Contributions to machine learning: the unsupervised, the supervised, and the Bayesian

Balazs Kégl

Orsay
n° d'ordre

LAL 11-221
septembre 2011

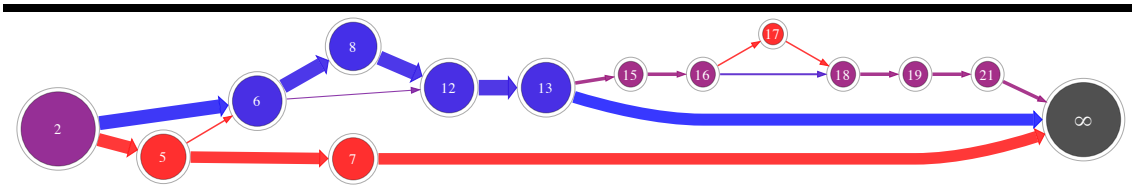
Contributions to machine learning: the unsupervised, the supervised, and the Bayesian

Habilitation à diriger des recherches

Balázs Kégl

Laboratoire de l'accélérateur linéaire / Laboratoire de recherche en informatique
IN2P3/CNRS & Université Paris-Sud, France

Soutenue le 28 septembre 2011 devant la comission d'examen
MM. Y. Grandvalet (rapporteur)
E. Moulines (rapporteur)
M. Sebag (examinateur)
A. Stocchi (président)
N. Vayatis (examinateur)
D. Veberič (rapporteur)
G. Wormser (examinateur)



Contents

Chapter 1

Introduction

Life is a random walk [1]. At the same time, we like pretending that there is a master narrative with us in the leading role. I'm not sure if I have such a script. Surely there is a drift of going from theory to algorithms and going from finding the nail for my hammers to finding the matching hammers to the nails, but other than that there is no grandiose reason why I did research on the specific subjects assembled in this thesis. I also have some vague scientific feelings about which way to go if we want to understand and mimic human intelligence, but I don't kid myself: we are far, and the chances that my findings would be part of the big theory are slim.

So then what *does* drive me, what does explain this seemingly random collection of works? A big part of it is concrete problems. I like solving puzzles, and I like finding algorithms that solve problems. There is also mathematical beauty, although I must say that I believe less and less that something is right for a problem just because the solution is beautiful. In machine learning too often we do invent the problems we solve; we inherited this from pure mathematics. Sometimes it happens that, as a side effect, we invent something useful, but that is often a result of sheer luck. Today I am fortunate to work in a research field (we call "application") where there are plenty of unsolved data mining problems. Sometimes they are not easy to isolate, but once they are there, they "only" have to be solved, they don't have to be invented. This thesis contains two such concrete problems and our solutions to them. To be honest, originally I thought about presenting only these two very fresh results, but it turned out that they would not fill the fifty page lower limit, so I added a lot of other research we have been doing, recently and less recently.

Chapter 2 is a short summary of my early research which I did during and right after my Ph.D. It is not an area in which I have been active recently; I included it for completeness. My work on principal curves is clearly my most significant research contribution in terms of impact. The theoretical model motivated the development of several new results in unsupervised learning theory (e.g., probabilistic principal surfaces, regularized principal manifolds, and principal curves with bounded turn). Variants of my polygonal-line algorithm have been used in a wide variety of applications ranging from data visualization and curve detection to clustering and feature extraction.

Chapter 3 is the thickest part of the thesis. Most of it is about a multivariate classification algorithm called AdaBoost. AdaBoost is one of those rare examples where mathematical beauty meets practical usefulness. I first used it for a prediction problem, I believe for credit card fraud, when some of the database entries were missing. The method did not work out of the box, so I invented a beautiful extension. It worked, I was ready to publish it when I discovered it was published two years earlier. This happened two more times before I actually found something new. The chapter contains most of the research we have done on AdaBoost since then. It also contains a formal but gentle introduction to supervised learning that I wrote knowing that two of the committee members are not machine learning experts. At the end of the chapter (Section 3.6) I describe the first method I mentioned earlier: a new algorithm motivated by trigger design in experimental physics.

Chapter 4 summarizes some of the research I have been doing in the Auger collaboration.

Naturally, this chapter is more oriented towards experimental physics, but, as in Chapter 3, I included an introduction readable for a non-expert. The rest is, I have to admit, will be difficult to read for both machine learners and physicists: it is about physics but written in a formal language we use in machine learning and statistics. The only exception is Section 4.3 which contains the second example of a physics-motivated technical result: I describe here a new adaptive Monte Carlo Markov chain algorithm for mixture models directly motivated by the signal processing challenges of Auger.

To ease the reading of the thesis for jury members of different backgrounds, Figure 1.1 provides a map with (clickable) section numbers on the plane defined by the dichotomies of technical/general and physics/machine learning.

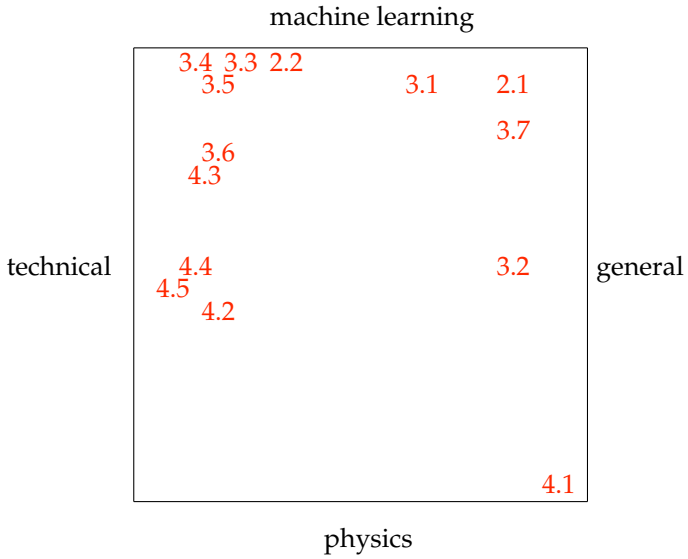


Figure 1.1: A clickable map of sections.

Acknowledgments

First I would like to mention how fortunate I feel to have a research position in the CNRS. Looking around, it is hard to find any other institute in the world where one can do free research with so few obligations *and* make a decent living. It is one of those places where, in my experience, the 80/20 rule[1] does not apply. In fact it is very rare that I have to do something I do not want to do.[2]. So first I would like to thank to Guy Wormser for believing in me and for creating this unique position at LAL, and Michèle Sebag and Cécile Germain-Renaud for having the idea of creating this interdisciplinary position in the first place, for helping to keep my local ties to the machine learning community, and for making my integration into the quite complex French research system as smooth as possible.

Most of the research described in this thesis is a result of scientific collaborations (which is also the reason why I will use the pronoun "we" throughout this thesis). I owe a great deal for launching my research carrier to three people: Adam Krzyżak, Tamás Linder, and Gábor Lugosi. My recent results could not have been achieved without the help of members of the AppStat team (Rémi Bardenet, Djalel Benbouzid, Róbert Busa-Fekete, François-David Collin, and Diego Garcia Gámez). On the physics side my infinite gratitude goes to Marcel Urban: he taught me everything I know about experimental physics. I know that I will never be a physicist, it is not my goal in any case, but what Marcel taught me is most of the time enough to camouflage myself within the

[1] http://ml.typepad.com/machine_learning_thoughts/2007/02/happiness_of_a__1.html
[2] Not considering writing this thesis of course. ☺

Auger collaboration. Another special thanks goes to Darko Veberič who was the first person in Auger who understood my machine learning gibberish, and who helped me a lot to translate my writing into the language of physicists. His early encouragements are one of the reasons why I persevered in Auger. I would also like to thank to all the past and present members of the LAL Auger team (Jean-Noel Albert, René Bilhaut, Alain Cordier, Sylvie Dagoret-Campagne, Olivier Dalifard, Paul Eschstruth, Xavier Garrido, Karim Louedec, and Delphine Monnier-Ragaigne) for their warm welcome and patience in "training" me.

Finally I would like to thank the French National Agency (ANR) for their financial support that helped creating the AppStat team and the fruitful collaboration with the Computer Science Laboratory (LRI) and Telecom ParisTech.

Chapter 2

Unsupervised learning

The goal of unsupervised learning is to "make sense" of high-dimensional data sets. Density estimation is the heavy machinery that can be used to formalize most of the unsupervised problems, but it is often not necessary to go through a formal probabilistic model to solve a particular problem. Two big sub-families of unsupervised learning are clustering and dimensionality reduction. In clustering the goal is to partition the data set into a finite number of groups with high "similarity" within groups and high "distance" between groups. The difficulty is that, unlike in supervised classification, there is no clear semantics of what the clusters are, and, above all, no feedback (label) whether we found the right cluster for a given data point (which explains the adjective "unsupervised"). In dimensionality reduction we take high dimensional observations and try to represent them with a small number of parameters without losing information. Both problems come with a natural trade-off between information preservation and compression (e.g., number of clusters vs. average distortion, number of dimensions vs. lost variance). Most of the time there is no universal solution of this trade-off. The most popular algorithms usually come with a complexity hyperparameter which the user can set to settle the solution into an acceptable compromise.

The two basic unsupervised methods are *Principal Component Analysis* (PCA) [2, 3, 4] and *vector quantization* with the *k-means* algorithm [5, 6]. They are basically the algorithmic prototypes of many more sophisticated methods. Some of the first non-linear techniques, Kohonen's *Self-Organizing Maps* and [7] Kramer's *Autoassociative Neural Networks* [8], are direct extensions of k-means and PCA, respectively. The field was revolutionized in the early 2000s by the appearance of two *neighborhood-based* dimensionality reduction methods, Roweis et al.'s *Local Linear Embedding* (LLE) [9] and Tenenbaum et al.'s ISOMAP [10]. These two seminal papers started a whole new research domain on diffusion kernels and the graph Laplacian [11, 12]. Although the main wave of hype has somewhat dwindled by now, unsupervised learning and feature induction remains a central topic in *deep learning*; some of the devotees of the subject go as far as portraying unsupervised learning as the holy grail for attaining human intelligence [13, 14].

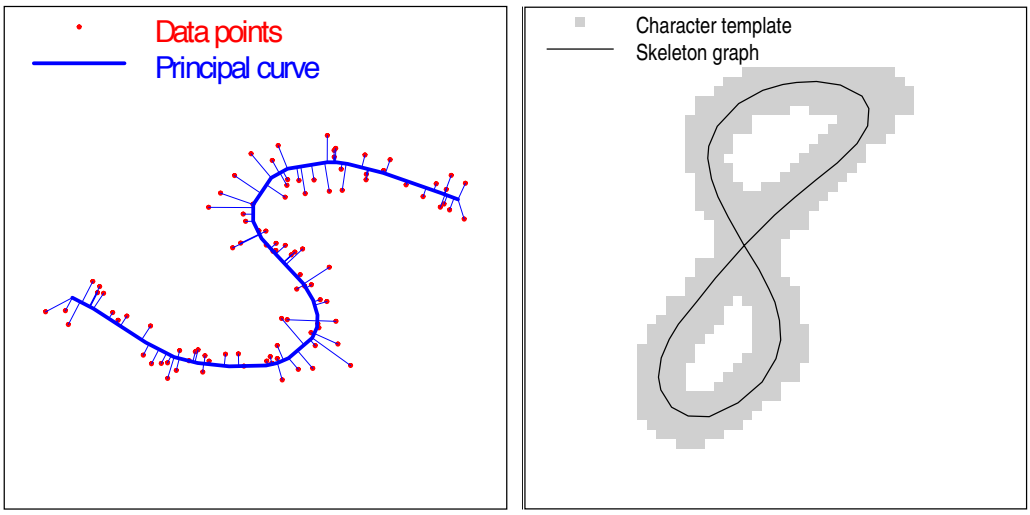
In this chapter we will summarize three of our contributions to unsupervised learning. In Section 2.1 we review our work on principal curves [15, 16]. Principal curves can be considered either as an extreme dimensionality reduction method where we project the data onto a 1D manifold, or as a clustering technique where clusters are adorned by a linear or a graph structure. Principal curves are not as general as LLE or ISOMAP, mainly because of the 1D constraint, but they scale much better than LLE and ISOMAP in the number of data points and they also handle additive noise better. Our approach had a considerable impact both in applications [17, 18, 19, 20, 21, 22, 23, 24] and inciting various extensions [25, 26, 27, 28, 29, 30]. We summarize some of these works in Section 2.1.1.

One of the basic questions one can ask before running any data analysis method is the number of degrees of freedom of the data that can be formalized as its *intrinsic dimensionality*. Knowing this value is a first step towards understanding the data. On a more pragmatic level, the intrinsic dimensionality is usually a design parameter that has to be input into nonlinear dimensionality

reduction techniques such as LLE and ISOMAP. Section 2.2 outlines our technique [31] to estimate the intrinsic dimensionality using *packing numbers*, and summarizes its impact.

2.1 Principal curves

Principal curves are one of the nonlinear extensions of *Principal Component Analysis*. They give a one-dimensional non-linear summary of a high-dimensional data set and also serve as an efficient feature extraction tool. Principal curves were originally defined by Hastie [32] and Hastie and Stuetzle [33] (hereafter HS) to formally capture the notion of a smooth curve passing through the "middle" of a d-dimensional probability distribution or data cloud (Figure 2.1(a)).



(a) Principal curve (b) Principal graph

Figure 2.1: A smooth curves passing through the "middle" of (a) a data cloud and (b) a set of pixels.

The original HS definition of principal curves is based on the concept of *self-consistency*. Intuitively, self-consistency means that each point of the curve is the average of all points that project there. Based on the self-consistency property, HS developed a theoretical and a practical algorithm for constructing principal curves of distributions and data sets, respectively.

There was an unsatisfactory aspect of the original definition of principal curves: Hastie and Stuetzle could not prove the existence of the theoretically defined object even for very simple data distributions. This made it difficult to theoretically analyze any learning scheme for principal curves. In [15] we proposed a new definition of principal curves, and proved their existence for a large class of data distributions. Then we described a theoretical learning scheme, proved its statistical consistency, and obtained convergence rate. These are also one of the first theoretical results in any non-linear unsupervised learning models. The theoretical algorithm also inspired the design of a new practical algorithm to estimate principal curves of data sets. The *polygonal line algorithm* beat previous methods both in terms of performance and computational complexity. In [16] we proposed an algorithm to find piecewise linear skeletons of hand-written characters by using principal curves based on the similarity between principal curves and the medial axis (Figure 2.1(b)). The algorithm substantially improved the pixelwise skeleton obtained by traditional thinning methods .

2.1.1 Impact

In this section we summarize subsequent advances on principal curves on both the theoretical and algorithmic side, and describe some of the applications. Many of the 300+ citations of [15, 16] are simple references to related work, but there is about half a dozen applications where a variant of the polygonal line algorithm was actually used in practice.

There was a slight discrepancy in [15] between the theoretical model and the practical algorithm: the theoretical principle curve definition used a constraint on the *length* of the curve whereas the polygonal line algorithm penalized small *angle* between segments. [25] elegantly solved this problem by proving existence and convergence for curves with bounded *turn* (integrated curvature). A second, very recent work [30] improved the convergence rate from $O(n^{-1/3})$ to $O(n^{-1/2})$ by replacing our covering number approach with a technique based on Rademacher averages [34, 35]. Based on the slope heuristics introduced by [36], [30] also redesigned the heuristic model selection procedure of the polygonal line algorithm.

There were two main improvements on the algorithmic side of estimating principle curves of data sets. The initialization procedure of the polygonal line algorithm uses the first principal component, and then refines the principal curve by iteratively adding more and more vertices and segments. When the data is close to self intersecting (such a the spiral toy data set), this procedure can easily get stuck in bad local minima. [26] and [27] proposed to overcome this problem by designing the principle curve in a bottom-up fashion. Although the algorithms have never been connected, we think that the best practical solution would be to initialize the principle curve using the method of [26], and then use the polygonal line algorithm to finish the design. A second major algorithmic advance is [28]'s elastic net that uses a similar penalized "energy function" to define principal curves. The great advantage of [28] over the polygonal line algorithm is that elastic nets can naturally be extended to higher dimensional manifolds. Finally, although not directly related to our work, we would like to mention [29]'s approach that defines principal curves and graphs based on local properties of the data distribution. The closest relative to their definition is [37]'s principal curves, but the most important contribution of [29] is an efficient practical algorithm based on the mean shift method.

Although these are not "real" applications *per se*, [22] demonstrated that the polygonal line algorithm can be used for clustering around curves, and [19] shows how to use it for feature extraction. The first concrete applications of the polygonal line algorithm were in traffic stream modeling [18], in analyzing temporal data [17], and in fault detection and process monitoring [20]. Image processing is one of the main application areas of principal curves in general. [21] used a simplified version of the polygonal line algorithm for solar filament detection in images, whereas [23] used it for extracting vessels in a 3D angiography. Arguably the most important application of the algorithm is in experimental physics. The goal of the ICARUS experiment [24] is to study cosmic rays, neutrino oscillation, and proton decay using a 760 ton liquid argon time projection chamber installed in the Gran Sasso underground National Laboratory in Italy. The polygonal line algorithm is in the official pipeline of the trajectory reconstruction of particles crossing the chamber.

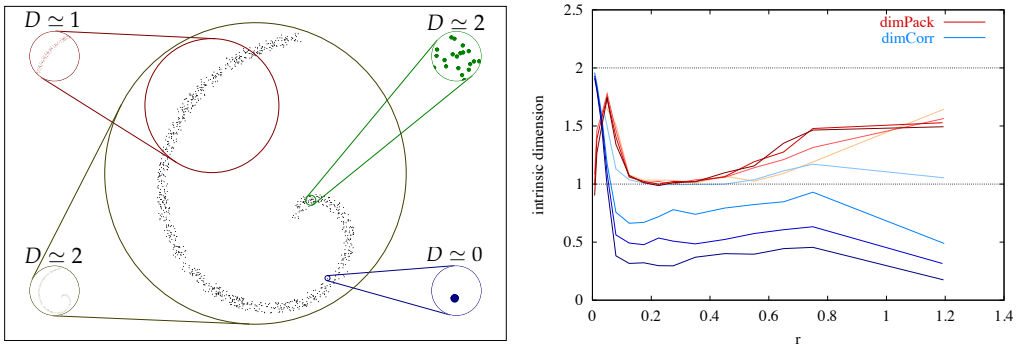
2.2 Intrinsic dimension estimation

High-dimensional data sets have several unfortunate properties that make them hard to analyze. The phenomenon that the computational and statistical efficiency of statistical techniques degrade rapidly with the dimension is often referred to as the "curse of dimensionality". One particular characteristic of high-dimensional spaces is that, as the volumes of constant diameter neighborhoods become large, exponentially many points are needed for reliable density estimation. Another important problem is that as the data dimension grows, sophisticated data structures constructed to speed up nearest neighbor searches rapidly become inefficient.

Fortunately, most meaningful, real life data do not uniformly fill the spaces in which they are represented. Rather, the data distributions are observed to concentrate to non-linear mani-

folds of low *intrinsic dimension*. Several methods have been developed to find low-dimensional representations of high-dimensional data, including classical methods such as Principal Component Analysis (PCA), Self-Organizing Maps (SOM) [38], and Multidimensional Scaling (MDS) [39], and, more modern techniques such as Local Linear Embedding (LLE) [9] and the ISOMAP algorithm [10].

There are two principal areas where a good estimate of the intrinsic dimension can be useful. First, the estimate can be used to set input parameters of dimension reduction algorithms. Certain methods (e.g., LLE and the ISOMAP algorithm) also require a scale parameter that determines the size of the local neighborhoods used in the algorithms. In this case, it is useful if the dimension estimate is provided as a function of the scale (see Figure 2.2(a) for an intuitive example where the intrinsic dimension of the data depends on the resolution). Nearest neighbor searching algorithms can also profit from a good dimension estimate. The complexity of search data structures (e.g., kd-trees and R-trees) increase exponentially with the dimension, and these methods become inefficient if the dimension is more than about 20. Nevertheless, it was shown by [40] that the complexity increases with the intrinsic dimension of the data rather then with the dimension of the embedding space.



(a) Intrinsic dimension at different resolutions

(b) Estimated dimension

Figure 2.2: (a) At very small scale the data looks zero-dimensional. If the scale is comparable to the noise level, the data fills the embedding space. At the "right" scale the data looks one-dimensional. At very large scale the embedding dimension dominates again. (b) The scale dependent packing dimension $\widehat{D}_{\text{PACK}}(r)$ and correlation dimension $\widehat{D}_{\text{CORR}}(r)$. The lighter the curve the larger p is in (2.2) (that is, the more uneven the distribution is on the manifold).

In [31] we presented a novel method for intrinsic dimension estimation. The estimate is based on geometric properties of the data, and requires no parameters to set. Experimental results on both artificial and real data show that the algorithm is able to capture the scale dependence of the intrinsic dimension. The main advantage of the method over existing techniques is its robustness in terms of the generating distribution.

Formally, the scale-dependent *packing dimension* is

$$\widehat{D}_{\text{pack}}(r_1, r_2) = -\frac{\log M(r_2) - \log M(r_1)}{\log r_2 - \log r_1}, \tag{2.1}$$

where $M(r)$ is the *r-packing number* of a set $\mathcal{S} \subset \mathcal{X}$, defined as the maximum cardinality of an r-separated subset of $\mathcal{S}$ (a set $\mathcal{V} \subset \mathcal{X}$ is said to be *r-separated* if $d(x, y) \geq r$ for all distinct $x, y \in \mathcal{V}$). Finding $M(r)$ for a data set $\mathcal{S}_n = \{x_1, \ldots, x_n\}$ is equivalent to finding the cardinality of a maximum independent vertex set $\text{MI}(G_r)$ of the graph $G_r(V, E)$ with vertex set $V = \mathcal{S}_n$ and edge set $E = \{(x_i, x_j) | d(x_i, x_j) < r\}$. This problem is known to be NP-hard. Nevertheless, the simple greedy algorithm of Figure 2.3 works will in practice. Given a data set $\mathcal{S}_n$, we start with an empty set of centers $\mathcal{C}$, and in an iteration over $\mathcal{S}_n$ we add to $\mathcal{C}$ data points that are at a distance

of at least r from all the centers in $\mathcal{C}$. The estimate $\widehat{M}(r)$ is the cardinality of $\mathcal{C}$ after every point in $\mathcal{S}_n$ has been visited.

PACKINGDIMENSION$(\mathcal{S}_n, r_1, r_2, \epsilon)$

for $\ell \leftarrow 1$ **to** ∞ **do**
  Permute $\mathcal{S}_n$ randomly
  for $k \leftarrow 1$ **to** 2 **do**
    $\mathcal{C} \leftarrow \emptyset$
    for $i \leftarrow 1$ **to** n **do**
      for $j \leftarrow 1$ **to** $|\mathcal{C}|$ **do**
        if $d\big(\mathcal{S}_n[i], \mathcal{C}[j]\big) < r_k$ **then**
          $j \leftarrow n+1$
      if $j < n+1$ **then**
        $\mathcal{C} \leftarrow \mathcal{C} \cup \{\mathcal{S}_n[i]\}$
    $\widehat{L}_k[\ell] = \log |\mathcal{C}|$
  $\widehat{D}_{\text{PACK}} = -\dfrac{\mu(\widehat{L}_2) - \mu(\widehat{L}_1)}{\log r_2 - \log r_1}$
  if $\ell > 10$ **and** $1.65 \dfrac{\sqrt{\sigma^2(\widehat{L}_1) + \sigma^2(\widehat{L}_2)}}{\sqrt{\ell}(\log r_2 - \log r_1)} < \widehat{D}_{\text{PACK}}(1-\epsilon)/2$ **then**
    return $\widehat{D}_{\text{PACK}}$

Figure 2.3: The algorithm returns the packing dimension estimate $\widehat{D}_{\text{PACK}}(r_1, r_2)$ of a data set $\mathcal{S}_n$ with ϵ accuracy nine times out of ten.

The procedure is designed to produce an r-packing but certainly underestimates the packing number of the manifold, first, because we are using a finite sample, and second, because in general $\widehat{M}(r) < M(r)$. Nevertheless, we can still obtain a good estimate for $\widehat{D}_{\text{PACK}}$ by using $\widehat{M}(r)$ in the place of $M(r)$ in (2.1). To see why, observe that, for a good estimate for $\widehat{D}_{\text{PACK}}$, it is enough if we can estimate $M(r)$ with a constant multiplicative bias independent of r. Although we have no formal proof that the bias of $\widehat{M}(r)$ does not change with r, the simple greedy procedure described above seems to work well in practice.

Even though the bias of $\widehat{M}(r)$ does not affect the estimation of $\widehat{D}_{\text{PACK}}$ as long as it does not change with r, the variance of $\widehat{M}(r)$ can distort the dimension estimate. The main source of the variance is the dependence of $\widehat{M}(r)$ on the order of the data points in which they are visited. To eliminate this variance, we repeat the procedure several times on random permutations of the data, and compute the estimate $\widehat{D}_{\text{PACK}}$ by using the average of the logarithms of the packing numbers. The number of repetitions depends on r_1, r_2, and a preset parameter that determines the accuracy of the final estimate (set to 99% in all experiments).

To test robustness, we used a two-sided multivariate power distribution with density

$$p(\mathbf{x}) = I_{\{\mathbf{x} \in [-1,1]^d\}} \left(\frac{p}{2}\right)^d \prod_{i=1}^{d} \left(1 - |x^{(i)}|\right)^{p-1} \tag{2.2}$$

with different exponents p to generate uniform ($p = 1$) and non-uniform data sets on the spiral manifold. Figure 2.2(b) shows that the estimate is clearly better and more robust than the most popupar correlation dimension estimate $\widehat{D}_{\text{CORR}}$ [41, 42, 43]. More details on the algorithm and more experiments can be found in [31].

Chapter 3

Supervised learning

In this chapter we give an extended synopsis of our contributions to the vast area of supervised learning. Section 3.1 is a crash course into supervised learning, concentrating mainly on binary classification. We describe the formal setup, explain some of the principles behind algorithmic design, and provide details of the three most influential binary classification algorithms. Section 3.2 concretizes the abstract concepts of Section 3.1 through applications. We explain how learning fits into the pattern recognition pipeline and describe the different problem types and solutions through real-life applications we have worked on in the last ten years. An extended subsection on trigger design in experimental physics motivates one of our contributions that we will detail in a separate section (Section 3.6).

Section 3.3 is devoted to multi-class classification in general and to the ADABOOST.MH algorithm in particular. The section is based on the seminal work on Freund, Schapire, and Singer [44, 45], although we give more details on some of the important practical issues. One such issue is efficient multi-class base learning which we explain in Section 3.4. Some of the algorithms (INDICATORBASE and PRODUCTBASE) come from our contribution [46] while others (STUMPBASE and TREEBASE) are variants of standard learning algorithms. Although they are not new, to our knowledge, details of their multi-class versions have never been published.

Sections 3.5 and 3.6 are preprints of two of our most recent results. In Section 3.5 we show how to use ADABOOST.MH in a simple pointwise approach for web page ranking. The most significant methodological results of this contribution are 1) a diverse output calibration step that allows to convert algorithmic tools designed for one problem (e.g., classification) to solve seemingly different tasks (e.g., ranking), and 2) an ensemble technique aligned with a recent paradigm change of going from hyperparameter optimization towards averaging of diverse models. In Section 3.6 we combine control with learning. In particular, we show that adding a reinforcement-learning-based postprocessing step to standard ensemble methods can significantly reduce the statistical and computational complexity of the final classifier without harming its performance. The development of the method was directly motivated by object detection and trigger design, but the resulting algorithm goes beyond the standard cascade architecture usually used to solve these problems.

To finish the chapter, in Section 3.7 we briefly summarize some of our most significant contributions to supervised learning, other than those explained in previous sections.

3.1 Introduction

3.1.1 The supervised learning setup

The goal of supervised learning is to infer a function $g : \mathcal{X} \to \mathcal{L}$ from a data set

$$\mathcal{D} = \{(\mathbf{x}_1, \ell_1), \ldots, (\mathbf{x}_n, \ell_n)\} \in (\mathcal{X} \times \mathcal{L})^n$$

comprised of pairs of *observations* and *labels*. The elements $x^{(j)}$ of d-dimensional *feature* or *observation* vector $\mathbf{x}$ are either real numbers or they come from an unordered set of cardinality M_j. In this latter case, without loss of generality, we will assume that $x^{(j)} \in \mathcal{I}^{(j)} = \{1, \dots, M^{(j)}\}$.

When the label space $\mathcal{L}$ is real-valued, the problem is known as *regression*, whereas when the labels come from a finite set of classes, we are talking about *classification*. The quality of g is measured by an *error* or *loss* function $L\big(g, (\mathbf{x}, \ell)\big)$ that depends on the type of problem. In regression, the goal is to get $g(\mathbf{x})$ as close to ℓ as possible, so the loss grows with the difference between $g(\mathbf{x})$ and ℓ. The most widely used loss function in this case is the *quadratic* loss

$$L_2\big(g, (\mathbf{x}, \ell)\big) = \big(g(\mathbf{x}) - \ell\big)^2.$$

In classification, typically there is no distance defined between the classes, so all we can measure is whether or not g predicts correctly the class ℓ. The usual loss in this case is the *one-loss*

$$L_{\mathbb{I}}\big(g, (\mathbf{x}, \ell)\big) = \mathbb{I}\left\{g(\mathbf{x}) \neq \ell\right\}, \tag{3.1}$$

where the indicator function $\mathbb{I}\{A\}$ is 1 if its argument A is true and 0 otherwise.

The goal of learning algorithms is *not to memorize* $\mathcal{D}$, rather to *generalize* from it. Indeed, it is rather trivial to construct a function g that has zero loss on every sample point $(\mathbf{x}_i, \ell_i)$ by explicitly setting $g(\mathbf{x}_i) \triangleq \ell_i$ on all sample points from $\mathcal{D}$, and, say, $g(\mathbf{x}_i) \triangleq 0$ everywhere else. We obviously have not *learned* anything from $\mathcal{D}$, we have simply memorized it. It is also clear that this function has very little chance to perform well on points not in the set $\mathcal{D}$ (unless 0 is a good prediction everywhere in which case the problem itself is trivial). To formalize this intuitive notion of generalization, it is usually assumed that all sample points (including those in $\mathcal{D}$) are independent samples from a fixed but unknown distribution $\mathfrak{D}$, and the goal is to minimize the *expected loss* (a.k.a., *risk* or *error*) conditioned on the data set $\mathcal{D}$

$$R(g) = \mathbb{E}_{(\mathbf{x},\ell)\sim\mathfrak{D}}\left\{L\big(g, (\mathbf{x}, \ell)\big) \mid \mathcal{D}\right\}.$$[1]

Note that with this notation, the misclassification probability $P\big(g(\mathbf{x}) \neq \ell\big)$ is the risk generated by the one-loss

$$R_{\mathbb{I}}(g) = \mathbb{E}_{(\mathbf{x},\ell)\sim\mathfrak{D}}\left\{L_{\mathbb{I}}\big(g, (\mathbf{x}, \ell)\big)\right\} = P\big(g(\mathbf{x}) \neq \ell\big).$$[2] \tag{3.2}

As our imaginary example demonstrates it, finding a function g that minimizes the *empirical risk* (or *training error*)

$$\widehat{R}(g) = \frac{1}{n}\sum_{i=1}^{n} L\big(g, (\mathbf{x}_i, \ell_i)\big) \tag{3.3}$$

is not necessarily a good idea. Nevertheless, it turns out that the estimator

$$\widehat{g}^* = \arg\min_{g\in\mathcal{G}} \widehat{R}(g)$$

can have good properties both in theory and in practice if the function class $\mathcal{G}$ is not too "rich" so the functions in $\mathcal{G}$ are not too complex. How to control the complexity of $\mathcal{G}$ and how to find an optimal function in $\mathcal{G}$ computationally efficiently are the main subjects of the design of supervised learning algorithms.

Finding $\widehat{g}^*$ in $\mathcal{G}$ is the subject of algorithmic design. The process is called *training* or *learning*, and the data set $\mathcal{D}$ on which $\widehat{R}(g)$ is minimized is the *training set*. Of course, once $\mathcal{D}$ is used for finding $\widehat{g}^*$, it is tainted from a statistical point of view: $\widehat{R}(\widehat{g}^*)$ is no longer an unbiased estimator of $R(\widehat{g}^*)$. *Overfitting* is the term that describes the situation when the risk $R(\widehat{g}^*)$ is significantly larger than the training risk $\widehat{R}(\widehat{g}^*)$. To detect and to assess overfitting, $\widehat{g}^*$ has to be evaluated on a *test set* $\mathcal{D}'$ independent of $\mathcal{D}$.

[1] Since statistical consistency is not an issue in this chapter, we will omit the explicit conditioning on the training data set $\mathcal{D}$ from now on.

[2] For denoting the risk generated by a particular loss, we will use the same index: the risk generated by the loss $L_\bullet$ will be denoted by $R_\bullet$.

3.1.2 Complexity regularization and hyperparamater optimization

The simplest way to control the complexity of the function class $\mathcal{G}$ is to fix it either explicitly (e.g., by taking the set of all linear functions) or implicitly (e.g., the set of all functions that a neural network with N neurons and one hidden layer can represent). Another possibility is *complexity regularization*: instead of finding the best function in a fixed class $\mathcal{G}$, we define a nested sequence of function classes $\mathcal{G}_1 \subset \ldots \subset \mathcal{G}_K$, find the empirically best functions $\widehat{g}_1^*, \ldots, \widehat{g}_K^*$ in each class, and then minimize the regularized loss to obtain

$$k^* = \arg\min_k \widehat{R}(\widehat{g}_k^*) + \mathcal{C}(\mathcal{G}_k),$$

where $\mathcal{C}(\cdot)$ is an appropriate complexity measure of the function classes $\mathcal{G}_k$. For example, one can let k be the number of neurons in a neural network, and define $\mathcal{G}_k$ as the set of all functions that a neural network with k neurons and one hidden layer can represent. A similar but more practical procedure is to let $\mathcal{G}$ be a large (complex) class of functions, but *penalize* the complexity $C(g)$ of the function itself, and obtain

$$\widehat{g}_\beta^* = \arg\min_{g \in \mathcal{G}} \widehat{R}(g) + \beta C(g),$$

where β is a penalty coefficient that has to be tuned. The classical practices of *weight decay* or *early stopping* in neural networks fall in the category of penalized risk minimization.

Tuning the complexity parameters $\mathcal{C}(\cdot)$ or β is possible analytically if the goal is *asymptotic* (as training sample size $n \to \infty$) optimality, but in practice, it is almost always done by validation. In the simplest setup, we choose $\widehat{g}_k^*$ or $\widehat{g}_{\beta^*}^*$ by minimizing the empirical risk on a *validation* set $\mathcal{D}''$ which has to be independent of both the training set $\mathcal{D}$ and the test set $\mathcal{D}'$. This procedure is known as *hyperparameter optimization* in machine learning. Hyperparameter optimization is often done "manually", and it is one the most grueling tasks of graduate students. It is also a source of controversy: hidden (and, hopefully, most of the times inadvertent) use of the test set can lead to tainted results when comparing different algorithms. We have recently started to work on a stochastic optimization approach to make the procedure automatic [47].

3.1.3 Convex losses in binary classification

Finding either $\widehat{g}^*$, $\widehat{g}_k^*$, or $\widehat{g}_{\beta^*}^*$ can be algorithmically challenging. For example, finding a binary classifier $\widehat{g}^* \in \mathcal{G}$ that minimizes the training error $\widehat{R}_{\mathbb{I}}(g)$ is NP hard even in the simple case of $\mathcal{G}$ being the set of linear function [48]. One way to make the learning problem tractable is to relax the minimization of $\widehat{R}_{\mathbb{I}}(g)$ by defining *convex losses* that upper bound the one-loss $L_{\mathbb{I}}\big(g, (\mathbf{x}, \ell)\big)$ (3.1).

To formalize this, we need to introduce some notions used in binary classification. First, we change the symbolic output space of g from $\mathcal{L}$ to $\{-1, +1\}$ and denote the numerical label by $y \in \{-1, +1\}$. [3] Most of the learning algorithms do not directly output a $\{\pm 1\}$-valued classifier g, rather, they learn a real-valued *discriminant function* f, and obtain g by thresholding the output of $f(\mathbf{x})$ at 0, that is,

$$g(\mathbf{x}) = \begin{cases} +1 & \text{if } f(\mathbf{x}) > 0, \\ -1 & \text{if } f(\mathbf{x}) \leq 0. \end{cases}$$

Using the real-valued output of the discriminant function, the classifier can be evaluated on a finer scale by defining the (*functional*) *margin*

$$\rho = \rho\big(f, (\mathbf{x}, y)\big) = f(\mathbf{x})y.$$

With this notation, the misclassification indicator (one loss) of a discriminant function f on a sample point $(\mathbf{x}, y)$ can be defined as a function of the margin ρ by

$$L_{\mathbb{I}}\big(f, (\mathbf{x}, y)\big) = L_{\mathbb{I}}\Big(\rho\big(f, (\mathbf{x}, y)\big)\Big) = L_{\mathbb{I}}(\rho) = \mathbb{I}\{\rho < 0\},$$

[3] The reason of the distinction between ℓ and y will be clear in Section 3.3.

but the magnitude of ρ also indicates the confidence or the robustness of the classification. Indeed, the combination of a large ρ with a Lipschitz-type (slope) penalty on f means that the *geometric* margin $\rho^{(g)}$ is also large: $\mathbf{x}$ is far from the decision border consisting of the set of points for which $f(\mathbf{x}) = 0$ (Figure 3.1).

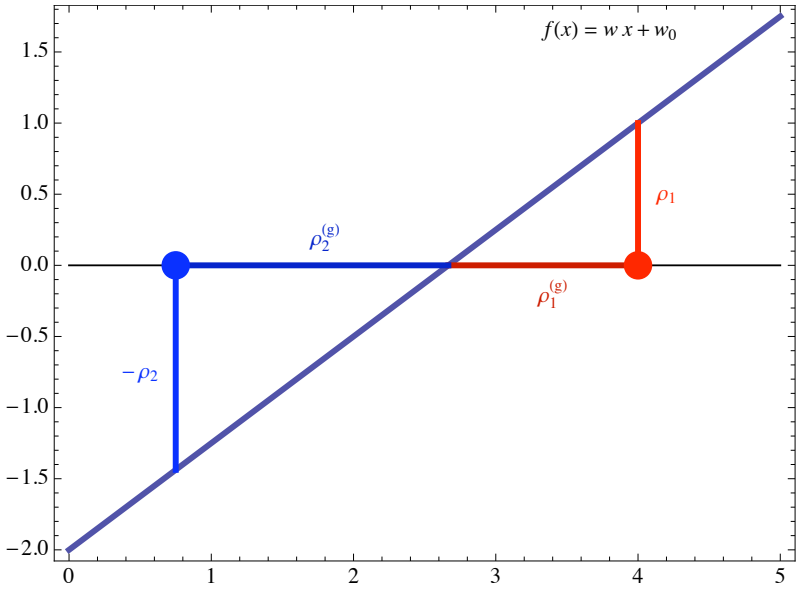


Figure 3.1: Geometric and functional margins. The labels of the red and blue points are $y = +1$ and $y = -1$, respectively. An upper bound ω on the slope w and a functional margin ρ implies a geometric margin (distance from the point $\mathbf{x} : f(\mathbf{x}) = 0$) of $\rho^{(g)} > \rho/\omega$.

To penalize misclassification on a fine scale, one can define smooth convex upper bounds of the margin-based one loss $L_{\mathbb{I}}(\rho)$, and minimize the corresponding empirical risks instead of the training error $\widehat{R}_{\mathbb{I}}$. The most common losses, depicted in Figure 3.2, are the *exponential loss*

$$L_{\text{EXP}}(\rho) = \exp(-\rho), \tag{3.4}$$

the *hinge loss*

$$L_{1+}(\rho) = \max(0, 1 - \rho), \tag{3.5}$$

and the *logistic loss*

$$L_{\text{LOG}}(\rho) = \log_2\left(\exp(-\rho) + 1\right). \tag{3.6}$$

Given the margin-based loss $L_\bullet(\rho)$, the corresponding margin-based empirical risk can be defined as

$$\widehat{R}_\bullet(f) = \frac{1}{n}\sum_{i=1}^{n} L_\bullet(f(\mathbf{x}_i)y_i) \tag{3.7}$$

Minimizing the margin-based empirical risks $\widehat{R}_{\text{EXP}}(f)$, $\widehat{R}_{1+}(f)$, or $\widehat{R}_{\text{LOG}}(f)$ have both algorithmic and statistical advantages. First, combining the minimization of these risks with Lipschitz-type penalties[4] often leads to convex optimization problems that can be solved with standard algorithms (e.g., linear, quadratic, or convex programming). Second, it can also be shown within the complexity regularization framework that the minimization of these penalized convex risks lead to *large margin* classifiers with good generalization properties.

[4] often of the form of L_1 or L_2 penalties on the coefficient vector of a *generalized linear* model; see Section 3.1.4

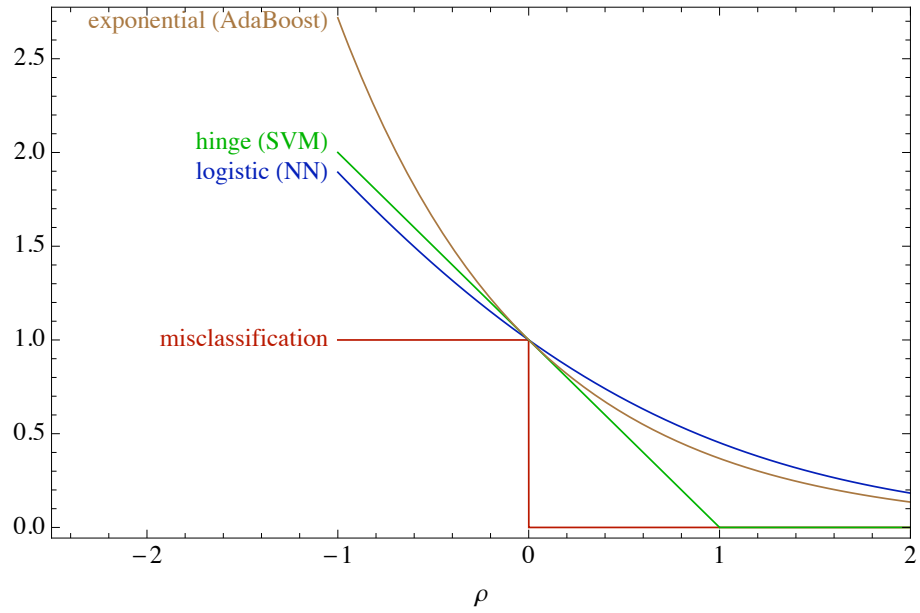


Figure 3.2: Convex margin losses used in binary classification.

3.1.4 Binary classification algorithms

The three most popular binary classification algorithms are the *multi-layer perceptron* or *neural network* (NN) [49, 50], the *support vector machine* (SVM) [51, 52, 53], and ADABOOST [44]. They have very different origins, and the algorithmic details also make them stand apart, nevertheless, they share some important concepts, and their practical success is explained, at least qualitatively, by the same theory on large margin classification.

An important pragmatic similarity is that they all learn *generalized linear* discriminant functions of the form

$$f(\mathbf{x}) = \sum_{t=1}^{T} \alpha^{(t)} h^{(t)}(\mathbf{x}). \tag{3.8}$$

In neural networks $h^{(t)}(\mathbf{x})$ is a *perceptron*, that is, a linear combination of the input features followed by a sigmoidal nonlinearity (such as arctan)

$$h^{(t)}(\mathbf{x}) = \sigma\left(w_0^{(t)} + \sum_{j=1}^{d} w_j^{(t)} x^{(j)}\right),$$

where $x^{(j)}$ is the jth element of the d-dimensional input vector $\mathbf{x}$. The output weight vector $\boldsymbol{\alpha} = \left(\alpha^{(1)}, \ldots, \alpha^{(T)}\right)$ and the $T \times (d+1)$-dimensional input weight matrix

$$\mathbf{W} = \begin{pmatrix} w_0^{(1)} & \ldots & w_d^{(1)} \\ \vdots & \ddots & \vdots \\ w_0^{(T)} & \ldots & w_d^{(T)} \end{pmatrix}$$

are learned using gradient descent optimization (called *back-propagation* in the NN literature). In the case of binary classification the neural network is usually trained to minimize the logistic loss (3.6). One of the advantages of NNs is their versatility: as long as the loss function is differentiable, it can be plugged into the algorithm. The differentiability condition imposed by the gradient descent optimization is, on the other hand, constrains the loss function and the nonlinearities used in the hidden units: one could not use, say, a Heaviside-type nonlinearity or the one loss.

The invention of the multilayer perceptron in 1986 [49] was a technological breakthrough: complex pattern recognition problems (e.g., handwritten character recognition) could suddenly be solved efficiently using neural networks. At the same time, the theory of machine learning at these early days was not yet developed to the point of being able to explain the principles behind algorithmic design. Most of the engineering techniques (such as *weight decay* or *early stopping*) were developed using a trial-and-error approach, and they were theoretically justified only much later within the complexity regularization and large margin framework [54, 55]. Partly because of the "empirical" nature of neural network design, a common belief developed about the "user-unfriendliness" of neural networks. Added to this reputation was the uneasiness of having a non-convex optimization problem: back-propagation cannot be guaranteed to converge to a global minimum of the empirical risk. This is basically a no-issue in practice (especially on today's large problems), still, it is a common criticism from people who usually have never experimented with neural networks on real problems. As a consequence, neural networks were slightly overshadowed in the 90s with the appearance of the support vector machine and ADABOOST. Today neural networks are, again, becoming more popular partly because of user-friendly program packages (e.g., [56]) and partly due to the computational efficiency of the training algorithm (especially its stochastic version).

Support vector machines also learn generalized linear discriminant functions of the form (3.8) with

$$h^{(t)}(\mathbf{x}) = y_t K(\mathbf{x}_t, \mathbf{x}),$$

where $K(\mathbf{x}, \mathbf{x}')$ is a positive semidefinite *kernel* function that expresses the similarity between two observations $\mathbf{x}$ and $\mathbf{x}'$. $K(\mathbf{x}, \mathbf{x}')$ is usually monotonically decreasing with the distance between $\mathbf{x}$ and $\mathbf{x}'$. The most common choice for K is the squared exponential (a.k.a. Gaussian) kernel. The index t ranges over $t = 1, \ldots, n$ which means that each training point $(\mathbf{x}_t, y_t) \in \mathcal{D}$ contributes to f a kernel function centered at $\mathbf{x}_t$ with a sign equal to the label y_t, so the final discriminant function can be interpreted as a weighted *nearest neighbor* classifier.

The objective of training the SVM is to find the weight vector $\boldsymbol{\alpha} = \left(\alpha^{(1)}, \ldots, \alpha^{(T)}\right)$ that minimizes the hinge loss (3.5) with a complexity penalty (the L_2 loss on the weights of features in the Hilbert space induced by $K(\mathbf{x}, \mathbf{x}')$). Unlike neural networks, the objective function was designed based on the theory of large margin classification to ensure good generalization properties. A great advantage of the setup is that the objective is a quadratic function of $\boldsymbol{\alpha}$ with linear constraints $0 \leq \alpha \leq C$, so the global optimum can be found using quadratic programming. The result is sparse: only a subset of the training points in $\mathcal{D}$ have nonzero coefficients. These points are called *support vectors*, giving the name of the method.

The biggest disadvantage of the technique is its training time: naïve quadratic programming solvers run in super-quadratic time, and even with the most sophisticated tricks it is hard to beat the $O(n^2)$ barrier. The second disadvantage of the method is that in high-dimensional problems the number of support vectors is comparable to the size of the training set, so, when evaluating f at the test phase is a bottleneck (see Section 3.2.2), SVMs can be prohibitively slow. Third, unlike neural networks, SVMs are designed for binary classification, and the generic extensions for multi-class classification and regression do not reproduce the performance of binary SVM. Despite these disadvantages, the appearance of the support vector machine in the middle of the 90s revolutionized the technology of pattern recognition. Beside its remarkable generalization performance, the small number of hyperparameters [5] and the appearance of turn-key software packages made SVMs the method of choice for a wide range of applications involving small-to-moderate size training sets. With the appearance of extra-large training sets in the last decade, training time and optimization became more important issues than generalization [57], so SVMs lost somewhat their dominant role.

ADABOOST learns a generalized linear discriminant function of the form (3.8) in an iterative fashion. It can be considered as constructing a neural network by adding one neuron at a time, but since back-propagation is no longer applied, there is no differentiability restriction on the *base*

[5] C in the bound of the αs and a length scale parameter of the kernel function $K(\mathbf{x}, \mathbf{x}')$

classifiers $h^{(t)}$. This opens the door to using domain-dependent features, making AdaBoost easy to adapt to a wide range of applications. The basic binary classification algorithm (Figure 3.3) consists of elementary steps that can be implemented by a first year computer science student in an hour. The only tricky step is the implementation of the base learner (line 3) whose goal is to return $h^{(t)}$ with a small *weighted* error

$$\widehat{R}_{\mathbb{I}}(h, \mathbf{w}) = \frac{1}{n} \sum_{i=1}^{n} w_i \mathbb{I}\left\{h(\mathbf{x}_i) \neq y_i\right\},$$

but most of the simple learning algorithms (e.g., decision trees, linear classifiers) can be easily adapted to this modified risk. The weights $\mathbf{w} = (w_1, \ldots, w_n)$ over the training points are initialized uniformly (line 1), and then updated in each iteration by a simple and intuitive rule (lines 7-10): if $h^{(t)}$ misclassifies $(\mathbf{x}_i, y_i)$, the weight w_i increases (line 8), whereas if $h^{(t)}$ correctly classifies $(\mathbf{x}_i, y_i)$, the weight w_i decreases (line 10). In this way subsequent base classifiers will concentrate on points that were "missed" by previous base classifiers. The coefficients $\alpha^{(t)}$ are also set analytically to the formula in line 5 which is monotonically decreasing with respect to the weighted error.

AdaBoost$\left(\mathcal{D}_n, \text{Base}(\cdot,\cdot), T\right)$

1 $\mathbf{w}^{(1)} \leftarrow (1/n, \ldots, 1/n)$ ▷ *initial weights*
2 **for** $t \leftarrow 1$ **to** T
3 $\quad h^{(t)} \leftarrow \text{Base}\left(\mathcal{D}_n, \mathbf{w}^{(t)}\right)$ ▷ *base classifier*
4 $\quad \epsilon^{(t)} \leftarrow \sum_{i=1}^{n} w_i^{(t)} \mathbb{I}\left\{h^{(t)}(\mathbf{x}_i) \neq y_i\right\}$ ▷ *weighted error of the base classifier*
5 $\quad \alpha^{(t)} \leftarrow \frac{1}{2} \ln\left(\frac{1-\epsilon^{(t)}}{\epsilon^{(t)}}\right)$ ▷ *coefficient of the base classifier*
6 $\quad$ **for** $i \leftarrow 1$ **to** n ▷ *re-weighting the training points*
7 $\quad\quad$ **if** $h^{(t)}(\mathbf{x}_i) \neq y_i$ **then** ▷ *error*
8 $\quad\quad\quad w_i^{(t+1)} \leftarrow \frac{w_i^{(t)}}{2\epsilon^{(t)}}$ ▷ *weight increases*
9 $\quad\quad$ **else** ▷ *correct classification*
10 $\quad\quad\quad w_i^{(t+1)} \leftarrow \frac{w_i^{(t)}}{2\left(1-\epsilon^{(t)}\right)}$ ▷ *weight decreases*
11 **return** $f^{(T)}(\cdot) = \sum_{t=1}^{T} \alpha^{(t)} h^{(t)}(\cdot)$ ▷ *weighted "vote" of base classifiers*

Figure 3.3: The pseudocode of binary AdaBoost. $\mathcal{D} = \left\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\right\}$ is the training set, $\text{Base}(\cdot,\cdot)$ is the base learner, and T is the number of iterations.

This simple algorithm has several beautiful mathematical properties. First it can be shown [44] that if each base classifier $h^{(t)}$ is slightly better then a random guess (that is, every weighted error $\epsilon^{(t)} \leq \frac{1}{2} - \delta$ with $\delta > 0$), then the (unweighted) training error $\widehat{R}_{\mathbb{I}}$ of the final discriminant function $f^{(T)}$ becomes zero after at most

$$T = \left\lceil \frac{\log n}{2\delta^2} \right\rceil + 1$$

steps. The logarithmic dependence of T on the data size n implies that the technique is formally

a *boosting* algorithm from which the second part of its name derives.[6] Second, it can be shown that the algorithm minimizes the exponential risk $\widehat{R}_{\text{EXP}}(f)$ (3.4) using a greedy functional gradient approach [58, 59]: in each iteration $h^{(t)}$ is selected to maximize the decrease of $\widehat{R}_{\text{EXP}}\left(f^{(t-1)} + \alpha h^{(t)}\right)$ at $\alpha = 0$, and then $\alpha^{(t)}$ is set to

$$\alpha^{(t)} = \arg\min_{\alpha} \widehat{R}_{\text{EXP}}\left(f^{(t-1)} + \alpha h^{(t)}\right).$$

Furthermore, for inseparable data sets (for which no linear combination of base classifiers achieves 0 training error), the exponential risk $\widehat{R}_{\text{EXP}}(f^{(T)})$ goes to the minimum achievable exponential risk as $T \to \infty$ [60]. It can also be proven [61] that for the *normalized* discriminant function

$$\widetilde{f}(\mathbf{x}) = \frac{\sum_{t=1}^{T} \alpha^{(t)} h^{(t)}(\cdot)}{\sum_{t=1}^{T} \alpha^{(t)}},$$

the *margin-based* training error

$$\widehat{R}_{\text{I}}^{(\theta)}(\widetilde{f}) = \frac{1}{n} \sum_{i=1}^{n} \text{I}\left\{\widetilde{f}(\mathbf{x}_i) y_i < \theta\right\}$$

also goes to zero exponentially fast for all $\theta < \widetilde{\rho}^*/2$, where $\widetilde{\rho}^*$ is the maximum achievable minimum margin. This results shows that ADABOOST, similarly to the support vector machine, leads to a large margin classifier. It also explains the surprising experimental observation that the generalization error $R_{\text{I}}(f)$ (estimated on a test set) often decreases even after the training error $\widehat{R}_{\text{I}}(f)$ reaches 0.

ADABOOST arrived to the pattern recognition scene in 1997 when support vector machines were in full bloom. ADABOOST has a lot of advantages over its main rival: it is fast (its time complexity is linear in n and T), it is an *any-time* algorithm, it has few hyperparameters, it is resistant to overfitting, and it can be directly extended to multi-class classification [45]. Due to these advantages, it rapidly became the method of choice of machine learning experts in certain types of problems where the natural setup is to define a large set of plausible features of which the final solution $f^{(T)}$ uses only a few (a so called *sparse* classifier). Arguably the most famous application is the first face detector that could be run real-time on 30 frame-per-second video [62]. At the same time, ADABOOST is much less known among users with no computer science background than the support vector machine. The reason is paradoxically the simplicity of ADABOOST: since it is so easy to implement with a little background in programming, no machine learning expert took the effort to provide a turn-key software package easily usable for non-experts.[7] In fact it is not surprising that the only large non-computer-scientist community in which ADABOOST is much more popular than SVM is experimental physics: physicists, especially in high energy physics, are heavy computer-users with considerable programming skills. In the last five years ADABOOST (and other similar *ensemble* methods) seem to have been taking over the field of large-scale applications. In recent large-scale challenges [63, 64] the top entries are dominated by ensemble-based solutions, and SVM is almost non-existent in the most efficient approaches.

Although binary ADABOOST (Figure 3.3) is indeed simple to implement, multi-class ADABOOST has some nontrivial tricks. There are several multi-class extensions of the original binary algorithm, and it is not well-known, even among machine learning experts, that the original ADABOOST.M1 and ADABOOST.M2 algorithms [44] are largely suboptimal compared to ADABOOST.MH published a couple of years later [45].[8] On the other hand, the ADABOOST.MH paper [45] did not specify the implementation details of multi-class base learning (see Section 3.3), making the implementation non-trivial.

[6] The algorithm is also ADA*ptive* because the coefficients $\alpha^{(t)}$ depend on the errors $\epsilon^{(t)}$ of the base classifiers $h^{(t)}$.

[7] We are attempting to improve the situation with our free multi-class program package available at http://multiboost.org.

[8] The commonly used WEKA package (http://www.cs.waikato.ac.nz/ml/weka/), for example, only contains a badly written implementation of ADABOOST.M1, effectively harming the reputation of boosting as a whole.

3.2 Applications

In this section we illustrate the versatility of the abstract supervised learning model described in Section 3.1 through presenting some of the machine learning applications we have worked on. It turns out that real-world applications are never so simple as just taking a turn-key implementation out of the box and running it. To make a system work, one needs both domain expertise and machine learning expertise, and so it often requires an interdisciplinary approach and considerable communication effort from experts of different backgrounds. Nevertheless, once the problem is reduced to the abstract setup described in Section 3.1, machine learning algorithms can be very efficient.

As the examples will show, classification or regression is only one step in the pattern recognition pipeline (Figure 3.4). Data collection is arguably the most costly step in most of the applications. In particular, harvesting the labels y_i usually involves human interaction. How to do this in a cost-effective way by, for example, using CAPTCHAs to obtain character labels [65] or making people label images by playing a game [66, 67], is itself a challenging research subject. On the other hand, in experimental physics simulators can be used to sample from the distribution $p(\mathbf{x} \mid y)$, so in these applications data collection is usually not an issue.

The second step of the pipeline is preprocessing the data. There is a wide range of techniques that can be applied here, usually with the objective of simplifying the supervised learning task. First, if the observations are not already in a format of a real vector $\mathbf{x} \in \mathbb{R}^d$, *features* should be extracted from the raw data. The goal here is to find features that are plausibly correlated with the label y, so domain knowledge is almost always necessary to carry out this step. Since learning algorithms usually deteriorate as the dimension of the input $\mathbf{x}$ increases (because of the so called *curse of dimensionality*), dimensionality reduction is usually part of data preprocessing. Principal component analysis is the basic tool that is usually applied, but nonlinear manifold algorithms [9, 10] may also be useful if the training set is not too large. The output space $\mathcal{L}$ can also be transformed to massage the original problem into a setup that can be solved by standard machine learning tools. This can be done in an ad hoc way, but there exist also principled *reduction* techniques between machine learning problems (binary/multi-class classification, regression, even reinforcement learning) [68].

After training, postprocessing the results can also be an important step. If, for example, the original complex problem was reduced to an easy-to-solve setup, re-transforming the obtained labels and even re-calibrating the solution is often a good idea. In well-documented challenges of the last five years [63, 69, 64] we have also learned that the most competitive results are obtained when divers models trained using different algorithms are combined, so *model aggregation* techniques are also becoming part of the generic machine learning toolbox. In Section 3.5 we show an example for both of these postprocessing techniques.

Sometimes it happens that an application poses a specific problem that can not be solved with existing tools. Solving such a problem and generalizing the solution to make it applicable for similar problems is one of the motivational engines behind the development of the machine learning domain. We will present a couple of examples of this kind.

3.2.1 Music classification, web page ranking, muon counting

In [70] we use ADABOOST for telling apart speech from music. The system starts by constructing the spectrogram on a set of recorded audio signals which constitute the observation vectors $\mathbf{x}$ in this case. ADABOOST is then run in a feature space inspired by image classification on the spectrogram "images". The output ℓ of the system is binary, that is, $\ell \in \mathcal{L} = \{\text{MUSIC}, \text{SPEECH}\}$. In [71] we stay within the music classification domain but we tackle a more difficult problem: finding the performing artist and the genre of a song based on the audio signal of the first 30 s of the song. The first module of the classifier pipeline is an elaborate signal processor that collects a vector of features for each 2 s segment and then aggregates them to constitute an observation vector $\mathbf{x}$ with about 800 elements per song. We train two systems, one for finding the artist performing the song and another for predicting its genre. Our algorithm was the best genre

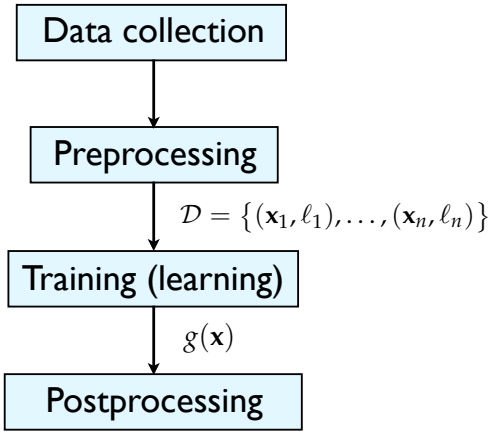


Figure 3.4: The pattern recognition pipeline.

classifier among 15 submissions and the second-best among 10 submissions at recognizing artists at the MIREX 2005 international contest in music information extraction. This application also stretches the limits of the classical multi-class (single-label) setup. It is plausible that a song belongs to several genres leading to a so-called *multi-label* classification. It may also be useful to train a unique system for predicting both the artist and the genre at the same time. This problem is known as *multi-task* learning. In Section 3.3.1 we present the framework in which these problems, along with the single-label setup, can be formalized and solved.

The regression problem can be illustrated by our recent work [72] in which we aim to predict the number of muons (an elementary particle) in a signal recorded in a water Cherenkov detector of the Pierre Auger experiment [73] (see Section 4.4 for a detailed description). The pipeline, again, starts by extracting features that are plausibly correlated with the muon content of the signal. The 172 features are quite correlated, so first we apply principal component analysis to reduce the size of the observation vector $\mathbf{x}$ to 19. Then we train a neural network [50, 56] and convert its output into a point estimate and a pointwise error bar (uncertainty) of the number of muons.

In [74, 75] we use ADABOOST.MH for web page ranking. The input $\mathbf{x}$ of the classifier g is a set of features representing a search query and a document, and the label ℓ represents the relevance of the document to the query. The goal is to rank the documents in order of their relevance. Of course, if the relevance of the document is correctly predicted, the implied ranking will also be good. Nevertheless, it is hard to formally derive a meaningful loss for each (document, query, relevance) triplet from the particular loss defined on rankings. The solution to this problem is a post-processing technique called *calibration*: instead of directly using the output of the classifier g, we send it through another regressor or ranker which is now trained to minimize the desired risk. Our concrete system, described in detail in Section 3.5, also contains another postprocessing module called *model aggregation*: instead of training one classifier g, we train a large number of classifiers by varying data preprocessing and algorithmic hyperparameters, and combine the results using a simple weighted voting scheme.

3.2.2 Trigger design in experimental physics

Our last example is a work in progress; we present it here because it is one of those problems for which no turn-key machine learning solution exists even though the problem definitely belongs to the domain of machine learning. The problem is trigger design in experimental physics. In experimental physics, *detectors* are built to collect observations of partially known or unknown phenomena in order to measure (estimate) some physical quantities. The phenomena can either occur naturally (e.g., cosmic rays) or they can be generated by man-made instruments (e.g., in

particle accelerators). A common feature of today's detectors is that the phenomena they want to measure are very rare which means that it is increasingly difficult to separate *signal* generated by the phenomenon from *background*, that is, an observed event that just happen to look like real signal because of random fluctuations or due to some uninteresting phenomena. The final goal of the *analysis* is to collect a large statistics of observations that, with high probability, were generated by the targeted phenomenon. Since the background is often several orders of magnitudes more frequent than the signal, part of the background/signal separation cannot be done offline. Due to either limited disk capacity or limited bandwidth, most of the raw signal has to be discarded by online *triggers*.

In the machine learning paradigm, a trigger is just a binary classifier with

$$\mathcal{L} = \{\text{SIGNAL}, \text{BACKGROUND}\}.$$

There are several attributes that make the trigger a *special* binary classifier. First, it is extremely unbalanced: the probability of BACKGROUND is practically 1 in a lot of cases. This makes the classical setup of minimizing the error probability $R_{\mathbb{I}}(g)$ (3.2) inadequate: it is very hard to beat the trivial constant classifier $g(\mathbf{x}) \equiv \text{BACKGROUND}$ which has an error of $R_{\mathbb{I}}(g) = P(\ell = \text{SIGNAL})$. Indeed, the natural *gain* in trigger design is the *true positive* or *hit* indicator

$$G_{\text{TP}}\big(g, (\mathbf{x}, \ell)\big) = \mathbb{I}\left\{g(\mathbf{x}) = \text{SIGNAL} \mid \ell = \text{SIGNAL}\right\}.$$

Taking the complement of the true positive indicator

$$L_{\overline{\text{TP}}}\big(g, (\mathbf{x}, \ell)\big) = 1 - \mathbb{I}\left\{g(\mathbf{x}) = \text{SIGNAL} \mid \ell = \text{SIGNAL}\right\} = \mathbb{I}\left\{g(\mathbf{x}) = \text{BACKGROUND} \mid \ell = \text{SIGNAL}\right\}$$

as the loss, the implied risk is

$$R_{\overline{\text{TP}}}(g) = \mathbb{E}_{(\mathbf{x},\ell)\sim\mathfrak{D}}\left\{L_{\overline{\text{TP}}}\big(g, (\mathbf{x}, \ell)\big)\right\} = P\big(g(\mathbf{x}) = \text{BACKGROUND} \mid \ell = \text{SIGNAL}\big).$$

Again, minimizing $R_{\overline{\text{TP}}}(g)$ is trivial by setting $g(\mathbf{x}) \equiv \text{SIGNAL}$ this time, so the goal is to minimize $R_{\overline{\text{TP}}}(g)$ with a *constraint* that the *false positive* rate

$$R_{\text{FP}}(g) = P\big(g(\mathbf{x}) = \text{SIGNAL} \mid \ell = \text{BACKGROUND}\big)$$

is kept below a fixed level p_{FP}. As we mentioned above, in triggers $P(\ell = \text{SIGNAL}) \ll P(\ell = \text{BACKGROUND})$, so the false positive rate $R_{\text{FP}}(g)$ is approximately equal to the unconditional positive rate $P(g(\mathbf{x}) = \text{SIGNAL})$. In experimental physics terminology this means that a constraint is imposed on the *trigger rate*.

The second attribute that makes trigger design special is that we have strict computational constraints imposed on the evaluation of the classifier g on test samples $\mathbf{x}$. Typically, observations $\mathbf{x}$ arrive at a given rate and $g(\mathbf{x})$ has to be run online. The designer can have some flexibility on the parallel handling of the incoming events, but computational resources are often limited. In the case of JEM EUSO [76], the detector will be installed on the International Space Station where electric consumption is limited and harsh conditions require the use of robust hardware with low clock-rate.

Trigger design shares these two attributes (unbalanced classes and test-time constraints) with object detection in computer vision where machine learning has been applied widely. For example, when the goal is to detect faces in images, the probability of the signal class (face) is much lower then the probability of background (everything else). We have also computational constraints at test time if the goal is to detect faces *online* in video recordings with given frame rate and the detector hardware must fit into a compact camera. What makes trigger design slightly more challenging is the extremely low signal probability and the fact that the computational cost of each feature (elements of $\mathbf{x}$) can vary in a large range. For example, the LHCb trigger [77] can use "cheap" observables that can be evaluated fast to rapidly obtain a rough classifier. One can also almost reconstruct the collision event which can take up a large portion of the allotted time, but the resulting feature can be used reliably to discard background events.

This example shows why a natural answer to these challenges is to design *cascade* classifiers both in experimental physics and in object detection [62, 78, 79, 80, 81, 82]. A cascade classifier $g(\mathbf{x})$ is composed of a list of simpler binary classifiers $h_1(\mathbf{x}), \dots, h_N(\mathbf{x})$ evaluated sequentially. For $j < N$, if $h_j(\mathbf{x})$ classifies the observation $\mathbf{x}$ negatively, the final classification of $g(\mathbf{x})$ is BACKGROUND, and if the output of $h_j(\mathbf{x})$ is positive, the observation is sent to the next $(j+1)$th *stage* (or *level* in experimental physics terminology). The classifier $h_N(\mathbf{x})$ at the last stage is the only one that can classify $\mathbf{x}$ as a SIGNAL. In computer vision, the stage classifiers $h_j(\mathbf{x})$ are usually learned using classification algorithms (most often ADABOOST). Although machine learning algorithms are now widely used in high-energy physics for data analysis [83, 84], their application for trigger design is quite new: the first such algorithm to our knowledge is now running in the LHCb trigger [77]. Some of the newest detection algorithms also attempt to learn the cascade structure automatically, nevertheless, manual experimentation is usually required to set hyperparameters (stagewise false positive/false negative rates, computational complexity of stage classifiers $h_j(\mathbf{x})$). In Section 3.6 we present a principled approach [85] that can be used to automatically design test-time constrained classifiers. The automatic design also allows us to go beyond the cascade structure which is, although quite intuitive, an artificial constraint to keep the classifier structure simple and to accommodate manual tuning.

3.3 Multi-class ADABOOST

The original ADABOOST paper of Freund and Schapire [44] also described two multi-class extensions, ADABOOST.M1 and ADABOOST.M2. Both required a quite strong performance from the base learners, partly defeating the purpose of boosting. The breakthrough came with Schapire and Singer's seminal paper [45], which described, among other interesting algorithms, ADABOOST.MH. In this section we first introduce the general multi-class learning setup (Section 3.3.1), then we give the details and ADABOOST.MH and show its algorithmic convergence (Section 3.3.2).

3.3.1 The multi-class setup: single-label, multi-label and multi-task

In *multi-class* classification, the label ℓ of the observation $\mathbf{x}$ comes from a finite set. Without loss of generality, we will suppose that $\ell \in \mathcal{L} = \{1, \dots, K\}$. We will refer to the label of the observation $\mathbf{x}$ as $\ell(\mathbf{x})$. For technical reasons that will become clear later, we will encode the label using a K-dimensional binary vector $\mathbf{y} \in \mathcal{Y} = \{\pm 1\}^K$. In the classical multi-class setup, $\mathbf{y}$ is known as the *one-hot* representation: the $\ell(\mathbf{x})$th element of $\mathbf{y}$ will be 1 and all the other elements will be -1, that is,

$$y_\ell = \begin{cases} +1 & \text{if } \ell = \ell(\mathbf{x}), \\ -1 & \text{otherwise.} \end{cases}$$

Beside reflecting well the architecture of multi-class neural network or multi-class ADABOOST, this representation has the advantage to be generalizable to *multi-label* or *multi-task* learning when and observation $\mathbf{x}$ can belong to several classes (see the music classification examples in Section 3.2.1). To avoid confusion, from now on we will call $\mathbf{y}$ and ℓ the label and the label *index* of $\mathbf{x}$, respectively. For emphasizing the distinction between multi-class and multi-label classification, we will use the term *single-label* for the classical multi-class setup, and reserve multi-class to situations we talk about the three setups in general.

The multi-class training data is

$$\mathcal{D} = \big\{(\mathbf{x}_1, \mathbf{y}_1), \dots, (\mathbf{x}_n, \mathbf{y}_n)\big\} \in (\mathcal{X} \times \mathcal{Y})^n.$$

and the goal of learning is to infer a vector-valued multi-class classifier $\mathbf{g} : \mathcal{X} \to \mathcal{Y}$ from $\mathcal{D}$. Sometimes we will use the notion of an $n \times d$ *observation matrix* of $\mathbf{X} = (\mathbf{x}_1, \dots, \mathbf{x}_n)$ and an $n \times K$ *label matrix* $\mathbf{Y} = (\mathbf{y}_1, \dots, \mathbf{y}_n)$ instead of the set of pairs $\mathcal{D}$.

As in binary classification, learning algorithms usually output a multi-class *discriminant* function $\mathbf{f} : \mathcal{X} \to \mathbb{R}^K$. The semantics of $\mathbf{f}$ is that the higher the ℓth element $f_\ell(\mathbf{x})$ of $\mathbf{f}(\mathbf{x})$, the more

likely is that the real label index of $\mathbf{x}$ is ℓ. The vector-valued classifier $\mathbf{g}$ is formally obtained by simply thresholding the elements of $\mathbf{f}(\mathbf{x})$ at 0, that is,

$$g_\ell(\mathbf{x}) = \operatorname{sign}\big(f_\ell(\mathbf{x})\big), \quad \ell = 1, \dots, K.$$

The single-label output of the algorithm is then the class index ℓ for which $f_\ell(\mathbf{x})$ is maximal, that is,

$$\ell_{\mathbf{f}}(\mathbf{x}) = \arg\max_{\ell} f_\ell(\mathbf{x}).$$

The classical measure of the performance of the multi-class discriminant function $\mathbf{f}$ is the *single-label one-loss*

$$L_{\mathbb{I}}\big(\mathbf{f}, (\mathbf{x}, \ell)\big) = \mathbb{I}\left\{\ell \neq \ell_{\mathbf{f}}(\mathbf{x}_i)\right\}$$

that defines the single-label training error

$$\widehat{R}_{\mathbb{I}}(\mathbf{f}) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{I}\left\{\ell(\mathbf{x}_i) \neq \ell_{\mathbf{f}}(\mathbf{x}_i)\right\}. \tag{3.9}$$

Another, perhaps more comprehensive, way to measure of the performance of $\mathbf{f}$ is by computing the *weighted Hamming loss*

$$L_{\mathrm{H}}\big(\mathbf{f}, (\mathbf{x}, \mathbf{y}), \mathbf{w}\big) = \sum_{\ell=1}^{K} w_\ell \mathbb{I}\left\{\operatorname{sign}\big(f_\ell(\mathbf{x})\big) \neq y_\ell\right\}$$

where $\mathbf{w} = \big[w_\ell\big]$ is an $\mathbb{R}^k$-valued "user-defined" weight vector over labels. The corresponding empirical risk (training error) is

$$\widehat{R}_{\mathrm{H}}(\mathbf{f}, \mathbf{W}) = \frac{1}{n}\sum_{i=1}^{n}\sum_{\ell=1}^{K} w_{i,\ell}\mathbb{I}\left\{\operatorname{sign}\big(f_\ell(\mathbf{x}_i)\big) \neq y_{i,\ell}\right\}, \tag{3.10}$$

where $\mathbf{W} = \big[w_{i,\ell}\big]$ is an $n \times k$ weight matrix over data points and labels.

In the multi-label/multi-task setup, when, for example, it is equally important to predict that a song is *not* "folk" as predicting that it *is* "classical", the Hamming loss with uniform weights

$$w_\ell = \frac{1}{K}, \quad \ell = 1, \dots, K \tag{3.11}$$

is a natural measure of performance: it represents the uniform error rate of missing any class sign y_ℓ of a given observation $\mathbf{x}$. In single-label classification, $\mathbf{w}$ is usually set asymmetrically to

$$w_\ell = \begin{cases} \frac{1}{2} & \text{if } \ell = \ell(\mathbf{x}) \text{ (i.e., if } y_\ell = 1), \\ \frac{1}{2(K-1)} & \text{otherwise (i.e., if } y_\ell = -1). \end{cases} \tag{3.12}$$

The idea behind this scheme is that it will create K well-balanced *one-against-all* binary classification problems: if each of the K classes have n/K examples in $\mathcal{D}$, then for each class ℓ, the sum of the weights of the positive examples in the column $\mathbf{w}_{\cdot,\ell}$ of the weight matrix $\mathbf{W}$ will be equal to the sum of the weights of the negative examples. Indeed, for positive examples in class ℓ we have

$$\frac{1}{n}\sum_{i=1}^{n} w_{i,\ell}\mathbb{I}\left\{y_{i,\ell} = 1\right\} = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{2}\mathbb{I}\left\{y_{i,\ell} = 1\right\} = \frac{1}{n} \times \frac{1}{2} \times \frac{n}{K} = \frac{1}{2K},$$

and for negative examples of the same column we have

$$\frac{1}{n}\sum_{i=1}^{n} w_{i,\ell}\mathbb{I}\left\{y_{i,\ell} = -1\right\} = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{2(K-1)}\mathbb{I}\left\{y_{i,\ell} = -1\right\} = \frac{1}{n} \times \frac{1}{2(K-1)} \times \frac{n(K-1)}{K} = \frac{1}{2K}.$$

Note that both schemes (3.11) and (3.12) boil down to the classical uniform weighting in binary classification.

It is easy to see that if the Hamming error $\widehat{R}_{\mathrm{H}}(\mathbf{f}, \mathbf{W})$ is 0 then the one-error $\widehat{R}_{\mathrm{I}}(\mathbf{f})$ is also 0, but not the other way around: it is possible that a point is correctly classified, that is $f_{\ell(\mathbf{x})}(\mathbf{x}) > f_{\ell}(\mathbf{x})$ for all $\ell \neq \ell(\mathbf{x})$, yet several of the negative classes $\ell : y_\ell = -1$ have positive outputs $f_\ell(\mathbf{x})$. When both $\widehat{R}_{\mathrm{H}}(\mathbf{f}, \mathbf{W})$ and $\widehat{R}_{\mathrm{I}}(\mathbf{f})$ are nonzero, both $\widehat{R}_{\mathrm{H}}(\mathbf{f}, \mathbf{W}) > \widehat{R}_{\mathrm{I}}(\mathbf{f})$ and $\widehat{R}_{\mathrm{H}}(\mathbf{f}, \mathbf{W}) < \widehat{R}_{\mathrm{I}}(\mathbf{f})$ can happen. What can be said is that for every misclassified point according to $\widehat{R}_{\mathrm{I}}(\mathbf{f})$, $\mathbf{f}(\mathbf{x})$ misses the sign of at least one of the labels y_ℓ, so $\widehat{R}_{\mathrm{I}}(\mathbf{f}) < K\widehat{R}_{\mathrm{H}}(\mathbf{f}, \mathbf{W})$ for (3.11) and $\widehat{R}_{\mathrm{I}}(\mathbf{f}) < 2K\widehat{R}_{\mathrm{H}}(\mathbf{f}, \mathbf{W})$ for (3.12).

3.3.2 ADABOOST.MH

The goal of the ADABOOST.MH algorithm ([45], Figure 3.5) is to return a vector-valued discriminant function $\mathbf{f}^{(T)} : \mathcal{X} \to \mathbb{R}^K$ with a small Hamming loss $\widehat{R}_{\mathrm{H}}(\mathbf{f}, \mathbf{W})$ (3.10) by minimizing the *weighted multi-class exponential margin-based error*

$$\widehat{R}_{\mathrm{EXP}}\big(\mathbf{f}^{(T)}, \mathbf{W}\big) = \frac{1}{n}\sum_{i=1}^{n} L_{\mathrm{EXP}}\big(\mathbf{f}, (\mathbf{x}, \mathbf{y}), \mathbf{w}_i\big) \tag{3.13}$$

with the convex loss L_{EXP} defined as

$$L_{\mathrm{EXP}}\big(\mathbf{f}, (\mathbf{x}, \mathbf{y}), \mathbf{w}\big) = \sum_{\ell=1}^{K} w_\ell \exp\big(-f_\ell(\mathbf{x}) y_\ell\big).$$

Since $\exp(-\rho) \geq \mathbb{I}\{\rho < 0\}$, the exponential loss $L_{\mathrm{EXP}}\big(\mathbf{f}, (\mathbf{x}, \mathbf{y}), \mathbf{w}\big)$ upper bounds the Hamming loss $L_{\mathrm{H}}\big(\mathbf{f}, (\mathbf{x}, \mathbf{y}), \mathbf{w}\big)$, and so

$$\widehat{R}_{\mathrm{EXP}}\big(\mathbf{f}^{(T)}, \mathbf{W}\big) \geq \widehat{R}_{\mathrm{H}}\big(\mathbf{f}^{(T)}, \mathbf{W}\big).$$

ADABOOST.MH builds the final discriminant function

$$\mathbf{f}^{(T)}(\mathbf{x}) = \sum_{t=1}^{T} \mathbf{h}^{(t)}(\mathbf{x}) \tag{3.14}$$

as a sum of T *base classifiers* $\mathbf{h}^{(t)} : \mathcal{X} \to \mathbb{R}^K$ returned by a *base learner* algorithm $\mathrm{BASE}\big(\mathbf{X}, \mathbf{Y}, \mathbf{W}^{(t)}\big)$ in each iteration t.

Algorithmic convergence

The following derivation shows that

$$\widehat{R}_{\mathrm{EXP}}\big(\mathbf{f}^{(T)}, \mathbf{W}\big) = \prod_{t=1}^{T} Z\big(\mathbf{h}^{(t)}, \mathbf{W}^{(t)}\big),$$

where

$$Z\big(\mathbf{h}, \mathbf{W}^{(t)}\big) = \sum_{i=1}^{n}\sum_{\ell=1}^{K} w_{i,\ell}^{(t)} \exp\big(-h_\ell(\mathbf{x}_i) y_{i,\ell}\big) \tag{3.15}$$

is the *base objective*. This means that the goal of the base learner in iteration t is to minimize $Z\big(\mathbf{h}, \mathbf{W}^{(t)}\big)$ in h. Minimizing the base objective (3.15) in each iteration is therefore equivalent to minimizing the weighted multi-class exponential margin-based error (3.13) in an iterative greedy

ADABOOST.MH$(\mathbf{X}, \mathbf{Y}, \mathbf{W}, \text{BASE}(\cdot,\cdot,\cdot), T)$

1 $\quad \mathbf{W}^{(1)} \leftarrow \frac{1}{n}\mathbf{W}$

2 $\quad$ **for** $t \leftarrow 1$ **to** T

3 $\quad\quad \left(\alpha^{(t)}, \mathbf{v}^{(t)}, \varphi^{(t)}(\cdot)\right) \leftarrow \text{BASE}\left(\mathbf{X}, \mathbf{Y}, \mathbf{W}^{(t)}\right)$

4 $\quad\quad \mathbf{h}^{(t)}(\cdot) \leftarrow \alpha^{(t)}\mathbf{v}^{(t)}\varphi^{(t)}(\cdot)$

5 $\quad\quad$ **for** $i \leftarrow 1$ **to** n **for** $\ell \leftarrow 1$ **to** K

6 $\quad\quad\quad w_{i,\ell}^{(t+1)} \leftarrow w_{i,\ell}^{(t)} \underbrace{\frac{\exp\left(-h_\ell^{(t)}(\mathbf{x}_i)y_{i,\ell}\right)}{\sum_{i'=1}^{n}\sum_{\ell'=1}^{K} w_{i',\ell'}^{(t)} \exp\left(-h_{\ell'}^{(t)}(\mathbf{x}_{i'})y_{i',\ell'}\right)}}_{Z\left(\mathbf{h}^{(t)}, \mathbf{W}^{(t)}\right)}$

7 $\quad$ **return** $\mathbf{f}^{(T)}(\cdot) = \sum_{t=1}^{T} \mathbf{h}^{(t)}(\cdot)$

Figure 3.5: The pseudocode of the ADABOOST.MH algorithm. $\mathbf{X}$ is the $n \times d$ observation matrix, $\mathbf{Y}$ is the $n \times K$ label matrix, $\mathbf{W}$ is the user-defined weight matrix used in the definition of the weighted Hamming error (3.10) and the weighted exponential margin-based error (3.13), $\text{BASE}(\cdot,\cdot,\cdot)$ is the base learner algorithm, and T is the number of iterations. $\alpha^{(t)}$ is the base coefficient, $\mathbf{v}^{(t)}$ is the vote vector, $\varphi^{(t)}(\cdot)$ is the scalar base (weak) classifier, $\mathbf{h}^{(t)}(\cdot)$ is the vector-valued base classifier, and $\mathbf{f}^{(T)}(\cdot)$ is the final (strong) discriminant function.

fashion. Indeed,

$$\widehat{R}_{\text{EXP}}\left(\mathbf{f}^{(T)}, \mathbf{W}\right) = \frac{1}{n}\sum_{i=1}^{n}\sum_{\ell=1}^{K} w_{i,\ell} \exp\left(-f_\ell^{(T)}(\mathbf{x}_i)y_{i,\ell}\right) \tag{3.16}$$

$$= \sum_{i=1}^{n}\sum_{\ell=1}^{K} w_{i,\ell}^{(1)} \exp\left(-f_\ell^{(T)}(\mathbf{x}_i)y_{i,\ell}\right) \tag{3.17}$$

$$= Z\left(\mathbf{h}^{(1)}, \mathbf{W}^{(1)}\right) \sum_{i=1}^{n}\sum_{\ell=1}^{K} w_{i,\ell}^{(2)} \frac{\exp\left(-f_\ell^{(T)}(\mathbf{x}_i)y_{i,\ell}\right)}{\exp\left(-h_\ell^{(1)}(\mathbf{x}_i)y_{i,\ell}\right)} \tag{3.18}$$

$$\vdots$$

$$= \prod_{t=1}^{T} Z\left(\mathbf{h}^{(t)}, \mathbf{W}^{(t)}\right) \sum_{i=1}^{n}\sum_{\ell=1}^{K} w_{i,\ell}^{(T+1)} \frac{\exp\left(-f_\ell^{(T)}(\mathbf{x}_i)y_{i,\ell}\right)}{\prod_{t=1}^{T}\exp\left(-h_\ell^{(t)}(\mathbf{x}_i)y_{i,\ell}\right)} \tag{3.19}$$

$$= \prod_{t=1}^{T} Z\left(\mathbf{h}^{(t)}, \mathbf{W}^{(t)}\right) \sum_{i=1}^{n}\sum_{\ell=1}^{K} w_{i,\ell}^{(T+1)} \tag{3.20}$$

$$= \prod_{t=1}^{T} Z\left(\mathbf{h}^{(t)}, \mathbf{W}^{(t)}\right). \tag{3.21}$$

In (3.16) we use the definition of the weighted exponential margin-based error (3.13). (3.17) follows from the weight initialization of line 1. In (3.18)-(3.19) we repeatedly apply the weight update formula of line 6. In (3.20) we use the definition (3.14). Finally, (3.21) follows from the fact that, by lines 1 and 6, the weight matrix $\mathbf{W}^{(t)}$ remains normalized

$$\sum_{i=1}^{n}\sum_{\ell=1}^{K} w_{i,\ell}^{(t)} = 1$$

in each iteration t.

In binary ADABOOST (Figure 3.3) with binary base classifiers $h(\mathbf{x}) \in \{\pm 1\}$ and coefficients α, (3.15) simplifies to

$$\begin{aligned} Z(h,\alpha) &= \sum_{i=1}^{n} w_i \exp\left(-\alpha h(\mathbf{x}_i) y_i\right) \\ &= \sum_{i=1}^{n} w_i \mathbb{I}\left\{h(\mathbf{x}_i) = y_i\right\} \exp(-\alpha) + \sum_{i=1}^{n} w_i \mathbb{I}\left\{h(\mathbf{x}_i) \neq y_i\right\} \exp(\alpha) \\ &= (1-\epsilon)\exp(-\alpha) + \epsilon \exp(\alpha), \end{aligned} \tag{3.22}$$

where

$$\epsilon = \epsilon(h, \mathbf{w}) = \sum_{i=1}^{n} w_i \mathbb{I}\left\{h(\mathbf{x}_i) \neq y_i\right\}$$

is the weighted error of the base classifier h. It is easy to see that for a given h, $Z\big(h,\alpha\big)$ is minimized by

$$\alpha = \frac{1}{2}\log\frac{1-\epsilon}{\epsilon} = \frac{1}{2}\log\frac{1+\gamma}{1-\gamma}. \tag{3.23}$$

where

$$\gamma = \gamma(h, \mathbf{w}) = 1 - 2\epsilon = \sum_{i=1}^{n} w_i h(\mathbf{x}_i) y_i$$

is the so called *edge* of the base classifier $h(\mathbf{x}_i)$. Plugging (3.23) back into (3.22), we have

$$Z(h,\alpha) = (1-\epsilon)\sqrt{\frac{\epsilon}{1-\epsilon}} + \epsilon\sqrt{\frac{1-\epsilon}{\epsilon}} = 2\sqrt{\epsilon(1-\epsilon)} = \sqrt{1-\gamma^2}.$$

Minimizing $Z\big(h,\alpha\big)$ is therefore equivalent to maximizing the edge γ or minimizing the weighted error ϵ. The chain of minimizing the exponential error $\widehat{R}_{\text{EXP}}\big(f^{(T)}\big) \rightarrow$ minimizing the base objective $Z(h,\alpha) \rightarrow$ minimizing the weighted error $\epsilon(h,\mathbf{w}) \rightarrow$ setting α to (3.23) explains the formulas of binary ADABOOST that might have been arbitrary-looking at the first sight. It is also easy to see that if

$$\gamma^{(t)} \triangleq \gamma\big(h^{(t)}, \mathbf{w}^{(t)}\big) \geq \delta > 0$$

(or, equivalently, if $\epsilon^{(t)} < \frac{1}{2} - \frac{\delta}{2}$), then the binary margin-based exponential error can be bounded from above by

$$\widehat{R}_{\text{EXP}}\big(f^{(T)}\big) \leq \sqrt{1-\delta^2}^{\,T} \leq \exp\left(-\frac{\delta^2 T}{2}\right), \tag{3.24}$$

where the second inequality follows from $1 - x \leq \exp(-x)$. This means that $\widehat{R}_{\text{EXP}}\big(f^{(T)}\big)$ becomes smaller than $\frac{1}{n}$ after at most

$$T^* = \left\lceil \frac{2\log n}{\delta^2} \right\rceil + 1$$

iterations. Since $\widehat{R}_{\text{EXP}}\big(f^{(T)}\big)$ is an upper bound of the training error $\widehat{R}_{\mathbb{I}}\big(f^{(T)}\big)$, and since the smallest non-zero value of $\widehat{R}_{\mathbb{I}}\big(f^{(T)}\big)$ is $\frac{1}{n}$, after T^* iterations $f^{(T)}$ cannot commit any error on the data set $\mathcal{D}$. To summarize, if the base classifiers $h^{(t)}$ are slightly better than a random guess, the final classifier $f^{(T)}$ has zero training error in a number of steps that is logarithmic in the size n of the data set $\mathcal{D}$.

In the general multi-class setup, base learning is more complicated (although still elementary) so we devote a separate section for the technical details (Section 3.4). But even without understanding how $\mathbf{h}$ is found, it can be seen, similarly to the binary case, that if $Z\big(\mathbf{h}, \mathbf{W}^{(t)}\big) \leq \sqrt{1-\delta^2}$,

then the exponential error $\widehat{R}_{\text{EXP}}(\mathbf{f}^{(T)}, \mathbf{W})$ (3.13) is also bounded above by (3.24). The smallest nonzero multi-class Hamming loss $\widehat{R}_{\text{H}}(\mathbf{f}, \mathbf{W})$ (3.10) is $w_{\min}/n$ where

$$w_{\min} = \min_{i,\ell: w_{i,\ell} \neq 0} w_{i,\ell}$$

is the smallest weight in the initial weight matrix $\mathbf{W}$, so $\widehat{R}_{\text{H}}(\mathbf{f}, \mathbf{W})$ will be zero after at most

$$T^* = \left\lceil \frac{2 \log(n/w_{\min})}{\delta^2} \right\rceil + 1$$

iterations. If $\mathbf{W}$ is set to (3.11), then $w_{\min} = \frac{1}{K}$, so

$$T^* = \left\lceil \frac{2 \log(nK)}{\delta^2} \right\rceil + 1, \tag{3.25}$$

whereas if $\mathbf{W}$ is set to (3.12), then $w_{\min} = \frac{1}{2(K-1)}$, so

$$T^* = \left\lceil \frac{2 \log\left(2n(K-1)\right)}{\delta^2} \right\rceil + 1.$$

We showed at the end of Section 3.3.1 that the one-error $\widehat{R}_{\text{I}}(\mathbf{f})$ (3.9) can be bounded from above by $K\widehat{R}_{\text{H}}(\mathbf{f}, \mathbf{W})$. This bound implies the same limit (3.25) as minimum weight argument. On the other hand, it can be shown that with the single-label initialization (3.12),

$$\widehat{R}_{\text{I}}\left(\mathbf{f}^{(T)}\right) \leq \sqrt{K-1} \prod_{t=1}^{T} Z\left(\mathbf{h}^{(t)}, \mathbf{W}^{(t)}\right)$$

which implies the limit

$$T^* = \left\lceil \frac{2 \log\left(n\sqrt{K-1}\right)}{\delta^2} \right\rceil + 1.$$

This result is another motivation for setting the weights to (3.12). Note, however, that these bounds have very few practical implications: on the one hand, the training error usually becomes 0 much before T^*, and on the other hand, ADABOOST is usually trained much longer after the training error becomes 0.

3.4 Multi-class base learning

The goal of multi-class base learning is to minimize $Z(\mathbf{h}, \mathbf{W})$ (3.15). In general, any vector-valued multi-class learning algorithm can be used here. Although this goal is clearly defined in [45], efficient base learning algorithms have never been described in detail. The algorithms are elementary: they are mostly based on exhaustive or greedy search, but some of the implementation details are somewhat tricky.[9]

In this section we will describe base learning algorithms that look for $\mathbf{h}(\mathbf{x})$ in the form of

$$\mathbf{h}(\mathbf{x}) = \alpha \mathbf{v} \varphi(\mathbf{x}), \tag{3.26}$$

where $\alpha \in \mathbb{R}^+$ is a positive real valued *base coefficient*, $\mathbf{v}$ is a binary $\mathbf{v} \in \{\pm 1\}^K$ *vote* vector, and $\varphi(\mathbf{x}) : \mathcal{X} \to \{\pm 1\}$ is a binary classifier. This setup is known as *discrete* ADABOOST.MH. The setup can be extended to real-valued classifiers $\varphi(\mathbf{x}) : \mathcal{X} \to \mathbb{R}$, also known as *confidence-rated*

[9] All the algorithms are implemented in our free software at http://multiboost.org, but we have not realized that our algorithms are mostly new, so, apart from the INDICATORBASE and PRODUCTBASE learners [46] this is the first document that describes them formally.

classifiers or, simply, discriminant functions, and it is also easy to make the vote vector $\mathbf{v}$ real-valued (in which case, without the loss of generality α would be set to 1). Both variants are known under the name of *real* ADABOOST.MH. Although there might be slight differences in the practical performance of real and discrete ADABOOST.MH, here we decided to stick to the discrete case for the sake of simplicity.

To start, in Section 3.4.1 we show how to set α and $\mathbf{v}$ in general if the scalar base classifier φ is given. Some of the elements defined here will be reused throughout the section. In Section 3.4.2 and Section 3.4.3 we describe two simple base classifiers used for numerical and nominal features, respectively. In practice, boosting these simple learners has often been found to be sub-optimal, mainly due to the lack of capacity of the resulted strong classifier (large approximation error). The most common solution for overcoming this problem is to use trees as base learners that call the simple base learner in a recursive fashion to partition the input space. We describe a basic tree learner adapted to the multi-class setup of ADABOOST.MH in Section 3.4.4. In Section 3.4.5 we introduce another meta-base learner that combines simple base learners in a multiplicative fashion. We conclude the section by describing some experimental results in Section 3.4.6.

3.4.1 Casting the votes

The intuitive semantics of (3.26) is the following. The binary classifier $\varphi(\mathbf{x})$ cuts the input space into a positive and a negative region. In binary classification this is the end of the story: we need $\varphi(\mathbf{x})$ to be well-correlated with the binary class labels y. In multi-class classification it is possible that $\varphi(\mathbf{x})$ correlates with some of the class labels y_ℓ and anti-correlates with others. This free choice is expressed by the binary votes $v_\ell \in \{\pm 1\}$. We say that $\varphi(\mathbf{x})$ votes *for* class ℓ if $v_\ell = +1$ and it votes *against* class ℓ if $v_\ell = -1$. As in binary classification, α expresses the overall quality of the classifier $\mathbf{v}\varphi(\mathbf{x})$: α is monotonically decreasing with respect to the weighted error of $\mathbf{v}\varphi(\mathbf{x})$ (or, equivalently, monotonically increasing with respect to the edge of $\mathbf{v}\varphi(\mathbf{x})$).

The advantage of the setup is that, given the binary classifier $\varphi(\mathbf{x})$, the optimal vote vector $\mathbf{v}$ and the coefficient α can be set in an efficient way. To see this, first let us define the *weighted per-class error rate*

$$\mu_{\ell -} = \sum_{i=1}^{n} w_{i,\ell} \mathbb{I}\left\{\varphi(\mathbf{x}_i) \neq y_{i,\ell}\right\}, \tag{3.27}$$

and the *weighted per-class correct classification rate*

$$\mu_{\ell +} = \sum_{i=1}^{n} w_{i,\ell} \mathbb{I}\left\{\varphi(\mathbf{x}_i) = y_{i,\ell}\right\} \tag{3.28}$$

for each class $\ell = 1, \ldots, K$. With this notation, $Z(\mathbf{h}, \mathbf{W})$ simplifies to

$$\begin{aligned} Z(\mathbf{h}, \mathbf{W}) &= \sum_{i=1}^{n} \sum_{\ell=1}^{K} w_{i,\ell} \exp\left(-h_\ell(\mathbf{x}_i) y_{i,\ell}\right) = \sum_{i=1}^{n} \sum_{\ell=1}^{K} w_{i,\ell} \exp\left(-\alpha v_\ell \varphi(\mathbf{x}_i) y_{i,\ell}\right) && (3.29) \\ &= \sum_{i=1}^{n} \sum_{\ell=1}^{K} \left(w_{i,\ell} \mathbb{I}\left\{v_\ell \varphi(\mathbf{x}_i) y_{i,\ell} = 1\right\} e^{-\alpha} + w_{i,\ell} \mathbb{I}\left\{v_\ell \varphi(\mathbf{x}_i) y_{i,\ell} = -1\right\} e^{\alpha} \right) \\ &= \sum_{\ell=1}^{K} \left(\mu_{\ell+} \mathbb{I}\left\{v_\ell = +1\right\} + \mu_{\ell-} \mathbb{I}\left\{v_\ell = -1\right\}\right) e^{-\alpha} \\ &\quad + \sum_{\ell=1}^{K} \left(\mu_{\ell-} \mathbb{I}\left\{v_\ell = +1\right\} + \mu_{\ell+} \mathbb{I}\left\{v_\ell = -1\right\}\right) e^{\alpha} && (3.30) \\ &= \sum_{\ell=1}^{K} \left(\mathbb{I}\left\{v_\ell = +1\right\} \left(e^{-\alpha} \mu_{\ell+} + e^{\alpha} \mu_{\ell-}\right) + \mathbb{I}\left\{v_\ell = -1\right\} \left(e^{-\alpha} \mu_{\ell-} + e^{\alpha} \mu_{\ell+}\right) \right) \\ &= \sum_{\ell=1}^{K} \left(\frac{1 + v_\ell}{2} \left(e^{-\alpha} \mu_{\ell+} + e^{\alpha} \mu_{\ell-}\right) + \frac{1 - v_\ell}{2} \left(e^{-\alpha} \mu_{\ell-} + e^{\alpha} \mu_{\ell+}\right) \right) \\ &= \frac{1}{2} \sum_{\ell=1}^{K} \left(\left(e^{\alpha} + e^{-\alpha}\right) \left(\mu_{\ell+} + \mu_{\ell-}\right) - v_\ell \left(e^{\alpha} - e^{-\alpha}\right) \left(\mu_{\ell+} - \mu_{\ell-}\right) \right) \\ &= \frac{e^{\alpha} + e^{-\alpha}}{2} - \frac{e^{\alpha} - e^{-\alpha}}{2} \sum_{\ell=1}^{K} v_\ell \left(\mu_{\ell+} - \mu_{\ell-}\right). && (3.31) \end{aligned}$$

(3.29) comes from the definition (3.26) of $\mathbf{h}$ and (3.30) follows from the definitions (3.27) and (3.28) of $\mu_{\ell-}$ and $\mu_{\ell+}$. In the final step (3.31) we used the fact that

$$\sum_{\ell=1}^{K} \left(\mu_{\ell+} + \mu_{\ell-}\right) = \sum_{i=1}^{n} \sum_{\ell=1}^{K} w_{i,\ell} = 1.$$

The quantity

$$\gamma_\ell = v_\ell \left(\mu_{\ell+} - \mu_{\ell-}\right) = \sum_{i=1}^{n} w_{i,\ell} v_\ell \varphi(\mathbf{x}_i) y_{i,\ell} \tag{3.32}$$

is called the *classwise edge* of $\mathbf{h}(\mathbf{x})$ with $\boldsymbol{\gamma}$ denoting the vector $(\gamma_1, \ldots, \gamma_K)$. The full multi-class edge of the classifier is then

$$\gamma = \gamma(\mathbf{v}, \varphi, \mathbf{W}) = \|\boldsymbol{\gamma}\|_1 = \sum_{\ell=1}^{K} \gamma_\ell = \sum_{\ell=1}^{K} v_\ell \left(\mu_{\ell+} - \mu_{\ell-}\right) = \sum_{i=1}^{n} \sum_{\ell=1}^{K} w_{i,\ell} v_\ell \varphi(\mathbf{x}_i) y_{i,\ell}. \tag{3.33}$$

With this notation, the optimal binary coefficient α (3.23) is recovered: it is easy to see that (3.31) is minimized when

$$\alpha = \frac{1}{2} \log \frac{1 + \gamma}{1 - \gamma}. \tag{3.34}$$

With this optimal coefficient, (3.31) becomes

$$Z(\mathbf{h}, \mathbf{W}) = \sqrt{1 - \gamma^2},$$

so $Z(\mathbf{h}, \mathbf{W})$ is minimized when γ is maximized. From (3.33) it then follows that $Z(\mathbf{h}, \mathbf{W})$ is minimized if v_ℓ agrees with the sign of $\left(\mu_{\ell+} - \mu_{\ell-}\right)$, that is,

$$v_\ell = \begin{cases} 1 & \text{if } \mu_{\ell+} > \mu_{\ell-} \\ -1 & \text{otherwise} \end{cases} \tag{3.35}$$

for all classes $\ell = 1, \dots, K$.

The setup of vector-valued based classification has another important consequence: the preservation of the weak-learning condition. Indeed, if $\varphi(\mathbf{x})$ is slightly better then a coin toss (in maximizing the edge), γ will be positive. Another way to look at it is to say that if a $(\varphi, \mathbf{v})$ combo has a negative edge $-\gamma < 0$, the edge of its *complement* (either $(-\varphi, \mathbf{v})$ or $(\varphi, -\mathbf{v})$) will be $\gamma > 0$. To understand the significance of this, consider a classical single-label base classifier $h : \mathcal{X} \to \mathcal{L} = \{1, \dots, K\}$, required by ADABOOST.M1. Now if $h(\mathbf{x})$ is slightly better than a coin toss, all one can hope for is an error rate slightly lower than $\frac{K-1}{K}$. To achieve the error of $\frac{1}{2}$, required for continuing the boosting iterations, one has to come up with a base learner which is significantly better than a coin toss.

Finally, note that computing $\mathbf{v}$ and α is a $\Theta(nK)$ operation which may be prohibitive if we have to do it for every base classifier φ in an exhaustive search. Fortunately, it turns out that the two simple base classifiers STUMPBASE (Section 3.4.2) and INDICATORBASE (Section 3.4.3) can make cheap updates to the edge (3.33) while doing the exhaustive search, keeping the time complexity of *full* base learning at $\Theta(nK)$.

3.4.2 Decision stumps for numerical features

The simplest scalar base learner used in practice on numerical features is the *decision stump*, a one-decision two-leaf decision tree of the form

$$\varphi_{j,b}(\mathbf{x}) = \begin{cases} 1 & \text{if } x^{(j)} \geq b, \\ -1 & \text{otherwise,} \end{cases}$$

where j is the index of the selected feature and b is the decision threshold. If the feature values $(x_1^{(j)}, \dots, x_n^{(j)})$ are pre-ordered before the first boosting iteration, a decision stump maximizing the edge (3.33) (or minimizing the energy (3.29)[10]) can be found very efficiently in $\Theta(ndK)$ time.

The pseudocode of the algorithm is given in Figure 3.6. STUMPBASE first calculates the edge vector $\boldsymbol{\gamma}^{(0)}$ of the constant classifier $\mathbf{h}^{(0)}(\mathbf{x}) \equiv \mathbf{1}$ which will serve as the initial edge vector for each featurewise edge-maximizer. Then it loops over the features, calls BESTSTUMP to return the best featurewise stump, and then selects the best of the best by minimizing the energy (3.29). BESTSTUMP loops over all (sorted) feature values $s_1, \dots, s_{n-1}$. It considers all thresholds b halfway between two non-identical feature values $s_i \neq s_{i+1}$. The main trick (and, at the same time, the bottleneck of the algorithm) is the update of the classwise edges in lines 4-5: when the threshold moves from $b = \frac{s_{i-1}+s_i}{2}$ to $b = \frac{s_i+s_{i+1}}{2}$, the classwise edge γ_ℓ of $\mathbf{1}\varphi(\mathbf{x})$ (that is, $\mathbf{v}\varphi(\mathbf{x})$ with $\mathbf{v} = \mathbf{1}$) can only change by $\pm w_{i,\ell}$, depending on the sign $y_{i,\ell}$ (Figure 3.7). The total edge of $\mathbf{v}\varphi(\mathbf{x})$ with optimal votes (3.35) is then the sum of the absolute values of the classwise edges of $\mathbf{1}\varphi(\mathbf{x})$ (line 7).

3.4.3 Subset indicators for nominal features

Nominal features $x_i^{(j)} \in \mathcal{I}^{(j)} = \{1, \dots, M^{(j)}\}$ are as abundant in practice as numerical features, and they need to be treated differently at the base learning. Their main feature is that they cannot be ordered naturally; they share this feature with the class variable ℓ. The textbook example is color, e.g., $\mathcal{I}^{(j)} = \{$, , , , $\}$, but applications range from text processing (word- or tag-valued features) to collaborative filtering (movie or user indices).

The classical way to handle nominal features with numerical learners is to encode a feature with $M^{(j)}$ possible values into $M^{(j)}$ binary (for example, $\{0,1\}$-valued) features. Learning a decision stump on such encoding means that each base classifiers selects one value $\iota \in \mathcal{I}^{(j)}$, and

[10] Note the distinction: for full binary $\mathbf{v}$ the two are equivalent, but for ternary or real valued $\mathbf{v}$ and/or real valued $\phi(\mathbf{x})$ they are not. In Figure 3.6 we are maximizing the edge within each feature (line 7 in BESTSTUMP) but across features we are minimizing the energy (line 7 in STUMPBASE). Updating the energy inside the inner loop (line 4) could not be done in $\Theta(K)$ time.

StumpBase$(\mathbf{X}, \mathbf{Y}, \mathbf{W})$

1 **for** $\ell \leftarrow 1$ **to** K ▷ *for all classes*

2 $\quad \gamma_\ell^{(0)} \leftarrow \sum_{i=1}^{n} w_{i,\ell} y_{i,\ell}$ ▷ *classwise edges (3.32) of constant classifier* $\mathbf{h}^{(0)}(\mathbf{x}) \equiv \mathbf{1}$

3 **for** $j \leftarrow 1$ **to** d ▷ *all (numerical) features*

4 $\quad \mathbf{s} \leftarrow \text{Sort}(x_1^{(j)}, \ldots, x_n^{(j)})$ ▷ *sort the jth column of* $\mathbf{X}$

5 $\quad (\mathbf{v}_j, b_j, \gamma_j) \leftarrow \text{BestStump}(\mathbf{s}, \mathbf{Y}, \mathbf{W}, \gamma^{(0)})$ ▷ *best stump per feature*

6 $\quad \alpha_j \leftarrow \frac{1}{2} \log \frac{1+\gamma_j}{1-\gamma_j}$ ▷ *base coefficient (3.34)*

7 $j^* \leftarrow \arg\min_j Z(\alpha_j \mathbf{v}_j \varphi_{j,b_j}, \mathbf{W})$ ▷ *best stump across features*

8 **return** $(\alpha_{j^*}, \mathbf{v}_{j^*}, \varphi_{j^*,b_{j^*}}(\cdot))$

BestStump$(\mathbf{s}, \mathbf{Y}, \mathbf{W}, \gamma^{(0)})$

1 $\gamma^* \leftarrow \gamma^{(0)}$ ▷ *best edge vector*

2 $\gamma \leftarrow \gamma^{(0)}$ ▷ *initial edge vector*

3 **for** $i \leftarrow 1$ **to** $n-1$ ▷ *for all points in order* $s_1 \leq \ldots \leq s_{n-1}$

4 $\quad$ **for** $\ell \leftarrow 1$ **to** K ▷ *for all classes*

5 $\quad\quad \gamma_\ell \leftarrow \gamma_\ell - 2 w_{i,\ell} y_{i,\ell}$ ▷ *update classwise edges of stump with* $\mathbf{v} = \mathbf{1}$

6 $\quad$ **if** $s_i \neq s_{i+1}$ **then** ▷ *no threshold if identical coordinates* $s_i = s_{i+1}$

7 $\quad\quad$ **if** $\sum_{\ell=1}^K |\gamma_\ell| > \sum_{\ell=1}^K |\gamma_\ell^*|$ **then** ▷ *found better stump*

8 $\quad\quad\quad \gamma^* \leftarrow \gamma$ ▷ *update best edge vector*

9 $\quad\quad\quad b^* \leftarrow \frac{s_i + s_{i+1}}{2}$ ▷ *update best threshold*

10 **for** $\ell \leftarrow 1$ **to** K ▷ *for all classes*

11 $\quad v_\ell^* \leftarrow \text{sign}(\gamma_\ell)$ ▷ *set vote vector according to (3.35)*

12 **if** $\gamma^* = \gamma^{(0)}$ ▷ *did not beat the constant classifier*

13 $\quad$ **return** $(\mathbf{v}^*, -\infty, \|\gamma^*\|_1)$ ▷ *constant classifier with optimal votes*

14 **else**

15 $\quad$ **return** $(\mathbf{v}^*, b^*, \|\gamma^*\|_1)$ ▷ *best stump*

Figure 3.6: Exhaustive search for the best decision stump. BestStump receives a sorted column (feature) $\mathbf{s}$ of the observation matrix $\mathbf{X}$. The sorting in line 4 can be done once for all features outside of the boosting loop. BestStump examines all thresholds b halfway between two non-identical coordinates $s_i \neq s_{i+1}$ and returns the threshold b^* and vote vector $\mathbf{v}^*$ that maximizes the edge $\gamma(\mathbf{v}, \varphi_{j,b}, \mathbf{W})$. StumpBase then sets the coefficient α_j according to (3.34) and stump across features that minimizes the energy (3.15).

separates observations with $x^{(j)} = \iota$ from $x^{(j)} \neq \iota$. We call this the SelectorBase learner, and define it formally as

$$\varphi_{j,\iota}(\mathbf{x}) = \begin{cases} 1 & \text{if } x^{(j)} = \iota, \\ -1 & \text{otherwise.} \end{cases} \tag{3.36}$$

Using our example, the base learner that selects the blue value can be represented by $\varphi_{j,\blacksquare} =$

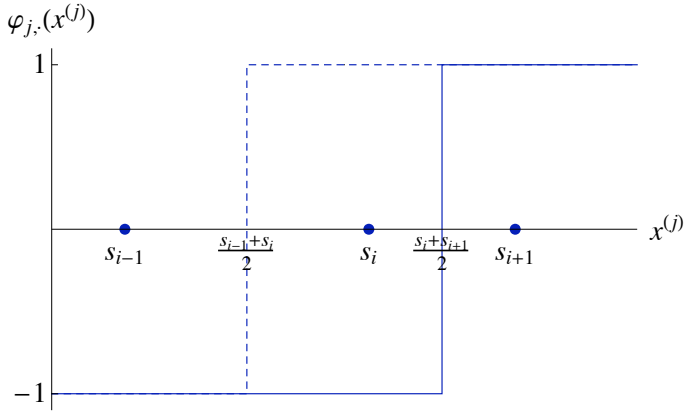


Figure 3.7: Updating the edge γ_ℓ in line 5 of BESTSTUMP. If $y_{i,\ell} = 1$, then γ_ℓ decreases by $2w_{i,\ell}$, and if $y_i = -1$, then γ_ℓ increases by $2w_{i,\ell}$.

$\{-, +, -, -, -\}$.

Optimizing this SELECTORBASE takes $\Theta(ndK + K\Sigma)$ time, where $\Sigma = \sum_{j=1}^{d} M^{(j)}$ is the number of all the different values of all features $j = 1, \ldots, d$. Hence, to obtain a strong learner that potentially uses (tests) $\Omega(M^{(j)})$ values for each feature, we will need $\Omega\big((ndK + K\Sigma)\Sigma\big)$ operations, which can be prohibitive if Σ is large. On the other hand, it turns out that we can optimize a base learner that makes a decision on *all* values of $\mathcal{I}^{(j)}$ in each iteration, using essentially the same number $\Theta(ndK + K\Sigma)$ of operations per iteration as the selector base learner. Formally, we consider base learners of the form

$$\varphi_{j,\mathbf{u}}(\mathbf{x}) = \mathbf{u}_{x^{(j)}}, \tag{3.37}$$

where $\mathbf{u}$ is a $\{\pm 1\}$-valued vector over the index set $\mathcal{I}^{(j)}$. For example, the base classifier

$$\varphi_{j,\{+1,-1,-1,+1,+1\}} = \{+, -, -, +, +\}$$

outputs $+1$ for observations $\mathbf{x}$ whose jth feature is , , or , and -1 for feature values or .

Learning a subset indicator is essentially a rank-1 matrix factorization problem. For showing this, we first construct the $M^{(j)} \times K$-dimensional *edge matrix* $\mathbf{\Gamma}^{(j)} = \big[\gamma_{\iota,\ell}^{(j)}\big]$ with elements

$$\gamma_{\iota,\ell}^{(j)} = \sum_{i=1}^{n} \mathbb{I}\left\{x_i^{(j)} = \iota\right\} w_{i,\ell} y_{i,\ell} \tag{3.38}$$

for each (nominal) feature $j = 1, \ldots, d$, label index $\ell = 1, \ldots, K$ and feature value $\iota = \{1, \ldots, M^{(j)}\}$.

The edge of $\varphi_{j,\mathbf{u}}(\mathbf{x})$ with vote vector $\mathbf{v}$ is

$$\begin{aligned}
\gamma(\mathbf{v}, \varphi_{j,\mathbf{u}}, \mathbf{W}) &= \sum_{i=1}^{n}\sum_{\ell=1}^{K} w_{i,\ell} v_\ell \varphi_{j,\mathbf{u}}(\mathbf{x}_i) y_{i,\ell} && (3.39)\\
&= \sum_{i=1}^{n}\sum_{\ell=1}^{K} w_{i,\ell} v_\ell u_{x_i^{(j)}} y_{i,\ell} && (3.40)\\
&= \sum_{\ell=1}^{K} v_\ell \sum_{i=1}^{n} w_{i,\ell} u_{x_i^{(j)}} y_{i,\ell} && (3.41)\\
&= \sum_{\ell=1}^{K} v_\ell \sum_{i=1}^{n}\sum_{\iota=1}^{M} w_{i,\ell} \mathbb{I}\left\{x_i^{(j)} = \iota\right\} u_\iota y_{i,\ell} && (3.42)\\
&= \sum_{\ell=1}^{K} v_\ell \sum_{\iota=1}^{M} u_\iota \sum_{i=1}^{n} \mathbb{I}\left\{x_i^{(j)} = \iota\right\} w_{i,\ell} y_{i,\ell} && (3.43)\\
&= \sum_{\ell=1}^{K}\sum_{\iota=1}^{M} v_\ell u_\iota \gamma_{\iota,\ell}^{(j)}. && (3.44)
\end{aligned}$$

(3.39) comes from the definition (3.33) of the multi-class edge $\gamma(\mathbf{v}, \varphi_{j,\mathbf{u}}, \mathbf{W})$. In (3.40) we used the transcription (3.37) of $\varphi_{j,\mathbf{u}}(\mathbf{x})$. In (3.41) and (3.43) we simply re-ordered the summations. In (3.42) we used the fact that

$$u_{x_i^{(j)}} = \sum_{\iota=1}^{M} \mathbb{I}\left\{x_i^{(j)} = \iota\right\} u_\iota.$$

Finally, (3.44) follows from the definition (3.38) of the edge matrix $\mathbf{\Gamma}^{(j)}$. Introducing the elementwise matrix multiplication operator $\otimes$, the edge of $\varphi_{j,\mathbf{u}}(\mathbf{x})$ with vote vector $\mathbf{v}$ is

$$\gamma(\mathbf{v}, \varphi_{j,\mathbf{u}}, \mathbf{W}) = \left\| \mathbf{\Gamma}^{(j)} \otimes \mathbf{v}\mathbf{u}^\intercal \right\|_1,$$

where $\mathbf{v}\mathbf{u}^\intercal$ is the outer product of $\mathbf{v}$ and $\mathbf{u}$ and $\|\cdot\|_1$ is the matrix 1-norm (sum of elements). Since

$$v_\ell u_\iota \gamma_{\iota,\ell}^{(j)} = |\gamma_{\iota,\ell}^{(j)}| \left(1 - 2\mathbb{I}\left\{v_\ell u_\iota \neq \operatorname{sign}(\gamma_{\iota,\ell}^{(j)})\right\}\right),$$

maximizing $\gamma(\mathbf{v}, \varphi_{j,\mathbf{u}}, \mathbf{W})$ is equivalent to finding $\mathbf{u}$ and $\mathbf{v}$ that minimize

$$\frac{\sum_{\ell=1}^{K}\sum_{\iota=1}^{M} |\gamma_{\iota,\ell}^{(j)}| - \gamma(\mathbf{v}, \varphi_{j,\mathbf{u}}, \mathbf{W})}{2} = \sum_{\ell=1}^{K}\sum_{\iota=1}^{M} |\gamma_{\iota,\ell}^{(j)}| \mathbb{I}\left\{v_\ell u_\iota \neq \operatorname{sign}(\gamma_{\iota,\ell}^{(j)})\right\},$$

which is a weighted (binary) one-error. Therefore, in a nutshell, maximizing the edge in $\mathbf{u}$ and $\mathbf{v}$ is equivalent to a rank-1 matrix factorization with an elementwise weighted one-loss.

Whatever beautiful name we give it, maximizing the edge $\gamma(\mathbf{v}, \varphi_{j,\mathbf{u}}, \mathbf{W})$ is a difficult combinatorial optimization problem. Fortunately, ADABOOST.MH does not require the maximization of the edge: we only need and edge $\gamma^{(t)} > 0$ in each boosting iteration t to be able to continue. The BESTINDICATOR[11] algorithm (Figure 3.8) carries out a local search for a good $(\mathbf{u}, \mathbf{v})$ pair. It is easy to see that the optimal $\mathbf{v}$ for a fixed $\mathbf{u}$ is

$$v_\ell = \operatorname{sign}\left(\sum_{\iota=1}^{M^{(j)}} \gamma_{\iota,\ell}^{(j)} u_\iota\right), \qquad (3.45)$$

[11] The name comes from the fact that $\mathbf{u}$ is a binary vector that formally indicates a subset of $\mathcal{I}$.

and, symmetrically, the optimal $\mathbf{u}$ for a fixed $\mathbf{v}$ is

$$u_\iota = \operatorname{sign}\left(\sum_{\ell=1}^{K} \gamma_{\iota,\ell}^{(j)} v_\ell\right). \tag{3.46}$$

After computing $\mathbf{\Gamma}^{(j)}$ (lines 2-4) and initializing $\mathbf{u}$ to a random $\{\pm 1\}$-valued vector (line 6), BESTINDICATOR iterates between (3.45) and (3.46) until no change in the edge. Convergence is guaranteed since $\gamma(\mathbf{v}, \varphi_{j,\mathbf{u}}, \mathbf{W})$ is bounded from above by 1, it must increase in each iteration, and the increase is bounded away from zero because the weights $w_{i,\ell}$ are bounded away from zero. In practice BESTINDICATOR always stops after a few iterations. Once it returns the pair $(\mathbf{u}_j, \mathbf{v}_j)$ for each feature j, INDICATORBASE, as STUMPBASE, takes care of setting the coefficient α_j and finding the best feature j^*.

Another advantage of INDICATORBASE over SELECTORBASE (besides being faster) is that the vote vector $\mathbf{v}$ is shared among the selectors while in the standard learner we learn a vote vector for each selector. This means that the capacity of the base classifier is considerably reduced (compared to the sum of Σ selectors), suggesting that the algorithm is less susceptible to overfitting. Sharing the vote vector may also help to "pick up" dependencies between feature values in the case where there is a structure in the feature space (e.g., using discrete ADABOOST.MH, each indicator $\varphi_{j,\mathbf{u}}$ clusters the feature values $\mathcal{I}^{(j)}$ into two groups).

In a broader sense, the proposed learner is related to Cohen's [86] model which allows set-valued features as well as the common real-valued and nominal features. The main difference is that in [86] it is the *features of the observations* that can take a subset of a large set of possible values but the decision functions are still *selectors* of the form (3.36), whereas what we propose here are subset-indicator *decision functions* (3.37). In another related algorithm (SLIPPER), Cohen and Singer [87] build rules that are *conjunctions of selectors* (3.36) using a growing/pruning procedure. The selectors in the same base rule can use different *features*, whereas INDICATORBASE acts on different *values* of *one* feature. Although superficially similar to our method, both approaches use different base classifiers and different algorithms to learn the classifiers. On the other hand, they can be incorporated into our approach with just a few modifications to INDICATORBASE.

3.4.4 Hamming trees

Classification trees [88] have been widely used for multivariate classification since the 80s. They are especially efficient when used as base learners in ADABOOST [89]. Their main disadvantage is their variance with respect to the training data, but when averaged over T different runs, this problem disappears. The reason of using trees is to increase the expressiveness (complexity) of base classifiers. For example, learning a linear combination of decision stumps on pixels of an image ignores completely the correlation between pixels. A small tree of even two nodes, on the other hand, can easily pick up this correlation.

The most commonly used tree learner is C4.5 [90]. Whereas this tree implementation is a perfect choice for binary ADABOOST, it is suboptimal for ADABOOST.MH since it outputs a single-label classifier with no guarantee of a positive multi-class edge (3.33). Although this problem can be solved in practice by building large trees, there is no reason why we should not optimize the multi-class edge using the tree learning framework.

The advantage of our formalization is that we can use any multi-class base classifier of the form (3.26) for the tree cuts. Formally, a binary classification tree with N leaves and $N-1$ inner nodes consists of a list of $N-1$ base classifiers $\mathfrak{H} = (\mathbf{h}_1, \ldots, \mathbf{h}_N)$ of the form

$$\mathbf{h}_j(\mathbf{x}) = \alpha_j \mathbf{v}_j \varphi_j(\mathbf{x}),$$

and two index lists $\mathfrak{l} = (\mathfrak{l}_1, \ldots, \mathfrak{l}_N)$ and $\mathfrak{r} = (\mathfrak{r}_1, \ldots, \mathfrak{r}_N)$ with $\mathfrak{l}, \mathfrak{r} \in (\mathbb{N} \cup \{\text{NULL}\})^N$. $\mathfrak{l}_j$ and $\mathfrak{r}_j$ represent the indices of the left and right children of the jth node of the tree, respectively. The

INDICATORBASE$(\mathbf{X}, \mathbf{Y}, \mathbf{W})$

1 **for** $j \leftarrow 1$ **to** d ▷ *all (nominal) features*
2 $(\mathbf{v}_j, \mathbf{u}_j, \gamma_j) \leftarrow$ BESTINDICATOR$(\mathbf{x}^{(j)}, \mathbf{Y}, \mathbf{W}, \mathcal{I}^{(j)})$ ▷ $\mathbf{x}^{(j)} \triangleq (x_1^{(j)}, \ldots, x_n^{(j)})$
3 $\alpha_j \leftarrow \frac{1}{2} \log \frac{1+\gamma_j}{1-\gamma_j}$ ▷ *base coefficient (3.34)*
4 $j^* \leftarrow \arg\min_j Z(\alpha_j \mathbf{v}_j \varphi_{j,\mathbf{u}_j}, \mathbf{W})$
5 **return** $(\alpha_{j^*}, \mathbf{v}_{j^*}, \varphi_{j^*,\mathbf{u}_{j^*}}(\cdot))$

BESTINDICATOR$(\mathbf{x}, \mathbf{Y}, \mathbf{W}, \mathcal{I})$

1 $M \leftarrow |\mathcal{I}|$ ▷ *number of feature values*
2 **for** $\ell \leftarrow 1$ **to** K ▷ *compute* $\boldsymbol{\Gamma}$ *(3.38)*
3 **for** $\iota \leftarrow 1$ **to** M $\gamma_{\iota,\ell} \leftarrow 0$
4 **for** $i \leftarrow 1$ **to** n $\gamma_{x_i,\ell} \leftarrow \gamma_{x_i,\ell} + w_{i,\ell} y_{i,\ell}$
5 $\gamma^* \leftarrow 0$
6 **for** $\iota \leftarrow 1$ **to** M $u_\iota \leftarrow$ RANDOM(± 1)
7 **while** TRUE
8 **for** $\ell \leftarrow 1$ **to** K ▷ *compute optimal* $\mathbf{v}$ *given* $\mathbf{u}$ *(3.45)*
9 $\gamma_\ell \leftarrow \sum_{\iota=1}^{M} \gamma_{\iota,\ell} u_\iota$
10 $v_\ell \leftarrow \operatorname{sign}(\gamma_\ell)$
11 **if** $\gamma^* \geq \sum_{\ell=1}^{K} |\gamma_\ell|$ **then return** $(\mathbf{u}, \mathbf{v}, \gamma^*)$
12 $\gamma^* \leftarrow \sum_{\ell=1}^{K} |\gamma_\ell|$
13 **for** $\iota \leftarrow 1$ **to** M ▷ *compute optimal* $\mathbf{u}$ *given* $\mathbf{v}$ *(3.46)*
14 $\gamma_\iota \leftarrow \sum_{\ell=1}^{K} \gamma_{\iota,\ell} v_\ell$
15 $u_\iota \leftarrow \operatorname{sign}(\gamma_\iota)$
16 **if** $\gamma^* \geq \sum_{\iota=1}^{M} |\gamma_\iota|$ **then return** $(\mathbf{u}, \mathbf{v}, \gamma^*)$
17 $\gamma^* \leftarrow \sum_{\iota=1}^{M} |\gamma_\iota|$

Figure 3.8: The pseudocode of the subset indicator base learner. After computing $\boldsymbol{\Gamma}^{(j)}$ (lines 2-4) and initializing $\mathbf{u}$ to a random $\{\pm 1\}$-valued vector (line 6), BESTINDICATOR iterates between (3.45) and (3.46) until no change in the edge. STUMPBASE, sets the coefficients α_j and finds the best feature j^*.

node classifier in the jth node is defined recursively as

$$\mathfrak{h}_j(\mathbf{x}) = \begin{cases} -\mathbf{v}_j & \text{if } \varphi_j(\mathbf{x}) = -1 \wedge \mathfrak{l}_j = \text{NULL} \quad \text{(left leaf)}, \\ \mathbf{v}_j & \text{if } \varphi_j(\mathbf{x}) = +1 \wedge \mathfrak{r}_j = \text{NULL} \quad \text{(right leaf)}, \\ \mathfrak{h}_{\mathfrak{l}_j}(\mathbf{x}) & \text{if } \varphi_j(\mathbf{x}) = -1 \wedge \mathfrak{l}_j \neq \text{NULL}, \quad \text{(left inner node)}, \\ \mathfrak{h}_{\mathfrak{r}_j}(\mathbf{x}) & \text{if } \varphi_j(\mathbf{x}) = +1 \wedge \mathfrak{r}_j \neq \text{NULL}, \quad \text{(right inner node)}. \end{cases} \tag{3.47}$$

The final tree classifier is then

$$\mathbf{h}_{\mathfrak{H},\mathfrak{l},\mathfrak{r}}(\mathbf{x}) = \alpha \mathfrak{h}_1(\mathbf{x}).$$

Note that $\mathbf{h}_{\mathfrak{H},\mathfrak{l},\mathfrak{r}}(\mathbf{x})$ does not exactly have the form of (3.26). In particular, $\mathbf{h}_{\mathfrak{H},\mathfrak{l},\mathfrak{r}}(\mathbf{x})$ uses the local vote vectors $\mathbf{v}_j$ determined by each leaf instead of a global vote vector. On the other hand, the coefficient α is unique, and it is determined in the standard way

$$\alpha = \frac{1}{2} \log \frac{1 + \gamma(\mathfrak{h}_1, \mathbf{W})}{1 - \gamma(\mathfrak{h}_1, \mathbf{W})}$$

based on the edge of the root classifier $\mathfrak{h}_1$. The local coefficients α_j returned by the base learners are thrown away.

Finding the optimal N-leaf tree is a difficult combinatorial problem. Most tree-building algorithms are therefore sub-optimal by construction. For ADABOOST, again, this is not a problem: we can continue boosting as long as the edge is positive. Classification trees are usually built in a greedy manner: at each stage we try to cut all the current leaves j by calling the base learner of the data points reaching the jth leaf, then select the best node to cut, convert the old leaf into an inner node, and add two new leaves. The difference between the different algorithms is in the way the best node is selected. Usually, we select the node that *improves* the most a gain function. In ADABOOST.MH the natural gain is the edge (3.33) of the base classifier. Since the data set $(\mathbf{X}, \mathbf{Y})$ is different in each node, we will include them explicitly in the argument of the full multi-class edge

$$\gamma(\mathbf{v}, \varphi, \mathbf{X}, \mathbf{Y}, \mathbf{W}) = \sum_{i=1}^{n} \sum_{\ell=1}^{K} \mathbb{I}\{x_i \in \mathbf{X}\} w_{i,\ell} v_\ell \varphi(\mathbf{x}_i) y_{i,\ell}.$$

Note that in this definition we do not require that the sum of the weights of the selected points add up to 1. Also note that this gain function is additive on subsets of the original data set $(\mathbf{X}, \mathbf{Y}, \mathbf{W})$, so the local edges in the leaves add up to the edge of the full tree. This means that any improvement in the local edge directly translates to improvement of the tree edge.

The basic operation when adding a tree node with a scalar classifier (cut) φ is to separate the data matrices $\mathbf{X}$, $\mathbf{Y}$, and $\mathbf{W}$ according to the sign of classification $\varphi(\mathbf{x}_i)$ for all $\mathbf{x}_i \in \mathbf{X}$. Figure 3.9 contains the pseudocode of this simple operation.

```
CUTDATASET(X, Y, W, φ(·))
 1   X₋ ← Y₋ ← W₋ ← X₊ ← Y₊ ← W₊ ← ()          ▷ empty vectors
 2   for i ← 1 to n
 3       if xᵢ ∈ X then
 4           if φ(xᵢ) = −1 then
 5               X₋ ← APPEND(X₋, xᵢ)
 6               Y₋ ← APPEND(Y₋, yᵢ)
 7               W₋ ← APPEND(W₋, wᵢ)
 8           else
 9               X₊ ← APPEND(X₊, xᵢ)
10               Y₊ ← APPEND(Y₊, yᵢ)
11               W₊ ← APPEND(W₊, wᵢ)
12   return (X₋, Y₋, W₋, X₊, Y₊, W₊)
```

Figure 3.9: The basic operation when adding a tree node is to separate the data matrices $\mathbf{X}$, $\mathbf{Y}$, and $\mathbf{W}$ according to the sign of classification $\varphi(\mathbf{x}_i)$ for all $\mathbf{x}_i \in \mathbf{X}$.

Building a tree is usually described in a recursive way but we find the iterative procedure easier to explain, so our pseudocode in Figure 3.10 contains this version. To main idea is to maintain a *priority queue*, a data structure that allows *inserting* objects with numerical *keys* into a set, and extracting the object with the *maximum* key [91]. The key will represent the improvement of the edge when cutting a leaf. We first call the base learner on the full data set (line 1) and insert it into the priority queue with its edge $\gamma(\mathbf{v}, \varphi, \mathbf{X}, \mathbf{Y}, \mathbf{W})$ (line 3) as the key. Then in each iteration, we extract the leaf that would provide the best edge improvement among all the leaves in the priority queue (line 7), we partition the data set (line 11), call the base learners on the two new leaves (line 12), and insert them into the priority queue using the difference between the old edge *on the partitioned data sets* and the new edges of the base classifiers in the two new leaves (line 13). When inserting a leaf into the queue, we also save the sign of the cut (left or right child) and the index of the parent, so the index vectors $\mathfrak{l}$ and $\mathfrak{r}$ can be set properly in line 8.

```
TREEBASE(X, Y, W, BASE(·,·,·), N)
 1  (α, v, φ(·)) ← BASE(X, Y, W)
 2  S ← PRIORITYQUEUE                ▷ O(log N) insertion and extraction of maximum key
 3  INSERT(S, (v, φ(·), X, Y, NULL, 0), γ(v, φ, X, Y, W))   ▷ key = edge γ
 4  𝔥 ← ()                           ▷ initialize classifier list
 5  for j ← 1 to N
 6      l_j ← r_j ← NULL             ▷ initialize child indices
 7      (v_j, φ_j(·), X_j, Y_j, •, j_P) ← EXTRACTMAX(S)   ▷ best node in the priority queue
 8      if • = − then l_{j_P} ← j else if • = + then r_{j_P} ← j   ▷ child index of parent
 9      𝔥 ← APPEND(𝔥, v_j φ_j(·))    ▷ adding h_j(·) = v_j φ_j(·) to 𝔥
10      (X_−, Y_−, W_−, X_+, Y_+, W_+) ← CUTDATASET(X_j, Y_j, W, φ_j(·))
11      for • ∈ {−, +}               ▷ insert children into priority queue
12          (α_•, v_•, φ_•(·)) ← BASE(X_•, Y_•, W_•)
13          INSERT(S, (v_•, φ_•(·), X_•, Y_•, •, j), γ(v_•, φ_•, X_•, Y_•, W_•) − γ(v_j, φ_j, X_•, Y_•, W_•))
                ▷ key = edge improvement over parent edge
14  α = (1/2) log (1 + γ(𝔥_1, W)) / (1 − γ(𝔥_1, W))   ▷ standard coefficient of the full tree classifier 𝔥_1 (3.47)
15  return (α, 𝔥, l, r)
```

Figure 3.10: The pseudocode of the tree base learner. N is the number of leaves. The algorithm returns a list of base classifiers $\mathfrak{h}$, two index lists $\mathfrak{l}$ and $\mathfrak{r}$, and the base coefficient α. The tree classifier is then defined by (3.47).

When the priority queue is implemented as a heap, both the insertion and the extraction of the maximum takes $O(\log N)$ time [91], so the total running time of the procedure is $O\big(N(T_{\text{BASE}} + n + \log N)\big)$, where T_{BASE} is the running time of the base learner. Since N cannot be more than n, the running time is $O\big(N(T_{\text{BASE}} + n)\big)$. If the base learners cutting the leaves are decision stumps, the total running time is $O(nKdN)$. In the procedure we have no explicit control over the shape of the tree, but if it happens to be balanced, the running time can further be improved to $O(nKd \log N)$.

3.4.5 Decision products

In this section we describe another meta-base learner that combines simple base classifiers in a generic way [46]. Similarly to trees, we call the base learner as a subroutine but in an iterative rather than recursive fashion. In the optimization loop we fix all but one of the base classifier terms, temporarily re-label the points, and call the base learner using the "virtual" labels.

Formally, we learn base classifiers of the form

$$\mathbf{h}(\cdot) = \alpha \bigotimes_{j=1}^{m} \mathbf{v}_j \varphi_j(\cdot),$$

where the vote vectors $\mathbf{v}_j$ are multiplied elementwise, and the number of terms m is a complexity parameter that should be validated (similarly to the number of leaves N of decision trees). To maximize the edge (3.33), we follow a simple iterative approach (Figure 3.11). In each iteration we fix each base learner except for one φ_j, and maximize the edge with respect to φ_j. Because of the product form of the edge (3.33), we can carry out this maximization by simply calling the base learner of φ_j with "virtual" labels defined as the product of the real labels and the outputs of the remaining base learners (line 9 in Figure 3.11). Formally, when optimizing the jth term $\mathbf{v}_j \varphi_j(\cdot)$, the edge can be written as

$$\gamma = \sum_{i=1}^{n} \sum_{\ell=1}^{K} w_{i,\ell} \prod_{j'=1}^{m} v_{j',\ell} \varphi_{j'}(\mathbf{x}_i) y_{i,\ell} = \sum_{i=1}^{n} \sum_{\ell=1}^{K} w_{i,\ell} v_j \varphi_j(\mathbf{x}_i) \left(\prod_{\substack{j'=1 \\ j' \neq j}}^{m} v_{j',\ell} \varphi_{j'}(\mathbf{x}_i) y_{i,\ell} \right),$$

so maximizing the edge can be simply done by setting the "virtual" labels to

$$y'_{i,\ell} = y_{i,\ell} \prod_{\substack{j'=1 \\ j' \neq j}}^{m} v_{j',\ell} \varphi_{j'}(\mathbf{x}_i) = y_{i,\ell} \frac{\prod_{j'=1}^{m} v_{j',\ell} \varphi_{j'}(\mathbf{x}_i)}{v_{j,\ell} \varphi_j(\mathbf{x}_i)}$$

and calling the base learner with these labels.

The procedure is guaranteed to converge since in each batch of m iterations at least one "virtual" sign $y'_{i,\ell}$ must change (otherwise the base learner in line 10 returns the same set of m classifiers and we have equality in line 11) and the number of different sign vectors is finite. As with INDICATORBASE in Section 3.4.3, we found that in practice PRODUCTBASE always returns after a few iterations.

REMARK 1: PRODUCTS OF STUMPS. In experiments (see Section 3.4.6) we found that ADABOOST.MH with products of stumps achieves excellent test results on benchmark data sets, for which, at this point, we can only provide a partial explanation. First, it is easy to see that the algorithm solves the XOR problem: a product of two stumps can obviously implement the XOR function. As an extension, the product of d stumps can implement the parity function on any subsequence of the (binary or thresholded real-valued) feature vector up to d elements. It can also be shown that the class of sums of products of d stumps is a universal approximator if and only if the observation space $\mathcal{X}$ is at most d-dimensional.[12] Second, it is clear that the class of sums of products of stumps is a subclass of decision trees: there are constraints on how homogeneous regions are created, and each base learner term of the product has a "global" contribution. Nevertheless, we also found that *one* product of stumps does not have enough capacity to achieve a low test error even if m is set to infinity: PRODUCTBASE underfits the data, probably due to the greediness of the optimization loop. Hence, unlike trees, products cannot be used as standalone strong learners. On the other hand, boosting products is less susceptible to overfitting than boosting trees, for which we also provide some experimental evidence.

[12] Here we shall only give an outline of the constructive proof: $\mathbf{f}$ can be set to an arbitrary value on any hypercube in $\mathbb{R}^d$, independently of the rest of the space by carefully selecting and weighting products of stumps placed at the corners of the hypercube.

PRODUCTBASE($\mathbf{X}, \mathbf{Y}, \mathbf{W}$, BASE$(\cdot,\cdot,\cdot), m$)

1 **for** $j \leftarrow 1$ **to** m
2 $\quad \varphi_j(\cdot) \leftarrow 1, \mathbf{v}_j \leftarrow \mathbf{1}$ ▷ *terms are initialized to the constant* 1 *function*
3 $\alpha \leftarrow 1$
4 **while** TRUE
5 $\quad$ **for** $j \leftarrow 1$ **to** m ▷ *loop over all terms while improvement*
6 $\quad\quad \varphi^*(\cdot) \leftarrow \prod_{j'=1}^{m} \varphi_{j'}(\cdot), \mathbf{v}^* \leftarrow \bigotimes_{j'=1}^{m} \mathbf{v}_{j'}, \alpha^* \leftarrow \alpha$ ▷ *save current optimal classifier*
7 $\quad\quad$ **for** $i \leftarrow 1$ **to** n
8 $\quad\quad\quad$ **for** $\ell \leftarrow 1$ **to** K
9 $\quad\quad\quad\quad y'_{i,\ell} \leftarrow y_{i,\ell} \dfrac{v^*_\ell \varphi^*(\mathbf{x}_i)}{v_{j,\ell}\varphi_j(\mathbf{x}_i)}$ ▷ *"virtual" labels*
10 $\quad\quad (\alpha, \mathbf{v}_j, \varphi_j(\cdot)) \leftarrow$ BASE$(\mathbf{X}, \mathbf{Y}', \mathbf{W})$
11 $\quad\quad$ **if** $Z\left(\alpha \bigotimes_{j'=1}^{m} \mathbf{v}_{j'}\varphi_{j'}, \mathbf{W}\right) \geq Z(\alpha^*\mathbf{v}^*\varphi^*, \mathbf{W})$ **then** ▷ *no improvement*
12 $\quad\quad\quad$ **return** $(\alpha^*, \mathbf{v}^*, \varphi^*(\cdot))$

Figure 3.11: The pseudocode of the product base learner. m is the number of base classifier terms. The vote vectors are multiplied elementwise in lines 6 and 11.

REMARK 2: PRODUCTS OF SUBSET INDICATORS. Boosting products of indicator base classifiers (3.37) is a solution of maximum margin matrix factorization (MMMF) [92], used to formalize and solve the collaborative filtering problem. Taking the simple case when the observations consist of two index features $x^{(1)} \in \mathcal{I}^{(1)}$ and $x^{(2)} \in \mathcal{I}^{(2)}$ (for example, movie IDs and user IDs), and the classification is binary (for example, $y = +1$ if user$_{x^{(1)}}$ liked movie$_{x^{(2)}}$ and $y = -1$ if he or she disliked it), individual base classifiers will take the form

$$\mathbf{h}^{(t)}_{\mathbf{u}^{(t)},\mathbf{z}^{(t)}}\left(x^{(1)}, x^{(2)}\right) = \alpha^{(t)} u^{(t)}_{x^{(1)}} z^{(t)}_{x^{(2)}},$$

where $\mathbf{u}^{(t)}$ and $\mathbf{z}^{(t)}$ are vectors over $\mathcal{I}^{(1)}$ and $\mathcal{I}^{(2)}$, respectively. The strong classifier $\mathbf{f}$ can then be used to classify *every* pair of $\mathcal{I}^{(1)} \times \mathcal{I}^{(2)}$. The resulting $M^{(1)} \times M^{(2)}$ matrix can be expressed as the product of two matrices, an $M^{(1)} \times T$ "user feature" matrix containing the column vectors $\sqrt{\alpha^{(t)}}\mathbf{u}^{(t)}$, and a $T \times M^{(2)}$ "movie feature" matrix containing the row vectors $\sqrt{\alpha^{(t)}}\mathbf{z}^{(t)}$. The maximization of the exponential margin-based error (3.13) leads to a large margin solution, and all the relevant results on the generalization error (e.g., [61]) apply immediately. Finding two low-rank (or otherwise low-complexity) matrices whose product produces large margins on the training data is the exact goal of MMMF [92]. In a certain sense, boosting products of two index learners is related to the semi-definite-programming-based solution of MMMF as the classical binary ADABOOST is related to support vector machines.

The algorithm can be applied directly to multi-valued preferences by adding the optimization of the vote vectors $\mathbf{v}^{(t)}$. In this case, the preference prediction matrix becomes a hyper-matrix with vector-valued elements. The formulation also permits one to boost products of more than two indices (or to use two or more times a base learner on the same index) which goes beyond the matrix-decomposition-based formulation of the collaborative filtering problem. It is also quite straightforward to combine collaborative features with "traditional" numerical or nominal covariates by allowing mixed products.

3.4.6 Comparative experiments of base learners

To test the base learners, we carried out two sets of experiments. In the first we boosted products of decision stumps on five benchmark datasets[13] using the standard train/test cuts. We used discrete ADABOOST.MH with multi-class initial weights (3.12). We found no overfitting whatsoever in terms of the number of boosting iterations T (Figure 3.12), so instead of validating T and performing an early stopping, we decided to run the algorithm for a long time after convergence ($T = 10^5$ in each experiment), and measure the average test error $\widehat{R}\left(\mathbf{f}^{(T)}\right)$ on the last $T/2$ iterations. The advantage of this approach is that this estimate is more robust in terms of random fluctuations after convergence than the error at a given iteration. It is also a pessimistic estimate of the error when there is a slight overfitting (since the average is always an upper bound of the minimum). For the two image datasets we also report results using ADABOOST.MH with stumps on Haar filters [62].[14]

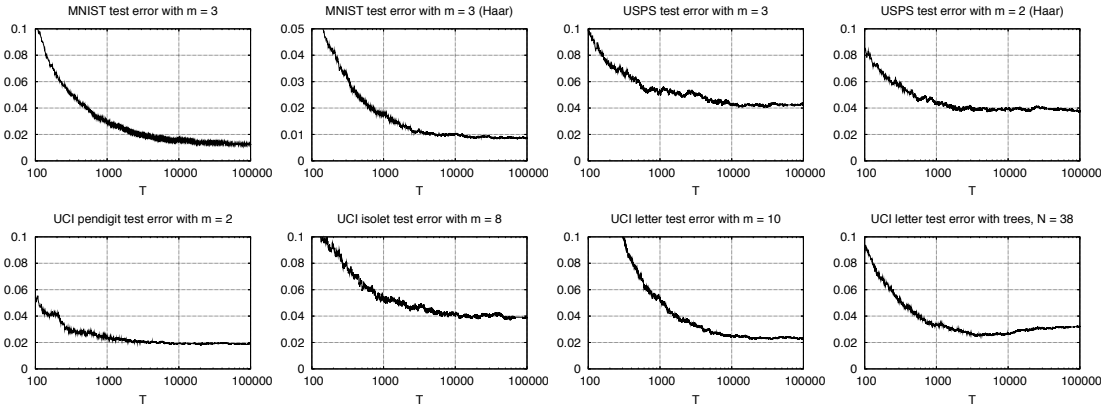


Figure 3.12: The "winning" learning curves on the five benchmark datasets indicate no overfitting in terms of the number of boosting iterations T when products are used. With a tree base learner (last plot), on the other hand, we do observe slight overfitting sometimes.

We conducted experiments using a range of values for the number of terms m for which we report the test errors in Table 3.1. The optimal number of terms (error rate in bold face) was selected by a 80%-20% simple validation on the training set. We compared PRODUCTBASE with TREEBASE. Since this base learner was more prone to overfitting (last plot in Figure 3.12), we validated both the number of leaves N and the number of boosting iterations T. Table 3.1 indicates that, with one exception, boosting products is significantly better than boosting trees. We also tested the Weka [93] implementation of boosted trees although the comparison is slightly unfair for two reasons: first, the Weka implementation uses ADABOOST.M1 which is known to be inferior to ADABOOST.MH for multi-class boosting, and second, the high computational complexity did not allow us a complete exploration of the hyperparameter space, so we cannot claim that we have found the optimal tree size and number of iterations T. Nevertheless, these results provide a reasonable idea about the novice user's experience with AdaBoost.

On an absolute scale the results are on a par with the state-of-the-art results reported in the literature on the same data sets. In particular, they are the best results obtained by any ensemble method. We would especially like to underline the 1.26% test error on the MNIST dataset, which is the best reported error rate among generic classification algorithms[15] after Hinton, G. E. and Salakhutdinov's [94] deep belief nets (1.00%); it is significantly better than the error rates

[13] The data sets are available at `yann.lecun.com/exdb/mnist` (MNIST), `www.kernel-machines.org/data.html` (USPS), `www.ics.uci.edu/~mlearn/MLRepository.html` (letter, pendigit, isolet), and `www.cs.umn.edu/Research/GroupLens` (MovieLens).

[14] We used five filter types ("bw" and "bwb", vertical and horizontal, and four-field checkerboard). Due to the large number of possible filters (order of 10^5), we selected just the best of a random 100 in each boosting iteration.

[15] Algorithms that do not make explicit use of the fact that the observation vectors represent images.

learner \ data set	MNIST	USPS	UCI pendigit	UCI isolet	UCI letter
Stump ($m = 1$)	7.71 (0.05)	6.48 (0.05)	4.97 (0.00)	4.88 (0.07)	14.74 (0.06)
Product / $m = 2$	1.56 (0.03)	4.92 (0.06)	**1.89** (**0.02**)	3.91 (0.10)	3.65 (0.04)
Product / $m = 3$	**1.26** (**0.02**)	**4.24** (**0.04**)	2.07 (0.03)	3.97 (0.04)	2.71 (0.02)
Product / $m = 4$	1.38 (0.02)	4.72 (0.07)		3.95 (0.05)	2.54 (0.06)
Product / $m = 5$				3.87 (0.07)	2.41 (0.04)
Product / $m = 6$				**3.92** (**0.08**)	2.40 (0.03)
Product / $m = 8$				3.89 (0.04)	2.38 (0.04)
Product / $m = 10$				4.11 (0.04)	**2.35** (**0.04**)
Haar ($m = 1$) [62]	1.02 (0.02)	4.29 (0.05)			
Haar / Product / $m = 2$	0.84 (0.02)	**3.84** (**0.07**)			
Haar / Product / $m = 3$	**0.87** (**0.02**)	4.03 (0.06)			
Haar / Product / $m = 4$	0.90 (0.02)	4.05 (0.04)			
Tree	1.53 (0.02)	4.73 (0.04)	2.14 (0.04)	3.69 (0.04)	2.62 (0.04)
Haar / Tree	1.08 (0.02)	4.98 (0.05)			
AdaBoost.M1 / C4.5	4.05	5.98	2.66	4.81	2.88

Table 3.1: Test error percentages $100\frac{2}{T}\sum_{t=T/2}^{T}\widehat{R}\left(\mathbf{f}^{(t)}\right)$ on benchmark datasets using discrete ADABOOST.MH with products of stumps. The results with the optimal number of terms m selected by 80%-20% simple validation on the training set are shown in bold. Tree and Haar/Tree use TREEBASE. In this case both T and the number of leaves N were validated. For AdaBoost.M1/C4.5 we used the Weka implementation.

of support vector machines (1.4%) and randomly initialized two-layer back-propagation neural nets using cross-entropy loss (1.6%).

set	WLRA [95]		MMMF SDP [92]		ADABOOST	
1	rank 2	57.5	max-norm, $C = 0.0012$	**56.2**	$T = 7275$	56.3
2	rank 2	56.2	trace norm, $C = 0.24$	55.2	$T = 9940$	**54.5**
3	rank 1	54.3	max-norm, $C = 0.0012$	**52.7**	$T = 475$	53.9
4	rank 2	55.3	max-norm, $C = 0.0012$	**55.0**	$T = 9515$	56.6
Avg		55.8		**54.8**		55.3

Table 3.2: Test error percentages on the MovieLens dataset using the experimental setup of [92]. The errors in columns 3 and 5 were taken directly from [92].

It seems also quite surprising that we were able to improve on boosting stumps over the feature space generated by Haar filters. Boosting stumps over Haar filters is already one of the best semi-generic method[16], achieving similar error rates to convolutional neural nets [96]. Boosting these feature extractors also outperforms boosting stumps and even boosting products of stumps (on MNIST). The feature space has a large dimension (of order 10^5) and the Haar filters are well-adapted to natural images, so one would expect that a simple linear combination over this space can achieve the best results. While this intuition is reaffirmed when using *trees* over the Haar space, we were genuinely surprised to see that using the *product* of a small number of Haar filters as a base learner can significantly outperform boosting single Haar filters.

In the second set of experiments we tested the algorithm on a small subset of the MovieLens collaborative filtering database using the same experimental settings as Srebro et al. [92]: the data is divided into four sets, for each of the four test sets the algorithms are trained and validated on the remaining three in a 3-fold cross validation. The two best of the three learners are selected and tested on the hold-out test set. In the case of ADABOOST, we validated the real/discrete

[16] Algorithms that do make explicit use of the fact that the observation vectors represent images, but not of the fact that the images depict characters.

version, the weight initialization, the number of terms m and the number of iterations T. In each experiment, using real ADABOOST.MH with uniform weight initialization $w_{i,\ell}^{(1)} = 1/(nK)$ and $m = 2$ proved to be the best approach. The results in Table 3.2 place this approach between WLRA [95] and semi-definite-programming-based MMMF [92]. The computational effort needed to produce the results was an order of magnitude smaller than in the case of SDP MMMF (minutes vs. hours). Since ADABOOST.MH scales linearly with the data size, this approach has a greater prospective on large collaborative filtering problems.

3.5 Calibration and aggregation for ranking

In this section we summarize our works on learning to rank [74, 75]. The method we developed came in 5th in one of the tracks in the ranking challenge organized by Yahoo! in 2010 [64]. There are three interesting methodological findings in these works. First, we show that a simple pointwise approach is competitive to the more sophisticated but computationally less efficient pairwise and listwise techniques. Second, we describe how to calibrate a seemingly inadequate multi-class classifier to obtain good rankers. Finally, we demonstrate that combining all the trained models in a simple ensemble is significantly better than the traditional approach of selecting the one best model based on validation. This latter fact is corroborated by the winning approaches in most of the recent last scale data-mining challenges [69, 63, 64].

3.5.1 Introduction

In the past, the result lists in Information Retrieval were ranked by probabilistic models (such as the BM25 measure [97]) based on a small number of attributes (the frequency of query terms in the document, in the collection, etc.). The parameters of these models were usually set empirically. As the number of useful features increase, these manually crafted models become increasingly laborious to configure. Alternatively, one can use as many (possibly redundant) attributes as possible, and employ Machine Learning techniques to induce a ranking model. This approach alleviates the human effort needed to design the ranking function, and also provides a natural way to directly optimize the retrieval performance for any particular application and evaluation metric. As a result, Learning to Rank has gained considerable research interest in the past decade.

Machine Learning based ranking systems are traditionally classified into three categories. In the simplest *pointwise* approach, the instances are first assigned a relevance score using classical regression or classification techniques and then ranked by posterior scores obtained using the trained model [98]. In the *pairwise* approach, the order of pairs of instances is treated as a binary label and learned by a classification method [99]. Finally, in the most complex *listwise* approach, the fully ranked lists are learned by a tailor-made learning method which aims to optimize a ranking-specific evaluation metric during the learning process [100].

In web page ranking or subset ranking [101] the training data is given in the form of query-document-relevance label triplets. The relevance label of a training instance indicates the usefulness of the document to its corresponding query, and the ranking for a particular query is usually evaluated using the (normalized) Discounted Cumulative Gain ((N)DCG) or the Expected Reciprocal Rank (ERR) [102] measure. It is rather difficult to extend classical learning methods to directly optimize these evaluation metrics. Nevertheless, since the DCG can be bounded by the one-loss [98], traditional classification error can be considered as a surrogate function for DCG.

Calibrating the output of a particular learning method is crucial in applications with quality measures different from the one-error [103]. Our approach is based on the calibration of a multi-class classification model learned by ADABOOST [44]. In our setup, the class labels are assumed to be random variables and the goal is the estimation of the probability distribution of the class labels given a feature vector representing the (document, query) pair. An important novelty in our approach is that, instead of using a single calibration technique, we apply several methods to estimate the same probability distribution and, in a final step, we combine these estimates.

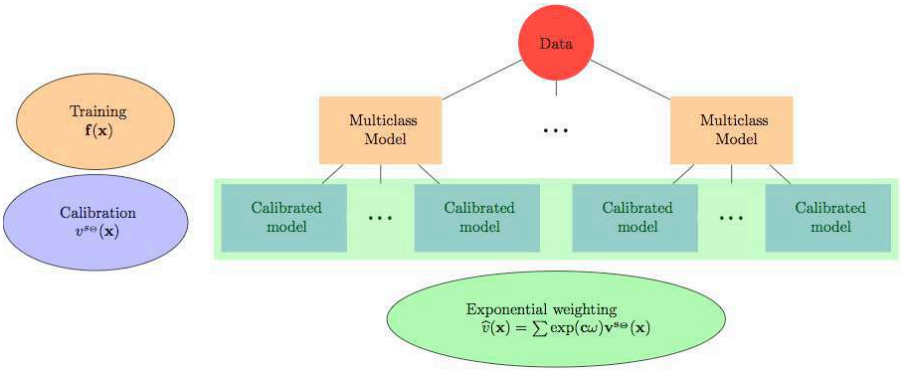


Figure 3.13: The schematic overview of our approach. In the first level a multi-class method (ADABOOST.MH) is trained using different hyperparameter settings. Then we calibrate the multi-class models in several ways to obtain diverse scoring functions. In the last step we aggregate the scoring functions using an exponential weighting.

We use both *regression-based calibration (RBC)* and *class probability-based calibration (CPC)* to transform the output scores of AdaBoost into relevance label estimates. In the case of RBC, the real-valued scores are obtained by regressing the relevance labels against the output of ADABOOST. These scores are then used to rank the objects. In the case of CPC, the posterior probability distribution is used to approximate the so-called Bayes-scoring function [101], which is shown to optimize the expected DCG in a probabilistic setup.

The proper choice of the prior on the set of conditional distributions obtained by the calibration of ADABOOST is an important decision in practice. In this paper, we use an exponential scheme based on the quality of the rankings which is theoretically more well-founded than the uniformly weighted aggregation used by MCRANK [98].

Figure 3.13 provides a structural overview of our system. Our approach belongs to the simplest pointwise category of learning-to-rank models. It is based on a series of standard techniques: 1) multi-class classification, 2) output score calibration and 3) exponentially weighted forecaster to combine the various hypotheses. As opposed to previous works, we found our approach to be competitive to the standard methods (ADARANK, LISTNET, RANKSVM, and RANKBOOST) of the theoretically more involved pairwise and listwise approaches.

The section is organized as follows. Section 3.5.2 describes the formal setup. Section 3.5.3 is devoted to the description of calibration technique. The ensemble scheme is presented in Section 3.5.6. Our experimental results are presented in Section 3.5.7.

3.5.2 Definition of the ranking problem

Let us assume that we are given a set of *query objects* $\mathbf{D} = \{\mathcal{D}^{(1)}, \ldots, \mathcal{D}^{(M)}\}$. Each query object $\mathcal{D}^{(k)}$ consists of a set of $n^{(k)}$ pairs

$$\mathcal{D}^{(k)} = \left\{ \left(\mathbf{x}_1^{(k)}, \ell_1^{(k)}\right), \ldots, \left(\mathbf{x}_1^{(k)}, \ell_{n^{(k)}}^{(k)}\right) \right\}.$$

The real-valued feature vectors $\mathbf{x}_i^{(k)}$ represent the kth query and the ith document received as a potential hit for the query. The *label index* $\ell_i^{(k)}$ of the query-document pair $\mathbf{x}_i^{(k)}$ is an integer between 1 and K. We assume that we are given a set of numerical *relevance grades*

$$\mathcal{Z} = \{z_1, \ldots, z_K\}.$$

The relevance grade $z_i^{(k)} = z_{\ell_i^{(k)}}$ expresses the relevance of the ith document to the kth query on a numerical scale. A popular choice for the numerical relevance grades is

$$z_\ell = 2^{\ell-1} - 1$$

for all $\ell = 1, \dots, K$.

The goal of the ranker is to output a permutation $\boldsymbol{j}^{(k)} = (j_1, \dots, j_{n^{(k)}})$ over the integers $(1, \dots, n^{(k)})$ for each query object $\mathcal{D}^{(k)}$. A widely used empirical measure of the quality of the permutation $\boldsymbol{j}^{(k)}$ is the Discounted Cumulative Gain (DCG)

$$\widehat{\mathrm{DCG}}(\boldsymbol{j}^{(k)}, \mathcal{D}^{(k)}) = \sum_{i=1}^{n^{(k)}} c_i z_{j_i}^{(k)}, \tag{3.48}$$

where c_i is the *discount factor* of the i^{th} document in the permutation. The most common discount factor is

$$c_i = \frac{1}{\log(1+i)}.$$

The rationale of this formula that a user will be increasingly happy when finding relevant documents early in the permutation. To normalize DCG between 0 and 1, (3.48) is usually divided with the DCG score of the best permutation (NDCG). It is also a common practice to truncate the sum (3.48) at $n_{\max}$, defining the $\mathrm{DCG}_{n_{\max}}$ and $\mathrm{NDCG}_{n_{\max}}$ scores. The reason is that a user would rarely leave the first page of search results containing the first $n_{\max}$ hits.

We will consider $\ell_i^{(k)}$ as a random variable with a conditional discrete probability distribution

$$p^*(\ell|\mathbf{x}_i^{(k)}) = P(\ell_i^{(k)} = \ell|\mathbf{x}_i^{(k)})$$

over the label indices ℓ for document i of query k. The *Bayes-scoring function*

$$v^*(\mathbf{x}_i^{(k)}) = \mathbb{E}\left\{z|\mathbf{x}_i^{(k)}\right\} = \sum_{\ell=1}^{K} z_\ell p^*(\ell|\mathbf{x}_i^{(k)})$$

is the conditional expectation of the relevance grade given the (query, document) pair. Then the *expected* DCG for any permutation $\boldsymbol{j}^{(k)}$ is

$$\mathrm{DCG}(\boldsymbol{j}^{(k)}, \mathcal{D}^{(k)}) = \sum_{i=1}^{n^{(k)}} c_i \mathbb{E}\left\{z|\mathbf{x}_{j_i}^{(k)}\right\} = \sum_{i=1}^{n^{(k)}} c_i v^*(\mathbf{x}_{j_i}^{(k)}).$$

Let the *Bayes optimal* permutation $\boldsymbol{j}^{(k)^*} = (j_1^{(k)^*}, \dots, j_{n^{(k)}}^{(k)^*})$ over $\mathcal{D}^{(k)}$ be the one which maximizes the expected DCG, that is,

$$\boldsymbol{j}^{(k)^*} = \arg\max_{\boldsymbol{j}^{(k)}} \mathrm{DCG}(\boldsymbol{j}^{(k)}, \mathcal{D}^{(k)}).$$

According to Theorem 1 of [101], $\boldsymbol{j}^{(k)^*}$ has the property that if $c_i > c_{i'}$, then for the Bayes-scoring function we have $v^*(\mathbf{x}_{j_i^{(k)^*}}) > v^*(\mathbf{x}_{j_{i'}^{(k)^*}})$. This means that the optimal $\boldsymbol{j}^{(k)^*}$ can be easily obtained from the Bayes-optimal scoring function v^* by ordering the query-document pairs $\mathbf{x}_i^{(k)}$ according to $v^*(\mathbf{x}_i^{(k)})$. This result justifies the *pointwise* approach that estimates v^* in a *regression* setup. In this paper we also use the *pointwise* approach in a *discrete density estimation* setup: our goal is to estimate $p^*(\ell|\mathbf{x}_j^{(k)})$ by $p^{\mathcal{A}}(\ell|\mathbf{x}_j^{(k)})$, where the label $\mathcal{A}$ will refer to the method that generates the probability estimates. To refer the scoring function generated by $p^{\mathcal{A}}$, we will use the notation

$$v^{\mathcal{A}}(\mathbf{x}_i^{(k)}) = \sum_{\ell=1}^{K} z_\ell p^{\mathcal{A}}(\ell|\mathbf{x}_i^{(k)}). \tag{3.49}$$

In the pointwise approach, the scoring function v induces the permutation j^v for which

$$v(\mathbf{x}_{j_1^v}^{(k)}) \geq \ldots \geq v(\mathbf{x}_{j_{n^{(k)}}^v}^{(k)}). \tag{3.50}$$

3.5.3 The calibration of multi-class classification models

Our basic modeling tool is multi-class AdaBoost.MH [45] (Section 3.3.2). To tackle the ranking problem, we first convert the set of query objects $\mathbf{D}$ into a multi-class setup. The input training set of AdaBoost.MH consists of the set of feature vectors $\mathbf{X} = \left(\mathbf{x}_1^1, \ldots, \mathbf{x}_{n^{(1)}}^1, \ldots, \mathbf{x}_1^M, \ldots, \mathbf{x}_{n^{(M)}}^M\right)$ and a set of labels $\mathbf{Y} = \left(\mathbf{y}_1^1, \ldots, \mathbf{y}_{n^{(1)}}^1, \ldots, \mathbf{y}_1^M, \ldots, \mathbf{y}_{n^{(M)}}^M\right)$, where each label vector $\mathbf{y}_i^{(k)}$ encodes the label index $\ell_i^{(k)}$ in a one-out-of-K scheme

$$y_{i,\ell}^{(k)} = \begin{cases} +1 & \text{if } \ell_i^{(k)} = \ell, \\ -1 & \text{otherwise.} \end{cases}$$

We used two base learners: *decision trees* (Section 3.4.4) and *decision products* [46] (Section 3.4.5). Instead of using uniform (3.11) or multi-class (3.12) weighting for training instances, we up-weighted relevant instances proportionally to their relevance grades. Formally, the initial (unnormalized) weight of ℓth label of the (k,i)th instance is

$$w_{i,\ell}^{(k)} = \begin{cases} 1 + z_i^{(k)} & \text{if } \ell_i^{(k)} = \ell, \\ (1 + z_i^{(k)})/(K-1) & \text{otherwise.} \end{cases}$$

The weights are then normalized to sum to 1. This weighting scheme was motivated by the evaluation metric: the weight of an instance in the DCG score (3.48) is proportional to the relevance grade.

AdaBoost.MH outputs a vector-valued strong classifier

$$\mathbf{f}^{(T)}(\mathbf{x}) = \sum_{t=1}^{T} \alpha^{(t)} \mathbf{h}^{(t)}(\mathbf{x}),$$

where $\mathbf{h}^{(t)}(\mathbf{x})$ is a $\{\pm 1\}^K$-valued base classifier and $\alpha^{(t)}$ is its weight. In multi-class classification the elements of $\mathbf{f}^{(T)}(\mathbf{x})$ are treated as posterior scores corresponding to the labels, and the predicted single label is

$$\widehat{\ell}_i^{(k)} = \arg\max_{\ell=1,\ldots,K} f_\ell^{(T)}(\mathbf{x}_i^{(k)}).$$

When posterior probability estimates are required, in the simplest case the output vector can be shifted into $[0,1]^K$ using

$$\mathbf{f}'^{(T)}(\mathbf{x}) = \frac{1}{2}\left(1 + \frac{\mathbf{f}^{(T)}(\mathbf{x})}{\sum_{t=1}^{T} \alpha^{(t)}}\right),$$

and then the posterior probabilities can be obtained by simple normalization

$$p^{\text{SHIFT}}(\ell|\mathbf{x}_i^{(k)}) = \frac{f_\ell'^{(T)}(\mathbf{x}_i^{(k)})}{\sum_{\ell'=1}^{K} f_{\ell'}'^{(T)}(\mathbf{x}_i^{(k)})}. \tag{3.51}$$

3.5.4 Regression based pointwise calibration

An obvious way to calibrate the output of ADABOOST.MH is to re-learn the relevance grades by applying a regression method. Using the raw K-dimensional output of ADABOOST.MH we can obtain relevance grade estimates in the form of

$$\widehat{z}_i^{(k)} = g\left(\mathbf{f}^{(T)}(\mathbf{x}_i^{(k)})\right),$$

where $g : \mathbb{R}^{(k)} \to \mathbb{R}$ is the regression function. We shall refer to this scheme as *regression-based calibration* (RBC).

In our experiments we used five different regression methods: Gaussian process regression, logistic regression, linear regression, neural network regression, and polynomial regression of degree between 2 and 5. From a practical point of view, the relevance grade estimates provided by the different regression methods for an unseen test document are on a different scale, so these values cannot be aggregated in a direct way. In Section 3.5.6 we explain how to normalize these values.

3.5.5 Class probability calibration

Let us recall that the output of ADABOOST.MH is

$$\mathbf{f}(\mathbf{x}_i^{(k)}) = \left(f_1(\mathbf{x}_i^{(k)}), \ldots, f_K(\mathbf{x}_i^{(k)})\right).$$

Then the class probability can be calibrated a using *sigmoidal* function

$$s_\theta(f) = s_{a,b}(f) = \frac{1}{1 + \exp\big(-a(f-b)\big)}$$

to obtain

$$p^{s_\theta}(\ell|\mathbf{x}_i^{(k)}) = \frac{s_\theta\left(f_\ell(\mathbf{x}_i^{(k)})\right)}{\sum_{\ell'=1}^{K} s_\theta\left(f_{\ell'}(\mathbf{x}_i^{(k)})\right)}. \tag{3.52}$$

The parameters of the sigmoid function can be tuned by minimizing a so-called *target calibration function (TCP)* $L^{\mathcal{A}}(\theta, \mathbf{f})$. Generally speaking, $L^{\mathcal{A}}(\theta, \mathbf{f})$ is a loss function calculated using the probability estimates (3.52). $L^{\mathcal{A}}$ is parametrized by the parameter $\theta = (a, b)$ of the sigmoid function and the output of the multi-class classifier $\mathbf{f}$.

Given a TCF $L^{\mathcal{A}}$ and a multi-class classifier $\mathbf{f}$, our goal is to find the optimal calibration parameters

$$\theta^{\mathcal{A},\mathbf{f}} = \arg\min_{\theta} L^{\mathcal{A}}(\theta, \mathbf{f}).$$

The output of this calibration step is a probability distribution $p^{\mathcal{A},\mathbf{f}}(\ell|\mathbf{x}_i^{(k)})$ and a corresponding Bayes-scoring function $v^{\mathcal{A},\mathbf{f}}(\mathbf{x}_i^{(k)})$ defined in (3.49). We will refer to this scheme as *class-probability-based calibration* (CPC). The upper index $\mathcal{A}$ refers to the type of the particular TCF, so the *ensemble* of probability distributions is indexed by the target calibration function $\mathcal{A}$ and the multi-class classifier $\mathbf{f}$ output by ADABOOST.MH.

Given the ensemble of probability distributions $p^{\mathcal{A},\mathbf{f}}(\ell|\mathbf{x}_i^{(k)})$ and an appropriately chosen weights $\pi(\mathcal{A}, \mathbf{f})$, we compute the combined conditional distribution by

$$p^{\mathrm{CPC}}(\ell|\mathbf{x}_i^{(k)}) = \sum_{\mathcal{A},\mathbf{f}} \pi(\mathcal{A}, \mathbf{f}) p^{\mathcal{A},\mathbf{f}}(\ell|\mathbf{x}_i^{(k)}).$$

Then we obtain a "posterior" estimate for the Bayes-scoring function

$$\begin{aligned} v^{\mathrm{CPC}}\left(\mathbf{x}_i^{(k)}\right) &= \sum_{\ell=1}^{K} z_\ell p^{\mathrm{CPC}}\left(\ell|\mathbf{x}_i^{(k)}\right) \\ &= \sum_{\ell=1}^{K} z_\ell \sum_{\mathcal{A},\mathbf{f}} \pi(\mathcal{A},\mathbf{f}) p^{\mathcal{A},\mathbf{f}}\left(\ell|\mathbf{x}_i^{(k)}\right) \\ &= \sum_{\mathcal{A},\mathbf{f}} \pi(\mathcal{A},\mathbf{f}) \sum_{\ell=1}^{K} z_\ell p^{\mathcal{A},\mathbf{f}}\left(\ell|\mathbf{x}_i^{(k)}\right) \\ &= \sum_{\mathcal{A},\mathbf{f}} \pi(\mathcal{A},\mathbf{f}) v^{\mathcal{A},\mathbf{f}}\left(\mathbf{x}_i^{(k)}\right). \end{aligned} \tag{3.53}$$

The proper selection of $\pi(\cdot,\cdot)$ can further increase the quality of the posterior estimation. In Section 3.5.6 we will describe a reasonable setup borrowed from the theory of experts.

In the simplest case, the TCF can be

$$L^{\mathrm{LS}}(\theta,\mathbf{f}) = \sum_{k=1}^{M}\sum_{i=1}^{n^{(k)}} -\log p^{s_\theta}\left(\ell_i^{(k)}|\mathbf{x}_i^{(k)}\right).$$

We refer to this function as the *log-sigmoid TCF*. The motivation of the log-sigmoid TCF is that the resulting probability distribution minimizes the relative entropy between the Bayes optimal probability distribution p^* and $p^{\mathcal{A},\mathbf{f}}$. In practice distributions being less (or more) uniform over the labels might be preferred. This preference can be expressed by introducing the entropy weighted version of the log-sigmoid TCF

$$L_C^{\mathrm{EWLS}}(\theta) = \sum_{k=1}^{M}\sum_{i=1}^{n^{(k)}} -\log p^{s_\theta}\left(\ell_i^{(k)}|\mathbf{x}_i^{(k)}\right) \times H_M\left(p^{s_\theta}\left(\ell_1|\mathbf{x}_i^{(k)}\right),\ldots,p^{s_\theta}\left(\ell_K|\mathbf{x}_i^{(k)}\right)\right)^C,$$

where

$$H_M\left(p_1,\ldots,p_K\right) = -\sum_{\ell=1}^{K} p_\ell \log p_\ell,$$

and C is a hyperparameter.

We also used TCFs based on the *expected loss*

$$L^{\mathrm{EL}}(\theta) = \sum_{k=1}^{M}\sum_{i=1}^{n^{(k)}}\sum_{\ell=1}^{K} \mathcal{L}\left(\ell,\ell_i^{(k)}\right) p^{s_\theta}\left(\ell|\mathbf{x}_i^{(k)}\right),$$

where $\mathcal{L}(\ell,\ell_i^{(k)})$ represents the loss if ℓ is predicted instead of the correct label $\ell_i^{(k)}$. We used the standard squared loss $\mathcal{L}(\ell,\ell') = (\ell-\ell')^2$.

If the label indices have some structure, e.g., they are ordinal as in our case, it is also possible to first compute the expected label

$$\overline{\ell}_i^{(k)} = \sum_{\ell=1}^{K} \ell p^{s_\theta}\left(\ell|\mathbf{x}_i^{(k)}\right)$$

and then compute the *expected label loss TCF*

$$L^{\mathrm{ELL}}(\theta) = \sum_{k=1}^{M}\sum_{i=1}^{n^{(k)}} \mathcal{L}\left(\overline{\ell}_i^{(k)},\ell_i^{(k)}\right).$$

Note that the definition of $\mathcal{L}(\cdot,\cdot)$ might need to be redefined for L^{ELL} since the weighted average of the label indices might not be a label index at all.

Finally, we can apply the idea of SMOOTHGRAD [105] to obtain a TCF. In SMOOTHGRAD a smooth surrogate function is used to optimize the NDCG metric. In particular, using the normalized *soft indicator* (or similarity function)

$$h_{\theta,\sigma}(\mathbf{x}_i^{(k)}, \mathbf{x}_{i'}^{(k)}) = \frac{\exp\left(-\frac{1}{\sigma}\left(v^{s_\theta}(\mathbf{x}_i^{(k)}) - v^{s_\theta}(\mathbf{x}_{i'}^{(k)})\right)^2\right)}{\sum_{i''=1}^{n^{(k)}} \exp\left(-\frac{1}{\sigma}\left(v^{s_\theta}(\mathbf{x}_i^{(k)}) - v^{s_\theta}(\mathbf{x}_{i''}^{(k)})\right)^2\right)}$$

the soft NDCG TCF can be written as

$$L_\sigma^{\text{SNDCG}}(\theta) = -\sum_{k=1}^{M}\sum_{i=1}^{n^{(k)}}\sum_{i'=1}^{n^{(k)}} z_i^{(k)} c_{i'} h_{\theta,\sigma}(\mathbf{x}_i^{(k)}, \mathbf{x}_{j_{i'}}^{(k)})$$

where $\boldsymbol{j} = (j_1, \ldots, j_{n^{(k)}})$ is the permutation defined by the scoring function $v^{s_\theta}(\cdot)$ (3.50). The parameter σ controls the smoothness of L_σ^{SNDCG}: the higher σ, the smoother is the function but the bigger is the difference between the NDCG value and the value of surrogate function. If $\sigma \to 0$ then L_σ^{SNDCG} tends to the NDCG value, but, at the same time, optimizing the surrogate function becomes harder.

3.5.6 Ensemble of ensembles

Selecting the best hyperparameters for ADABOOST.MH and the best calibration function is normally done in a validation step. This procedure would "throw away" most of the diverse information represented by the different predictors. Instead of selecting the best, we use all the relevance predictions $v^{\mathcal{A},\mathbf{f}}(\mathbf{x}, S)$ obtained by different ADABOOST classifiers $\mathbf{f}$ using different TCFs $\mathcal{A}$. Each relevance prediction can be used as a scoring function to rank the query-document pairs $\mathbf{x}_i^{(k)}$ according to (3.50). Until this point our method is a purely pointwise approach (except for the calibration using L^{SNDCG}). To fine-tune the algorithm and to make use of the diversity of our models, we combine the scoring functions $v^{\mathcal{A},\mathbf{f}}(\mathbf{x}, S)$ using an exponentially weighted forecaster [106]. The reason of using this particular weighting scheme is twofold. First, it is simple and computationally efficient to tune which is important when we have a large number of models. Second, theoretical guarantees over the cumulative regret of a mixture of experts on individual (modelless) sequences [106] makes the technique robust against overfitting the validation set.

The weights of the models are tuned on the NDCG score, giving a slight listwise touch to our approach. Formally, the final scoring function is obtained by using $\pi(\mathcal{A}, \mathbf{f}) = \exp(c\omega^{\mathcal{A},\mathbf{f}})$, where $\omega^{\mathcal{A},\mathbf{f}}$ is the NDCG_{10} score of the ranking obtained by using $v^{\mathcal{A},\mathbf{f}}(\mathbf{x})$. Plugging it into (3.53), the combined scoring function is

$$v^{\text{ENSEMBLE}}(\mathbf{x}) = \sum_{\mathcal{A},\mathbf{f}} \exp\left(c\omega^{\mathcal{A},\mathbf{f}}\right) v^{\mathcal{A},\mathbf{f}}(\mathbf{x}). \tag{3.54}$$

The parameter c controls the dependence of the weights on the NDCG_{10} values.

A similar ensemble method can be applied to outputs calibrated by regression methods. A major difference between the two types of calibration is that the regression-based scores have to be normalized/rescaled before the exponentially weighted ensemble scheme is applied. We simply rescaled the output of the regression models into $[0, 1]$ before using them in the exponential ensemble scheme.

3.5.7 Experiments

In our experiments we used the Ohsumed dataset taken from LETOR 3.0 and both datasets of LETOR 4.0[17]. We are only interested in datasets that contain more than 2 levels of relevance. On

[17] http://research.microsoft.com/en-us/um/beijing/projects/letor/

the one hand, this has a technical reason: calibration for binary relevance labels does not make too much sense. On the other hand, we believe that in this case the difference between various learning algorithms is more significant. All LETOR datasets we used contain 3 levels of relevance. We summarize their main statistics in Table 3.3.

Table 3.3: The statistics of the datasets we used in our experiments.

	Number of documents	Number of queries	Number of features	Documents per query
LETOR 3.0/Ohsumed	16140	106	45	152
LETOR 4.0/MQ2007	69623	1692	46	41
LETOR 4.0/MQ2008	15211	784	46	19

For each LETOR dataset there is a 5-fold train/valid/test split given. We used this split except that we divided the official train set by a random $80\% - 20\%$ split into training and calibration sets which were used to adjust the parameters of the different calibration methods. We did not apply any feature engineering or preprocessing to the official feature set. The NDCG values we report in this section were computed using the provided evaluation tools.

We compared our algorithm to five state-of-the-art ranking methods whose outputs are available at the LETOR website for each dataset we used:

1. ADARANK-MAP [107]: a listwise boosting approach aiming to optimize MAP.
2. ADARANK-NDCG [107]: a listwise boosting approach with the NDCG as objective function, minimized by the ADABOOST mechanism.
3. LISTNET [108]: a probabilistic listwise method which employs cross entropy loss as the listwise loss function in gradient descent.
4. RANKBOOST [99]: a pairwise approach which casts the ranking problem into a binary classification task. In each boosting iteration the weak classifier is chosen based on NDCG instead of error rate.
5. RANKSVM [109] a pairwise method based on SVM, also based on binary classification as RANKBOOST.

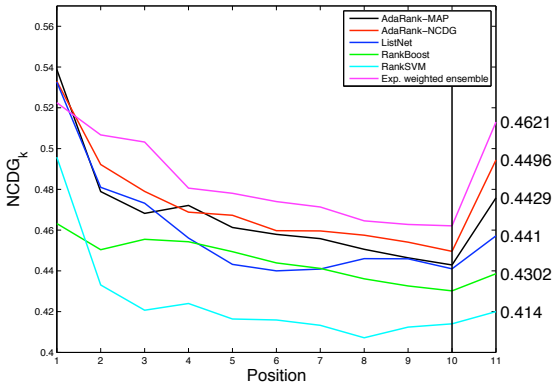
To train ADABOOST.MH, we used our open source implementation available at http://multiboost.org. We did not validate the hyperparameters of weak learners. Instead, we calibrated and used all the trained models with various number of tree leafs and product terms. The number of tree leaves ranged from 5 to 45 uniformly with a step size of 5, and the number of product terms from 2 to 10. The training was performed on a grid[18] which allowed us to fully parallelize the training process, and thus it took less than one day to obtain all strong classifiers.

We only tuned the number of iterations T and the base parameter c in the exponential weighting scheme (3.54) on the validation set. In the exponential weighting combination (3.54) we set the weights using the NDCG_{10} performances of the calibrated models and c and T were selected based on the performance of the combined scoring function $v^{\text{ENSEMBLE}}(\cdot)$ in terms of NDCG_{10}. The hyperparameter optimization was performed using a simple grid search where c ranged from 0 (corresponding to uniform weighting) to 200 and for T from 10 to 10000. Interestingly, the best number of iterations is very low compared to the ones reported by [46] for classification tasks. For LETOR 3.0 the best number of iterations is $T = 100$ and for both LETOR 4.0 datasets $T = 50$. The best base parameter is $c = 100$ for all databases. This value is relatively high considering that it is used in the exponent, but the performances of the best models were relatively close to each other. We used fixed parameters $C = 2$ in the TCN function L_C^{EWLS}, and $\sigma = 0.01$ in L_σ^{SN}.

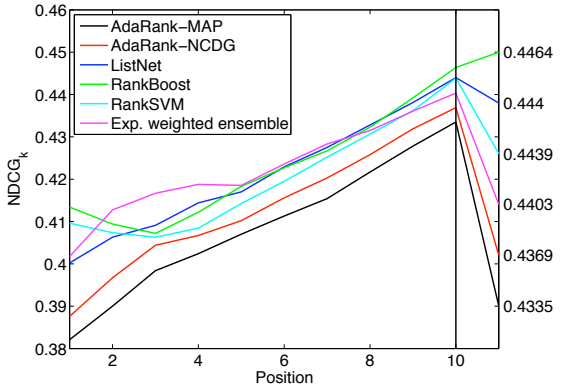
[18] http://www.egi.eu

Comparison to standard learning to rank methods

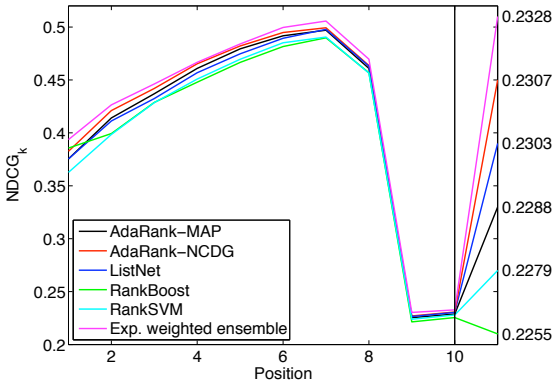
Figure 3.14 shows the $NDCG_k$ values for different truncation levels k. Our approach consistently outperforms the baseline methods for almost every truncation level on the LETOR 3.0 and MQ2008 datasets. In the case of LETOR 3.0, our method is noticeably better for high truncation levels whereas it outperforms only slightly but consistently the baseline methods on MQ2008. The picture is not so clear for MQ2007 because our approach shows some improvement only for low truncation levels.[19]



(a) LETOR 3.0/Ohsumed



(b) LETOR 4.0/MQ2007



(c) LETOR 4.0/MQ2008

Figure 3.14: $NDCG_k$ values on the LETOR datasets. We blow up the $NDCG_{10}$ values to see the differences.

Table 3.4 shows the average NDCG values for LETOR 4.0 along with $NCDG_{10}$ for LETOR 3.0 (the tool for LETOR 3.0 does not output the average NDCG). Our approach consistently achieves the best performance among all methods.

We evaluated the performance of our approach where we use only RBC or only CPC in the pool of scoring functions used in the aggregation step. We also evaluated the original ADABOOST.MH with decision tree and decision product using "traditional" validation, i.e., without our calibration and ensemble setup (last two rows in Table 3.4). The posteriors were calculated according to (3.51) and here we validated the i) number of iterations and ii) the hyperparameters of base learners (the number of leaves and number of product terms) in order to select a single best setting. These results shows that our meta-ensemble learning scheme where we calibrate

[19]The official evaluation tool of LETOR datasets returns with zero $NDCG_k$ value for a given query if k is bigger than the number of relevant documents. This results in poor performance of the ranking methods for $k \geq 9$ in the case of MQ2008. In our ensemble scheme, we used the $NDCG_{10}$ values calculated according to (3.48) with a truncation level of 10.

Table 3.4: The NDCG values for various ranking algorithms.

Method	Letor 3.0 Ohsumed	Letor 4.0 MQ2007	Letor 4.0 MQ2008
Eval. metric	$NDCG_{10}$	Avg. NDCG	Avg. NDCG
AdaRank-MAP	0.4429	0.4891	0.4915
AdaRank-NDCG	0.4496	0.4914	0.4950
ListNet	0.4410	0.4988	0.4914
RankBoost	0.4302	0.5003	0.4850
RankSVM	0.4140	0.4966	0.4832
Exp. w. ensemble	0.4561	0.4974	0.5006
Exp. w. ensemble with CPC	0.4621	0.4975	0.4998
Exp. w. ensemble with RBC	0.4493	0.4976	0.5004
AdaBoost.MH best Decision Tree	0.4164	0.4868	0.4843
AdaBoost.MH best Decision Product	0.4162	0.4785	0.4768

and combine the individual classifiers improves the standard AdaBoost.MH ranking setup significantly.

3.6 Accelerating testing using Markov decision processes

There are numerous applications where the computational requirements of classifying a test instance are as important as the performance of the classifier itself. Object detection in images [62] and web page ranking [64] are well-known examples. A more recent application domain with similar requirements is trigger design in high energy physics [77]. Most of these applications come with another common feature: the negative class (usually called noise or background) have sometimes orders of magnitudes higher probability than the positive class. Beside the test-time constraints, this also makes training difficult: traditional classification-error-based measures are not adequate, and using prior class probabilities in constructing training samples leads to either enormous data sizes or little representativity of the positive class.

A common solution to these problems is to design *cascade classifiers*. A cascade classifier consists of *stages*. In each stage a binary classifier attempts to eliminate background instances by classifying them negatively. Positive classification in inner stages sends the instance to the next stage, so detection can only be made in the last stage. By using simple and fast classifiers in the first stages, "easy" background instances can be rejected fast, shortening the expected testing time. The cascade structure also allows the use of different training sets at different stages, having more difficult background samples at later stages.

Classical cascade classifiers (such as the seminal Viola-Jones (VJ) algorithm [62]) have many disadvantages, succinctly summarized in [78]. First, only the binary decision of rejecting or not is carried down in the cascade, losing the information on the *margin* of the test instance in different stages. ChainBoost [79] resolves this problem by using a so-called *embedded cascade architecture* in which the output of a stage will be reused in the subsequent stages as well. Second, positive instances have to pass through all the stages for correct detection, which means that variability of the positive class (e.g. multi-view face detection) has to be handled by the last stage. To solve this problem, WaldBoost [80] decides between classifying an instance (either positively or negatively) or continuing evaluating the cascade based on the current margin of the instance. The approach of [81] is similar: they also look at the margin of the instance but the quitting/continuing decision is learned in a *Markov decision process* (MDP) framework. Third, the training process of the VJ cascade requires a lot of hand-tuning of control parameters, and it is non-trivial how to handle the trade-off between the performance and the complexity of the cascade. FCBoost [82] offers an improvement in this respect: it provides an automatic way for

controling the accuracy/complexity trade-off of the cascade through a hyperparameter. Finally, extending the cascade architecture to the multi-class case is non-trivial. For example in web page ranking, it is as crucial to make a fast prediction on the relevance of a web page to a query as in object detection [110], but, unlike in object detection, the human annotation often provides more than two relevance grades.

In this section we propose a method to overcome all these problems. As [81], we cast the cascade design problem in an MDP framework and use the current margin of the given instance to decide what to do. The main idea is to add a third action at each stage: beside quitting (classifying) and continuing, we also allow the classifier to *skip*, that is, continue without using the stage. This minor-looking design decision has numerous advantageous consequences. First, we can eliminate stages. Similarly to SOFTCASCADE [78], we will assume that we are given a sequence of *base-classifiers* sorted by importance or quality, but not organized into a small number of stages neither before nor during learning the cascade. Second, we can easily control the trade-off between the average number of *evaluated* base classifiers and the quality of the classification by combining these two competing goals into an appropriate reward. The form of the reward can also easily accommodate cost-sensitivity [82] of the base classifiers although we do not investigate this avenue in this section. Allowing skipping has also an unintended positive side-effect: the resulting classifier is *sparse*, moreover, the number and identities of base classifiers is data-dependent. Third, eliminating stages allows each instance to "decide" its own path within the list of base-classifiers. A-priori, we could have as many different paths as training instances, nevertheless, a-posteriori, we observe clustering in the "path-space". Moreover, when we artificially control the clusters within one of the classes, this observed clustering of paths corresponds to the injected sub-classes, which is an appealing feature for, e.g., multi-view object detection. Fourth, eliminating stages also greatly simplifies the design. Our algorithm is basically turn-key: it comes with an important design parameter (the trade-off coefficient between accuracy and speed) and a couple of technical hyperparameters of the MDP algorithm that can be kept constant across the benchmark problems we use. Finally, the multi-class extension of the technique is straightforward.

Another important consequence of introducing skipping is that the structure of the learned classifier is no longer a cascade neither a tree: it is a *directed acyclic graph* or a *decision DAG*. In fact, the main reason of sticking to the cascade design is that it is easy to control with semi-manual heuristics. Once the construction is automatic, keeping the cascade architecture is an unnecessary constraint.

MDDAG has several close relatives in the family of supervised methods. It is obviously related to algorithms from the vast field of sparse methods. The main advantage here is that the MDP setup allows us to achieve sparsity in a dynamical data-dependent way. This feature relates the technique to unsupervised sparse *coding* [111, 112] rather than to sparse classification or regression. On a more abstract level, MDDAG is also similar to [113]'s approach to "learn where to look". Their goal is to find a sequence of two-dimensional features for classifying images in a data-dependent way, whereas we do a similar search in a one-dimensional ordered sequence of features. Finally, the fact that our final classifier is a *tree* (more precisely, a directed acyclic graph) where decisions are made by accumulating base classifications *along* the whole path relates our technique to alternating decision trees (ADT) [114], except that ADTs use the output of the base classifiers directly for navigating in the tree, whereas in MDDAG the classifications are only indirectly coupled with control decisions.

The section is organized as follows. In Section 3.6.1 we describe the algorithm, in Section 3.6.2 we show experimental results, and we conclude in Section 3.6.3.

3.6.1 The MDDAG algorithm

Similarly to SOFTCASCADE [78], we describe MDDAG as a *post-processing* method that takes the output of a trained classifier and "sparsifies" it. Formally, we assume that we are given a sequence of N *base classifiers* $\mathcal{H} = (\mathbf{h}_1, \ldots, \mathbf{h}_N)$. Although in most of the cases cascades are built for binary classification, we describe the method for the more general multi-class case, which means

that $\mathbf{h}_j : \mathcal{X} \to \mathbb{R}^K$, where $\mathcal{X}$ is the input space and K is the number of classes. The semantics of $\mathbf{h}$ is that, given an observation $\mathbf{x} \in \mathcal{X}$, it *votes for* class ℓ if its ℓth element $h_\ell(\mathbf{x})$ is positive, and votes against class ℓ if $h_\ell(\mathbf{x})$ is negative. The absolute value $|h_\ell(\mathbf{x})|$ can be interpreted as the confidence of the vote. Although it is not a formal requirement, we will also assume that $\mathcal{H}$ is sorted in order of "importance" or performance of the base classifiers. These assumptions are naturally satisfied by the output of ADABOOST.MH [45] (Section 3.3.2), but in principle any algorithm that builds its final classifier as a linear combination of simpler functions can be used to provide $\mathcal{H}$. In the case of ADABOOST.MH or multi-class neural networks, the *final* (or *strong* or *averaged*) classifier defined by the full sequence $\mathcal{H}$ is $\mathbf{f}(\mathbf{x}) = \sum_{j=1}^{N} \mathbf{h}_j(\mathbf{x})$, and its prediction for the class index of $\mathbf{x}$ is $\widehat{\ell} = \arg\max_\ell f_\ell(\mathbf{x})$. In binary *detection*, $\mathbf{f}$ is usually used as a scoring function. The observation $\mathbf{x}$ is classified as signal if $f_1(\mathbf{x}) = -f_2(\mathbf{x}) > \theta$ and background otherwise. The threshold θ is a free parameter that can be tuned to achieve, e.g., a given false positive rate.

The goal of MDDAG is to build a *sparse* final classifier from $\mathcal{H}$ that does not use all the base classifiers. Moreover, we would like the selection to be data-dependent. For a given observation $\mathbf{x}$ we process the base classifiers in their original order. At each base classifier $\mathbf{h}_j$ we choose among three actions: 1) we either EVALuate $\mathbf{h}_j$ and continue, or 2) we SKIP $\mathbf{h}_j$ and continue, or 3) we QUIT and return the classifier built so far. Let

$$b_j(\mathbf{x}) = 1 - \mathbb{I}\left\{a_j = \text{SKIP} \;\text{ OR }\; \exists j' < j : a_{j'} = \text{QUIT}\right\} \tag{3.55}$$

be the indicator that $\mathbf{h}_j$ is evaluated, where $a_j \in \{\text{EVAL}, \text{SKIP}, \text{QUIT}\}$ is the action taken at step j and the indicator function $\mathbb{I}\{A\}$ is 1 if its argument A is true and 0 otherwise. Then the final classifier built by the procedure is

$$\mathbf{f}^{(N)}(\mathbf{x}) = \sum_{j=1}^{N} b_j(\mathbf{x})\mathbf{h}_j(\mathbf{x}). \tag{3.56}$$

Inspired by WALDBOOST [80] and the method of [81], the decision on action a_j will be made based on the index of the base classifier j and the output vector of the classifier

$$\mathbf{f}^{(j)}(\mathbf{x}) = \sum_{j'=1}^{j} b_{j'}(\mathbf{x})\mathbf{h}_{j'}(\mathbf{x}). \tag{3.57}$$

built up to step j.[20] Formally, $a_j = \pi\big((\mathbf{s}_j(\mathbf{x})\big)$, where

$$\mathbf{s}_j(\mathbf{x}) = \left(f_1^{(j-1)}(\mathbf{x}), \ldots, f_K^{(j-1)}(\mathbf{x}), j-1\right) \in \mathbb{R}^K \times \mathbb{N}^+ \tag{3.58}$$

is the *state* where we are before visiting $\mathbf{h}_j$, and π is a *policy* that determines the action in state $\mathbf{s}_j$. The initial state $\mathbf{s}_1$ is the zero vector with $K+1$ elements.

This setup formally defines a Markov decision process (MDP). An MDP is a 4-tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$, where $\mathcal{S}$ is the (possibly infinite) state space and $\mathcal{A}$ is the countable set of actions. $\mathcal{P} : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to [0,1]$ is the transition probability kernel which defines the random transitions $\mathbf{s}^{(t+1)} \sim \mathcal{P}(\cdot|\mathbf{s}^{(t)}, a^{(t)})$ from a state $\mathbf{s}^{(t)}$ applying the action $a^{(t)}$, and $\mathcal{R} : \mathbb{R} \times \mathcal{S} \times \mathcal{A} \to [0,1]$ defines the distribution $\mathcal{R}(\cdot|\mathbf{s}^{(t)}, a^{(t)})$ of the *immediate reward* $r^{(t)}$ for each state-action pair. A *deterministic policy* π assigns an action to each state $\pi : \mathcal{S} \to \mathcal{A}$. We will only use *undiscounted* and *episodic* MDPs where the policy π is evaluated using the *expected sum of rewards*

$$\varrho = \mathbb{E}\left\{\sum_{t=1}^{T} r^{(t)}\right\} \tag{3.59}$$

[20] When using ADABOOST.MH, the base classifiers are binary $\mathbf{h}_j(\mathbf{x}) = \{\pm\alpha_j\}^K$, and we normalize the output (3.57) by $\sum_{j=1}^{N} \alpha_j$, but since this factor is constant, the only reason to do it is to make the range of the state space uniform across experiments.

with a finite horizon T. In the episodic setup we also have an *initial* state ($\mathbf{s}_1$ in our case) and a *terminal* state $\mathbf{s}_\infty$ which is impossible to leave.

In our setup, the state $\mathbf{s}^{(t)}$ is equivalent to $\mathbf{s}_j(\mathbf{x})$ (3.58) with $j = t$. The action QUIT brings the process to the terminal state $\mathbf{s}_\infty$. Figure 3.15 illustrates the MDP designed to learn the sparse stageless DAG.

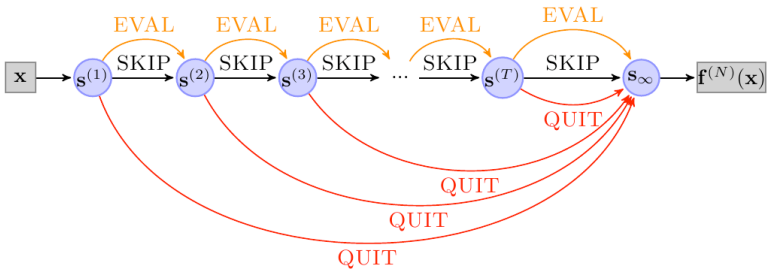


Figure 3.15: The schematic overview of the process. The input is an instance $\mathbf{x}$ to be classified. During the process the policy decides which base classifier will be evaluated by choosing one of the three available actions in each state. The output is the score vector provided by the classifier $\mathbf{f}^{(N)}(\mathbf{x})$ (3.56).

To define the rewards, our primary goal is to achieve high accuracy, so we penalize the error of $\mathbf{f}^{(t)}$ when the action $a^{(t)} = \text{QUIT}$ is applied. For the formal definition, we suppose that, at training time, the observations $\mathbf{x}$ arrive with the *index* $\ell \in \{1, \dots, K\}$ of their class. The *multi-class margin* is defined as

$$\rho^{(t)}(\mathbf{x}, \ell) = f_\ell^{(t)}(\mathbf{x}) - \max_{\ell' \neq \ell} f_{\ell'}^{(t)}(\mathbf{x}).$$

With these notations, the classical (one-loss) multi-class reward for the QUIT action is

$$r_{\mathbb{I}}^{(t)}(\mathbf{x}, \ell) = \mathbb{I}\left\{\rho^{(t)}(\mathbf{x}, \ell) > 0\right\}. \tag{3.60}$$

For well-known reasons we also use the convex upper bound

$$r_{\text{EXP}}^{(t)}(\mathbf{x}, \ell) = \exp\left(\rho^{(t)}(\mathbf{x}, \ell)\right). \tag{3.61}$$

In principle, any of the usual convex upper bounds (3.5, 3.6) could be used in the MDP framework. The exponential reward is inspired by the setup of ADABOOST [44, 45]. The rewards can easily be adapted to *cost-sensitive* classification when the misclassification cost is different for the classes, although we do not explore this issue here. Note also that in the binary case, $r_{\mathbb{I}}$ and r_{EXP} recover the classical binary notions. From now on we will refer to our algorithm as MDDAG.$\mathbb{I}$ when we use the indicator reward $r_{\mathbb{I}}$ and MDDAG.EXP when we use the exponential reward r_{EXP}.

For encouraging sparsity, we will also penalize each evaluated base classifier $\mathbf{h}$ by a uniform fixed negative reward

$$\mathcal{R}(r|\mathbf{s}, \text{EVAL}) = \delta(-\beta - r), \tag{3.62}$$

where δ is the Dirac delta and β is a hyperparameter that represents the accuracy/speed trade-off.

The goal of reinforcement learning (RL) in our case is to learn a policy which maximizes the expected sum of rewards (3.59). Since in our setup the transition $\mathcal{P}$ is deterministic *given* the observation $\mathbf{x}$, the expectation in (3.59) is taken with respect to the random input point $(\mathbf{x}, \ell)$. This means that the global goal of the MDP is to maximize

$$\varrho = \mathbb{E}_{(\mathbf{x},\ell)\sim\mathfrak{D}}\left\{r(\mathbf{x}, \ell) - \beta \sum_{j=1}^{N} b_j(\mathbf{x})\right\} \tag{3.63}$$

where $r(\mathbf{x}, \ell)$ is one of our margin-based rewards and $\mathfrak{D}$ is the distribution that generates the instances.

Note that, strictly speaking, the rewards (3.60) and (3.61) are not stationary given the state $\mathbf{s}^{(t)}$: it is possible that two different policies bring two different subsets of $\mathcal{X}$ with different label distributions into the same state. This problem could be tackled in an *adversarial* setup, recently analyzed by [115]. However, the results of [115] are rather theoretical, and we found that, similarly to [81] whose method also suffers from non-Markovianness, classical MDP algorithms work well for solving our problem.

Learning the policy

There are several efficient algorithms to learn the policy π using an iid sample

$$\mathcal{D} = \big((\mathbf{x}_1, \ell_1), \ldots, (\mathbf{x}_n, \ell_n)\big)$$

drawn from $\mathfrak{D}$ [116]. When $\mathcal{P}$ and $\mathcal{R}$ are unknown, model-free methods are commonly used for learning the policy π. These methods directly learn a *value function*, (the expected reward in a state or for a state-action pair), and derive a policy from it. Among model-free RL algorithms, *temporal-difference* (TD) learning algorithms are the most commonly used. They can be divided into two groups: *off-policy* and *on-policy* methods. In the case of off-policy methods the policy search method learns about one policy while following another, whereas in the on-policy case the policy search algorithm tries to improve the current policy by maintaining sufficient exploration. On-policy methods have an appealing practical advantage: they usually converge faster to the optimal policy than the off-policy methods.

We use the SARSA(λ) algorithm [117] with *replacing traces* to learn the policy π. For more details we refer the reader to [118]. SARSA(λ) is an on-policy method, so, to make sure that all policies can be visited with nonzero probability, we use an ϵ-greedy exploration strategy. Concretely, we apply SARSA in an episodic setup: we use a random training instance $\mathbf{x}$ from $\mathcal{D}$ per episode. The instance follows the current policy with probability $1 - \epsilon$ and chooses a random action with probability ϵ. The instance observes the immediate rewards (3.60), (3.61), or (3.62) after each action. The policy is updated during the episode according to SARSA(λ). In preliminary experiments we also tested Q-LEARNING [119], one of the most popular off-policy methods, but SARSA(λ) slightly but consistently outperformed Q-LEARNING.

In all experiments we used ADABOOST.MH to obtain a pool of weak classifiers $\mathcal{H}$. We ran ADABOOST.MH for $N = 1000$ iterations, and then trained SARSA(λ) on the same training set. The hyperparameters of SARSA(λ) were fixed across the experiments. We set λ to 0.95. In principle, the learning rate should decrease to 0, but we found that this setting forced the algorithm to converge too fast to suboptimal solutions. Instead we set the learning rate to a constant 0.2, we evaluated the current policy after every 10000 episodes, and we selected the best policy based on their performance also on the training set (overfitting the MDP was a no-issue). The exploration term ϵ was decreased gradually as $0.3 \times 1/\lceil \frac{10000}{\tau} \rceil$ where τ is the number of training episodes. We trained SARSA(λ) for 10^6 episodes.

As a final remark, note that maximizing (3.63) over the data set $\mathcal{D}$ is equivalent to minimizing a margin-based loss with an L_0 constraint. If r_{I} (3.60) is used as a reward, the loss is also non-convex, but minimizing a loss with an L_0 constraint is NP-hard even if the loss is convex [120]. So, what we are aiming for is an MDP-based heuristics to solve an NP-hard problem, something that is not without precedent [121]. This equivalence implies that even though the algorithm would converge in the ideal case (decreasing learning rate, proper Markovian rewards), in principle, convergence can be exponentially slow in n. In practice, however, we had no problem finding good policies in reasonable training time.

3.6.2 Experiments

In the next subsection we first verify the sparsity and heterogeneity hypotheses on a synthetic toy example. In the second subsection we compare MDDAG to state-of-the-art cascade detectors on

three object detection benchmarks. Finally, in the third subsection we show how the multi-class version of MDDAG performs on a benchmark web page ranking problem.

Synthetic data

The goal of this experiment was to verify whether MDDAG can learn the subset of "useful" base classifiers in a *data-dependent* way. We created a two-dimensional binary dataset with real-valued features where the positive class is composed of two well-separable clusters (Figure 3.16(a)). This is a typical case where ADABOOST or a traditional cascade is suboptimal since they both have to use *all* the base classifiers for all the positive instances [78].

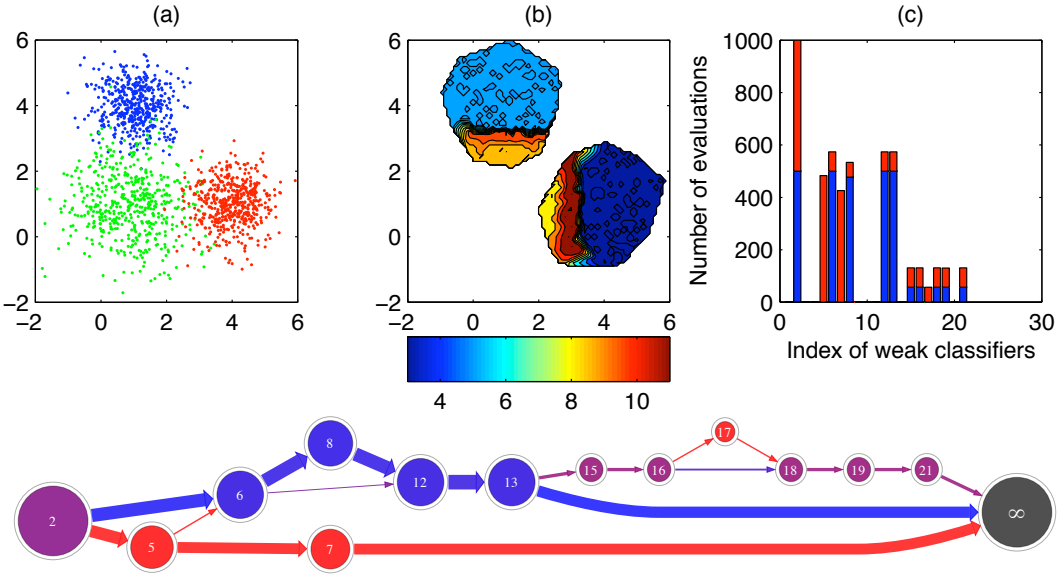


Figure 3.16: Experiments with synthetic data. (a) The positive class is composed of the blue and red clusters, and the negative class is the green cluster. (b) The number of base classifiers used for each individual positive instance as a function of the two-dimensional feature coordinates. (c) The number of positive instances from the blue/red clusters on which a given base classifier was applied according to the policy learned by MDDAG.𝕀. Lower panel: the decision DAG for the positive class. Colors represent sub-class probabilities (proportions) and the node sizes and arrow widths represent the number of instances in the states and following the actions, respectively.

We ran MDDAG.𝕀 with $\beta = 0.01$ on the 1000 decision stumps learned by ADABOOST.MH. In Figure 3.16(b) we plot the number of base classifiers used for each individual positive instance as a function of the two-dimensional instance itself. As expected, the "easier" the instance, the less base classifiers it needs for classification. Figure 3.16(c) confirms our second hypothesis: base classifiers are used selectively, depending on whether the positive instance is in the blue or red cluster.

Binary detection benchmarks

In these experiments we applied MDDAG on three image data sets often used for benchmarking object detection cascades. VJ [62] and CBCL[21] are face recognition benchmarks, and DPED [122] is a pedestrian recognition data set. We divided the data sets into training and test set.

We compared MDDAG to three state-of-the-art object detection algorithms: the original Viola-Jones cascade VJCASCADE [62], FCBOOST [82], and SOFTCASCADE [78]. VJCASCADE

[21] http://cbcl.mit.edu/software-datasets/FaceData2.html

builds the cascade stage-by-stage by running AdaBoost in each stage. It stops adding base classifiers to the mth stage when the false positive rate (FPR) falls below p^m_{FPR} and true positive rate (TPR) exceeds p^m_{TPR}, where p_{TPR} and p_{FTR} are hyperparameters of the algorithm. The total number of stages is also a hyperparameter. FCBoost also adds base classifiers iteratively to the cascade, but the base classifier can be inserted into any of the stages. The goal is to minimize a global criterion that, similarly to (3.63), is composed of a performance based term and a complexity based term. The number of iterations and the parameter η that determines the trade-off between the two competing objectives are hyperparameters of the algorithm. SoftCascade, as MDDAG, builds a cascade on the output of AdaBoost where each stage consists of exactly one base classifier. The final number of base classifiers are decided beforehand by the hyperparameter N. In each iteration j, the base classifier with the highest balanced edge is selected, and the detection threshold θ_j is set to achieve a TPR of $1 - \exp(\alpha j/N - \alpha \mathbb{I}\{\alpha < 0\})$, where α is a second hyperparameter of the algorithm. Both the TPR and the number of base classifiers increase with α, so, although not as explicitly as our β or FCBoost's η, the choice of α influences the speed/accuracy trade-off.

Comparing test-time-constrained detection algorithms is quite difficult. The usual trade-off between false positive rate (FPR) and true positive rate (TPR) can be captured by ROC curves, but here we also have to take into account the computational efficiency of the detector. In [78] this problem is solved by showing three-dimensional FPR/TPR/number-of-features surfaces. Here we decided to show two-dimensional slices of these surfaces: we fix the FPR to reasonable values and plot the TPR against the detection time. In all the algorithms we used Haar features as base classifiers, so the detection time can be uniformly measured by the average number of base classifiers needed for detection. In typical detection problems the number of background (negative) instances is orders of magnitudes higher than the number of signal (positive) instances, so we computed this average only over the negative test set. It turns out that using this measure, vanilla AdaBoost is quite competitive to tailor-made cascade detectors, so we also included it in the comparison.

Computing the TPR vs. number-of-features curve at a fixed FPR cannot be done in a generic algorithm-independent way. For AdaBoost, in each iteration j (that is, for each number j of base classifiers) we tune the detection threshold θ to achieve the given test FPR, and plot the achieved test TPR vs. j. In the other three algorithms we have 2-3 hyperparameters that explicitly or implicitly influence the average number of base classifiers and the performance. We ran the algorithms using different hyperparameter combinations. In each run we set the detection threshold θ to achieve the given test FPR. With this threshold, each run k determines a TPR/average-number-of-features pair (p_k, N_k) on the training set and (p'_k, N'_k) on the test set. For each N we find the run $k^*(N) = \arg\max_{k:N_k \le N} p_k$ that achieves the best training p_k using at most N base classifiers, and plot the test TPR $p'_{k^*(N)}$ vs. N. Using the training scores for hyperparameter selection is usually prone to overfitting, but in the case of cascade detectors the complexity of the classifiers is so low that this is not an issue.

Figure 3.17 shows the results. Although the differences are quite small, MDDAG outperforms the three benchmarks algorithms consistently in the regime of low number base classifiers, and it is competitive to them on the full range. MDDAG.$\mathbb{I}$ is slightly better at low complexities whereas MDDAG.EXP is more effective at slightly higher number of base classifiers. This is not surprising: in the low complexity regime the 0-1 error is more aggressive and closer to the measured TPR, whereas when most of the instances are classified correctly, without the margin information it is impossible to further improve the policy.

Ranking with multi-class DAGs

Although object detection is arguably the most well-known test-time-constrained problem, it is far from being the only one. In web page ranking (Section 3.5) the problem is similar: training time can be almost unlimited, but the learned ranker must be fast to execute. One of the difficulties in this case is that relevance labels can be non-binary, so classical object-detection cascades

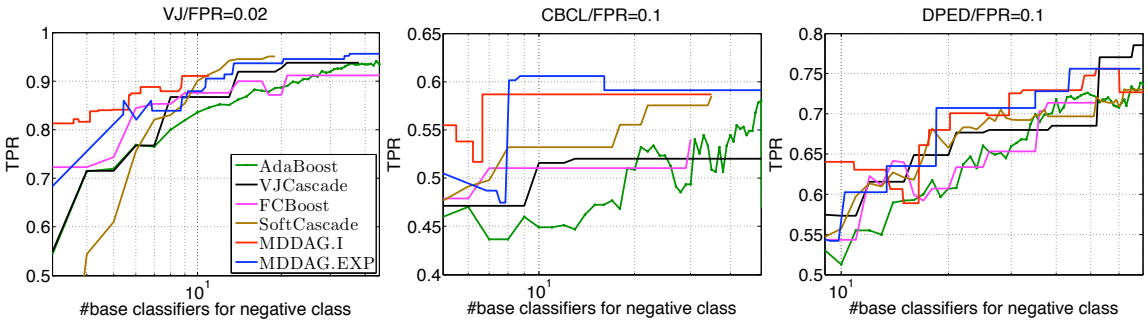


Figure 3.17: The true positive rate (TPR) vs. the average number of base classifiers evaluated on negative test instances. The results are computed for different fixed false positive rates (FPR) shown in the title of panels.

are unusable. At the same time, the principles used to design cascades re-surface also in this domain, although the setup is rather new and the algorithms require quite a lot of manual tuning [123]. On the other hand, MDDAG can be used for this task *as is*.

To evaluate MDDAG on a multi-class classification/ranking problem, we show results on the MQ2007 and MQ2008 data sets taken from LETOR 4.0.[22] In web page ranking, observations come in the form of query-document pairs, and the performance of the ranker is evaluated using tailor-made loss or gain functions that take as input the ordering of all the documents given a query. To train a ranker, query-document pairs come with manually annotated *relevance labels* that are usually multi-valued ($\{0,1,2\}$ in our case). One of the most common performance measures is the Normalized Discounted Cumulative Gain (NDCG$_m$) [124] which is based on the first m documents in the order output by the ranker. We used the *averaged* NDCG score $\overline{\text{NDCG}}$, provided by LETOR 4.0, that takes a weighted average of first ten NDCG$_m$ values to evaluate the algorithms. In these experiments the base learners were decision trees (Section 3.4.4) with eight leaves.

The goal of MDDAG is similar to the binary case: to achieve a comparable performance to AdaBoost.MH using less base learners. To make the comparison fair, we use the same calibration method (Section 3.5.3) to convert the output of the multi-class classifiers to a scoring function and then to a ranking [98]. Since the goal this time is not detection, we simply evaluate the average NDCG for each run k to obtain $(\overline{\text{NDCG}}_k, N_k)$ on the training set and $(\overline{\text{NDCG}}'_k, N'_k)$ on the test set. We then select $k^*(N) = \arg\max_{k:N_k \le N} \overline{\text{NDCG}}_k$ and plot the test $\text{NDCG}'_{k^*(N)}$ vs. N. Figure 3.18 shows that MDDAG performs as well as AdaBoost.MH with roughly two-fold savings in the number of base classifiers.

3.6.3 Conclusions and future work

In this section we described an MDP-based design of decision DAGs. The output of the algorithm is a data-dependent sparse classifier which means that every instance "chooses" the base classifiers or features that it needs for predicting its class index. The algorithm is competitive to state-of-the-art cascade detectors on object detection benchmarks, and it is also directly applicable to test-time-constrained problems involving multi-class classification (e.g., web page ranking). In our view, however, the main advantage of the algorithm is not necessarily its performance but its simplicity and versatility. First, MDDAG is basically a turn-key procedure: it comes with one user-provided hyperparameter with a clear semantics of directly determining the accuracy/speed trade-off. Second, MDDAG can be easily extended to problems different from classification by redefining the rewards on the Quit and Eval actions. For example, one can easily design *regression* or *cost-sensitive* classification DAGs by using an appropriate reward in (3.60), or add a weighting to (3.62) if the features have different evaluation costs.

[22] http://research.microsoft.com/en-us/um/beijing/projects/letor

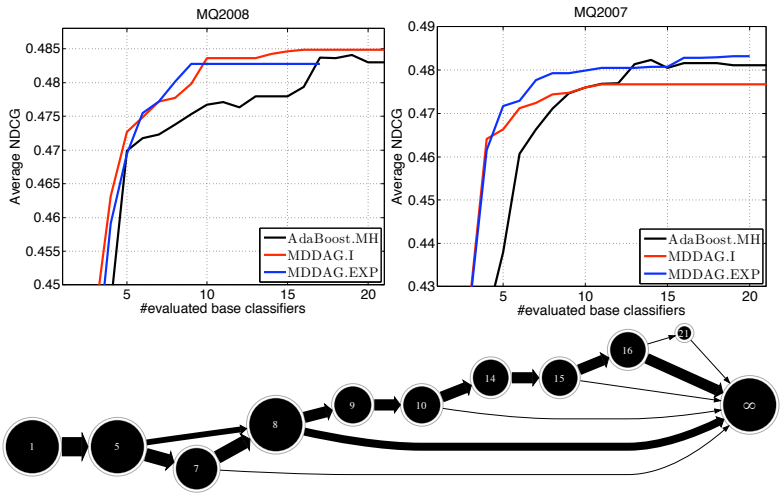


Figure 3.18: The average NDCG vs. the average number of base classifiers evaluated on test queries, and one of the decision DAGs generated by MDDAG.EXP on MQ2008.

It is exactly because of the flexibility of the algorithm that we have numerous avenues to explore. Beside the straightforward extensions mentioned above, MDDAG should also be particularly useful in multi-view object detection when the positive class has considerable variability. Exploring the connections between MDDAG and sparse coding is also an interesting direction. At this point the algorithm is designed for post-processing a fixed output of another learning method. Beside trying it on the output of different methods (e.g., for filtering support vectors), it would also be interesting to see if the filtering mechanism could be used within the learning loop. Finally, it is an open question at this point whether the RL algorithm converges under the condition of our specific non-Markovian rewards; proving convergence is an interesting challenge.

3.7 Other contributions to supervised learning

In this section we briefly summarize some of our most significant contributions to supervised learning that we have not detailed in previous sections.

In [125, 126] we combine margin theory and data-dependent complexities to improve generalization error bounds in the framework of supervised classification. The main feature of these bounds is that they can be calculated using the training data and therefore may be effectively used for model selection purposes. In [127] we extend ADABOOST to regression in a natural manner, and prove the algorithmic convergence of the algorithm. We also analyze the theoretical properties of the algorithm in a separate paper [128]. In [129] we develop a natural way to incorporate traditional complexity penalization into the boosting framework. In [130] we consider a distributed learning model in which participants want to learn a common task without sharing their training data (for example, two banks want to automatically detect fraudulent credit card transactions based on historical data without sharing secret client records). We extended ADABOOST to this case and show that the participants can significantly improve their classifiers by cooperating, even when they share very little direct information on their training data. In [131] we show how to accelerate ADABOOST using multi-armed bandits (MABs). The basic idea is that we gradually learn the utility of the features (or, in general, subsets of base learners) using MABs, and use this utility to bias the random selection in each boosting iteration. In [132] we improve on [131] by using an adversarial bandit algorithm instead of stochastic bandits. The practical performance is comparable but this choice allows us to prove a weak-to-strong-learning theorem, which means that the proposed technique remains a boosting algorithm in a formal sense.

Chapter 4

Bayesian inference in the Pierre Auger experiment: modeling and computational challenges

This chapter describes some of our contributions to the data analysis of the Pierre Auger experiment. The experiment studies high-energy cosmic rays by observing the secondary particle cascade that these cosmic rays generate when they interact with the atmosphere of the earth. Our long-term goal in the Auger collaboration is to develop a comprehensive generative model for these so-called particle showers and the signal they make in the detectors, and use Bayesian techniques to infer the parameters of the primary cosmic ray particle (energy, arrival direction, and chemical composition) as well as the parameters of the physical model that describes the first high-energy interaction between the primary particle and the atomic nucleus it collides with. Along the way of accomplishing this goal, we would also like to develop novel inference techniques that can deal with the challenges posed by large-scale data analysis problems.

At this point the project resembles a giant interdisciplinary jigsaw puzzle with some parts more-or-less finished and others completely missing. The unfinished parts also contain some unknown unknowns especially on the computational feasibility side. To avoid scientific frustration due to the immense task ahead of us, we try to make the process of *getting there* as much fun as possible and contribute standalone parts of the project independently of the master plan. The sections of this chapter summarize some of the subparts, mainly those in which we have made the most progress.

After a short introduction to the Pierre Auger experiment in Section 4.1, we describe the generative model of of the surface detector signal given some parameters of the incoming secondary particles (Section 4.2, [133, 134]). This model is the first step in creating a bottom-up model for the full generative process: we will need to parametrize at least two more levels of the graphical model to get to the parameters of the primary cosmic ray particle. Nevertheless, the full parametrization of the detector already allows us to test the inference techniques being developed in parallel. Not having had background in experimental physics, developing the formal model also helped us to understand the physical models relevant to the inference problem.

In the classical machine learning setup, models (priors, likelihoods) are usually chosen for inferential convenience. In our case models come from physical processes, so we have little freedom in choosing parametrizations to make the inference easy. Our inference method of choice is therefore numerical integration, more precisely, MCMC-type numerical integration. The enormous data produced by the experiment (more than 40K showers and 200K detector signals) makes Bayesian inference computationally challenging. First, we have to be able to *parallelize* the inference, and second, we have to make it *automatic* (no manual tuning). The adaptive Metropolis algorithm [135] is a natural choice for this latter reason, but we rapidly realized that it is not completely adapted for doing inference in our mixture models. In Section 4.3 we describe a contri-

bution [136] that solves the label switching problem, optimizing adaptive Metropolis for mixture models. For counting the mixture components, our plan is to develop an adaptive version of the *reversible jump* MCMC algorithm [137] although we also leave other avenues open.

In Section 4.4 we summarize a nonparametric multivariate approach for counting particles in the detector signal (i.e., mixture components), developed in the first half of the note [72]. In a sense, it is a poor man's solution for the difficult inference problem that we plan to solve with adaptive Metropolis once all the modeling and computational challenges will have been solved. In the meantime, the approach described in this section is the best we can to come up within the "classical" paradigm of attacking the inverse problem directly.

Section 4.5 contains the second part of [72]: we describe an empirical Bayes [138, 139] approach to estimate the lateral distribution function of the particle cascade. Beside producing our first "real" physics results, this method is also a precursor for solving the parallelization problem for the future Bayesian treatment of the full inference problem.

4.1 The Pierre Auger experiment

The existence of cosmic rays has been known since the daring balloon experiments of Victor Hess in beginning of the 20th century. Since the initial discovery we have learned a lot about cosmic rays: we know that they are sub-atomic particles (electrons, protons, and nuclei of heavier elements up to iron and even uranium) with energies that vary on a large scale (from a few billion eV to more than 10^{20} eV – the energy of a tennis ball flying at 200 km per hour!). Cosmic rays are produced by known or unknown astrophysical mechanisms, so studying the composition, the energy, and the sources of these particles is important for understanding the universe tracing back to its origins.

The most interesting and enigmatic cosmic particles are those with the highest energies. Whereas there are known mechanisms that produce particles up to 10^{15} eV, the acceleration mechanisms involved in producing the highest energy cosmic rays are still unknown. These particles are almost certainly extra-galactic, and one intriguing possibility is that they are generated by very massive particles produced at the beginning of time. Some of the recent experiments seem to measure energies that are higher than the so-called "GZK cut-off" (4×10^{19} eV), a theoretical limit derived from the deceleration effect of the cosmic background radiation (the light left over from the big-bang). While they are interesting, high energy cosmic ray particles are also extremely rare: they arrive at a rate of a few per km^2 per century. The low rate makes direct detection (usually by high altitude air balloons) impossible. Fortunately, when one if these particles collides with the atmosphere, it generates a huge cascade of atmospheric particles, a so-called "extensive air shower" (hereafter referred to as the "shower"), that covers several square kilometers on the Earth's surface.

The objective of the Pierre Auger experiment [73] is to study the properties of ultra-high energy cosmic ray particles by observing the showers. To obtain reasonable statistics at the higher end of the spectrum, the detector has to be huge. Indeed, the Auger detector, built on the Argentinean Pampas, covers 3000 km^2. The detector contains two independent measuring devices, a surface detector consisting of 1600 water tanks placed on hexagonal grid at a 1.5 km resolution, and a fluorescence detector consisting of 24 fluorescence telescopes placed at the edges of the detector area, looking inside (Figure 4.1). The fluorescence detector is able to reconstruct the longitudinal development of the shower high in the atmosphere, whereas the surface detector samples the shower particles as they hit the surface of the earth. Using two independent measurement devices is a great way to calibrate different estimation techniques to each other.

The goal of the statistical data analysis is to estimate the four generating parameters of the cosmic particle (two angular directions, energy, and type of nucleus) based on two independent measurements (surface detectors and fluorescence detectors). The analysis faces huge challenges: the air-cascade is an intrinsically probabilistic process and only partially understood, atmospheric effects are barely known, and real measurements deviate seriously from the simulations. Since the measured signal is not connected directly to the parameters of the cosmic particle, the "tradi-

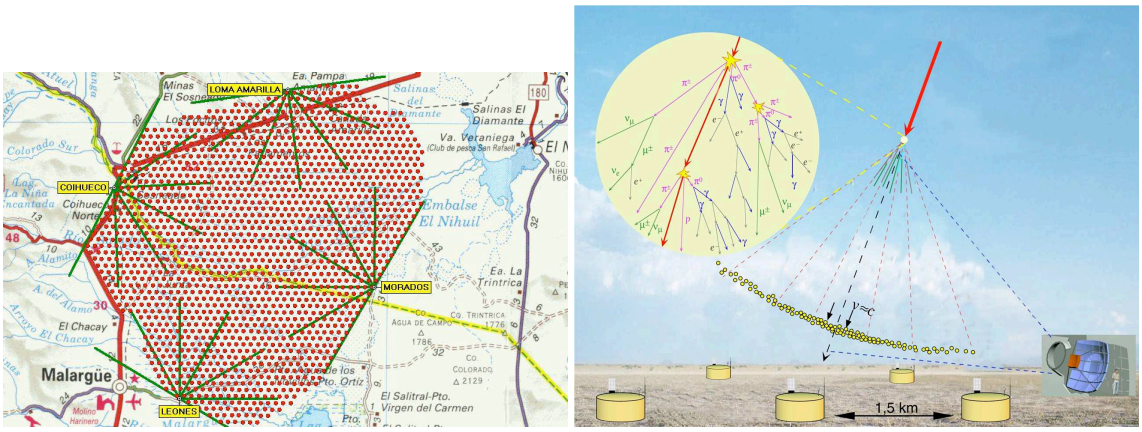


Figure 4.1: The Pierre Auger Observatory in Malargüe, Argentina. The bird-eye view of the detector (left panel) and a schematic representation of detecting an air shower (right panel).

tional" approach is to estimate separately a series of hidden variables with well-defined physical meanings and use the non-probabilistic estimates as "observations" to determine higher level variables.

Our main goal is the estimation of one of the generating parameters: the chemical composition of the cosmic rays. The main candidates are protons (hydrogen nuclei[1]) and iron nuclei, although other stable elements cannot be excluded either. Whereas the traditional direct approach works well for estimating the other generative parameters (the incoming angles and the energy; although for this latter we start observing the limitations of the standard reconstruction), the estimation of the chemical composition seems very difficult.

There are two basic observables that are used traditionally to estimate the type of incoming particle. In general, heavier nuclei collide earlier in the atmosphere, so they generate shallower showers with small population variance, whereas proton showers are deeper and vary more. The shower maximum can be relatively well-estimated using the fluorescence detector, but this sub-detector works only on clear moonless nights (about 10% of the time). On the other hand, the surface detector works all the time but it has only a limited view of the shower. The main surface detector observable sensitive to the chemical composition is the number of muons at the surface: in general, iron showers generate about 40% more muons as proton showers. There are two major problems that this analysis has to face. First, although individual muon signals have a unique "signature", the water tank signals are mixtures of individual muon signals and a low background of an electromagnetic component (Figure 4.2). The second problem is more fundamental: even though there is a 40% relative margin between the muon density of an iron and a proton shower, the absolute muon numbers in the simulations seem to be shifted by a constant that is comparable to this 40%. The reason of this shift is unknown: it is probably related to the deficiencies of the hadronic models applied in the air shower simulators. The main consequence of this fact that the traditional discriminative approach of training on simulation for classifying real events is infeasible.

4.2 The single muon response model

When muons cross the water tank, they emit Cherenkov photons at a rate that depends on their energy. These photons are emitted at a precise angle along the muon trajectory. Most of them are absorbed by the water after a number of reflections on the walls of the tank, but some of them are detected by one of the three photodetectors (PMTs) placed below the lid of the tank,

[1]When accelerated to these energies, the atoms lose their electrons.

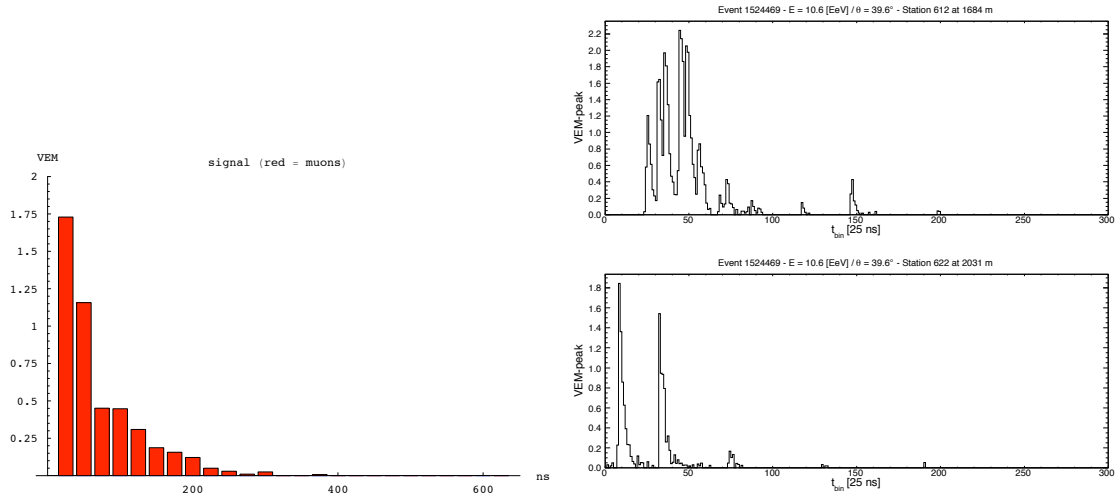


Figure 4.2: A simulated signal of an individual muon (left panel) and two typical surface detector signals of real events at different core distances (right panel).

looking down into the water. Some of the detected photons generate an analog signal through the emission of photoelectrons (PEs) which is then digitized and recorded as a vector of integers $\mathbf{x} = (x_1, \ldots, x_N)$. We interpret x_i as the sum of the signals deposited by photoelectrons, corrupted by noise, in the time interval

$$[t_{i-1}, t_i) \triangleq [t_0 + (i-1)t_\Delta, t_0 + it_\Delta), \tag{4.1}$$

where t_0 is the absolute starting time of the signal, and $t_\Delta = 25\,\text{ns}$ is the signal resolution (size of one bin). The signal is measured in adc unit[2].

The goal of this section is to formalize this so called FADC signal generated by an individual muon. The "input" to this generative model is the time of arrival t_μ of the muon, its kinetic energy T_μ, and its zenith angle θ_μ, so the formal goal of this section is to describe

$$p(\mathbf{x}|t_\mu, T_\mu, \theta_\mu). \tag{4.2}$$

Since the rate at which Cherenkov photons are emitted is almost constant, the amplitude of the muon signal is mostly dependent on the tracklength L of the muon in the water tank. The photon emission rate[3] also depends on the energy T_μ of the muon. We will model this dependence by introducing a unitless factor ϕ such that the average emission rate of a muon with kinetic energy $T_\mu = 1\,\text{GeV}$ is set to $\phi = 1$. Since the distribution of the tracklength L depends only on the zenith angle, and ϕ depends only on the energy of the muon[4], (4.2) factorizes into three basic terms

$$p(\mathbf{x}|t_\mu, T_\mu, \theta_\mu) = \int p(\mathbf{x}|t_\mu, L, \phi)p(L|\theta_\mu)p(\phi|T_\mu)\mathrm{d}L\mathrm{d}\phi.$$

We treat these three modules in the following three subsections.

4.2.1 The tracklength distribution

In this section we derive the tracklength distribution $p(L|\theta)$ of a muon that crosses the tank at zenith angle θ [133]. The distribution is purely geometric: it depends on the height and radius

[2] It represents either voltage or current integrated by the FADC inside the bin width t_Δ. The true unit is thus V ns or A ns.

[3] mostly Cherenkov but also due to other physical processes

[4] not considering second order dependencies of the effective rate on the tracklength

of the tank which are, in the case of Auger, $h = 1.2\,\mathrm{m}$ and $R = 1.8\,\mathrm{m}$, respectively. The distribution can be decomposed into three components: the *lid–base* (lb) term of the form $p_{\mathrm{lb}}(L|\theta)\,P(\mathrm{lb}|\theta)$ representing muons entering at the lid of the tank and leaving at the base, the *lid–lateral* (li) term $p_{\mathrm{li}}(L|\theta)\,P(\mathrm{li}|\theta)$ representing muons entering at the lid and leaving at the lateral (or, symmetrically, entering at the lateral and leaving at the base), and the *lateral–lateral* (la) term $p_{\mathrm{la}}(L|\theta)\,P(\mathrm{la}|\theta)$ representing muons entering and leaving at the lateral (for these regions see projected tank schematic in Fig. 4.3). We have thus

$$p(L|\theta) = p_{\mathrm{lb}}(L|\theta)\,P(\mathrm{lb}|\theta) + p_{\mathrm{li}}(L|\theta)\,P(\mathrm{li}|\theta) + p_{\mathrm{la}}(L|\theta)\,P(\mathrm{la}|\theta). \tag{4.3}$$

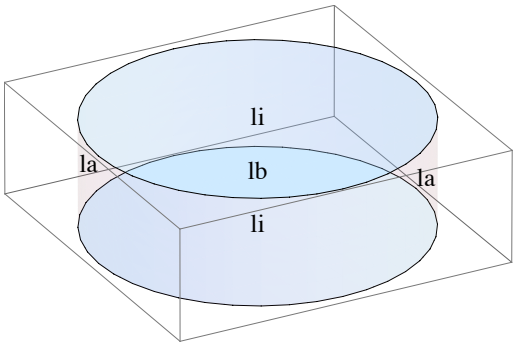


Figure 4.3: Projection of the Auger tank along the incoming muon direction at $\theta = 50^\circ$. The three components of the tracklength distribution $p(L|50^\circ)$ are visible: the *lid–base* (lb), the *lid–lateral* (li), and the *lateral–lateral* (la) terms.

Setting

$$u = \frac{h}{2R}|\tan\theta|,$$

the case probabilities are given by

$$P(\mathrm{lb}|\theta) = \frac{A_{\mathrm{lb}}(\theta)}{A(\theta)}, \qquad P(\mathrm{li}|\theta) = \frac{A_{\mathrm{li}}(\theta)}{A(\theta)}, \quad \text{and} \quad P(\mathrm{la}|\theta) = \frac{A_{\mathrm{la}}(\theta)}{A(\theta)},$$

where

$$A(\theta) = 2R^2|\cos\theta|(\tfrac{\pi}{2} + 2u)$$

is the area of the tank projection for zenith angle θ, and

$$\begin{aligned}
A_{\mathrm{lb}}(\theta) &= 2R^2|\cos\theta| \begin{cases} \arccos u - u\sqrt{1-u^2} & \text{if } u \le 1, \\ 0 & \text{otherwise,} \end{cases} \\
A_{\mathrm{li}}(\theta) &= 4R^2|\cos\theta| \begin{cases} \arcsin u + u\sqrt{1-u^2} & \text{if } u \le 1, \\ \frac{\pi}{2} & \text{otherwise,} \end{cases} \\
A_{\mathrm{la}}(\theta) &= 2R^2|\cos\theta| \begin{cases} 2u - \arcsin u - u\sqrt{1-u^2} & \text{if } u \le 1, \\ 2u - \frac{\pi}{2} & \text{otherwise,} \end{cases}
\end{aligned}$$

are the corresponding partial areas.

The lid–base term is a simple Dirac delta

$$p_{\mathrm{lb}}(L|\theta) = \delta(L - L_{\max}),$$

where $L_{\max}(\theta)$ is the *maximum tracklength* (see Fig. 4.4)

$$L_{\max}(\theta) = \begin{cases} \frac{h}{|\cos\theta|} & \text{if } u \le 1, \\ \frac{2R}{\sin\theta} & \text{otherwise.} \end{cases}$$

The lid–lateral and lateral–lateral terms are

$$p_{\mathrm{li}}(L|\theta) = \frac{2R^2}{A_{\mathrm{li}}(\theta)} \begin{cases} |\sin 2\theta| \sqrt{1 - \left(\frac{L}{2R}\sin\theta\right)^2} & \text{if } 0 \le L \le L_{\max}(\theta), \\ 0 & \text{otherwise,} \end{cases}$$

$$p_{\mathrm{la}}(L|\theta) = \frac{2R^2}{A_{\mathrm{la}}(\theta)} \begin{cases} \dfrac{L\sin^3\theta(h - L|\cos\theta|)}{\sqrt{4R^2 - (L\sin\theta)^2}} & \text{if } 0 \le L \le L_{\max}(\theta), \\ 0 & \text{otherwise,} \end{cases}$$

respectively.

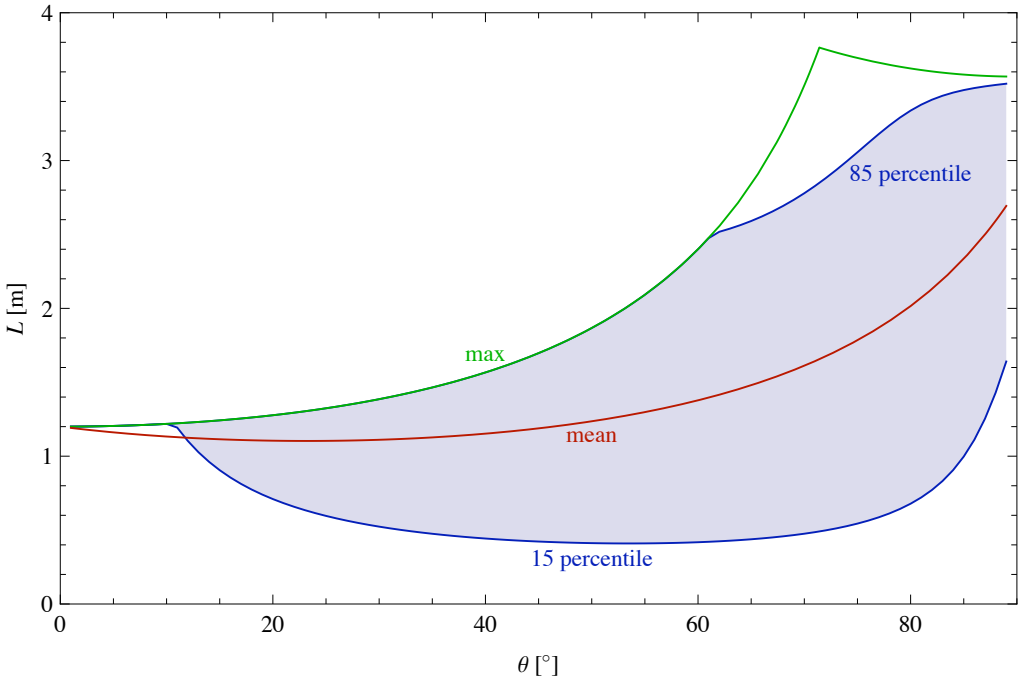


Figure 4.4: The mean tracklength $\bar{L}(\theta) = \mathbb{E}\{p(L|\theta)\}$ (red), the maximum tracklength $L_{\max}(\theta)$ (green), and the region that contains 70% of the probability mass (blue) as a function of the zenith angle θ.

4.2.2 The photon production rate distribution

The rate at which Cherenkov photons produced when a particle with mass m and kinetic energy T passes through a substance with refractive index n is

$$\phi_{\mathrm{Cherenkov}}(T, m, n) = \max\left(0, \frac{n^2 - \frac{(m+T)^2}{2mT+T^2}}{n^2 - 1}\right), \tag{4.4}$$

where both T and m are measured in the same unit, usually in MeV. As $T \to \infty$, $\phi_{\mathrm{Cherenkov}}$ tends to 1, and it becomes 0 at

$$T_{\min} = m\left(\frac{n}{\sqrt{n^2 - 1}} - 1\right).$$

Muons also lose energy as they pass through water at a rate of about 200 MeV/m, so for the *effective* Cherenkov rate, (4.4) must be integrated over the track of the muon. The yellow curve in

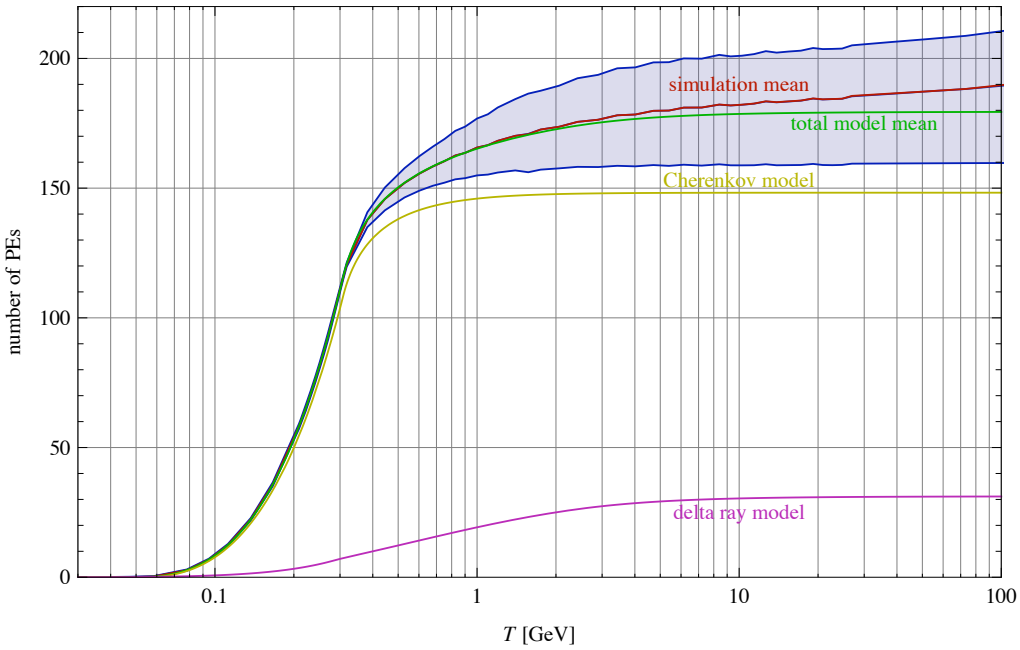


Figure 4.5: Photoelectron distribution $p(\#\mathrm{PE}|T_\mu)$ of vertical throughgoing muons as a function of the kinetic energy t_μ of a muon. The Cherenkov contribution and the mean of the delta ray fraction can be computed analytically, but the delta ray distribution and the contribution due to high-energy phenomena can only be assessed by simulations. The red curve is the mean of the conditional distribution and the blue region contains 70% of the conditional probability mass. The conversion between ϕ and the number of photoelectrons is simply done by reading out the mean at $T = 1\,\mathrm{GeV}$.

Figure 4.5 shows the result of this integration. The rate is measured in number of photoelectrons: the full rate of $\phi_{\mathrm{Cherenkov}}(T, m, n) = 1$ corresponds to about 149 photoelectrons.

Although the Cherenkov effect is the dominating light source when a high-energy charged particle passes through water, it is not the only one. Delta rays and other more exotic processes also produce light. Whereas the number of Cherenkov photons is practically constant given the muon energy, these other sources are intrinsically probabilistic, generating a non-Dirac probability distribution $p(\phi|T_\mu)$. These phenomena are also quite hard to model analytically. To obtain $p(\phi|T_\mu)$ numerically, we simulated a large number of muons going through the Auger tank at its vertical axis of symmetry, and counted the number of PEs detected in the simulated PMTs. After unfolding the Poisson fluctuation of the PEs, we obtained the distribution depicted in 4.5. The absolute number of PEs are related to the detection efficiency in the simulations; it is effectively a free parameter. To calibrate ϕ, we simply read out the mean number of PEs at $T = 1\,\mathrm{GeV}$ and make $\phi = 1$ correspond to about 165 PEs.

4.2.3 The generative model of the FADC signal

The goal of this section is to develop a model for the FADC signal

$$p(\mathbf{x}|t_\mu, L_\mu, \phi_\mu) \tag{4.5}$$

given the time of arrival t_μ of the muon, its tracklength L_μ, and its photon production rate ϕ_μ.[5] We will construct the model in a bottom-up fashion, that is, we start from the signal, and develop the model by adding explanatory parameters and nuisance parameters. The difference between the two kinds of parameters is that we will keep explanatory parameters in the final model (and possibly estimate them on data), while nuisance parameters only play the role of simplifying the likelihoods, and we will eliminate them by integrating (or summing) over them. Table 4.1 summarizes all the variables of the model and Figure 4.6 illustrates the steps of the generative process.

name	**notation**	**unit**	**law (priors or conditionals)**
bin width	t_Δ	ns	δ_{25}
signal decay time	τ	ns	δ_{60} or $\mathcal{N}_{60,5}$
signal risetime	t_d	ns	δ_{13} or $\mathcal{N}_{13,1}$
signal start time	t_0	s	given for each signal
muon arrival time	t_μ	ns	$\mathcal{U}$niform
muon tracklength	L_μ	m	see Section 4.2.1
muon photon production factor	ϕ_μ	unitless	δ_1 or $\mathcal{N}_{1,0.1}$ or see Section 4.2.2
avg number of PEs per 1 m tracklength	ν	m^{-1}	δ_{76} or $\mathcal{N}_{79,14}$ [140, 141, 142]
PE signal mean (gain)	μ	adc	$\delta_{1.8}$ or $\Gamma_{18,0.1}$
PE signal shape	k	unitless	δ_2 or $\Gamma_{20,0.1}$
PE signal noise std	σ	adc	$\delta_{0.55}$ or $\mathcal{N}_{0.55,0.05}$
PE signal baseline	b	adc	$\mathcal{N}_{55,5}$
expected number of PEs in bin i	$\bar{n}_i$	unitless	$\phi_\mu L_\mu \nu \int_{t_{i-1}}^{t_i} p_{\tau,t_d}(t - t_\mu)\mathrm{d}t$
number of PEs in bin i	n_i	unitless	$\mathrm{Poi}_{\bar{n}_i}$
noiseless signal in bin i	$\bar{x}_i$	adc	$\Gamma_{n_i k, \mu/k}$
signal in bin i	x_i	adc	$\mathcal{N}_{b+\bar{x},\sigma}$

Table 4.1: Explanatory (first 12 lines) and nuisance parameters. For explanatory parameters the fourth column is the prior distribution. For nuisance parameters we give the conditional distributions. δ_a is the Dirac delta distribution centered at a (i.e., $\delta_a(x) = \delta(x - a)$), $\mathcal{U}_{a,b}$ is the uniform distribution in the interval $[a, b]$, $\Gamma_{k,\theta}$ is the Gamma distribution, Poi_n is the Poisson distribution, and $\mathcal{N}_{\mu,\sigma}$ is the Normal distribution.

The signal given the expected photoelectron counts

First we will rewrite (4.5) as

$$p(\mathbf{x}|t_\mu, L_\mu, \phi_\mu) = \int_{\mathbb{R}^{+N}} p(\mathbf{x}, \bar{\mathbf{n}}|t_\mu, L_\mu, \phi_\mu)\mathrm{d}\bar{\mathbf{n}} = \int_{\mathbb{R}^{+N}} p(\mathbf{x}|\bar{\mathbf{n}}, t_\mu, L_\mu, \phi_\mu)p(\bar{\mathbf{n}}|t_\mu, L_\mu, \phi_\mu)\mathrm{d}\bar{\mathbf{n}}, \tag{4.6}$$

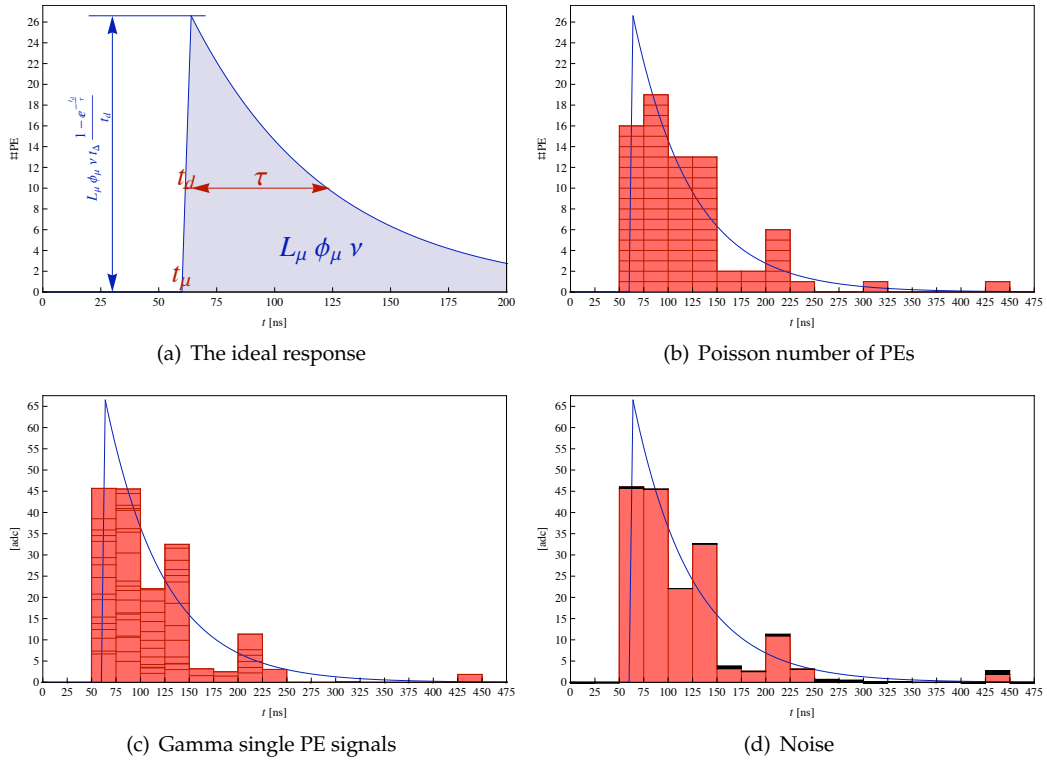
where $\bar{\mathbf{n}} = (\bar{n}_1, \ldots, \bar{n}_N) \in \mathbb{R}^{+N}$, and $\bar{n}_i$ is the expected number of PEs in the bin $[t_{i-1}, t_i)$. Since the signal is independent of t_μ, L_μ, and ϕ_μ *given* $\mathbf{n}$, we can simplify (4.6) to

$$p(\mathbf{x}|t_\mu, L_\mu, \phi_\mu) = \int_{\mathbb{R}^{+N}} p(\mathbf{x}|\bar{\mathbf{n}})p(\bar{\mathbf{n}}|t_\mu, L_\mu, \phi_\mu)\mathrm{d}\bar{\mathbf{n}}, \tag{4.7}$$

Since the second term is a Dirac delta, ($\bar{\mathbf{n}}$ is a deterministic function of t_μ, L_μ, ϕ_μ, and some other explanatory parameters), the convolution further simplifies to a simple product

$$p(\mathbf{x}|t_\mu, L_\mu, \phi_\mu) = p(\mathbf{x}|\bar{\mathbf{n}})\ \delta_{\bar{\mathbf{n}}(t_\mu, L_\mu, \phi_\mu)}. \tag{4.8}$$

[5] There are some important effects that we do not model, mainly azimuthal dependencies due to the asymmetries of the photodetectors, and the effect of direct light in the photodetectors that can affect significantly the signal in the first FADC bin.



(a) The ideal response

(b) Poisson number of PEs

(c) Gamma single PE signals

(d) Noise

Figure 4.6: Generating the FADC signal.

The function $\bar{\mathbf{n}}(t_\mu, L_\mu, \phi_\mu)$ in the second term is the binwise integral of the *ideal muon response* (Figure 4.6(a)). It depends mainly on the time response profile which we will describe in Section 4.2.4. Here we start by developing the first term $p(\mathbf{x}|\bar{\mathbf{n}})$ of (4.8). First note that given the expected number of PEs, the bin-wise PE signals x_i are independent of each other. This is a quite realistic assumption since the signals vary due to some random events in the PMT, although some dependence can occur due to some secondary effects.[6] In this model we ignore these effects, and factorize $p(\mathbf{x}|\bar{\mathbf{n}})$ to obtain

$$p(\mathbf{x}|\bar{\mathbf{n}}) = \prod_{i=1}^{N} p(x_i|\bar{n}_i). \tag{4.9}$$

We now introduce a second nuisance parameter, the actual number of PEs n_i in the bin $[t_{i-1}, t_i)$, and rewrite $p(x_i|\bar{n}_i)$ as

$$p(x_i|\bar{n}_i) = \sum_{n_i=0}^{\infty} p(x_i, n_i|\bar{n}_i) = \sum_{n_i=0}^{\infty} p(x_i|n_i, \bar{n}_i) p(n_i|\bar{n}_i). \tag{4.10}$$

Given n_i, the signal x_i is independent of the expected number of PEs $\bar{n}_i$, so $p(x_i|n_i, \bar{n}_i) = p(x_i|n_i)$. Since we can assume that the PEs are generated by capturing a small portion of Cherenkov photons, the number of PEs n_i is Poisson with parameter $\bar{n}_i$ (Figure 4.6(b)), and so (4.10) simplifies to

$$p(x_i|\bar{n}_i) = \sum_{n_i=0}^{\infty} p(x_i|n_i) \mathrm{Poi}_{\bar{n}_i}(n_i). \tag{4.11}$$

The first term $p(x_i|n_i)$ depends on the spectrum of single PEs, the baseline, and the bin-wise noise. To model this convolution, we introduce our last nuisance parameter, the noiseless signal

[6] See Figure A-52 in [142].

$\bar{x}_i$ in the bin $[t_{i-1}, t_i)$, and rewrite $p(x_i|n_i)$ as

$$p(x_i|n_i) = \int_0^\infty p(x_i, \bar{x}_i|n_i)\mathrm{d}\bar{x}_i = \int_0^\infty p(x_i|\bar{x}_i, n_i)p(\bar{x}_i|n_i)\mathrm{d}\bar{x}_i. \tag{4.12}$$

Given $\bar{x}_i$, the signal x_i is independent of the PE count n_i (it only depends on the noise and the baseline), so we can rewrite (4.12) as

$$p(x_i|n_i) = \int_0^\infty p(x_i|\bar{x}_i)p(\bar{x}_i|n_i)\mathrm{d}\bar{x}_i. \tag{4.13}$$

The bin-wise noiseless signal $\bar{x}_i$ (Figure 4.6(c)) is shifted by a baseline b, and also corrupted by an additive Gaussian noise (Figure 4.6(d)) with zero mean and standard deviation σ so the first term of (4.13) becomes

$$p(x_i|\bar{x}_i) = \mathcal{N}_{b+\bar{x}_i,\sigma}(x_i). \tag{4.14}$$

The noiseless signal $\bar{x}_i$ is a sum of n_i independent PE signals. It was measured [143, 144, 142] that single PEs deposit a signal which is almost exponential, having a relative variance

$$\frac{\text{variance}}{\text{mean}^2} \approx 0.5. \tag{4.15}$$

The actual form of the distribution is not very important since the total signal is a sum of several PE signals, so we have decided to model the single PE spectrum by a Gamma distribution with shape parameter $k = \text{mean}^2/\text{variance}$ and scale parameter $\theta = \text{variance}/\text{mean}$. We parametrize the distribution using the gain $\mu = k\theta$ and the shape parameter k. Since the distribution of the sum of n independent Gamma variates with parameters k and θ is $\Gamma_{nk,\theta}$, the distribution of the noiseless signal $\bar{x}_i$ given the number of PEs n_i is[7]

$$p(\bar{x}_i|n_i) = \Gamma_{n_i k,\mu/k}(\bar{x}_i). \tag{4.16}$$

Combining equations (4.11), (4.13), (4.14), and (4.16), our final bin-wise signal model is

$$\begin{aligned} p(x_i|\bar{n}_i) &= \sum_{n_i=0}^{\infty} \mathrm{Poi}_{\bar{n}_i}(n_i) \int_0^\infty \Gamma_{n_i k,\mu/k}(\bar{x}_i)\mathcal{N}_{b+\bar{x}_i,\sigma}(x_i)\mathrm{d}\bar{x}_i \\ &= \int_0^\infty \mathcal{N}_{b,\sigma}(x_i - \bar{x}_i) \underbrace{\sum_{n_i=0}^{\infty} \mathrm{Poi}_{\bar{n}_i}(n_i)\Gamma_{n_i k,\mu/k}(\bar{x}_i)}_{\text{compound Poisson}} \mathrm{d}\bar{x}_i \end{aligned} \tag{4.17}$$

Figure 4.7 shows the single PE spectrum under the model, using different values for the gain μ and the shape parameter k. The spectra contains the additive noise, that is, we numerically integrate the convolution of (4.17) with $n_i = 1$. The curves are reasonably close to those depicted by Figure 2 in [144], Figures 1-2 in [143], or Figure A-53 in [142].

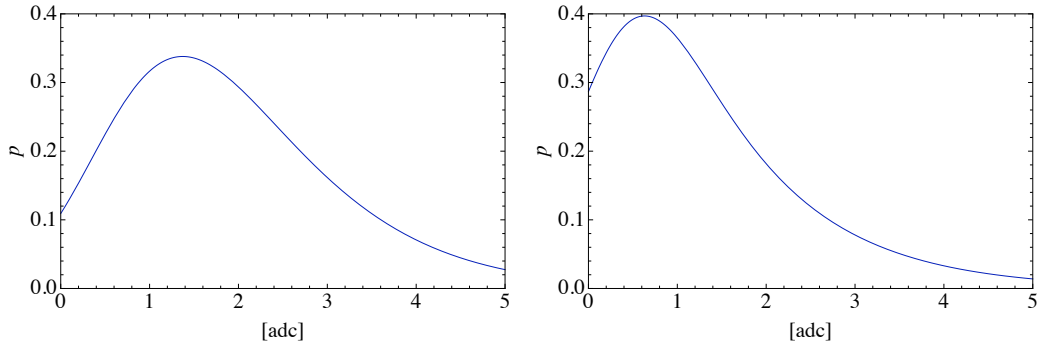
The distribution $p(x_i|\bar{n}_i)$ is the convolution of a Gaussian and a compound Poissonian (a sum of independent Gamma variables) so its mean is $b + \bar{n}_i\mu$ and its variance is

$$\bar{n}_i\mu^2\left(1 + \frac{1}{k} + \frac{\sigma^2}{\bar{n}_i\mu^2}\right).$$

Setting aside the baseline translation, the relative variance is

$$\frac{1}{\bar{n}_i}\left(1 + \frac{1}{k} + \frac{\sigma^2}{\bar{n}_i\mu^2}\right), \tag{4.18}$$

[7] For avoiding the singularity, we will define $\Gamma_{0,\theta}(x) = \delta_0(x)$ where δ_a is the Dirac delta distribution centered at a.



(a) $k = 2.5$, $\mu = 1.9$, $\sigma = 0.6$ (b) $k = 1.05$, $\mu = 1.2$, $\sigma = 0.6$

Figure 4.7: "Ideal" single PE spectra with additive noise.

where the third term in the parentheses can safely be ignored for large $\bar{n}_i$. The range of k goes from 1 (exponential PE spectrum) to ∞ (Dirac PE spectrum), with the relative signal variance (4.18) being between $1/\bar{n}_i$ (Dirac) and $2/\bar{n}_i$ (exponential).

Generating random variates from (4.17) is easy: one just has to go through the convolution chain. On the other hand, evaluating (4.17) given x_i and $\bar{n}_i$ (which we need in parameter estimation) can only be done numerically. One can approximate (4.17) by simpler formulas, but we found that these approximations may corrupt the estimation of signal parameters μ, k, and σ. In case these parameters are known, we can approximate the convolution (4.17) by a Gamma distribution with the same mean and variance:

$$p(x_i|\bar{n}_i) \approx \Gamma_{k',\theta'}(x_i - b) \tag{4.19}$$

with

$$k' = \bar{n}_i \frac{k}{(k+1)\left(1 + \frac{k\sigma^2}{(k+1)\bar{n}_i\mu^2}\right)} \tag{4.20}$$

$$\theta' = \mu\frac{k+1}{k} + \frac{\sigma^2}{\bar{n}_i\mu} \tag{4.21}$$

If the signal parameters are estimated with a Monte Carlo Markov chain (MCMC) algorithm, we can explicitly introduce the nuisance parameters $\bar{n}_i$, n_i, and $\bar{x}_i$, and let the MCMC do the numerical integration.

Discretization

The FADC count is a discretized value of the original signal. The likelihood (4.17) is a good approximation of the real likelihood in case $\sigma > 0.5$ times the resolution (which is 1 in our case). In this case the discretization variance $1/12$ is included in the estimated σ. In experiments we found that σ is around the critical value of 0.5, and we started to observe both a slight bias in the baseline estimate and a fluctuation of the noise estimate, depending where the real baseline was with respect to the FADC bin boundaries, so we opted to include the discretization in the likelihood by using

$$p(x_i|\bar{n}_i) = \sum_{n_i=0}^{\infty} \mathrm{Poi}_{\bar{n}_i}(n_i) \int_0^{\infty} \Gamma_{n_i k,\mu/k}(\bar{x}_i) \int_{x_i-0.5}^{x_i+0.5} \mathcal{N}_{b+\bar{x}_i,\sigma}(x_i')\mathrm{d}x_i'\mathrm{d}\bar{x}_i. \tag{4.22}$$

instead of (4.17).

4.2.4 The distribution of the expected photoelectron count in time

The second term $\bar{n}_i(L_\mu, \phi_\mu, t_\mu)$ of (4.8) determines the expected number of PEs in the bin $[t_{i-1}, t_i)$, given L_μ, ϕ_μ, and t_μ. We will use a simple model with three additional explanatory parameters, the risetime t_d, the rate of the exponential decay τ (both measured in ns), and the mean number ν of PEs generated by an average vertical atmospheric muon of unit (1 m) tracklength and captured in a PMT. We found that this parametrization works fine for vertical centered muons, which is the subject of this note. The model will have to be refined for inclined non-centered muons due to azimuthal asymmetries of the PMTs and direct light.

Our model is based on the assumption that a muon generates a number of Cherenkov photons along its trajectory at a rate that depends on its energy. The photons are generated at a precise angle around the trajectory. They can be reflected several times on the walls of the tank before arriving into the PMT. It was measured that the photon distribution "mixes" (becomes uniform) after around two reflections within another 5 to 10 ns. This means that the risetime is in the first two bins (at most). After that the rate of arrival in the PMT becomes exponential because of the constant decay rate due to the absorption of photons in the water and reflection losses. The rate can change from one tank to another, and it was observed to change also in time. To model the risetime, we assume that the photon generation is uniform in a window of width t_d. The decay phase is modeled by an exponential with parameter τ. The convolution can be solved analytically to obtain the time response distribution

$$p_{\tau,t_d}(t) = \frac{1}{t_d} \cdot \begin{cases} 0 & \text{if } t < 0, \\ 1 - \exp(-t/\tau) & \text{if } 0 \le t < t_d, \\ \exp\left(-(t-t_d)/\tau\right) - \exp(-t/\tau) & \text{if } t_d \le t. \end{cases} \tag{4.23}$$

Figure 4.8 shows the time response distribution with typical parameter values $\tau = 60$ ns and $t_d = 13$ ns.

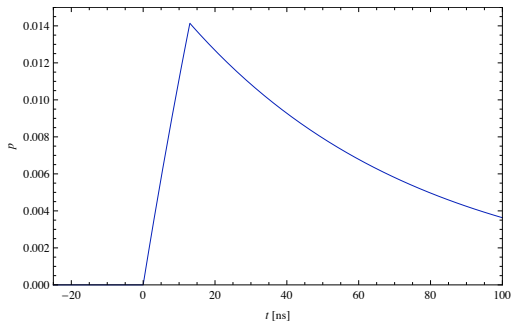


Figure 4.8: The time response distribution with typical values $\tau = 60$ ns and $t_d = 13$ ns.

To obtain the expected number of PEs in a bin, we first have to integrate $p_{\tau,t_d}(t)$ in the bin, and then we have to multiply it with (a) the tracklength L_μ, (b) the average number of PEs per unit tracklength ν, and (c) the energy factor ϕ_μ,

$$\bar{n}_i(L_\mu, \phi_\mu, t_\mu) = \phi_\mu L_\mu \nu \int_{t_{i-1}}^{t_i} p_{\tau,t_d}(t - t_\mu)\mathrm{d}t. \tag{4.24}$$

4.3 An adaptive Metropolis algorithm for mixture models

Adaptive Metropolis (AM) [135] is a powerful recent algorithmic tool in numerical Bayesian data analysis. AM builds on a well-known Markov Chain Monte Carlo (MCMC) algorithm but optimizes the rate of convergence to the target distribution by automatically tuning the design parameters of the algorithm on the fly. In the case of AM, the adaptive parameter tuning rule relies

on the availability of an online estimate of the covariance of the target distribution. AM is considered a pioneering contribution in *adaptive MCMC* methodology, and it represents a significant step towards developing self-tuning, turn-key sampling algorithms which are necessary for using MCMC approaches in high throughput data analysis. The optimality criterion considered in AM relies on theoretical results derived when the target distribution is multivariate Gaussian [145, 146]. When the target is multi-modal or concentrated around a non-linear manifold, the algorithm still applies but its optimality is no longer guaranteed, and its practical performance is often suboptimal.

An important case where AM usually fails is when the target is the posterior distribution in Bayesian inference on a mixture model. In this case the mixture likelihood is invariant to permuting the mixture components, and the chosen prior often does not favor any permutation either. Although we usually have "nice" genuinely unimodal posteriors of the parameters of *well-identified* components, the posterior is highly multimodal with exponentially many peaks, one for each permutation of the components. Running a well-mixing MCMC on such a model results in useless marginal estimates for the parameters of the individual components due to the confusion between all possible labelings. This phenomenon is called *label switching*. Several approaches have been proposed to deal with this problem, usually in a *post-processing* step *after* the posterior sample has been produced [147, 148, 149, 150, 151, 152, 153]. They all aim to solve the identifiability problem in order to produce meaningful marginal posteriors for the mixture parameters. Running vanilla AM on such a model has two pitfalls. If the chain does not mix well, we get stuck in one of the modes. The adaptive proposal can then be efficient but we know that we are not sampling from the actual mixture posterior and it is quite likely that the results of inference will be ultimately biased. On the other hand, if the MCMC chain does switch between components, the online sample covariance estimate will be too broad, resulting in poor adaptive proposals and slow convergence. Note that the latter situation is usually prevalent when using trans-dimensional samplers based on the Reversible Jump approach [147]: in this case, the dimension-changing moves force label switching and running an embedded AM algorithm is, in our experience, almost useless.

To solve these challenging issues, we develop an Adaptive Metropolis algorithm with Online Relabeling (AMOR). The main idea is to combine the online relabeling strategy of [148] with the AM algorithm of [135]. This results in a doubly adaptive algorithm that uses the sample statistics both for *(i)* adapting its proposal and for *(ii)* redefining the region onto which it constrains the chain by attempting relabeling in each iteration.

The section is organized as follows. In Section 4.3.1 we formalize the problem and motivate AMOR on a real-world example. In Section 4.3.2, we derive the new online relabeling procedure based on AM and present our main convergence results.

4.3.1 Adaptive Metropolis and label switching

In this section we briefly provide some more background regarding AM and the label switching problem. We consider using MCMC sampling to explore posterior distribution

$$\pi(x) \triangleq p(x|\mathbf{data}) \propto p(\mathbf{data}|x)p(x)$$

of the parameters $x \in \mathcal{X}$ given the observation **data**. The posterior $\pi(x)$, whose normalization constant is usually unknown, can be explored by MCMC by running a Markov chain (X_t) with stationary distribution π. In this context, π is also said to be the *target distribution* of the MCMC chain. The Symmetric Random Walk Metropolis algorithm (SRWM; [154]; corresponding to the blue steps in Figure 4.10 in Section 4.3.2) is one of the most popular techniques for simulating such a chain (X_t). In SRWM the user has to provide a symmetric proposal kernel that will be used to propose a new sample $\tilde{X}$ given the previous sample X_{t-1}. When the posterior is a distribution over a continuous space $\mathcal{X} = \mathbb{R}^d$, the most common proposal kernel is a multivariate Gaussian $\mathcal{N}(\,\cdot\,|X_{t-1}, \Sigma)$.

The goal of *Adaptive Metropolis* (AM) (corresponding to the blue and green steps in Figure 4.10) is to automatically calibrate the design parameter Σ of SRWM. When the target $\pi(x)$ is multivariate Gaussian with covariance Σ_π, the optimal choice of Σ is of the order of $(2.38)^2\Sigma_\pi/d$ [145, 146]. In practice, Σ_π is unknown thus motivating the use of an estimate of the covariance of the posterior based on samples $(X_1, \ldots, X_{t-1})$ generated so far. From a theoretical point of view, the conditional distribution of X_t given the past then depends on the whole past rendering the analysis of AM more challenging. The convergence of AM has been recently addressed under quite general conditions (see, e.g., [155, 156] and references therein).

The optimal choice for Σ is appropriate only when the target distribution is strongly unimodal [146]. In a mixture model, the likelihood is of the form

$$p(\mathbf{data}|x) = \sum_{m=1}^{M} \alpha_m f(\mathbf{data}|\phi^{(m)}),$$

where $\sum_i \alpha_i = 1$ and $\alpha_i \geq 0$, $\phi^{(m)}$ denotes the n-dimensional parameter vector of the mth component, and the parameter space is a subset of $(\mathbb{R}^+ \times \mathbb{R}^n)^M$. The likelihood $p(\mathbf{data}|x)$ in this case is invariant under any permutation of the mixture components. If the prior $p(x)$ is *exchangeable*, that is, it also does not favor a particular permutation, then the posterior $\pi(x)$ inherits the permutation invariance.

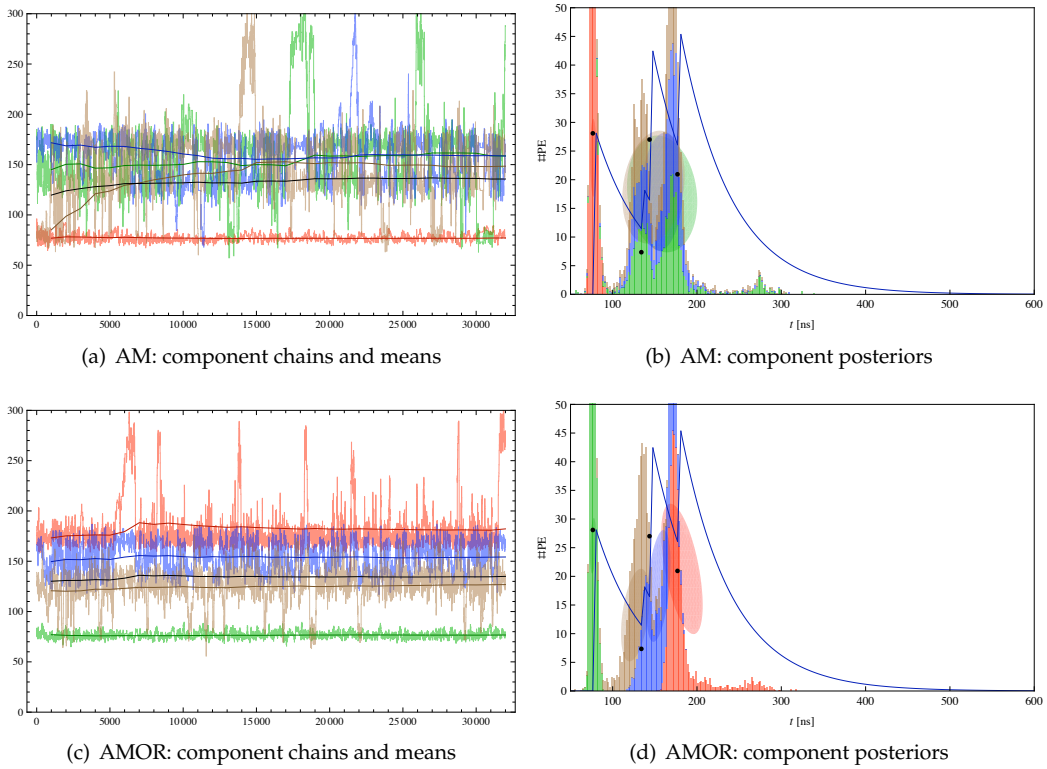
To illustrate the challenges of inference in this model, Figure 4.9(a)-4.9(b) display the MCMC sample corresponding to a single run of AM on a mixture of muon signals simulated from the model described in Section 4.2. The red variable gets stuck in one of the mixture components, whereas the blue, green, and brown variables visit all the three remaining components. Marginal estimates computed for the blue, green, and brown variables are then mostly identical as seen in Figure 4.9(b). In addition, the resulting empirical covariance estimate is very broad, resulting in poor efficiency of the adaptive algorithm.

Several approaches have been proposed to deal with the label switching problem. The first solution consists in modifying the prior in order to make it select a single permutation of the variables, introducing an *identifiability constraint* [147]. This solution is known to cause artificial biases by not respecting the topology of the posterior [150]. An effort has then been made to adapt to the estimated posterior surface through the design of relabeling algorithms [149, 150] that process the MCMC sample *after* the completion of the simulation run. These techniques look for a permutation P_t of each individual sample point X_t so as to minimize a posterior-based criterion depending on the *whole* chain history. [148] proposed an *online* version of the relabeling procedure in which the simulation of each X_t is followed by a permutation P_t of its components. The permutation P_t is chosen to minimize a user-defined criterion that depends only on the *past* history of the chain up to time t. The major advantage of this online approach is that it is compatible with our objective of solving label switching "on the fly" in order to optimize AM for permutation-invariant models.

In both batch and online relabeling algorithms, inference is carried out by using *relabeled* samples. Since the permutation steps in the MCMC procedure modify the distribution of the chain (X_t), the target distribution is no longer the posterior $\pi(x)$. [151] showed recently that, empirically, relabeling induces the learning of an appropriate identifiability constraint, but the existence of a target distribution and its relation with the original target $\pi(x)$ has been an open problem, meaning that these relabeling techniques have remained mostly heuristics.

4.3.2 AMOR: An adaptive online relabeling algorithm

From now on, we consider the general case of MCMC sampling from a target distribution with density $\pi(x)$ on $\mathcal{X} \subseteq \mathbb{R}^d$ with respect to (w.r.t.) the Lebesgue measure, with $d = qM$. Let $\mathcal{P}$ be a finite group of $d \times d$ block permutation matrices, indexed by some permutations ν of $\{1, ..., M\}$, acting on $\mathcal{X}$ as follows: for $x = (x_1, ..., x_M) \in \mathbb{R}^d$, $P_\nu x \triangleq (x_{\nu^{-1}(1)}, ..., x_{\nu^{-1}(M)})$. As an example, let us take $d = 6$ dimensions separated in three blocks of size 2. This would correspond to having



(a) AM: component chains and means

(b) AM: component posteriors

(c) AMOR: component chains and means

(d) AMOR: component posteriors

Figure 4.9: The results of AM (top raw) and AMOR (bottom raw) algorithms on a mixture of muon signals (Section 4.2). The right panels show the parameters of the four components. The x-coordinates of the black dots are the location parameters (muon arrival times t_μ) and the y-coordinates are the mixture coefficients $\alpha^{(m)}$ (proportional to $L_\mu \phi_\mu$). The (unnormalized) mixture distribution (sum of the ideal muon signals) are in blue. The colored histograms are the marginal posteriors of the four muon arrival times. The left panels show the four chains of the location parameters t_μ (light colors), the running means (dark colors), and the mean of the running means (black curve). The AM algorithm shows heavy label switching among the three rightmost components whereas the AMOR algorithm separates the components nicely.

a mixture model with 3 components and 2 parameters each. The 6×6 matrix associated to the permutation ν that sends 1 onto 2, 2 onto 3 and 3 onto 1 is

$$P_\nu = \begin{pmatrix} 0 & 0 & I_2 \\ I_2 & 0 & 0 \\ 0 & I_2 & 0 \end{pmatrix},$$

and for $x = (1, 2, ..., 6)^T$, $P_\nu x = (5, 6, 1, 2, 3, 4)^T$.

Let us now assume that π is invariant under the action of $\mathcal{P}$, i.e. $\pi(Px) = \pi(x)$ for any $P \in \mathcal{P}$. Our goal is to isolate a single mode out of the many identically repeated symmetric modes of the posterior. Formally, we are interested in restricting the target π to a *fundamental domain* $\mathcal{D}$ of the action of $\mathcal{P}$, that is, finding a subset $\mathcal{D} \subset \mathcal{X}$ which is minimal for the inclusion and for which $\{Px : x \in \mathcal{D}, P \in \mathcal{P}\} = \mathcal{X}$. Minimality is to be understood here as "up to sets of Lebesgue measure zero". Following [148, 149], we will select $\mathcal{D}$ so that the sample looks as Gaussian as possible, since we want to select a single mode of the symmetric modes of the target π. For this purpose, the domain $\mathcal{D}$ will be defined through the minimization of a Mahalanobis-type criterion.

For a $d \times d$ invertible covariance matrix Σ, let

$$\mathcal{N}(x|\mu,\Sigma) = \frac{1}{(2\pi)^{d/2}} \frac{1}{\sqrt{\det\Sigma}} \exp\left(-\frac{1}{2} L_{(\mu,\Sigma)}(x)\right)$$

be the density of a Gaussian distribution on $\mathbb{R}^d$ with mean μ and covariance matrix Σ, where

$$L_{(\mu,\Sigma)}(x) \triangleq (x-\mu)^T \Sigma^{-1} (x-\mu).$$

AMOR$(\pi(x), x_0, T, \mu_0, \Sigma_0, c)$

1 $\quad \mathcal{S} \leftarrow \emptyset$
2 $\quad$ **for** $t \leftarrow 1$ **to** T
3 $\quad\quad \Sigma \leftarrow c\Sigma_{t-1}$ ▷ *scaled adaptive covariance*
4 $\quad\quad \tilde{x} \sim \mathcal{N}(\cdot|x_{t-1}, \Sigma)$ ▷ *proposal*
5 $\quad\quad \tilde{P} \sim \arg\min_{P\in\mathcal{P}} L_{(\mu_{t-1},\Sigma_{t-1})}(P\tilde{x})$ ▷ *pick an optimal permutation*
6 $\quad\quad \tilde{x} \leftarrow \tilde{P}\tilde{x}$ ▷ *permute*
7 $\quad\quad$ **if** $\frac{\pi(x)\sum_P \mathcal{N}(Px_{t-1}|x,\Sigma)}{\pi(x_{t-1})\sum_P \mathcal{N}(Px|x_{t-1},\Sigma)} > \mathcal{U}[0,1]$ **then**
8 $\quad\quad\quad x_t \leftarrow x$ ▷ *accept*
9 $\quad\quad$ **else**
10 $\quad\quad\quad x_t \leftarrow x_{t-1}$ ▷ *reject*
11 $\quad\quad \mathcal{S} \leftarrow \mathcal{S} \cup \{x_t\}$ ▷ *update posterior sample*
12 $\quad\quad \mu_t \leftarrow \mu_{t-1} + \frac{1}{t}(x_t - \mu_{t-1})$ ▷ *update running mean and covariance*
13 $\quad\quad \Sigma_t \leftarrow \Sigma_{t-1} + \frac{1}{t}\left((x_t-\mu_t)(x_t-\mu_t)^\intercal - \Sigma_{t-1}\right)$
14 $\quad$ **return** $\mathcal{S}$

Figure 4.10: The pseudocode of the AMOR algorithm. The steps of the classical SRWM algorithm are in blue, the adaptive MH algorithm adds the green steps, and the new online relabeling steps are in red. Remark the adaptation of both the proposal (line 4) and the selection mechanism through the dependence of $L_{(\mu,\Sigma)}$ on (μ,Σ). Note that for practical reasons, a small εI_d is often added to the covariance matrix in line 3, but [157] recently confirmed that core AM does not lead to degenerate covariances. Note also that line 5 is usually a simple assignment of the optimal permutation. In case of ties, we draw uniformly from the finite set of optimal permutations.

4.3.3 Derivation of the algorithm

Let $\mathcal{C}_d^+$ be the set of real $d \times d$ (symmetric) positive definite matrices, and let $\theta \in \mathbb{R}^d \times \mathcal{C}_d^+$. θ will later on be taken to be the concatenation of the mean and covariance of the chain (X_t), but for now, it is fixed to an arbitrary value. AMOR combines two actions: *(i)* sample a chain with target proportional to $\pi\mathbb{I}\{\mathcal{D}\}$ where $\mathcal{D}$ is a fundamental domain of the action of P and *(ii)* learn the domain $\mathcal{D}$ on the fly.

To show this, first assume that the adaptation is frozen, that is, consider AMOR when steps 12 and 13 in the pseudocode in Figure 4.10 are removed. In this case, we prove in [136] that our algorithm is a MCMC sampler with target distribution

$$\pi_\theta(x) = Z_\theta^{-1}\pi(x)\mathbb{I}\{x \in V_\theta\}, \quad \text{where } Z_\theta = \int_{V_\theta} \pi(x)dx, \tag{4.25}$$

and V_θ is defined by $V_\theta \triangleq \{x : L_\theta(x) = \min_{P\in\mathcal{P}} L_\theta(Px)\}$. In other words, $\pi_\theta(x)$ is the original target $\pi(x)$ restricted to the nearest-neighbor (Voronoï) cell V_θ defined by the distortion measure L_θ.

The question now is to find a convenient θ in such a way that MH is optimized: based on [146], we want to choose the covariance matrix Σ such that Σ is proportional to the covariance matrix of π_θ. This implies that $\theta = (\mu, \Sigma)$ solves the fixed point equations

$$\mu = \int x\, \pi_\theta(x)dx, \qquad \Sigma = \int (x-\mu)(x-\mu)^T \pi_\theta(x)dx. \tag{4.26}$$

To achieve this goal, the *MH-online relabeling* (steps 3 to 11 in Figure 4.10) is combined with the adaptive steps 12 and 13. Further analysis of the algorithm can be found in [136].

4.4 Counting muons using neural networks

Counting muons in the FADC traces is a notoriously difficult problem. In principle, the analytical single muon response model [133, 134] (Section 4.2) would allow for efficient inference in a *pure* muonic signal. Unfortunately, in showers with zenith angle $\theta < 60°$, the electromagnetic "contamination" is non-negligible, so the main difficulty is to decouple the muonic and electromagnetic components. Unlike muons, the more numerous electromagnetic particles generate smaller signals, so, when added up, they constitute a smooth background to the muonic peaks. When the number of muons is relatively small a simple approach to count muons in the "contaminated" signal is to compute the discrete derivatives $(x_{i+1} - x_i)$ in each bin and count the number of bins when $(x_{i+1} - x_i)$ exceeds a threshold δ [158, 159] (Figure 4.11. More formally, the goal is to construct an estimator for the number of muons N_μ based on the "jump" statistics

$$J_\delta = \sum_{\text{FADC bin } i} \underbrace{(x_{i+1} - x_i)}_{\text{jump}} \mathbb{I}\{x_{i+1} - x_i > \delta\}. \tag{4.27}$$

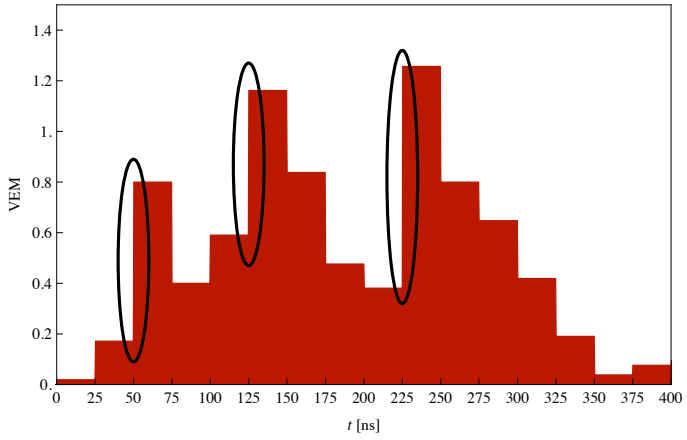


Figure 4.11: The FADC trace of three vertical throughgoing muons arriving at 45, 120, and 225 ns. A simple statistics that correlates well with the number of muons is the number of "jumps" (discrete derivatives) exceeding a threshold δ.

To assess the overall quality of the estimator J_δ, Figure 4.12 shows the histograms of the relative error[8]

$$\eta_\delta = \frac{N_\mu + 1}{J_\delta + 1} \tag{4.28}$$

[8]We offset the counts by 1 to avoid the singularity at $J_\delta = 0$.

for $\delta = 0.1$ and $\delta = 0.5$. The figures indicate that $J_{0.1}$ is a better estimator than $J_{0.5}$ both in terms of variance and regularity (shorter tails). The main problem with J_δ is, in fact, not overall bias or variance but its dependence on various shower parameters (mainly distance from the core r, energy of the shower E, or the zenith angle θ) that may generate energy- and zenith angle-trends in the bias (Figure 4.13).

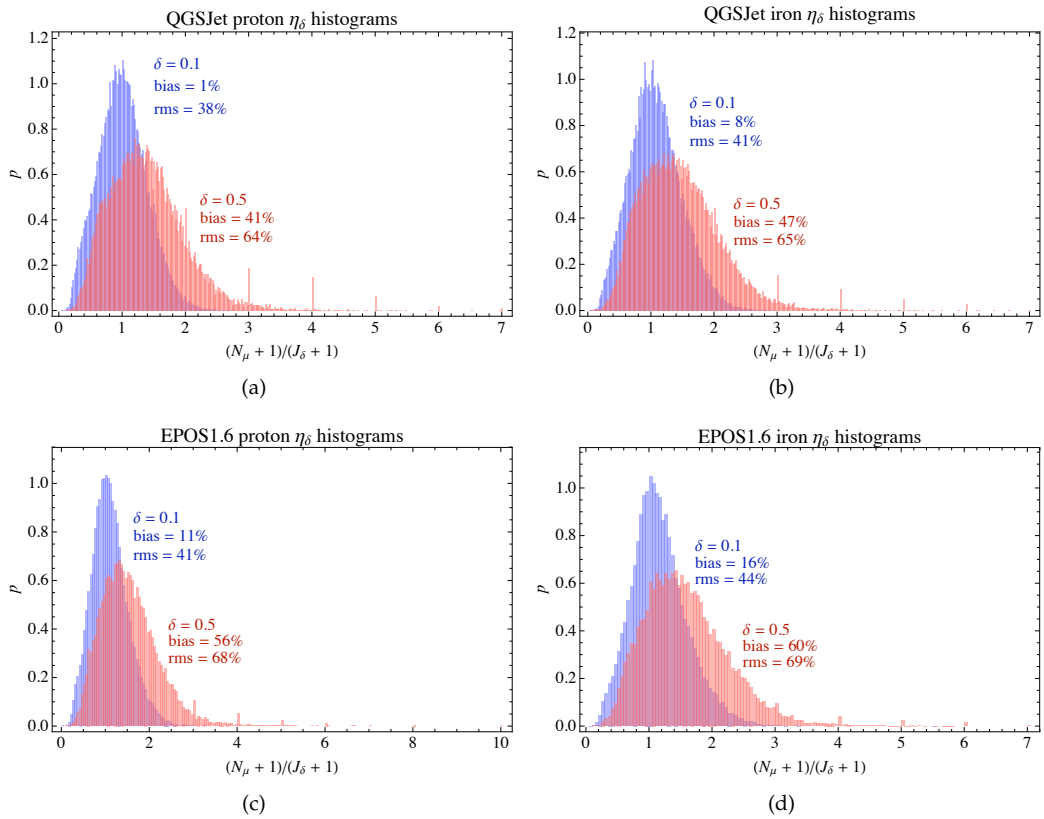


Figure 4.12: Regularized η_δ (4.28) histograms for $\delta = 0.1$ and $\delta = 0.5$.

The goal of this section is to estimate the correction factor $\eta_{0.1}$ based on a large set of simulated individual FADC signals. Although we only use proton and iron showers simulated by QGSJetII, the detectorwise muon estimator does not have access to any shower parameters, so it cannot bias its estimate explicitly. The procedure does not eliminate the bias completely, but the bias is smaller than the bias of the jump method and it is easier to control. Using independent simulations, we can obtain reliable upper bounds on the systematic bias.

4.4.1 The non-parametric multivariate correction factor

The FADC signal has several high-level characteristics that are known to be correlated to the number of muons crossing the detector. The integrated thresholded discrete derivative ("jump") (4.27) is only one of them. The main idea of this analysis is to extract as much information from the FADC signal as possible through variables that are plausibly correlated to muon content, and let a non-parametric predictor to construct a correction-factor estimator $\widehat{\eta}$ based on a large set of divers simulations.

The 172 extracted variables, described in detail in [72], include signal integrals, time quantiles, "jump"-style parameters, and asymmetry statistics between the three PMTs. Of course, most of the extracted variables are correlated even if the number of muons is given, so the first step of the analysis is to run Principal Component Analysis (PCA) to reduce the dimensionality of the data.

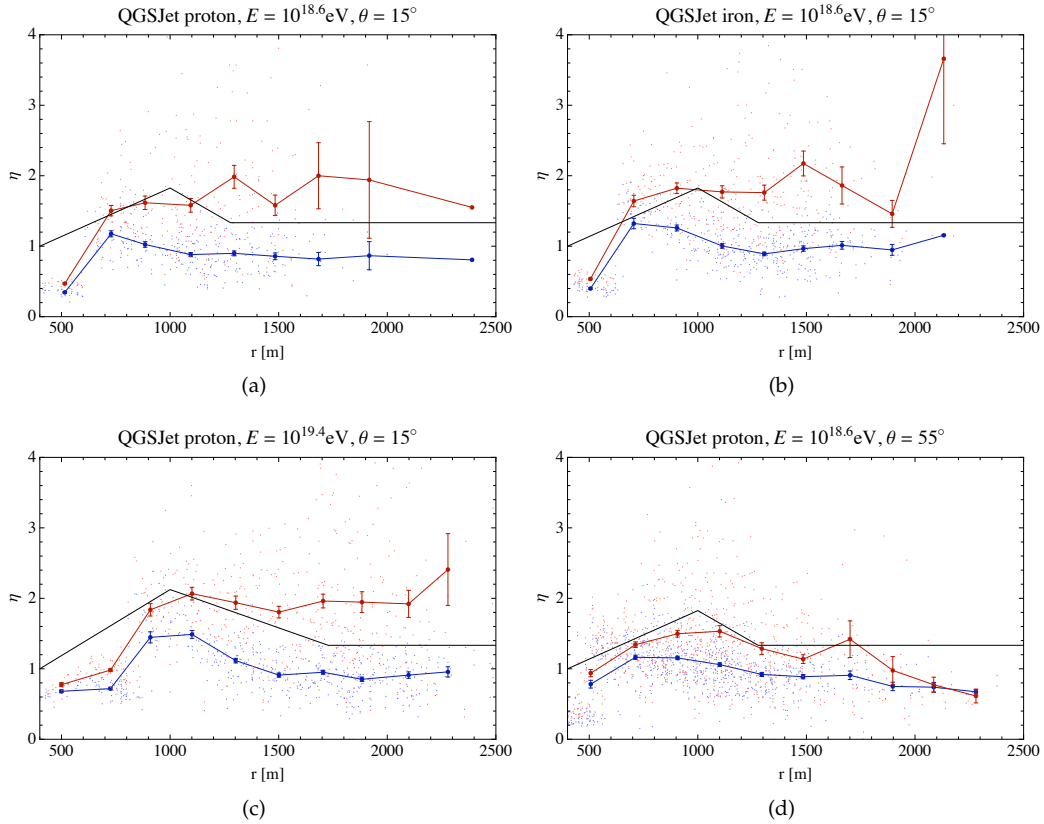


Figure 4.13: Trends of η. The red dots and curve depict the statistics $N_\mu / J_{0.5}$ used in the original jump method [158, 159]. The black curve is the "hand-made" correction factor of [159]. The blue dots and curve depict the statistics $\eta_{0.1} = (N_\mu + 1)/(J_{0.1} + 1)$ used in this section.

After this step, each FADC signal is represented by a real vector of length 19, capturing 98% of the total variance of the original 172-dimensional data set (Figure 4.14).

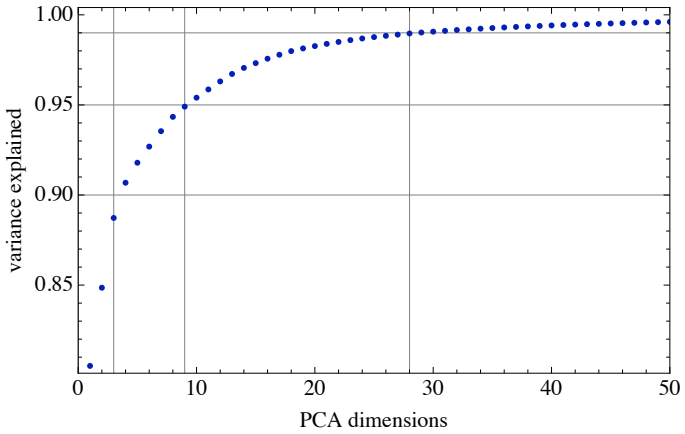


Figure 4.14: The proportion of the total variance explained by the first ℓ principal components. The gray lines represent the 0.9, 0.95 and 0.99 quantiles and the corresponding numbers of principal components.

The next step is the actual training of the nonparametric predictor. We use the NETLAB implementation [56] of the neural network technique described in [50] for this purpose. A useful property of this implementation is that, beside point estimates, it provides reliable uncertainty estimates on individual points which we can easily turn into uncertainties on the detectorwise muon number estimate. Since the global goal of the analysis is to obtain a single observable for each *shower*, these uncertainties greatly improve the lateral distribution function (LDF) fit and, in turn, our final estimator of muon content.

Neural networks are usually trained to minimize the squared error between the predictions and the target "labels". Since we are interested in minimizing the relative error of the muon estimate, we define the target labels as

$$y = \log \eta_{0.1} = \log \frac{N_\mu + 1}{J_{0.1} + 1}. \tag{4.29}$$

The output of the neural network will be interpreted as a Gaussian over the target, so using the logarithm in (4.29) will assure that negative N_μ's receive no probability (more precisely, the output distribution of $\widehat{N}_\mu$ will be log-normal). Details on the training and validation process can be found in [72].

The neural network provides point estimates $\widehat{y}$ and error bar estimates $\widehat{\sigma}_y$ for the target label $y = \log \eta_{0.1}$ of every FADC signal. Assuming that the posterior distribution of y is Gaussian $\mathcal{N}_{\widehat{y},\widehat{\sigma}_y}$, $\frac{N_\mu+1}{J_{0.1}+1}$ has a log-normal distribution. Considering that N_μ is an integer, we integrate the posterior of $\log \frac{N_\mu+1}{J_{0.1}+1}$ between $\log \frac{N+0.5}{J_{0.1}+1}$ and $\log \frac{N+1.5}{J_{0.1}+1}$ to obtain the probability (likelihood) of $N_\mu = N$. Formally, we first compute the unnormalized probabilities

$$\widetilde{P}(N_\mu = N) = \operatorname{erf}\left(\frac{\log \frac{N+1.5}{J_{0.1}+1} - \widehat{y}}{\sqrt{2}\widehat{\sigma}_y}\right) - \operatorname{erf}\left(\frac{\log \frac{N+0.5}{J_{0.1}+1} - \widehat{y}}{\sqrt{2}\widehat{\sigma}_y}\right),$$

then normalize them using the interval $[\widehat{y} - 5\widehat{\sigma}_y, \widehat{y} + 5\widehat{\sigma}_y]$, making also sure that only nonnegative counts receive any probability:

$$P(N_\mu = N) = \frac{\widetilde{P}(N_\mu = N)}{\sum_{N'=\max\left(0,(J_{0.1}+1)\exp(\widehat{y}-5\widehat{\sigma}_y)-1\right)}^{(J_{0.1}+1)\exp(\widehat{y}+5\widehat{\sigma}_y)-1} \widetilde{P}(N_\mu = N')}.$$

The output of this detectorwise reconstruction step is therefore a normalized table that assigns probabilities to a set of nonnegative integers in an interval also determined by the ANN. Since, through a complex process, this estimate depends only on the FADC signal $\mathbf{x}$, we will denote this table as

$$P_{\mathrm{ANN}}(N|\mathbf{x}) \triangleq P(N_\mu = N|\mathbf{x}).$$

This table will be used as an input likelihood in the reconstruction of the lateral distribution function described in Section 4.5.

4.4.2 Detectorwise results

The overall quality of the new estimator has improved significantly compared to the jump estimator (Figure 4.15). The range of the relative biases between QGSJet proton and EPOS1.6 iron is 17% for the ANN estimator and 22% for the jump estimator. The relative standard deviation of the ANN estimator is less then half of the standard deviation of the jump estimator.

The 15-20% systematic bias (across primary masses and hadronic models) of the output of the ANN estimator roughly corresponds to the relative error of estimating N_μ, confirmed by Figure 4.16. The overall trends depicted by Figure 4.16 are also what we expected. The main source of the trends is related to overestimating N_μ when N_μ is small and underestimating it when it is large. This trend can be almost completely eliminated using a simple correction that depends on the estimated number of muons.

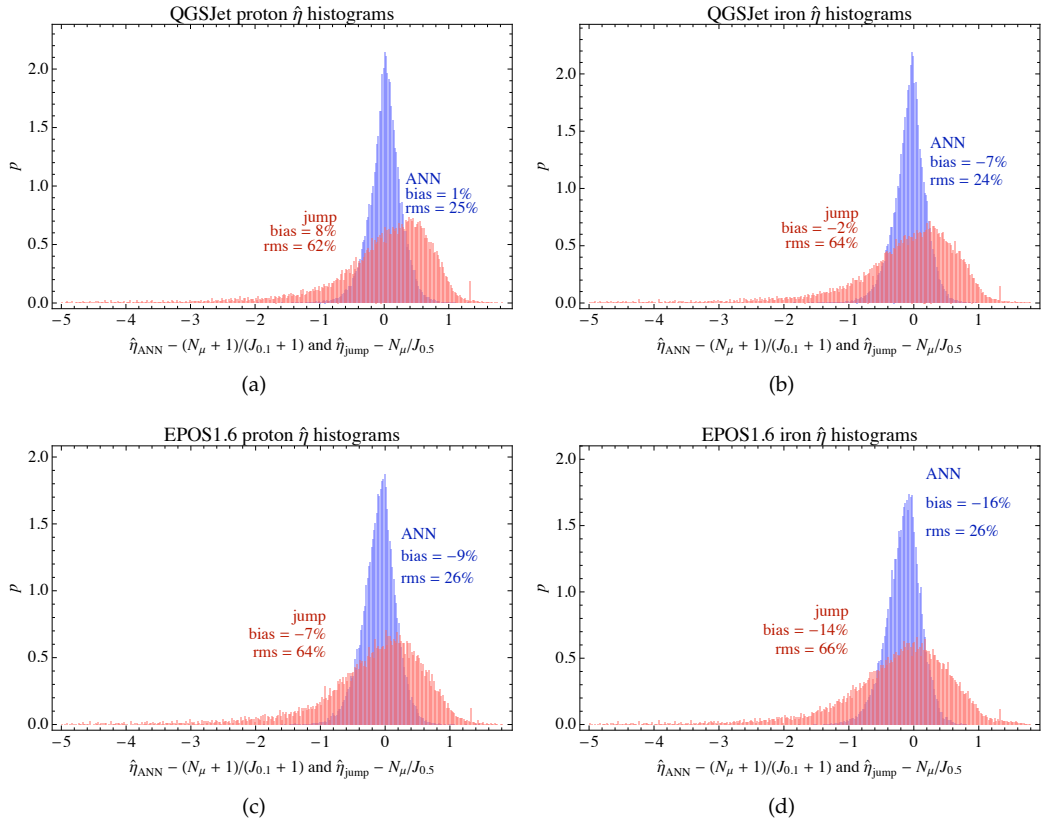


Figure 4.15: $\widehat{\eta}_{\mathrm{ANN}} - \eta_{0.1}$ and $\eta(E,r) - N_\mu/J_{0.5}$ histograms. These quantities roughly correspond to the relative error of estimating N_μ.

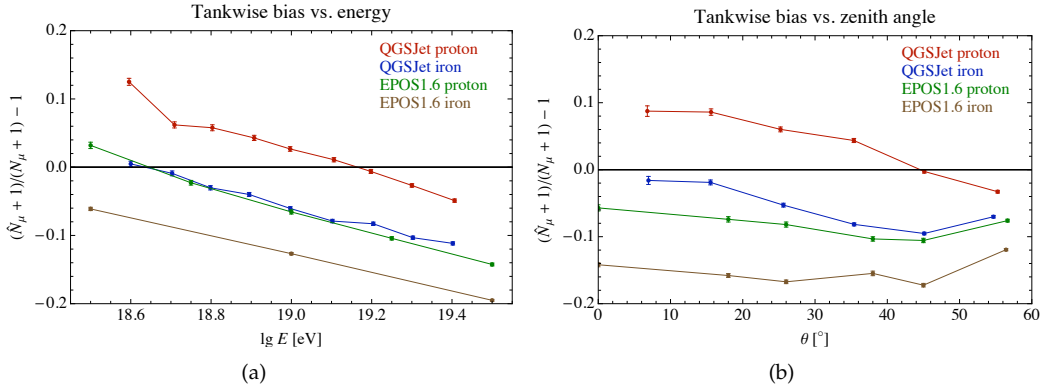


Figure 4.16: Detectorwise bias trends in energy and zenith angle.

4.5 An empirical Bayes approach for reconstructing the muonic LDF

In this section we describe a reconstruction method that fits a parametric lateral distribution function to the estimated number of muons in the detectors, and, at the same time, fits smooth functions to the dependencies of the LDF parameters on the energy and zenith angle of the shower. In particular, it automatically estimates the mean of the number of muons at 1000 m as a function

of the energy and the zenith angle.

The justification of the method is mathematically rather involved, but the global setup of the algorithm itself is quite intuitive. It consists in iterating two steps: 1) fixing global priors or constraints on the three LDF parameters and estimating the fit for each shower, and 2) fixing the individual fits and estimating the global constraints, that is, the dependencies of the LDF parameters on the energy and zenith angle of the shower. The constrained LDF fit is more flexible than a fixed-slope fixed-curvature fit, allowing for shower-to-shower fluctuation of the fit. At the same time, the constraints make sure that the fits of low-quality events do not vary too much, essentially "imitating" a fixed-slope fixed-curvature fit for these events. The other advantage of the method is that the constraints are non-parametric and estimated entirely form the data.

The section is organized as follows. In Section 4.5.1 we describe the basic setup: the LDF parametrizations, likelihoods, and the constraints. The description of the algorithm is divided into two parts. In Section 4.5.2 we give a short high-level overview of the empirical Bayes principle and its application to the concrete problem. Section 4.5.3 describes the algorithmic details of the approximate Expectation-Maximization (EM) algorithm we are using within the empirical Bayes framework. We illustrate the two steps of the algorithm on an example and show the expected bias and variance reduction in Section 4.5.4.

4.5.1 The basic setup

After introducing some new notations, we describe the LDF parametrizations, the shower likelihoods, and the constraints of the LDF parameters.

Notation

First, let us introduce some notation. We assume that we have n showers, indexed by i. For each shower, we observe its energy E_i and zenith angle θ_i. The ith shower has m_i active detectors, indexed by j. Out of these m_i detectors, k_i have triggered and $m_i - k_i$ are non-triggering. Without loss of generality we will assume that the first k_i have triggered and the last $m_i - k_i$ are non-triggering. The real number of muons in the jth detector of the ith shower will be denoted by $N_{i,j}$ (we omit the index $_\mu$ for simplicity). $N_{i,j}$ is, of course, only observed in simulations. When $N_{i,j}$ is not available, we observe the FADC signal $\mathbf{x}_{i,j}$, and use the probability table $P_{\mathrm{ANN}}(N|\mathbf{x}_{i,j})$ provided by the neural network to estimate (Section 4.4) the number of muons in the detector. For each detector, we also observe its distance from the shower core $r_{i,j}$ and an indicator $\iota_{i,j}$ whether it has triggered or not (that is, $\iota_{i,j} = 1$ if the detector triggers, 0 if it is non-triggering). When we talk about the whole event i, the vector of the numbers of muons in all the detectors of the event will be denoted by $\mathbf{N}_i$. Similarly, the vector of radii, the vector of FADC signals and the vector of triggers indicators will be $\mathbf{r}_i$, $\mathbf{X}_i$, and $\boldsymbol{\iota}_i$, respectively.

The lateral distribution function

After some preliminary experiments, we decided to use a log-log parabola for the LDF parametrization. The three parameters of the fit are the number of muons at 1000 m $N_\mu(1000)$, the slope of the fit β and the curvature parameter γ. To simplify the notation, we will use ν for $\log N_\mu(1000)$, so the LDF function is

$$\overline{N}(r, \nu, \beta, \gamma) = \exp\left(\nu + \beta \log \frac{r}{1000\,\mathrm{m}} + \gamma \log^2\left(\frac{r}{1000\,\mathrm{m}}\right)\right). \tag{4.30}$$

We use the notation $\overline{N}$ to emphasize that the function expresses the expected number of muons in a detector at a distance r from the shower core. Accordingly, the parameter $\exp(\nu) = N_\mu(1000)$ will target the expected number of muons in an imaginary detector at 1000 m from the core.

The shower likelihood

To simplify the notation, in some of the formulas we will omit the dependence of $\overline{N}$ on r, ν, β, and γ. Given the lateral distribution function $\overline{N}(r,\nu,\beta,\gamma)$, the likelihood of the number of muons is a simple Poisson

$$p(N|\overline{N}) = \mathrm{Poi}_{\overline{N}}(N).$$

We suppose that the probability that a detector triggers

$$p(\iota = 1|N) = f_\iota(N)$$

is a deterministic non-parametric function of the number of muons in the detector. This is definitely a simplification and it could be refined in a subsequent analysis. We do not parametrize f_ι, rather, we provide it as a table of probabilities. The trigger probability in a detector given the fit is then

$$\begin{aligned} p(\iota = 1|r,\nu,\beta,\gamma) &= p(\iota = 1|\overline{N}(r,\nu,\beta,\gamma)) \\ &= \sum_N p(\iota = 1|N)p(N|\overline{N}) \\ &= \sum_N f_\iota(N)\mathrm{Poi}_{\overline{N}}(N), \end{aligned}$$

where we use the simplifying assumption in the second equality. Similarly, the no-trigger probability is

$$\begin{aligned} p(\iota = 0|r,\nu,\beta,\gamma) &= p(\iota = 0|\overline{N}(r,\nu,\beta,\gamma)) \\ &= \sum_N p(\iota = 0|N)p(N|\overline{N}) \\ &= \sum_N (1 - f_\iota(N))\mathrm{Poi}_{\overline{N}}(N). \end{aligned} \tag{4.31}$$

Since we do not observe the number of muons in non-triggering detectors, (4.31) is itself the likelihood of a detector being non-triggering given the fit and the distance r of the detector from the shower core. Given that we observe the *real* number of muons N in the detector, the likelihood of the pair (ι, N) of a detector that has triggered is

$$\begin{aligned} p_{\mathrm{real}}(\iota = 1, N|r,\nu,\beta,\gamma) &= p(\iota = 1, N|\overline{N}(r,\nu,\beta,\gamma)) \\ &= p(\iota = 1|N,\overline{N})p(N|\overline{N}) \\ &= p(\iota = 1|N)p(N|\overline{N}) \\ &= f_\iota(N)\mathrm{Poi}_{\overline{N}}(N). \end{aligned}$$

In this case the likelihood of the triplet ν_i, β_i, γ_i is

$$\begin{aligned} p_{\mathrm{real}}(\mathbf{N}_i, \boldsymbol{\iota}_i, \mathbf{r}_i|\nu_i,\beta_i,\gamma_i) = &\prod_{j=1}^{k_i} f_\iota(N_{i,j})\mathrm{Poi}_{\overline{N}(r_{i,j},\nu_i,\beta_i,\gamma_i)}(N_{i,j}) \\ &\times \prod_{j=k_i+1}^{m_i} \sum_N (1 - f_\iota(N))\mathrm{Poi}_{\overline{N}(r_{i,j},\nu_i,\beta_i,\gamma_i)}(N). \end{aligned} \tag{4.32}$$

When using the probability table $P_{\mathrm{ANN}}(N|\mathbf{x})$ provided by the neural network estimator (Section 4.4), the likelihood of the pair of the trigger indicator ι and the FADC signal $\mathbf{x}$ is also a sum

$$p_{\mathrm{ANN}}(\iota = 1, \mathbf{x}|r,\nu,\beta,\gamma) = \sum_N P_{\mathrm{ANN}}(N|\mathbf{x})f_\iota(N)\mathrm{Poi}_{\overline{N}}(N),$$

so the likelihood of the triplet ν_i, β_i, γ_i is

$$\begin{aligned} p_{\text{ANN}}(\mathbf{X}_i, \boldsymbol{\iota}_i, \mathbf{r}_i | \nu_i, \beta_i, \gamma_i) = &\prod_{j=1}^{k_i} \sum_N P_{\text{ANN}}(N|\mathbf{x}_{i,j}) f_\iota(N) \text{Poi}_{\overline{N}(r_{i,j},\nu_i,\beta_i,\gamma_i)}(N) \\ &\times \prod_{j=k_i+1}^{m_i} \sum_N (1 - f_\iota(N)) \text{Poi}_{\overline{N}(r_{i,j},\nu_i,\beta_i,\gamma_i)}(N). \end{aligned} \tag{4.33}$$

The priors (constraints)

Maximizing the likelihoods (4.32) or (4.33) with completely free parameters is possible, but it can lead to degenerate fits with large uncertainties especially for lower energy events or events with bad geometry. As an alternative, one could fix β and/or γ and fit only the log number of muons ν at 1000 m, but this would eliminate the shower-to-shower fluctuation of β and γ and bias the fits unnecessarily for high quality events. In this analysis we opt for a best-of-both-worlds solution: we define "soft constraints" that can be formally interpreted as priors over the parameters ν, β, and γ.

In particular, the parameters ν, β, and γ will be modeled with independent Gaussians

$$\begin{aligned} p(\beta|\mu_\beta, \sigma_\beta) &= \mathcal{N}_{\mu_\beta,\sigma_\beta}(\beta) \\ p(\gamma|\mu_\gamma, \sigma_\gamma) &= \mathcal{N}_{\mu_\gamma,\sigma_\gamma}(\gamma) \\ p(\nu|\mu_\nu, \sigma_\nu) &= \mathcal{N}_{\mu_\nu,\sigma_\nu}(\nu). \end{aligned} \tag{4.34}$$

In the simplest setup, we could fix the mean and the variance of these Gaussians, but this would obviously bias the fits since we know that all parameters depend on the energy and the zenith angle of the shower. This dependence is very strong for ν, but β and γ can also have slight but clear trends in energy and zenith angle. The solution to this problem is to parametrize the means and the standard deviations in E and θ. In "physical terms", $\mu_\beta(E,\theta)$, $\mu_\gamma(E,\theta)$, and $\mu_\nu(E,\theta)$ are means of the LDF slope, curvature, and the logarithm of $N_\mu(1000)$, respectively, whereas $\sigma_\beta(E,\theta)$, $\sigma_\gamma(E,\theta)$, and $\sigma_\nu(E,\theta)$ represent the shower-to-shower fluctuations of the slope, curvature, and the logarithm of the number of muons at 1000 m, respectively. In this setup, the likelihood of the ith shower (conditioned on μ_β, μ_γ, μ_ν, σ_β, σ_γ, and σ_ν) is

$$\begin{aligned} p_{\text{real}}(\mathbf{N}_i, \boldsymbol{\iota}_i, \mathbf{r}_i, \nu_i, \beta_i, \gamma_i | \mu_\beta, \mu_\gamma, \mu_\nu, \sigma_\beta, \sigma_\gamma, \sigma_\nu) = \\ p_{\text{real}}(\mathbf{N}_i, \boldsymbol{\iota}_i, \mathbf{r}_i | \nu_i, \beta_i, \gamma_i) \times \mathcal{N}_{\mu_\beta(E_i,\theta_i),\sigma_\beta(E_i,\theta_i)}(\beta_i) \\ \times \mathcal{N}_{\mu_\gamma(E_i,\theta_i),\sigma_\gamma(E_i,\theta_i)}(\gamma_i) \times \mathcal{N}_{\mu_\nu(E_i,\theta_i),\sigma_\nu(E_i,\theta_i)}(\nu_i), \end{aligned} \tag{4.35}$$

when the number of muons is observed in each detector, and

$$\begin{aligned} p_{\text{ANN}}(\mathbf{X}_i, \boldsymbol{\iota}_i, \mathbf{r}_i, \nu_i, \beta_i, \gamma_i | \mu_\beta, \mu_\gamma, \mu_\nu, \sigma_\beta, \sigma_\gamma, \sigma_\nu) = \\ p_{\text{ANN}}(\mathbf{X}_i, \boldsymbol{\iota}_i, \mathbf{r}_i | \nu_i, \beta_i, \gamma_i) \times \mathcal{N}_{\mu_\beta(E_i,\theta_i),\sigma_\beta(E_i,\theta_i)}(\beta_i) \\ \times \mathcal{N}_{\mu_\gamma(E_i,\theta_i),\sigma_\gamma(E_i,\theta_i)}(\gamma_i) \times \mathcal{N}_{\mu_\nu(E_i,\theta_i),\sigma_\nu(E_i,\theta_i)}(\nu_i), \end{aligned} \tag{4.36}$$

when the number of muons is estimated using the neural network.

In our first attempt, the mean functions $\mu_\beta(E,\theta)$, $\mu_\gamma(E,\theta)$, and $\mu_\nu(E,\theta)$ were simple polynomial parametrizations of E and θ, but we found that the natural shape of some of these functions did not follow any simple polynomial, so this rigid setup either led to unnecessary biases or the number of parameters (degrees of the polynomials) had to be unreasonably high. To overcome this problem, we settled in a nonparametric solution in which overall trends were modeled with low-order polynomials, and the residuals were then fitted with smooth nonparametric function using Gaussian processes [160]. The standard deviation functions $\sigma_\beta(E,\theta)$, $\sigma_\gamma(E,\theta)$, and $\sigma_\nu(E,\theta)$ are linear functions of E and θ.

4.5.2 The empirical Bayes setup and an approximate EM algorithm

In a classical Bayesian analysis, the goal would be to obtain the posterior distribution

$$p(\boldsymbol{\phi}|\mathbf{data}) = \frac{p(\mathbf{data}|\boldsymbol{\phi})p(\boldsymbol{\phi})}{p(\mathbf{data})}, \tag{4.37}$$

and estimate the parameters by either the mean or the maximizer of $p(\boldsymbol{\phi}|\mathbf{data})$. In our case "**data**" represents either $\{(\mathbf{N}_i, \boldsymbol{\iota}_i, \mathbf{r}_i)\}_{i=1}^n$ or $\{(\mathbf{X}_i, \boldsymbol{\iota}_i, \mathbf{r}_i)\}_{i=1}^n$, and the parameter vector $\boldsymbol{\phi}$ is $\{(\nu_i, \beta_i, \gamma_i)\}_{i=1}^n$. Since the events are independent, the likelihood is just the product of (4.32) or (4.33) for all events. The prior $p(\boldsymbol{\phi})$ is usually a fixed probability distribution over the parameters that encodes prior knowledge ("constraints").

In our case, the prior $p(\boldsymbol{\phi})$ is further parametrized by

$$\boldsymbol{\xi} = (\mu_\beta, \mu_\gamma, \mu_\nu, \sigma_\beta, \sigma_\gamma, \sigma_\nu).$$

In the *Empirical Bayes* setup [139], we do not *fix* the prior, rather we *estimate* it by maximizing the *marginal likelihood*

$$p(\mathbf{data}|\boldsymbol{\xi}) = \int p(\mathbf{data}, \boldsymbol{\phi}|\boldsymbol{\xi})\mathrm{d}\boldsymbol{\phi} = \int p(\mathbf{data}|\boldsymbol{\phi})p(\boldsymbol{\phi}|\boldsymbol{\xi})\mathrm{d}\boldsymbol{\phi}, \tag{4.38}$$

where we used the fact that, given $\boldsymbol{\phi}$, the **data** is independent of the hyperparameters $\boldsymbol{\xi}$. It is a mixed frequentist/Bayesian approach that *integrates* over nuisance parameters $\boldsymbol{\phi}$ (which is a Bayesian step), but estimates the hyper-parameters $\boldsymbol{\xi}$ by maximum likelihood (which is frequentist). Once the *maximum marginal likelihood estimator* (MMLE)

$$\widehat{\boldsymbol{\xi}} = \arg\max_{\boldsymbol{\xi}} p(\mathbf{data}|\boldsymbol{\xi})$$

is found, the showerwise parameters $\boldsymbol{\phi} = \{\phi_i\}_{i=1}^n = \{(\nu_i, \beta_i, \gamma_i)\}_{i=1}^n$ can be estimated by using the distributions $p(\phi_i|\mathrm{data}_i, \widehat{\boldsymbol{\xi}})$ by maximization (MAP estimate) or by averaging (MEP estimate). Since

$$p(\phi_i|\mathrm{data}_i, \boldsymbol{\xi}) \propto p(\mathrm{data}_i|\phi_i)p(\phi_i|\boldsymbol{\xi}),$$

in case the MAP estimate is used, this step is equivalent to maximizing the likelihoods (4.35) or (4.36).

Directly maximizing (or even computing) (4.38) is infeasible in our case. To approximately maximize it, we iterate between two simple steps. We start with an initial prior $\boldsymbol{\xi}^{(0)}$, then iterate between

1. estimating the parameters $\phi_i = (\nu_i, \beta_i, \gamma_i)$ of each shower by fitting the LDF (4.30) to the detector signals under the constraints (4.34) defined by $\boldsymbol{\xi}^{(t)}$, and
2. estimating the energy- and zenith-angle-dependencies $\boldsymbol{\xi}^{(t+1)}$ by fitting a smooth function $\mu_\bullet(E, \theta)$ to the set of triplets $\{(E_i, \theta_i, \bullet_i)\}_{i=1}^n$ and fitting $\sigma_\bullet(E, \theta)$ to the residuals, where $\bullet$ is either β, γ, or ν.

This simple algorithm can have several variants with various interpretations, depending on how the fits are carried out in the two steps. For example, a purely frequentist solution consists in simple maximum likelihood fits in both steps, which leads to a variant of the Expectation Maximization (EM) algorithm [161] that converges to a (locally) maximum likelihood solution. We opted for a slightly more complicated solution because we wanted to compute individual uncertainty estimates for each shower parameter $\phi_i = (\nu_i, \beta_i, \gamma_i)$, and use these uncertainty estimates in the fits of Step 2) in a setup similar to weighted least squares. Our intuition was that, depending on the number of detectors m_i and their geometry, there can be significant differences in the parameter uncertainties between showers, and we wanted to give more weight to "nice" showers in determining the means and the shower-to-shower fluctuations of the three LDF parameters.[9]

[9] In fact, a heuristic version of this algorithm was already used in [158, 159]: we selected an unbiased set of "nice"

4.5.3 The approximate EM algorithm

Maximizing the marginal log likelihood

$$\log p(\mathbf{data}|\boldsymbol{\xi}) = \log \int p(\mathbf{data}, \boldsymbol{\phi}|\boldsymbol{\xi}) \mathrm{d}\boldsymbol{\phi} = \log \int p(\mathbf{data}|\boldsymbol{\phi}) p(\boldsymbol{\phi}|\boldsymbol{\xi}) \mathrm{d}\boldsymbol{\phi}$$

directly is infeasible even if we suppose that the underlying densities factorize and the factors $p(\mathrm{data}_i|\phi_i)$ and $p(\phi_i|\boldsymbol{\xi})$ have simple forms (e.g., Gaussians). To overcome this problem, we use the convexity of the log function and Jensen's inequality to define the lower bound

$$\begin{aligned} \log \int p(\mathbf{data}, \boldsymbol{\phi}|\boldsymbol{\xi}) \mathrm{d}\boldsymbol{\phi} &= \log \int q(\boldsymbol{\phi}) \frac{p(\mathbf{data}, \boldsymbol{\phi}|\boldsymbol{\xi})}{q(\boldsymbol{\phi})} \mathrm{d}\boldsymbol{\phi} \\ &\geq \int q(\boldsymbol{\phi}) \log \frac{p(\mathbf{data}, \boldsymbol{\phi}|\boldsymbol{\xi})}{q(\boldsymbol{\phi})} \mathrm{d}\boldsymbol{\phi} \triangleq \mathcal{F}(q, \boldsymbol{\xi}). \end{aligned} \tag{4.39}$$

Note that the inequality is true for *any* density q over the shower parameters $\boldsymbol{\phi} = \{(\nu_i, \beta_i, \gamma_i)\}_{i=1}^n$. The main idea of the expectation maximization (EM) algorithm is to alternately optimize $\mathcal{F}(q, \boldsymbol{\xi})$ in its two parameters.

1. In the *E-step*, we fix the hyperparameters $\boldsymbol{\xi}^{(t)}$, and optimize $\mathcal{F}(q, \boldsymbol{\xi})$ in q. The difference between the marginal likelihood and $\mathcal{F}(q, \boldsymbol{\xi}^{(t)})$ is

$$\begin{aligned} \log p(\mathbf{data}|\boldsymbol{\xi}^{(t)}) - \mathcal{F}(q, \boldsymbol{\xi}^{(t)}) &= \log p(\mathbf{data}|\boldsymbol{\xi}^{(t)}) - \int q(\boldsymbol{\phi}) \log \frac{p(\mathbf{data}, \boldsymbol{\phi}|\boldsymbol{\xi}^{(t)})}{q(\boldsymbol{\phi})} \mathrm{d}\boldsymbol{\phi} \\ &= \log p(\mathbf{data}|\boldsymbol{\xi}^{(t)}) - \int q(\boldsymbol{\phi}) \log \frac{p(\boldsymbol{\phi}|\mathbf{data}, \boldsymbol{\xi}^{(t)}) p(\mathbf{data}|\boldsymbol{\xi}^{(t)})}{q(\boldsymbol{\phi})} \mathrm{d}\boldsymbol{\phi} \\ &= - \int q(\boldsymbol{\phi}) \log \frac{p(\boldsymbol{\phi}|\mathbf{data}, \boldsymbol{\xi}^{(t)})}{q(\boldsymbol{\phi})} \mathrm{d}\boldsymbol{\phi}, \end{aligned}$$

   which is the so-called *Kullback-Leibler (KL) divergence* between $q(\boldsymbol{\phi})$ and the *posterior* distribution $p(\boldsymbol{\phi}|\mathbf{data}, \boldsymbol{\xi}^{(t)})$ given the *fixed* hyperparameters $\boldsymbol{\xi}^{(t)}$. The KL divergence is always non-negative, and it is zero if and only if the two densities are identical. This means that $\mathcal{F}(q, \boldsymbol{\xi}^{(t)})$ is maximized when

$$q^{(t)}(\boldsymbol{\phi}) = \arg\max_q \mathcal{F}(q, \boldsymbol{\xi}^{(t)}) = p(\boldsymbol{\phi}|\mathbf{data}, \boldsymbol{\xi}^{(t)}). \tag{4.40}$$

   Finding the exact posterior $p(\boldsymbol{\phi}|\mathbf{data}, \boldsymbol{\xi}^{(t)})$ is still infeasible. Since the shower events are independent, the posterior factorizes into

$$p(\boldsymbol{\phi}|\mathbf{data}, \boldsymbol{\xi}^{(t)}) = \prod_{i=1}^n p(\phi_i|\mathrm{data}_i, \boldsymbol{\xi}^{(t)}),$$

   so we can fit each shower independently. Furthermore, by Bayes' theorem

$$p(\phi_i|\mathrm{data}_i, \boldsymbol{\xi}^{(t)}) = \frac{p(\mathrm{data}_i|\phi_i) p(\phi_i|\boldsymbol{\xi}^{(t)})}{p(\mathrm{data}_i)} \propto p(\mathrm{data}_i|\phi_i) p(\phi_i|\boldsymbol{\xi}^{(t)}),$$

   which is exactly our likelihood (4.35) or (4.36) with fixed constraints $\boldsymbol{\xi}^{(t)} = \{\mu_\bullet^{(t)}, \sigma_\bullet^{(t)}\}_{\bullet=\beta,\gamma,\nu}$. We approximate[10] $p(\phi_i|\mathrm{data}_i, \boldsymbol{\xi}^{(t)})$ with the product of three independent Gaussians

$$p(\phi_i|\mathrm{data}_i, \boldsymbol{\xi}^{(t)}) \approx \mathcal{N}_{\widehat{\beta}_i, \widehat{\sigma}_{\beta_i}}(\beta_i) \mathcal{N}_{\widehat{\gamma}_i, \widehat{\sigma}_{\gamma_i}}(\gamma_i) \mathcal{N}_{\widehat{\nu}_i, \widehat{\sigma}_{\nu_i}}(\nu_i), \tag{4.41}$$

showers based on geometry, and used this set to estimate the slope prior μ_β. We then used a second round with μ_β as constraint for all showers.

[10] From where the title "approximate EM" comes from.

and estimate the means $\widehat{\beta}_i^{(t)}$, $\widehat{\gamma}_i^{(t)}$, and $\widehat{\nu}_i^{(t)}$ by maximizing (4.32) or (4.33) numerically. Finally, we estimate the error bars $\widehat{\sigma}_{\beta_i}^{(t)}$, $\widehat{\sigma}_{\gamma_i}^{(t)}$, and $\widehat{\sigma}_{\nu_i}^{(t)}$ by matching one-sigma regions. In a nutshell, this step is very similar to "classical" maximum likelihood fitting with Gaussian ($\equiv$ squared) constraints.

2. In the *M-step*, we fix the posterior $q^{(t)}(\boldsymbol{\phi})$ and maximize $\mathcal{F}(q^{(t)}, \boldsymbol{\xi})$ (4.39) in its second parameter to obtain

$$\boldsymbol{\xi}^{(t+1)} = \arg\max_{\boldsymbol{\xi}} \mathcal{F}(q^{(t)}, \boldsymbol{\xi}).$$

Since the posterior entropy

$$H(\boldsymbol{\phi}) = -\int q^{(t)}(\boldsymbol{\phi}) \log q^{(t)}(\boldsymbol{\phi}) \mathrm{d}\boldsymbol{\phi}$$

is constant in this step, maximizing $\mathcal{F}(q^{(t)}, \boldsymbol{\xi})$ is equivalent to maximizing the *expected* complete data likelihood

$$\boldsymbol{\xi}^{(t+1)} = \arg\max_{\boldsymbol{\xi}} \int q^{(t)}(\boldsymbol{\phi}) \log p(\mathbf{data}, \boldsymbol{\phi}|\boldsymbol{\xi}) \mathrm{d}\boldsymbol{\phi}.$$

Since $q^{(t)}(\boldsymbol{\phi}) p(\mathbf{data}|\boldsymbol{\phi})$ does not depend on $\boldsymbol{\xi}$, this further simplifies to

$$\begin{aligned} \boldsymbol{\xi}^{(t+1)} &= \arg\max_{\boldsymbol{\xi}} \int q^{(t)}(\boldsymbol{\phi}) \log p(\mathbf{data}, \boldsymbol{\phi}|\boldsymbol{\xi}) \mathrm{d}\boldsymbol{\phi} \\ &= \arg\max_{\boldsymbol{\xi}} \int q^{(t)}(\boldsymbol{\phi}) \big(\log p(\mathbf{data}|\boldsymbol{\phi}) + \log p(\boldsymbol{\phi}|\boldsymbol{\xi}) \big) \mathrm{d}\boldsymbol{\phi} \\ &= \arg\max_{\boldsymbol{\xi}} \int q^{(t)}(\boldsymbol{\phi}) \log p(\boldsymbol{\phi}|\boldsymbol{\xi}) \mathrm{d}\boldsymbol{\phi}. \end{aligned} \tag{4.42}$$

Plugging (4.41) into $q^{(t)}(\boldsymbol{\phi})$ and (4.34) into $p(\boldsymbol{\phi}|\boldsymbol{\xi})$ we end up with three independent minimizations for β, γ, and ν

$$\begin{aligned} (\mu_\beta^{(t+1)}, \sigma_\beta^{(t+1)}) &= \arg\max_{\mu_\beta, \sigma_\beta} \sum_{i=1}^{n} \int \mathcal{N}_{\widehat{\beta}_i^{(t)}, \widehat{\sigma}_{\beta_i}^{(t)}}(\beta) \log \mathcal{N}_{\mu_\beta(E_i,\theta_i), \sigma_\beta(E_i,\theta_i)}(\beta) \mathrm{d}\beta \\ &= \arg\min_{\mu_\beta, \sigma_\beta} \sum_{i=1}^{n} \log \sigma_\beta(E_i,\theta_i) + \frac{1}{2} \frac{(\mu_\beta(E_i,\theta_i) - \widehat{\beta}_i^{(t)})^2 + \widehat{\sigma}_{\beta_i}^{(t)^2}}{\sigma_\beta(E_i,\theta_i)^2}, \\ (\mu_\gamma^{(t+1)}, \sigma_\gamma^{(t+1)}) &= \arg\min_{\mu_\gamma, \sigma_\gamma} \sum_{i=1}^{n} \log \sigma_\gamma(E_i,\theta_i) + \frac{1}{2} \frac{(\mu_\gamma(E_i,\theta_i) - \widehat{\gamma}_i^{(t)})^2 + \widehat{\sigma}_{\gamma_i}^{(t)^2}}{\sigma_\gamma(E_i,\theta_i)^2}, \\ (\mu_\nu^{(t+1)}, \sigma_\nu^{(t+1)}) &= \arg\min_{\mu_\nu, \sigma_\nu} \sum_{i=1}^{n} \log \sigma_\nu(E_i,\theta_i) + \frac{1}{2} \frac{(\mu_\nu(E_i,\theta_i) - \widehat{\nu}_i^{(t)})^2 + \widehat{\sigma}_{\nu_i}^{(t)^2}}{\sigma_\nu(E_i,\theta_i)^2}. \end{aligned} \tag{4.43}$$

The minimizations can be done over any family of functions for $\mu_\bullet(E,\theta)$ and $\sigma_\bullet(E,\theta)$. For the mean dependencies $\mu_\bullet(E,\theta)$, we started by trying to match the natural shape of the data using polynomials of increasing degree. We found that some of these functions (especially the zenith-angle-dependencies) were smooth but non-polynomial in the sense that for a good fit we would have needed a large number of terms. Our solution was to model first order trends using (at most) third-order polynomials, and then fit the residuals with smooth nonparametric functions using Gaussian processes [160]. The residuals $\sigma_\bullet(E,\theta)$ were linear functions in all cases. As a last technical detail, we carried out these minimizations alternating between minimizing in $\mu_\bullet(E,\theta)$ and in $\sigma_\bullet(E,\theta)$. The fits converged after 2-3 iterations.

The minimizations in (4.43) are almost χ^2 fits onto the estimated parameters $(\widehat{\beta}_i^{(t)}, \widehat{\gamma}_i^{(t)}, \widehat{\nu}_i^{(t)})$ with parametrized standard deviations, done regularly in other analyses. The only difference is the individual error bar terms $\widehat{\sigma}_{\bullet_i}$ added in quadrature to the squared deviations $(\mu_\bullet(E_i, \theta_i) - \widehat{\bullet})^2$ that, quite intuitively, capture the fact that the total variance is the sum of the variance coming from the shower-to-shower fluctuation and the variance coming from the uncertainty of the parameter estimates.

The two EM iterations are often summarized in one step where the goal is to maximize

$$Q(\boldsymbol{\xi}|\boldsymbol{\xi}^{(t)}) = \mathbb{E}_{\boldsymbol{\phi}|\mathbf{data},\boldsymbol{\xi}^{(t)}} \{\log p(\mathbf{data}, \boldsymbol{\phi}|\boldsymbol{\xi})\}$$

in $\boldsymbol{\xi}$, which is indeed the combination of (4.40) and (4.42).

Under regularity conditions on $p(\mathbf{data}|\boldsymbol{\xi})$, the sequence $(\boldsymbol{\xi}^{(t)})$ converges to the marginal MLE. In practice we started with wide Gaussian priors $\mathcal{N}_{-3,2}$ for β and $\mathcal{N}_{-1,2}$ for γ, and a flat (uniform) prior for ν. This meant that the first fits were quite "wild" sometimes, and the mean functions $\mu_\bullet(E, \theta)$ and the fits changed quite considerably between the first and the second iterations. One of the effects was the disappearance of outliers, but the most important consequence of the EM iteration is that the detector resolution was increased two-fold on average. We saw very little change after the third iteration, so in all our analyses the number of EM iterations is set to 3.

The trigger probability table $f_\iota(N)$ can, in principle, be also optimized in the E-step. We found that letting $f_\iota(N)$ free gave too much flexibility to the fit, so we decided to fix it to $\{f_\iota(0) = 0, f_\iota(1) = 0.1, f_\iota(2) = 0.6, f_\iota(3) = 0.9, f_\iota(4) = 1.0, \ldots\}$. Not using the non-triggering detectors at all generated rather large biases and trends. Among the three shower parameters, only the curvature γ is sensitive to the actual values in the probability table which was one of the reasons we kept γ free.

4.5.4 Results

We first illustrate the two steps of the algorithm on an example then show some results on four simulation libraries. A more detailed analysis as well as results on Auger data can be found in [72].

Individual LDF reconstructions

Figure 4.17 shows the effect of the EM iterations on some example reconstructions. The first row contains "nice" events with detectors spaced uniformly around 1000 m and with good detectorwise estimations of the number of muons. These events are reconstructed well in the first iteration, so the gradually tightening prior or constraint (4.34) has no significant effect on the estimated $\widehat{N}_\mu(1000)$.

The second row contains events with bad geometry: the detector near the shower core is saturated, so all usable detectors are further from shower core than 1000 m. In the standard LDF reconstruction, these events profit from the fixed slope. The EM rounds have a similar effect here: the slope and the curvature is constrained by the tightening prior (4.34). The advantages are that 1) we do not have to fix the slope beforehand, the average slope (given the energy and the zenith angle is estimated from data), and 2) we do not fix the slope at all, allowing for shower-to-shower fluctuation. "Nice" but fluctuating events profit from this additional flexibility.

The third row contains some "exotic" events from the point of view of reconstruction. Inclined events (Figure 4.17(e)) often have non-triggering detectors downstream relatively close to the shower core. The upstream and downstream detectors usually average out, but, depending the actual geometry, the shape of unconstrained fit might be significantly altered, biasing also the $\widehat{N}_\mu(1000)$ estimate. The solution to this problem would be a parametrization that can account for the asymmetry. In the meantime, pulling the slope and curvature back towards the mean using our iterative reconstruction can correct this bias.

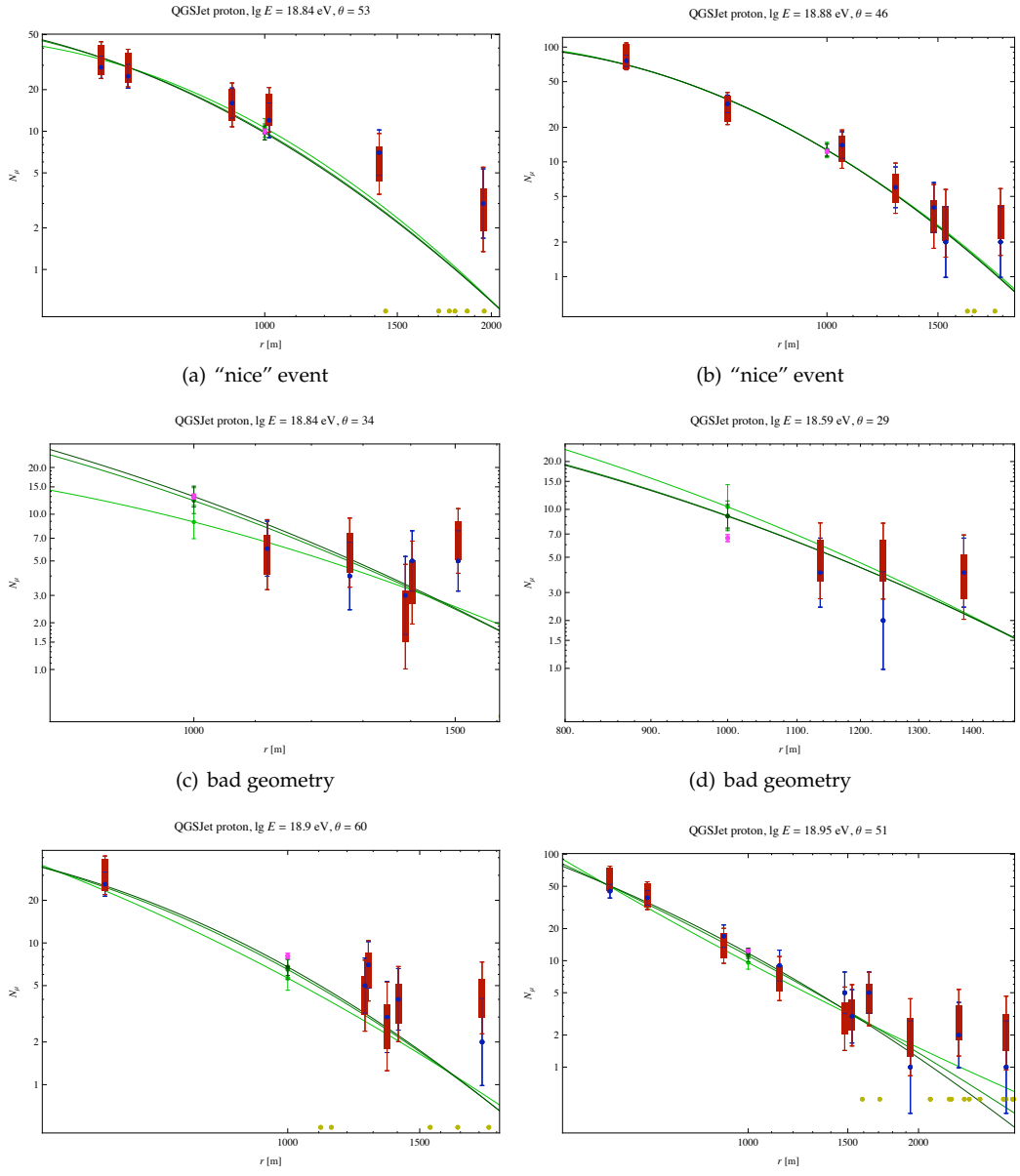
The detectorwise estimator is far from being perfect, but the estimation error usually cancels out when the event contains a relatively large number of detectors. However, in some unlucky cases, the estimation error can have a "structure". Figure 4.17(f) depicts an event where the number of muons are underestimated near 1000 m and overestimated at the edges of the fit, causing the unconstrained (log-log) LDF to be convex. Again, the effect of the data-dependent constraints (4.34) is that the curvature is pulled back towards the population mean, greatly improving the $\widehat{N}_\mu(1000)$ estimate.

Fitting the constraints

Figure 4.18 shows the results of fitting the constraints in the M-step (4.43) after the first EM iteration on the QGSJet proton set. The left panels show the original polynomial fit and the fit after the small Gaussian process correction. The right panels show the residuals after the polynomial fit and the GP fit onto the residuals.

Bias and variance reduction

In this section we show some of the results we obtained on simulated shower libraries. We used our method to reconstruct the showers in the four simulation libraries using the real number of muons $N_{i,j}$ and the corresponding likelihood (4.35). Figures 4.19 and 4.20 show the reconstruction bias and standard deviation vs. the energy. These figures illustrate the main force of the iterative reconstruction: the detector resolution (the estimation error) improves significantly. Observe that the improvement is larger for the iron sets than for the proton simulations. This is not surprising. Iron showers fluctuate less than proton showers, so, assuming that the reconstruction quality of the shower sets is the same, it is more advantageous to bias the individual estimates towards the mean for iron than for proton. In fact if the individual error bars of the unconstrained fits are larger than the shower-to-shower fluctuation, then the most efficient estimator for *individual* showers is the population mean. We run the EM loop up to 8 iterations.



(a) "nice" event

(b) "nice" event

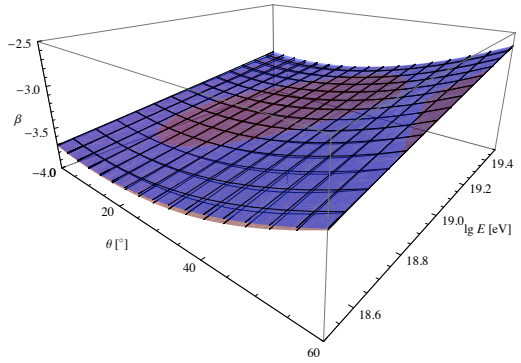
(c) bad geometry

(d) bad geometry

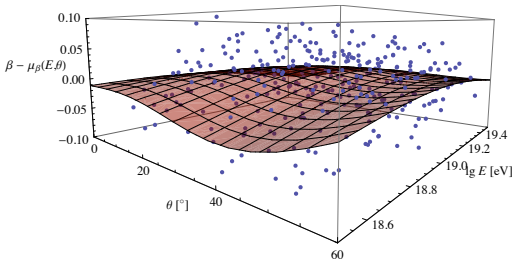
(e) inclined, non-triggering detectors, asymmetry

(f) correcting the detectorwise

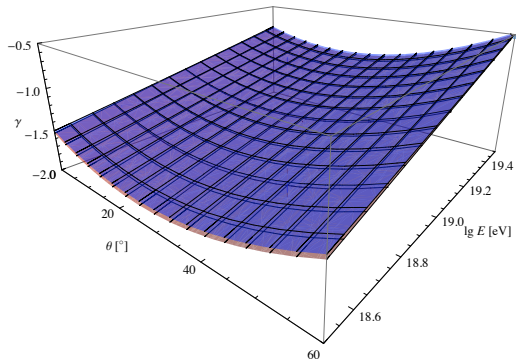
Figure 4.17: Example reconstructions. The blue dots are real number of muons N_μ with $\sqrt{N_\mu}$ Poisson error bars. The red bars are one-sigma regions of the estimated number of muons $\widehat{N}_\mu = \mathbb{E}\left\{P_{\mathrm{ANN}}(N|\mathbf{x})\right\}$ with $\widehat{\sigma}_{N_\mu} = \sqrt{\mathrm{Var}\left\{P_{\mathrm{ANN}}(N|\mathbf{x})\right\}}$ of the ANN, and the red error bars are the complete approximate error bars $\sqrt{\widehat{N}_\mu + \widehat{\sigma}^2_{N_\mu}}$. The yellow dots are non-triggering detectors. The magenta dot is the real number of muons at 1000 m, estimated from the 12 artificial detectors. The green curves are the LDF fits: light green = first EM iteration, green = second EM iteration, dark green = third EM iteration. The green dots and error bars are the corresponding $\widehat{N}_\mu(1000)$ estimates.



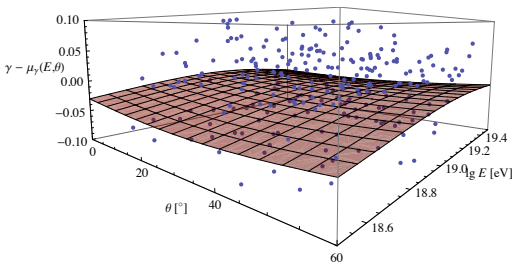
(a) The polynomial and GP mean constraints $\mu_\beta(E,\theta)$



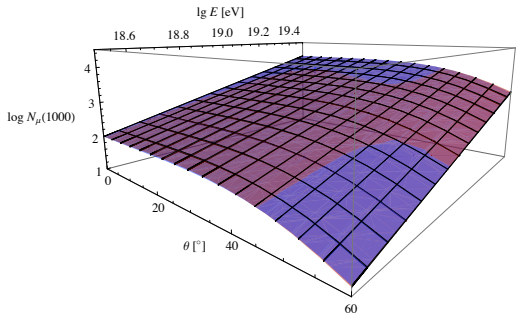
(b) β residuals and the GP fit



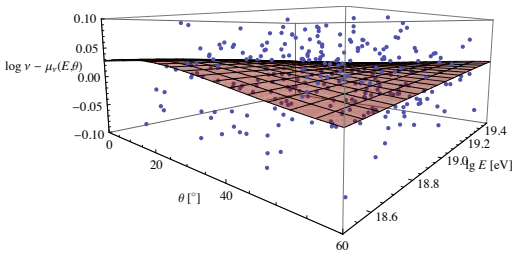
(c) The polynomial and GP mean constraints $\mu_\gamma(E,\theta)$



(d) γ residuals and the GP fit

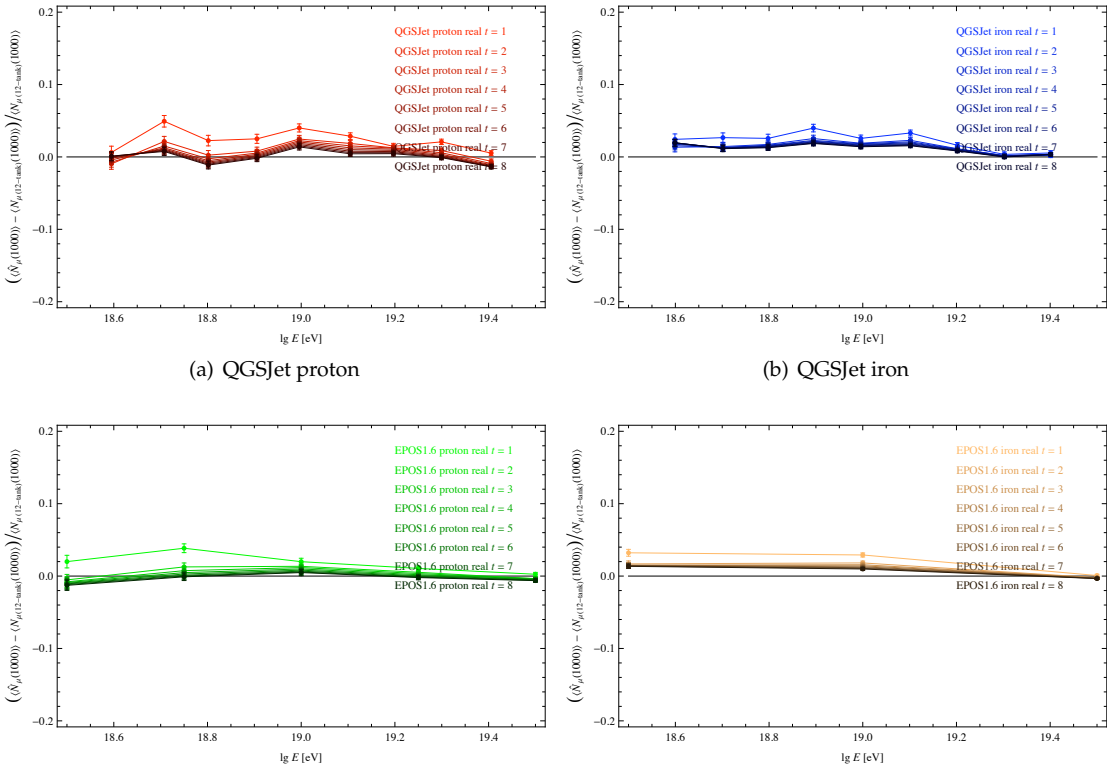


(e) The polynomial and GP mean constraints $\mu_\nu(E,\theta)$



(f) $\log N_\mu(1000)$ residuals and the GP fit

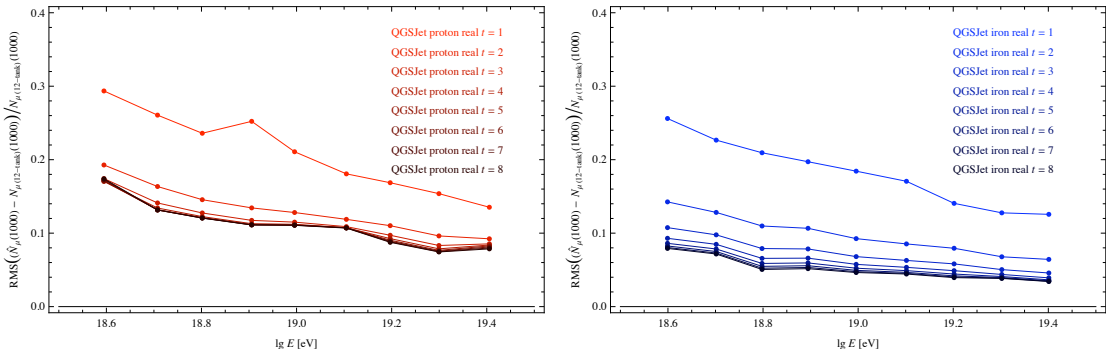
Figure 4.18: Left panels: the mean constraints $\mu_\bullet(E,\theta)$ after the polynomial fit (blue) and after the GP correction (red). Right panels: the residuals after the polynomial fit and the GP fit onto the residuals (that is, the difference between the blue and red surfaces of the left panels).



(a) QGSJet proton (b) QGSJet iron

(c) EPOS1.6 proton (d) EPOS1.6 iron

Figure 4.19: Relative bias vs. the energy of the reconstructed $\widehat{N}_\mu(1000)$ using the real number of muons in the detectors, and the nominal (simulation) energy, zenith angle, and shower core.



(a) QGSJet proton (b) QGSJet iron

(c) EPOS1.6 proton (d) EPOS1.6 iron

Figure 4.20: Relative RMS vs. the energy of the reconstructed $\widehat{N}_\mu(1000)$ using the real number of muons in the detectors, and the nominal (simulation) energy, zenith angle, and shower core.

Bibliography

[1] L. Devroye, L. Györfi, and G. Lugosi, *A Probabilistic Theory of Pattern Recognition*, Springer, New York, 1996.

[2] K. Pearson, "On lines and planes of closest fit to systems of points in space," *Philos. Mag.*, vol. 6, no. 2, pp. 559–572, 1901.

[3] H. Hotelling, "Analysis of a complex of statistical variables into principal components," *Journal of Educational Psychology*, vol. 24, pp. 417–441, 498–520, 1933.

[4] K. Karhunen, "Über lineare methoden in der wahrscheinlichkeitsrechnung," *Amer. Acad. Sci., Fennicade, Ser. A, I*, vol. 37, pp. 3–79, 1947, (Translation: RAND Corporation, Santa Monica, California, Rep. T-131, Aug. 1960).

[5] J. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the Fifth Berkeley Symposium on Mathematics, Statistics and Probability*, 1967, pp. 281–296.

[6] Y. Linde, A. Buzo, and R. M. Gray, "An algorithm for vector quantizer design," *IEEE Transactions on Communications*, vol. COM-28, no. 1, pp. 84–95, 1980.

[7] T. Kohonen, "Clustering, taxonomy, and topological maps of patterns," in *Proceedings of the 6th International Conference on Pattern Recognition*, Munich, 1982, pp. 114–128.

[8] M.A. Kramer, "Nonlinear principal component analysis using autoassociative neural networks," *AIChE Journal*, vol. 37, no. 2, pp. 233–243, 1991.

[9] S. Roweis and Saul L. K., "Nonlinear dimensionality reduction by locally linear embedding," *Science*, vol. 290, pp. 2323–2326, 2000.

[10] J. B. Tenenbaum, V. de Silva, and Langford J. C., "A global geometric framework for nonlinear dimensionality reduction," *Science*, vol. 290, pp. 2319–2323, 2000.

[11] R. I. Kondor and J. Lafferty, "Diffusion kernels on graphs and other discrete input spaces," in *Proceedings of the 19th International Conference on Machine Learning*, 2002.

[12] M. Belkin and P. Niyogi, "Laplacian eigenmaps for dimensionality reduction and data representation," *Neural Computation*, vol. 15, no. 6, pp. 1373–1396, 2003.

[13] G. E. Hinton and Salakhutdinov. R., "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5785, pp. 504–507, 2006.

[14] D. Erhan, Y. Bengio, A. Courville, P. Manzagol, P. Vincent, and S. Bengio, "Why does unsupervised pre-training help deep learning?," *Journal of Machine Learning Research*, vol. 11, pp. 625–660, 2010.

[15] B. Kégl, A. Krzyżak, T. Linder, and K. Zeger, "Learning and design of principal curves," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 3, pp. 281–297, 2000.

[16] B. Kégl and A. Krzyżak, "Piecewise linear skeletonization using principal curves," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 1, pp. 59–74, 2002.

[17] A. Imiya and N. Yamagishi, "Principal curve analysis for temporal data," in *17th International Conference on Pattern Recognition*, 2004, vol. 2, pp. 475–478.

[18] D. Chen, J. Zhang, S. Tanks, and J. Wang, "Freeway traffic stream modelling based on principal curves and its analysis," *IEEE Transactions on Intelligent Transportation Systems*, vol. 5, no. 4, pp. 246–258, 2004.

[19] I. Cleju, P. Fränti, and Wu. X., "Clustering based on principal curve," *Image Analysis*, pp. 61–73, 2005.

[20] B. Bhushan, J. A. Romagnoli, and Wang D., "Fault detection using radial basis function network and polygonal line," in *16th IFAC world congress*, 2005.

[21] P. N. Bernasconi, D. M. Rust, and D. Hakim, "Advanced automated solar filament detection and characterization code: Description, performance, and results," *Solar Physics*, vol. 228, pp. 97–117, 2005.

[22] F. Zhang, "Nonlinear feature extraction and dimension reduction by polygonal principal curves," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 20, no. 1, pp. 63–78, 2006.

[23] W. Wilbur and A. Chung, "Principal curves to extract vessels in 3D angiograms," in *Computer Vision and Pattern Recognition Workshops*, 2008, pp. 1–8.

[24] C. Rubbia et al., "Underground operation of the ICARUS T600 LAr-TPC: first results," *Journal of Instrumentation (submitted)*, 2011.

[25] S. Sandilya and S. R. Kulkarni, "Principal curves with bounded turn," *IEEE Transactions on Information Theory*, vol. 48, no. 10, pp. 2789–2793, 2002.

[26] J.J. Verbeek, N. Vlassis, and B. Kröse, "A k-segments algorithm for finding principal curves," *Pattern Recognition Letters*, vol. 23, no. 8, pp. 1009–1017, 2002.

[27] J. Einbeck, G. Tutz, and L. Evers, "Local principal curves," *Statistics and Computing*, vol. 15, no. 4, pp. 301–313, 2005.

[28] A. Gorban and A. Zinovyev, "Elastic principal graphs and manifolds and their practical applications," *Computing*, vol. 75, no. 4, pp. 359–379, 2005.

[29] U. Ozertem and D. Erdogmus, "Locally defined principal curves and surfaces," *Journal of Machine Learning Research*, pp. 1249–1286, 2011.

[30] G. Biau and A. Fischer, "Parameter selection for principal curves," *(submitted)*, 2011.

[31] B. Kégl, "Intrinsic dimension estimation using packing numbers," in *Advances in Neural Information Processing Systems*. Dec. 2002, vol. 15, pp. 681–688, The MIT Press.

[32] T. Hastie, *Principal curves and surfaces*, Ph.D. thesis, Stanford University, 1984.

[33] T. Hastie and W. Stuetzle, "Principal curves," *Journal of the American Statistical Association*, vol. 84, pp. 502–516, 1989.

[34] P. L. Bartlett, S. Boucheron, and G. Lugosi, "Model selection and error estimation," *Machine Learning*, vol. 48, pp. 85–113, 2001.

[35] G. Biau, L. Devroye, and G. Lugosi, "On the performance of clustering in Hilbert spaces," *IEEE Transactions on Information Theory*, vol. 54, pp. 781–790, 2008.

[36] L. Birgé and P. Massart, "Minimal penalties for Gaussian model selection," *Probability Theory and Related Fields*, vol. 138, pp. 33–73, 2007.

[37] P. Delicado, "Another look at principal curves and surfaces," *Journal of Multivariate Analysis*, vol. 77, no. 1, pp. 84–116, 2002.

[38] T. Kohonen, *The Self-Organizing Map*, Springer-Verlag, 2nd edition, 1997.

[39] T. F. Cox and M. A. Cox, *Multidimensional Scaling*, Chapman & Hill, 1994.

[40] E. Chávez, G. Navarro, R. Baeza-Yates, and J. Marroquín, "Searching in metric spaces," *ACM Computing Surveys*, p. to appear, 2001.

[41] P. Grassberger and I. Procaccia, "Measuring the strangeness of strange attractors," *Physica*, vol. D9, pp. 189–208, 1983.

[42] F. Camastra and A. Vinciarelli, "Estimating intrinsic dimension of data with a fractal-based approach," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2002, to appear.

[43] A. Belussi and C. Faloutsos, "Spatial join selectivity estimation using fractal concepts," *ACM Transactions on Information Systems*, vol. 16, no. 2, pp. 161–201, 1998.

[44] 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*, vol. 55, pp. 119–139, 1997.

[45] R. E. Schapire and Y. Singer, "Improved boosting algorithms using confidence-rated predictions," *Machine Learning*, vol. 37, no. 3, pp. 297–336, 1999.

[46] B. Kégl and R. Busa-Fekete, "Boosting products of base classifiers," in *International Conference on Machine Learning*, Montreal, Canada, 2009, vol. 26, pp. 497–504.

[47] J. Bergstra, R. Bardenet, Bengio Y., and B. Kégl, "Algorithms for hyper-parameter optimization," in *submitted*, 2011.

[48] D. S. Johnson and F. P. Preparata, "The densest hemisphere problem," *Theoretical Computer Science*, vol. 6, pp. 93–107, 1978.

[49] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, 1986.

[50] C. M. Bishop, *Neural Networks for Pattern Recognition*, Oxford University Press, 1995.

[51] B. Boser, I. Guyon, and V. Vapnik, "A training algorithm for optimal margin classifiers," in *Fifth Annual Workshop on Computational Learning Theory*, 1992, pp. 144–152.

[52] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.

[53] V. N. Vapnik, *Statistical Learning Theory*, Wiley, New York, 1998.

[54] P. Bartlett, "For valid generalization, the size of the weights is more important than the size of the network," in *Advances in Neural Information Processing Systems*, 1997, vol. 9, pp. 134–140.

[55] P. Bartlett, "The sample complexity of pattern classification with neural networks: the size of the weights is more important than the size of the network," *IEEE Transactions on Information Theory*, vol. 44, no. 2, pp. 525–536, 1998.

[56] I. Nabney, *Netlab: Algorithms for Pattern Recognition*, Springer, 2002.

[57] L. Bottou and O. Bousquet, "The tradeoffs of large scale learning," in *Advances in Neural Information Processing Systems*, 2008, vol. 20, pp. 161–168.

[58] L. Mason, P. Bartlett, J. Baxter, and M. Frean, "Boosting algorithms as gradient descent," in *Advances in Neural Information Processing Systems*. 2000, vol. 12, pp. 512–518, The MIT Press.

[59] L. Mason, P. Bartlett, and J. Baxter, "Improved generalization through explicit optimization of margins," *Machine Learning*, vol. 38, no. 3, pp. 243–255, March 2000.

[60] M. Collins, R.E. Schapire, and Y. Singer, "Logistic regression, AdaBoost and Bregman distances," *Machine Learning*, vol. 48, pp. 253–285, 2002.

[61] R. E. Schapire, Y. Freund, P. Bartlett, and W. S. Lee, "Boosting the margin: a new explanation for the effectiveness of voting methods," *Annals of Statistics*, vol. 26, no. 5, pp. 1651–1686, 1998.

[62] P. Viola and M. Jones, "Robust real-time face detection," *International Journal of Computer Vision*, vol. 57, pp. 137–154, 2004.

[63] G. Dror, M. Boullé, I. Guyon, V. Lemaire, and D. Vogel, Eds., *Proceedings of KDD-Cup 2009 competition*, vol. 7 of *JMLR Workshop and Conference Proceedings*, 2009.

[64] Olivier Chapelle and Yi Chang, "Yahoo! learning to rank challenge overview," in *Yahoo Learning to Rank Challenge (JMLR W&CP)*, Haifa, Israel, 2010, vol. 14, pp. 1–24.

[65] L. von Ahn, B. Maurer, C. McMillen, D. Abraham, and Blum M., "reCAPTCHA: Human-based character recognition via web security measures," *Science*, vol. 321, no. 5895, pp. 1465–1468, 2008.

[66] L. Von Ahn and L. Dabbish, "Labeling images with a computer game," in *Conference on Human factors in computing systems (CHI04)*, 2004, pp. 319–326.

[67] L. Von Ahn, "Games with a purpose," *Computer*, vol. 39, no. 6, 2006.

[68] A. Beygelzimer, J. Langford, and B. Zadrozny, *Performance Modeling and Engineering*, chapter Machine Learning Techniques–Reductions Between Prediction Quality Metrics, Springer, 2008.

[69] J. Bennett and S. Lanning, "The Netflix prize," in *KDDCup 2007*, 2007.

[70] N. Casagrande, D. Eck, and B. Kégl, "Geometry in sound: A speech/music audio classifier inspired by an image classifier," in *International Computer Music Conference*, Sept. 2005, vol. 17.

[71] J. Bergstra, N. Casagrande, D. Erhan, D. Eck, and B. Kégl, "Aggregate features and AdaBoost for music classification," *Machine Learning Journal*, vol. 65, no. 2/3, pp. 473–484, 2006.

[72] B. Kégl, R. Busa-Fekete, K. Louedec, R. Bardenet, X. Garrido, I.C. Mariş, D. Monnier-Ragaigne, S. Dagoret-Campagne, and M. Urban, "Reconstructing $N_{\mu 19}(1000)$," Tech. Rep. 2011-054, Auger Project Technical Note, 2011.

[73] Pierre Auger Collaboration, "Pierre Auger project design report," Tech. Rep., Pierre Auger Observatory, 1997.

[74] R. Busa-Fekete, B. Kégl, T. Éltető, and Gy. Szarvas, "Ranking by calibrated AdaBoost," in *Yahoo! Ranking Challenge 2010 (JMLR workshop and conference proceedings)*, 2011, vol. 14, pp. 37–48.

[75] R. Busa-Fekete, B. Kégl, T. Éltető, and Gy. Szarvas, "A robust ranking methodology based on diverse calibration of AdaBoost," in *European Conference on Machine Learning*, 2011, vol. 22.

[76] Y. Takizawa, T. Ebisuzaki, Y. Kawasaki, M. Sato, M. E. Bertaina, H. Ohmori, Y. Takahashi, F. Kajino, M. Nagano, N. Sakaki, N. Inoue, H. Ikeda, Y. Arai, Y. Takahashi, T. Murakami, James H. Adams, and the JEM-EUSO Collaboration, "JEM-EUSO: Extreme Universe Space Observatory on JEM/ISS," *Nuclear Physics B - Proceedings Supplements*, vol. 166, pp. 72–76, 2007.

[77] V. Gligorov, "A single track HLT1 trigger," Tech. Rep. LHCb-PUB-2011-003, CERN, 2011.

[78] L. Bourdev and J. Brandt, "Robust object detection via soft cascade," in *Conference on Computer Vision and Pattern Recognition*. 2005, vol. 2, pp. 236–243, IEEE Computer Society.

[79] R. Xiao, L. Zhu, and H. J. Zhang, "Boosting chain learning for object detection," in *Ninth IEEE International Conference on Computer Vision*, 2003, vol. 9, pp. 709–715.

[80] J. Sochman and J. Matas, "WaldBoost – learning for time constrained sequential detection," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2005, pp. 150–156.

[81] B. Póczos, Y. Abbasi-Yadkori, Cs. Szepesvári, R. Greiner, and N. Sturtevant, "Learning when to stop thinking and do something!," in *Proceedings of the 26th International Conference on Machine Learning*, 2009, pp. 825–832.

[82] M. Saberian and N. Vasconcelos, "Boosting classifier cascades," in *Advances in Neural Information Processing Systems 23*, pp. 2047–2055. MIT Press, 2010.

[83] T. Aaltonen et al., "Observation of electroweak single top-quark production," *Physical Review Letters*, vol. 103, no. 9, 2009.

[84] V.M. Abazov et al., "Observation of single top-quark production," *Physical Review Letters*, vol. 103, no. 9, 2009.

[85] R. Busa-Fekete, D. Benbouzid, and B. Kégl, "MDDAG: designing sparse decision DAGs using Markov decision processes," in *submitted*, 2011.

[86] W. Cohen, "Learning trees and rules with set-valued features," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 1996, pp. 709–716.

[87] W. Cohen and Y. Singer, "A simple, fast, and effective rule learner," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 1999, pp. 335–342.

[88] J. Quinlan, "Induction of decision trees," *Machine Learning*, vol. 1, no. 1, pp. 81–106, 1986.

[89] J. Quinlan, "Bagging, boosting and C4.5," in *Proceedings of the 13th National Conference on Artificial Intelligence*, 1996, pp. 725–730.

[90] J. Quinlan, *C4.5: Programs for Machine Learning*, Morgan Kaufmann, 1993.

[91] T. Cormen, C. Leiserson, and R. Rivest, *Introduction á l'algorithmique*, Dunod, 1994.

[92] N. Srebro, J. D. M. Rennie, and T. Jaakkola, "Maximum margin matrix factorization," in *Advances in Neural Information Processing Systems*. 2005, vol. 17, pp. 1329–1336, The MIT Press.

[93] I.H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques*, Morgan Kaufmann, 2005.

[94] R. Salakhutdinov and G. Hinton, "Learning a nonlinear embedding by preserving class neighbourhood structure," in *International Conference on Artificial Intelligence and Statistics*, 2007, pp. 412–419.

[95] B. Marlin, "Collaborative filtering: a machine learning perspective," M.S. thesis, University of Toronto, 2004.

[96] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[97] S. Robertson and H. Zaragoza, "The probabilistic relevance framework: BM25 and beyond," *Found. Trends Inf. Retr.*, vol. 3, pp. 333–389, 2009.

[98] P. Li, C. Burges, and Q. Wu, "McRank: Learning to rank using multiple classification and gradient boosting," in *Advances in Neural Information Processing Systems*. 2007, vol. 19, pp. 897–904, The MIT Press.

[99] Y. Freund, R. Iyer, R. E. Schapire, and Y. Singer, "An efficient boosting algorithm for combining preferences," *Journal of Machine Learning Research*, vol. 4, pp. 933–969, 2003.

[100] H. Valizadegan, R. Jin, R. Zhang, and J. Mao, "Learning to rank by optimizing NDCG measure," in *Advances in Neural Information Processing Systems 22*, 2009, pp. 1883–1891.

[101] D. Cossock and T. Zhang, "Statistical analysis of Bayes optimal subset ranking," *IEEE Transactions on Information Theory*, vol. 54, no. 11, pp. 5140–5154, 2008.

[102] Olivier Chapelle, Donald Metlzer, Ya Zhang, and Pierre Grinspan, "Expected reciprocal rank for graded relevance," in *Proceeding of the 18th ACM conference on Information and knowledge management*, New York, NY, USA, 2009, pp. 621–630, ACM.

[103] A. Niculescu-Mizil and R. Caruana, "Obtaining calibrated probabilities from boosting," in *Proceedings of the 21st International Conference on Uncertainty in Artificial Intelligence*, 2005, pp. 413–420.

[104] Q. Wu, C. J. C. Burges, K. M. Svore, and J. Gao, "Adapting boosting for information retrieval measures," *Inf. Retr.*, vol. 13, no. 3, pp. 254–270, 2010.

[105] Olivier Chapelle and Mingrui Wu, "Gradient descent optimization of smoothed information retrieval metrics," *Information Retrievel*, vol. 13, no. 3, pp. 216–235, 2010.

[106] N. Cesa-Bianchi and G. Lugosi, *Prediction, Learning, and Games*, Cambridge University Press, New York, NY, USA, 2006.

[107] J. Xu and H. Li, "AdaRank: a boosting algorithm for information retrieval," in *SIGIR '07: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*. 2007, pp. 391–398, ACM.

[108] Z. Cao, T. Qin, T. Liu, M. Tsai, and H. Li, "Learning to rank: from pairwise approach to listwise approach," in *Proceedings of the 24rd International Conference on Machine Learning*, 2007, pp. 129–136.

[109] R. Herbrich, T. Graepel, and K. Obermayer, "Large margin rank boundaries for ordinal regression," in *Advances in Large Margin Classifiers*, Smola, Bartlett, Schoelkopf, and Schuurmans, Eds. 2000, pp. 115–132, MIT Press, Cambridge, MA.

[110] O. Chapelle, Y. Chang, and Liu T. Y., "Future directions in learning to rank," in *Yahoo Learning to Rank Challenge (JMLR W&CP)*, Haifa, Israel, 2011, vol. 14, pp. 91–100.

[111] H. Lee, A. Battle, R. Raina, and A. Y. Ng, "Efficient sparse coding algorithms," in *Advances in Neural Information Processing Systems*. 2007, vol. 19, pp. 801–808, The MIT Press.

[112] M. Ranzato, C. Poultney, S. Chopra, and Y. LeCun, "Efficient learning of sparse representations with an energy-based model," in *Advances in Neural Information Processing Systems 19*, pp. 1137–1144. MIT Press, Cambridge, MA, 2007.

[113] H. Larochelle and G. Hinton, "Learning to combine foveal glimpses with a third-order Boltzmann machine," in *Advances in Neural Information Processing Systems 23*, pp. 1243–1251. MIT Press, 2010.

[114] Y. Freund and L. Mason, "The alternating decision tree learning algorithm," in *Proceedings of the 16th International Conference on Machine Learning*, 1999, pp. 124–133.

[115] G. Neu, A. György, and Cs. Szepesvári, "The online loop-free stochastic shortest-path problem," in *Proceedings of the 23th Annual Conference on Computational Learning Theory*, 2010, pp. 231–243.

[116] R.S. Sutton and A.G. Barto, *Reinforcement learning: an introduction*, Adaptive computation and machine learning. MIT Press, 1998.

[117] G. A. Rummery and M. Niranjan, "On-line Q-learning using connectionist systems," Tech. Rep. CUED/F-INFENG/TR 166, Cambridge University, Engineering Department, 1994.

[118] Cs. Szepesvári, *Algorithms for Reinforcement Learning*, Morgan and Claypool, 2010.

[119] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3, pp. 279–292, 1992.

[120] G. Davis, S. Mallat, and M. Avellaneda, "Adaptive greedy approximations," *Constructive Approximation*, vol. 13, no. 1, pp. 57–98, 1997.

[121] V. Ejov, J. Filar, and J. Gondzio, "An interior point heuristic for the Hamiltonian cycle problem via Markov Decision Processes," *Journal of Global Optimization*, vol. 29, no. 3, pp. 315–334, 2004.

[122] S. Munder and D. M. Gavrila, "An experimental study on pedestrian classification," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 28, pp. 1863–1868, 2006.

[123] B. B. Cambazoglu, H. Zaragoza, O. Chapelle, J. Chen, C. Liao, Z. Zheng, and J. Degenhardt, "Early exit optimizations for additive machine learned ranking systems," in *Proceedings of the third ACM International Conference on Web Search and Data Mining*, 2010, pp. 411–420.

[124] K. Järvelin and J. Kekäläinen, "Cumulated gain-based evaluation of IR techniques," *ACM Trans. Inf. Syst.*, vol. 20, pp. 422–446, 2002.

[125] A. Antos, B. Kégl, T. Linder, and G. Lugosi, "Data-dependent margin-based generalization bounds for classification," *Journal of Machine Learning Research*, pp. 73–98, 2002.

[126] B. Kégl, T. Linder, and G. Lugosi, "Data-dependent margin-based generalization bounds for classification," in *Proceedings of the 14th Conference on Computational Learning Theory*, 2001, pp. 368–384.

[127] B. Kégl, "Robust regression by boosting the median," in *Proceedings of the 16th Conference on Computational Learning Theory*, 2003, pp. 258–272.

[128] B. Kégl, "Generalization error and algorithmic convergence of median boosting," in *Advances in Neural Information Processing Systems*, Vancouver, Canada, 2004, vol. 17, pp. 657–664, The MIT Press.

[129] B. Kégl and L. Wang, "Boosting on manifolds: adaptive regularization of base classifiers," in *Advances in Neural Information Processing Systems*, Vancouver, Canada, 2004, vol. 17, pp. 665–672, The MIT Press.

[130] S. Gambs, B. Kégl, and E. Aïmeur, "Privacy-preserving boosting," *Data Mining and Knowledge Discovery*, vol. 14, no. 1, pp. 131–170, 2007.

[131] R. Busa-Fekete and B. Kégl, "Accelerating AdaBoost using UCB," in *KDDCup 2009 (JMLR workshop and conference proceedings)*, 2009, vol. 7, pp. 111–122.

[132] R. Busa-Fekete and B. Kégl, "Fast boosting using adversarial bandits," in *International Conference on Machine Learning*, 2010, vol. 27, pp. 143–150.

[133] B. Kégl and D. Veberič, "Single muon response: Tracklength," Tech. Rep. 2009-043, Auger Project Technical Note, 2009.

[134] R. Bardenet, B. Kégl, and D. Veberič, "Single muon response: The signal model," Tech. Rep. 2010-110, Auger Project Technical Note, 2010.

[135] H. Haario, E. Saksman, and J. Tamminen, "An adaptive Metropolis algorithm," *Bernoulli*, vol. 7, pp. 223–242, 2001.

[136] R. Bardenet, O. Cappé, G. Fort, and B. Kégl, "Adaptive Metropolis with online relabeling," in *submitted*, 2011.

[137] P. J. Green, "Reversible jump Markov chain Monte Carlo computation and Bayesian model determination," *Biometrika*, vol. 82, no. 4, pp. 711–732, 1995.

[138] H. Robbins, "An empirical Bayes approach to statistics," *roceedings of the Third Berkeley Symposium on Mathematical Statistics and Probability*, vol. 1: Contributions to the Theory of Statistics, pp. 157–163, 1956.

[139] B.P. Carlin and T.A. Louis, *Bayes and Empirical Bayes Methods for Data Analysis*, Chapman & Hall / CRC, 2000.

[140] P.S. Allison and D. Barnhill, "Calculation of the number of photoelectrons produced per tank based on PMT test data and station monitoring data," Tech. Rep. GAP-04-046, Auger Project Technical Note, 2004.

[141] M. Aglietta, "A direct measurement of the photoelectron number per vertical muon in the Capisa SD detector," Tech. Rep. GAP-05-010, Auger Project Technical Note, 2005.

[142] D. Dornic, *Développement et caractérisation de photomultiplicateurs hémisphériques pour les expériences d'astroparticules d'étalonnage des détecteurs de surface et analyse des gerbes horizontales de l'Observatoire Pierre Auger*, Ph.D. thesis, Université Paris XI, 2006, pdf.

[143] D. Ravignani et al., "Calculation of the number of photoelectrons with the water Cherenkov detector model," Tech. Rep. GAP-97-024, Auger Project Technical Note, 1997.

[144] B. Genolini, T. Nguyen Trunc, J. Pouthas, P. Lavoute, C. Meunier, M. Aglietta, and C. Morello, "Photonis XP1805 and PAO SD bases: effects of the temperature and of the Earth's magnetic field," Tech. Rep. GAP-03-017, Auger Project Technical Note, 2003.

[145] G. Roberts, A. Gelman, and W. Gilks, "Weak convergence of optimal scaling of random walk Metropolis algorithms," *The Annals of Applied Probability*, vol. 7, pp. 110–120, 1997.

[146] Roberts G. O. and Rosenthal J. S., "Optimal scaling for various Metropolis-Hastings algorithms," *Statistical Science*, vol. 16, pp. 351–367, 2001.

[147] S. Richardson and P. J. Green, "On Bayesian analysis of mixtures with an unknown number of components," *Journal of the Royal Statistical Society*, vol. 59, no. 4, pp. 731–792, 1997.

[148] G. Celeux, "Bayesian inference for mixtures: The label-switching problem," in *COMPSTAT 98*, R. Payne and P. Green, Eds. 1998, Physica-Verlag.

[149] M. Stephens, "Dealing with label switching in mixture models," *Journal of the Royal Statistical Society, Series B*, vol. 62, pp. 795–809, 2000.

[150] J.M. Marin, K. Mengersen, and C.P. Robert, "Bayesian modelling and inference on mixtures of distributions," *Handbook of Statisics*, vol. 25, 2004.

[151] A. Jasra, C. C. Holmes, and D. A. Stephens, "Markov Chain Monte Carlo methods and the label switching problem in Bayesian mixture modelling," *Statistical Science*, vol. 20, no. 1, pp. 50–67, 2005.

[152] A. Jasra, *Bayesian inference for mixture models via Monte Carlo.*, Ph.D. thesis, Imperial College London, 2005.

[153] M. Sperrin, T. Jaki, and E. Wit, "Probabilistic relabelling strategies for the label switching problem in Bayesian mixture models," *Statistics and Computing*, vol. 20, pp. 357–366, 2010.

[154] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, and E. Teller, "Equations of state calculations by fast computing machines," *Journal of Chemical Physics*, vol. 21, no. 6, pp. 1087–1092, 1953.

[155] E. Saksman and M. Vihola, "On the ergodicity of the adaptive Metropolis algorithm on unbounded domains," *Annals of Applied Probability*, vol. 20, pp. 2178–2203, 2010.

[156] G. Fort, E. Moulines, and P. Priouret, "Convergence of adaptive MCMC algorithms: ergodicity and law of large numbers," Tech. Rep., Telecom ParisTech, 2010.

[157] M. Vihola, "Can the adaptive Metropolis algorithm collapse without the covariance lower bound?," *Electronic Journal of Probability*, vol. 16, pp. 45–75, 2011.

[158] X. Garrido, A. Cordier, S. Dagoret-Campagne, B. Kégl, D. Monnier-Ragaigne, and M. Urban, "Measurement of the number of muons in Auger tanks by the FADC jump counting method," Tech. Rep. 2007-060, Auger Project Technical Note, 2007.

[159] X. Garrido, B. Kégl, A. Cordier, S. Dagoret-Campagne, D. Monnier-Ragaigne, and M. Urban, "Update and new results from the FADC jump counting method," Tech. Rep. 2009-023, Auger Project Technical Note, 2009.

[160] C.E. Rasmussen and C. Williams, *Gaussian Processes for Machine Learning*, MIT Press, 2006.

[161] A. Dempster, N. Laird, and D. Rubin, "Maximum likelihood from incomplete data via the em algorithm," *Journal of the Royal Statistical Society*, vol. Series B, 39, no. 1, pp. 1–38, 1977.