealing with imbalanced classes or the *number of expensive class misclassifications* when only dealing with asymmetric costs. In other words, counting the number of false positives (if $c_{FP} > c_{FN}$) or of false negatives (if $c_{FN} > c_{FP}$).

Although this metric has been used in the literature [112, 117], it is easy to see that a low number of expensive class misclassifications is not very indicative of a good cost-sensitive classifier. Simply by assigning all examples to the expensive class, we minimize the number of important class misclassifications. This is not a learned, but rather a deterministic rule (constant classifier) and a very basic one for that matter. Possibly we can do much better than that. A more desirable approach would be to formulate the problem as a *Neyman-Pearson detection problem*.

The general premise is that there exists a *maximum error rate of a given type* that we consider *acceptable*. By fixing the error rate of that type to that value, we then try to find an appropriate classification threshold that *minimizes the error of the other type*. More concretely, a common example is that of a *detection problem*. We want the TPR to be at least α, which implies that the FNR must be at most $1-\alpha$. We fix the FNR and choose the classification threshold θ via e.g. cross-validation for which the FPR is minimized.

Note that the aim here is different from the one we identified as that of this thesis, namely minimizing the expected loss of our misclassifications under known costs.

Neyman-Pearson detection would be more suited for solving *novelty detection*[10] tasks: identifying data that are in some sense *different* than the data used during training. This is essentially a different problem to the one we are addressing, but we deem it prudent to mention it for the sake of completeness. Examples of works in the cost-sensitive boosting literature that are cast as detection problems can be found in [78, 101, 123].

2.5 Making learners asymmetric

There are various approaches to dealing with asymmetric problems. A comprehensive analysis can be found in [52]. Below we discuss some general strategies. We will see that cost-sensitive boosting variants can broadly be classified under the most popular of these.

2.5.1 Shifting the decision threshold

We already discussed a strategy for making a learner asymmetric: *shifting the decision threshold* of the classifier [30, 32, 71]. This approach, is only applicable when the classifier returns a *score*[11]. In particular, if the scores can be viewed as *probability estimates*, we use a threshold of c, i.e. equal to the the skew ratio, instead of the cost-insensitive 0.5, as per Eq. (2.8). Boosting ensembles –as we will see– can produce scores which can also be transformed to probability estimates. As a result they can become cost-sensitive by threshold-shifting.

2.5.2 Manipulating the training data

Another commonly employed tactic for handling asymmetries is preprocessing the training data before giving them as input to a skew-insensitive learner. One popular example is *resampling the data*. This can mean either *oversampling the important class* (rare in imbalanced data, expensive in cost-sensitive), or *undersampling the unimportant class* (common in imbalanced data, cheap in cost-sensitive), or *generating artificial* data based on the original data, or a *combination* of these techniques.

Methods of this family can be used in conjunction with any classifier, since effectively what changes is the dataset we provide the algorithm and not the algorithm

[10]Also known as *anomaly detection*. For a comprehensive overview, see [76, 77, 91].

[11]The concept will be analyzed in more depth in Chapter 7. For now it suffices to think of the score of a classifier on a given example as *a measure of how positive the classifier deems the example*.

itself. The disadvantage of undersampling is that we do not get to use all the available data, thus we lose information about the unimportant class. Oversampling on the other hand can lead to overfitting the important class. These problems are more pronounced when the distribution that generates the data is complex, in the sense that there are distinct clusters (sub-populations) of examples within the classes. Ideally, sampling requires knowing the optimal class distribution of the training set, which is generally unknown. Even if we try to tackle these problems, e.g. by stratified sampling, we still need additional processing time to analyze and process the data [112]. The same is true for methods like SMOTE [15], which create synthetic samples from the minor class instead of exact copies.

As shown empirically in [74], resampling (via oversampling the important class or undersampling the unimportant one) and adjusting the decision threshold yield virtually identical results. Our results and theoretical analysis will also lend support to this observation.

Another method of manipulating the training data, *MetaCost*, proposed by [25] and refined in [75] consists of *relabelling the training examples in such a way that the relabelled dataset can be used to train a skew-insensitive algorithm*. Again, *Bayesian decision theory* is used to assign each example to its risk-minimizing class before using the data as inputs to the cost-insensitive learner. As in the case of resampling, the relabelled data can be used by any classifier, but it does require extra data preprocessing time to reassign the labels.

2.5.3 Modifying the model

Another strategy is to modify the learning algorithm itself. This translates to imposing an additional cost on the model for misclassifying examples of the important class during training. This penalty can bias the model towards the important class. In practice, this translates to changing the underlying loss function minimized by the learning algorithm and is thus specific to each learning algorithm.

2.5.4 Using an asymmetric meta-technique

Margineantu [75] considers the above as the three main methods for incorporating cost sensitivity into learning. Another strategy is to *use a skew-sensitive meta-technique*. This can then be applied to most classifiers, without the need to modify the algorithms themselves (this includes threshold manipulation). Adaboost is a meta-technique that can be used with different base learners, but is not skew-sensitive. The focus of the rest of this thesis will be to examine asymmetric versions of boosting, all of which can be categorized under one of the three main strategies discussed in the previous subsections.

2.6 Potential complications

A number of factors can complicate asymmetric learning tasks. For instance, a *small sample size* can make an imbalanced data problem more difficult [112]. This agrees with our general intuitions regarding generalizing from examples. In an imbalanced dataset, the small sample size is especially detrimental to uncovering regularities in the rare class, since the rare class examples will be too few to allow for reliable estimates.

The *separability* (*complexity of the underlying learning concept*, i.e. class boundary) of the two classes is also an issue. Given linearly separable data, any degree of skew does not seem to cause much problem based on empirical studies [5, 56]. But as the *degree of overlap* among classes grows, the number of rare class misclassifications will also grow significantly in an imbalanced dataset. This makes sense intuitively. The relative importance of errors of different types starts mattering only when the classifier is forced (by linear inseperability) to commit errors and the more errors it is forced to commit (by increasing the class overlap) the more detrimental the effect of how poorly it accounts for it.

Finally, there can also be *within-class imbalances* in the data [55, 112]. The classes can contain within-class *subconcepts* (*distinct clusters of examples*), with different numbers of examples each. The presence of such *subpopulations* within classes increases the dataset's underlying learning concept's complexity and -to make matters worse- it is *usually not known explicitly* (its very *existence*, let alone its exact *nature*).

Although all these nuances will not particularly concern us in this work, a review of the asymmetric learning literature could not have been complete without them. This concludes our introduction to asymmetric learning. We defined our problem as that of assigning new instances to classes so that the expected risk of the classifications is minimized. We discussed the duality between cost-sensitive and imbalanced class learning, establishing why we can focus on the former case without loss of generality. We presented the evaluation measure we will be using, the Brier curve and its properties and advantages over other evaluation criteria. In the next chapter we will cover the theory and existing literature behind AdaBoost, discuss its importance, its limitations and its variants. Finally, we will connect it to the material covered in this chapter by discussing existing ways to make AdaBoost cost-sensitive.

Chapter 3

Boosting and its interpretations

In this chapter we will present a brief history of the principle of *hypothesis boosting* and its most representative algorithm, the celebrated *AdaBoost*. We will discuss the success and rich theoretical depth of AdaBoost, focusing on the many different perspectives by which it can be understood. We will make extensive use of many of these interpretations in the next four chapters to shed light on different aspects of cost-sensitive boosting algorithms. We will also discuss the limitations of AdaBoost, skew-sensitivity being one among them. Finally, we review variants targeted to addressing these limitations, with special emphasis given on cost-sensitive boosting algorithms. We provide a critical overview of the cost-sensitive boosting literature, unifying all existing approaches under a common notation. This will pave the way for a unified theoretical treatment in the chapters to follow.

3.1 Hypothesis boosting

The history of *boosting* formally begins in 1988 with Kearns' seminal paper [59]. However, we can see some of the ideas behind boosting in the optimization and statistics literature as far back as, the *Gauss-Southwell* algorithm for solving linear systems of equations (greedy coordinate descent) and Tukey's method of "twicing", also known as *running median smoothing* [120] (repeated least squares fitting of residuals). A more detailed explanation of these connections can be found in [11].

To understand the basic premise of boosting we need to define two concepts: A *weak learner*[1] is one that is marginally more accurate than random guessing and a *strong learner* is one that achieves error rates arbitrarily close to the irreducible Bayes

[1]We will be using the terms *learner*, *classifier*, *predictor*, *hypothesis* and *model* interchangeably.

error rate. In a seminal paper [59], Kearns formulates the *hypothesis boosting* problem: *Can we convert (i.e. "boost") a weak learner into a strong learner*?

The answer was given by Schapire in [102] and it was *affirmative*. The proposed procedure is as follows: We start with a weak learner h_1, the hypothesis we wish to "boost", trained on a sample of N datapoints. We then train two more models h_2, h_3 on "filtered" versions of the original data. More precisely, the dataset on which we train h_2 consists of a new sample of N datapoints, half of which are misclassified by h_1 and a sample of N points on which h_1 and h_2 *disagree* are used to train h_3. The hypothesis described by the majority vote among the 3 hypotheses, $H(\mathbf{x}) = Majority\ Vote(h_1(\mathbf{x}), h_2(\mathbf{x}), h_3(\mathbf{x}))$ has improved performance over the original hypothesis h_1. We thus manage to "boost" the original hypothesis.

A next step was the celebrated *adaptive boosting* –or *AdaBoost*– algorithm of Freund and Schapire, presented in [42]. It extends the original boosting idea from 3 to M models built 'adaptively'. The basic idea is to construct a (strong) model *sequentially* by combining multiple (weak) models. It is 'adaptive' in the sense that each model tries to 'correct the mistakes' of the previous one and it operates regardless of 'how weak' the weak learner is or how many total rounds of boosting will be performed. It achieves this by training each subsequent model on a new dataset in which the examples misclassified by the previous model are emphasized more and the ones that were correctly classified are emphasized less. We will expand on how this is achieved in the following subsection.

Finally, the AdaBoost algorithm took its popular form -the one we discuss in the present thesis- in [107], by employing predictions that are *weighted* –more precisely *confidence-rated*– combinations of the weak hypotheses instead of *unweighted*[2] (majority voting). We shall now present and analyse the algorithm in detail.

3.2 The AdaBoost algorithm

AdaBoost[3] is an *ensemble learning technique* which constructs a strong classifier H from a weighted vote of multiple weak classifiers h_t, $t = 1, ..., M$. The idea is to train each subsequent model on a new dataset in which the weights of examples misclassified by the previous model are increased and the weights of the correctly classified

[2]Technically, a more precise term than *unweighted* would be *equally weighted with weights* $\frac{1}{M}$.

[3]Here we analyze the *discrete AdaBoost* algorithm, whose predictions on an instance $\mathbf{x}$ are binary, i.e. $h_t(\mathbf{x}) \in \{-1, 1\}$. As we will see, there also exists a version of AdaBoost whose predictions are $h_t(\mathbf{x}) \in [0, 1]$, *real AdaBoost*.

instances are decreased. This can be achieved either by reweighting or by resampling the dataset on each round. In this work we followed the reweighting[4] approach as some evidence exists that it is more stable [44] and works better in practice [94]. We use the version of AdaBoost that employs confidence-rated predictions [107], where each *base learner* is assigned a *confidence coefficient* α_t. The lower the weighted error of the learner, the higher its α_t and the larger its contribution to the final decision.

The algorithm is given N training examples of the form $(\mathbf{x}_i, y_i)$, where $\mathbf{x}_i \in \mathcal{X}$ and $y_i \in \{-1, +1\}$ and a maximum number of rounds M. On the first round of AdaBoost, all training examples are assigned equal weights $D_i^1 = \frac{1}{N}$. On each round t, we learn a model h_t to minimize the weighted misclassification error $\varepsilon_t = \sum_{i: h_t(\mathbf{x}_i) \neq y_i} D_i^t$, and add it to the ensemble, with a voting weight

$$\alpha_t = \frac{1}{2} \log \left(\frac{1 - \varepsilon_t}{\varepsilon_t} \right). \tag{3.1}$$

The distribution D^t is then updated for timestep $t+1$ as

$$D_i^{t+1} = e^{-y_i \alpha_t h_t(\mathbf{x}_i)} \times D_i^t, \tag{3.2}$$

and normalised, each D_i^{t+1} divided by $Z_t = \sum_{j=1}^N e^{-y_j \alpha_t h_t(\mathbf{x}_j)} D_j^t$, to make $\{D_i^{t+1} | i = 1, \ldots, N\}$ a valid distribution.

These will be the weights of each example on the next round[5]. The algorithm terminates when the maximum number M of weak learners have been added to the ensemble or when a base learner with $\varepsilon_t < 1/2$ cannot be found[6]. In the latter case, h_t violates the *weak learning condition* [42], i.e. is not a weak learner. As long as the weak learning condition holds, it is guaranteed that $\alpha_t > 0$. The *final prediction* on a test datapoint $\mathbf{x}$ is given by the sign of the sum of the weak learner predictions $h_t(\mathbf{x})$

[4]Furthermore, the reweighting approach lends itself more naturally to theoretical analysis. Resampling would involve working with expectations which would only complicate the exposition. Reweighting requires that the weak learner be able to handle weighted instances, resampling can be more generally applicable but any results drawn from the former case, carry over –in expectation– to the latter.

[5]An interesting observation is that the weight update rule means that half of the mass of the weight distribution of round $t+1$ is assigned to the misclassified examples of round t and half to the correctly classified ones.

[6]Note that in the binary classification case, a hypothesis h_t with error $\varepsilon_t > 1/2$ can be turned into one with $\varepsilon_t < 1/2$ simply by flipping its predictions.

weighted by their corresponding confidence coefficients

$$H(\mathbf{x}) = sign\left[\sum_{t=1}^{M} \alpha_t h_t(\mathbf{x})\right]. \tag{3.3}$$

The above are summarized in Algorithm 1.

Algorithm 1 The discrete AdaBoost algorithm.

```
Input: Training Data $\{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_N, y_N)\}$, Maximum number of rounds $M$.
Training:
$D_i^1 = \frac{1}{N}$, for $i = 1, 2, \dots, N$.
for $t = 1$ to $M$ do
    Define $\varepsilon_t = \sum_{i:h_t(\mathbf{x}_i) \neq y_i} D_i^t$.
    Obtain a hypothesis $h_t$ that minimizes $\varepsilon_t$ and satisfies the condition $\varepsilon_t < \frac{1}{2}$.
    $\alpha_t = \frac{1}{2} \log\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$.
    $D_i^{t+1} = e^{-y_i h_t(\mathbf{x}_i)\alpha_t} D_i^t$.
    $D_i^{t+1} \leftarrow \frac{D_i^{t+1}}{\sum_{j=1}^{N} D_j^{t+1}}$.
end for
Prediction: $H(\mathbf{x}) = sign\left[\sum_{t=1}^{M} \alpha_t h_t(\mathbf{x})\right]$.
```

3.2.1 AdaBoost as an ensemble method

AdaBoost is an ensemble method, as it aggregates multiple base learners (the weak learners h_t) into one combined classifier system H. As such, boosting is a *meta-algorithm* which can be applied to any supervised base learner. The prediction of the combined system is the weighted majority vote on the weak learners. In this sense Boosting resembles *bootstrap aggregating (bagging)*, where we train multiple models on resampled versions of the original data and then combine them by e.g. majority vote to obtain a prediction.

Upon closer inspection, however, we see that the two approaches are fundamentally different. Bagging is a *parallel model building* technique that reduces the *variance* of the base learner. The variance is the component of the model's error that is due to differences across datasets. By training multiple models on resampled versions of the dataset, i.e. on new datasets with different distributions of data, when we average their predictions, the errors that arise due to variations in the datasets are canceled out. Boosting on the other hand is a *sequential model building* technique that also reduces *bias*. The bias is the component of the error that is due to systematic errors across datasets. By building the model sequentially, on subsequent rounds we can focus more on the examples –or rather, the regions of the input space– that the base learner fails to classify correctly, thus correcting the systematic weakness (bias) of the model.

The above distinction between bagging and boosting is evident from the fact that while bagging does not attain high performance with base learners that exhibit low variance and high bias (e.g. *decision stumps*), boosting does. In fact the most commonly used base learners for AdaBoost are decision stumps (univariate linear models). The resampling aspect of boosting appears to introduce some randomization, a tool used –as we saw in the case of bagging– to reduce the variance component of the model's error, but as we said we can remove this randomization factor by reweighting instead of resampling, in which case boosting actually becomes more stable according to [44].

Compared to other ensemble methods, Breiman conjectures in [9] that AdaBoost imitates the behavior of a *random forest* in its later stages. He recognizes the absence of randomization on the part of boosting, but attributes to the dynamics of AdaBoost any form of apparent random behaviour, while providing some empirical evidence to support this claim.

3.2.2 The successes of AdaBoost

The popularity and resonance of the AdaBoost algorithm in the statistics, machine learning and data mining community merits a special mention. In [131] it was praised as "one of the most influential algorithms in the community". For starters, AdaBoost and its variants have been used in many successful applications including face detection in phone cameras [123] and the Yahoo search engine for ranking webpages [19].

AdaBoost has also been shown in extensive experimental comparisons spanning multiple datasets, such as the ones conducted in [13, 36], to consistently perform well compared to other learning algorithms. This is a pretty strong statement to make about a learning algorithm, given the *no-free-lunch (NFL) theorem* [130] that suggests that no single learning algorithm is guaranteed to exhibit a better than random performance under all data distributions[7]. In addition to that, it usually requires minimal parameter

[7]Of course these theorems make the implicit assumption that the *distribution of such distributions* is *uniform*, i.e. they are all *equally likely*. In practice we are usually looking for patterns in datasets in which patterns *do* exist. Also, most –or even all– such patterns could conceivably have certain underlying commonalities that constitute certain algorithms better at uncovering them than others. For example, a recent paper by Lin & Tegmark [70] argues that the class of physics-based functions of

tuning compared to other well-rounded models like *deep neural networks* and *support vector machines*.

The recognition of AdaBoost as an influential algorithm both as a theoretical tool and in terms of applications it generated even goes beyond the confines of our specialized community. Freund and Schapire were awarded the *2003 Gödel Prize*, one of the most prestigious awards in theoretical computer science, for their original AdaBoost algorithm [42]. The same principles behind AdaBoost, a central one being its adaptive multiplicative weight update rule, have been employed by various subfields of theoretical computer science, as summarized by [3]. As for its more 'exotic' applications, it has been recently used to explain evolution in the presence of sexual recombination [14, 81].

3.2.3 Interpretations

Besides the many applications of boosting, an active subject of ongoing research is its theoretical interpretation. In fact, there have been multiple interpretations of AdaBoost and boosting in general. The various interpretations are –for the most part– complementary, each explaining some aspects of why and how boosting works and on that matter, why it works so well in practice. For example, it has been observed that often adding new models after the training error reaches zero, continues to decrease the test error, i.e. boosting continues to improve its *generalization*. How is AdaBoost in that sense *resistant to overfitting*? This is but one of the questions the various interpretations are called to answer. In addition to explaining its function and justifying its positive properties, the different interpretations proposed can offer insights on how we can improve the algorithm, or how we can adapt it to specific tasks.

Boosting was originally derived under the *Probably Approximately Correct (PAC) learning* framework [102], and so was AdaBoost [42]. In each case, it was proved that the algorithms can, with high probability (the "probably" part), have low *generalization error* (the "approximately correct" part), under any arbitrary choice of desired error, probability of success, or distribution of the data. This directly translates to establishing *theoretical generalization bounds*. However, these bounds are *too lose* to be practical since, paraphrazing [44], "boosting achieves results far more impressive than these bounds would suggest". So, this interpretation, despite its many merits, is not fully satisfactory, as it doesn't explain, for instance, why AdaBoost is resistant to

practical interest (i.e. functions we would like our learners to approximate) is relatively small, because of properties like symmetry, locality, compositionality and polynomial log-probability.

overfitting.

In [43], a *game-theoretic* view of boosting is established. Freund & Schapire define a 2 *player zero-sum game*, the '*boosting game*', with one player being the booster, whose N *pure strategies* are the examples $(\mathbf{x}_i, y_i)$ and the other the weak learner, whose M pure strategies are the weak hypotheses h_t. The loss incurred by the booster is defined as 1 if $h_t(\mathbf{x}_i) \neq y_i$ and 0 otherwise. Applying the *minmax theorem* to solve for the *value (mixed Nash equilibrium) of the game* and adhering to the weak learner condition, the authors prove that there exists a weighted majority of hypotheses which correctly classifies all examples. In other words we can achieve an arbitrarily high accuracy. This is another explanation of why boosting works, which connects it to *linear programming* (and *convex optimization* in general). This link has given rise to new variants such as *LP-boost* [21]. This framework also demonstrates that the choice of weights a_t AdaBoost assigns to the corresponding weak learners, since not calculated *jointly* but rather *sequentially* (i.e. *greedily*), is *sub-optimal.*

A refinement of the PAC-theoretic interpretation of AdaBoost, based on the *Vapnik Chervonenkis (VC) theory* is given in [42]. VC theory captures three conditions that need to be met by a learner in order to be effective and accurate in its predictions: it should be trained with '*sufficient*' training examples, it should *fit those training examples well* (i.e. achieve low training error) and it should be '*simple*'[8]. In the case of AdaBoost, the analysis proves that the training error of the final hypothesis reaches zero in $O(\log(N))$ rounds. New bounds on the generalization error can thus be derived that also account for the complexity of the final hypothesis as captured by its *VC-dimension*[9] which, computed with the use of combinatorial arguments [1, 8], is shown to increase roughly linearly with the number of boosting rounds M. These bounds are again too loose to be of any practical importance. The theory captures the cases in which AdaBoost *does* overfit, i.e. when the training set is not sufficiently large or when the base learners are very complex, but also predicts (incorrectly) that AdaBoost will always overfit i.e. as M grows. Despite factoring in the complexity of the final hypothesis, we still have no explanation as to why AdaBoost exhibits overfitting resistance as the number of learners increases.

An attempt to explain this property was given in [106] through *margin theory*. The idea is that the training error alone only partially captures the quality of a learning

[8]This demand for *low complexity* is an application of the *Occam's razor* principle, as explained in the introduction of the thesis.

[9]In non-technical terms, the VC-dimension of a hypothesis family is the largest number of datapoints, that a hypothesis of that family can perfectly classify, irrespective of their (true) labels.

algorithm. We also need to consider how *confident* the algorithm is in its predictions. AdaBoost often continues to generalize even after all training examples are correctly classified because the confidence by which it correctly classifies them increases with additional rounds of boosting.

The *hypothesis margin* (a.k.a. the *(voting) margin*) is *a combined measure of confidence and correctness* of the classification of a given training example. In its unnormalized form the margin of a given training example $(\mathbf{x}, y)$ is simply the product $yF_t(\mathbf{x})$, where $F_t(\mathbf{x}) = \sum_{\tau=1}^{t} \alpha_\tau h_\tau(\mathbf{x})$ is the output of the ensemble. Once properly rescaled, this can be shown to be equal to the difference between the weighted fraction of the weak classifiers predicting the correct label for that example and the weighted fraction predicting the incorrect label. The higher this difference is, the more confident the final hypothesis. Thus the *magnitude* of the margin measures the confidence of the final hypothesis, and it is easy to see that its *sign* encodes whether the example was correctly classified (positive) or misclassified (negative). This approach has also lead to more refined generalization error bounds based on the margin. Initially [106], it was proposed that AdaBoost maximizes the *minimum margin*. This explanation was later shown to be flawed, after contradictory empirical evidence given in [8], where an algorithm (*arc-gv*) was devised that could maximize the minimum margin over the training set but still achieve higher generalization error. Although [96] proposed that this behaviour could be attributed to the choice of weak learners used by arc-gv, serious doubt was cast over the initial margin explanation.

So in the same paper [96], the fault of the initial interpretation was suggested to be the fact that it does not take into account the *entire margin distribution*. This led to generalization bounds based on the *average margin*. Since then, other bounds have been established, like the *Equilibrium margin (Emargin)* [125], which also take into account the entire margin distribution. More recently, both minimum margin bounds and Emargin bounds have been shown to be special cases of the *k-th margin* bound, and all the previous margin bounds are single margin bounds that are not really based on the whole margin distribution [48]. In other words, margin theory has yet to give a definitive answer as to which function of the margin distribution AdaBoost minimizes. Attempts at devising variants of AdaBoost based on direct maximization of margins have not been entirely successful [104]. Despite its own failings, this is the only explanation for the aforementioned resistance to overfitting that AdaBoost exhibits after the training error reaches zero. In Chapter 6, we will offer a more detailed treatment of the concept of hypothesis margin. We will also be making extensive use of the connection between margins and generalization performance, relating it to properties of the loss functions minimized by different cost-sensitive boosting variants and supporting it with additional experimental results.

An *information-theoretic perspective* came from [60], who viewed boosting as an *entropy projection* technique. In AdaBoost, the new distribution of weights over the training examples is based on the old distribution of weights and the mistakes made by the current weak learner. The authors show that AdaBoost's choice of the new distribution can be seen as an approximate solution to finding a new distribution that is '*closest*' to the old one (i.e. has *minimal Kullback-Leibler divergence* from the old one) subject to the constraint that the new distribution is orthogonal to the vector of mistakes of the current weak learner. In other words, AdaBoost approximately projects the distribution vector onto a hyperplane defined by the mistake vector.

In [18], the authors treat both boosting and logistic regression in a unified way as problems of *minimizing Bregman divergences*. Their framework can generate a large family of *sequential* and *parallel* algorithms. A special case of the former kind is AdaBoost.

In a seminal paper [44], Friedman et al. interpreted the original AdaBoost algorithm as a procedure to minimize the expected exponential loss[10]:

$$L_{Ada}(F_t) = \mathbb{E}_{\mathbf{x},y}\{e^{-yF_t(\mathbf{x})}\},$$

by iterative fitting of terms in the additive model $F_t = \sum_{\tau=1}^{t} \alpha_\tau h_\tau(\mathbf{x})$.

Friedman et al. showed that if we minimize $L_{Ada}(F_t)$ in a *greedy stagewise* manner –i.e. if at stage t we choose the optimal h_t and α_t considering all h_τ and α_τ, for $\tau < t$ as constants– we naturally derive the steps of AdaBoost described in Section 3.2. This is known in the literature as the *'statistical interpretation'* of boosting. Under this light, boosting is just logistic regression where the features of the i-th example are the outputs of the weak learners on that example $h_\tau(\mathbf{x}_i), \tau = 1, \dots, t$, the coefficients of which are fit by *coordinate descent* (first α_1, then α_2 and so on).

A closely related interpretation, given by Mason et al. [80] shows that *functional gradient descent* to greedily fit an additive model $F_t(\mathbf{x}) = \sum_{\tau=1}^{t} \alpha_\tau h_\tau(\mathbf{x})$, derives the

[10]One motivation behind the use of the exponential loss is that it forms a differentiable upper bound to the $0/1$-*loss* [107] as we shall see in Figure 6.1. Also, minimizing $E\{e^{-yF(\mathbf{x})}\}$ is *equivalent to* 2*nd order* to minimizing the negative expected conditional log-likelihood $-E\{\log(P(y|\mathbf{x}))\}$ [44].

same steps, allowing us to choose a different loss function[11] if we wish as long as it is a monotonically decreasing, differentiable, function of the margin. The idea is similar to performing gradient descent in some parameter space. Only now, the 'parameters' are base learners, i.e. functions $h_t \in \mathcal{H}$ defined on the training points, $\mathcal{H}$ being a specific family of models (e.g. decision stumps). The direction of the update (i.e. the specific h_τ to be added at round τ) is chosen to be the one in $\mathcal{H}$ that minimizes the loss[12]. For certain special cases of loss functions, like that of AdaBoost, the size of the optimal (greedy) step α_τ can also be determined in closed form. This interpretation plays a central role in this thesis. It will be discussed in a more technical light in Chapter 4.

A *probabilistic interpretation* is presented in [67], where it is shown that AdaBoost's solution of the minimization of exponential loss and the maximum likelihood for exponential models differ only in the normalization factor employed by the latter to derive a conditional distribution over labels which is absent from AdaBoost. Other than that, both models minimize the same *KL-divergence* objective function subject to the same feature constraints.

Another probabilistic view is that of boosting as a *Product of Experts (PoE)*, in which *experts* are incrementally added [31]. Starting with just 1 expert, we progressively add more to the product. The $(t+1)$-th expert corresponds to the weak learner added on the t-th round of boosting to the ensemble. The condition to add an expert is that the *conditional likelihood* of the $(t+1)$-th PoE be greater or equal to that of the t-th PoE. Edakunni et al., determined that this requirement corresponds to the weak learning condition and derived the AdaBoost algorithm as a special case under this framework. In Chapter 7, we will expand upon this interpretation by giving exact PoE models for the cost-sensitive variants covered in Section 3.3 and examining how their probability estimates are systematically distorted and how to correct for this.

In [100], AdaBoost is studied in the context of *dynamical systems*. AdaBoost is expressed as a *nonlinear iterated map* and the authors analyze the *evolution* of the weight vectors. This perspective offers another understanding of AdaBoost's convergence properties and in the cases where *stable cycles* are observed in the evolution of weights, we can explicitly solve for AdaBoost's output. Furthermore, it is shown that if AdaBoost cycles, it cycles among 'support vectors', i.e., examples that all achieve the smallest margin. The framework shows that AdaBoost does not always converge

[11] Technically, a *functional* defined over some *function space*.

[12] To find the h_τ that minimizes the loss, we compute the functional gradient of the loss functional w.r.t. the h_τ. We then pick the one h_τ that is closest to the functional gradient by minimizing the inner product between h_τ and the gradient over the training examples.

to a maximum margin combined classifier and that 'non-optimal' AdaBoost (where the weak learning algorithm does not necessarily choose the best weak classifier at each iteration) may fail to converge to a maximum margin classifier, even if 'optimal' AdaBoost produces a maximum margin. These findings appear to be suggesting that the margin theory explanation of boosting is not the whole picture. Sure, when its behaviour is cyclic AdaBoost maximizes the margin, but in practice the dynamics of AdaBoost can be *chaotic*. How does it achieve its strong generalization performance, then?

One explanation could be that AdaBoost minimizes a loss function (e.g. exponential loss) but performs *implicit regularization via early stopping* [99, 134]. This approach starts with a variant of AdaBoost in which the α_t coefficients of the ensemble are initially set to a small fixed constant $\alpha > 0$. It can then be shown that running M rounds of boosting is equivalent to applying ℓ_1-*regularization* to the loss function, with the ℓ_1-*norm* of the weights of the weak hypotheses being bounded by a constant equal to $\alpha M > 0$. Therefore early stopping (running AdaBoost for a small number of rounds) is equivalent to regularization. As we said, this interpretation technically does not apply to AdaBoost but to a variant of it. Furthermore, for a large number of rounds, i.e. as $M \to \infty$, the resulting regularization becomes vanishingly weak. In this extreme scenario, it is shown in [99] that the resulting solutions asymptotically maximize the margins of the training examples. In other words, we again reduce to a margin theory explanation.

3.2.4 Limitations

Despite its success, AdaBoost has some important *limitations*. Perhaps the most glaring is that it can be very sensitive to noisy data and outliers as demonstrated in [72]. Some examples are misclassified for good reason: they are anomalies with respect to the underlying pattern. But AdaBoost will continue increasing their weights, no matter how few they are or how far from the current decision boundary or for how many rounds they evade correct classification.

Another obvious limitation is that it is –by the definition of a strong learner– geared towards maximizing accuracy (minimizing missclassification error). Therefore, to adapt it to cost-sensitive applications or when applied to imbalanced-class scenarios, where accuracy is not the desired evaluation measure, we need to modify the algorithm –or how we use its output– appropriately. It has been shown [57] that the weak learner condition imposes no restriction on the precision and recall of the base learner, leading to ensembles of poor overall precision and recall. The same paper also shows that even a cost-sensitive version of boosting, AdaCost [33] depends heavily on the base learner's balance of precision and recall to achieve a good overall performance. The next section is dedicated to the extensive cost-sensitive boosting literature.

Finally, the basic algorithm can be applied only to binary classification. If we deal with a multiclass problem, we will have to use a multiclass variant instead, such as *ECOC* [103], *AdaBoost.MH* [107], *SAMME* [135], *multi-class LogitBoost* [44].

3.2.5 Variants of boosting

Besides those mentioned already, many more variants of boosting have been developed. We will briefly mention a few, just to demonstrate the flexibility of boosting, the power that the various interpretations afford us and to illustrate how simple modifications can help alleviate some of the weaknesses of the original AdaBoost algorithm. If, for example, we regard AdaBoost as a stagewise optimization of an exponential loss function, then by changing the *loss function*, the *optimization method* or -of course-aspects of the *parameterization* of the problem, we get a new boosting algorithm.

For instance, by changing a small aspect of the parameterization and allowing the base learners to output real predictions $h_t(\mathbf{x}) \in [0,1]$ instead of class labels $h_t(\mathbf{x}) \in \{0,1\}$ (or $\{-1,1\}$ to use our notation), we get *Real AdaBoost*, also known as *RealBoost* [44]. We can thus treat the real predictions as *estimates of posterior probabilities* $h_t(\mathbf{x}) = \hat{p}(y=1|\mathbf{x})$. The idea is that now we make use of this measure of how confident the weak learner is in its prediction of each training instance. It is shown experimentally that this algorithm can attain better performance than discrete AdaBoost.

If we substitute the exponential loss of the discrete AdaBoost with the *binomial log-likelihood* and optimize it using *Newton steps*, we get *LogitBoost* [44]. This variant can also attain better performance than discrete AdaBoost as demonstrated in the same paper.

GentleBoost [44], is another variation of AdaBoost, which optimizes the exponential loss using *Newton steps* of *bounded* size, i.e. bounds $\alpha_t h_t$ by y. This algorithm is more stable than both RealBoost and LogitBoost. The reason is that while the latter two can allow their α_t to become infinite, Gentleboost, by its more conservative update rule cannot. This prevents numerical instabilities and produces results comparable, or often better to RealBoost and LogitBoost.

Another variant, *LPBoost* [21], is based on the connections of boosting to *linear* *programming* established by the game-theoretic interpretation. It optimizes a linear loss function similar in spirit[13] to the one that *Support Vector Machines (SVMs)* use in that it combines maximizing the margin between the two classes and allowing some *slack* (some tolerance to misclassified instances). This tradeoff is controlled by a *penalization parameter* as in the case of the SVMs. The motivation behind this approach is to limit AdaBoost's sensitivity to outliers and noise, by allowing a fraction of the training examples to be misclassified.

BrownBoost [40] is another attempt to solve the problem of the susceptibility of AdaBoost to noise. Unlike the variants we have been describing so far, it uses a non-convex loss. BrownBoost 'gives up' on trying to correctly classify hard examples after a number of efforts. It also needs an additional parameter (the training error to be tolerated).

SmoothBoost [110] also attempts to make boosting more resistant to noise. This variant however requires three hyperparameters to be specified by the user, the desired error rate of the final hypothesis, the guaranteed *edge* (the advantage over a random guess) of the hypothesis returned by the weak learner, and the desired margin of the final hypothesis.

An algorithm inspired by the information-theoretic and margin-theoretic interpretations of boosting, is *TotalBoost* [128]. The motivation stems from the entropy projection procedure that AdaBoost performs. AdaBoost is '*corrective*' with respect to the last hypothesis by constraining its edge to be at most $\gamma = 0$. TotalBoost takes this idea one step further by being '*totally corrective*', i.e. by constraining the edges of all previous hypotheses to be at most γ. This γ value is in turn suitably adapted and as a result the hypothesis margin is maximized.

Again, for purposes of countering the sensitivity to noise, a modification to TotalBoost was proposed in the form of *SoftBoost* [127], with an extra *'capping' parameter* added that forces a soft margin upon the final hypothesis, much like the slack variables do in the SVM case. In the paper, the authors show that SoftBoost and LPBoost converge faster and attain lower error rates than BrownBoost and SmoothBoost.

Floatboost [69] is a variant that generally uses a smaller number of weak learners than AdaBoost. It does so by applying a *backtrack step* after each iteration of AdaBoost in order to remove weak learners that hurt the overall accuracy. Floatboost achieves better accuracy than AdaBoost, but at the cost of a longer training time.

[13]Optimizing the objective function of the *SVMs* is a *quadratic programming (QP)* problem, while *LPBoost* is posed as a *linear programming (LP)* problem.

We can also modify AdaBoost to perform *regression* instead of classification. One such variant, which follows directly from the view of boosting as functional gradient descent [80] (and the related statistical interpretation by Friedman et al. [44]), is *Gradient Boosting* [45, 46]. In this case, the final prediction outputs a real number and is built greedily as the weighted combination of base learners (restricted to be a function from a given family) that also output real predictions, so as to minimize the average value of some loss over the training set. The optimal weak learners are selected by performing steepest descent in the space of the predictions on the individual examples of the training set. The optimal weights are chosen by a *line search*. Of course this variant can also be further adapted to solve *ranking problems* [12, 19, 47]. Since any *regressor* can be turned into a *classifier* by appropriately *thresholding* its output, Gradient Boosting can also be used to perform classification. We will not expand upon these variations. Having established the importance, adaptability and theoretical richness of the basic principle of boosting we will now move to the last part of our literature review: cost-sensitive boosting.

3.3 Cost-sensitive boosting algorithms

In the introduction we mentioned that there exist a number of AdaBoost variants proposed to handle cost-sensitive learning tasks [33, 58, 64, 78, 79, 113, 117, 123, 124]. Most of these methods are proposed heuristically, i.e. by introducing ad-hoc changes to steps of the AdaBoost algorithm, rather than by starting from a cost-sensitive problem formulation, e.g. by defining a different *loss function* in place of AdaBoost's exponential loss $L_{Ada} = \mathbb{E}_{\mathbf{x},y}\{e^{-y\sum_t \alpha_t h_t(\mathbf{x})}\}$ and deriving the steps from it. Cost-sensitive boosting methods can broadly be classified into three groups: those that introduce modifications in the training phase of the original AdaBoost, either by modifying the weight update rule of Eq. (3.2), or the calculation of the α_t coefficients of Eq. (3.1), those that modify the initial weights and those that modify the prediction rule Eq. (3.3). The vast majority of approaches fall into the first category.

In this section we will perform a critical review of the cost sensitive Boosting literature spanning 1997-2016, under a consistent and unified notation. Tables 3.1

& 3.2 summarize the changes proposed by each variant[14]. Table 3.1 gives the proposed weight update rule and the initial weight for each method and the corresponding entry in Table 3.2 gives the proposed formula for α_t for the same method.

As a reminder, we focus on cost-sensitive tasks under a constant cost matrix. The cost of misclassifying the i-th example only depends on its class label y_i and is given by

$$c(y_i) = \begin{cases} c_{FN}, & \text{if } y_i = 1 \\ c_{FP}, & \text{if } y_i = -1. \end{cases}$$

All of the methods presented here, with the exception of CSB0, CSB1, AdaCost & AdaCost(β_2), reduce to the original AdaBoost when the task is symmetric, i.e. when $c_{FP} = c_{FN} = 1$. For simplicity, we shall ignore the normalization step of the weight distribution in the subsequent discussion, but we account for it in Table 3.1.

3.3.1 Modifying the training algorithm

CSB0, CSB1 & CSB2

Ting [117] proposed three different modifications of the weight update rule of Eq. (3.2) of AdaBoost which leave the remaining steps of the algorithm intact. The first variant, *CSB0*, changes the weight update rule to

$$D_i^{t+1} = \gamma_t^i D_i^t. \tag{3.4}$$

CSB1 instead uses the following rule to update the weights

$$D_i^{t+1} = e^{-y_i h_t(\mathbf{x}_i)} \gamma_t^i D_i^t. \tag{3.5}$$

Finally, *CSB2* updates the weights according to the rule

$$D_i^{t+1} = e^{-y_i h_t(\mathbf{x}_i)\alpha_t} \gamma_t^i D_i^t. \tag{3.6}$$

[14]Minor variations in e.g. the weight initialization of different approaches have also been examined [65, 117] further increasing the number of variants used in practice. These are excluded from our analysis as, for reasons that will become clear in the subsequent chapters, these changes are not sufficient to grant any of the favourable missing properties to a variant.

In all of them, there is a 'cost parameter' γ_t^i, which assumes the values

$$\gamma_t^i = \begin{cases} c(y_i), & \text{if } h_t(\mathbf{x}_i) \neq y_i \\ 1, & \text{if } h_t(\mathbf{x}_i) = y_i. \end{cases} \tag{3.7}$$

Notice that in all three cases the weights of true positives and true negatives are penalized irrespective of their costs. Also note that *CSB0* and *CSB1* do not involve the confidence coefficient α_t in the weight update rule. Of the three approaches, only *CSB2* reduces to the original AdaBoost when $c_{FP} = c_{FN} = 1$.

Based on the results of comparative studies [78, 79, 112, 117], *CSB1* has been found to outperform *CSB0* and both variants are typically dominated by *CSB2*.

Asymmetric-AdaBoost

Viola & Jones [123] proposed changing the weight update rule of AdaBoost to

$$D_i^{t+1} = e^{-y_i h_t(\mathbf{x}_i)\alpha_t} c(y_i)^{1/M} D_i^t \tag{3.8}$$

and the calculation of α_t to

$$\alpha_t = \frac{1}{2} \log \frac{\sum_{i:h_t(\mathbf{x}_i)=y_i} D_i^t c(y_i)^{1/M}}{\sum_{i:h_t(\mathbf{x}_i)\neq y_i} D_i^t c(y_i)^{1/M}}. \tag{3.9}$$

The rationale behind this approach –dubbed Asymmetric-AdaBoost, henceforth *AsymAda*– is that rather than initializing the weights in a cost-proportional manner, it "distributes the asymmetry evenly across the M weak learners"[15].

Hence, this variant requires the *final* number of weak learners M to be fixed in advance. This is somewhat limiting as the exact number of weak learners in all other boosting variants is set dynamically, the optimal number varying from problem to problem.

[15] Viola & Jones [123] use a multiplicative factor $\left(\frac{c_{FN}}{c_{FP}}\right)^{y_i/2M}$ in Eqs. (3.8- 3.9) rather than $c(y_i)^{1/M}$ used here. But note that both cost setups translate to the same cost ratio (cost of a positive over cost of a negative, Eq. (2.4)), $c_r = \left(\frac{c_{FN}}{c_{FP}}\right)^{1/M}$. Hence they both imply decision problems with equivalent cost matrices, as explained in Chapter 2. We chose to use $c(y_i)^{1/M}$ for simplicity and consistency.

AdaCost & Alternative AdaCost

Fan et al. [33] proposed the use of a 'cost adjustment function' β_t^i in the weight update rule

$$D_i^{t+1} = e^{-y_i h_t(\mathbf{x}_i)\alpha_t \beta_t^i} D_i^t, \tag{3.10}$$

and the calculation of the alpha coefficients[16]

$$\alpha_t = \frac{1}{2}\log\frac{1+\sum_{i:h_t(\mathbf{x}_i)=y_i} D_i^t\beta_t^i - \sum_{i:h_t(\mathbf{x}_i)\neq y_i} D_i^t\beta_t^i}{1-\sum_{i:h_t(\mathbf{x}_i)=y_i} D_i^t\beta_t^i + \sum_{i:h_t(\mathbf{x}_i)\neq y_i} D_i^t\beta_t^i}. \tag{3.11}$$

Fan et al. reasoned that the cost adjustment function must be non-decreasing with respect to $c(y_i)$ when the i-th example is misclassified and non-increasing with respect to $c(y_i)$ when the i-th example is classified correctly. Out of all possible functions, Fan et al. [33] chose to use

$$\beta_t^i = \begin{cases} 0.5(1-c(y_i)), & \text{if } h_t(\mathbf{x}_i) = y_i \\ 0.5(1+c(y_i)), & \text{if } h_t(\mathbf{x}_i) \neq y_i. \end{cases} \tag{3.12}$$

AdaCost does not reduce to AdaBoost when $c_{FP} = c_{FN}$. As for the choice of β_t^i, it makes sense for it to be non-decreasing with respect to $c(y_i)$ in case of misclassification, so that the weights of misclassified examples of the expensive class are increased more than those of the low-cost one. However, as pointed out by Ting [117], the function being non-increasing with respect to $c(y_i)$ when the i-th example is classified correctly is counter-intuitive. It means that, the weights of correctly classified examples of the expensive class are penalized more heavily than those of the cheaper one. In Chapter 6 we will revisit this phenomenon and its implications as an instance of the –undesirable– property of *asymmetry swapping*.

Fan et al. [33] also proposed another variant of *AdaCost*. It also uses Eq. (3.11) to calculate the confidence coefficients, but its weight update rule instead of Eq. (3.10) is

$$D_i^{t+1} = e^{-y_i h_t(\mathbf{x}_i)\alpha_t} \beta_t^i D_i^t, \tag{3.13}$$

In addition to the problems of the first variant, when $c(y_i) = 1$ this algorithm sets the

[16]Alternatively, we could say it leaves the calculation of the α_t coefficients the same as that of AdaBoost, i.e. given by Eq. (3.1), but the calculation of the weighted error ε_t of AdaCost is changed w.r.t. AdaBoost to $\varepsilon_t = \frac{1-\sum_{i:h_t(\mathbf{x}_i)=y_i}\beta_t^i D_i^t + \sum_{i:h_t(\mathbf{x}_i)\neq y_i}\beta_t^i D_i^t}{2}$. In fact this is how Fan et al. [33] present the algorithm. Here we chose to express the changes induced by all algorithms in the formula of α_t, aiming for consistency and ease of comparisons across methods.

weights of correctly classified examples D_i^{t+1} to zero after the first iteration. This variant is thus not examined in the literature and all references to *AdaCost* refer to the first variant. We will also exclude it from further consideration.

AdaCost(β_1) & AdaCost(β_2)

The problems with *AdaCost* led Ting [117] to propose two variants of the algorithm. The first, dubbed *AdaCost(*β_1*)* replaces the weight update rule of AdaBoost with

$$D_i^{t+1} = e^{-y_i h_t(\mathbf{x}_i)\alpha_t c(y_i)} D_i^t, \tag{3.14}$$

and the calculation of α_t with

$$\alpha_t = \frac{1}{2}\log\frac{1+\sum_{i:h_t(\mathbf{x}_i)=y_i} D_i^t c(y_i) - \sum_{i:h_t(\mathbf{x}_i)\neq y_i} D_i^t c(y_i)}{1-\sum_{i:h_t(\mathbf{x}_i)=y_i} D_i^t c(y_i) + \sum_{i:h_t(\mathbf{x}_i)\neq y_i} D_i^t c(y_i)} \tag{3.15}$$

The second variant, *AdaCost(*β_2*)* also uses Eq. (3.14) to update the weights, but leaves the calculation of the confidence coefficients the same as that of AdaBoost. Both variants reduce to AdaBoost when $c_{FP} = c_{FN} = 1$. Both of these, were found to outperform the original *AdaCost*.

AdaC1, AdaC2 & AdaC3

Sun et al. [113], propose three variants of AdaBoost for cost-sensitive classification that involve changes in both the weight update rule and the calculation of the confidence coefficients (or equivalently the calculation of the weighted error). The first one, *AdaC1* substitutes the weight update rule of the original AdaBoost, given in Eq. (3.2), by Eq. (3.14) and the calculation of the α_t with Eq. (3.15). Hence it is the same algorithm as *AdaCost(*β_1*)*.

The second variant, *AdaC2* substitutes the weight update rule of the original AdaBoost, given in Eq. (3.2), by

$$D_i^{t+1} = e^{-y_i h_t(\mathbf{x}_i)\alpha_t} c(y_i) D_i^t, \tag{3.16}$$

and the calculation of the α_t coefficients to

$$\alpha_t = \frac{1}{2}\log\frac{\sum_{i:h_t(\mathbf{x}_i)=y_i} D_i^t c(y_i)}{\sum_{i:h_t(\mathbf{x}_i)\neq y_i} D_i^t c(y_i)}. \tag{3.17}$$

Finally, the third variant, *AdaC3* substitutes the weight update rule of the original AdaBoost, given in Eq. (3.2), by

$$D_i^{t+1} = e^{-y_i h_t(\mathbf{x}_i)\alpha_t c(y_i)} c(y_i) D_i^t, \tag{3.18}$$

and the calculation of the α_t coefficients to

$$\alpha_t = \frac{1}{2}\log\frac{\sum_{i=1}^N D_i^t c(y_i) + \sum_{i:h_t(\mathbf{x}_i)=y_i} D_i^t c(y_i)^2 - \sum_{i:h_t(\mathbf{x}_i)\neq y_i} D_i^t c(y_i)^2}{\sum_{i=1}^N D_i^t c(y_i) - \sum_{i:h_t(\mathbf{x}_i)=y_i} D_i^t c(y_i)^2 + \sum_{i:h_t(\mathbf{x}_i)\neq y_i} D_i^t c(y_i)^2}. \tag{3.19}$$

All three variants reduce to AdaBoost when $c_{FP} = c_{FN} = 1$. Like the rest of the algorithms discussed so far, *AdaC1*, *AdaC2* & *AdaC3* were also originally proposed heuristically [113]. It was later claimed [112] that *AdaC2* can be justified theoretically as a stagewise minimization of a *cost-weighted* version of the expected exponential loss, which for an ensemble of base learners h_t has the form $\mathbb{E}_{\mathbf{x},y}\{c'(y)e^{-y\sum_t \alpha_t h_t(\mathbf{x})}\}$, for some *cost parameter* $c'(y)$. While this is technically true, note that $c'(y)$ is not the actual cost $c(y)$ of example $(\mathbf{x}, y)$. In Chapter 5 we will see what it is equal to and how this deviates from a desirable property imposed by decision theory, making AdaC2 suboptimal. In older empirical studies [78, 79, 113], *AdaC2* has been shown to outperform *CSB2*, *AdaCost*, *AdaC1* and *AdaC3*.

CS-Ada & AdaDB

Mashnadi-Shirazi & Vasconselos [78, 79] proposed replacing the exponential loss function $\mathbb{E}_{\mathbf{x},y}\{e^{-y\sum_t \alpha_t h_t(\mathbf{x})}\}$ minimized by AdaBoost by $\mathbb{E}_{\mathbf{x},y}\{e^{-yc(y)\sum_t \alpha_t h_t(\mathbf{x})}\}$. In Chapter 6 we will see that this loss function has an undesirable property.

The resulting algorithm, *CS-AdaBoost*, lacks a closed form solution for α_t. It consists of generating a pool of weak learners $\{h^k : k = 1, \ldots, K\}$. Subsequently, for each of these weak learners the algorithm calculates a corresponding optimal coefficient α_t^k by solving a hyperbolic equation for α_t^k. Finally, the combination (α_t, h_t) that minimizes the loss is selected to be added to the ensemble at round t. Thus, the optimization of the parameters of *CS-AdaBoost* relies heavily on the pool of the initial weak learners and the hyperparameters of the numerical method involved to solve the hyperbolic equation. It has therefore been characterized as computationally intensive and imprecise [64, 65, 66, 126].

AdaBoostDB [64] is a reformulation of *CS-AdaBoost*. It optimizes the same loss function but follows an alternative optimization scheme. Again, a pool of weak learners $\{h^k : k = 1, \dots, K\}$ is used but the hyperbolic equation, the solution of which is the corresponding optimal coefficient α_t^k is reformulated to a polynomial one which is solved faster (again with the use of an appropriate numerical method). Both approaches were found to perform equally well, subject to numerical errors. The optimization scheme of *AdaBoostDB* was found to be considerably faster (about 200 times) than that of *CS-AdaBoost*, but still very slow compared to the slowest of other approaches discussed here (about 20 times), according to the creators of the method [65, 66].

3.3.2 Modifying the weight initialization

This work is not the first to question the need for cost-sensitive boosting variants. Landesa-Vázquez & Alba-Castro have also criticized their heuristic nature, the fact that they lack the theoretical guarantees of the original AdaBoost algorithm and that many of them are prone to *saturating* (i.e. assigning all examples to the expensive class) [63, 65, 66]. Their suggestion, which they consider 'overlooked or undervalued in the related literature'[65, page 17] is to change the weight initialization of the *i*-th example from $D_i^1 = \frac{1}{N}$ to[17]

$$D_i^1 = c(y_i) \tag{3.20}$$

According to the authors, this approach, which they call '*cost generalized AdaBoost*' – henceforth *CGAda*– provides superior empirical results to all approaches mentioned in the last subsection while preserving the theoretical guarantees of AdaBoost. *CGAda* will also be included in our comparisons.

As for the theoretical properties of *CGAda*, indeed it does not modify the training algorithm, but note that it still suffers from the need to retrain the entire ensemble if the cost imbalance changes between training and testing time, a weakness shared with all other methods excluding *AdaMEC*, the method we will describe next. Moreover, like all other methods examined here, *CGAda* treats the (normalized) AdaBoost predictions as probability estimates. As we will see in Chapter 7, this approach is flawed.

[17]More precisely, Landesa-Vázquez & Alba-Castro [63, 65, 66] propose the following weight initialization: $D_i^1 = \begin{cases} \frac{c_{FN}}{(c_{FP}+c_{FN})\cdot N_+} = \frac{1-c}{N_+}, & \text{if } y_i = 1 \\ \frac{c_{FP}}{(c_{FP}+c_{FN})\cdot N_-} = \frac{c}{N_-}, & \text{if } y_i = -1. \end{cases}$, where N_- is the number of negative training examples and N_+ the number of positive ones. Assuming that the costs c_{FP} and c_{FN} also absorb the class imbalance –as we discussed in Chapter 2– this rule becomes $D_i^1 = \frac{c(y_i)}{c_{FP}+c_{FN}}$. Ignoring the constant scaling factor $\frac{1}{c_{FP}+c_{FN}}$, which –again as discussed in Chapter 2– being the same for all instances does not change the cost-matrix of the problem, we end up with the simple rule we give in Eq. (3.20).

3.3.3 Modifying the prediction rule

Ting [117] proposes an appealing alternative, to train with the *original* AdaBoost, but modify the decision rule in a cost-respecting decision-theoretic manner. This is the AdaBoost with Minimum Expected Cost (AdaMEC) rule:

$$H_{AdaMEC}(\mathbf{x}) = sign\left[\sum_{y\in\{-1,1\}} c(y) \sum_{\tau:h_\tau(\mathbf{x})=y} \alpha_\tau h_\tau(\mathbf{x})\right]. \tag{3.21}$$

Eq. (3.21) reduces to the original AdaBoost decision rule of Eq. (3.3) when the task is symmetric. AdaMEC exploits Bayesian decision theory – assuming the weighted votes are proportional to class probabilities – that is, $\sum_{\tau:h_\tau(x)=1} \alpha_\tau \propto p(y=1|\mathbf{x})$ and $\sum_{\tau:h_\tau(x)=-1} \alpha_\tau \propto p(y=-1|\mathbf{x})$. In Chapter 5 we are going to see that the decision rule of AdaMEC as given by Ting [117] is but a special case of a more general one that does not make the above assumption and allows for generic parametrization of the class probabilities. In the subsequent analysis, we shall refer to this more general approach as AdaMEC.

For now, we have completed our survey of the existing cost sensitive boosting literature. Tables 3.1 & 3.2 summarize the changes to the original AdaBoost proposed by each variant. The original AdaBoost is included for reference. AdaMEC does not modify any of the training steps of AdaBoost, only its prediction rule (hence in both tables it can be viewed as the same entry as AdaBoost). As for AdaCost(β_1), it is the exact same algorithm as AdaC1.

3.4 Preview of the rest of the thesis

In the next 4 chapters, we will use tools from *four* theoretical frameworks: decision theory, functional gradient descent, margin theory, and probabilistic modelling to analyse the variants presented here. Each one of these theoretical angles will turn out to have its own advantages and perspective in the analysis, and the work could not be complete without all four. This is useful to identify properties of the methods, as well as interesting in its own right, since it reflects the diversity of positions from which AdaBoost can be derived [105].

Each of the four perspectives will allow us to formulate a desirable property for a cost-sensitive boosting algorithm. Based on these properties we can make predictions regarding the behaviour and performance of each of the boosting variants under study,

Table 3.1: A summary of cost-sensitive variants showing how they modify the weight updates, along with the weight initialization. The definition of α_t also varies across methods and can be looked up in Table 3.2. The original AdaBoost is included for reference. AdaMEC does not modify any of the training steps of AdaBoost, only its prediction rule. AdaCost(β_1) is the same algorithm as AdaC1.

Algorithm	**Weight Update Rule** $D_i^{t+1} \propto [...] \times D_i^t$	**Initial Weights** D_i^1 **& Cost Adjustment Functions**
AdaBoost [42]	$e^{-y_i \alpha_t h_t(\mathbf{x}_i)}$	where $D_i^1 = \frac{1}{N}$
CGAda [63]	$e^{-y_i \alpha_t h_t(\mathbf{x}_i)}$	where $D_i^1 = c(y_i)$
AdaC1 [112] CSAda [78] AdaDB [64]	$e^{-c(y_i) y_i \alpha_t h_t(\mathbf{x}_i)}$	where $D_i^1 = c(y_i)$
AdaC2 [113]	$c(y_i) e^{-y_i \alpha_t h_t(\mathbf{x}_i)}$	where $D_i^1 = c(y_i)$
AdaC3 [113]	$c(y_i) e^{-c(y_i) y_i \alpha_t h_t(\mathbf{x}_i)}$	where $D_i^1 = c(y_i)$
AsymAda [123]	$c(y_i)^{1/M} e^{-y_i \alpha_t h_t(\mathbf{x}_i)}$	where $D_i^1 = c(y_i)^{1/M}$ (fixed M)
CSB0 [117] CSB1 [117] CSB2 [117]	γ_t^i $\gamma_t^i e^{-y_i h_t(\mathbf{x}_i)}$ $\gamma_t^i e^{-y_i \alpha_t h_t(\mathbf{x}_i)}$	where $D_i^1 = c(y_i)$ and $\gamma_t^i = \begin{cases} c(y_i), & \text{if } h_t(\mathbf{x}_i) \neq y_i \\ 1, & \text{if } h_t(\mathbf{x}_i) = y_i \end{cases}$
AdaCost [33] AdaCost(β_2) [117]	$e^{-\beta_t^i y_i \alpha_t h_t(\mathbf{x}_i)}$	where $D_i^1 = c(y_i)$ and $\beta_t^i = \begin{cases} \frac{1+c(y_i)}{2}, & \text{if } h_t(\mathbf{x}_i) \neq y_i \\ \frac{1-c(y_i)}{2}, & \text{if } h_t(\mathbf{x}_i) = y_i \end{cases}$

which we then verify experimentally in Chapter 8. As we will see in Chapter 7, in some cases we can also correct an algorithm to grant it a missing property, thus improving its performance.

Table 3.2: A summary of cost-sensitive variants showing how they modify the base learner weights α_t. The weight initialization and weight update rule also vary across methods and can be looked up in Table 3.1. The original AdaBoost is included for reference. AdaMEC does not modify any of the training steps of AdaBoost, only its prediction rule. AdaCost(β_1) is the same algorithm as AdaC1.

Algorithm	**Base learner weight α_t**
AdaBoost [42]	$\alpha_t = \frac{1}{2}\log\frac{\sum_{i:h_t(\mathbf{x}_i)=y_i} D_i^t}{\sum_{i:h_t(\mathbf{x}_i)\neq y_i} D_i^t}$
AdaCost(β_2) [117]	
CSB(0,1,2) [117]	
CGAda [63]	
AdaC1 [113]	$\alpha_t = \frac{1}{2}\log\frac{1+\sum_{i:h_t(\mathbf{x}_i)=y_i} D_i^t c(y_i)-\sum_{i:h_t(\mathbf{x}_i)\neq y_i} D_i^t c(y_i)}{1-\sum_{i:h_t(\mathbf{x}_i)=y_i} D_i^t c(y_i)+\sum_{i:h_t(\mathbf{x}_i)\neq y_i} D_i^t c(y_i)}$
AdaC2 [113]	$\alpha_t = \frac{1}{2}\log\frac{\sum_{i:h_t(\mathbf{x}_i)=y_i} D_i^t c(y_i)}{\sum_{i:h_t(\mathbf{x}_i)\neq y_i} D_i^t c(y_i)}$
AdaC3 [113]	$\alpha_t = \frac{1}{2}\log\frac{\sum_{i=1}^N D_i^t c(y_i)+\sum_{i:h_t(\mathbf{x}_i)=y_i} D_i^t c(y_i)^2-\sum_{i:h_t(\mathbf{x}_i)\neq y_i} D_i^t c(y_i)^2}{\sum_{i=1}^N D_i^t c(y_i)-\sum_{i:h_t(\mathbf{x}_i)=y_i} D_i^t c(y_i)^2+\sum_{i:h_t(\mathbf{x}_i)\neq y_i} D_i^t c(y_i)^2}$
AsymAda [123]	$\alpha_t = \frac{1}{2}\log\frac{\sum_{i:h_t(\mathbf{x}_i)=y_i} D_i^t c(y_i)^{1/M}}{\sum_{i:h_t(\mathbf{x}_i)\neq y_i} D_i^t c(y_i)^{1/M}}$ (fixed M)
AdaCost [33]	$\alpha_t = \frac{1}{2}\log\frac{1+\sum_{i:h_t(\mathbf{x}_i)=y_i} D_i^t \beta_t^i-\sum_{i:h_t(\mathbf{x}_i)\neq y_i} D_i^t \beta_t^i}{1-\sum_{i:h_t(\mathbf{x}_i)=y_i} D_i^t \beta_t^i+\sum_{i:h_t(\mathbf{x}_i)\neq y_i} D_i^t \beta_t^i}$ See β_t^i in Table 3.1.
CSAda [78]	Numerical solution of hyperbolic equation. No closed form.
AdaDB [64]	Numerical solution of polynomial equation. No closed form.

Chapter 4

The functional gradient descent view

We now begin our journey of examining the cost-sensitive boosting literature through different theoretical perspectives. Each of these will provide a distinctive insight into how different variants operate and each will define a desired property that some variants will satisfy and others will not. In this chapter we shall delve deeper into the *functional gradient descent* view of boosting. We will use it to infer the *loss function* minimized by each variant. This way, we make the objective function of each algorithm explicit, which allows us to examine them all, even many previously regarded as heuristics –which constitute the majority of the proposed methods– under the same theoretical framework.

We will see which algorithms are efficiently optimizing their loss function and which take suboptimal steps. This distinction allows us to divide the set of cost-sensitive boosting methods into two groups: those that satisfy the property of being consistent with the functional gradient descent and those that do not. We do make a note though that this property, although desirable is not necessarily crucial for the success of a cost sensitive algorithm and briefly mention how suboptimal steps can actually lead to better generalization.

Most importantly, the loss function itself will play an integral role for the three chapters to follow. It is the loss function each algorithm attempts to minimize that defines the properties of each variant that we will introduce in these chapters. Ultimately these will all be properties of the loss function we derive in the present chapter.

4.1 Boosting as functional gradient descent

We shall follow the view of boosting as functional gradient descent (FGD) [44, 80], adopting Mason et al's formulation. This perspective views boosting as a procedure that greedily fits an additive model $F_t(\mathbf{x}) = \sum_{\tau=1}^{t} \alpha_\tau h_\tau(\mathbf{x})$ by minimizing the *empirical risk* (i.e the average loss on a training set),

$$J(F_t(\mathbf{x})) = \frac{1}{N}\sum_{i=1}^{N} L(y_i F_t(\mathbf{x}_i)), \tag{4.1}$$

where $L(y_i F_t(\mathbf{x}_i))$ is a *monotonically decreasing* loss function[1] of $y_i F_t(\mathbf{x}_i)$, the *margin* on the i-th example.

The FGD approach tells us to add the model h_{t+1} that most rapidly reduces Eq. (4.1). This model turns out to be that which minimizes the weighted error for a new weight distribution D^{t+1}, which can be written, as Mason et al. observed, in terms of the functional derivative:

$$D_i^{t+1} = \frac{\frac{\partial}{\partial y_i F_t(\mathbf{x}_i)} J(F_t(\mathbf{x}))}{\sum_{j=1}^{N} \frac{\partial}{\partial y_j F_t(\mathbf{x}_j)} J(F_t(\mathbf{x}))} = \frac{\frac{\partial}{\partial y_i F_t(\mathbf{x}_i)} L(y_i F_t(\mathbf{x}_i))}{\sum_{j=1}^{N} \frac{\partial}{\partial y_j F_t(\mathbf{x}_j)} L(y_j F_t(\mathbf{x}_j))}. \tag{4.2}$$

The voting weight α_t is the *step size* in the direction of the new weak model h_t.The *optimal* step size is the one that minimizes the empirical risk on the training set, i.e.

$$\alpha_t^* = \arg\min_{\alpha_t} \left[\frac{1}{N}\sum_{i=1}^{N} L\Big(y_i(F_{t-1}(\mathbf{x}_i) + \alpha_t h_t(\mathbf{x}_i))\Big)\right]. \tag{4.3}$$

Under Eq. (4.2), a given loss function $L(y_i F_t(\mathbf{x}_i))$ implies a specific form of weight updates

$$D_i^{t+1} \propto -\frac{\partial}{\partial y_i F_t(\mathbf{x}_i)} L(y_i F_t(\mathbf{x}_i)). \tag{4.4}$$

Conversely, it will then be the case that the weight updates D_i^{t+1} imply a specific family of equivalent loss functions via

$$L(y_i F_t(\mathbf{x}_i)) \propto \int -D_i^{t+1} d(y_i F_t(\mathbf{x}_i)). \tag{4.5}$$

[1]To avoid confusion, when we refer to the *loss (function)*, we will mean the loss function $L(y_i F_t(\mathbf{x}_i))$ minimized on the t-th round of boosting. In the next chapter we will briefly introduce another concept, the *global loss (function)*, which will be the loss function of which the *final* model built by the boosting algorithm is a minimizer.

Having made this observation we can now use Eq. (4.5) to derive the loss function of any given boosting variant.

4.1.1 Example: AdaBoost

Let us now take a moment to give a concrete example of this two way connection between loss functions and weight updates, for the special case of AdaBoost. In AdaBoost the loss w.r.t. the margin of the i-th example is

$$L_{Ada}(y_i F_t(\mathbf{x}_i)) = e^{-y_i F_t(\mathbf{x}_i)}, \tag{4.6}$$

so, by Eq. (4.4), the weight update rule will be

$$D_i^{t+1} = \frac{e^{-y_i h_t(\mathbf{x}_i)\alpha_t} \times D_i^t}{\sum_{j=1}^N e^{-y_j h_t(\mathbf{x}_j)\alpha_t} \times D_j^t}, \tag{4.7}$$

which is indeed the familiar weight update rule of AdaBoost.

Following the inverse derivation, taking the weight update rule of Eq. (4.7) as given and inferring the loss function via Eq. (4.5), we recover that any member of the family of loss functions

$$L(y_i F_t(\mathbf{x}_i)) \propto e^{-y_i F_t(\mathbf{x}_i)} + K, \tag{4.8}$$

where K is some constant w.r.t. the margin $y_i F_t(\mathbf{x}_i)$, would lead to weights updated via Eq. (4.7). Setting the integration constant to $K = 0$, we obtain the loss of Eq. (4.6), scaled by some constant. The constant can also be ignored, giving us the familiar exponential loss of AdaBoost. However, any function of the family defined by Eq. (4.8) once minimized w.r.t. α_t across the entire training set as per Eq. (4.3) will give us the original α_t from Adaboost, in Eq. (3.1).

4.1.2 Why a margin loss?

Mason et al's functional gradient descent (FGD) framework requires that the loss function be a monotonically decreasing function of the margin $y_i F_t(\mathbf{x}_i)$ [80]. Let us now explain why this is a reasonable requirement, before we start investigating the loss functions of the various cost-sensitive boosting variants.

First, we remind the reader that the *(voting) margin* is a combined measure of confidence and correctness of the classification of the i-th training example. Its sign indicates the correctness of the classification ($y_iF_t(\mathbf{x}_i) > 0$ if the example has been classified correctly and $y_iF_t(\mathbf{x}_i) < 0$ if it has been misclassified). The magnitude of $y_iF_t(\mathbf{x}_i)$ represents the confidence of the model F_t in the classification of $\mathbf{x}_i$. If $L(y_iF_t(\mathbf{x}_i))$ is monotonically decreasing, the more it deviates from 0 hence the higher the confidence of the ensemble in its prediction is, the lower its loss will be if it classifies $\mathbf{x}_i$ correctly and –conversely– the higher its loss if $\mathbf{x}_i$ is misclassified.

Thus it is *sensible* for the loss function to be a monotonically decreasing function of the margin. Misclassifications are penalized more than correct classifications, high-confidence misclassifications are penalized more than low-confidence ones and low-confidence correct classifications are penalized more than high-confidence ones. We can see that such a loss function will guide the ensemble towards high confidence correct classifications.

4.2 Uncovering the hidden loss

Only a handful of the existing cost-sensitive boosting variants, namely CSAda [78, 79], AdaDB [64], CGAda [63, 65] have been derived by first *explicitly* specifying a loss function L and then following the steps of FGD. Hence the first step in our analysis is to follow the inverse derivation for the remaining methods, taking their modified weight update rule and inferring the loss function via Eq. (4.5). Once this is done, we can study all methods under a common theoretical framework and compare them on the basis of four properties that follow from their loss function.

In Table 4.1 we give the underlying loss function for each cost-sensitive boosting variant. To get it, we first express D_i^{t+1} as a function of the margin $y_iF_t(\mathbf{x}_i)$, by recursively substituting D_i^t up to the initial weight D_i^1. The equation for the weight update D_i^{t+1} of each method and its weight initialization D_i^1 were given in Table 3.1. We then use Eq. (4.5) to derive a loss function, itself being a function of the margin. This approach works fine for all but the four methods we discuss separately in the next subsection.

4.2.1 Methods that do not minimize a margin loss

Note that for CSB0, CSB1, AdaCost and AdaCost(β_2) we cannot express the underlying loss as a function of the margin $y_iF_t(\mathbf{x}_i)$. This is because D_i^{t+1} itself is not a function of the margin. In fact, in these four methods, D_i^{t+1} is not even a function of the additive model $F_t(\mathbf{x}_i) = \sum_{\tau=1}^{t} \alpha_\tau h_\tau(\mathbf{x}_i)$ used in our classifications, but of some other function of the weak learner outputs $\{h_\tau(\mathbf{x}_i)|\tau = 1,\dots,t\}$. It is therefore not very surprising that, as we discussed in Chapter 3, these four variants also happen to be the only methods examined here that do not reduce to the original AdaBoost when the task is symmetric. As we will see in Chapter 8, they are also found to be among the five worst-performing variants in our empirical comparisons.

Bearing in mind that due to the heuristic nature of these methods we should not expect them to necessarily neatly fit into any theoretical framework, we can still derive a loss function for them, albeit not of the form $L(y_i F_t(\mathbf{x}_i))$. We do so as follows. Again we recursively substitute D_i^t up to D_i^1 (again given in Table 3.1) so as to express D_i^{t+1} as some function g of the true label y_i and the individual weak learner predictions $\{h_\tau(\mathbf{x}_i)|\tau = 1,\dots,t\}$. We then choose a function L such that

$$L(g(y_i, \{h_\tau(\mathbf{x}_i)|\tau = 1,\dots,t\})) \propto \int -D_i^{t+1} d(g(y_i, \{h_\tau(\mathbf{x}_i)|\tau = 1,\dots,t\}))$$

as the underlying loss function.

Hence we can make our framework general enough to derive loss functions for these methods as well. But not being monotonically decreasing functions of the margin –or even functions of the margin for that matter– these loss functions are not as well-behaved as those addressed by other methods, as explained in Subsection 4.1.2. Therefore, for all subsequent discussion it is sufficient[2] to just note that their loss is not a monotonically decreasing function of the margin.

4.3 FGD-consistency

The functional gradient descent view allows us to infer the loss function that each method attempts to minimize through its weight update rule. We found that some algorithms, namely CSB0, CSB1, AdaCost & AdaCost(β_2) do not minimize a monotonically decreasing function of the margin. These, of course, do not follow the FGD framework of Mason et al.

[2]For the interested reader, the loss function of CSB0 is a function of the quantity $c(y)^{q-1}$, where $q \geq 0$ is the number of models that misclassified example $\mathbf{x}$. That of CSB1 is a function of $c(y)^{q-1} e^{-y\sum_{\tau=1}^{t} h_\tau(\mathbf{x})}$. Finally, the loss of AdaCost & AdaCost(β_2) is a function of $e^{-y\sum_{\tau=1}^{t} \beta_\tau(\mathbf{x})\alpha_\tau h_\tau(\mathbf{x})}$, where $\beta_\tau(\mathbf{x})$ was given in Table 3.1.

Table 4.1: Loss function $L(yF_t(\mathbf{x}))$ minimized by each cost-sensitive boosting variant. To derive it, we used Eq. (4.5) and the weight update rule and initial weight of each method given in Table 3.1.

Method	Loss Function $L(yF_t(\mathbf{x}))$
Adaboost [42]	$e^{-yF_t(\mathbf{x})}$
AdaMEC	”
CGAda [63, 65, 66]	$c(y)e^{-yF_t(\mathbf{x})}$
AsymAda [123]	$c(y)^{t/M}e^{-yF_t(\mathbf{x})}$
CSAda [78, 79]	$e^{-c(y)yF_t(\mathbf{x})}$
AdaDB [64]	”
AdaC1 [112, 113]	”
CSB2 [117]	$c(y)^{q-1}e^{-yF_t(\mathbf{x})}$, where q models have misclassified $\mathbf{x}$.
AdaC2 [112, 113]	$c(y)^t e^{-yF_t(\mathbf{x})}$
AdaC3 [112, 113]	$c(y)^{t-1}e^{-c(y)yF_t(\mathbf{x})}$
CSB0 [118] CSB1 [117] AdaCost(β_2) [117] AdaCost [33]	Cannot express loss as a function of $yF_t(\mathbf{x})$.

For the remaining algorithms, since modifying the weight updates implies a specific family of equivalent loss functions by Eq. (4.5), then any change in the distribution update should be reflected in the calculation of voting weights α_t, according to Eq. (4.3). Otherwise, the chosen α_t are sub-optimal for the purpose of the stagewise minimization of the loss.

Therefore, the FGD view divides the literature into two families – those which are *consistent* with it and those that are not.

Definition: FGD-consistent *A boosting method is functional gradient descent (FGD)-consistent if it uses a distribution update rule and voting weights α_t that are both consequences of greedily optimising the same, monotonically decreasing, loss function of the margin. Otherwise the method is FGD-inconsistent.*

4.3.1 Checking for FGD-consistency

For generality and to avoid unnecessary repetition of what is effectively the same thought process, we shall not present proofs for each method individually, but rather a general scheme for checking any given boosting algorithm for FGD-consistency. This has the added benefit to cover more variants than those studied in this thesis.

We start with Eq. (4.5), where D_i^{t+1} is the weight update rule of the given algorithm. If its *RHS* is not a monotonically decreasing function of the margin $y_i F_t(\mathbf{x}_i)$, then neither is the loss $L(y_i F_t(\mathbf{x}_i))$ and the method is FGD-inconsistent under our definition. As we saw this is the case for CSB0, CSB1, AdaCost & AdaCost(β_2).

For all other methods examined here, by recursively applying the weight update given in Table 3.1 up to D_i^1, it can be shown that D_i^{t+1} can be written as a monotonically decreasing function of the margin. In all methods it is of the general form:

$$D_i^{t+1} \propto K_1(i) e^{-K_2(i) y_i F_t(\mathbf{x}_i)} \tag{4.9}$$

where $K_1(i)$ and $K_2(i)$ are non-decreasing functions of $c(y_i)$, the cost of the i-th example (e.g. $K_1(i) = 1$ or $K_2(i) = 1$ are also admissible).

Combining Eq. (4.5) and Eq. (4.9), we obtain

$$\begin{aligned} L(y_i F_t(\mathbf{x}_i)) &\propto \int -K_1(i) e^{-K_2(i) y_i F_t(\mathbf{x}_i)} d(y_i F_t(\mathbf{x}_i)) \\ &= \frac{K_1(i)}{K_2(i)} e^{-K_2(i) y_i F_t(\mathbf{x}_i)} + K, \end{aligned} \tag{4.10}$$

where K is constant w.r.t. the margin $y_i F_t(\mathbf{x}_i)$.

Setting $K = 0$ and ignoring the scaling factor, we can limit ourselves for simplicity to a single member of the family of functions $L(y_i F_t(\mathbf{x}_i))$ and use

$$L(y_i F_t(\mathbf{x}_i)) = \frac{K_1(i)}{K_2(i)} e^{-K_2(i) y_i F_t(\mathbf{x}_i)}. \tag{4.11}$$

The loss functions of this form are the ones we present in Table 4.1. This is of course an optional step, as any loss of the form of Eq. (4.10) is equivalent for optimization purposes, i.e. it is minimized by the same $F_t(\mathbf{x}_i)$.

If the given form of the voting weight of the t-th base learner minimizes the empirical risk under $L(y_i F_t(\mathbf{x}_i))$ on the training set, i.e. is the one given by Eq. (4.3), then the method is by our definition FGD consistent. Otherwise it is FGD-inconsistent. The results are shown below.

FGD-consistent	**FGD-inconsistent**
Ada, AdaMEC, CGAda, AsymAda,	CSB0, CSB1, CSB2, AdaC1,
AdaC2, CSAda[3], AdaDB[3]	AdaC3, AdaCost, AdaCost(β_2)

For the rest of the thesis, the treatment of FGD-consistency serves two purposes. Firstly, to identify the precise loss function an algorithm is minimizing. Once this is known, additional properties can be evaluated, revealing whether this loss is sensible in a cost-sensitive scenario. Secondly, FGD-consistency is useful for knowing whether the loss is being *efficiently* optimized.

4.3.2 When 'suboptimal' steps are better than 'optimal'

Our initial hypothesis was that methods that are FGD-inconsistent will be outperformed by those that are FGD-consistent. However, as the empirical results of Chapter

[3] CSAda & AdaDB are FGD-consistent under our definition. However, the α_t under the loss function they minimize has no closed form and requires approximation. The two variants use different approximations; CSAda requires the solution of a hyperbolic equation, AdaDB that of a polynomial. This makes both methods computationally intensive and subject to approximation error in the α coefficients. Thus *in practice* they are not guaranteed to be FGD-consistent.

8 will show, this only reveals part of the story. As the history of Machine Learning has shown many times, an intuitive choice of a good heuristic can result in practical advances that outperform a sophisticated theory. We certainly do not regard the methods we name above as 'inconsistent' as *necessarily* inferior to 'consistent' ones.

We already explained why minimizing a loss which is not a monotonically decreasing function of the margin is not as sensible as minimizing one that is. CSB0, CSB1, AdaCost and AdaCost(β_2) are thus expected to not perform as well as the other methods. And indeed they do not. But the definition of FGD-consistency also classifies as FGD-inconsistent methods like CSB2, AdaC1 and AdaC3 which *do* minimize a margin loss, albeit take *suboptimal steps* in the direction of minimizing it, in the sense that the α_t coefficients they use are not the ones that minimize said loss on the training set. For methods of the latter category, claiming that they are *necessarily* inferior to the ones that use optimal steps would be unjustified for two reasons.

The first reason is that the loss itself might not be a very good choice. As we will see in the next chapters, some methods use loss functions that overemphasize the costs or that swap class importance during training. In these situations, not optimizing an objective function which is problematic might be a better choice than actually optimizing it. This might explain –for instance– why AdaC1 tends to outperform CSAda in our experiments presented in Chapter 8. Both methods employ the same loss for the purpose of determining the weight updates. In Chapter 6 we will see that this loss has a flaw which hurts generalization. While CSAda attempts to numerically estimate optimal steps α_t in the direction of minimizing said loss, AdaC1 doesn't. And interestingly, we see it tends to perform better.

The second reason why suboptimal step sizes might lead to better generalization performance is related to *regularization*. Friedman [45] was the first to suggest rescaling the optimal steps α_t of Eq. (4.3) by a factor $\nu \in (0,1)$ which can be thought of as a *learning rate*. The concept is known as *shrinkage* or *regularization via rescaling*, and the idea is to take smaller gradient descent steps. As a result of scaling down the step size, the *ideal path*[4] towards the minimum is more finely approximated. The drawback, of course, is that as the steps are smaller, more of them are needed to reach the minimum. This means more boosting rounds need to be performed hence additional computational time both during training and during querying (deployment). All this is very nicely summarized in Figure 4.1, taken from [116].

[4]We can think of the *ideal path* as the shortest path in the parameter space that connects the current model to the minimum, if full knowledge of the parameter space is given and there is no restriction on how we traverse it.

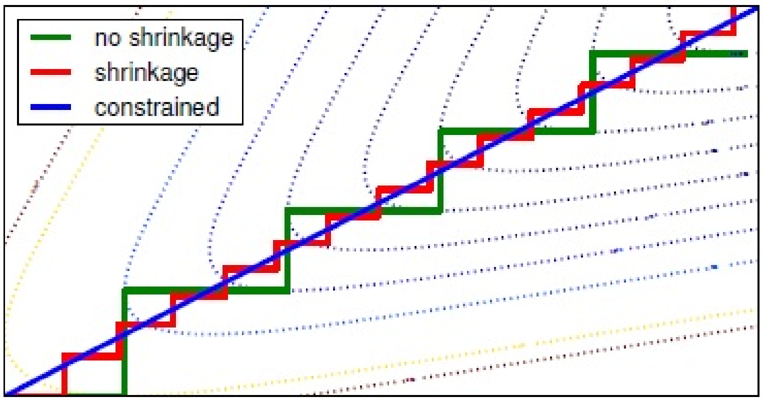

Figure 4.1: Blue: the ideal path towards the minimum of the loss function. Green: path followed by taking boosting steps of optimal size. Red: path followed by taking boosting steps in which shrinkage is applied. The red path more finely approximates the optimal one (blue), than the green one, but requires more steps to reach the minimum. Image taken from [116].

We cannot directly relate this second situation to any of our empirical results. We have not explored to what extend the suboptimal step sizes taken by FGD-inconsistent methods can be viewed as applying shrinkage. This goes beyond the scope of this thesis and is left as a subject for future work.

For now we can consider FGD-consistency a generally favourable property (given a sensible loss function). If anything, once we do have the optimal step size, shrinkage can be applied to it to explore its regularization effect. In the following chapters, we will use the loss functions we derived in the previous section to examine three additional properties of the cost-sensitive boosting variants, further investigating if their loss is indeed sensible in the context of cost sensitive boosting.

Chapter 5

The decision-theoretic view

In this chapter we will examine cost-sensitive boosting algorithms through the lens of decision theory. Decision theory is the study of making decisions in the general case where the costs and probabilities of different outcomes can vary. It gives us optimal decision rules when these quantities –or reasonably good estimates thereof– are known. We already saw the optimal decision rule for a given cost matrix when the goal is to minimize the expected misclassification cost in Chapter 2.

The first step in our analysis from the decision-theoretic perspective is to observe that each cost-sensitive boosting variant constructs (an approximation to) the minimizer of the corresponding loss function we derived in Chapter 4 by applying the FGD framework. After deriving the minimizer of said loss, we use it in conjunction with the ensemble prediction rule to derive the decision rule the variant implements in terms of the probability of a given example being positive and the misclassification costs. As we will see, some of these decision rules do indeed correspond to the optimal decision rule given the probability of a given example being of the positive class (or in the absence of this, its estimate) and the costs, but others do not. Based on this we can divide the literature into those algorithms that are consistent with minimizing the expected cost of misclassifications given the cost matrix of the problem and those that are inconsistent.

A byproduct of this analysis is a generalized version of AdaMEC, one that –unlike the original one proposed by Ting [117]– does not assume a specific form of probability estimates. To facilitate discussion, we will also introduce a new concept, that of the *global loss*, to refer to the loss function of which the *final* model built by the boosting algorithm is a minimizer (as opposed to the loss function minimized in each round). It is actually this global loss that defines the final decision rule a variant implements.

5.1 Loss minimizers

In the previous chapter, we investigated boosting variants from a functional gradient descent viewpoint. In doing so, we uncovered the loss function each cost-sensitive boosting variant *attempts to minimize in a greedy way*. This means that each variant is attempting to construct the model that minimizes its corresponding loss. The minimizer each method is approximating via a greedily fitted additive model can be inspected in Table 5.1.

Table 5.1: The population minimizer of the loss function of each method. This is directly derived from the loss function of Table 5.2 and results to the decision rule shown on the same table (after replacing true probabilities with estimates as is the case in practice). We have made use of the fact that for AsymAda the final round of boosting will be $t = M$. For AdaMEC the decision rule was *defined* as in Table 5.2 which in turn specifies a loss whose minimizer is the one showed here.

Method	Population Minimizer $F^*(\mathbf{x})$ of Loss $L(F)$
Adaboost [42]	$\frac{1}{2}\log\frac{p(y=1\vert\mathbf{x})}{p(y=-1\vert\mathbf{x})}$
AdaMEC	$\frac{1}{2}\log\frac{p(y=1\vert\mathbf{x})}{p(y=-1\vert\mathbf{x})}+\frac{1}{2}\log\frac{c_{FN}}{c_{FP}}$
CGAda [63, 65, 66]	”
AsymAda [123]	”
CSAda [78, 79]	$\frac{1}{c_{FN}+c_{FP}}\log\frac{p(y=1\vert\mathbf{x})}{p(y=-1\vert\mathbf{x})}+\frac{1}{c_{FN}+c_{FP}}\log\frac{c_{FN}}{c_{FP}}$
AdaDB [64]	”
AdaC1 [112, 113]	”
CSB2 [117]	$\frac{1}{2}\log\frac{p(y=1\vert\mathbf{x})}{p(y=-1\vert\mathbf{x})}+\frac{q-1}{2}\log\frac{c_{FN}}{c_{FP}}$, where q models have misclassified $\mathbf{x}$.
AdaC2 [112, 113]	$\frac{1}{2}\log\frac{p(y=1\vert\mathbf{x})}{p(y=-1\vert\mathbf{x})}+\frac{t}{2}\log\frac{c_{FN}}{c_{FP}}$, where t is the number of boosting rounds.
AdaC3 [112, 113]	$\frac{1}{c_{FN}+c_{FP}}\log\frac{p(y=1\vert\mathbf{x})}{p(y=-1\vert\mathbf{x})}+\frac{t}{c_{FN}+c_{FP}}\log\frac{c_{FN}}{c_{FP}}$, where t is the number of boosting rounds.
CSB0 [118] CSB1 [117] AdaCost(β_2) [117] AdaCost [33]	Cannot express population minimizer as a function of c_{FP}, c_{FN} & $p(y=1\vert\mathbf{x})$

5.1.1 Minimizer derivation

The minimizer shown in Table 5.1 is directly derived from the loss function of the boosting variant in question, shown in Table 4.1 and repeated in Table 5.2 for ease of reference. For most methods examined here, the minimizer F^* of their expected loss on the *final* round[1] of boosting $L(F_t(\mathbf{x}))$ can be written as a function Ψ of the true conditional class probability $p(y=1|\mathbf{x})$ and the cost setup, i.e.

$$F^*(\mathbf{x}) \;=\; \arg\min_{F_t} E_{\mathbf{x}y}\Big\{L(F_t(\mathbf{x}))\Big\} = \Psi(p(y=1|\mathbf{x}), c_{FN}, c_{FP}).$$

The loss denoted by $L(F_t(\mathbf{x}))$ is none other than the loss $L(yF_t(\mathbf{x}))$ we discussed in the previous chapter. Here there is no need to write it as a function of the margin $yF_t(\mathbf{x})$, so we only consider it a function of the output of the additive model constructed at round t, $F_t(\mathbf{x})$, for simplicity. Of course, the exact form of the population minimizer $F^*(\mathbf{x})$ of a given loss function $L(F_t(\mathbf{x}))$ which is differentiable w.r.t. $F_t(\mathbf{x})$ can be obtained by simply setting,

$$\frac{\partial L(F_t(\mathbf{x}))}{\partial F_t(\mathbf{x})} = 0,$$

and solving for $F_t(\mathbf{x})$.

A notable exception to the above pattern is AdaMEC –added in Table 5.1 for completeness. The decision rule of AdaMEC is *defined* as per Table 5.2, motivated by decision theory, as we will explain in Section 5.3. Thus for AdaMEC, we follow an inverse reasoning to obtain the minimizer shown in Table 5.1, starting from the decision rule, as will be explained in that section.

It should be noted here that all boosting methods greedily approximate the true minimizer F^* by an additive model $F_t(\mathbf{x}) = \sum_{\tau=1}^{t} \alpha_\tau h_\tau(\mathbf{x})$ estimated on a finite training set. So we replace true probabilities with estimates[2], regardless of how they are estimated:

$$F^*(\mathbf{x}) = \Psi(p(y=1|\mathbf{x}), c_{FN}, c_{FP}) \approx F_t(\mathbf{x}) = \Psi(\hat{p}(y=1|\mathbf{x}), c_{FN}, c_{FP}). \tag{5.1}$$

[1]For all methods but AsymAda, this means that the t-th round is the final boosting round, whether it reached its maximal allowed number M or not. For AsymAda, this means that $t=M$.

[2]To examine a method for cost-consistency it would have sufficed to inspect the decision rule under the optimal model F^* shown in Table 5.1, i.e. a decision rule involving true probabilities rather than estimates. We chose to use estimates for generality and notational consistency. It should be clear though that cost-consistency is ultimately only due to the loss function L rather than the specific way probabilities are estimated.

In the rest of this chapter we will use the minimizers of Table 5.1 to examine whether the decision rule implemented by each variant is optimal for the purpose of minimizing the expected misclassification cost. But first, let us remind ourselves what the optimal decision rule is.

5.2 Cost-consistency

Decision theory gives us straightforward and optimal rules for dealing with cost sensitive binary problems [30, 32, 38]. We have reviewed this in Chapter 2, the result being a simple rule: given the probability $p(y=1|\mathbf{x})$, or –in practice– a probability estimate $\hat{p}(y=1|\mathbf{x})$, we should make predictions using the rule:

$$\hat{p}(y=1|\mathbf{x}) \begin{cases} > c, & \text{predict } y=+1 \\ < c, & \text{predict } y=-1 \end{cases}, \tag{5.2}$$

where $c = \frac{c_{FP}}{c_{FP}+c_{FN}}$ is the threshold defined by the misclassification costs.

As mentioned earlier in this chapter, for generality, we shall give all the decision rules in terms of probability estimates $\hat{p}(y=1|\mathbf{x})$, as we did with Eq. (5.2). These can be estimated in more than one ways, as we will see in Sections 5.3 and 7.3 . Of course, if the decision maker (i.e. the classifier) has knowledge of the true probability $p(y=1|\mathbf{x})$, then all rules presented here can be written in terms of $p(y=1|\mathbf{x})$.

The optimal decision rule for the purposes of minimizing the expected misclassification cost under a given cost matrix, can be used to divide the cost-sensitive boosting literature into two groups, those that implement it and those that do not. We can thus define a new desirable property for a cost-sensitive boosting variant:

Definition: Cost-consistent *A method is cost-consistent, if the prediction rule it constructs is equivalent to the rule of Eq. (5.2), for any given cost matrix of the form of Table 2.2. Otherwise the method is cost-inconsistent.*

Table 5.2, shows the decision rule implemented by each method in terms of probability estimates –as in practice we do not have access to true probabilities. We can classify the methods as so[3]:

[3]Some methods in practice are used in conjunction with replacing c_{FP} & c_{FN} with hyperparameters to be set via cross-validation. This has been criticized in e.g. [101] as *heuristic*. Our decision theoretic analysis shows why they resort to this choice. Being cost-inconsistent their decision rule is not geared towards minimizing the expected cost using c_{FP} & c_{FN} directly from the cost matrix, despite them being fixed, known problem characteristics.

Cost-consistent	**Cost-inconsistent**
AdaMEC, CGAda, AsymAda,	Ada, CSB0, CSB1, CSB2, AdaC2,
AdaC1, CSAda, AdaDB	AdaC3, AdaCost, AdaCost(β_2)

5.2.1 Checking for cost-consistency

Let us now present a general strategy for showing whether a given method is cost-consistent or cost-inconsistent. As in the general scheme for checking for FGD-consistency in Chapter 4, the same steps described here can be applied to check any given algorithm for cost-consistency, despite the fact that we are going to be specifically focusing on the variants studied in the present thesis.

We shall assume that we know the loss function $L(F_t(\mathbf{x}))$ that the algorithm minimizes at stage t. Given only the weight update rule of a method, we saw how to derive $L(F_t(\mathbf{x}))$ in Chapter 4.

All methods –with the exception of AdaMEC, which will be discussed separately in the next section– once the final ensemble $F_t(\mathbf{x})$ is constructed, classify $\mathbf{x}$ according to the decision rule:

$$H(\mathbf{x}) = sign[F_t(\mathbf{x})]. \tag{5.3}$$

As explained in Eq. (5.1), once we derive the minimizer of the loss $L(F_t(\mathbf{x}))$, we can express $F_t(\mathbf{x})$ in terms of the probability estimates $\hat{p}(y=1|\mathbf{x})$ and the costs c_{FN} and c_{FP}, i.e. write the decision rule in the general form

$$H(\mathbf{x}) = sign[\Psi(\hat{p}(y=1|\mathbf{x}), c_{FN}, c_{FP})]. \tag{5.4}$$

Simplifying Eq. (5.4), we derive the decision rules given in Table 5.2. If a method implements a decision rule equivalent to that of Eq. (5.2) under any cost matrix C it is cost-consistent. Otherwise it is not.

Finally, if $L(F_t(\mathbf{x}))$ cannot be expressed as a function of $F_t(\mathbf{x}) = \sum_{\tau=1}^{t} \alpha_\tau h_\tau(\mathbf{x})$, i.e. the output of the actual ensemble constructed, then we cannot derive a decision rule as a function of $\hat{p}(y=1|\mathbf{x})$, c_{FN} & c_{FP}. This is the case for some of the methods examined here, namely CSB0, CSB1, AdaCost & AdaCost(β_2). As explained in Chapter 4, the

Table 5.2: A summary of the cost-sensitive variants showing the loss function minimized w.r.t. the ensemble constructed on round t and the final decision rule, written in terms of probability estimates and misclassification costs. In the second column, we have made use of the fact that for AsymAda the final round of boosting will be $t = M$. AdaMEC's decision rule was *defined* as shown here. AdaBoost is included for reference.

Method	Loss Function $L(F_t(\mathbf{x}))$	Decision rule $sign[\hat{p}(y=1\|\mathbf{x}) - \theta]$, where $\theta = ...$
Adaboost [42]	$e^{-yF_t(\mathbf{x})}$	$\frac{1}{2}$
AdaMEC	”	$\frac{c_{FP}}{c_{FP}+c_{FN}}$
CGAda [63, 65, 66]	$c(y)e^{-yF_t(\mathbf{x})}$	”
AsymAda [123]	$c(y)^{t/M}e^{-yF_t(\mathbf{x})}$	”
CSAda [78, 79]	$e^{-c(y)yF_t(\mathbf{x})}$	”
AdaDB [64]	”	”
AdaC1 [112, 113]	”	”
CSB2 [117]	$c(y)^{q-1}e^{-yF_t(\mathbf{x})}$ where q models have misclassified $\mathbf{x}$.	$\frac{(c_{FP})^{q-1}}{(c_{FP})^{q-1}+(c_{FN})^{q-1}}$
AdaC2 [112, 113]	$c(y)^t e^{-yF_t(\mathbf{x})}$	$\frac{(c_{FP})^t}{(c_{FP})^t+(c_{FN})^t}$
AdaC3 [112, 113]	$c(y)^{t-1}e^{-c(y)yF_t(\mathbf{x})}$	”
CSB0 [118] CSB1 [117] AdaCost(β_2) [117] AdaCost [33]	Cannot express loss as a function of $F_t(\mathbf{x})$.	Cannot express decision rule as a function of c_{FP}, c_{FN} & $\hat{p}(y=1\|\mathbf{x})$

loss function of these four variants is not a function of the additive model $F_t(\mathbf{x})$ itself, but some other function of the weak learners. As there can be no guarantee that they satisfy Eq. (5.2), the methods are classified as cost-inconsistent.

To dispel any confusion, ultimately, the argument about whether a method is cost-consistent (or not) hinges on the form of the population minimizer of its loss function, i.e. *what the method is attempting to approximate*. The loss function is either explicitly specified by the authors (e.g. CSAda, CGAda) or we can infer it by the proposed weight update rule via the FGD mechanism in Eq. (4.5). If we plug that population minimizer in the decision rule of Eq. (5.3) and the rule can be rearranged into the form of Eq. (5.2), then the method is cost consistent, otherwise it is not. Therefore, the property of cost-consistency is a direct consequence of the underlying loss function.

To provide some concrete examples, the population minimizer of the loss of AdaC2 is $F^*(\mathbf{x}) = \frac{1}{2}\log\frac{p(y=1|\mathbf{x})}{p(y=-1|\mathbf{x})} + \frac{M}{2}\log\frac{c_{FN}}{c_{FP}}$, so it is cost-inconsistent. On the other hand, in the case of CSAda it is $F^*(\mathbf{x}) = \frac{1}{c_{FP}+c_{FN}}\log\frac{p(y=1|\mathbf{x})c_{FN}}{p(y=-1|\mathbf{x})c_{FP}}$, so it is cost-consistent.

5.3 A generalization of AdaMEC

We now present an interesting observation on the original AdaMEC procedure by Ting [117]. Acknowledging that Eq. (5.2), is the optimal decision strategy for a given probability estimate, we can reformulate AdaMEC's Eq. (3.21) in a slightly different way:

$$H_{AdaMEC}(\mathbf{x}) = sign\left[c_{FN} \times \hat{p}(y=1|\mathbf{x}) - c_{FP} \times \hat{p}(y=-1|\mathbf{x})\right]. \tag{5.5}$$

where $\hat{p}(y=1|\mathbf{x}) = \frac{\sum_{\tau:h_\tau(\mathbf{x})=1}\alpha_\tau}{\sum_{\tau=1}^{t}\alpha_\tau}$. This highlights that Ting's formulation of AdaMEC makes optimal decisions, but only when estimates of probabilities are made in a very specific way – which we are not necessarily constrained to. We can therefore define a generalized form of AdaMEC, a special case of which is the version proposed by Ting [117]. All this is captured by Theorem 1.

Theorem 1: *The AdaMEC rule of Eq. (5.5) is a special case of the more general*

$$H_{AdaMEC}(\mathbf{x}) = sign\left[\hat{p}(y=1|\mathbf{x}) - c\right], \tag{5.6}$$

This generalised formulation of AdaMEC, Eq. (5.6), reduces to Eq. (3.21) when probability estimates are raw scores of the form $\hat{p}(y=1|\mathbf{x}) = \frac{\sum_{\tau:h_\tau(\mathbf{x})=1}\alpha_\tau}{\sum_{\tau=1}^{t}\alpha_\tau}$.

Proof: *In Appendix A.*

This generalization of AdaMEC, true to its name, does indeed aim to minimize the expected cost of misclassifications and is the one used throughout this thesis. By viewing AdaMEC in this form, we have *separated* the cost matrix C from the estimation of the probabilities $\hat{p}(y=1|\mathbf{x})$, whereas in Eq. (3.21) they are somewhat tangled. In this way, we can *choose* how we estimate the probabilities from the base learner outputs. One such way is to do exactly as Ting proposes and use the (normalized) weighted vote to represent each class probability, but we are not restricted to this choice. This will be discussed in more detail in Chapter 7.

5.3.1 Cost-consistency of AdaMEC and global loss

We can see from Theorem 1 that AdaMEC is by definition cost-consistent. The formulation of the algorithm discussed in this paper specifically changes the decision rule of AdaBoost into that of Eq. (5.2), regardless of how probability estimates are calculated. In other words, AdaMEC constructs the same model $[F_t(\mathbf{x})]_{AdaBoost}$ as AdaBoost, but appropriately shifts the decision threshold to account for the cost imbalance. That is, it can be seen as using a prediction rule of the form

$$H(\mathbf{x}) = sign\Big[[F_t(\mathbf{x})]_{AdaBoost} + \frac{1}{2}\log\frac{c_{FN}}{c_{FP}}\Big]. \tag{5.7}$$

An equivalent interpretation of AdaMEC is that it uses the prediction rule of Eq. (5.3) as all other variants, but the additive model $F_t(\mathbf{x})$ is replaced with

$$[F_t(\mathbf{x})]_{AdaBoost} + \frac{1}{2}\log\frac{c_{FN}}{c_{FP}} = \frac{1}{2}\log\frac{\hat{p}(y=1|\mathbf{x})}{\hat{p}(y=-1|\mathbf{x})} + \frac{1}{2}\log\frac{c_{FN}}{c_{FP}}, \tag{5.8}$$

i.e. its threshold-shifted counterpart, as Eq. (5.7) suggests.

Replacing the estimates in Eq. (5.8) with true probabilities and taking the resulting expression to be the minimizer of some '*global loss*', as we do so in Table 5.1, we find that this global loss is

$$L(F_t(\mathbf{x})) = c(y)e^{-yF_t(\mathbf{x})}, \tag{5.9}$$

i.e. the same loss as that minimized by CGAda.

5.3.2 Loss at round t versus global loss

At this point it can be useful to make a distinction regarding the two different loss functions characterizing each variant. The *loss (function) minimized at round t* is the one we have been referring to until now as the '*loss (function)*'. It is the one minimized at each round of boosting to greedily fit the additive model $F_t(\mathbf{x})$. The *global loss (function)* is the function of which the *final model* constructed by the variant (i.e. the one used inside Eq. (5.3)) is an approximate minimizer. The former is the loss function greedily minimized during training and the latter is the loss function minimized when making predictions. For most variants the two functions coincide so there is no need to make this separation. The global loss is merely $L(F_t(\mathbf{x}))$, where t is the *final boosting round*. However in the case of AdaMEC and AsymAda this distinction is useful to perform, for different reasons.

More concretely, in the case of AdaMEC the loss minimized at round t is $L(F_t(\mathbf{x})) = e^{-yF_t(\mathbf{x})}$, just as AdaBoost, since the two variants train exactly the same model. But because of the threshold shifting applied after the full ensemble is constructed, the final model constructed by AdaMEC (i.e the one used inside the prediction rule of Eq. (5.3)) can be seen as being $\frac{1}{2}\log\frac{\hat{p}(y=1|\mathbf{x})}{\hat{p}(y=-1|\mathbf{x})} + \frac{1}{2}\log\frac{c_{FN}}{c_{FP}}$, hence as the (approximate) minimizer of the global loss of Eq. (5.9). So its global loss is that of Eq. (5.9), the same (global) loss as that minimized by CGAda. Note that the reason that the loss minimized at round t and the global loss differ is that the prediction rule of AdaMEC –unlike all other variants– does not use the minimizer of the former directly for predictions in Eq. (5.3), but rather threshold-shifts it –i.e. accounts for the asymmetry post-training.

The reader might have also noticed that the form of the loss given in Table 5.2 for AsymAda does not agree with its minimizer given in Table 5.1 and the decision rule given in Table 5.2. More precisely, the exponent t/M applied to the costs $c(y)$ in the former is missing from the latter two. To understand why this is the case, we must examine the global loss. We remind the reader that AsymAda performs *exactly* M boosting rounds. Therefore, the final model is constructed at round $t = M$, when the factor t/M becomes equal to 1. In other words, since the loss minimized at round t by AsymAda is $c(y)^{t/M}e^{-yF_t(\mathbf{x})}$, its global loss –the loss minimized by the final model $F_M(\mathbf{x})$ it constructs at round $t = M$– is $c(y)e^{-yF_M(\mathbf{x})}$, i.e. the same as CGAda and AdaMEC. Note that here the global loss and the loss at round t do not actually disagree in their functional form, but the global loss *is* the loss at the *predetermined* round $t = M$, which simplifies the expression.

This discussion reveals that CGAda, AdaMEC and AsymAda are just different procedures to minimize the same cost-sensitive global loss function on their predictions. CGAda reweights/resamples the training examples in a cost-proportional way, AdaMEC shifts the decision threshold after training to account for the cost imbalance and AsymAda modifies the training algorithm itself by evenly dividing the asymmetry across all M rounds. Of course, being different procedures, their parameter estimates –hence also their probability estimates– will differ in practice. But all three aim at approximating the same minimizer $F^*(\mathbf{x}) = \frac{1}{2}\log\frac{p(y=1|\mathbf{x})}{p(y=-1|\mathbf{x})} + \frac{1}{2}\log\frac{c_{FN}}{c_{FP}}$ of the same global loss function $L(F_t(\mathbf{x})) = c(y)e^{-yF_t(\mathbf{x})}$, hence they all implement the same decision rule $H(\mathbf{x}) = sign[\hat{p}(y=1|\mathbf{x}) - c]$, albeit with different probability estimates in practice. This rule happens to be the optimal one, so all three are cost-consistent.

Thus concludes our discussion of cost-sensitive boosting variants through a decision-theoretic perspective. In the previous chapter we encountered the property of FGD-consistency and noted that despite being desirable it is not necessarily crucial for the success of a cost-sensitive boosting variant. The property we saw in this chapter, cost-consistency, on the other hand, is. A cost-consistent algorithm always implements an optimal decision rule, given its probability estimates and the cost matrix, while a cost inconsistent one does not. In the next chapter, we will examine the effect of the loss –i.e. the loss minimized at round t by each variant– on its training behaviour and generalization capacity, via the lens of margin theory. This will reveal another important property that a cost-sensitive boosting algorithm should satisfy.

Chapter 6

The margin-theoretic view

In Chapter 4 we presented a mechanism for uncovering the loss function of all methods examined in this thesis. We also divided the literature on the grounds of satisfying the property of FGD-consistency, i.e. based on whether or not the methods can be viewed as efficient greedy minimizers of that loss. In Chapter 5 we used this loss function to characterize the methods with respect to the property of cost consistency, which assesses whether the final decision rule implemented by the algorithm is optimal, given the probability estimates and the costs.

In this chapter, a closer inspection of the same loss function from the viewpoint of margin theory leads us to interesting observations regarding the training behaviour of the method that minimizes it. This will result in different margin optimization properties. And since high margin values have been linked to good generalization performance by Schapire et al. [42] and other authors [48, 125], as mentioned in Chapter 3, we can ultimately link the loss function a method minimizes to its generalization capability.

To facilitate this, we will define a new desirable property for a cost-sensitive boosting algorithm, that of *asymmetry preservation*. This property ensures that the examples of the costly class are always deemed more costly to misclassify than those of the lower-cost class that produce the same margin value during training. Failure to do so would lead to the phenomenon of *asymmetry swapping*, first observed in the case of CSAda & AdaDB by Landesa-Vázquez & Alba-Castro [65, 66], which effectively leads to training behaviour that goes contrary to the cost-sensitive nature of the task.

We will classify the boosting variants studied in this thesis with respect to the property of asymmetry preservation. Furthermore, for methods that exhibit asymmetry swapping we will give exact theoretical margin values for when the swapping occurs that can be visually verified in the figures provided.

Finally, we note that the training behaviour of asymmetry swapping methods will tend to result in margin distributions concentrated around lower values than those of asymmetry preserving methods. Leveraging the link between high margin values and good generalization performance, we then argue that asymmetry swapping methods will exhibit poorer generalization than asymmetry preserving ones. Empirical results in support of this hypothesis are also provided.

6.1 Asymmetry-preservation

We shall begin by an interesting observation regarding the loss functions. We will translate this to two different types of training behaviour, one that always puts more emphasis on examples of the important class than those of the unimportant one with the same margin value, and one that does not do so. We will then relate these two strategies to the resulting margin distribution produced during training. Finally, we will invoke the established margin theory [42, 48, 125] to connect the loss function to the generalization error.

In Figures (6.1 & 6.2) we present loss functions of two different types versus the margin $y_i F_t(\mathbf{x}_i)$. Contrary to what we see in Figure 6.1, in Figure 6.2 we notice that the lines indicating the loss on positive and negative examples cross. When this happens, the emphasis placed on the two classes is reversed: the weights of correctly classified examples of the expensive class are penalized more than those of correctly classified examples of the low-cost class, contrary to the fact that the cost matrix dictates that preserving the former is more important. This was first observed in the case of CSAda & AdaDB by Landesa-Vázquez & Alba-Castro [65, 66] and motivates the following definition:

Definition: Asymmetry-preserving *A method is asymmetry-preserving if the ratio $r_L(m)$ of the loss on an example of the important class over the loss on an example of the unimportant one – given equal $m = yF_t(\mathbf{x})$– remains greater or equal to 1 during training. Otherwise the method is asymmetry-swapping.*

The property of asymmetry preservation will again allow us to divide the methods examined into two categories as follows:

Asymmetry-preserving	**Asymmetry-swapping**
Ada, AdaMEC, CGAda, AsymAda,	AdaC1, CSAda, AdaDB,
AdaC2, CSB0, CSB1, CSB2	AdaC3, AdaCost, AdaCost(β_2)

6.1.1 Checking for asymmetry-preservation

We now present a general scheme for checking a given boosting variant for asymmetry-preservation. We will be making references to all the methods discussed in this thesis but the same reasoning can be applied to any given boosting variant, similarly to the schemes for checking for the other two properties presented so far.

Again, we shall assume that we know the loss function $L(yF_t(\mathbf{x}),c(y))$ that the algorithm minimizes at stage t. Given only the weight update rule of a method, we saw how to derive $L(yF_t(\mathbf{x}),c(y))$ in Chapter 4. Note that here we make explicit the fact that L also generally depends on the cost of each example, unlike other parts of the paper [1] where we simplified notation for clarity.

Based on the above definition, a method is asymmetry preserving if for any two examples $(\mathbf{x}_i,y_i)$ and $(\mathbf{x}_j,y_j)$ such that $c(y_i) > c(y_j)$ and $y_iF_t(\mathbf{x}_i) = y_jF_t(\mathbf{x}_j) = m$, its loss function L satisfies the following property:

$$r_L(m) = \frac{L(y_iF_t(\mathbf{x}_i),c(y_i))}{L(y_jF_t(\mathbf{x}_j),c(y_j))} = \frac{L(m,c(y_i))}{L(m,c(y_j))} \geq 1, \forall m$$

Hence, variants that minimize a loss of the form $K_1(i)e^{-y_iF_t(\mathbf{x}_i)}$, where $K_1(i)$ is a non-decreasing function of $c(y_i)$ (i.e. AdaBoost, AdaMEC, CGAda, AsymAda, AdaC2 & CSB2) are asymmetry preserving, as it is always the case that $r_L(m) = K_1(i)/K_1(j) \geq 1$, as shown in Figure 6.1.

More specifically, AdaBoost & AdaMEC ignore costs during training, thus are asymmetry-preserving under our definition, since the ratio of the loss on a positive example over the loss on a negative example which the model classifies with equal confidence $yF_t(\mathbf{x})$, is fixed to 1. AdaMEC only takes costs into account during the

[1] More specifically, in Chapter 4 only the fact that that the loss L depended on $yF_t(\mathbf{x})$ was of importance and in Chapter 4 only that it depended on $F_t(\mathbf{x})$ was.

prediction phase.

As for CGAda, AsymAda, AdaC2 & CSB2, they are asymmetry-preserving as it is always the case that the ratio of the loss on an example $(\mathbf{x}_i, y_i)$ of the expensive class over that of an example $(\mathbf{x}_j, y_j)$ of the cheap class given that $y_i F_t(\mathbf{x}_i) = y_j F_t(\mathbf{x}_j)$ will be $K(i)/K(j) > 1$.

On the other hand, variants that minimize a loss of the form $K_1(i)e^{-K_2(i)y_iF_t(\mathbf{x}_i)}$, where $K_1(i)$ and $K_2(i)$ are a non-decreasing and an increasing function of $c(y_i)$, respectively (i.e. AdaC1, AdaC3, CSAda & AdaDB), have

$$r_L(m) = \frac{K_1(i)e^{-K_2(i)y_iF_t(\mathbf{x}_i)}}{K_1(j)e^{-K_2(j)y_jF_t(\mathbf{x}_j)}} = \frac{K_1(i)}{K_1(j)}e^{m(K_2(j)-K_2(i))}.$$

In this case, it can be shown that $\exists m : r_L < 1$. Specifically when

$$m > \frac{1}{K_2(i) - K_2(j)} \log\left(\frac{K_1(i)}{K_1(j)}\right), \tag{6.1}$$

the importance of the two classes is flipped. Hence such methods are asymmetry-swapping, as can be demonstrated by Figure 6.2.

More specifically, AdaC1, CSAda & AdaDB will exhibit asymmetry swapping on any training example with margin value $m > 0$, while AdaC3 will do so on any training example with margin value

$$m > \frac{t-1}{c_{FN} - c_{FP}} \log(\frac{c_{FN}}{c_{FP}}), \tag{6.2}$$

as suggested by Eq. (6.1) and the form of their loss functions[2].

Finally, of the methods whose loss function cannot be expressed in terms of $y_i F_t(\mathbf{x}_i)$, CSB0 & CSB1 are asymmetry-preserving as their weight updates can only increase the relative importance of the important class over the unimportant one. On the other hand AdaCost & AdaCost(β_2) do not offer such a guarantee, hence are classified as asymmetry-swapping.

[2]Inspecting the loss functions showed on Table 4.1, we can see that for AdaC1, CSAda & AdaDB, we have $K_1(i) = K_1(j) = 1$, $K_2(i) = c_{FN}$ and $K_2(j) = c_{FP}$, so Eq. (6.1) will give us $m > 0$ as the logarithmic term becomes 0. Similarly, for AdaC3 we have $K_1(i) = c_{FN}^{t-1}$, $K_1(j) = c_{FP}^{t-1}$, $K_2(i) = c_{FN}$ and $K_2(j) = c_{FP}$, so Eq. (6.1) takes the form of Eq. (6.2).

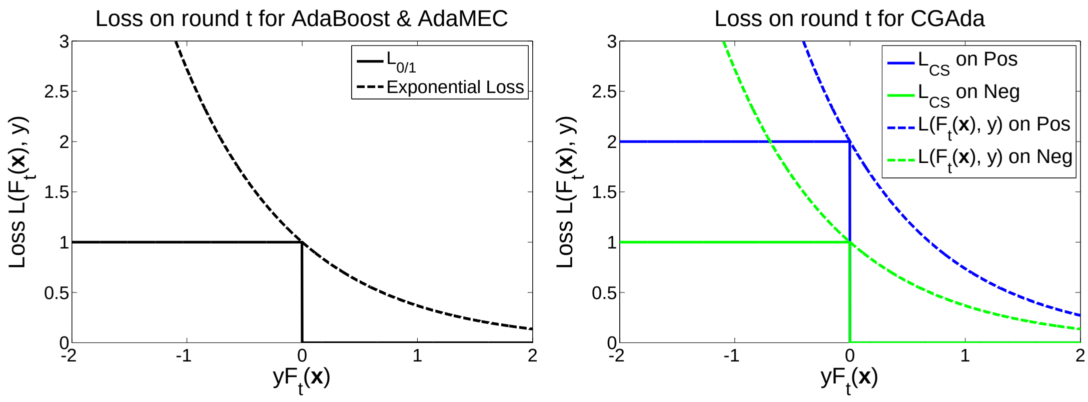

Figure 6.1: LEFT: The loss function used in AdaBoost. This illustrates the reason why AdaBoost is seen as a margin-maximising method – the loss is non-zero even when an example has been correctly classified ($y_iF_t(\mathbf{x}_i) > 0$). RIGHT: The loss for CGAda in a $2:1$ cost ratio – note that the same margin maximising properties hold, and that examples of the positive (expensive) class always have a loss greater than that of examples of the negative (cheap) class, given an equal $y_iF_t(\mathbf{x}_i)$.

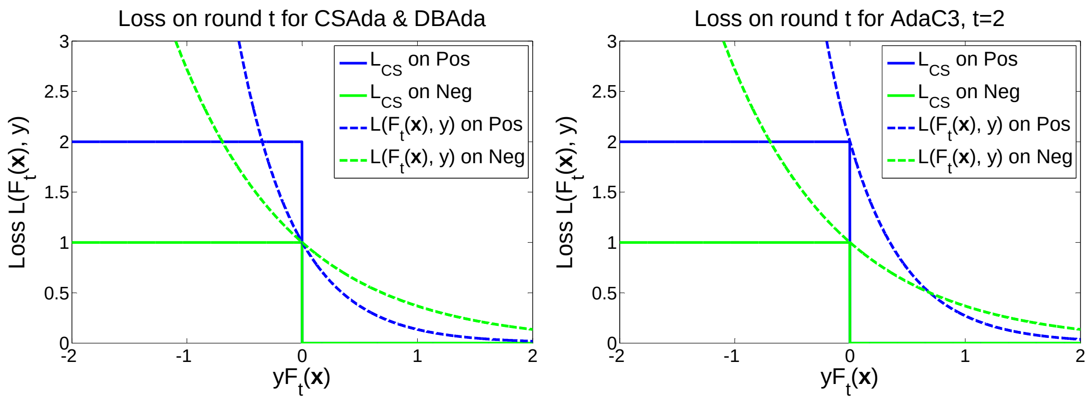

Figure 6.2: LEFT: The loss of CSAda & AdaDB does not preserve the class asymmetry leading to asymmetry swapping when $y_iF_t(\mathbf{x_i}) > 0$. Beyond that point, examples of the positive (expensive) class have a lower loss than that of examples of the negative (cheap) class, given an equal $y_iF_t(\mathbf{x}_i)$. RIGHT: The loss of AdaC3, plotted here for $t = 3$, also exhibits asymmetry swapping. More specifically it does so for margin values $y_iF_t(\mathbf{x_i}) > ((t-1)/(c_{FN} - c_{FP}))\log(\frac{c_{FN}}{c_{FP}})$, i.e. in this example, when $y_iF_t(\mathbf{x_i}) > 2\log 2$.

6.2 Asymmetry swapping and poor generalization

Following the observation of Landesa-Vázquez & Alba-Castro [65, 66] that asymmetry swapping behaviour has an adverse effect on the the generalization properties of the final ensemble constructed, we will now investigate the effect of each loss function from an empirical margin-theoretic perspective.

Given any cost-sensitive boosting variant, we shall call the minimal margin value

$$m_{swapping} = \frac{1}{K_2(i) - K_2(j)} \log\left(\frac{K_1(i)}{K_1(j)}\right),$$

for which asymmetry swapping will occur, its *asymmetry swapping point*. It is easy to see that asymmetry preserving methods have an infinite asymmetry swapping point, since for them it holds that $K_2(i) = K_2(j) = 1$.

Note that the training behaviour of asymmetry swapping methods forces them to effectively alternate between placing more emphasis on the important than the unimportant class and vice-versa, once the margin values cross the asymmetry swapping point. This is expected to lead to margin distributions concentrated close to the asymmetry swapping point, as given by Eq. (6.1). We can also expect AdaC1, CSAda & AdaDB to suffer more from this issue than AdaC3, as the latter will only swap the asymmetry of any training example with margin $y_i F_t(\mathbf{x_i}) > ((t-1)/(c_{FN} - c_{FP}))\log(\frac{c_{FN}}{c_{FP}}) > 0$ while the former three variants will do so for any example with a positive margin value, according to Eq. (6.1). In other words, AdaC3 has a higher asymmetry swapping point.

Normalizing the margin $y_i F_t(\mathbf{x_i})$ by dividing by the *1-norm* of the vector α comprised of all confidence coefficients $\alpha_1, \ldots, \alpha_t$, we get

$$m_i = \frac{y_i F_t(\mathbf{x_i})}{||\alpha||_1} \in [-1, 1]$$

The effect that the different loss functions have on the resulting normalized *margin distribution* $\{m_i | i = 1, \ldots, N\}$ is demonstrated by the results shown in Figure 6.3. The figure shows the cumulative margin distributions[3] produced by *AdaBoost*/AdaMEC, CGAda, AsymAda, CSAda & AdaC3 for four different degrees of imbalance on the

[3]In Figure 6.3 we do not distinguish between the margin distributions of positive and negative examples. Considering an equal number of positives and negatives, the margin distribution produced by cost-sensitive methods on high cost examples is on average higher than that of low cost examples, as a larger number of low cost examples than high cost ones tend to be misclassified (hence have negative margin). The effect is more pronounced as the skew ratio increases. AdaBoost/AdaMEC, being cost-insensitive in its training phase, produces margin distributions for the two classes that are -in expectation- identical.

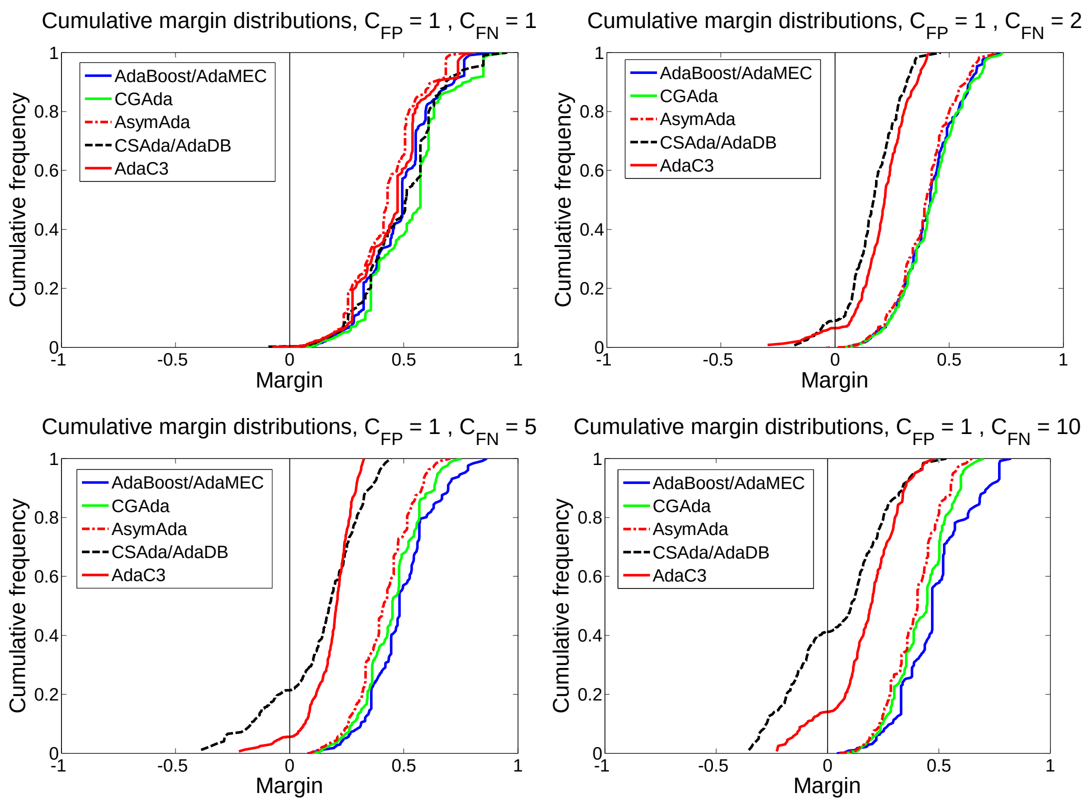

Figure 6.3: Cumulative (normalized) margin distributions for *AdaBoost*/AdaMEC, CGAda, AsymAda, CSAda & AdaC3 for degrees of imbalance $\frac{c_{FN}}{c_{FP}} = \{1, 2, 5, 10\}$. When $\frac{c_{FN}}{c_{FP}} = 1$ all methods reduce to the original AdaBoost and produce similar distributions, indicative of margin maximization. But when $\frac{c_{FN}}{c_{FP}} > 1$, CSAda and AdaC3 produce margins closer to 0. This is a result of asymmetry swapping and margin theory suggests it has a negative effect on generalization.

congress dataset.

The results demonstrate that CSAda & AdaC3 are generating lower average margins than the rest of the methods. This is attributed to the asymmetry swapping effect we analyzed earlier. It also agrees with the observation of Landesa-Vázquez & Alba-Castro that asymmetry swapping is having a detrimental effect on the generalization behavior, the latter being dependent on the margin distribution.

As explained in Chapter 3, the established margin theoretic view of boosting has related the margin distribution $\{m_i | i = 1, \ldots, N\}$ to the generalization error via several proposed bounds, for example the *minimum margin bound* [42], the *equilibrium margin bound* [125] and the *k-th margin bound* [48]. This area is still being actively researched with the bounds being refined and tightened, but it is generally accepted that all else being equal, a higher margin distribution leads to better generalization.

Summarizing the above insights, we can expect asymmetry preserving methods to exhibit better generalization than asymmetry swapping ones. We can also expect an asymmetry swapping method with a higher asymmetry swapping point to achieve better generalization than a method with a lower symmetry swapping point.

On the next chapter we will examine the cost sensitive boosting variants from a fourth and final perspective: probabilistic modelling. We will define one more property, that of producing calibrated probability estimates and show why all variants studied here fail to satisfy it. This will prompt us to correct for it by applying probability calibration to the probability estimates of the boosting ensembles, a process that we shall also describe in depth.

Chapter 7

The probabilistic view

Thus far, we have examined the cost-sensitive boosting variants from a perspective of functional gradient descent (Chapter 4), decision theory (Chapter 5) and margin theory (Chapter 6). The three perspectives have led us to define three desirable theoretical properties for a cost-sensitive boosting algorithm: *FGD-consistency*, *cost-consistency* and *asymmetry-preservation*, respectively. We saw that each property is only satisfied by a different subset of the variants examined.

This chapter will examine the variants from a *probabilistic* perspective. It will introduce a fourth and final desirable theoretical property –one that *no existing variant satisfies*, unless special action is taken: the property of having *calibrated probability estimates*. We will show that the probability estimates produced by boosting algorithms are subject to a systematic form of distortion. Simply put, the normalized outputs of the ensemble are 'poor' probability estimates and should best be treated as *raw scores*, i.e. quantifications of 'how positive' (or negative) an example is that do not necessarily satisfy the properties of probability. So when using them to make cost-sensitive decisions under the decision rules derived in Chapter 5 –even when an optimal decision rule is used– these decisions can be far from optimal as they assume that the probability estimates are 'good'.

To demonstrate this, we will first make use of the view of Adaboost as a *Product of Experts (PoE)* [31]. We will then extend this view to cover other cost-sensitive variants examined in this thesis. The probability estimates produced by boosting ensembles can all be shown to be of the form of a PoE. But a PoE is subject to systematic biases, related to the individual expert probabilities and the number of experts. Thus our first conclusion is that the probability estimates produced by boosting ensembles are subject to the same form of systematic distortion with respect to true probabilities. Fortunately, the distortion being systematic is amenable to a cure.

And here comes the concept of *calibration*, effectively the process of correcting the biases of a set of raw scores so as to transform them to proper probability estimates. We will discuss in detail the different ways of producing scores using the output of a boosting ensemble as well as the different ways of calibrating them, their advantages and disadvantages.

At the end of the chapter, before we move on to the empirical comparison of the techniques covered in the thesis, we summarize the methods studied here and the set of properties each satisfies in Table 7.1. We see that only three of the methods examined, namely CGAda, AsymAda and AdaMEC satisfy all three properties proposed in the previous three chapters.

We then propose also granting them the property of calibrated probability estimates by appropriately calibrating the probability estimates they produce. The resulting calibrated versions of CGAda, AsymAda and AdaMEC now satisfy all four theoretical properties as can be seen in Table 7.1. Pseudocode for the calibrated versions of these three variants, along with details of the specific implementation used in our experiments is also provided at the end of the chapter.

7.1 AdaBoost as a Product of Experts

We shall start with the case of the original AdaBoost algorithm. In Chapter 3 we saw that Edakunni et al. [31] viewed Adaboost as an iterative process to construct a *Product of Experts (PoE)*. The *PoE* is a probabilistic model introduced by Hinton [54], under which the probability of an outcome y is expressed as a normalized product of *–not necessarily normalized–* probabilities of y as assigned by different *experts* (i.e. base predictors).

To better understand the nature of the probability estimates generated by an AdaBoost ensemble, we now follow the inverse process of [31]. Starting from the probability estimates themselves, we show, without introducing any assumptions, that they correspond to a specific PoE model. As we will see in Section 7.2, the same reasoning allows us to derive similar models for cost sensitive boosting variants thus extending [31] to cover these variants as well.

AdaBoost builds an additive model $F_t(\mathbf{x}) = \sum_\tau \alpha_\tau h_\tau(\mathbf{x})$ to approximate

$$F^*(\mathbf{x}) \;=\; \arg\min_{F(.)} E_{\mathbf{x}y}\left\{e^{-yF(\mathbf{x})}\right\} = \frac{1}{2}\log\frac{p(y=1|\mathbf{x})}{p(y=-1|\mathbf{x})}.$$

We can get an estimate of $p(y=1|\mathbf{x})$ using the AdaBoost outputs $F_t(\mathbf{x}) \approx F^*(\mathbf{x})$,

$$\hat{p}(y=1|\mathbf{x}) = \frac{1}{1+e^{-2F_t(\mathbf{x})}}. \tag{7.1}$$

This is the form of probability estimates proposed by Friedman et al. [44] and the most popular choice in the boosting literature.
If we substitute $F_t(\mathbf{x}) = \sum_\tau \alpha_\tau h_\tau(\mathbf{x})$ into Eq. (7.1), we find that the conditional probability estimates under AdaBoost have the form of a *PoE*.

Theorem 2: *The probability estimate of Eq. (7.1), assigned to class $y=1$ by an AdaBoost ensemble F_t on an example $\mathbf{x}$ has the form of a product of experts*:

$$\hat{p}(y=1|\mathbf{x}) \quad = \quad \frac{\prod_{\tau=1}^{t} \hat{p}_\tau(y=1|\mathbf{x})}{\prod_{\tau=1}^{t} \hat{p}_\tau(y=1|\mathbf{x}) + \prod_{\tau=1}^{t} \hat{p}_\tau(y=-1|\mathbf{x})},$$

with experts of the form

$$\hat{p}_\tau(y=1|\mathbf{x}) = \begin{cases} \varepsilon_\tau \quad , & \text{if } h_\tau(\mathbf{x}) = -1 \\ 1-\varepsilon_\tau, & \text{if } h_\tau(\mathbf{x}) = 1, \end{cases}$$

$$\hat{p}_\tau(y=-1|\mathbf{x}) = \begin{cases} 1-\varepsilon_\tau, & \text{if } h_\tau(\mathbf{x}) = -1 \\ \varepsilon_\tau \quad , & \text{if } h_\tau(\mathbf{x}) = 1, \end{cases}$$

where ε_τ is the weighted error of the τ-th weak learner and $h_\tau(\mathbf{x}) \in \{-1,1\}$ its prediction on example $\mathbf{x}$.
Proof: *In Appendix A.*

Note that each expert can assign two distinct values to the probability estimate of a given example $\mathbf{x}$ being positive:

$$\hat{p}_\tau(y=1|\mathbf{x}) = \begin{cases} \varepsilon_\tau \quad , & \text{if } h_\tau(\mathbf{x}) = -1 \\ 1-\varepsilon_\tau, & \text{if } h_\tau(\mathbf{x}) = 1, \end{cases}$$

i.e. its (weighted) error rate if it predicts that $\mathbf{x}$ is actually negative and its (weighted) accuracy if it predicts that $\mathbf{x}$ is indeed positive.

The more experts added to the PoE (i.e. the more weak learners added to the AdaBoost ensemble), the more fine-grained the resulting ensemble probability estimates

$\hat{p}(y=1|\mathbf{x})$ will be. The $\hat{p}(y=1|\mathbf{x})$ of an ensemble consisting of a single expert –as we saw– has only two possible values. That of an ensemble of M experts will have up to 2^M distinct possible values. Thus a larger ensemble is able to capture an exponentially richer set of conditional class probability distributions.

In the next section we will generalize Theorem 2 to the cost-sensitive boosting variants examined here. For now, the case of AdaBoost can serve as a simple and useful tool to demonstrate the systematic distortion biases that are inherent to a product of experts model, hence also to the probability estimates of an AdaBoost ensemble.

7.1.1 A systematic distortion

Since typically ε_τ is only slightly smaller than 0.5 for weak learners (this is how a weak learner is defined, as we saw in Chapter 3), Theorem 2 suggests that the overall $\hat{p}(y=1|\mathbf{x})$ remains close to 0.5, for reasonably small numbers of experts M. If on the other hand the base learners 'are powerful enough' i.e. at least one ε_τ tends to 0, then it dominates the *PoE* and the overall probability estimate tends to 0 or 1 (0 if the corresponding $h_\tau = -1$ and 1 if it is $h_\tau = 1$). Moreover, we can expect the distortion to be more pronounced as more experts are added to the ensemble, i.e. as M increases. These are exactly the effects observed by Rosset et al. [99] and Caruana et al. [86] regarding the probability estimates produced by AdaBoost ensembles.

In Figures 7.1 & 7.2 we visually demonstrate the systematic biases described above with two simple examples. Figure 7.1 shows how a single expert can dominate the ensemble as its corresponding probability tends to 0 or 1 even when the distribution of the probabilities of the remaining experts is centred around the opposite side of the 0.5 threshold. Figure 7.2 shows the effect that the number of experts has on the overall probability. For small numbers of experts M, the PoE probability remains close to 0.5 (as most experts will be producing values close to that in the case of boosting, being weak learners), but as M increases, the PoE probability tends to either 0 or 1.

Finally, we should note that the more correlated the experts' errors are, the greatest the deviation of the PoE's predictions w.r.t. the true class probabilities will be. AdaBoost does decorrelate the errors of successive learners, but the more learners we add to the ensemble, the more likely it is that we end up with experts producing correlated errors.

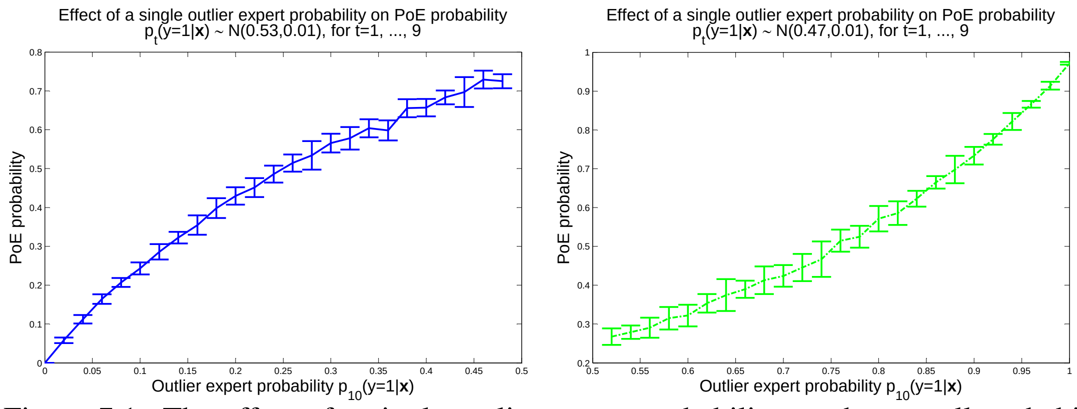

Figure 7.1: The effect of a single outlier expert probability on the overall probability of the PoE, showing how a single high-confidence expert can dominate the PoE's prediction, leading it to the opposite direction of the rest of the ensemble. LEFT: The PoE probability for a product of 10 experts. The first 9 experts have probabilities $p_t(y=1|\mathbf{x}) \sim \mathcal{N}(0.53, 0.01)$, i.e predominantly predict the positive class. The probability of the 10-th expert varies in the range $p_{10}(y=1|\mathbf{x}) \in (0, 0.49)$, i.e. predicts the negative class. As $p_{10}(y=1|\mathbf{x})$ approaches 0 it tends to rapidly dominate the PoE probability. RIGHT: Same as before, but now the first 9 experts have probabilities $p_t(y=1|\mathbf{x}) \sim \mathcal{N}(0.47, 0.01)$, i.e predominantly predict the negative class. The probability of the 10-th expert varies in $p_{10}(y=1|\mathbf{x}) \in (0.51, 1)$, i.e. predicts the positive class. As $p_{10}(y=1|\mathbf{x})$ approaches 1 it tends to rapidly dominate the PoE probability.

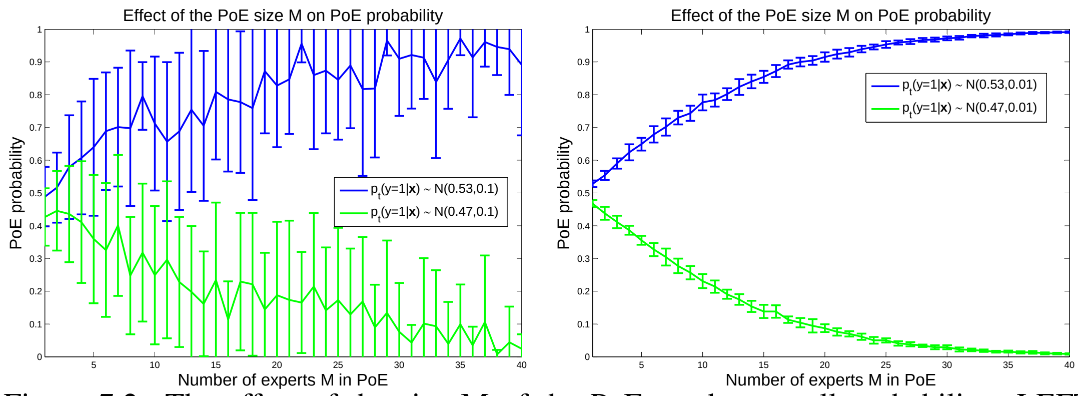

Figure 7.2: The effect of the size M of the PoE on the overall probability. LEFT: The blue curve corresponds to a PoE whose expert probabilities $p_t(y=1|\mathbf{x})$ follow a Gaussian distribution with mean 0.53 and the green curve to a PoE whose expert probabilities $p_t(y=1|\mathbf{x})$ follow a Gaussian distribution with mean 0.47. In both cases the standard deviation is 0.1. We can see that for small numbers of experts M, the PoE probabilities remain close to 0.5. But as the number of experts M increases, the PoE probability of the former PoE tends to 1 and that of the latter to 0. RIGHT: The same experiment but the standard deviation of both Gaussians is now 0.01. The two PoE probabilities converge to 1 and 0, respectively, much faster.

7.2 Cost-sensitive boosting as a Product of Experts

Following the same reasoning behind Theorem 2, we can construct *PoE* models for the probability estimates of other boosting variants, thus showing that they are also subject to the same form of distortion.

7.2.1 A simple example

A straightforward example of this is the case of AdaMEC. As the model constructed by AdaMEC is just a threshold-shifted version of the one built by AdaBoost, i.e. $F_t(\mathbf{x}) = [F_t(\mathbf{x})]_{Ada} + \frac{1}{2}\log\frac{c_{FN}}{c_{FP}}$, substituting $F_t(\mathbf{x})$ into Eq. (7.1), we get that the resulting probability estimates have the form

$$\hat{p}_w(y=1|\mathbf{x}) \quad = \quad \frac{c_{FN}\cdot\prod_{\tau=1}^{t}\hat{p}_\tau(y=1|\mathbf{x})}{c_{FN}\cdot\prod_{\tau=1}^{t}\hat{p}_\tau(y=1|\mathbf{x}) + c_{FP}\cdot\prod_{\tau=1}^{t}\hat{p}_\tau(y=-1|\mathbf{x})}, \qquad (7.2)$$

with expert probabilities as given in Theorem 2.

Thus, an alternative view of AdaMEC is that it classifies examples as $H(\mathbf{x}) = sign[\hat{p}_w(y=1|\mathbf{x}) - 0.5]$, as AdaBoost does, but changes the underlying probabilistic model of the conditional probabilities for the positive class to that of Eq. (7.2)[1]. This new model is a weighted version of the *PoE* of Theorem 2, where the probability of each class is reinforced by a multiplicative factor equal to its relative importance. We can think of this as adding an additional expert to the *PoE* that captures our *prior knowledge* over the asymmetry. These factors c_{FP} and c_{FN} can of course be appropriately renormalized to $\frac{c_{FP}}{c_{FN}+c_{FP}}$ and $\frac{c_{FN}}{c_{FN}+c_{FP}}$, respectively, which can also be interpreted as prior probabilities for the positive and negative class, respectively. This duality between cost and class imbalance was discussed extensively in Chapter 2. This model is again a PoE, so subject to the same form of systematic distortions we explored in the previous subsection.

7.2.2 Generalizing to other boosting variants

We can generalize these observations to most other boosting variants examined here. In Chapter 5, we saw that for most methods, the minimizer $F^*(\mathbf{x})$ of the loss $L(F(\mathbf{x}))$

[1]In Chapter 5, we defined AdaMEC as the method that assigns $\mathbf{x}$ according to the decision rule $H(\mathbf{x}) = sign[\hat{p}(y=1|\mathbf{x}) - c]$. In that case $\hat{p}(y=1|\mathbf{x})$ was a *non-prior-weighted* probability estimate, and more specifically, the same as that produced by AdaBoost (Theorem 2). Here $\hat{p}_w(y=1|\mathbf{x})$ is a *prior-weighted* probability estimate. These are two different approaches to handling imbalance: either move the decision threshold, or reweigh/resample the examples. More on this subject in Subsection 7.2.3.

can be written as some function Ψ of the conditional probability $p(y=1|\mathbf{x})$ and the misclassification costs c_{FN} and c_{FP}, i.e. $F^*(\mathbf{x}) = \Psi(p(y=1|\mathbf{x}), c_{FN}, c_{FP})$.

More specifically, inspecting Table 5.1, we can see that Ψ is of the general form

$$F^*(\mathbf{x}) = K_p \log \frac{p(y=1|\mathbf{x})}{1-p(y=1|\mathbf{x})} + K_c \log \frac{c_{FN}}{c_{FP}}, \tag{7.3}$$

where K_p and K_c are constants.

Eq. (7.3) relates the optimal model $F^*(\mathbf{x})$ to the true probabilities $p(y=1|\mathbf{x})$. If we solve this expression w.r.t. $p(y=1|\mathbf{x})$, we get that the true probability is a function of the optimal model $F^*(\mathbf{x})$,

$$p(y=1|\mathbf{x}) = \frac{1}{1+\left(\frac{c_{FN}}{c_{FP}}\right)^{K_c/K_p} e^{-F^*(\mathbf{x})/K_p}}.$$

Replacing the optimal model $F^*(\mathbf{x})$ with the additive model $F_t(\mathbf{x}) = \sum_\tau \alpha_\tau h_\tau(\mathbf{x})$, constructed by the boosting variant to approximate it, we obtain an approximation of the true probability, i.e. the estimate $\hat{p}(y=1|\mathbf{x})$ as

$$\hat{p}(y=1|\mathbf{x}) = \frac{1}{1+\left(\frac{c_{FN}}{c_{FP}}\right)^{K_c/K_p} e^{-F_t(\mathbf{x})/K_p}}. \tag{7.4}$$

It can now be shown that Theorem 2 is but a special case of the more general Theorem 3 given below.

Theorem 3: *Let there be a boosting variant approximating*

$$F^*(\mathbf{x}) = K_p \log \frac{p(y=1|\mathbf{x})}{1-p(y=1|\mathbf{x})} + K_c \log \frac{c_{FN}}{c_{FP}},$$

where K_p and K_c are constants, with an additive model $F_t(\mathbf{x}) = \sum_\tau \alpha_\tau h_\tau(\mathbf{x})$. The probability estimate assigned to class $y=1$ by the ensemble F_t, on an example **x** *has the form of a product of experts*:

$$\hat{p}_w(y=1|\mathbf{x}) \quad = \quad \frac{\hat{p}_0(y=1)\prod_{\tau=1}^t \hat{p}_\tau(y=1|\mathbf{x})}{\hat{p}_0(y=1)\prod_{\tau=1}^t \hat{p}_\tau(y=1|\mathbf{x}) + \hat{p}_0(y=-1)\prod_{\tau=1}^t \hat{p}_\tau(y=-1|\mathbf{x})},$$

with experts of the form

$$\hat{p}_0(y=1) = \frac{1}{1+\left(\frac{c_{FN}}{c_{FP}}\right)^{K_c/K_p}}$$

$$\hat{p}_0(y=-1) = \frac{\left(\frac{c_{FN}}{c_{FP}}\right)^{K_c/K_p}}{1+\left(\frac{c_{FN}}{c_{FP}}\right)^{K_c/K_p}}$$

$$\hat{p}_\tau(y=1|\mathbf{x}) = \frac{(1-\varepsilon_\tau)^{h_\tau(\mathbf{x})/2K_p}}{(1-\varepsilon_\tau)^{h_\tau(\mathbf{x})/2K_p}+{\varepsilon_\tau}^{h_\tau(\mathbf{x})/2K_p}}$$

$$\hat{p}_\tau(y=-1|\mathbf{x}) = \frac{{\varepsilon_\tau}^{h_\tau(\mathbf{x})/2K_p}}{(1-\varepsilon_\tau)^{h_\tau(\mathbf{x})/2K_p}+{\varepsilon_\tau}^{h_\tau(\mathbf{x})/2K_p}}$$

where ε_τ is the weighted error of the τ-th weak learner, whose definition is given in the description of the algorithm, $\alpha_\tau = \frac{1}{2}\log\frac{1-\varepsilon_\tau}{\varepsilon_\tau}$ the corresponding confidence coefficient and $h_\tau(\mathbf{x}) \in \{-1,1\}$ its prediction on example $\mathbf{x}$.
Proof: *In Appendix A.*

For example, for AdaBoost $K_p = 1/2$ and $K_c = 0$, so Theorem 3 reduces to the special case of Theorem 2. For AdaMEC, CGAda and AsymAda $K_p = 1/2$ and $K_c = 1/2$. So the PoE takes the form of Eq. (7.2), but the probability estimates produced by the individual experts differ as the definition of ε_τ is different for each variant.

7.2.3 Prior-weighted vs. non-prior-weighted PoE

Focusing on the cases of AdaMEC, CGAda and AsymAda, a clarification is in order. If a *prior-weighted* probability estimate of the form of Eq. (7.2) is used for predictions, an example $\mathbf{x}$ is classified according to the prediction rule $H(\mathbf{x}) = sign[\hat{p}_w(y=1|\mathbf{x})-0.5]$. This is one way to take costs into account, but as we described in Chapter 2, not the only one. The prior-weighting can be replaced with threshold-shifting. Notice that

$$\hat{p}_w(y=1|\mathbf{x}) = \frac{(1-\hat{p}_0(y=1))\hat{p}(y=1|\mathbf{x})}{(1-\hat{p}_0(y=1)\hat{p}(y=1|\mathbf{x}))+\hat{p}_0(y=1)(1-\hat{p}(y=1|\mathbf{x}))}, \tag{7.5}$$

is the product of experts *excluding the prior term* (i.e. without factoring-in the costs). It can be now be shown that $\hat{p}(y=1|\mathbf{x})-c>0 \iff \hat{p}_w(y=1|\mathbf{x})-0.5>0$. Therefore, the prediction rule $H(\mathbf{x}) = sign[\hat{p}_w(y=1|\mathbf{x})-0.5]$ is equivalent to the optimal decision rule $H(\mathbf{x}) = sign[\hat{p}(y=1|\mathbf{x})-c]$ of Eq. (5.2), i.e. the methods are cost-consistent as discussed in Chapter 5.

Whether the prior-weighted $\hat{p}_w(y=1|\mathbf{x})$ or non-prior-weighted $\hat{p}(y=1|\mathbf{x})$ probability estimate is used is of minor importance in our discussion. It will only affect the threshold of the decision rule as described above. Most approaches (including CGAda & AsymAda) inherently compute $\hat{p}_w(y=1|\mathbf{x})$ and do not apply threshold shifting, while AdaMEC inherently computes $\hat{p}(y=1|\mathbf{x})$ and then applies threshold shifting. The important thing at this point is that in either case, the underlying probabilistic model of a boosting algorithm satisfying the condition of Theorem 3 is of the form of a PoE and thus distorted in a predictable, systematic way. The next step will be to correct for that distortion and this will be the focus of Section 7.3.

7.2.4 Variants not covered by Theorem 3

Before we move on to discussing the calibration of probability estimates, let us make a final side note of the algorithms that do not fit into Theorem 3. These are the usual suspects: CSB0, CSB1, AdaCost and AdaCost(β_2). We saw multiple times that these four variants needed separate treatment under our framework. This is due to their heuristic nature criticised in e.g. [65, 66, 101]. The main issue is that their weight updates are not a function of the constructed additive model $F_t(\mathbf{x})$ (let alone the margin $yF_t(\mathbf{x})$), but of some other function of its weak learner components. As a result, if we were to use the form of the weight updates to derive the underlying loss function[2], this too will not be a function of the constructed additive model $F_t(\mathbf{x})$. Thus it is difficult to come up with a satisfactory probability estimate for the model constructed under these variants. We could use e.g. Eq. (7.6) –which will describe shortly– as an estimate, but as we will see, this is simply a *raw score*, subject to systematic biases and not a *calibrated* probability estimate.

7.3 Calibration

7.3.1 Calibrated probability estimates versus raw scores

Probability estimates are not always straightforward to obtain from the outputs of a classifier. The majority of classifiers allow for their output to be treated as a *score* for each test example $\mathbf{x}$ that indicates '*how positive*' $\mathbf{x}$ is. One choice for the score of an

[2]We remind the reader that these variants were proposed before 1999, i.e. before the view of boosting algorithms as greedy stagewise minimizers of a well defined loss function was established by Friedman et al. [44, 45] and Mason et al. [80].

AdaBoost ensemble on a given instance $\mathbf{x}$ is

$$s(\mathbf{x}) = \frac{\sum_{\tau:h_\tau(\mathbf{x})=1}\alpha_\tau}{\sum_{\tau=1}^{t}\alpha_\tau} \in [0,1], \tag{7.6}$$

the weighed fraction of base learners voting for the positive class. Another is

$$s'(\mathbf{x}) = \frac{1}{1+e^{-2F_t(\mathbf{x})}} \in [0,1] \tag{7.7}$$

which is the quantity we have been denoting with $\hat{p}(y=1|\mathbf{x})$ and using as the estimate of the probability[3] of $\mathbf{x}$ belonging to the positive class throughout the previous sections, following the framework of Friedman et al. [44] and a large body of the boosting literature.

We already gave a generalization of scores of the form $s'(\mathbf{x})$ for a more general boosting algorithm in Eq. (7.4). Scores of the form $s(\mathbf{x})$ are generalized directly by Eq. (7.6), bearing in mind that these are *prior weighted* scores in all cases but that of AdaMEC which –like AdaBoost– trains the model in a cost-insensitive way (α_τ does not factor-in the costs), so they are to be compared to a threshold of 0.5.

The act of converting *raw scores* to actual probability estimates is called *calibration*. Denoting with N the total number of examples, N_s the number of examples with score s and $N_{+,s}$ the number of positives with score s, Zadrozny & Elkan [133] give the following definition:

Definition: Calibrated classifier *A classifier is said to be calibrated if the empirical probability of an example with score $s(\mathbf{x}) = s$ belonging to the positive class, $N_{+,s}/N_s$, tends to the score value s, as $N \to \infty$, $\forall s$.*

A *raw score*, be it of the form $s(\mathbf{x})$ or $s'(\mathbf{x})$, is not sufficient for making a cost-sensitive decision regarding the instance $\mathbf{x}$. What we need is a calibrated estimate of $p(y=1|\mathbf{x})$, given which, classifying $\mathbf{x}$ according to the decision rule of Eq. (5.2) would give us a *Bayes-optimal* decision.

We saw why scores of the form of $s'(\mathbf{x})$ are not calibrated: namely due to their PoE form. To see why scores of the form $s(\mathbf{x})$ are not calibrated, let us remind ourselves that boosting methods maximize the margin. In Chapter 6, we gave the normalized version

[3]The terms '*score*' and '*probability estimate*' are often conflated. A score needs not be normalized within the range $[0,1]$, but even if it is, it is not necessarily a *good* probability estimate –in terms of satisfying the measure-theoretic properties of probability– even though it might often be used as such in the literature. As we will see, we can convert *raw scores* to actual probability estimates by properly *calibrating* them.

of the margin of an example $(\mathbf{x}, y)$ under an additive model $F_t(\mathbf{x}) = \sum_{\tau=1}^{t} \alpha_\tau h_\tau(\mathbf{x})$ constructed by boosting as $m = yF_t(\mathbf{x})/\sum_{\tau=1}^{t} \alpha_\tau \in [-1, 1]$. Using this, we can re-express Eq. (7.6) as

$$s(\mathbf{x}) = \frac{1}{2}\big(1 + \frac{m}{y_i}\big) = \begin{cases} \frac{1}{2}(1+m) & , \quad \text{if } y = 1 \\ \frac{1}{2}(1-m), & \quad \text{if } y = -1. \end{cases} \tag{7.8}$$

So the score of the form $s(\mathbf{x})$ is just a rescaled version of the margin m. As boosting aims to produce large margins m (at least on the training set), it tends to produce 'over-confident' scores (i.e. close to 0 for negative examples and close to 1 for positives), which require calibration. In that sense, maximum margin classification appears to be at conflict with producing calibrated probability estimates.

7.3.2 Calibrating boosting

Niculescu-Mizil & Caruana [86] showed empirically that the scores produced by AdaBoost exhibit a '*sigmoid distortion*'[4] – which agrees with our aforementioned observations. The authors also showed that once properly calibrated, AdaBoost produced superior probability estimates to any other model included in their study.

Niculescu-Mizil & Caruana produced probability estimates using three different approaches. The first approach, which they dubbed *logistic correction*, was to use scores of the form $s'(\mathbf{x})$ as implied by the framework of Friedman et al. [44]. The second method used scores of the form $s(\mathbf{x})$ and calibrated them via *Platt scaling* [92], a method originally used to map *SVM* outputs to conditional class probabilities. Platt scaling consists of finding the parameters A and B for a sigmoid mapping of scores to probability estimates, $\hat{p}(y = 1|\mathbf{x}) = \frac{1}{1+e^{As(\mathbf{x})+B}}$, such that the likelihood of the data is maximized. Fitting A and B requires the use of a separate validation set. The third method, was to use again scores of the form $s(\mathbf{x})$ and calibrate them via *isotonic regression* [98]. The latter is non-parametric and more general as it can be used to calibrate scores which exhibit any form of monotonic distortion [132].

Among the three methods, Platt scaling produced the best probability estimates on small sample sizes, closely followed by isotonic regression [86]. Due to its higher complexity, isotonic regression is more prone to overfitting, but for the same reason it can be a better choice than Platt scaling when sufficient data are available, as it is not restricted to capturing only sigmoid score distortions, but non-decreasing distortions of any kind.

[4]The scores produced by AdaBoost are a sigmoid transformation of actual probability estimates.

7.4 Satisfying all desirable properties

Before we close this chapter, note that of all the methods proposed, as we can see in Table 7.1, only three satisfy all three properties of FGD-consistency, cost-consistency and asymmetry-preservation: CGAda, AsymAda & AdaMEC. Interestingly, each of these is drawn from one of the three main approaches for making a learning algorithm cost-sensitive: cost-proportional resampling/reweighting of the dataset, modifying the training algorithm to take costs into account, and shifting the decision threshold to account for the cost imbalance, respectively.

FGD-consistency ensures that the steps of the algorithm are coherent and geared towards greedily minimizing a monotonically decreasing loss function of the margin. Asymmetry-preservation grants them good generalization by connecting said loss to margin maximization in a cost-sensitive setting. Cost-consistency ensures that the probability estimates are used in a way that is consistent with the goal of minimizing the expected cost of future classifications. What remains is a reasonable guarantee that said probability estimates are good approximations of the true underlying probabilities. As we saw this can be achieved by calibration.

In this work we generate scores of the form $s(\mathbf{x})$ under AdaMEC, CGAda & AsymAda[5]. We then apply Platt scaling to calibrate them. We account for class imbalance in the calibration set by the following correction detailed in [92]: if the calibration set has N_+ positive examples and N_- negatives, Platt calibration uses values $\frac{N_+ +1}{N_+ +2}$ and $\frac{1}{N_- +2}$, rather than 1 and 0, for the *target probability estimates* of positive and negative examples, respectively. Pseudocode for this full process is provided in Algorithm 2 for AdaMEC (cost-insensitive training & threshold-shifting) and in Algorithm 3 for CGAda, AsymAda and all other variants (cost-sensitive training & no threshold-shifting).

With these final changes, the Platt-calibrated versions of the AdaMEC, CGAda & AsymAda algorithms now satisfy all the properties shown on Table 7.1 – we proceed to our experimental results chapter, where we compare these methods to all other variants discussed. As a closing remark, we should note that calibrating using isotonic regression instead could improve performance, especially on larger datasets. However, our experimental results show that the simpler Platt scaling is enough to showcase the superior performance of calibrated AdaMEC, CGAda & AsymAda over the existing variants and their calibrated counterparts.

[5]Calibrated versions of all other variants were also included in the study, for completeness.

Table 7.1: Properties of cost-sensitive boosting algorithms and their calibrated counterparts. AdaBoost added for completeness.

Method	FGD-consistent	Cost-consistent	Asymmetry-preserving	Calibrated estimates
Ada [42]	✓		✓	
AdaCost [33]				
AdaCost(β_2) [117]				
CSB0 [118]			✓	
CSB1 [117]			✓	
CSB2 [117]			✓	
AdaC1 [112, 113]		✓		
AdaC2 [112, 113]	✓		✓	
AdaC3 [112, 113]				
CSAda [78, 79]	✓	✓		
AdaDB [64]	✓	✓		
AdaMEC [87, 117]	✓	✓	✓	
CGAda [63, 65, 66]	✓	✓	✓	
AsymAda [123]	✓	✓	✓	
Calibrated Ada	✓		✓	✓
Calibrated AdaCost				✓
Calibrated AdaCost(β_2)				✓
Calibrated CSB0			✓	✓
Calibrated CSB1			✓	✓
Calibrated CSB2			✓	✓
Calibrated AdaC1		✓		✓
Calibrated AdaC2	✓		✓	✓
Calibrated AdaC3				✓
Calibrated CSAda	✓	✓		✓
Calibrated AdaDB	✓	✓		✓
Calibrated AdaMEC	✓	✓	✓	✓
Calibrated CGAda	✓	✓	✓	✓
Calibrated AsymAda	✓	✓	✓	✓

Algorithm 2 Platt-Calibrated AdaMEC

```
**Input:** Number of weak learners $M$, data $\{(\mathbf{x}_i, y_i)|i = 1, \dots, N\}$, where $y_i \in \{-1, 1\}$,
cost of false negatives $c_{FN}$, cost of false positives $c_{FP}$

**Training Phase:**
  1. Split data into training $D_{tr}$ & calibration set $D_{cal}$
  2. On $D_{tr}$:
    2.1. Train AdaBoost ensemble $F(\mathbf{x}) = \sum_{t=1}^{M} \alpha_t h_t(\mathbf{x})$
  3. On $D_{cal}$:
    3.1. Calculate scores $s(\mathbf{x_i}) = \frac{\sum_{\tau:h_\tau(\mathbf{x_i})=1} \alpha_t}{\sum_{\tau=1}^{t} \alpha_t} \in [0, 1], \forall \mathbf{x}_i \in D_{cal}$
    3.2. Calculate the number of positives $N_+$ and negatives $N_-$ in $D_{cal}$
    3.3. Find $A, B$ s. t. $\sum_{i \in D_{cal}} (\hat{p}(y = 1|\mathbf{x_i}) - y_i')^2$ is minimized,
         where $\hat{p}(y = 1|\mathbf{x}) = \frac{1}{1+e^{As(\mathbf{x})+B}}$ and $y_i' = \begin{cases} \frac{N_+ + 1}{N_+ + 2}, & \text{if } y_i = 1 \\ \frac{1}{N_- + 2}, & \text{if } y_i = -1 \end{cases}$

**Prediction Phase:**
  4. On new example $\mathbf{x}$:
    4.1. Calculate *non-prior-weighted* score $s(\mathbf{x}) = \frac{\sum_{\tau:h_\tau(\mathbf{x})=1} \alpha_t}{\sum_{\tau=1}^{t} \alpha_t} \in [0, 1]$
    4.2. Obtain *non-prior-weighted* probability estimate $\hat{p}(y = 1|\mathbf{x}) = \frac{1}{1+e^{As(\mathbf{x})+B}}$
    4.3. Predict class $H(\mathbf{x}) = sign\left[\hat{p}(y = 1|\mathbf{x}) > \frac{c_{FP}}{c_{FP}+c_{FN}}\right]$
```

Implementation details: In the experiments of Chapter 8, a 50% / 50% split was chosen for Step 1. Step 3.3 was performed using the matlab command *nlinfit* with a tolerance of 10^{-10} and a maximum number of 600 iterations.

Algorithm 3 Platt-Calibrated AsymAda / Platt-Calibrated CGAda

Input: Number of weak learners M, data $\{(\mathbf{x}_i, y_i)|i = 1, \ldots, N\}$, where $y_i \in \{-1, 1\}$, cost of false negatives c_{FN}, cost of false positives c_{FP}

```
Training Phase:
  1. Split data into training $D_{tr}$ & calibration set $D_{cal}$
  2. On $D_{tr}$:
     2.1. Train AsymAda/CGAda ensemble $F(\mathbf{x}) = \sum_{t=1}^{M} \alpha_t h_t(\mathbf{x})$
  3. On $D_{cal}$:
     3.1. Calculate scores $s(\mathbf{x_i}) = \frac{\sum_{\tau: h_\tau(\mathbf{x_i})=1} \alpha_t}{\sum_{\tau=1}^{t} \alpha_t} \in [0,1], \forall \mathbf{x}_i \in D_{cal}$
     3.2. Calculate the number of positives $N_+$ and negatives $N_-$ in $D_{cal}$
     3.3. Find $A, B$ s. t. $\sum_{i \in D_{cal}} (\hat{p}(y=1|\mathbf{x_i}) - y_i')^2$ is minimized,
          where $\hat{p}(y=1|\mathbf{x}) = \frac{1}{1+e^{As(\mathbf{x})+B}}$ and $y_i' = \begin{cases} \frac{N_+ + 1}{N_+ + 2}, & \text{if } y_i = 1 \\ \frac{1}{N_- + 2}, & \text{if } y_i = -1 \end{cases}$
Prediction Phase:
  4. On new example $\mathbf{x}$:
     4.1. Calculate prior-weighted score $s(\mathbf{x}) = \frac{\sum_{\tau: h_\tau(\mathbf{x})=1} \alpha_t}{\sum_{\tau=1}^{t} \alpha_t} \in [0,1]$
     4.2. Obtain prior-weighted probability estimate $\hat{p}_w(y=1|\mathbf{x}) = \frac{1}{1+e^{As(\mathbf{x})+B}}$
     4.3. Predict class $H(\mathbf{x}) = sign\left[\hat{p}_w(y=1|\mathbf{x}) > \frac{1}{2}\right]$
```

Implementation details: In the experiments of Chapter 8, a 50% / 50% split was chosen for Step 1. Step 3.3 was performed using the matlab command *nlinfit* with a tolerance of 10^{-10} and a maximum number of 600 iterations.

Chapter 8

Empirical comparison

8.1 Experimental setup

In our empirical analysis we compare the performance of all methods[1] we discussed to those of calibrated AdaMEC, AsymAda & CGAda under various degrees of cost imbalance. Calibrated versions of all other variants are also included for completeness. Univariate logistic regression models, trained with conjugate gradient descent, were chosen as base learners. Their maximum number M was set to 100.

We used 18 datasets from the *UCI repository*. Multiclass problems were handled with a *one-vs-rest* approach: one class was deemed as positive and all others formed the negative one. Our goal is to compare the methods under 21 different cost setups, namely $\frac{c_{FN}}{c_{FP}} \in \{\frac{100}{1}, \frac{50}{1}, \frac{25}{1}, \frac{20}{1}, \frac{15}{1}, \frac{10}{1}, \frac{5}{1}, \frac{2.5}{1}, \frac{2}{1}, \frac{1.5}{1}, \frac{1}{1}, \frac{1}{1.5}, \frac{1}{2}, \frac{1}{2.5}, \frac{1}{5}, \frac{1}{10}, \frac{1}{15}, \frac{1}{20}, \frac{1}{25}, \frac{1}{50}, \frac{1}{100}\}$. We selected an equal number of positive and negative examples, to suppress the additional effects of class imbalance, the same approach followed by Landesa-Vázquez and Alba-Castro [66]. This was achieved by uniformly undersampling the majority class rather than by oversampling the minority class. Thus we avoid overfitting due to duplicates in training/testing/validation sets. A summary of the datasets used can be found in Appendix B.

We use a random 25% of the data for testing. The remaining 75% is used for training[2]. To perform calibration using Platt scaling, we needed to also reserve a separate

[1]As CSAda & AdaDB are equivalent within numerical precision [64], we only present results for CSAda. The α_t values were calculated using *Newton steps* and a tolerance of 10^{-6}.

[2]This is sufficient for comparing the methods on a common ground and is common practice in the literature. In practice, a 75% / 25% train/test split is usually unrealistic. In most real world situations it is not the case that we have access to the majority of a population during the training phase. This footnote serves as a reminder to the reader that practical deployment of machine learning results often involves additional considerations to be made.

validation set to fit the parameters of the sigmoid without overfitting. A third of the training data was used to this end. For uncalibrated methods, the entire training set was used to fit the models. After training the models (and calibrating on the validation set, where applicable), we evaluated them on the test set. The entire procedure is repeated 30 times. For each method, we report average values and 95% confidence intervals.

To assess the performance of the different approaches, we use the *Brier curves* [53] (loss $Q(z)$ vs. z), for a given *operating condition* (i.e. skew in deployment phase) $z_{dep} = z$, computed over the test set. We remind the reader that the skew [28, 53] is a combined measure of cost and class imbalance, defined as

$$z = \frac{\pi_- \cdot c_{FP}}{\pi_- \cdot c_{FP} + \pi_+ \cdot c_{FN}} \in [0,1].$$

A skew $z < 0.5$ signifies that negative examples are more important than positives, values $z > 0.5$ that positive examples are more important and $z = 0.5$ corresponds to the symmetric case of all examples being equally important, as discussed in Chapter 2.

We also presented the properties of Brier Curves in detail in Chapter 2. To recap the most important ones, the *area below the Brier curve* is equal[3] to the *Brier Score* (BS). The lower the BS is, the better the estimates of the model. The BS has been shown [84] to be decomposable into two components,

$$BS = calibration\ loss + refinement\ loss.$$

The *calibration loss* measures how close the probability estimates are to the true probabilities given the class label and the *refinement loss* is the loss due to the same scores being assigned to instances from different classes [62]. Thus the difference of the area under the Brier curve of AdaMEC, AsymAda & CGAda and that of their calibrated counterparts is due to the reduction of the calibration loss component of the BS. The $Q(z) = z$ diagonal of a Brier curve corresponds to the '*all-negatives*' classifier, while the $Q(z) = 1 - z$ diagonal to the '*all-positives*'.

[3]In our experiments we approximate the BS by generating the Brier curve with non-uniform skew samples – the 21 values of $\frac{c_{FN}}{c_{FP}}$ examined.

8.2 Analysis of experimental results

8.2.1 Overall trend

In total we examined $15 \times 18 \times 21$ combinations[4] of $(method, dataset, skew\ ratio)$. For clarity, we shall not present all results in the form of Brier curves. As an overall measure of the performance of each method across all degrees of imbalance, we calculate the average *area under the Brier curve* it attains per dataset, which equals its expected BS. The results are shown in Table 8.1. It should be noted for reference that the model that assigns all examples to the expensive class has expected $BS = 0.25$ (the area below the envelope defined by the two diagonals). In Figure 8.1 we rank all methods according to their average area under the Brier curve, across all datasets –higher rank meaning lower average area under the curve, i.e. better performance.

Overall, in Figure 8.1 we see that calibrated versions of AdaMEC, CGAda & AsymAda outrank their uncalibrated counterparts. This can solely be attributed to the decrease of the *calibration loss* component of the *BS*. These results agree with our theoretical observations about the probability estimates of boosting variants being distorted and requiring to be calibrated. Moreover, the uncalibrated versions of AdaMEC, CGAda & AsymAda dominate the remaining variants. This is again in accordance with our theoretical findings as these three methods are the only ones that satisfy all three properties of FGD-consistency, cost-consistency and asymmetry-preservation.

After AdaMEC, CGAda & AsymAda, the best performing variants are CSB2 & AdaC2. AdaC1 exhibits an erratic behaviour; on about half of the datasets examined it ranks among the top-performing methods in terms of average *BS*, while in the rest of them its performance is among the poorest. AdaCost, AdaCost(β_2), CSAda and on some datasets AdaC1, were found to yield the highest *BS*. What these methods have in common is the asymmetry-swapping effect[5]. We also compared the statistical significance of the difference in the average ranking of their Brier scores (low rank indicating better performance) across all datasets. The resulting *critical difference*

[4]The combinations were actually $26 \times 18 \times 21$ after the additional experiments including the original AdaBoost and calibrated versions of all variants presented at the end of the Chapter.

[5]AdaCost & AdaC1 have been criticised in recent studies as 'unstable, repeatedly producing meaningless negative, or even imaginary, α_t values' [78, page 8]. AdaCost was found in [66] to exhibit a similar 'worse-than-baseline' behaviour. The authors also attribute it to the α_t coefficients not being guaranteed to be positive reals. In our experiments we prevented this situation from occurring by forcing the ensemble to terminate training if it cannot find a learner with a positive real α_t. Rather than adding 'meaningless' learners, our strategy led to ensembles consisting of few experts, as attested by Figures (8.2 & 8.9), a potential reason for the poor performance of AdaCost, AdaCost(β_2) and –on some datasets– AdaC1.

Table 8.1: Average *area under the Brier curves* produced by each of the 15 methods examined for all 18 datasets. The area is equal to the average *Brier score*, so lower values are desirable. The lowest value per dataset is marked in bold. AsymAda, AdaMEC & CGAda outperform the other approaches and are in turn outperformed by their calibrated versions. The best performing method overall is *calibrated AsymAda*.

Dataset	CSB0	CSB1	CSB2	AdaC1	AdaC2	AdaC3	AdaCost	AdaCost(β_2)
survival	0.2335	0.2537	0.2326	0.3277	0.2342	0.2341	0.3773	0.3248
ionosphere	0.2292	0.2251	0.2215	0.2830	0.2159	0.2213	0.4272	0.2974
congress	0.1981	0.2131	0.1883	0.0349	0.2036	0.2044	0.2151	0.2222
liver	0.2481	0.2493	0.2448	0.2719	0.2407	0.2446	0.4696	0.3276
pima	0.2396	0.2512	0.2369	0.3129	0.2363	0.2370	0.4241	0.3034
parkinsons	0.2012	0.2332	0.2162	0.2359	0.2199	0.2207	0.4099	0.2799
landsat	0.2132	0.2472	0.2225	0.2131	0.2178	0.2357	0.3619	0.3079
krvskp	0.2265	0.2431	0.2036	0.1838	0.2060	0.2117	0.4175	0.2632
heart	0.2294	0.2435	0.2160	0.2887	0.2180	0.2177	0.3831	0.2836
wdbc	0.2012	0.2117	0.2002	0.1128	0.1993	0.2065	0.2696	0.2482
credit	0.2384	0.2529	0.2370	0.2766	0.2316	0.2321	0.4555	0.3064
sonar	0.2274	0.2290	0.2232	0.2944	0.2216	0.2215	0.4173	0.2953
semeion	0.2111	0.2131	0.1944	0.1431	0.2077	0.2133	0.3581	0.2442
splice	0.2078	0.2325	0.2017	0.1234	0.2073	0.2096	0.3217	0.2495
spambase	0.2279	0.2413	0.2145	0.2343	0.2090	0.2242	0.4109	0.2834
waveform	0.1786	0.2465	0.2103	0.1910	0.2116	0.2108	0.3984	0.2683
musk2	0.2322	0.2394	0.2237	0.2641	0.2186	0.2206	0.4504	0.3126
mushroom	0.2306	0.2350	0.2037	0.1743	0.1965	0.2123	0.4851	0.3205

Dataset	CSAda	AdaMEC	AsymAda	CGAda	Calibrated AdaMEC	Calibrated AsymAda	Calibrated CGAda
survival	0.3593	0.2623	0.2337	**0.2260**	0.2302	0.2334	0.2287
ionosphere	0.3157	0.2043	0.2016	0.2090	0.1642	**0.1333**	0.1814
congress	0.0665	0.0840	0.1040	0.0906	**0.0336**	0.0348	**0.0336**
liver	0.3024	0.2729	**0.2378**	0.2454	0.2485	0.2391	0.2491
pima	0.3383	0.2243	0.2261	0.2297	0.2234	**0.2127**	0.2263
parkinsons	0.2693	0.1727	0.1833	0.1725	0.1388	0.1378	**0.1327**
landsat	0.2452	0.3474	0.1656	0.2065	0.2150	**0.1242**	0.2004
krvskp	0.2143	0.1846	0.1727	0.1859	0.1009	**0.0448**	0.1062
heart	0.3177	0.1643	0.1858	0.1802	0.1471	**0.1450**	0.1532
wdbc	0.1418	0.1230	0.1409	0.1243	0.0537	**0.0505**	0.0579
credit	0.3083	0.2239	0.2202	0.2223	0.2131	**0.2009**	0.2157
sonar	0.3304	0.1969	0.2029	0.2005	0.1826	**0.1823**	0.1839
semeion	0.1788	0.1626	0.1413	0.1600	0.0890	**0.0447**	0.0895
splice	0.1556	0.1286	0.1546	0.1400	0.0683	**0.0500**	0.0622
spambase	0.2663	0.2218	0.1785	0.1981	0.1461	**0.0682**	0.1537
waveform	0.2171	0.1317	0.1380	0.1340	0.0695	**0.0686**	0.0696
musk2	0.2943	0.1963	0.2031	0.1970	0.1432	**0.1225**	0.1344
mushroom	0.2066	0.1686	0.0374	0.1039	0.1071	**0.0353**	0.1118

diagrams [22], showing when the differences in loss ranking are statistically significant at the 0.05 level under a Nemenyi post-hoc test [85] are shown in Figure 8.6.

8.2.2 A deeper examination of AdaMEC, CGAda & AsymAda

Among AdaMEC, CGAda & AsymAda, the third variant outranks the other two. This pattern carries over to the calibrated versions of the three algorithms. This might indicate some benefit of making the training phase itself cost-sensitive, but there could be a simpler explanation: AsymAda creates an ensemble of predefined size $M = 100$. AdaMEC & CGAda typically use about 40 weak learners of the maximum M allowed.

Table 8.2: Average *area under the Brier curves* produced by the calibrated versions of AsymAda, AdaMEC & CGAda ensembles of equal ensemble size, for all 18 datasets. The area is equal to the average *Brier score*, so lower values are desirable. The lowest value per dataset is marked in bold. *By constraining AsymAda to use as many weak learners as AdaMEC & CGAda, it loses its advantage over the other two methods. In Figure 8.7 we provide evidence that the three methods do not significantly differ in performance.*

Dataset	Calibrated AdaMEC	Calibrated AsymAda	Calibrated CGAda
survival	0.2337	0.2343	**0.2328**
ionosphere	**0.1711**	0.1994	0.1931
congress	0.0330	0.0358	**0.0328**
liver	0.2494	0.2622	**0.2491**
pima	0.2268	0.2338	**0.2330**
parkinsons	**0.1431**	0.1534	0.1474
landsat	0.2182	0.2421	**0.2137**
krvskp	**0.0991**	0.1405	0.1178
heart	**0.1491**	0.1522	0.1524
wdbc	**0.0557**	0.0626	0.0620
credit	**0.2156**	0.2260	0.2200
sonar	**0.1828**	0.1846	0.1829
semeion	**0.0898**	0.1341	0.1120
splice	**0.0668**	0.1049	0.0729
spambase	**0.1421**	0.2060	0.1699
waveform	0.0699	**0.0688**	0.0702
musk2	0.1397	**0.1367**	0.1408
mushroom	**0.1051**	0.1817	0.1281

This results in AsymAda producing higher margin distributions than the other two methods, which leads to better generalization. Some evidence for this is provided in Figures (8.2 & 8.9), where we have plotted both the average rank attained by each method and the average ensemble size. As we can see calibrated AdaMEC and AsymAda lie on the *Pareto front* of the two objectives: attaining a high rank while building a parsimonious model.

To verify the above hypothesis we also included another set of experiments: we fixed the ensemble size for the calibrated versions of AdaMEC, CGAda & AsymAda to be equal for each run[6]. We then compared the statistical significance of the difference in the average ranking of their Brier scores (low rank indicating better performance) across all datasets. The resulting critical difference diagrams [22], showing when the differences in loss ranking are statistically significant at the 0.01 level under a Nemenyi post-hoc test are shown in Figure 8.7. The results suggest that there is no clearly dominant method among these three[7]. Therefore, any performance benefit AsymAda

[6]AdaMEC & AsymAda were forced to use as many learners as CGAda, with a maximum of 100.

[7]This is not very surprising. As discussed in Chapter 5, all 3 approaches are approximating the minimizer $F^*(\mathbf{x}) = \frac{1}{2}\log\frac{p(y=1|\mathbf{x})}{p(y=-1|\mathbf{x})} + \frac{1}{2}\log\frac{c_{FN}}{c_{FP}}$ of the same global loss $L(F_M) = \mathbb{E}_{\mathbf{x},y}[c(y)e^{-yF_M(\mathbf{x})}]$, albeit in different ways: AdaMEC by shifting the decision threshold, CGAda by reweighting and AsymAda by modifying the base algorithm to simulate splitting the asymmetry equally among all M rounds.

had over the other two methods in our main results was due to its larger number of weak learners. Table 8.2 shows the average Brier Score for each of these three methods once we restrict M, for each dataset.

AdaMEC has been largely overlooked – its generalized form was first presented in [88] in the context of this thesis– despite having some clear practical benefits. It is more flexible than AsymAda, allowing us to e.g. grow the ensemble after the original training if needed. It is faster than AsymAda, as fewer weak learners will be added to the ensemble–weak learner optimization being the computational bottleneck of training. Perhaps most importantly, unlike all other methods included in this study, AdaMEC does not need to retrain the model if the cost ratio changes in deployment[8].

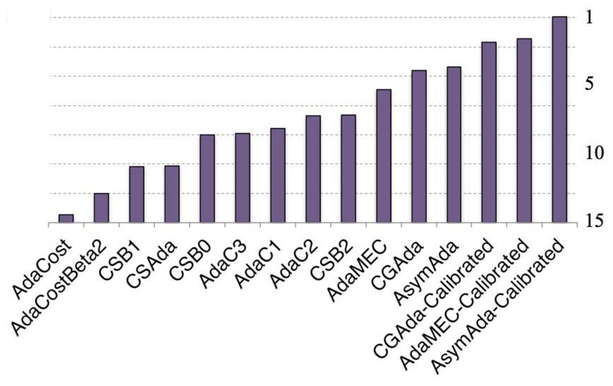

Figure 8.1: All methods, across all datasets, ranked by their average area under the Brier curve (BS) – higher rank (higher bar) is better. This illustrates the resilience of each method to a spectrum of changing cost ratios. The highest ranked method is *AsymAda-Calibrated.*

Finally, any modification of AsymAda w.r.t. AdaBoost specified by Eqs. (3.8 & 3.9) becomes vanishingly small as $M \to \infty$, and AsymAda reduces to AdaBoost. In

[8]That said, we can generalize Eq. (7.5) by interpreting $\hat{p}_0(y=1)$ as capturing the required *change in skew from training to deployment* z', which relates to the skew ratio during training z_{tr} and the skew ratio during deployment z_{dep} via $z_{dep} = \frac{z_{tr} \cdot z'}{z_{tr} \cdot z' + (1-z_{tr}) \cdot (1-z')}$. This allows us to do a 'post-training correction' to the probability estimate computed by AsymAda or CGAda,

$$\hat{p}_{w'}(y=1|\mathbf{x}) = \frac{(1-z')\hat{p}_w(y=1|\mathbf{x})}{(1-z')\hat{p}_w(y=1|\mathbf{x}) + z'(1-\hat{p}_w(y=1|\mathbf{x}))}, \tag{8.1}$$

to account for changes in the imbalance (i.e. for a *prior probability shift* [83, 95]). Still, this requires knowledge of the skew-setup used in training z_{tr}. Additionally, overfitting during training (building an overconfident model that assigns probabilities close to 0 or 1 to the examples) will be impossible to correct using Eq. (8.1). In Chapter 7 we saw that a single overconfident expert suffices for this to happen.

this case, it becomes necessary to use threshold shifting to make the decisions cost-sensitive. In other words, we end up using AdaMEC. For these reasons, as a representative of this group of methods we pick AdaMEC in subsequent comparisons, including both its calibrated and uncalibrated versions to showcase the benefits of calibration.

8.2.3 Examining individual Brier curves

The area under the Brier curve does not take into account the variance across runs, neither does it allow us to observe the different behaviours exhibited by each method under the different degrees of skew. To observe these effects we need to examine the entire Brier curve. For clarity we will only compare some representative methods against AdaMEC & calibrated AdaMEC. CSB2 is chosen for its relatively good performance across datasets. AdaC1 is chosen despite its erratic behaviour, since, on some datasets it ranks among the top-performing methods. The Brier curves for all datasets can be found in Figure 8.3. Some additional comparisons of calibrated AdaMEC, AdaMEC, CSB2 and AdaC2 under different evaluation measures can be found in [87].

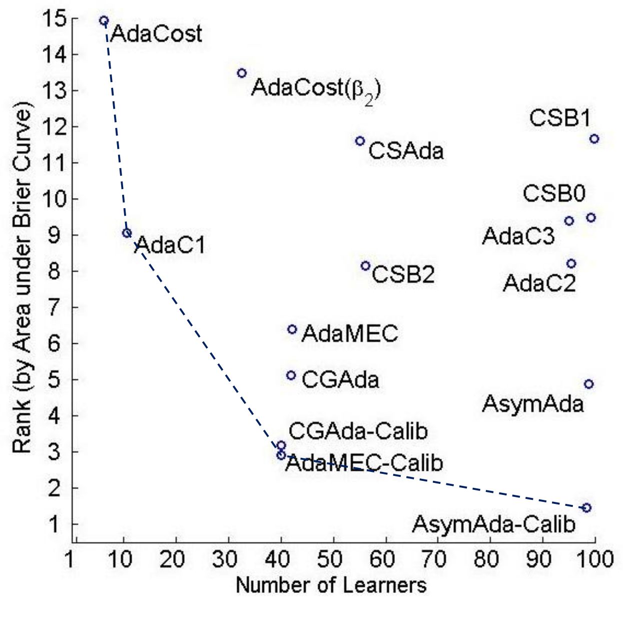

Figure 8.2: Pareto plot showing the tradeoff between the size of the final ensemble and the rank by average area under the Brier curve (BS). The dashed curve demarcates the Pareto front. The plot shows that *calibrated* AdaMEC and *calibrated* AsymAda are pareto-optimal, with the former to be preferred for parsimonious models.

CSB2 performs very poorly under low values of skew ($0.3 < z < 0.7$). This is because of the *saturation* phenomenon, also observed by [66], i.e. the tendency of

CSB2 to construct '*all-positives*' or '*all-negatives*' models. The eagerness of CSB2 to classify examples to the costly class is explained by our theory by the fact that CSB2 overemphasises the costs as we saw in Table 5.2. This strategy starts paying off when the degree of skew becomes very high ($z \leq 0.1$ or $z \geq 0.9$) when, it becomes one of the dominant methods, second only to calibrated AdaMEC.

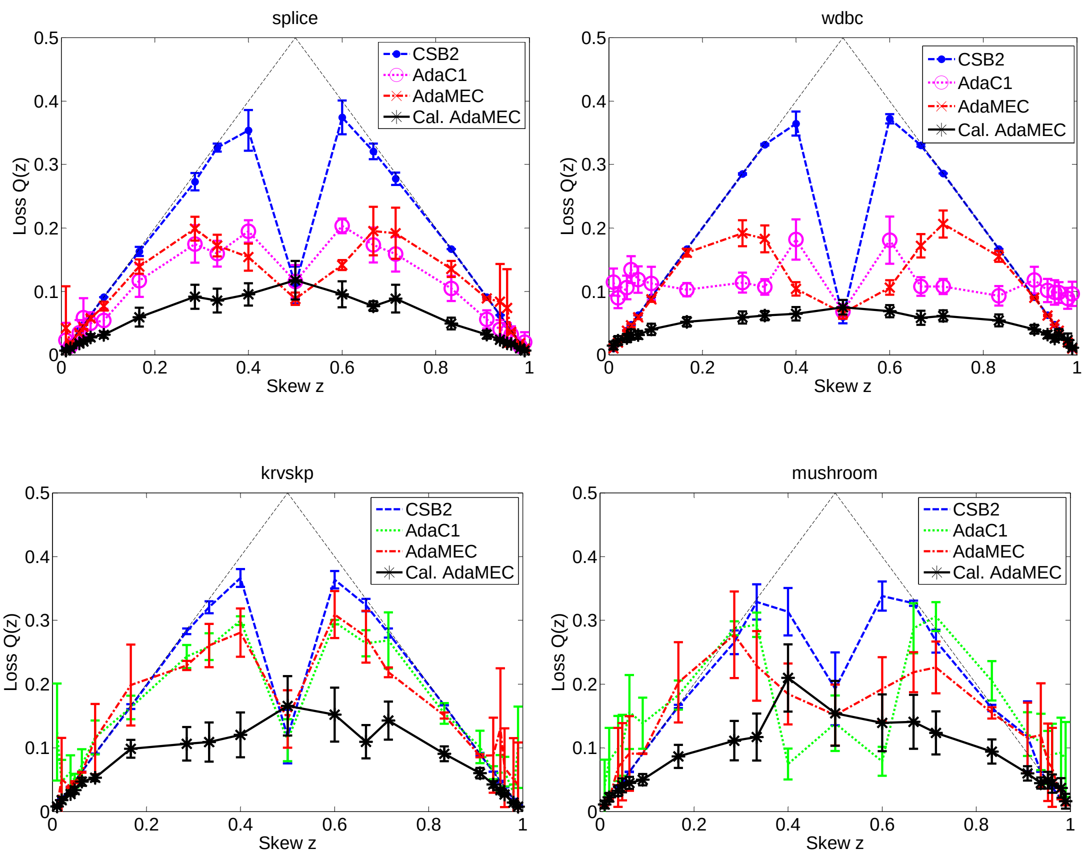

Figure 8.3: Loss Q under various degrees of skew z for some characteristic datasets included in the study. Lower values indicate better cost-sensitive classification performance. The area under each (Brier) curve corresponds to the *Brier Score* of the final model. The difference of the areas under the curve of AdaMEC and that of *calibrated AdaMEC* is due to the reduction of the *calibration loss* component of the BS. The $Q(z) = z$ diagonal, corresponds to the '*all-negatives*' classifier, while the $Q(z) = 1 - z$ diagonal to the '*all-positives*'. CSB2 is prone to saturating leading to high loss when the skew is high. AdaC1 is prone to ignoring the asymmetry, leading to high loss as z moves away from 0.5. *Calibrated AdaMEC*, adopts in each case an asymmetric behaviour that leads to low loss. As a result it consistently attains the lowest –or tied for lowest– loss.

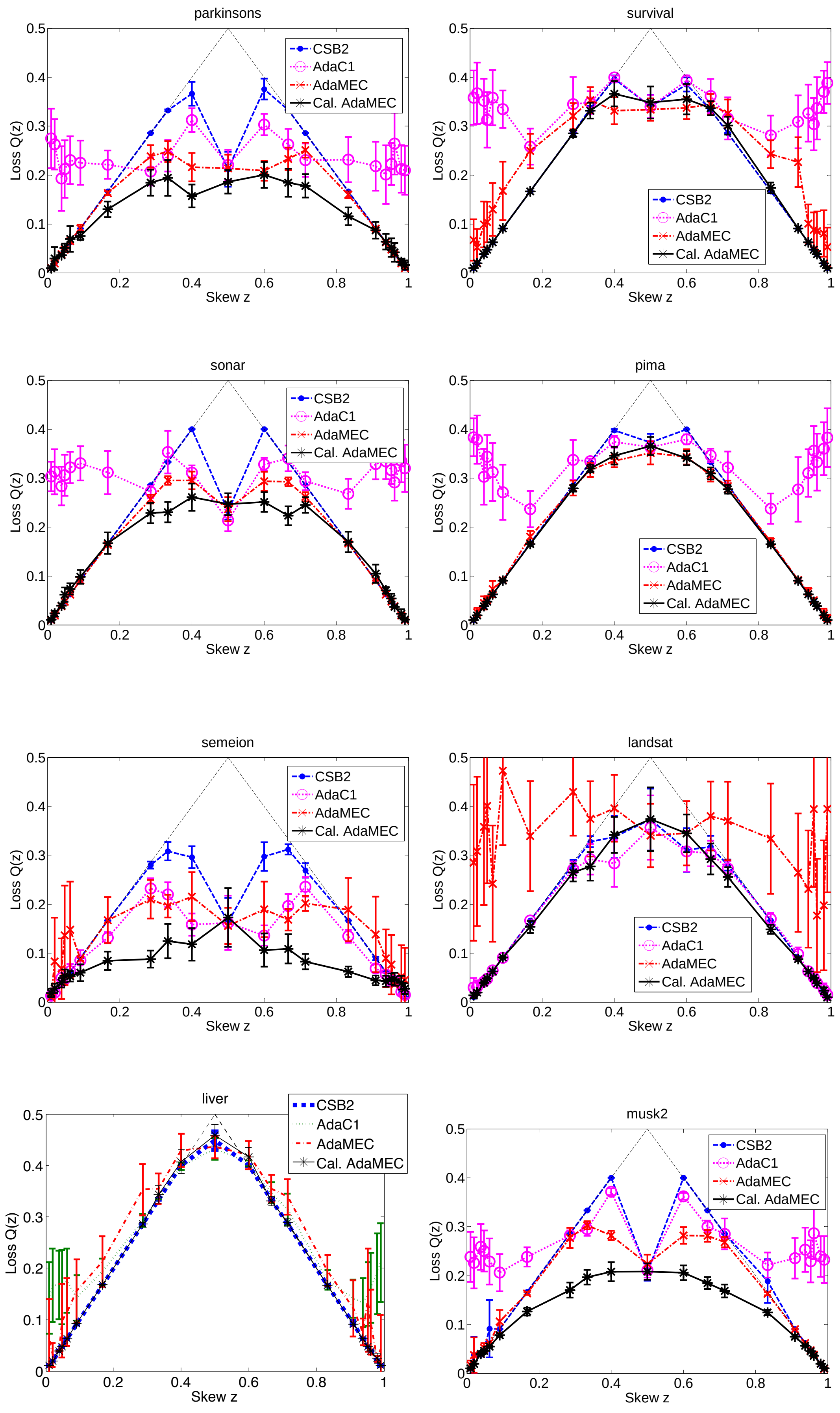

Figure 8.3, continued

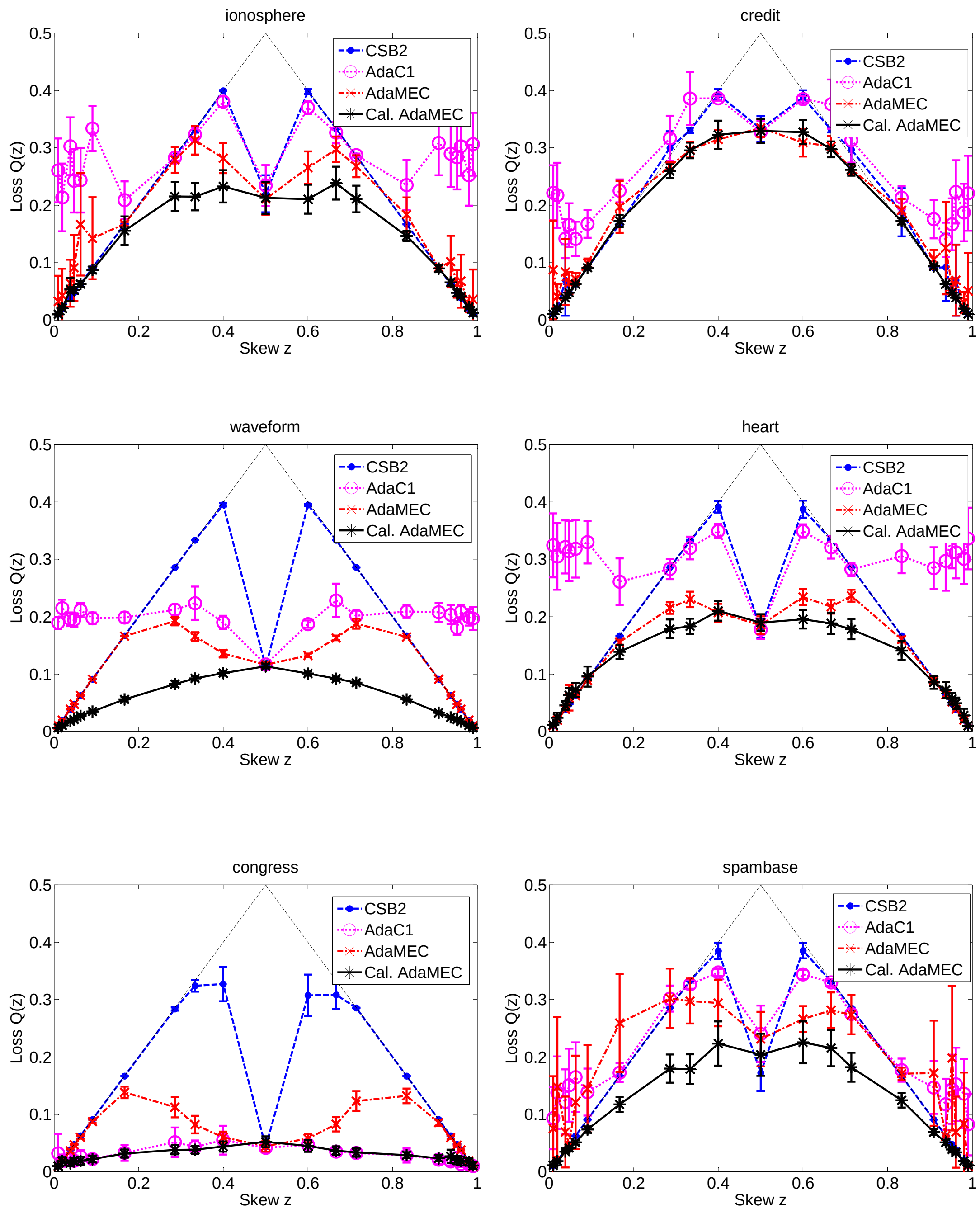

Figure 8.3, continued

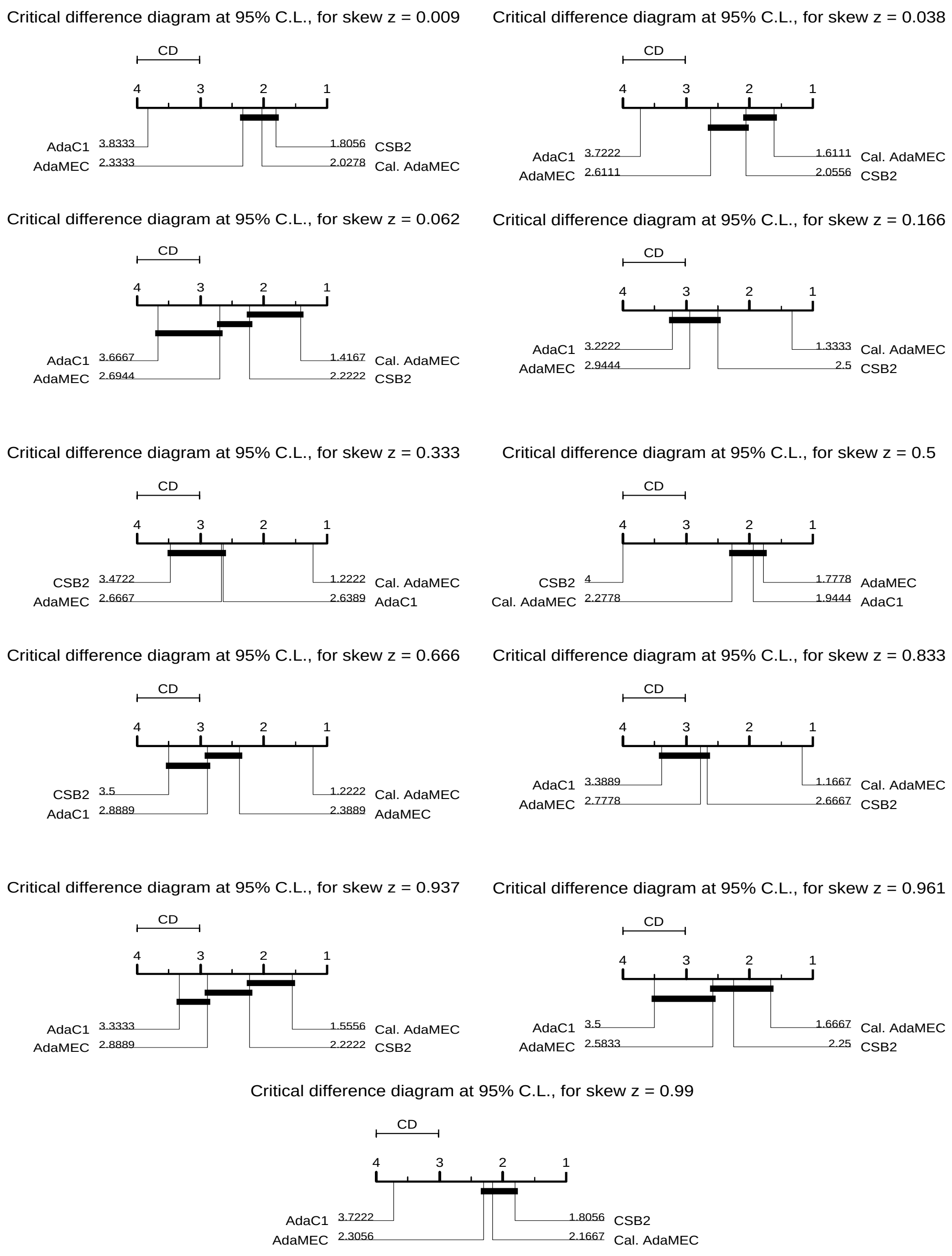

Figure 8.6: Comparison of the average Brier Score ranks across all datasets attained by CSB2, AdaC1, AdaMEC & calibrated AdaMEC against each other under the Nemenyi test, for every other skew ratio examined. Groups of methods that are not significantly different at the 0.05 level are connected. We see that calibrated AdaMEC is consistently ranked best or tied for best, often significantly better than the second best.

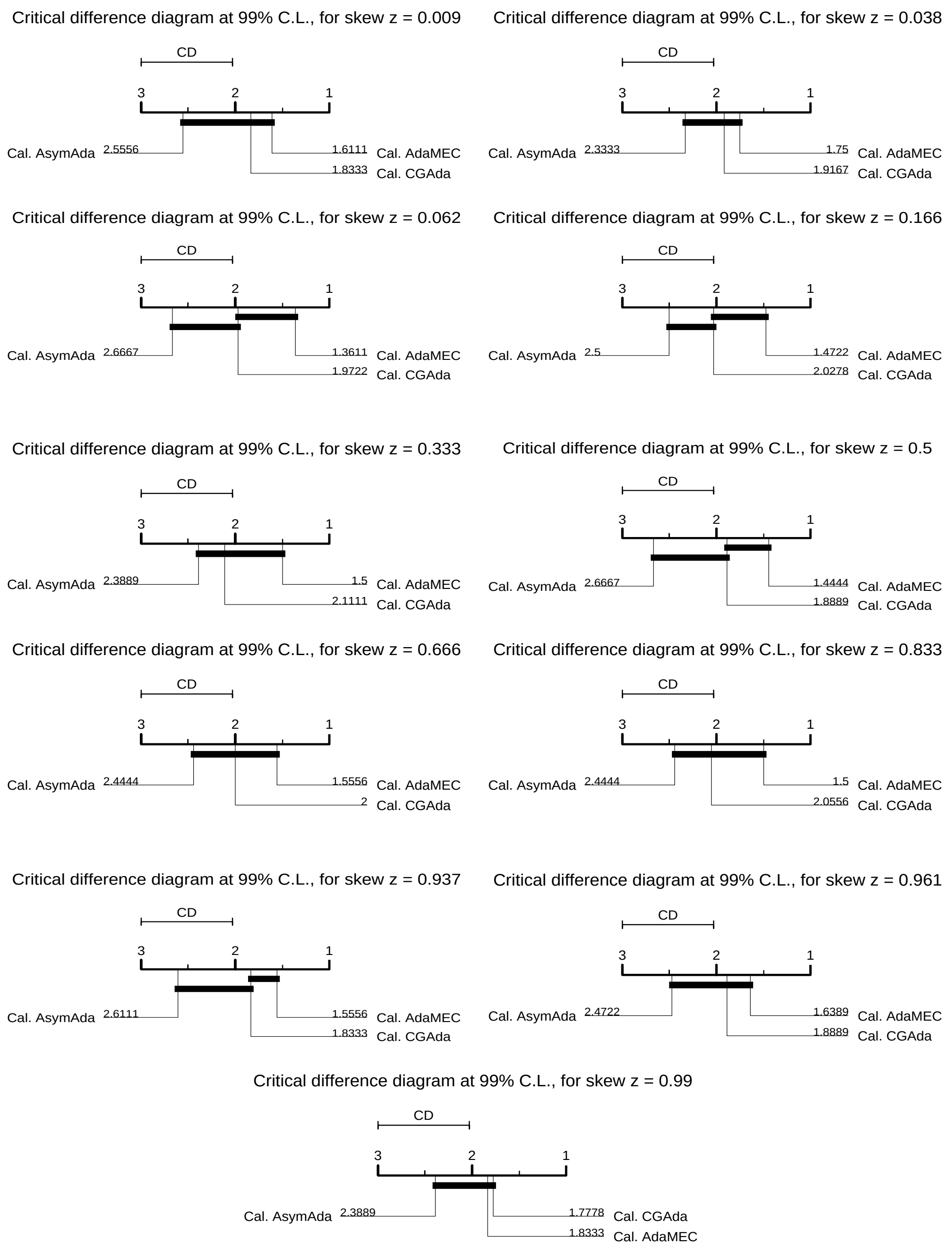

Figure 8.7: Comparison of the average Brier Score ranks across all datasets attained by AdaMEC, CGAda & AsymAda against each other under the Nemenyi test, for every other skew ratio examined. All methods were calibrated and used the *same ensemble size on each run*. Groups of methods that are not significantly different at the 0.01 level are connected. We see no evidence that any method clearly dominates the others.

On the other hand, AdaC1 exhibits particularly poor performance when the skew ratio is high ($z < 0.2$ or $z > 0.8$). AdaMEC tends to perform well for low values of skew, but for higher degrees of imbalance ($z \leq 0.3$ or $z \geq 0.7$), it is outranked.

Calibrated AdaMEC sacrifices part of the dataset to solve the harder problem of probability estimation. As a result, on average it is outranked by the other methods when the task is skew-insensitive ($z = 0.5$). The effect is barely detectable on most datasets as the confidence intervals overlap. A simple solution is the *use of cross-validation for calibration*. But as the imbalance increases, the investment of calibrated AdaMEC in estimating calibrated probabilities pays off and it clearly dominates all other methods. On nearly all datasets and for all values of z examined, it ranks first or tied for first among the 4 methods studied here.

A closer inspection of the individual datasets shows that calibrated AdaMEC is most clearly outperforming the competitors on larger datasets, since a large training set allows it to compute better probability estimates. This effect is more pronounced in datasets which are also high-dimensional (*splice*, *musk2*, *krvskp*, *waveform*, *spambase*). In lower-dimensional datasets, like *mushroom*, the confidence intervals for most methods tend to overlap as the problem is easier.

8.2.4 Calibrating all variants

In a final series of experiments, we included calibrated versions of all cost-sensitive boosting variants covered in this study. For completeness, we also included the original (i.e. cost-insensitive) AdaBoost and its calibrated version. This way, all 26 boosting variants of Table 7.1 were compared against each other on all 18 datasets and 21 degrees of imbalance. The results are presented in Figures (8.8 & 8.9), which are the expanded versions of Figures (8.1 & 8.2) respectively.

Figure 8.8 shows the average rank attained by each method across all datasets in terms of average area under the Brier curve (BS). We can see that calibrated AsymAda, AdaMEC & CGAda outperform their uncalibrated counterparts and they in turn outrank all other variants. Note that again the ensemble size of AsymAda was not restricted, i.e. it always added $M = 100$ learners to the ensemble. Therefore any advantage of AsymAda w.r.t. AdaMEC & CGAda is solely attributed to its larger ensemble size, as we saw in Subsection 8.2.2. Interestingly, the original (i.e. cost-insensitive) AdaBoost and its calibrated version rank immediately after AsymAda, AdaMEC & CGAda and outrank all remaining variants.

These results support our theoretical analysis. The three methods satisfying all four properties of Table 7.1 (calibrated AsymAda, AdaMEC & CGAda) outrank all others. The three methods satisfying fgd-consistency, cost-consistency & asymmetry preservation (AsymAda, AdaMEC & CGAda) outperform all those that lack one or more of these properties –be they calibrated or not.

In other words, calibration alone cannot make up for the absence of one of the other three properties. If the decision rule is not consistent with the costs (lack of cost-consistency), then the classification threshold is poorly chosen. Calibrating the estimates has no effect on the threshold and will not correct for this. If the steps of the algorithm are not consistent with one another (lack of fgd-consistency) calibration cannot correct for this either, as it is performed after the model is constructed. The same is true when the emphasis on the two classes is flipped during training (lack of asymmetry preservation). In the absence of one or more of these properties performing calibration leads to worst performance for almost all methods. The reason for this is the smaller training sample used to train the classifier.

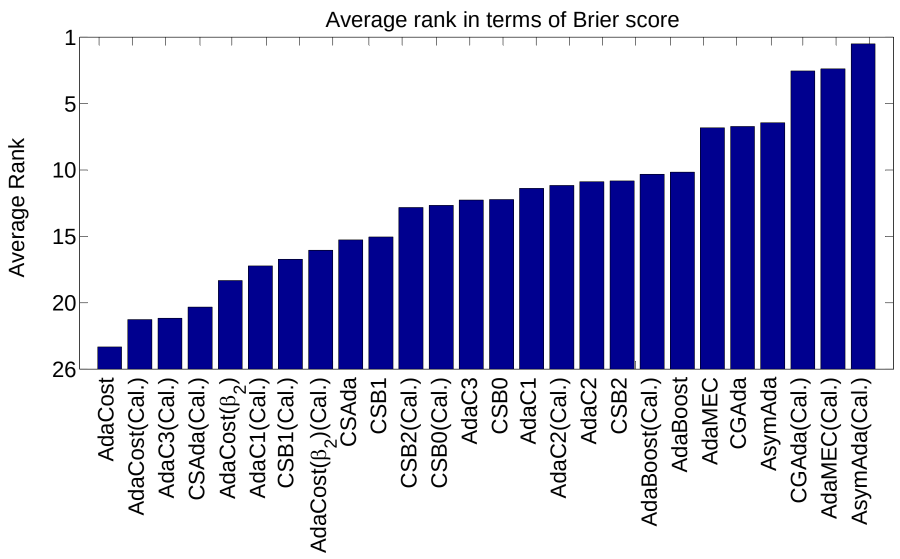

Figure 8.8: All boosting variants of Table 7.1, across all datasets, ranked by their average area under the Brier curve (BS) – higher rank (higher bar) is better. This illustrates the resilience of each method to a spectrum of changing cost ratios. Calibrated AsymAda, AdaMEC & CGAda outperform their uncalibrated counterparts and they in turn outrank all other variants. Interestingly, the methods that rank immediately after AsymAda, AdaMEC & CGAda are the original (i.e. cost-insensitive) AdaBoost and its calibrated version outrank all remaining variants.

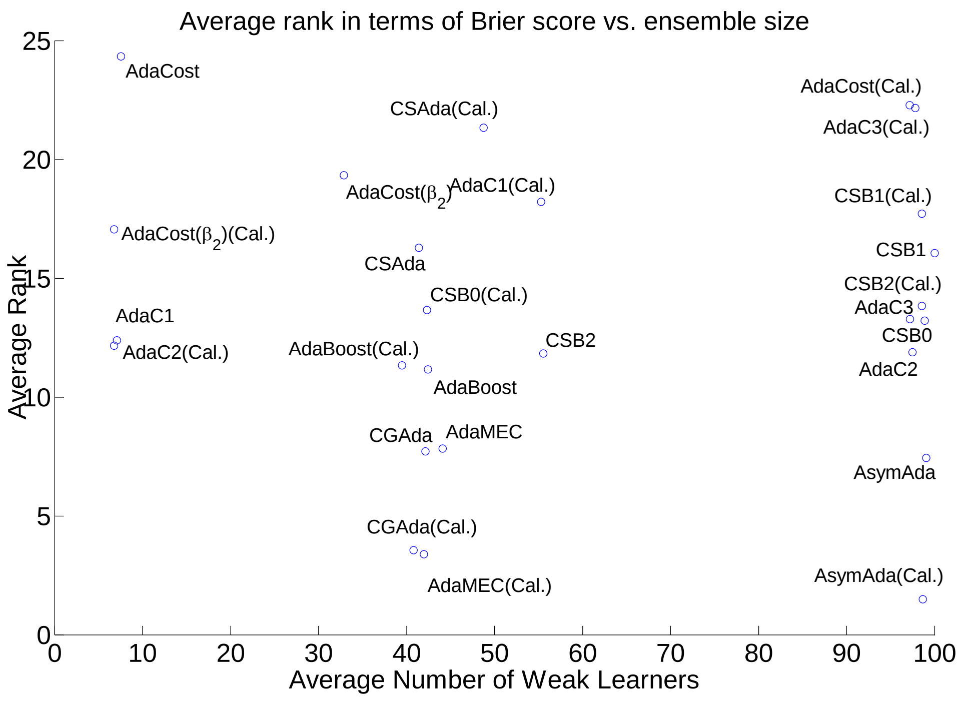

Figure 8.9: Pareto plot showing the tradeoff between the size of the final ensemble and the rank by average area under the Brier curve (BS) for all boosting variants of Table 7.1.

Figure 8.9 shows the ranking of each method in terms of average area under the Brier curve (BS) versus the average size of the final ensemble. Typically, calibrated methods produce smaller ensembles, as they use a smaller dataset to train the additive model. Again, calibrated AsymAda, AdaMEC & CGAda lie on the Pareto front of the two objectives (low BS across all datasets and small average ensemble size).

Chapter 9

Conclusions and future directions

9.1 Conclusions

We analysed the cost sensitive boosting literature spanning the last two decades under a variety of theoretical frameworks. We used tools from four different perspectives: decision theory, functional gradient descent, margin theory, and probabilistic modelling. Here we summarize the key findings and contributions of this work.

Main result:
Our conclusion is that cost-sensitive modifications seem unnecessary for AdaBoost, if proper calibration is applied. This finding is supported by theoretical, empirical and practical arguments analysed below.

Theoretical contributions:
We expanded upon the existing tools of understanding AdaBoost. The tools from the different theory frameworks developed in this thesis are useful in their own right for understanding boosting variants in general and developing new algorithms.

We provided a unified theoretical treatment of cost-sensitive boosting variants and defined desirable theoretical properties for them to satisfy. Each framework we explored provides a different perspective and strengths for the analysis of cost-sensitive boosting variants. Algorithms that do not fit the functional gradient descent framework cannot be viewed as efficient procedures to greedily minimize a loss function of the margin. The decision theoretic analysis shows that certain methods are not implementing decision rules that align with the goal of minimizing the expected cost of future classifications. Margin theory predicts that methods that invert class importance during their execution will exhibit poor generalization performance. Finally, from a probabilistic perspective, the scores generated by boosting variants deviate from true probabilities and need to be properly calibrated before they can be used as actual probability estimates.

We proposed a new, generalized version of an existing cost-sensitive boosting variant with numerous practical benefits. We showed that AdaMEC, as proposed in [117] is a special case of the version given in Chapter 5. The generalized version we propose –unlike the original– is not restricted in terms of the way probability estimates are generated and is thus more flexible and amenable to calibration. Like the original, it does not modify the training algorithm w.r.t. AdaBoost, just the prediction rule. As a result, it maintains all convergence properties of AdaBoost, eschews the need for additional hyperparameters, does not require the ensemble size to be fixed in advance and can be easily adapted to changes in asymmetry between training and deployment without resorting to re-training the model.

We identified weaknesses in the design of existing cost-sensitive boosting variants. Only three algorithms turn out to be consistent with the rules of the functional gradient descent view of boosting, margin theory and decision theory – AdaMEC (special case proposed in [117], generalized in this work), CGAda [63] & AsymAda [123]. However, in our final theory angle, we find they –along with all other variants examined– share a common flaw: they assume that boosting ensembles produce well-calibrated probability estimates. By reinterpreting boosting as a *Product of Experts*, extending [31], we showed that this assumption is *violated* and the estimates produced by boosting variants deviate from true posterior probabilities in a *predictable* fashion.

We produced variants that satisfy all desirable properties. To correct for the systematic distortion of the probability estimates, we applied calibration using *Platt scaling* [92] to AdaMEC, CGAda & AsymAda. The calibrated versions of these variants satisfy all the desirable properties we identified.

Empirical contributions:

We conducted the largest empirical comparison of cost-sensitive boosting variants to date. Experiments on 18 datasets across 21 degrees of imbalance, backed by statistical hypothesis tests, support our theoretical results – showing that once calibrated, AdaMEC, CGAda & AsymAda perform equivalently, and outperform all others.

Final recommendation to practitioners:

Based on simplicity, flexibility, theoretical soundness and performance, we recommend the use of calibrated AdaMEC as a boosting solution to a cost-sensitive or imbalanced class problem when the goal is to minimize the expected risk under known misclassification costs. In other words, we recommend using the *original Adaboost* algorithm with a *shifted decision threshold*, and *calibrated probability estimates*. A detailed pseudocode for the implementation of AdaMEC calibrated via Platt scaling has been provided in Chapter 7[1].

9.2 Future work

This work opens up multiple directions for future research. Here we summarize the most important of those:

Refining probability estimation / calibration

First of all, we note that even though the calibrated algorithms examined match or exceed the performance of other cost-sensitive approaches, there are various parameters of the calibration procedure we left unoptimized, suggesting room for improvement. First and foremost is the choice of calibration method used. In our work we used the logistic sigmoid calibration (also known as Platt-scaling) of raw scores of the form of Eq. (7.6). As mentioned in Section 7.3, this is not the only choice we have, but previous experiments by Niculescu-Mizil & Caruana [86] suggest that it is the overall best in the case of boosting univariate linear base learners when the available data is limited.

An obvious alternative is *isotonic regression*, also examined in [86]. This is more general as it is able to capture any non-decreasing distortion of scores. Although there is both practical [86, 99] and theoretical [99] –besides this work– evidence that the

[1]Matlab code can also be found in the link: http://www.cs.man.ac.uk/∼gbrown/software/ and the original implementation in: http://www.cs.man.ac.uk/∼nikolaon/∼nikolaon_files/mlj2016code.zip.

distortion of scores is roughly sigmoid, it is conceivable that in practice it is not as smooth but rather 'stepwise' i.e. of an isotonic form that isotonic regression can capture. Having a more complex model, isotonic regression also requires more datapoints than Platt-scaling as it is generally more prone to overfitting when the data is limited.

Another method to calibrate probabilities is *quantile-based calibration* [27], effectively a binning (histogram-based) calibration approach [132] with some randomization. This has been recently explored in the context of boosting and found to produce better probability estimates than those of Eq. (7.1). In the same work it was also compared against other techniques that employ randomization –like bagging multiple estimates of the form of Eq. (7.1) or producing estimates of the form of Eq. (7.1) with noise added– and outperformed them.

Finally the *cumulative distribution function (CDF)* of the *Weibull distribution* has also been used to calibrate classifier scores [6], motivated by the *extreme value theory* [108]. This CDF is also of sigmoid form, like the logistic of Platt-scaling, but the idea is that only the tails of the score distribution (i.e. the scores close to the decision boundary) are used for calibration, so it needs fewer data and is possibly more suitable for large-margin classifiers like boosting. Based on this idea, it has been used before in the context of another type of large-margin classifier, the SVM.

Optimizing the balance between model training and calibration

Another aspect of the calibration procedure left unoptimized was how the data was split between training and calibration sets. The optimal ratio of the number of examples used to train the model over that of those used to calibrate its probability estimates will generally depend on the characteristics of the problem: number of examples, complexity of weak learner, complexity of underlying concept to be learned, to name a few. It will also be sensitive to the choice of calibration method. As mentioned above, methods that allow a more complex mapping between scores and posterior probabilities generally require more data to avoid overfitting. Conversely those that look for a simpler mapping will not be much improved by being given more calibration data.

Another possibility to improve the performance of calibrated methods in practice is the use of *leave-one-out cross-validation* for calibration, rather than splitting the data into separate train and calibration sets. As mentioned in Chapter 8 this is expected to reduce the performance loss of a calibrated method when the problem is symmetric. An example on how to use cross-validation to calibrate the model is given in [82].

An interesting theoretical direction would be relating the Brier score decomposition [62, 84] to the bias-variance decomposition of the mean squared error [26, 49]. After all, the Brier score is defined as a mean squared error and such a connection would allow us to explore the *train vs. calibrate* tradeoff, while also accounting for the complexity of the learning algorithm and the calibration method.

Directly constructing good probability estimates via boosting

In order to answer the asymmetric problem posed, our strategy was to effectively phrase it as one of probability estimation. The specific approach taken to do so consisted of two steps: (i) train a model and (ii) calibrate the scores produced by said model to produce probability estimates.

An interesting research direction would be to investigate performing training and calibration in a single step, i.e. choosing at each round of boosting the weak learner so that the new ensemble produces calibrated probability estimates. A good starting point would be to formulate this target as stagewise minimization of an appropriate loss function on an additive model. This modification would then carry over to the steps of the algorithm (weight update, calculation of α coefficients). The fact that scores and margins are just rescaled versions of one another pointed out by Eq. (7.8) should also be leveraged in such an investigation.

Extending analysis and methods to multiclass classification

In this work we were concerned with binary classification and only provided some limited pointers on how to handle multiclass classification, mainly by suggesting splitting the problem into multiple binary ones taking the *one-vs-one* or the *one-vs-all* approach. Interestingly these two methodologies are often as good in terms of classification performance as other approaches (e.g. *error correcting codes (ECOC)*, *single machine methods*[2]) despite their simplicity [97].

In the case of boosting there has been considerable research into producing variants that handle multiple classes via any of the aforementioned approaches (see Chapter 3, or [2] for more details). The same strategies can be used for extending calibration to multiple class cases [133], but this is far from a solved problem [39]. By providing these pointers, we effectively imply that the same 2-step procedure described above (train, then calibrate) can be extended to the multiclass case. Of course, this is true. But is it the optimal way to perform multiclass cost sensitive boosting?

[2]Methods solving a single optimization problem that trains many binary classifiers simultaneously.

A very recent work by Appel et al. [2], takes a different approach. Again the goal is the same as that of this thesis: to classify examples to the minimum risk class. But Appel et al. are concerned with the multiclass case. They argue against calculating scores for each class and then assigning to the minimum risk class by treating said scores as probability estimates. In this thesis we also argued against doing that. What Appel et al. propose as a solution is a single-step approach, dubbed *REBEL*, that aims to minimize a multiclass loss function that factors costs into account, to directly assign examples to the minimum risk class. Their approach is theoretically sound and outperforms methods that treat raw scores as probability estimates. But we note that the authors did not compare their approach to treating *calibrated* versions of the scores as probability estimates. Such a comparison would be very interesting. It could show that either calibration leads to better or at least comparable performance as the single step approach even in multiclass boosting, or that a single-step approach can replace the one we propose here – basically advancing towards a solution to the problem posed in the previous future work direction. Either result would be of particular theoretical and practical interest.

Extending analysis and methods to online learning

Another interesting research avenue is extending our analysis to online boosting. The tools developed in this thesis can be applied to the online learning setting. The online boosting algorithm proposed by Oza [89] is the most popular online boosting variant. It can be expressed as the stagewise minimization of the same loss function as AdaBoost but using *stochastic* gradient descent steps in a function space. All our theoretical analysis (what are the desirable properties, which variants satisfy them, how to derive variants that satisfy them) depends on deriving the underlying loss function, so it can carry over straightforwardly to online boosting.

However in practice performing calibration in an online learning setting, where examples arrive sequentially, is challenging, as splitting into separate training and calibration sets is impossible. A natural question in such a scenario is how should the choice between each of the two actions (training the model or calibrating its probability estimates) be made on a new data point.

Simple solutions would treat the ratio of the number of examples used to train the model over that of those used to calibrate its probability estimates as a hyperparameter to be optimized. More refined ones would make the decision on whether to train or to calibrate on a given example via reinforcement learning approaches like *bandit optimization* [129], aiming to perform at each round the action with the highest expected payoff.

Extending analysis and methods to other goals than minimum risk

In our work the goal was to minimize the risk of classifications, so a large part of it, namely the property of cost-consistency is based on Bayesian decision theory. Had the goal been different, this property would have to be traded with some new one. What would this new property –and the resulting optimal algorithm(s)– be when faced with a Neyman-Pearson problem (Chapter 2), instead of a minimum risk one? Answering this would open new directions for adapting our methodologies to novelty detection applications [76, 77, 91].

An alternative approach would be to change the way the classifier scores are calibrated. This translates to adopting a more general definition of calibration, under which, *'a well-calibrated classifier calculates the cost parameters under which the expected cost for the instance under consideration is the same regardless of the predicted class'* [39]. In [62] and [39] we see how calibration needs to be adapted if the goal is to optimize for the F_β-measure, rather than the expected cost. It would be interesting to see what effect such an approach would have in the case of boosting and how it compares to changes the loss function as done e.g. in [111].

Examining analysis to other theory frameworks

In Chapter 3 we mentioned that AdaBoost has been interpreted under many distinct theoretical frameworks, each shedding light to different aspects of the algorithm. Cost-sensitive boosting variants could also benefit from such a diverse theoretical examination; the current thesis is, after all, a testament to this. Extending our analysis of cost-sensitive boosting to connect it to different mathematical domains could lead to better understanding of the existing algorithms, to new variants or more efficient implementations.

For instance, it should be easy to define cost-sensitive boosting in *game theoretic* terms. This could be achieved by modifying the payoff matrix of [41] to be consistent with the cost matrix of the problem. A direct consequence would be to obtain a cost-sensitive extension of *LP-boost* [21]. Would this new variant coincide with *LPUBoost* by Leskovec and Shawe-Taylor [68]? How would it compare to the ones studied here?

Borrowing ideas from *information theory* [20], we can examine AdaBoost from the perspective of *optimal betting strategies* [73]. The generalization of AdaBoost presented in [73], *AdaBoost*-ρ, which incorporates knowledge about the likely *winners* (here: correctly classified training examples) in its parameter ρ ($\rho = 0.5$ corresponds to zero knowledge, i.e. the original Adaboost) can be used to assign non-equal weight mass to correctly and incorrectly classified examples. This can potentially be leveraged to perform cost-sensitive learning. Will then one (or more) of the existing cost-sensitive boosting variants arise as a special case of *AdaBoost*-ρ?

Another future research direction could be to examine cost-sensitive boosting variants using tools from the domain of *dynamical systems*. Following the work of Rudin et al. [100] on AdaBoost, we can leverage the iterative nature of the weight updates to also express cost-sensitive boosting variants as non-linear iterated maps. Exploring the dynamics of the evolution of the weights could e.g. point out interesting differences among CGAda, AsymAda and AdaMEC in terms of convergence properties. The theoretical tools used in the present thesis show that these three variants are different methods to approximate the same optimal model, but are not sufficient for identifying potential differences in performance (speed or quality of approximation) among them.

Extending analysis and methods beyond boosting

Finally, our key findings have the potential to carry over to other learning algorithms beyond Adaboost. It would be interesting to investigate in what situations calibrating the scores produced by an algorithm and choosing an appropriate decision threshold to assign to the minimum risk class outperforms modifications of the algorithm itself. Doing this would mean reducing cost-sensitive learning to probability estimation, with all the theoretical guarantees and practical advantages discussed in this thesis. A good starting point would be the work of Dmochowski et al. [24]. It suggests that the optimality of threshold-shifting will partially depend on the complexity of the learner, but the theory needs to be extended to explicitly take calibration into account.

Asymptotically, calibration can only improve classification performance [17]. But in practice, we have a finite sample which we need to split into a training and a calibration dataset –or to perform cross-validation. How can we predict in advance whether calibrating the classifier's scores is worth the potential sacrifice in model fitting –or the extra computational cost of cross-validation? Such finite sample issues of calibration need to be studied more extensively, potentially extanding traditional learning theory to include bounds on the performance of scoring classifiers –accounting not only for the classification, but also the scores they produce, i.e. the confidence in their predictions. We saw that these considerations fall within the domain of margin theory and existing margin-based bounds are a subject of ongoing research. But interesting parallels can also be drawn between this line of thinking about the confidence of a learner in its predictions and Vapnik's recent *Learning Using Privileged Information (LUPI)* framework [121, 122].

9.3 Closing word: an argument for simplicity

As a closing word to this thesis, we should note that a key advantage of calibrated AdaMEC is its simplicity. Simplicity is a multifaceted virtue. It can translate to ease of conception, ease of implementation, ease of interpretation and ease of adaptation to changing environments. In this case, calibrated AdaMEC satisfies all of these notions. Moreover, compared to the other calibrated variants, it does not introduce any more free parameters on top of those of the additive model constructed by AdaBoost and those of the calibration function, so the model is also 'simple' in the statistical sense we described in the Introduction.

Simplicity in this sense can be seen as robustness to overfitting, robustness to changes between the training and testing conditions, between laboratory experimentation and real world application with all its nuances. David Hand, in his influential paper '*Classifier technology and the illusion of progress*' [51] expands on these ideas and argues that simpler methods typically yield performance comparable to that of more sophisticated ones, 'to the extend that the difference in performance may be swamped by other sources of uncertainty that generally are not considered in the classical supervised learning paradigm'.

The results of this thesis lend additional support to the case for simpler learning methods. Moreover, the analysis followed a similar reductionist approach by simplifying the relevant literature, unifying it under a common theoretical framework and identifying the key aspects in which the proposed cost-sensitive boosting variants differ. We already elaborated on the benefits of simpler methods. Simplicity on the level of analysis of machine learning approaches can mean unifying seemingly distinct research areas, thus facilitating the flow of understanding between them, or reducing the various approaches to their key elements, making their similarities and differences more explicit. This is particularly useful in machine learning, a field whose methods originate from such diverse areas as statistics, optimization, psychology, physics, artificial intelligence, to name but a few. It is the author's hope that the present thesis, besides the specific problem it addresses, also contributes useful arguments towards a need for simplicity in both the methods themselves and their analysis.

Appendix A

Theorem Proofs

Proof of Theorem 1

Theorem 1: *The generalised formulation of AdaMEC,*

$$H_{AdaMEC}(\mathbf{x}) = sign\left[\hat{p}(y=1|\mathbf{x}) - c\right], \tag{A.1}$$

reduces to

$$H_{AdaMEC}(\mathbf{x}) = sign\left[\sum_{y\in\{-1,1\}} c(y) \sum_{\tau:h_\tau(\mathbf{x})=y} \alpha_\tau h_\tau(\mathbf{x})\right],$$

where

$$c(y) = \begin{cases} c_{FN}, & \text{if } y=1 \\ c_{FP}, & \text{if } y=-1 \end{cases},$$

when probability estimates are raw scores of the form $\hat{p}(y=1|\mathbf{x}) = \frac{\sum_{\tau:h_\tau(\mathbf{x})=1}\alpha_\tau}{\sum_{\tau=1}^{t}\alpha_\tau}$.

Proof: Our generalized formulation of AdaMEC's prediction rule is

$$H_{AdaMEC}(\mathbf{x}) = sign\left[\hat{p}(y=1|\mathbf{x}) - c\right],$$

where $\hat{p}(y=1|\mathbf{x})$ denotes the probability estimate the AdaBoost ensemble F_t assigns to example $\mathbf{x}$ belonging to the positive class, regardless of how it is estimated.

Suppose we opt to use as probability estimates raw scores of the form

$$\hat{p}(y=1|\mathbf{x}) = s(\mathbf{x}) = \frac{\sum_{\tau:h_\tau(\mathbf{x})=1}\alpha_\tau}{\sum_{\tau=1}^{t}\alpha_\tau}.$$

Then Eq. (A.1) becomes

$$\begin{aligned} H_{AdaMEC}(\mathbf{x}) &= sign\left[(1-c)\hat{p}(y=1|\mathbf{x}) - c(1-\hat{p}(y=1|\mathbf{x}))\right] \\ &= sign\left[\frac{c_{FN}}{c_{FP}+c_{FN}}\frac{\sum_{\tau:h_\tau(\mathbf{x})=1}\alpha_\tau}{\sum_{\tau=1}^{t}\alpha_\tau} - \frac{c_{FP}}{c_{FP}+c_{FN}}(1-\frac{\sum_{\tau:h_\tau(\mathbf{x})=1}\alpha_\tau}{\sum_{\tau=1}^{t}\alpha_\tau})\right]. \end{aligned}$$

Since $c_{FP}+c_{FN}$ is bounded, constant and non-negative, this is equivalent to

$$\begin{aligned} H_{AdaMEC}(\mathbf{x}) &= sign\left[c_{FN}\frac{\sum_{\tau:h_\tau(x)=1}\alpha_\tau}{\sum_{\tau=1}^{t}\alpha_\tau} + c_{FP}(1-\frac{\sum_{\tau:h_\tau(\mathbf{x})=1}\alpha_\tau}{\sum_{\tau=1}^{t}\alpha_\tau})\right] \\ &= sign\left[c_{FN}\frac{\sum_{\tau:h_\tau(\mathbf{x})=1}\alpha_\tau}{\sum_{\tau=1}^{t}\alpha_\tau} - c_{FP}\frac{\sum_{\tau:h_\tau(\mathbf{x})=-1}\alpha_\tau}{\sum_{\tau=1}^{t}\alpha_\tau}\right]. \end{aligned}$$

As $\sum_{\tau=1}^{t}\alpha_\tau$ is bounded[1], constant and non-negative, we get the equivalent rule

$$H_{AdaMEC}(\mathbf{x}) = sign\left[c_{FN}\sum_{\tau:h_\tau(\mathbf{x})=1}\alpha_\tau - c_{FP}\sum_{\tau:h_\tau(\mathbf{x})=-1}\alpha_\tau\right].$$

Rearranging gives us

$$H_{AdaMEC}(\mathbf{x}) = sign\left[\sum_{y\in\{-1,1\}} c(y)\sum_{\tau:h_\tau(\mathbf{x})=y}\alpha_\tau h_\tau(\mathbf{x})\right],$$

where

$$c(y) = \begin{cases} c_{FN}, & \text{if } y=1 \\ c_{FP}, & \text{if } y=-1. \end{cases}$$

□

[1]If we had $\sum_{\tau=1}^{t}\alpha_\tau = \infty$, this would mean that at least one of $\{\alpha_\tau|\tau=1,\ldots,t\}$ is ∞, or equivalently has an error $\varepsilon_\tau = 0$. In that case we could discard the rest of the ensemble and use only (one of) the weak learner(s) with $\varepsilon_\tau = 0$ to make predictions, as such a weak learner alone perfectly classifies the training data.

Proof of Theorem 2

Theorem 2: *The probability estimate assigned to class $y = 1$ by an AdaBoost ensemble F_t on an example $\mathbf{x}$ constitutes a product of experts*

$$\hat{p}(y=1|\mathbf{x}) = \frac{\prod_{\tau=1}^{t} \hat{p}_\tau(y=1|\mathbf{x})}{\prod_{\tau=1}^{t} \hat{p}_\tau(y=1|\mathbf{x}) + \prod_{\tau=1}^{t} \hat{p}_\tau(y=-1|\mathbf{x})},$$

with experts of the form

$$\hat{p}_\tau(y=1|\mathbf{x}) = \begin{cases} \varepsilon_\tau \quad , & \text{if } h_\tau(\mathbf{x}) = -1 \\ 1-\varepsilon_\tau, & \text{if } h_\tau(\mathbf{x}) = 1, \end{cases}$$

$$\hat{p}_\tau(y=-1|\mathbf{x}) = \begin{cases} 1-\varepsilon_\tau, & \text{if } h_\tau(\mathbf{x}) = -1 \\ \varepsilon_\tau \quad , & \text{if } h_\tau(\mathbf{x}) = 1, \end{cases}$$

where ε_τ is the weighted error of the τ-th weak learner and $h_\tau(\mathbf{x}) \in \{-1,1\}$ its prediction on example $\mathbf{x}$.

Proof: Assume an unknown distribution $p(\mathbf{x}, y)$. Define $F^*(\mathbf{x})$ as the population minimiser of an exponential loss function:

$$F^*(\mathbf{x}) = \arg\min_F E_{\mathbf{x}y}\left\{e^{-yF(\mathbf{x})}\right\}$$

To find $F^*(\mathbf{x})$, it is sufficient to minimize $E_{y|\mathbf{x}}\left\{e^{-yF(\mathbf{x})}\right\}$, for any $\mathbf{x}$. Therefore,

$$\frac{\partial E_{y|\mathbf{x}}\left\{e^{-yF(\mathbf{x})}\right\}}{\partial F(\mathbf{x})} = 0 \implies \frac{\partial(p(y=1|\mathbf{x})e^{-F(\mathbf{x})} + p(y=-1|\mathbf{x})e^{F(\mathbf{x})})}{\partial F(\mathbf{x})} = 0 \implies$$
$$-p(y=1|\mathbf{x})e^{-F^*(\mathbf{x})} + p(y=-1|\mathbf{x})e^{F^*(\mathbf{x})} = 0 \implies$$
$$\frac{e^{F^*(\mathbf{x})}}{e^{-F^*(\mathbf{x})}} = \frac{p(y=1|\mathbf{x})}{1-p(y=1|\mathbf{x})} \implies F^*(\mathbf{x}) = \frac{1}{2}\log\frac{p(y=1|\mathbf{x})}{1-p(y=1|\mathbf{x})}$$

which also implies that

$$p(y=1|\mathbf{x}) = \frac{1}{1+e^{-2F^*(\mathbf{x})}}$$

Now assume $F^*(\mathbf{x})$ is approximated by an additive model:

$$F^*(\mathbf{x}) \approx F_t(\mathbf{x}) = \sum_{\tau=1}^{t} \alpha_\tau h_\tau(\mathbf{x})$$

where $\forall\tau$, we have $\alpha_\tau \in \mathbb{R}$ and $h_\tau(\mathbf{x}) \in \{-1,+1\}$, then we have,

$$p(y=1|\mathbf{x}) \approx \hat{p}(y=1|\mathbf{x}) \quad = \quad \frac{1}{1+e^{-2\sum_{\tau=1}^{t}\alpha_\tau h_\tau(\mathbf{x})}} \tag{A.2}$$

Adaboost minimises $E_{\mathbf{x}y}\left\{e^{-yF(\mathbf{x})}\right\}$ via a greedy stage-wise addition of terms to the model $F_t(\mathbf{x})$, using an empirical risk approximation:

$$E_{\mathbf{x}y}\left\{e^{-yF(\mathbf{x})}\right\} \approx \frac{1}{N}\sum_{i=1}^{N} e^{-y_i\sum_{\tau=1}^{t}\alpha_\tau h_\tau(\mathbf{x}_i)} = J_{Ada}$$

Under the greedy optimization scheme, the optimal value for α_τ is

$$\frac{\partial J_{Ada}(\alpha_\tau)}{\partial\alpha_\tau} = 0 \implies \alpha_\tau = \frac{1}{2}\log\frac{1-\varepsilon_\tau}{\varepsilon_\tau}, \tag{A.3}$$

where $\varepsilon_\tau = \frac{\sum_{i:h_\tau(\mathbf{x}_i)\neq y_i} D_i^\tau}{\sum_{i=1}^{N} D_i^\tau}$ is the weighted error of the weak learner added on round τ.

Substituting α_τ from Eq. (A.3) into Eq. (A.2) gives us that the probability estimate assigned to class $y=1$ by an AdaBoost ensemble on an example $\mathbf{x}$ is

$$\hat{p}(y=1|\mathbf{x}) = \frac{1}{1+e^{-2\sum_{\tau=1}^{t}\frac{1}{2}\log\frac{1-\varepsilon_\tau}{\varepsilon_\tau}h_\tau(\mathbf{x})}}$$

which can be rearranged to

$$\begin{aligned}\hat{p}(y=1|\mathbf{x}) &= \frac{1}{1+\prod_{\tau=1}^{t}(\frac{\varepsilon_\tau}{1-\varepsilon_\tau})^{h_\tau(\mathbf{x})}} \\ &= \frac{\prod_{\tau=1}^{t}(1-\varepsilon_\tau)^{h_\tau(\mathbf{x})}}{\prod_{\tau=1}^{t}(1-\varepsilon_\tau)^{h_\tau(\mathbf{x})}+\prod_{\tau=1}^{t}(\varepsilon_\tau)^{h_\tau(\mathbf{x})}}.\end{aligned}$$

So the probability estimates of AdaBoost have the form for a product of (unnormalized) experts

$$\hat{p}(y=1|\mathbf{x}) \quad = \quad \frac{\prod_{\tau=1}^{t}\phi_\tau(y=1|\mathbf{x})}{\prod_{\tau=1}^{t}\phi_\tau(y=1|\mathbf{x})+\prod_{\tau=1}^{t}\phi_\tau(y=-1|\mathbf{x})}$$

$$\begin{aligned}\phi_\tau(y=1|\mathbf{x}) &= (1-\varepsilon_\tau)^{h_\tau(\mathbf{x})}\\ \phi_\tau(y=-1|\mathbf{x}) &= \varepsilon_\tau^{h_\tau(\mathbf{x})},\end{aligned}$$

which can be normalised to give:

$$\hat{p}_\tau(y=1|\mathbf{x}) = \frac{(1-\varepsilon_\tau)^{h_\tau(\mathbf{x})}}{(1-\varepsilon_\tau)^{h_\tau(\mathbf{x})}+\varepsilon_\tau^{h_\tau(\mathbf{x})}} = \begin{cases}\varepsilon_\tau \quad , & \text{if } h_\tau(\mathbf{x})=-1\\ 1-\varepsilon_\tau, & \text{if } h_\tau(\mathbf{x})=1,\end{cases}$$

$$\hat{p}_\tau(y=-1|\mathbf{x}) = \frac{\varepsilon_\tau^{h_\tau(\mathbf{x})}}{(1-\varepsilon_\tau)^{h_\tau(\mathbf{x})}+\varepsilon_\tau^{h_\tau(\mathbf{x})}} = \begin{cases}1-\varepsilon_\tau, & \text{if } h_\tau(\mathbf{x})=-1\\ \varepsilon_\tau \quad , & \text{if } h_\tau(\mathbf{x})=1.\end{cases}$$

□

Proof of Theorem 3

Theorem 3: *Let there be a boosting variant approximating*

$$F^*(\mathbf{x}) = K_p \log \frac{p(y=1|\mathbf{x})}{1-p(y=1|\mathbf{x})} + K_c \log \frac{c_{FN}}{c_{FP}}, \tag{A.4}$$

where K_p and K_c are constants, with an additive model $F_t(\mathbf{x}) = \sum_\tau \alpha_\tau h_\tau(\mathbf{x})$. The probability estimate assigned to class $y=1$ by the ensemble F_t, on an example $\mathbf{x}$ has the form of a product of experts:

$$\hat{p}_w(y=1|\mathbf{x}) \quad = \quad \frac{\hat{p}_0(y=1)\prod_{\tau=1}^t \hat{p}_\tau(y=1|\mathbf{x})}{\hat{p}_0(y=1)\prod_{\tau=1}^t \hat{p}_\tau(y=1|\mathbf{x}) + \hat{p}_0(y=-1)\prod_{\tau=1}^t \hat{p}_\tau(y=-1|\mathbf{x})},$$

with experts of the form

$$\hat{p}_0(y=1) = \frac{1}{1+\left(\frac{c_{FN}}{c_{FP}}\right)^{K_c/K_p}}$$

$$\hat{p}_0(y=-1) = \frac{\left(\frac{c_{FN}}{c_{FP}}\right)^{K_c/K_p}}{1+\left(\frac{c_{FN}}{c_{FP}}\right)^{K_c/K_p}}$$

$$\hat{p}_\tau(y=1|\mathbf{x}) = \frac{(1-\varepsilon_\tau)^{h_\tau(\mathbf{x})/2K_p}}{(1-\varepsilon_\tau)^{h_\tau(\mathbf{x})/2K_p} + \varepsilon_\tau{}^{h_\tau(\mathbf{x})/2K_p}}$$

$$\hat{p}_\tau(y=-1|\mathbf{x}) = \frac{\varepsilon_\tau{}^{h_\tau(\mathbf{x})/2K_p}}{(1-\varepsilon_\tau)^{h_\tau(\mathbf{x})/2K_p} + \varepsilon_\tau{}^{h_\tau(\mathbf{x})/2K_p}}$$

where ε_τ is the weighted error of the τ-th weak learner, whose definition is given in the description of the algorithm, $\alpha_\tau = \frac{1}{2}\log\frac{1-\varepsilon_\tau}{\varepsilon_\tau}$ the corresponding confidence coefficient and $h_\tau(\mathbf{x}) \in \{-1,1\}$ its prediction on example $\mathbf{x}$.

Proof: Solving Eq. (A.4) w.r.t. the probability $p(y=1|\mathbf{x})$, we get that $p(y=1|\mathbf{x})$ is a function of the true minimizer $F^*(\mathbf{x})$,

$$p(y=1|\mathbf{x}) \;=\; \frac{1}{1+\left(\frac{c_{FN}}{c_{FP}}\right)^{K_c/K_p} e^{-F^*(\mathbf{x})/K_p}} \tag{A.5}$$

Now since $F^*(\mathbf{x})$ is approximated by an additive model:

$$F^*(\mathbf{x}) \approx F_t(\mathbf{x}) = \sum_{\tau=1}^{t} \alpha_\tau h_\tau(\mathbf{x})$$

where $\forall\tau$, we have $\alpha_\tau \in \mathbb{R}$ and $h_\tau(\mathbf{x}) \in \{-1,+1\}$, we can approximate the true probability of Eq. (A.5) by the estimate

$$\hat{p}(y=1|\mathbf{x}) = \frac{1}{1+\left(\frac{c_{FN}}{c_{FP}}\right)^{K_c/K_p} e^{-(\sum_{\tau=1}^{t}\alpha_\tau h_\tau(\mathbf{x}))/K_p}}. \tag{A.6}$$

In Table 3.2, we see that all boosting variants have the following general form for the α_τ coefficients:

$$\alpha_\tau = \frac{1}{2}\log\frac{1-\varepsilon_\tau}{\varepsilon_\tau}, \tag{A.7}$$

where ε_τ is a measure of weighted error of the weak learner added on round τ, whose definition differs across algorithms [2].

Substituting α_τ from Eq. (A.7) into Eq. (A.6) gives us that the probability estimate assigned to class $y=1$ by an AdaBoost ensemble on an example $\mathbf{x}$ is

$$\hat{p}(y=1|\mathbf{x}) = \frac{1}{1+\left(\frac{c_{FN}}{c_{FP}}\right)^{K_c/K_p} e^{-(\sum_{\tau=1}^{t}\log\frac{1-\varepsilon_\tau}{\varepsilon_\tau}h_\tau(\mathbf{x}))/2K_p}}$$

which can be rearranged to

$$\begin{aligned}\hat{p}(y=1|\mathbf{x}) &= \frac{1}{1+\left(\frac{c_{FN}}{c_{FP}}\right)^{K_c/K_p}\prod_{\tau=1}^{t}\left(\frac{\varepsilon_\tau}{1-\varepsilon_\tau}\right)^{h_\tau(\mathbf{x})/2Kp}} \\ &= \frac{c_{FP}{}^{K_c/K_p}\prod_{\tau=1}^{t}(1-\varepsilon_\tau)^{h_\tau(\mathbf{x})/2Kp}}{c_{FP}{}^{K_c/K_p}\prod_{\tau=1}^{t}(1-\varepsilon_\tau)^{h_\tau(\mathbf{x})/2Kp}+c_{FN}{}^{K_c/K_p}\prod_{\tau=1}^{t}(\varepsilon_\tau)^{h_\tau(\mathbf{x})}/2Kp}.\end{aligned}$$

So the probability estimates of the boosting ensemble have the form for a prior-weighted product of (unnormalized) experts

$$\hat{p}_w(y=1|\mathbf{x}) = \frac{\phi_0(y=1)\prod_{\tau=1}^{t}\phi_\tau(y=1|\mathbf{x})}{\phi_0(y=1)\prod_{\tau=1}^{t}\phi_\tau(y=1|\mathbf{x})+\phi_0(y=-1)\prod_{\tau=1}^{t}\phi_\tau(y=-1|\mathbf{x})},$$

[2]Table 3.2 mentions that CSAda & AdaDB do not have a closed-form solution for α_τ. This fact does not affect this proof. We can still express the numerical value of α_τ these two algorithms calculate in the form of Eq. (A.7) if we define $\varepsilon_\tau = \frac{1}{1+e^{2\alpha_\tau}}$. It is the additive nature of $F_t(\mathbf{x})$ and the exponential form of $\hat{p}(y=1|\mathbf{x})$ that result in the latter corresponding to a PoE.

with experts

$$\begin{aligned}
\phi_0(y=1) &= {c_{FP}}^{K_c/K_p} \\
\phi_0(y=-1) &= {c_{FN}}^{K_c/K_p} \\
\phi_\tau(y=1|\mathbf{x}) &= (1-\varepsilon_\tau)^{h_\tau(\mathbf{x})/2K_p} \\
\phi_\tau(y=-1|\mathbf{x}) &= \varepsilon_\tau^{h_\tau(\mathbf{x})/2K_p},
\end{aligned}$$

which can be normalised to give:

$$\hat{p}_0(y=1) = \frac{1}{1+\left(\frac{c_{FN}}{c_{FP}}\right)^{K_c/K_p}}$$

$$\hat{p}_0(y=-1) = \frac{\left(\frac{c_{FN}}{c_{FP}}\right)^{K_c/K_p}}{1+\left(\frac{c_{FN}}{c_{FP}}\right)^{K_c/K_p}}$$

$$\hat{p}_\tau(y=1|\mathbf{x}) = \frac{(1-\varepsilon_\tau)^{h_\tau(\mathbf{x})/2K_p}}{(1-\varepsilon_\tau)^{h_\tau(\mathbf{x})/2K_p}+{\varepsilon_\tau}^{h_\tau(\mathbf{x})/2K_p}}$$

$$\hat{p}_\tau(y=-1|\mathbf{x}) = \frac{{\varepsilon_\tau}^{h_\tau(\mathbf{x})/2K_p}}{(1-\varepsilon_\tau)^{h_\tau(\mathbf{x})/2K_p}+{\varepsilon_\tau}^{h_\tau(\mathbf{x})/2K_p}}$$

$\square$

Appendix B

Datasets Used

Table B.1: Characteristics of the datasets used in our experiments; number of instances used, number of features and number of classes. The class chosen as ‘positive’ was the minority class in the original file. In multiclass datasets, we followed a 1-vs-all approach, where the negative class consisted of uniformly sampled examples from the remaining classes. All datasets are taken from the UCI repository and can be found in the link: https://archive.ics.uci.edu/ml/datasets.html.

Dataset	# Instances	# Features	# Classes
parkinsons	96	22	2
survival	162	3	2
sonar	194	60	2
heart	240	13	2
ionosphere	252	34	2
liver	290	6	2
semeion	322	256	10
congress	336	16	2
wdbc	424	31	2
pima	576	8	2
credit	600	24	2
landsat	1252	36	6
splice	1524	60	3
musk2	2034	166	2
krvskp	3054	36	2
waveform	3306	40	3
spambase	3626	57	2
mushroom	7832	21	2

Bibliography

[1] E. L. Allwein, R. E. Schapire, and Y. Singer. Reducing multiclass to binary: A unifying approach for margin classifiers. *Journal of Machine Learning Research*, 1:113–141, 2000.

[2] R. Appel, X. Burgos-Artizzu, and P. Perona. Improved multi-class cost-sensitive boosting via estimation of the minimum-risk class. *arXiv:1607.03547v1*, 2016.

[3] S. Arora, E. Hazan, and S. Kale. The multiplicative weights update method: a meta-algorithm and applications. *Theory of Computing*, 8(1):121–164, 2012.

[4] D. Barber. *Bayesian Reasoning and Machine Learning*. Cambridge University Press, 2012.

[5] G. E. A. P. A. Batista, R. C. Prati, and M. C. Monard. A study of the behavior of several methods for balancing machine learning training data. *SIGKDD Explorations Newsletter*, 6(1):20–29, 2004.

[6] A. Bendale and T. Boult. Reliable posterior probability estimation for streaming face recognition. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 56–63, 2014.

[7] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., 2006.

[8] L. Breiman. Prediction games and arcing algorithms. *Neural Computation*, 11(7):1493–1517, 1999.

[9] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[10] G. W. Brier. Verification of forecasts expressed in terms of probability. *Monthly Weather Review*, 78:1–3, 1950.

[11] P. Bühlmann and T. Hothorn. Boosting algorithms: Regularization, prediction and model fitting. *Statistical Science*, 22(4):477–505, 2007.

[12] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender. Learning to rank using gradient descent. In *International Conference on Machine Learning*, pages 89–96, 2005.

[13] R. Caruana and A. Niculescu-Mizil. An empirical comparison of supervised learning algorithms. In *International Conference on Machine Learning*, pages 161–168, 2006.

[14] E. Chastain, A. Livnat, C. Papadimitriou, and U. Vazirani. Algorithms, games, and evolution. *Proceedings of the National Academy of Sciences*, 111(29):10620–10623, 2014.

[15] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. SMOTE: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16(1):321–357, 2002.

[16] N. V. Chawla, N. Japkowicz, and A. Kotcz. Editorial: Special issue on learning from imbalanced data sets. *SIGKDD Explorations Newsletter*, 6(1):1–6, 2004.

[17] I. Cohen and M. Goldszmidt. Properties and benefits of calibrated classifiers. In *European Conference on Principles of Data Mining and Knowledge Discovery*, pages 125–136. Springer, 2004.

[18] M. Collins, R. E. Schapire, and Y. Singer. Logistic regression, AdaBoost and Bregman distances. In N. Cesa-Bianchi and S. A. Goldman, editors, *Conference On Learning Theory*, pages 158–169. Morgan Kaufmann, 2000.

[19] D. Cossock and T. Zhang. Statistical analysis of Bayes optimal subset ranking. *IEEE Transactions on Information Theory*, 54(11):5140–5154, 2008.

[20] T. M. Cover and J. A. Thomas. *Elements of information theory*. John Wiley & Sons, 2012.

[21] A. Demiriz, K. P. Bennett, and J. Shawe-Taylor. Linear programming boosting via column generation. *Machine Learning*, 46(1–3):225–254, 2002.

[22] J. Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006.

[23] T. G. Dietterich and G. Bakiri. Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, 2:263–286, 1995.

[24] J. P. Dmochowski, P. Sajda, and L. C. Parra. Maximum likelihood in cost-sensitive learning: Model specification, approximations, and upper bounds. *Journal of Machine Learning Research*, 11, December 2010.

[25] P. Domingos. Metacost: A general method for making classifiers cost-sensitive. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 155–164, New York, NY, USA, 1999. ACM.

[26] P. Domingos. A unified bias-variance decomposition. In *International Conference on Machine Learning*, pages 231–238, 2000.

[27] F. Dong. Improving discrete AdaBoost for classification by randomization methods. *M. Phil. Thesis, University of Hong Kong*, 2016.

[28] C. Drummond and R. C. Holte. Explicitly representing expected cost: An alternative to ROC representation. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 198–207, 2000.

[29] C. Drummond and R. C. Holte. Cost curves: An improved method for visualizing classifier performance. *Machine Learning*, 65(1):95–130, 2006.

[30] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. Wiley, New York, 2 edition, 2001.

[31] N. U. Edakunni, G. Brown, and T. Kovacs. Boosting as a product of experts. *Uncertainty in Artificial Intelligence*, 2011.

[32] C. Elkan. The foundations of cost-sensitive learning. In *International Joint Conference on Artificial Intelligence*, 2001.

[33] W. Fan, S. J. Stolfo, J. Zhang, and P. K. Chan. AdaCost: Misclassification cost-sensitive boosting. In *International Conference on Machine Learning*, pages 97–105, 1999.

[34] M. Fauvel, J. Chanussot, and J. A. Benediktsson. Decision fusion for the classification of urban remote sensing images. *IEEE Transactions on Geoscience and Remote Sensing*, 44:2828–2838, 2006.

[35] T. Fawcett. An introduction to ROC analysis. *Pattern Recognition Letters*, 27(8):861–874, 2006.

[36] M. Fernández-Delgado, E. Cernadas, S. Barro, and D. Amorim. Do we need hundreds of classifiers to solve real world classification problems? *Journal of Machine Learning Research*, 15:3133–3181, 2014.

[37] P. A. Flach. The geometry of ROC space: understanding machine learning metrics through ROC isometrics. In *AAAI Conference on Artificial Intelligence*, pages 194–201, 2003.

[38] P. A. Flach. *Machine Learning: The Art and Science of Algorithms that Make Sense of Data*. Cambridge University Press, 2012.

[39] P. A. Flach. *Classifier Calibration*. Springer US, 8 2016.

[40] Y. Freund. An adaptive version of the boost by majority algorithm. In *Conference on Computational Learning Theory*, pages 102–113, 2000.

[41] Y. Freund and R. E. Schapire. Game theory, on-line prediction and boosting. In *Conference on Computational Learning Theory*, pages 325–332. ACM, 1996.

[42] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55 (1):119–139, 1997.

[43] Y. Freund and R. E. Schapire. Adaptive game playing using multiplicative weights. *Games and Economic Behavior*, 29:79–103, 1999.

[44] J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: a statistical view of boosting. *Annals of Statistics*, 28:337–407, 2000.

[45] J. H. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29:1189–1232, 2000.

[46] J. H. Friedman. Stochastic gradient boosting. *Computational Statistics and Data Analysis*, 38(4):367–378, 2002.

[47] Y. Ganjisaffar, R. Caruana, and C. V. Lopes. Bagging gradient-boosted trees for high precision, low variance ranking models. In *International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 85–94, New York, NY, USA, 2011.

[48] W. Gao and Z. H. Zhou. On the doubt about margin explanation of boosting. *arXiv:1009.3613v4*, 2012.

[49] S. Geman, E. Bienenstock, and R. Doursat. Neural networks and the bias/variance dilemma. *Neural Computing and Applications*, 4(1):1–58, 1992.

[50] J. Han, L. Shao, D. Xu, and J. Shotton. Enhanced computer vision with Microsoft Kinect sensor: A review. *IEEE Transactions on Cybernetics*, 43(5):1318–1334, 2013.

[51] D. J. Hand. Classifier technology and the illusion of progress. *Statistical Science*, 21(1):1–14, 2006.

[52] H. He and E. A. Garcia. Learning from imbalanced data. *IEEE Transactions on Knowledge and Data Engineering*, 21(9):1263–1284, 2009.

[53] J. Hernández-Orallo, P. A. Flach, and C. Ferri. Brier curves: a new cost-based visualisation of classifier performance. In *International Conference on Machine Learning*, pages 585–592, 2011.

[54] G. E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14:1771–1800, 2002.

[55] N. Japkowicz. Concept-learning in the presence of between-class and within-class imbalances. In *Conference of the Canadian Society on Computational Studies of Intelligence: Advances in Artificial Intelligence*, AI '01, pages 67–77, London, UK, 2001. Springer-Verlag.

[56] N. Japkowicz and S. Stephen. The class imbalance problem: A systematic study. *Intelligent Data Analysis*, 6(5):429–449, 2002.

[57] M. V. Joshi, R. C. Agarwal, and V. Kumar. Predicting rare classes: Can boosting make any weak learner strong? In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 297–306, New York, NY, USA, 2002. ACM.

[58] M. V. Joshi, V. Kumar, and R. C. Agarwal. Evaluating boosting algorithms to classify rare classes: Comparison and improvements, 2001.

[59] M. Kearns. Thoughts on hypothesis boosting. *Unpublished manuscript*, 1988.

[60] J. Kivinen and M. K. Warmuth. Boosting as entropy projection. In *Conference on Computational Learning Theory*, pages 134–144, New York, NY, USA, 1999. ACM.

[61] M. Kubat, R. C. Holte, and S. Matwin. Machine learning for the detection of oil spills in satellite radar images. *Machine Learning*, 30(2–3):195–215, 1998.

[62] M. Kull and P. Flach. Novel decompositions of proper scoring rules for classification: Score adjustment as precursor to calibration. In *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, pages 68–85. Springer, 2015.

[63] I. Landesa-Vázquez and J. L. Alba-Castro. Shedding light on the asymmetric learning capability of AdaBoost. *Pattern Recognition Letters*, 33 (3):247–255, 2012.

[64] I. Landesa-Vázquez and J. L. Alba-Castro. Double-base asymmetric AdaBoost. *Neurocomputing*, 118:101 – 114, 2013.

[65] I. Landesa-Vázquez and J. L. Alba-Castro. Revisiting AdaBoost for cost-sensitive classification. part i: Theoretical perspective. *arXiv:1507.04125v1*, 2015.

[66] I. Landesa-Vázquez and J. L. Alba-Castro. Revisiting AdaBoost for cost-sensitive classification. part ii: Empirical analysis. *arXiv:1507.04126v1*, 2015.

[67] G. Lebanon and J. D. Lafferty. Boosting and maximum likelihood for exponential models. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Neural Information Processing Systems*, pages 447–454. MIT Press, 2001.

[68] J. Leskovec and J. Shawe-Taylor. Linear programming boosting for uneven datasets. In *International Conference on Machine Learning*, pages 456–463, 2003.

[69] S. Z. Li, Z. Q. Zhang, H. Y. Shum, and H. J. Zhang. FloatBoost learning for classification. In S. Becker, S. Thrun, and K. Obermayer, editors, *Neural Information Processing Systems*, pages 993–1000. MIT Press, 2002.

[70] H. W. Lin and M. Tegmark. Why does deep and cheap learning work so well? *arXiv preprint arXiv:1608.08225*, 2016.

[71] Y. Lin, Y. Lee, and G. Wahba. Support vector machines for classification in nonstandard situations. *Machine Learning*, 46(1–3):191–202, 2002.

[72] P. M. Long and R. A. Servedio. Random classification noise defeats all convex potential boosters. *Machine Learning*, 78(3):287–304, 2010.

[73] P. Malacaria and F. Smeraldi. On AdaBoost and optimal betting strategies. In *International Conference in Data Mining*, pages 326–332, 2009.

[74] M. A. Maloof. Learning when data sets are imbalanced and when costs are unequal and unknown. In *International Conference on Machine Learning-2003 Workshop on Learning from Imbalanced Data Sets II*, 2003.

[75] D. D. Margineantu. Class probability estimation and cost-sensitive classification decisions. In *European Conference on Machine Learning*, pages 270–281, London, UK, 2002. Springer-Verlag.

[76] M. Markou and S. Singh. Novelty detection: a review–part 1: statistical approaches. *Signal Processing*, 83(12):2481–2497, 2003.

[77] S. Marsland. Novelty detection in learning systems. *Neural Computing Surveys*, 3(2):157–195, 2002.

[78] H. Masnadi-Shirazi and N. Vasconcelos. Asymmetric boosting. In *International Conference on Machine Learning*, 2007.

[79] H. Masnadi-Shirazi and N. Vasconcelos. Cost-sensitive boosting. *IEEE Transactions of Pattern Analysis and Machine Intelligence*, 33 (2):294–309, 2011.

[80] L. Mason, J. Baxter, P. Bartlett, and M. Frean. Boosting algorithms as gradient descent. *Neural Information Processing Systems*, pages 512–518, 2000.

[81] R. Meir and D. Parkes. On sex, evolution, and the multiplicative weights update algorithm. In *International Conference on Autonomous Agents and Multiagent Systems*, pages 929–937, 2015.

[82] J. Milgram, M. Cheriet, R. Sabourin, and École De Technologie Supérieure Montréal. One Against One or One Against All: Which one is better for handwriting recognition with SVMs. In *Proceedings of 10th International Workshop on Frontiers in Handwriting Recognition*, 2006.

[83] J. G. Moreno-Torres, T. Raeder, R. Alaiz-Rodríguez, N. V. Chawla, and F. Herrera. A unifying view on dataset shift in classification. *Pattern Recognition*, 45(1):521 – 530, 2012.

[84] A. H. Murphy. A new vector partition of the probability score. *Journal of Applied Meteorology*, 12(4):595–600, 1973.

[85] P. B. Nemenyi. Distribution-free multiple comparisons. *PhD Thesis, Princeton University*, 1963.

[86] A. Niculescu-Mizil and R. Caruana. Obtaining calibrated probabilities from boosting. In *Uncertainty in Artificial Intelligence*, 2005.

[87] N. Nikolaou and G. Brown. Calibrating AdaBoost for asymmetric learning. In *Multiple Classifier Systems*, pages 112–124, 2015.

[88] N. Nikolaou, N. Edakunni, M. Kull, P. Flach, and G. Brown. Cost-sensitive boosting algorithms: Do we really need them? *Machine Learning*, 104(2):359–384, 2016.

[89] N. C. Oza. Online bagging and boosting. In *IEEE International Conference on Systems, Man and Cybernetics*, volume 3, pages 2340–2345, 2005.

[90] N. C. Oza and K. Tumer. Classifier ensembles: Select real-world applications. *Information Fusion*, 9(1):4–20, 2008.

[91] M. A. F. Pimentel, D. A. Clifton, L. Clifton, and L. Tarassenko. A review of novelty detection. *Signal Processing*, 99:215–249, 2014.

[92] J. C. Platt. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In *Advances in Large Margin Classifiers*, pages 61–74. MIT Press, 1999.

[93] F. Provost and T. Fawcett. Analysis and visualization of classifier performance: Comparison under imprecise class and cost distributions. In *International Conference on Knowledge Discovery and Data Mining*, pages 43–48. AAAI Press, 1997.

[94] J. R. Quinlan. Bagging, boosting, and C4.5. In *National Conference on AI - Vol. 1*, pages 725–730, 1996.

[95] J. Quionero-Candela, M. Sugiyama, A. Schwaighofer, and N. D. Lawrence. *Dataset Shift in Machine Learning*. MIT Press, 2009.

[96] L. Reyzin and R. E. Schapire. How boosting the margin can also boost classifier complexity. In *International Conference on Machine Learning*, pages 753–760, 2006.

[97] R. Rifkin and A. Klautau. In defense of one-vs-all classification. *Journal of Machine Learning Research*, 5:101–141, 2004.

[98] T. Robertson, F.T. Wright, and R. Dykstra. *Order Restricted Statistical Inference*. Probability and Statistics Series. Wiley, 1988.

[99] S. Rosset, J. Zhu, T. Hastie, and R. Schapire. Boosting as a regularized path to a maximum margin classifier. *Journal of Machine Learning Research*, 5:941–973, 2004.

[100] C. Rudin, I. Daubechies, R. E. Schapire, and D. Ron. The dynamics of AdaBoost: Cyclic behavior and convergence of margins. *Journal of Machine Learning Research*, 5:1557–1595, 2004.

[101] M. J. Saberian and N. Vasconcelos. Learning optimal embedded cascades. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34, 2012.

[102] R. E. Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197–227, 1990.

[103] R. E. Schapire. Using output codes to boost multiclass learning problems. In *International Conference on Machine Learning*, 1997.

[104] R. E. Schapire. The boosting approach to machine learning: An overview. In MSRI Workshop on Nonlinear Estimation and Classification, Berkeley, CA, USA, 2001.

[105] R. E. Schapire. Explaining AdaBoost. In *Empirical inference*, pages 37–52. Springer, 2013.

[106] R. E. Schapire, Y. Freund, P. Bartlett, and W. S. Lee. Boosting the margin: a new explanation for the effectiveness of voting methods. *The Annals of Statistics*, 26(5):1651–1686, 1998.

[107] R. E. Schapire and Y. Singer. Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, 37 (3):297–336, 1999.

[108] W. J. Scheirer, A. Rocha, R. J. Micheals, and T. E. Boult. Meta-recognition: The theory and practice of recognition score analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(8):1689–1695, 2011.

[109] K. Sechidis, N. Nikolaou, and G. Brown. Information theoretic feature selection in multi-label data through composite likelihood. In *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*, pages 143–152. Springer, 2014.

[110] R. A. Servedio. Smooth boosting and learning with malicious noise. In D. P. Helmbold and B. Williamson, editors, *Conference On Learning Theory*, volume 2111, pages 473–489. Springer, 2001.

[111] R. Soleymani, E. Granger, and G. Fumera. Loss factors for learning boosting ensembles from imbalanced data. In *International Conference on Pattern Recognition*, 2016.

[112] Y. Sun, M. S. Kamel, A. K. C. Wong, and Y. Wang. Cost-sensitive boosting for classification of imbalanced data. *Pattern Recognition*, 40 (12):3358–3378, 2007.

[113] Y. Sun, A. K. C. Wong, and Y. Wang. Parameter inference of cost-sensitive boosting algorithms. In *Machine Learning and Data Mining in Pattern Recognition*, pages 21–30, 2005.

[114] P. N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining, (First Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005.

[115] D. M. J. Tax and R. P. W. Duin. Using two-class classifiers for multiclass classification. In *International Conference on Pattern Recognition*, volume 2, pages 124–127, 2002.

[116] M. Telgarsky. Margins, shrinkage, and boosting. In *International Conference on Machine Learning*, pages 307–315, 2013.

[117] K. M. Ting. A comparative study of cost-sensitive boosting algorithms. In *International Conference on Machine Learning*, pages 983–990, 2000.

[118] K. M. Ting and Z. Zheng. Boosting cost-sensitive trees. In *Discovery Science: First International Conference*, pages 244–255, 1998.

[119] A. Töscher, M. Jahrer, and R. M. Bell. The BigChaos solution to the Netflix grand prize, 2009.

[120] J. Tukey. *Exploratory Data Analysis*. Addison-Wesley, Reading, MA, 1977.

[121] V. Vapnik and R. Izmailov. Learning using privileged information: Similarity control and knowledge transfer. *Journal of Machine Learning Research*, 16:2023–2049, 2015.

[122] V. Vapnik and A. Vashist. A new learning paradigm: Learning using privileged information. *Neural Networks*, 22(5):544–557, 2009.

[123] P. Viola and M. Jones. Fast and robust classification using asymmetric AdaBoost and a detector cascade. In *Neural Information Processing Systems*, 2002.

[124] P. Viola and M. J. Jones. Robust real-time face detection. *International Journal of Computer Vision*, 57(2):137–154, 2004.

[125] L. Wang, M. Sugiyama, Z. Jing, C. Yang, Z. H. Zhou, and J. Feng. A refined margin analysis for boosting algorithms via equilibrium margin. *Journal of Machine Learning Research*, 12:1835–1863, 2011.

[126] Z. Wang, C. Fang, and X. Ding. Asymmetric real AdaBoost. In *International Conference on Pattern Recognition*, pages 1–4, 2008.

[127] M. K. Warmuth, K. A. Glocer, and G. Rätsch. Boosting algorithms for maximizing the soft margin. In J. C. Platt, D. Koller, Y. Singer, and S. T. Roweis, editors, *Neural Information Processing Systems*. MIT Press, 2007.

[128] M. K. Warmuth, J. Liao, and G. Rätsch. Totally corrective boosting algorithms that maximize the margin. In W. W. Cohen and A. Moore, editors, *International Conference on Machine Learning*, volume 148 of *ACM International Conference Proceeding Series*, pages 1001–1008. ACM, 2006.

[129] R. Weber. On the Gittins index for multiarmed bandits. *The Annals of Applied Probability*, 2(4):1024–1033, 1992.

[130] D. H. Wolpert. The lack of a priori distinctions between learning algorithms. *Neural Computation*, 8(7):1341–1390, October 1996.

[131] X. Wu, V. Kumar, J. R. Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. Ng, B. Liu, S. Y. Philip, et al. Top 10 algorithms in data mining. *Knowledge and information systems*, 14(1):1–37, 2008.

[132] B. Zadrozny and C. Elkan. Obtaining calibrated probability estimates from decision trees and naive Bayesian classifiers. In *International Conference on Machine Learning*, pages 609–616, 2001.

[133] B. Zadrozny and C. Elkan. Transforming classifier scores into accurate multiclass probability estimates, 2002.

[134] P. Zhao and B. Yu. Stagewise Lasso. *Journal of Machine Learning Research*, 8:2701–2726, 2007.

[135] J. Zhu, H. Zou, S. Rosset, and T. Hastie. Multi-class AdaBoost. In *Statistics and Its Interface*, 2009.