Boosting Boosting

Thesis by
Ron Appel

In Partial Fulfillment of the Requirements for the
degree of
Doctor of Philosophy

CALIFORNIA INSTITUTE OF TECHNOLOGY
Pasadena, California

2017
Defended May 22, 2017

To my parents, Mam and Tat.

ACKNOWLEDGEMENTS

I would like to thank:

Pietro Perona – as an academic mentor, for teaching me, allowing and encouraging me to work on problems that interest me, and for being patient with me through my stubbornness – and as a friend, for your kindness, your calm, your humor, and your heartfelt, sound advice in matters of life outside of academia.

My lab mates and non-lab mates – for all the fun times, all the love and support, all the help in stressful situations; I could not have found a better group of friends.

My family – for everything.

ABSTRACT

Machine learning is becoming prevalent in all aspects of our lives. For some applications, there is a need for simple but accurate white-box systems that are able to train efficiently and with little data.

"Boosting" is an intuitive method, combining many simple (possibly inaccurate) predictors to form a powerful, accurate classifier. Boosted classifiers are intuitive, easy to use, and exhibit the fastest speeds at test-time when implemented as a cascade. However, they have a few drawbacks: training decision trees is a relatively slow procedure, and from a theoretical standpoint, no simple unified framework for cost-sensitive multi-class boosting exists. Furthermore, (axis-aligned) decision trees may be inadequate in some situations, thereby stalling training; and even in cases where they are sufficiently useful, they don't capture the intrinsic nature of the data, as they tend to form boundaries that overfit.

My thesis focuses on remedying these three drawbacks of boosting. Ch. 3 outlines a method (called QuickBoost) that trains identical classifiers at an order of magnitude faster than before, based on a proof of a bound. In Ch. 4, a unified framework for cost-sensitive multi-class boosting (called REBEL) is proposed, both advancing theory and demonstrating empirical gains. Finally, Ch. 5 describes a novel family of weak learners (called Localized Similarities) that guarantee theoretical bounds and outperform decision trees and Neural Nets (as well as several other commonly used classification methods) on a range of datasets.

The culmination of my work is an easy-to-use, fast-training, cost-sensitive multi-class boosting framework whose functionality is interpretable (since each weak learner is a simple comparison of similarity), and whose performance is better than Neural Networks and other competing methods. It is the tool that everyone should have in their toolbox and the first one they try.

PUBLISHED CONTENT AND CONTRIBUTIONS

[1] R. Appel, X. P. Burgos-Artizzu, and P. Perona. "Improved Multi-Class Cost-Sensitive Boosting via Estimation of the Minimum-Risk Class". In: *arXiv* 1607.03547 (2016). URL: arxiv.org/abs/1607.03547
Ron conceived and carried out the project, with help from Xavier on some experiments. Co-authors helped in writing the paper.

[2] R. Appel and P. Perona. "A Simple Multi-Class Boosting Framework with Theoretical Guarantees and Empirical Proficiency". In: *International Conference on Machine Learning* (2017). [Accepted]
Ron conceived, carried out the project, and wrote the paper.

[3] R. Appel et al. "Quickly boosting decision trees-pruning underachieving features early". In: *International Conference on Machine Learning* (2013).
Ron conceived and carried out the project. Co-authors helped write the paper.

[4] P. Dollár, R. Appel, and W. Kienzle. "Crosstalk Cascades for Frame-Rate Pedestrian Detection". In: *European Conference on Computer Vision* (2012).
Ron implemented fast feature extraction code and helped write the paper.

[5] P. Dollár et al. "Fast Feature Pyramids for Object Detection". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2014).
Ron contributed substantially to the codebase and to some of the methods, and helped write the paper.

TABLE OF CONTENTS

NOTATION

$\mathbb{R}$	Set of real numbers $(-\infty, \infty)$
$\overline{\mathbb{R}}$	Set of extended real numbers $[-\infty, \infty]$ (i.e. including $\pm\infty$)
$P_{x,y}$	Joint probability distribution (i.e. $P(x, y)$)
$P_{y\|x}$	Posterior probability distribution (i.e. $P(y \mid x)$)
x	Scalar (non-bold font)
$f(\cdot)$	Scalar function (i.e. returns a scalar)
$\mathbb{1}(...)$	Indicator function, returns 1 if logical expression (represented as "...") evaluates to true, otherwise returns 0
$\mathbf{x}$	Vector (**bold** font)
$\mathbf{0}$	Zero vector (i.e. $[0, 0, ..., 0]$)
$\mathbf{1}$	One vector (i.e. $[1, 1, ..., 1]$)
$\boldsymbol{\delta}_k$	Indicator vector (i.e. $\mathbf{0}$ with a 1 in the k^{th} index)
$\mathbf{H}(\cdot)$	Vector-valued function (i.e. returns a vector)
x_k	Value of the k^{th} index in vector $\mathbf{x}$ (i.e. $x_k \equiv \langle \mathbf{x}, \boldsymbol{\delta}_k \rangle$)
$\underline{\mathbf{C}}$	Matrix (**Bold Underlined** font)
$\langle \mathbf{a}, \mathbf{b} \rangle$	Inner product of $\mathbf{a}$ and $\mathbf{b}$
$\mathbf{a} \odot \mathbf{b}$	Element-wise product of $\mathbf{a}$ and $\mathbf{b}$ (i.e. $[a_1 b_1, a_2 b_2, ..., a_K b_K]$)
$\mathbf{exp}[\mathbf{x}]$	Element-wise exponentiation (i.e. $[e^{x_1}, e^{x_2}, ..., e^{x_K}]$)
$\mathbf{g}[\mathbf{x}]$	Square brackets indicate element-wise function (i.e. $[g(x_1), g(x_2), ..., g(x_K)]$)
$\tilde{y}$	Estimate (tilde-hat) of a quantity y
x^*	Optimal value (star) of a quantity x

Chapter *1*

INTRODUCTION

The year is 2017 and we have just recently entered the age of machine learning. Algorithms that automatically analyze data are indispensable tools and are being heavily applied in all aspects of our lives. Companies are competing to hire machine learning experts to properly implement these algorithms. However, such powerful tools should be accessible to all members of society, not just the experts. With this body of work, I propose a simple framework that is ready to deploy with the push of a button, no expertise necessary. But first, I recount a brief history of events that motivated my work.

At the turn of the (21^{st}) century, the first object detection system capable of functioning in real-time was proposed: the Viola-Jones face detector [14]. One of the main aspects of computer vision is object detection, i.e. finding and/or identifying a particular object within in image or video frame. As humans, faces are prevalent in our daily lives and hence are particularly relevant as detectable "objects", marking the Viola-Jones detector as a major milestone. The detector was intuitively clear and easily implementable, largely owing its success to the "brains" behind its operation: the machine learning system, namely, *boosting* [7].

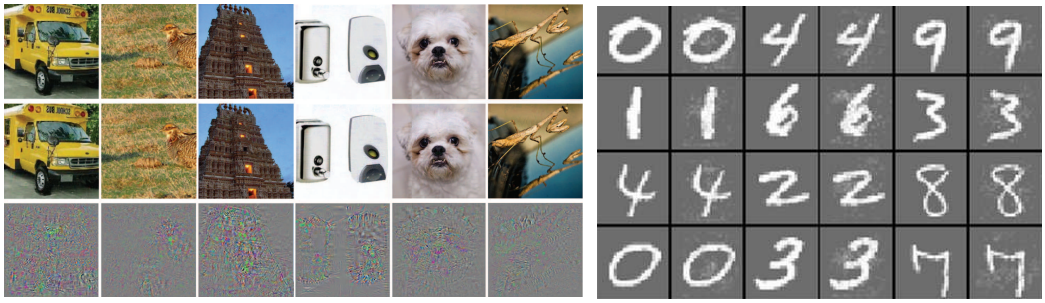
Boosting was able to produce very good visual object classifiers by combining many simple features (e.g. the average pixel intensity of a rectangular image patch). Indeed, methods based on boosting remained state-of-the-art in many fields for over a decade [6, 3, 1]. However, the good times didn't last; in 2012, AlexNet [9] paved the way for the resurgence of neural networks, leaving boosting *almost* forgotten. As of this year, deep nets of varying (obscenely large and complex) architectures have taken over, reaching almost *if not better than* human performance in many domains [10].

One of the key strengths of these networks is their ability to transform the input data by learning complex feature representations to facilitate classification [12]. However, there are several considerable drawbacks to employing such networks.

A first drawback is that the complex representations achieved by these networks are difficult to interpret and to analyze. For many riskier applications (e.g. self-driving cars, robotic surgeries, military drones, etc.), it would be better if a machine was

allowed to run autonomously only if it could *explain* its every decision and action. Further, when used towards the scientific analysis of phenomena (e.g. understanding animal behavior, weather patterns, financial market trends, etc.), the goal is to extract a causal interpretation of the system in question; hence, to be useful, a machine should be able to provide a clear explanation of its internal logic.

To demonstrate the potential flaws in the almost *magical* inner-workings of deep nets, Fig 1.1 is borrowed from [13]. Seemingly identical inputs are classified as completely different classes (in both cases, with high reported levels of confidence). [13] claims that every sample can be appropriately adjusted to be classified as any class. Although these adjustments are very specific, the fact remains that the more complex the network, the more mysterious its functionality.



Subtle adjustments to random samples from Imagenet images and MNIST digits

Figure 1.1: (Results from [13]) (left) Random Imagenet images [5], correctly classified samples (top row), incorrectly classified subtly adjusted samples (middle row), and the amplified pixel-wise difference between the two *seemingly identical* images (bottom row). (right) Random MNIST digits [11], correctly classified samples (odd columns), incorrectly classified subtly adjusted samples (even columns).

A second drawback is that training a deep network (i.e. validating through many architectures, each of which may have millions or billions of parameters) requires a lot of data and a lot of time. In many fields (e.g. pathology of not-so-common diseases, expert curation of esoteric subjects, etc.), gathering large amounts of data is expensive or even impossible [16]. Autonomous robots that need to learn on the fly may not be able to afford the large amount of processing power or time required to train more complex networks simply due to their hardware constraints. Moreover, most potential users (e.g. non-machine-learning scientists, small business owners, hobbyists, etc.) may not have the expertise or artistry required to even validate through the appropriate models.

For these reasons, it is desirable to have a simple white-box machine learning system that can train quickly and with little data. To this end, I have sped up training time, theoretically unified and empirically improved cost-sensitive multi-class boosted classification, and have proposed better constituents for use in the overall boosted classifier. My proposed boosting framework is the accurate, data-efficient, easy-to-use tool that should be in everyone's tool-bag.

To demonstrate its versatility, Fig. 1.2 shows a plot of test error rates on ten datasets ranging in amount of training data N, input sample dimensionality d, and number of output classes K, (refer to Table 1.1 for specifications). I compare my proposed method to neural networks (trained using four different architectures: $[d-4d-K]$, $[d-4K-K]$, $[d-2d-d-K]$, $[d-4K-2K-K]$, finally reporting the one that performed best on the test set) and SVMs (support vector machines [4] – another standard machine-learning method; cross-validating to find optimal parameters C and γ using a 5×6 grid search)[1]. As described above, the neural nets and SVMs required several runs in order to select appropriate hyper-parameters. My method did not require validation and led to the best accuracy in almost all cases. It is decisively the best choice of algorithm based on these results.

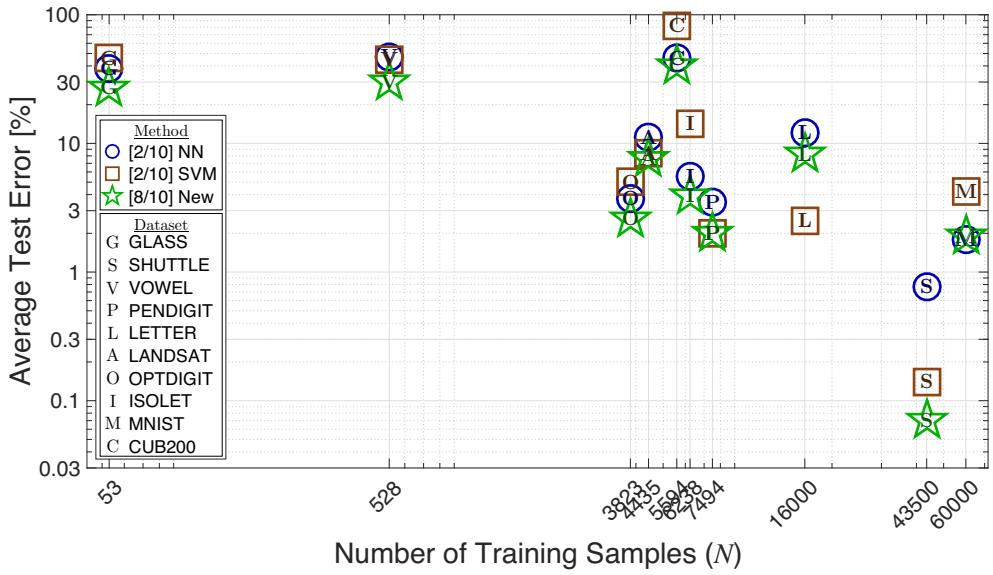


Figure 1.2: Comparison of my (New) method versus neural networks and support vector machines on ten datasets of varying sizes and difficulties. My method is the most accurate on almost every dataset.

[1] Training times for my method and neural networks were approximately equal; the SVMs took over an order of magnitude longer to train, with an astonishing 20 hours for MNIST.

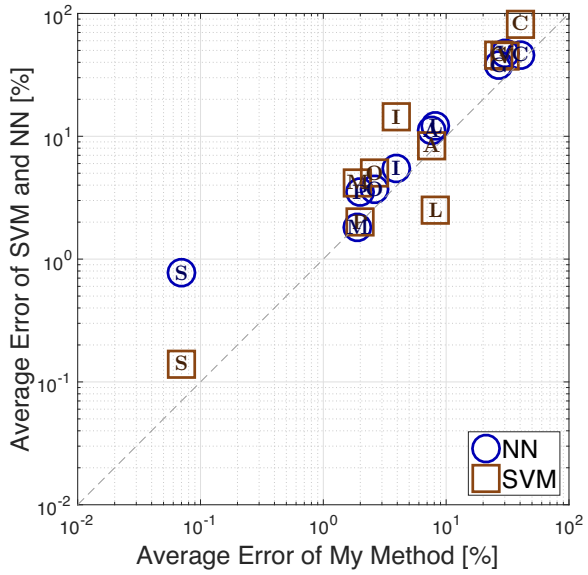


Figure 1.3: Comparison of error rates achieved using my method versus neural networks and support vector machines on ten datasets of varying sizes and difficulties. Note that almost all of the points lie in the top-left half of the plot, signifying that my method is the most accurate method on almost every dataset.

Dataset	# Input dims.	# Output classes	# Training samples	# Test samples
GLASS	9	6	53	159
SHUTTLE	9	7	43500	14500
VOWEL	10	11	528	462
PENDIGIT	16	10	7494	3498
LETTER	16	26	16000	4000
LANDSAT	36	6	4435	2000
OPTDIGIT	64	20	3823	1797
ISOLET	617	26	6238	1559
MNIST	728	10	60000	10000
CUB200	4096	200	5594	5794

Table 1.1: Specifications of datasets shown in Fig. 1.2. The first eight are UCI datasets [2], the ninth: MNIST digits [11], and the tenth: CUB200 birds [15]; input dimensions are the 4096 output features of a pre-trained ConvNet [8].

References

[1] N. Asadi and J. Lin. "Training Efficient Tree-Based Models for Document Ranking". In: *European Conference on Information Retrieval*. 2013.

[2] K. Bache and M. Lichman. *UCI Machine Learning Repository (UC Irvine)*. 2013. URL: `http://archive.ics.uci.edu/ml`.

[3] X.P. Burgos-Artizzu et al. "Social Behavior Recognition in continuous videos". In: *CVPR*. 2012.

[4] C. Cortes and V. Vapnik. "Support-vector networks". In: *Machine Learning* (1995).

[5] J. Deng et al. "ImageNet: A Large-Scale Hierarchical Image Database". In: *CVPR*. 2009.

[6] P. Dollár, R. Appel, and W. Kienzle. "Crosstalk Cascades for Frame-Rate Pedestrian Detection". In: *ECCV*. 2012.

[7] Y. Freund and R. E. Schapire. "A desicion-theoretic generalization of on-line learning and an application to boosting". In: *European Conference on Computational Learning Theory*. 1995.

[8] Y. Jia et al. "Caffe: Convolutional Architecture for Fast Feature Embedding". In: *arXiv* 1408.5093 (2014).

[9] A. Krizhevsky, I. Sutskever, and G. E. Hinton. "Imagenet classification with deep convolutional neural networks". In: *NIPS*. 2012.

[10] Y. LeCun, Y. Bengio, and G. E. Hinton. "Deep learning". In: *Nature Research* (2015).

[11] Y. LeCun and C. Cortes. *The MNIST database of handwritten digits*. Tech. rep. 1998.

[12] Y. LeCun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* (1998).

[13] C. Szegedy et al. "Intriguing properties of neural networks". In: *arXiv* 1312.6199 (2013).

[14] P. Viola and M. J. Jones. "Robust Real-Time Face Detection". In: *IJCV* (2004).

[15] P. Welinder et al. *Caltech-UCSD birds 200*. Tech. rep. 2010.

[16] F. Yu et al. "LSUN: Construction of a Large-scale Image Dataset using Deep Learning with Humans in the Loop". In: *arXiv* 1506.03365 (2015).

Chapter 2

MACHINE LEARNING

Whether or not we believe in a higher power, we can all agree that there is a certain order to our existence (however chaotic it might be). This "order" in our world is the governing laws of nature (many of which we are still uncovering) and the "chaos" is the stochasticity – or randomness – inherent in every process. These laws (which can be thought of as pre-defined patterns or phenomena) ensure that for a given set of input conditions, we can expect a specific set of output results. The scientific method itself is based on this notion of repeatability – if the same procedure is repeated multiple times, similar results should occur.

All living entities thrive from being able to leverage these patterns, if not by fully comprehending them then at least by being able to predict them. More importantly, we are able to generalize our specific experiences to the infinite number of possible experiences that we have not already explicitly encountered.

As an example, from birth, we are subject to the pull of gravity, without knowing what it is. We notice that if we don't hold ourselves up, we fall. If we let go of a ball, it falls. So does a piece of paper, although not as fast. Without having to grab and drop every object around us, we come to expect that all objects fall when released, some quickly and some slowly. And then on our birthday, we are given a helium balloon. We let it go and it immediately soars up and out of reach (and we cry). But we quickly learn to correctly predict which objects fall quickly, slowly, and which don't fall at all – even without having to hold them first!

This learning mechanism is not intrinsic to animals, it can be achieved in machines as well, hence the term *Machine Learning*: training/teaching computers to predict (i.e. having them learn) the structure and/or input-output relationships based on previously-observed patterns.

Back to our ball-dropping example, we can reduce any object to a set of summarizing features such as color, mass, and volume. A tennis ball, for example, would instead be known as *"object that is neon-green, 60 grams, and 0.15 liters"*. Our goal could be to train a predictor (or *classifier*) to correctly hypothesize whether an object – known only by its color, mass, and volume – drops like a rock, more like a feather, or floats up like a balloon.

We start by inspecting a number of samples (e.g. objects of various kinds); for each sample, we record the values of its input features (color, mass, and volume) and corresponding output *class* (whether it falls quickly, slowly, or floats). This set of samples is the *training data*, which is then analyzed using a pre-chosen algorithm. The "machine" crunches the data, attempting to extract the underlying input-output relationships. Upon completion, a new classifier is returned; able to hypothesize the class of any object given only its input features.

But how does this work? And which algorithm should we use? There are many to choose from, each resulting in different types of classifiers. In the following sections, we give a more formal perspective of machine learning, briefly overview various existing families of algorithms, and go into more detail on one specific family: the focus of our work, *boosting*.

2.1 Learning to Predict

Algebraically, we can represent any object as a set of d features, just as we did in the example above. Note that in general, an "object" can refer to any form of entity that we are dealing with (e.g. a digital image, bouts of animal behavior, financial trends, etc.; not necessarily a physical object). Accordingly, we abstract any such entity as a d-dimensional vector $\mathbf{x}$, with each dimension encoding the value of some corresponding feature. Note that categorical (i.e. nominal) features can be represented as numerical features by assigning a discrete index to each category. As such, we define the input space $\mathcal{X}$ as the region of the overall space $\mathbb{R}^d$ in which the data is concentrated:

$$\mathbf{x} \in \mathcal{X} \subseteq \mathbb{R}^d$$

Similarly, we can represent any output properties in which we are interested (i.e. observations) as a K-dimensional vector $\mathbf{y}$, and define $\mathcal{Y}$ as the output space[1]:

$$\mathbf{y} \in \mathcal{Y} \subseteq \mathbb{R}^K$$

We assume that there is an underlying "order" to our existence, encoded as the probabilistic distribution P_Ω. By marginalizing over all variables that do not comprise $\mathcal{X}$ or $\mathcal{Y}$ (i.e. ignoring all variables that are not input features or output observations), we define $P_{x,y}$ as the distribution that governs the layout and density of our specific input and output data.

[1] For simplicity, we forego the possibility of structured output [4, 20] in this section.

Our high-level goal is to determine a function $\mathbf{H} : \mathcal{X} \to \mathcal{Y}$ that accurately predicts the output $\mathbf{y}$ given a specific input $\mathbf{x}$, i.e. $\mathbf{H}(\mathbf{x}) \approx \mathbf{y}$. To characterize the performance of this function, we need to define a notion of error. Let err be some appropriate measure of error between a prediction $\tilde{\mathbf{y}} \equiv \mathbf{H}(\mathbf{x})$ and an expected output $\mathbf{y}$. Accordingly, we can define the expected error $\varepsilon_{\mathbb{E}}$:

$$\varepsilon_{\mathbb{E}} \;\equiv\; \mathbb{E}_{x,y}\big\{\, \mathrm{err}\big(\mathbf{H}(\mathbf{x}), \mathbf{y}\big)\big\} \tag{2.1}$$

The lower the expected error, the more accurate the predictor. However, the joint space $\mathcal{X} \times \mathcal{Y}$ (i.e. all possible input-output combinations) may consist of an infinite number of points, each with corresponding probabilities that cannot be explicitly computed. So how do we gauge this error? The best we can do is analyze a finite subset by gathering N points and consolidating them into a dataset $\mathcal{S}$. Collecting data in an unbiased way is equivalent to sampling from the joint distribution $P_{x,y}$:

$$\mathcal{S} \;\equiv\; \{(\mathbf{x}_n, \mathbf{y}_n)\}_{n=1}^{N} \;\sim\; P_{x,y}$$

With this dataset in hand, we can compute the corresponding empirical error $\varepsilon_{\mathcal{S}}$:

$$\varepsilon_{\mathcal{S}} \;\equiv\; \frac{1}{N}\sum_{n=1}^{N} \mathrm{err}\big(\mathbf{H}(\mathbf{x}_n), \mathbf{y}_n\big) \tag{2.2}$$

Throughout this work, we assume that our empirical error is a *good enough* approximation of the expected error: $\varepsilon_{\mathcal{S}} \approx \varepsilon_{\mathbb{E}}$. There is a lot of formal math rigorously defining the notion of "good enough"; however, it is outside the scope of our work. Please refer to [1] for a more comprehensive discussion on the topic.

Unsupervised Learning

In many situations, we only have access to the input points $\{\mathbf{x}_n\}$, not their corresponding outputs $\{\mathbf{y}_n\}$. This could happen if the output observations or class labels are very difficult or impossible to obtain or are simply hidden from us. These situations constitute the *Unsupervised Learning* regime, so-named because no supervision (i.e. labels or expected outputs) are provided.

In this regime, algorithms can aim to cluster the data based on its input statistics (e.g. K-Means and variants [14] and Gaussian Mixture Models [8]), or to reduce the dimensionality of the input data (e.g. Principal Component Analysis (PCA), Linear Discriminant Analysis (LDA), and variants [10]), or to transform the input

data into more meaningful feature representations (e.g. Self-Organizing Maps [16] and t-Distributed Stochastic Neighborhood Encoding (tSNE) [17]), to name a few.

However, throughout this work, we assume that we do have access to fully labeled data, or all of $\mathcal{S}$. Accordingly, our work focuses on the *Supervised Learning* regime. In this regime, we can still carry out any of the methods mentioned above, but we can also perform two other important tasks: regression and classification.

Regression

Back to our falling objects example, given only the color, mass, and volume of an object, we can try to predict the amount of time it takes that object to fall when dropped from a height of 1 m. In this case, our training set $\mathcal{S}$ consists of a scalar-valued output observation y_n, namely, the falling time associated with each object n. A pre-chosen regression method crunches the data, attempting to extract the underlying input-output relationships. Upon completion, a new *regressor* is returned, able to estimate the falling time of any object given only its input feature values.

More concretely, a regressor $\mathbf{H}$ estimates the value of one or more output observations (dependent variables) $\mathbf{y}$ given the input features (independent variables) $\mathbf{x}$:

$$\mathbf{H} : \mathcal{X} \to \mathcal{Y} \qquad \text{such that: } \mathbf{H}(\mathbf{x}) \approx \mathbf{y}$$

To quantify the accuracy of $\mathbf{H}$, we make use an error function err, as in Eq. 2.1. A correct estimate should always incur zero error. In the case of regression, the larger the estimate relative to the expected output, the worse the error should be; similarly, the smaller the estimate relative to the expected output, the worse the error should be. Many different functions achieve this goal. Depending on the specific problem, some may be more suitable than others. Arguably the most intuitive and mathematically-simple function used for regression is the squared L_2 norm between an estimate and its expected output:

$$\mathrm{err}(\tilde{\mathbf{y}}, \mathbf{y}) \equiv \|\tilde{\mathbf{y}} - \mathbf{y}\|^2$$

Classification

Instead of dropping objects in air, what if we wish to predict whether they float or they sink in water? In this case, our training set $\mathcal{S}$ consists of an output class associated with each object: whether it floats or sinks. Predicting one of several classes is aptly known as *classification*. A pre-chosen classification method crunches the data,

attempting to extract the underlying input-output relationships. Upon completion, a new *classifier* is returned, able to hypothesize the class of any object given only its input feature values.

More formally, a classifier F hypothesizes the class label y given input features $\mathbf{x}$:

$$F : \mathcal{X} \to \mathcal{Y} \qquad \text{such that: } F(\mathbf{x}) \approx y$$

In the context of classification, the output class is represented as an integer. For instance, if there are K possible classes, the output space $\mathcal{Y}$ can be defined as:

$$\mathcal{Y} \equiv \{1,2,...,K\}$$

Quantifying the error of a classifier is different than that of a regressor. A class label is a discrete quantity; thus, there is no smooth notion of "almost" – a hypothesis is either right or it is wrong[2]. Accordingly, an apt error function for use in Eq. 2.1 is the misclassification indicator (error = 0 for the right answer, error = 1 for wrong):

$$\text{err}(\tilde{y}, y) \equiv \mathbb{1}(\tilde{y} \neq y)$$

Estimating the Underlying Distributions

Recall our assumption that there exists an intrinsic order, encoded as a distribution $P_{x,y}$ over the joint input-output space $\mathcal{X} \times \mathcal{Y}$. Given a specific input $\mathbf{x}$, the posterior distribution $P_{y|x}$ can be computed using Bayes' theorem:

$$P(y \,|\, \mathbf{x}) \equiv \frac{P(\mathbf{x}, y)}{\sum_{k \in \mathcal{Y}} P(\mathbf{x}, k)}$$

Our goal is to understand the very phenomenon that is being described by these distributions; thus, by definition, we do not know them and so we cannot directly evaluate them. However, if we *were* somehow able to evaluate $P_{y|x}$ for any $\mathbf{x}$ and y, we *could* classify any given point $\mathbf{x}$ as the class corresponding to the highest posterior probability. This (purely theoretical) strategy guarantees optimal classification, yielding the lowest possible expected error, i.e. the *Bayes classifier* F^*:

$$F^*(\mathbf{x}) \equiv \arg\max_{y \in \mathcal{Y}} \{P(y \,|\, \mathbf{x})\} \tag{2.3}$$

[2] Actually, some mistakes may be better (or less severe) than others. This scenario is known as cost-sensitive classification, and is discussed in more detail in Ch. 4.

But as we have just asserted, we cannot directly access $P_{y|x}$ in practice. Fortunately, we can make use of our dataset $\mathcal{S}$ to form an empirical estimate $\tilde{P}$, using this estimate as a basis for a *Maximum A Posteriori* (MAP) classifier F:

$$F(\mathbf{x}) \;\equiv\; \arg\max_{y\in\mathcal{Y}}\{\tilde{P}(y\,|\,\mathbf{x})\} \tag{2.4}$$

Being a finite dataset (i.e. having only N discrete points), we need to generalize over the entire input-output space. To this end, several approaches exist.

Non-parametric methods generalize the labels of discrete data points to their surroundings [21]. Kernel Density Estimation uses a pre-chosen *kernel* to smooth the discrete points in $\mathcal{S}$ so that they cover a much larger range of the problem space. Using K-Nearest Neighbors, the estimated posterior probability of a query point depends on distances from the K closest points in $\mathcal{S}$ and their corresponding labels. Similarly, Random Ferns use an ensemble of random input-space-hashing functions (i.e. *ferns*), each with a corresponding empirical posterior distribution, averaging them to form an overall estimate [18].

Alternatively, parametric models, explicitly defined over the entire space, can be used to estimate the joint distribution (and hence also the posterior distribution). Using Bayesian-style methods (e.g. Naive Bayes), model parameters are optimized to maximize the likelihood of the dataset [13].

With these estimation techniques, the choice of kernel or model can be rather heuristic, especially when little is known about the underlying structure of the problem. Regardless, the act of classification requires making a specific decision. In determining the maximal *a posteriori* class (as in Eq. 2.4), a probability estimate is used only in relative comparisons; its absolute value is unimportant. This fact is particularly apparent when there are few classes.

Binary Classification

In the special case of *binary* classification (i.e. when $K=2$), we are distinguishing between the smallest possible number of distinct classes: two. Accordingly, data is typically split into two, a *positive* and a *negative* set, and the output space $\mathcal{Y}$ is commonly defined as:

$$\mathcal{Y} \;\equiv\; \{\pm 1\}$$

The binary MAP classifier (i.e. the two-class equivalent of Eq. 2.4) reduces to:

$$F(\mathbf{x}) \;\equiv \begin{cases} +1, & \tilde{P}(y{=}+1\,|\,\mathbf{x}) > 1/2 \\ -1, & \tilde{P}(y{=}+1\,|\,\mathbf{x}) \leq 1/2 \end{cases} \;\equiv\; \operatorname{sign}\big(\tilde{P}(y{=}+1\,|\,\mathbf{x}) - 1/2\big) \tag{2.5}$$

Note that the estimate $\tilde{P}_{y|x}$ may itself be inaccurate due to various sources of error: poor model selection, dataset bias, and unaccounted noise, to name a few. Moreover, classification is simply the comparison between the probability and a single fixed threshold (e.g. $1/2$ in Eq. 2.5). Consequently, requiring the estimator to be a valid probabilistic distribution may add unnecessary constraints and complexity.

Estimating Class Boundaries

Instead of applying the indirect procedure of *first* estimating the posterior and *then* classifying using MAP, we can ignore probabilities entirely and focus directly on decision boundaries. After all, this is what any classifier essentially reduces to.

Linear models split the input space into two using a hyperplane (i.e. a comparison between the weighted average of feature values and a threshold). Logistic Regression [6] and Perceptron Learning [12] train (i.e. fine-tune) the hyperplane weights to improve the separation between positive and negative data points.

In many cases, the given input features are not ideal for linear classification. Rather than using these feature directly, we can apply various non-linear transformations to them to generate a new, potentially more suitable *representation* of the data, thereby leading to more expressive classifiers.

Support Vector Machines (SVMs) work with a *kernelization* (i.e. a user-defined high-dimensional non-linear transformation) of the input data [7], maximizing the margin (i.e. minimum distance) between points of differing classes with a hyperplane in the kernel (transformed) space. Instead of requiring a user-defined kernel (i.e. a *hand-coded* feature transform), Neural Networks infer a transform from the data itself. This transform is implemented using multiple layers of interconnected perceptrons (i.e. linear projections followed by a non-linear transform) [15], and is optimized to facilitate more accurate classification.

Using multiple hyperplanes, Decision Trees divide the input space into *leaves* (i.e. non-overlapping regions), each with a specific class label [19]. Assigning a label to a given query point involves a set of comparisons, starting at the root node (i.e. the trunk of the tree) and traversing through the branches – depending on the outcome of each comparison – until finally reaching a leaf node[3].

The methods described above lead to *learners* (i.e. regressors or classifiers) that can function on their own. Some learners are more accurate and some are less

[3] Sec 3.3 describes decision trees in more detail.

so. However, by appropriately combining the predictions of an ensemble of such learners, we can potentially improve (or *boost*) their individual accuracies. This "boosting" procedure is the focus of our work, and is described in greater detail in the following section.

2.2 Boosting

Boosting is a beautifully simple yet effective procedure: given a set of relatively-poor classifiers (or *weak learners*), a *strong* classifier can be induced through a linear combination (i.e. a weighted sum) of their outputs. The basic procedure for boosting is intuitively described as follows:

1. **Do your best**: find (or train) the best weak learner given the original data (i.e. the learner that achieves the lowest training error as in Eq. 2.2).
2. **Check your performance**: based on the accuracy of the cumulative sum of previous learner(s), give more importance to individual samples that have incurred a larger error than to those with smaller error.
3. **Focus on your mistakes**: again, find (or train) a new weak learner, this time, accounting for the updated importance weights of the samples when determining the corresponding error (see Eq. 2.11 below).
4. **Keep on keepin' on**: repeat steps 2 and 3 until some satisfactory stopping condition is met.

One of the earliest and most popular algorithms for binary boosting is AdaBoost [11]. In the following sections, we motivate the AdaBoost algorithm from a practical classification standpoint. AdaBoost can also be derived from a statistically-motivated standpoint; please see Appendix 7.1.

Greedily Minimizing a Surrogate Loss

In the context of binary classification, we are trying to generate a strong classifier $F : \mathcal{X} \to \{\pm 1\}$. Let us assume that we have a set $\mathcal{F}$ of binary weak learners f:

$$\mathcal{F} \subseteq \left\{ f \,\middle|\, f : \mathcal{X} \to \{\pm 1\} \right\}$$

We define a *confidence* function $h : \mathcal{X} \to \mathbb{R}$ as a linear combination of T weak learners; $f_t \in \mathcal{F}$ are the learners and $\alpha_t \in \mathbb{R}$ are their corresponding weights:

$$h(\mathbf{x}) \equiv \sum_{t=1}^{T} \alpha_t f_t(\mathbf{x}) \tag{2.6}$$

The more weak learners individually agree with each others' predictions, the further h is from 0, and the more *confident* the prediction. Our strong classifier F simply predicts the more confident of the two possible labels; equivalent to the sign of h:

$$F(\mathbf{x}) \;\equiv\; \text{sign}\big(h(\mathbf{x})\big) \tag{2.7}$$

With all T weak learners in the mix, the average training error ε is defined as:

$$\varepsilon \;\equiv\; \frac{1}{N}\sum_{n=1}^{N} \mathbb{1}_{\big(F(\mathbf{x}_n)\neq y_n\big)} \tag{2.8}$$

In this form, finding the T optimal $(f_t,\, \alpha_t)$ pairs is very difficult (if not intractable) for two reasons. Firstly, due to the indicator function, the error ε is non-convex in the T pairs (f_t, α_t). Secondly, this optimization requires an exponential number of operations (i.e. on the order of $|\mathcal{F}|^T$, where $|\mathcal{F}|$ is the number of weak learners in the set $\mathcal{F}$; potentially quite large).

To address the first issue, we use a convex *surrogate* loss function $\mathcal{L}$. A valid loss should also upper-bound the training error, guaranteeing that error minimization is an implied consequence of loss minimization. The exponential function e^{-yh} is convex and upper-bounds the misclassification error for an individual point. It is the basis of AdaBoost, resulting in the following surrogate loss:

$$\varepsilon \;\leq\; \mathcal{L} \;\equiv\; \frac{1}{N}\sum_{n=1}^{N} e^{-y_n h(\mathbf{x}_n)} \tag{2.9}$$

Note that: $\mathbb{1}_{\big(\text{sign}(h)\neq y\big)} \leq e^{-yh}$ therefore: $\mathcal{L} = 0 \;\Rightarrow\; \varepsilon = 0$

To address the second issue, we apply a *greedy*, iterative training procedure. Instead of simultaneously optimizing all T learners, we "greedily" consider only the best one at each iteration. To start, we find the single best learner f_1. Holding f_1 fixed, we find the best subsequent learner f_2. Then, holding all previously-chosen learners fixed, we find the next one, and so on. This greedy procedure may be suboptimal due to its stage-wise nature, but it reduces the number of operations down to an order of only $|\mathcal{F}|\cdot T$, a tractable endeavor.

Optimal Weak Learner and Weight

Let's assume that we have just trained the first $I < T$ iterations. Consolidating Eqs. 2.9 and 2.6, the corresponding loss can be explicitly defined as:

$$\mathcal{L}_I \;=\; \frac{1}{N}\sum_{n=1}^{N} \exp\Big(-y_n \sum_{t=1}^{I} \alpha_t f_t(\mathbf{x}_n)\Big)$$

The loss can be expressed as a sum of its current sample-wise constituents w_n, referred to as "sample weights" (not to be confused with weak learner weights α_t):

$$\mathcal{L}_I \;=\; \frac{1}{N}\sum_{n=1}^{N} w_n \qquad \text{where:}\;\; w_n \;\equiv\; \exp\Big(-y_n \sum_{t=1}^{I} \alpha_t f_t(\mathbf{x}_n)\Big)$$

The sample weight w_n summarizes the cumulative performance of previously-chosen weak learners in classifying sample n; the worse the classification, the greater its weight (and vice-versa). Since all previous learners f_t and weights α_t are held fixed due to the greedy training procedure, the next iteration's loss $\mathcal{L}$ is defined as a function of only the *next* weak learner f and corresponding weight α, as follows:

$$\begin{aligned}
\mathcal{L}(f,\alpha) \;&=\; \frac{1}{N}\sum_{n=1}^{N} \exp\Big(-y_n\Big(\sum_{t=1}^{I} \alpha_t f_t(\mathbf{x}_n) \;+\; \alpha f(\mathbf{x}_n)\Big)\Big) \\
&=\; \frac{1}{N}\sum_{n=1}^{N} \underbrace{\exp\Big(-y_n \sum_{t=1}^{I} \alpha_t f_t(\mathbf{x}_n)\Big)}_{w_n} e^{-\alpha\, y_n f(\mathbf{x}_n)} \;=\; \frac{1}{N}\sum_{n=1}^{N} w_n\, e^{-\alpha\, y_n f(\mathbf{x}_n)}
\end{aligned} \tag{2.10}$$

Since class labels and binary weak learner outputs can only be ± 1, therefore:

$$\begin{aligned}
\text{incorrect classification:} \quad & f(\mathbf{x}) \neq y \;\Rightarrow\; e^{-\alpha\, y f(\mathbf{x})} = e^{\alpha} \\
\text{correct classification:} \quad & f(\mathbf{x}) = y \;\Rightarrow\; e^{-\alpha\, y f(\mathbf{x})} = e^{-\alpha}
\end{aligned}$$

Furthermore, given a weak learner f, the relative weighted error ε_f is defined as the (weighted) proportion of samples that are incorrectly classified by f:

$$\varepsilon_f \;\equiv\; \frac{\sum_{n=1}^{N} \mathbb{1}\big(f(\mathbf{x}_n)\neq y_n\big)\, w_n}{\sum_{n=1}^{N} w_n} \;=\; \frac{1}{N\mathcal{L}_I}\sum_{n=1}^{N} \mathbb{1}\big(f(\mathbf{x}_n)\neq y_n\big)\, w_n \tag{2.11}$$

$$\therefore \quad \frac{1}{N}\sum_{n=1}^{N} \mathbb{1}\big(f(\mathbf{x}_n)\neq y_n\big)\, w_n \;=\; \varepsilon_f\, \mathcal{L}_I \qquad \therefore \quad \frac{1}{N}\sum_{n=1}^{N} \mathbb{1}\big(f(\mathbf{x}_n)= y_n\big)\, w_n \;=\; (1-\varepsilon_f)\, \mathcal{L}_I$$

Accordingly, Eq. 2.10 can be expressed as the sum of two terms (*misclassifications* and *correct classifications*):

$$\begin{aligned}
\mathcal{L}(f,\alpha) \;&=\; \frac{e^{\alpha}}{N}\sum_{n=1}^{N} \mathbb{1}\big(f(\mathbf{x}_n)\neq y_n\big)\, w_n \;+\; \frac{e^{-\alpha}}{N}\sum_{n=1}^{N} \mathbb{1}\big(f(\mathbf{x}_n)= y_n\big)\, w_n \\
&=\; \big(\varepsilon_f\, e^{\alpha} \;+\; (1-\varepsilon_f)\, e^{-\alpha}\big)\, \mathcal{L}_I
\end{aligned}$$

Given a weak learner f, the optimal corresponding weight α^* is obtained by setting the derivative to zero and solving:

$$\frac{\partial \mathcal{L}}{\partial \alpha} \;=\; 0 \;=\; \varepsilon_f \, e^{\alpha^*} \;-\; (1-\varepsilon_f) \, e^{-\alpha^*}$$

$$\therefore \;\; e^{2\alpha^*} = \frac{1-\varepsilon_f}{\varepsilon_f} \qquad \therefore \;\; \alpha^* = \frac{1}{2} \ln\Big(\frac{1-\varepsilon_f}{\varepsilon_f}\Big) \tag{2.12}$$

The AdaBoost surrogate loss (as in Eq. 2.9) not only leads to a simple, closed-form solution for the optimal learner weight α^*, but for the resulting loss as well:

$$e^{\alpha^*} \;=\; \sqrt{\frac{1-\varepsilon_f}{\varepsilon_f}}$$

$$\therefore \;\; \mathcal{L}(f) \;=\; \Big(\varepsilon_f \sqrt{\frac{1-\varepsilon_f}{\varepsilon_f}} \;+\; (1-\varepsilon_f) \sqrt{\frac{\varepsilon_f}{1-\varepsilon_f}}\Big) \, \mathcal{L}_I \;=\; 2\sqrt{\varepsilon_f (1-\varepsilon_f)} \, \mathcal{L}_I \tag{2.13}$$

Consequently, the optimal weak learner f^* is the minimizer of the above loss. In practice, f^* is determined by looping through and evaluating Eq. 2.14 for weak learners $f \in \mathcal{F}$, finally recalling the best one.

$$f^* \;=\; \arg\min_{f \in \mathcal{F}} \{\mathcal{L}(f)\} \tag{2.14}$$

Training a Weak Learner

At each iteration, instead of passively finding the best weak learner from a set of predetermined learners $\mathcal{F}$, a weak learner can be specifically trained for use in the current iteration. Accordingly, any of the methods discussed in Sec. 2.1 can be used, appropriately modified to account for the specific sample weights w_n (resulting from the previous boosting iterations).

In particular, shallow decision trees are easily trainable (refer to Sec 3.3) and have proven to be very suitable weak learners in practice, used in a multitude of domains such as computer vision, behavior analysis, and document ranking [9, 5, 2]. Shallow trees are also particularly quick to evaluate, enabling applications that require fast output speeds, especially when combined with *cascades*.

Cascaded Evaluation

Recall that a strong classifier F returns the sign of the confidence function h:

$$F(\mathbf{x}) \;\equiv\; \operatorname{sign}\big(h(\mathbf{x})\big) \qquad \text{where:} \;\; h(\mathbf{x}) \;\equiv\; \sum_{t=1}^{T} f_t(\mathbf{x}) \, \alpha_t$$

Instead of evaluating all T weak learners, the final decision may already be determined (with a high probability or even definitely) after only $I < T$ learners:

$$F(\mathbf{x}) \;\approx\; \operatorname{sign}\big(h_I(\mathbf{x})\big) \qquad \text{where:}\;\; h_I(\mathbf{x}) \;\equiv\; \sum_{t=1}^{I} f_t(\mathbf{x})\,\alpha_t$$

This observation gives rise to the idea of cascaded evaluation: instead of accumulating the confidence of all T learners at once, keep track of the cumulative confidence score. If a lower or upper threshold is reached before all T learners are evaluated, an early classification can be made. Thresholds are typically set by trading-off accuracy and aggressiveness of early termination (e.g. by finding upper and lower thresholds $h_t^{\pm}$ for which 95% of validation samples are correctly classified after only 10% of weak learners have been evaluated). As a result, early classification can be as accurate or aggressive as desired [3].

Accordingly, the following procedure (called soft cascades [3]) can be used to evaluate a boosted classifier:

0. initialize to: $t = 0, \;\; h = 0$
1. increment t and update the confidence score: $\;\; t \leftarrow t{+}1, \;\; h \leftarrow h + f_t(\mathbf{x})\,\alpha_t$
2. if a lower or upper threshold is reached, return with the appropriate label (i.e. if $h \leq h_t^-$ then return $F{=}{-}1$, if $h \geq h_t^+$ then return $F{=}{+}1$)
3. if all T weak learners have been evaluated, return $F{=}\operatorname{sign}(h)$ otherwise, goto step 1.

Note that a simple extension to the above procedure is to repeat step 1 for a total of M times before continuing to step 2, thereby coarsening the cascade.

Boosting Boosting

The combination of simple weak learners and the potential for fast cascaded evaluation makes boosted shallow decision trees a very strong candidate for many machine learning tasks. In this work, we aim to further improve boosting by:

- Proposing a procedure for speeding up the training of decision trees (Ch. 3)
- Proposing a unified method for improved cost-sensitive multi-class boosting (Ch. 4)
- Proposing better weak learners to improve classification boundaries (Ch. 5)

References

[1] Y. S. Abu-Mostafa, M. Magdon-Ismail, and H.-T. Lin. *Learning from data*. AML-Book New York, NY, USA: 2012.

[2] N. Asadi and J. Lin. "Training Efficient Tree-Based Models for Document Ranking". In: *European Conference on Information Retrieval*. 2013.

[3] L. Bourdev and J. Brandt. "Robust object detection via soft cascade". In: *CVPR*. 2005.

[4] S. Branson, O. Beijbom, and S. Belongie. "Efficient Large-Scale Structured Learning". In: *CVPR*. 2013.

[5] X.P. Burgos-Artizzu et al. "Social Behavior Recognition in continuous videos". In: *CVPR*. 2012.

[6] M. Collins, R. E. Schapire, and Y. Singer. "Logistic Regression, AdaBoost and Bregman Distances". In: (2000).

[7] C. Cortes and V. Vapnik. "Support-vector networks". In: *Machine Learning* (1995).

[8] A. P. Dempster, N. M. Laird, and D. B. Rubin. "Maximum likelihood from incomplete data via the EM algorithm". In: *Journal of the royal statistical society* (1977).

[9] P. Dollár, R. Appel, and W. Kienzle. "Crosstalk Cascades for Frame-Rate Pedestrian Detection". In: *ECCV*. 2012.

[10] I. Fodor. *A Survey of Dimension Reduction Techniques*. Tech. rep. 2002.

[11] Y. Freund and R. E. Schapire. "A desicion-theoretic generalization of on-line learning and an application to boosting". In: *European Conference on Computational Learning Theory*. 1995.

[12] Y. Freund and R. E. Schapire. "Large margin classification using the perceptron algorithm". In: *Machine Learning* (1999).

[13] J. H. Friedman, T. Hastie, and R. Tibshirani. *The elements of statistical learning*. Springer, 2001.

[14] T. Kanungo et al. "An efficient k-means clustering algorithm: Analysis and implementation". In: *PAMI* (2002).

[15] Y. LeCun et al. "Backpropagation applied to handwritten zip code recognition". In: *Neural Computation* (1989).

[16] Y. Liu, R. H. Weisberg, and C. N. K. Mooers. "Performance evaluation of the self-organizing map for feature extraction". In: *Journal of Geophysical Research: Oceans* (2006).

[17] L. J. P. van der Maaten and G. E. Hinton. "Visualizing High-Dimensional Data Using t-SNE". In: *Journal of Machine Learning Research* (2008).

[18] M. Ozuysal et al. "Fast keypoint recognition using random ferns". In: *PAMI* (2010).

[19] J. R. Quinlan. "Induction of decision trees". In: *Machine Learning* (1986).

[20] G. Shen, G. Lin, and A. van den Hengel. "StructBoost: Boosting methods for predicting structured output variables". In: *PAMI* (2014).

[21] G. R. Terrell and D. W. Scott. "Variable kernel density estimation". In: *The Annals of Statistics* (1992).

Chapter 3

QUICKLY BOOSTING DECISION TREES

Corresponding paper: [1]

Boosted decision trees are among the most popular learning techniques in use today. While exhibiting fast speeds at test time, relatively slow training renders them impractical for applications with real-time learning requirements. We propose a principled approach to overcome this drawback. We prove a bound on the error of a decision stump given its *preliminary* error on a subset of the training data; the bound may be used to prune unpromising features early in the training process. We propose a fast training algorithm that exploits this bound, yielding speedups of an order of magnitude with no loss in the final performance of the classifier. Our method is not a new variant of boosting; rather, it is used in conjunction with existing boosting algorithms and other sampling methods to achieve even greater speedups.

3.1 Introduction

Boosting is a technique of combining many weak learners to form a single strong one [30, 17, 18]. Shallow decision trees are commonly used as weak learners due to their simplicity and robustness in practice [26, 5, 28, 22]. This powerful combination (of boosting and decision trees) is the learning backbone behind many methods across a variety of domains such as computer vision, behavior analysis, and document ranking to name a few [11, 8, 2], with the added benefit of exhibiting very fast speeds at test time.

Learning speed is important as well. In active or real-time learning situations such as for human-in-the-loop processes or when dealing with data streams, classifiers must learn quickly to be practical. This is our motivation: fast training without sacrificing accuracy. To this end, we propose a principled approach. Our method

offers a speedup of an order of magnitude over prior approaches while maintaining identical performance.

Our contributions are the following:

1. Given the performance on a subset of data, we prove a bound on a stump's classification error, information gain, Gini impurity, and variance.
2. Based on this bound, we propose an algorithm guaranteed to produce identical trees as classical algorithms, and experimentally show it to be one order of magnitude faster for classification tasks.
3. We outline an algorithm for quickly boosting decision trees using our quick tree-training method, applicable to any variant of boosting.

In the following sections, we discuss related work, inspect the tree-boosting process, describe our algorithm, prove our bound, and conclude with experiments on several datasets, demonstrating our gains.

3.2 Related Work

Many variants of boosting [18] have proven to be competitive in terms of prediction accuracy in a variety of applications [7]; however, the slow training speed of boosted trees remains a practical drawback. A large body of literature is devoted to speeding up boosting, mostly categorizable as methods that subsample features or data points and methods that speed up training of the trees themselves.

In many situations, groups of features are highly correlated. By carefully choosing exemplars, an entire set of features can be pruned based on the performance of its exemplar. [12] propose clustering features based on their performances in previous stages of boosting. [21] partition features into many subsets, deciding which to inspect at each stage using adversarial multi-armed bandits. [25] use random projections to reduce the data dimensionality, in essence merging correlated features.

Other approaches subsample the data. In Weight-trimming [20], all samples with weights smaller than a certain threshold are ignored. With Stochastic boosting [19], each weak learner is trained on a random subset of the data. For very large datasets or in the case of on-line learning, elaborate sampling methods have been proposed, e.g. Hoeffding trees [15] and Filter Boost [14, 4]. To this end, probabilistic bounds can be computed on the error rates given the number of samples used [24]. More recently, Laminating [16] trades off number of features for number of samples considered as training progresses, enabling constant-time boosting.

Although all these methods can be made to work in practice, they provide no performance guarantees.

A third line of work focuses on speeding up training of decision trees. Building upon the C4.5 tree-training algorithm of [27], using shallow (depth-D) trees and quantizing features values into $B \ll N$ bins leads to an $O(D{\times}d{\times}N)$ implementation where d is the number of features, and N the number of samples [34, 31].

Orthogonal to all of the above methods is the use of parallelization; multiple cores or GPUs. Recently, [32] distributed computation over cluster nodes for the application of ranking; however, reporting lower accuracies as a result. GPU implementations of GentleBoost exist for object detection [9] and for medical imaging using Probabilistic Boosting Trees (PBT) [3]. Although these methods offer speedups in their own right, we focus on the single-core paradigm.

Regardless of the subsampling heuristic used, once a subset of features or data points is obtained, weak learners are trained on that subset in its entirety. Consequently, each of the aforementioned strategies can be viewed as a two-stage process; in the first, a smaller set of features or data points is collected, and in the second, decision trees are trained on that entire subset.

We propose a method for speeding up this second stage; thus, our approach can be used in conjunction with all the prior work mentioned above for even greater speedup. Unlike the aforementioned methods, our approach provides a performance guarantee: highly sped-up training with identical performance as classical training.

3.3 Boosting Trees

A boosted classifier (or regressor) having the form $F(\mathbf{x}) \equiv \text{sign}\big(\sum_t \alpha_t f_t(\mathbf{x})\big)$ can be trained by greedily minimizing a loss function $\mathcal{L}$; i.e. by optimizing *weak learner* f_t and corresponding weight α_t at each iteration t. Before training begins, each data sample n is assigned a non-negative weight w_n (as discussed in Sec. 2.2). After each iteration, misclassified samples are weighted more heavily, thereby increasing the severity of misclassifying them in following iterations. Regardless of the flavor of boosting used (i.e. AdaBoost, LogitBoost, L_2Boost, etc.), each iteration requires training a new weak learner given the sample weights. We focus on the case when the weak learners are shallow trees.

Training Decision Trees

In the context of classification, a binary decision tree $f_{\text{TREE}}(\mathbf{x})$ processes an input $\mathbf{x}$ and outputs a class label (either $+1$ or -1). Internally, the tree is composed of a decision stump (i.e. a two-way split) $s_j(\mathbf{x})$ at every non-leaf node j. Each stump corresponds to a binary decision, parametrized with a polarity $p \in \{\pm 1\}$, a threshold $\tau \in \mathbb{R}$, and a feature index $k \in \{1,2,...,d\}$, and expects an input $\mathbf{x} \in \mathbb{R}^d$:

$$s_j(\mathbf{x}) \equiv p_j \operatorname{sign}\big(x_{k_j} - \tau_j\big) \qquad \text{where: } x_{k_j} \equiv \langle \mathbf{x}, \boldsymbol{\delta}_{k_j} \rangle$$

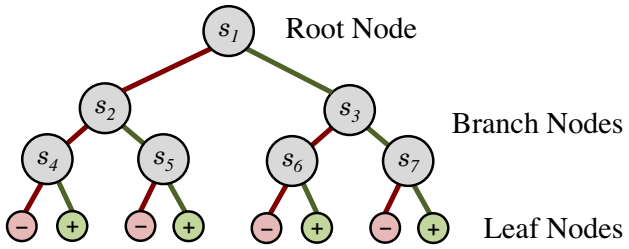


Figure 3.1: A shallow (depth-3) binary tree. Each non-leaf node corresponds to a binary decision stump $s_j(\mathbf{x})$. The output of a stump determines which stump is evaluated next: an output of $+1$ means the next stump is to the right, otherwise the next stump is to the left. Upon reaching a leaf node, the output of the tree corresponds to the output of the last stump to be evaluated.

Trees are commonly grown using a greedy procedure as described in [6, 27], recursively setting one stump at a time, starting at the *root* node (i.e. the trunk) and working through to the extremities (i.e. the pre-leaf nodes). In this work, we focus on classification error; for other split criteria (e.g. information gain, Gini impurity, or variance), refer to Appendix 7.2 for similar derivations.

In the context of classification, the goal in each stage of stump-training is to find the optimal parameters that minimize ε, the average weighted classification error:

$$\varepsilon \equiv \frac{1}{Z} \sum_{n=1}^{N} \mathbb{1}_{(s_j(\mathbf{x}_n) \neq y_n)} \, w_n \qquad \text{where: } Z \equiv \sum_{n=1}^{N} w_n \tag{3.1}$$

Note that while the root node evaluates every input datapoint, stumps further down the precessing chain only evaluate subsets of the data, based on the results of earlier-evaluated stumps in the tree. For example, in Fig. 3.1, stump s_6 only evaluates points n such that: $s_1(\mathbf{x}_n) = +1$ and $s_3(\mathbf{x}_n) = -1$. From hereon-in, we implicitly assume that when training stump s_j, all points n that do not reach s_j have $w_n = 0$.

Consequently, in training a decision stump (specifically on the k^{th} feature of the data), we can expand the weighted classification error (Eq. 3.1) as:

$$\varepsilon_k \;=\; \frac{1}{Z}\Big(\sum_{n=1}^{N} \mathbb{1}\big(x_{nk} \leq \tau \;\wedge\; y_n{=}{+}p\big)\, w_n \;+\; \sum_{n=1}^{N} \mathbb{1}\big(x_{nk} > \tau \;\wedge\; y_n{=}{-}p\big)\, w_n\Big)$$

In practice, this error is minimized by finding k^*; the best single feature of them all:

$$(p^*, \tau^*, k^*) \;\equiv\; \arg\min_{p,\tau,k} \varepsilon_k \qquad \varepsilon^* \;\equiv\; \varepsilon_{k^*}$$

In current implementations of boosting [31], feature values can first be quantized into B bins by linearly distributing them in $[1, B]$ (outer bins corresponding to the min/max, or to a fixed number of standard deviations from the mean), or by any other quantization method. Not surprisingly, using too few bins reduces threshold precision and hence overall performance. We find that $B = 256$ is large enough to incur no loss in practice.

Determining the optimal threshold τ^* requires accumulating each sample's weight into discrete bins corresponding to that sample's feature value x_{nk}. This procedure turns out to still be quite costly: for each of the d features, the weights of each of the N samples have to be accumulated in bins, causing the training of a single stump to be an $O(d{\times}N)$ operation; the very bottleneck of training boosted decision trees.

In the following sections, we examine the training process in greater detail and develop an intuition for how we can reduce computational costs.

Progressively Increasing Subsets

Let us assume that at the start of each boosting iteration, the data samples are sorted in order of decreasing weight, i.e: $r < m \;\Rightarrow\; w_r \geq w_m$. Consequently, we define Z_m, the mass of the *heaviest* subset of m data points:

$$Z_m \;\equiv\; \sum_{n=1}^{m} w_n \qquad \text{[note: } Z_N \;\equiv\; Z\text{]}$$

Clearly, Z_m is greater or equal to the sum of any m other sample weights. As we increase m, the m-subset includes more samples, and accordingly, its mass Z_m increases (although at a diminishing rate).

In Figure 3.2, we plot Z_m/Z for progressively increasing m-subsets, averaged over multiple iterations. An interesting empirical observation can be made about boosting: a large fraction of the overall weight is accounted for by just a few samples.

Since training time is dependent on the number of samples and not on their cumulative weight, we should be able to leverage this fact and train using only a subset of the data. The more non-uniform the weight distribution, the more we can leverage; boosting is an ideal situation where few samples account for most of the weight.

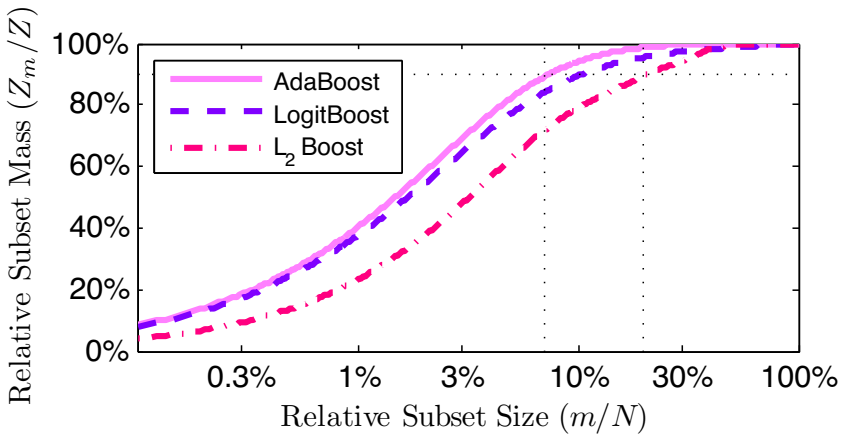


Figure 3.2: Progressively increasing m-subset mass using different variants of boosting. Relative subset mass Z_m/Z exceeds 90% after only $m/N \approx 7\%$ for AdaBoost and $m/N \approx 20\%$ for L_2Boost.

The same observation was made by [20], giving rise to the idea of *weight-trimming*, which proceeds as follows. At the start of each boosting iteration, samples are sorted in order of weight (from largest to smallest). For that iteration, only the samples belonging to the smallest m-subset such that $Z_m/Z \geq \eta$ are used for training (where η is some predefined threshold); all the other samples are temporarily ignored – or *trimmed*. Friedman et al. claimed this to *"dramatically reduce computation for boosted models without sacrificing accuracy."* In particular, they prescribed $\eta = 90\%$ to 99% *"typically"*, but they left open the question of how to choose η from the statistics of the data [20].

Their work raises an interesting question: *Is there a principled way to determine (and train on only) the smallest subset after which including more samples does not alter the final stump?* Indeed, we prove that a relatively small m-subset contains enough information to set the optimal stump; thereby saving a lot of computation.

Preliminary Errors

Definition: given a feature k and an m-subset, the **best preliminary error** $\varepsilon_k^{(m)}$ is the lowest achievable training error if only the data points in that subset are considered. This is equivalent to the error if all samples not in that m-subset are *trimmed*.

$$\varepsilon_k^{(m)} \equiv \frac{1}{Z_m}\Big(\sum_{n=1}^{m} \mathbb{1}\big(x_{nk} \le \tau_k^{(m)} \wedge y_n = +p_k^{(m)}\big)\, w_n \;+\; \sum_{n=1}^{m} \mathbb{1}\big(x_{nk} > \tau_k^{(m)} \wedge y_n = -p_k^{(m)}\big)\, w_n\Big)$$

[where $p_k^{(m)}$ and $\tau_k^{(m)}$ are optimal preliminary parameters]

The emphasis on *preliminary* indicates that the subset does not contain all data points, i.e: $m < N$, and the emphasis on *best* indicates that no choice of polarity p or threshold τ can lead to a lower preliminary error using that feature.

As described above, given a feature k, a stump is trained by accumulating sample weights into bins. We can view this as a progression; initially, only a few samples are binned, and as training continues, more and more samples are accounted for. As revealed by the smoothness of the traces, $\varepsilon_k^{(m)}$ may be computed incrementally.

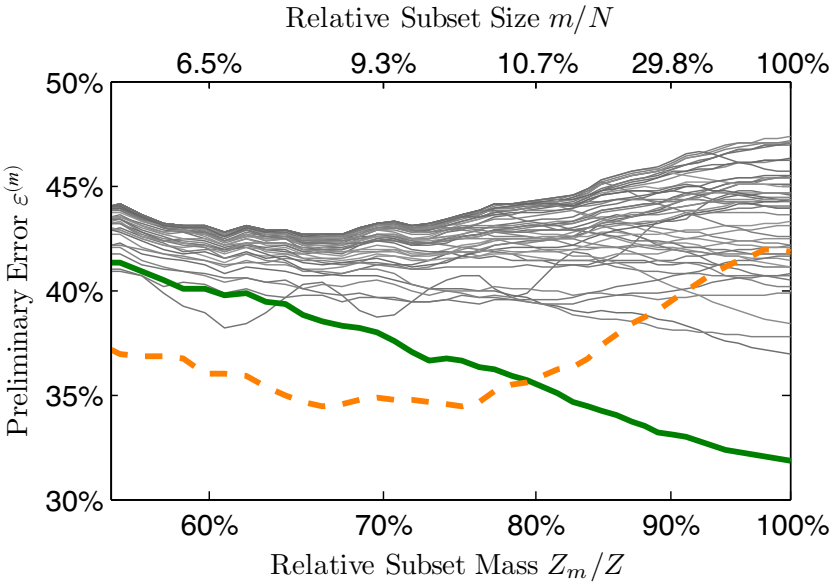


Figure 3.3: Traces of the best preliminary errors over increasing Z_m/Z. Each curve corresponds to a different feature k. The dashed orange curve corresponds to a feature that turns out to be misleading (i.e. its final error drastically worsens when all the sample weights are accumulated). The thick green curve corresponds to the optimal feature; note that it is among the best performing features even when training with relatively few samples.

Figure 3.3 shows the best preliminary error for each of the features in a typical experiment. By examining Figure 3.3, we can make four observations:

1. Most of the overall weight is accounted for by the first few samples. (This can be seen by comparing the top and bottom axes in the figure)

2. Feature errors increasingly stabilize as $m \to N$
3. The best performing features at smaller m-subsets (i.e. $Z_m/Z < 80\%$) can end up performing quite poorly once all of the weights are accumulated (dashed orange curve)
4. The optimal feature is among the best features for smaller m-subsets as well (solid green curve)

From the full training run shown in Figure 3.3, we note (in retrospect) that using an m-subset with $m/N \approx 0.2$ would suffice in determining the optimal feature. But how can we know *a priori* which m is good enough?

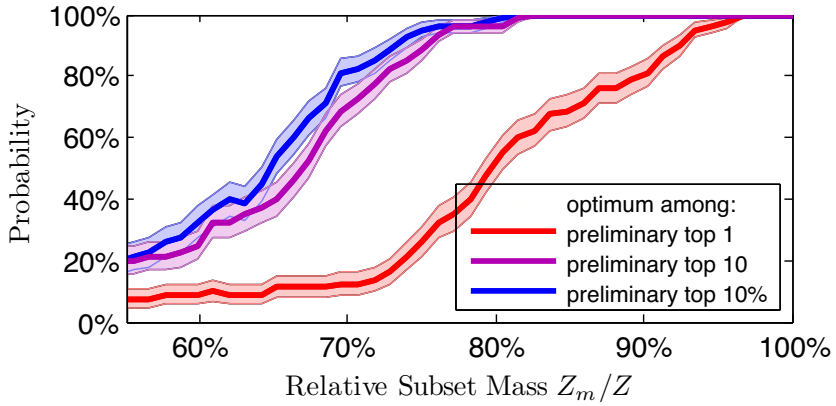


Figure 3.4: Probability that the optimal feature is among the $\mathcal{K}$ top-performing features. The red curve corresponds to $\mathcal{K} = 1$, the purple to $\mathcal{K} = 10$, and the blue to $\mathcal{K} = 1\%$ of all features. Note the knee-point around $Z_m/Z \approx 75\%$ at which the optimal feature is among the 10 preliminary-best features over 95% of the time.

Averaging over many training iterations, in Figure 3.4, we plot the probability that the optimal feature is among the top-performing features when trained on only an m-subset of the data. This gives us an idea as to how small m can be while still correctly predicting the optimal feature.

From Figure 3.4, we see that the optimal feature is not amongst the top performing features until a large enough m-subset is used – in this case, $Z_m/Z \approx 75\%$. Although *"optimality"* is not quite appropriate to use in the context of greedy stage-wise procedures (such as boosted trees), consistently choosing sub-optimal parameters at each stump empirically leads to substantially poorer performance, and should be avoided. In the following section, we outline our approach, which determines the *optimal* stump parameters using the smallest possible m-subset.

3.4 Pruning Underachieving Features

Figure 3.4 suggests that the optimal feature can often be estimated at a fraction of the computational cost using the following heuristic:

Faulty Stump Training
1. Train each feature only with samples in the m-subset where $Z_m/Z \approx 75\%$.
2. Prune all but the 10 best performing features.
3. For each of un-pruned feature, complete training on the entire data set.
4. Finally, report the best performing feature (and corresponding parameters).

This heuristic does not guarantee to return the optimal feature, since premature pruning can occur in step 2. However, if we were somehow able to bound the error, we would be able to prune features that would provably underachieve (i.e. would no longer have any chance of being optimal in the end).

Definition: a feature k is denoted **underachieving** if it is guaranteed to perform worse than the best-so-far feature k° on the entire training data.

Proposition 3.2: for a feature k, the following bound holds (proof given in Section 3.4): given two subsets (where one is larger than the other), the product of subset mass and preliminary error is always greater for the larger subset:

$$r \leq m \quad \Rightarrow \quad Z_r \, \varepsilon_k^{(r)} \leq Z_m \, \varepsilon_k^{(m)} \tag{3.2}$$

Let us assume that the best-so-far error ε° has been determined over a few of the features (and the parameters that led to this error have been stored). Hence, this is an upper-bound for the error of the stump currently being trained. For the next feature in the queue, even after a smaller ($m < N$)-subset, then:

$$Z_m \, \varepsilon_k^{(m)} \geq Z \, \varepsilon^\circ \quad \Rightarrow \quad Z \, \varepsilon_k \geq Z \, \varepsilon^\circ \quad \Rightarrow \quad \varepsilon_k \geq \varepsilon^\circ$$

Therefore, if: $Z_m \, \varepsilon_k^{(m)} \geq Z \, \varepsilon^\circ$ then feature k is *underachieving* and can safely be pruned. Note that the lower the best-so-far error ε°, the harsher the bound; consequently, it is desirable to train a relatively low-error feature early on.

Accordingly, we propose a new method based on comparing feature performance on subsets of data, and consequently pruning underachieving features:

Quick Stump Training

1. Train each feature only using data in a *relatively small* m-subset.
2. Sort the features based on their preliminary errors (from best to worst).
3. Continue training one feature at a time on progressively larger subsets, updating $\varepsilon_k^{(m)}$ after accumulating the samples in each subset.
   - if it is underachieving, prune immediately.
   - if it trains to completion, save it as best-so-far.
4. Finally, report the best performing feature (and corresponding parameters).

Subset Scheduling

Deciding which schedule of m-subsets to use is a subtlety that requires further explanation. Although this choice does not effect the optimality of the trained stump, it may effect speedup. If the first *"relatively small"* m-subset (as prescribed in step 1) is too small, we may lose out on low-error features leading to less-harsh pruning. If it is too large, we may be doing unnecessary computation. Furthermore, since the calculation of preliminary error does incur some (albeit, low) computational cost, it is impractical to use every m when training on progressively larger subsets.

To address this, we implement a simple schedule: The first m-subset is determined by the parameter η_{kp} such that $Z_m/Z \approx \eta_{\text{kp}}$. M following subsets are equally spaced out between η_{kp} and 1. Figure 3.5 shows a parameter sweep over η_{kp} and M, from which we fix $\eta_{\text{kp}} = 90\%$ and $M = 20$ and use this setting for all of our experiments.

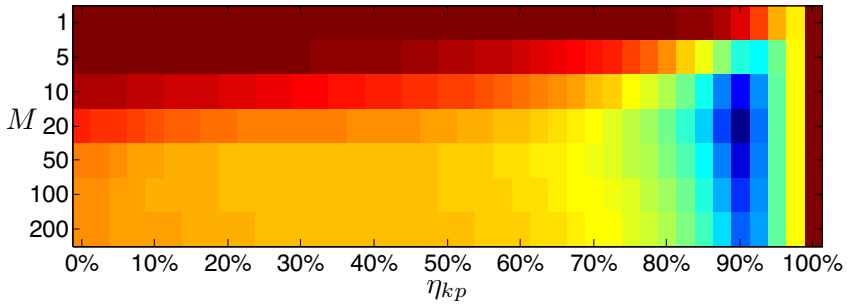


Figure 3.5: Computational cost of training boosted trees over a range of η_{kp} and M, averaged over several types of runs (with varying numbers and depths of trees). Red corresponds to higher cost, blue to lower cost. All final classifiers are identical. Standard training corresponds to $\eta_{\text{kp}} = 100\%$, resulting in the highest computational cost. The lowest computational cost is achieved at $\eta_{\text{kp}} = 90\%$ and $M = 20$.

Using our quick stump training procedure, we determine the optimal parameters without having to consider every sample for each feature. By pruning underachiev-

ing features, a lot of computation is saved. We now outline the full boosting procedure using our quick training method:

Quickly Boosting Decision Trees

1. Initialize weights (sorted in decreasing order).
2. Train decision tree f_t (one node at a time) using **Quick Stump Training**.
3. Perform standard boosting steps:
   (**a**) determine optimal α_t (e.g. closed-form optimum or using line-search).
   (**b**) update sample weights given the misclassification error of f_t, according to the specific variant of boosting being used.
   (**c**) if more boosting iterations are needed, sort sample weights in decreasing order, increment iteration number t; goto step 2.

We note that sorting the weights in step 3(c) above is an $O(N)$ operation. Given an initially sorted set, boosting updates the sample weights based on whether the samples were correctly classified or not. All correctly classified samples are weighted down, but they maintain their respective ordering. Similarly, all misclassified samples are weighted up, also maintaining their respective ordering. Finally, these two sorted lists are merged in $O(N)$.

We now give a proof for the bound that our method is based on, and in the following section, we demonstrate its effectiveness in practice.

Proof of Proposition 3.2

As previously defined, $\varepsilon_k^{(m)}$ is the preliminary weighted classification error computed using the feature k on samples in the m-subset; thus:

$$Z_m\, \varepsilon_k^{(m)} \;=\; \sum_{n=1}^{m} \mathbb{1}\big(x_{nk} \le \tau_k^{(m)} \wedge y_n = +p_k^{(m)}\big)\, w_n \;+\; \sum_{n=1}^{m} \mathbb{1}\big(x_{nk} > \tau_k^{(m)} \wedge y_n = -p_k^{(m)}\big)\, w_n$$

Proposition 3.2: $\quad r \le m \quad \Rightarrow \quad Z_r\, \varepsilon_k^{(r)} \;\le\; Z_m\, \varepsilon_k^{(m)}$

Proof: $\varepsilon_k^{(r)}$ is the best achievable preliminary error on the r-subset (and correspondingly, $(p_k^{(r)}, \tau_k^{(r)})$ are the best preliminary parameters); therefore:

$$Z_r\, \varepsilon_k^{(r)} \;\le\; \sum_{n=1}^{r} \mathbb{1}\big(x_{nk} \le \tau \wedge y_n = +p\big)\, w_n \;+\; \sum_{n=1}^{r} \mathbb{1}\big(x_{nk} > \tau \wedge y_n = -p\big)\, w_n \quad \forall\, p, \tau$$

Hence, switching the optimal parameters $(p_k^{(r)}, \tau_k^{(r)})$ for potentially sub-optimal ones $(p_k^{(m)}, \tau_k^{(m)})$ (note the subtle change in indices on the right side of the inequality):

$$Z_r\, \varepsilon_k^{(r)} \;\leq\; \sum_{n=1}^{r} \mathbb{1}\big(x_{nk} \leq \tau_k^{(m)} \wedge y_n = +p_k^{(m)}\big)\, w_n \;+\; \sum_{n=1}^{r} \mathbb{1}\big(x_{nk} > \tau_k^{(m)} \wedge y_n = -p_k^{(m)}\big)\, w_n$$

The resulting sum can only increase when summing over a larger subset ($m \geq r$):

$$Z_r\, \varepsilon_k^{(r)} \;\leq\; \sum_{n=1}^{m} \mathbb{1}\big(x_{nk} \leq \tau_k^{(m)} \wedge y_n = +p_k^{(m)}\big)\, w_n \;+\; \sum_{n=1}^{m} \mathbb{1}\big(x_{nk} > \tau_k^{(m)} \wedge y_n = -p_k^{(m)}\big)\, w_n$$

But the right-hand side of the inequality is equivalent to $Z_m\, \varepsilon_k^{(m)}$; thus:

$$r \leq m \quad \Rightarrow \quad Z_r\, \varepsilon_k^{(r)} \;\leq\; Z_m\, \varepsilon_k^{(m)}$$

Q.E.D.

For similar proofs using information gain, Gini impurity, or variance minimization as split criteria, refer to Appendix 7.2.

3.5 Experiments

In the previous section, we proposed an efficient stump training algorithm and showed that it has a lower expected computational cost than the traditional method. In this section, we describe experiments that are designed to assess whether the method is practical and whether it delivers significant training speedup. We train and test on three real-world datasets and empirically compare the speedups.

Datasets

We trained AdaBoosted ensembles of shallow decision trees of various depths on the following three datasets:

1. CMU-MIT Faces dataset [29]; $8.5 \cdot 10^3$ training and $4.0 \cdot 10^3$ test samples, $4.3 \cdot 10^3$ features used are the result of convolutions with Haar-like wavelets [33], using 2000 stumps as in [33].

2. INRIA Pedestrian dataset [10]; $1.7 \cdot 10^4$ training and $1.1 \cdot 10^4$ test samples, $5.1 \cdot 10^3$ features used are Integral Channel Features [13]. The classifier has 4000 depth-2 trees as in [13].

3. MNIST Digits [23]; $6.0 \cdot 10^4$ training and $1.0 \cdot 10^4$ test samples, $7.8 \cdot 10^2$ features used are grayscale pixel values. The ten-class classifier uses 1000 depth-4 trees based on [21].

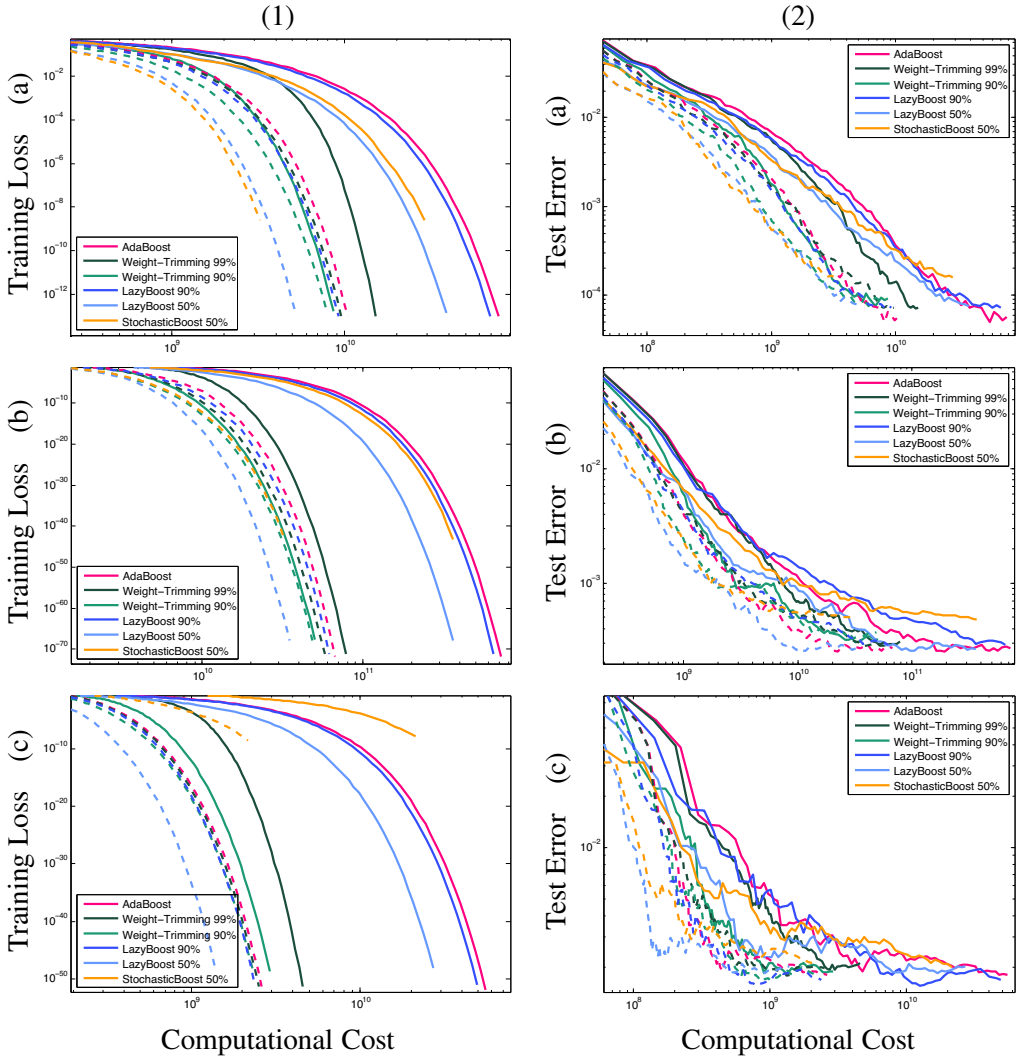


Figure 3.6: Computational cost versus training loss (left plots) and versus test error (right plots) for the various heuristics on three datasets: ($a_{1,2}$) CMU-MIT Faces, ($b_{1,2}$) INRIA Pedestrians, and ($c_{1,2}$) MNIST Digits [see text for details]. Dashed lines correspond to the "quick" versions of the heuristics (using our proposed method) and solid lines correspond to the original heuristics. Test error is defined as the area over the ROC curve.

Comparisons

Quick Boosting can be used in conjunction with all previously mentioned heuristics to provide further gains in training speed. We report all computational costs in units proportional to Flops, since running time (in seconds) is dependent on compiler optimizations which are beyond the scope of this work.

In Figure 3.6, we plot the computation cost versus training loss and versus test error.

We compare *vanilla* (no heuristics) AdaBoost, Weight-Trimming with η = 90% and 99% [see Sec. 3.3], LazyBoost 90% and 50% (only 90% or 50% randomly selected features are used to train each weak learner), and StochasticBoost (only a 50% random subset of the samples are used to train each weak learner). To these six heuristics, we apply our method to produce six "quick" versions.

We further note that our goal in these experiments is not to tweak and enhance the performance of the classifiers, but to compare the performance of the heuristics with and without our proposed method.

Results and Discussion

From Figure 3.6, we make several observations. "Quick" versions require less computational costs (and produce identical classifiers) as their *slow* counterparts. From the training loss plots ($5a_1$, $5b_1$, $5c_1$), we gauge the speed-up offered by our method, often around an order of magnitude. Quick-LazyBoost-50% and Quick-Stochastic-Boost-50% are the least computationally-intensive heuristics, and *vanilla* AdaBoost always achieves the smallest training loss and attains the lowest test error in two of the three datasets.

The motivation behind this work was to speed up training such that (i) for a fixed computational budget, the best possible classifier could be trained, and (ii) given a desired performance, a classifier could be trained with the least computational cost.

For each dataset, we find the lowest-cost heuristic and set that computational cost as our budget. We then boost as many weak learners as our budget permits for each of the heuristics (with and without our method) and compare the test errors achieved, plotting the relative gains in Figure 3.7. For most of the heuristics, there is a two to eight-fold reduction in test error, whereas for weight-trimming, we see less of a benefit. In fact, for the second dataset, Weight-Trimming-90% runs at the same cost with and without our speedup.

Conversely, in Figure 3.8, we compare how much less computation is required to achieve the best test error rate by using our method for each heuristic. Most heuristics see an eight to sixteen-fold reduction in computational cost, whereas for weight-trimming, there is still a speedup, albeit only between one and two-fold.

As discussed in Sec. 3.3, weight-trimming is similar to our method in that it prunes features, although it does so naively - without adhering to a provable bound. This results in a speed-up (at times almost equivalent to our own), but also leads to classifiers that do not perform as well as those trained using the other heuristics.

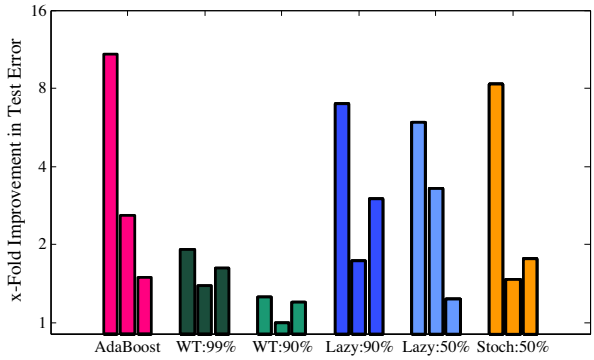


Figure 3.7: Relative *(x-fold)* reduction in test error (area over the ROC) due to our method, given a fixed computational cost. Bar triplets of the same color correspond to the three datasets: CMU-MIT Faces, INRIA Pedestrians, and MNIST Digits.

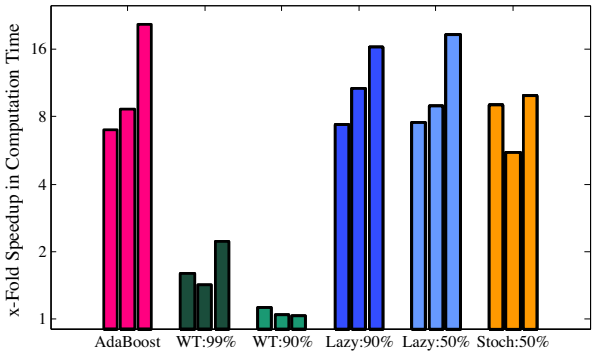


Figure 3.8: Relative speedup in training time due to our proposed method to achieve the desired minimal error rate. Bar triplets of the same color correspond to the three datasets: CMU-MIT Faces, INRIA Pedestrians, and MNIST Digits.

3.6 Conclusions

We presented a principled approach for speeding up training of boosted decision trees. Our approach is built on a novel bound on classification or regression error, guaranteeing that gains in speed do not come at a loss in classification performance.

Experiments show that our method is able to reduce training cost by an order of magnitude or more, or given a computational budget, is able to train classifiers that reduce errors on average by two-fold or more.

Our ideas may be applied concurrently with other techniques for speeding up boosting (e.g. subsampling of large datasets) and do not limit the generality of the method, enabling boosting for applications where fast classifier training is key.

References

[1] R. Appel et al. “Quickly boosting decision trees-pruning underachieving features early”. In: *ICML*. 2013.

[2] N. Asadi and J. Lin. “Training Efficient Tree-Based Models for Document Ranking”. In: *European Conference on Information Retrieval*. 2013.

[3] N. Birkbeck, M. Sofka, and S. K. Zhou. “Fast boosting trees for classification, pose detection, and boundary detection on a GPU”. In: *CVPR Workshops*. 2011.

[4] J. K. Bradley and R. E. Schapire. “FilterBoost: Regression and Classification on Large Datasets”. In: *NIPS*. 2007.

[5] L. Breiman. “Arcing classifiers”. In: *The Annals of Statistics*. 1998.

[6] L. Breiman et al. *Classification and Regression Trees*. Chapman & Hall, New York, NY, 1984.

[7] P. Bühlmann and T. Hothorn. “Boosting Algorithms: Regularization, Prediction and Model Fitting”. In: *Statistical Science*. 2007.

[8] X.P. Burgos-Artizzu et al. “Social Behavior Recognition in continuous videos”. In: *CVPR*. 2012.

[9] A. Coates et al. “Scalable learning for object detection with GPU hardware”. In: *IROS*. 2009.

[10] N. Dalal and B. Triggs. “Histograms of Oriented Gradients for Human Detection”. In: *CVPR*. 2005.

[11] P. Dollár, R. Appel, and W. Kienzle. “Crosstalk Cascades for Frame-Rate Pedestrian Detection”. In: *ECCV*. 2012.

[12] P. Dollár et al. “Feature Mining For Image Classification”. In: *CVPR*. 2007.

[13] P. Dollár et al. “Integral Channel Features”. In: *BMVC*. 2009.

[14] C. Domingo and O. Watanabe. “Scaling up a Boosting-Based Learner via Adaptive Sampling”. In: *PAC-KDD*. 2000.

[15] P. Domingos and G. Hulten. “Mining high-speed data streams”. In: *SIGKDD*. 2000.

[16] C. Dubout and F. Fleuret. “Boosting with Maximum Adaptive Sampling”. In: *NIPS*. 2011.

[17] Y. Freund. “Boosting a weak learning algorithm by majority”. In: *Information and Computation* (1995).

[18] Y. Freund and R. E. Schapire. “Experiments with a new boosting algorithm”. In: *Machine Learning International Workshop*. 1996.

[19] J. H. Friedman. “Stochastic gradient boosting”. In: (2002).

[20] J. H. Friedman, T. Hastie, and R. Tibshirani. “Additive logistic regression: a statistical view of boosting”. In: *The Annals of Statistics*. 2000.

[21] B. Kégl and R. Busa-Fekete. “Accelerating AdaBoost using UCB”. In: *KDD-Cup* (2009).

[22] S. B. Kotsiantis. “Supervised Machine Learning: A Review of Classification Techniques. Informatica 31:249–268”. In: *Informatica* (2007).

[23] Y. LeCun and C. Cortes. *The MNIST database of handwritten digits*. Tech. rep. 1998.

[24] V. Mnih, C. Szepesvári, and J. Audibert. “Empirical Bernstein stopping”. In: *ICML*. 2008.

[25] B. Paul, G. Athithan, and M. N. Murty. “Speeding up AdaBoost Classifier with Random Projection”. In: *ICAPR*. 2009.

[26] J. R. Quinlan. “Bagging, Boosting, and C4.5”. In: *National Conference on Artificial Intelligence*. 1996.

[27] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[28] G. Ridgeway. “The State of Boosting”. In: *Computing Science and Statistics* (1999).

[29] H. Rowley, S. Baluja, and T. Kanade. “Neural Network-Based Face Detection”. In: *CVPR*. 1996.

[30] R. E. Schapire. “The strength of weak learnability”. In: *Machine Learning* (1990).

[31] T. Sharp. “Implementing Decision Trees and Forests on a GPU”. In: *ECCV*. 2008.

[32] K. M. Svore and C. J. Burges. “Large-scale learning to rank using boosted decision trees”. In: *Scaling Up Machine Learning: Parallel and Distributed Approaches* (2011).

[33] P. Viola and M. J. Jones. “Robust Real-Time Face Detection”. In: *IJCV* (2004).

[34] J. Wu et al. “Fast Asymmetric Learning for Cascade Face Detection”. In: *PAMI* (2008).

Chapter 4

COST-SENSITIVE MULTI-CLASS BOOSTING

Corresponding paper: [2]

We present a simple unified framework for multi-class cost-sensitive boosting. The minimum-risk class is estimated directly, rather than via an approximation of the posterior distribution. Our method jointly optimizes binary weak learners and their corresponding output vectors, requiring classes to share features at each iteration. By training in a cost-sensitive manner, weak learners are invested in separating classes whose discrimination is important, at the expense of less relevant classification boundaries. Additional contributions are a family of loss functions along with a proof that our algorithm is boostable in the theoretical sense, as well as an efficient procedure for growing decision trees for use as weak learners. We evaluate our method on a variety of datasets: a collection of synthetic planar data, common UCI datasets, MNIST digits, SUN scenes, and CUB-200 birds. Results show state-of-the-art performance across all datasets against several strong baselines, including non-boosting multi-class approaches.

4.1 Introduction

Boosted classifiers are easy and quick to train [3], and yield the fastest classifiers when computed as a cascade [7], enabling real-time applications such as object detection and animal tracking [41, 17, 26].

In the case of binary classification, boosting is well understood [35, 21, 22]. From a statistical standpoint, boosting can be seen as iteratively improving an estimator of the underlying posterior distribution of the data [23]; refer to Appendix 7.1 for details. However, the multi-class case is not yet as well understood. Generalizing binary boosting to multi-class often requires various heuristics in order to work (see Sec. 4.2). Moreover, different errors may incur different costs, with such situations

arising in a wide variety of domains such as computer vision, behavior analysis, and medical diagnosis, to name a few [15, 10, 33].

An interesting case in point is the categorization of objects naturally organized into a taxonomy, such as birds [8]. Misclassifying taxonomically close categories (eg. confusing two types of duck) is usually more forgivable than distant categories (eg. mistaking a vulture for a duck) [14]. Even in cases without inherent taxonomies, different errors may have different severities. At a security checkpoint, letting a knife through is worse than a water bottle. Accordingly, classifiers should be trained to minimize classification cost rather than just error rates.

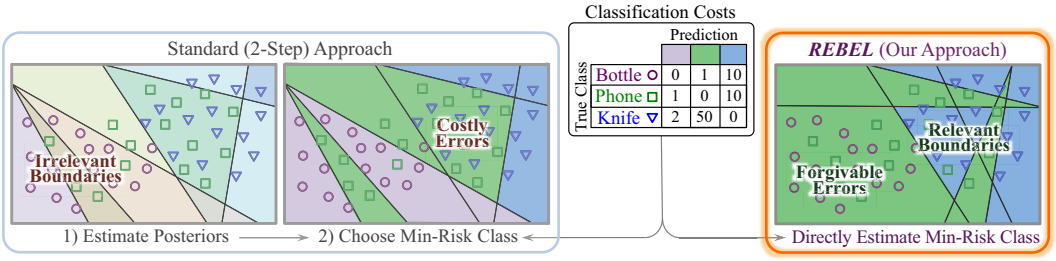


Figure 4.1: Example scenario of an airport security checkpoint with three classes and corresponding classification costs. With standard training, cost-sensitive classification is a two-step (potentially inefficient) process. Weak learners may be *wasted* on irrelevant (low-cost) boundaries. Using cost-sensitive training (e.g. REBEL), relevant boundaries are set first, giving less weight to the distinction between classes with forgivable errors.

We present a novel approach to multi-class cost-sensitive boosting, offering both a clearer theoretical framework and improved performance over prior art. Instead of first approximating the class-wise posterior distributions and then taking costs into account at test time, our approach directly estimates the class with minimum risk (i.e. minimum expected cost) by using the costs during training, as illustrated in Fig. 4.1. Since our proposed loss function estimates the underlying risk, and in particular, is based on a sum of exponentials, we call our method Risk Estimation Boosting using an Exponential Loss, or *REBEL*.

We prove that REBEL is a true boosting algorithm in the theoretical sense and outline a method for jointly optimizing decision trees as weak learners, facilitating feature sharing among classes.

In summary, the contributions of our work are:

1. a novel family of loss functions that facilitate multi-class cost-sensitive training (refer to Sec. 4.3)
2. a proof that REBEL employs the appropriate weak learning condition required for *boostability* (refer to Sec. 4.5)
3. a greedy optimization procedure for growing binary stumps into shallow decision trees (refer to Sec. 4.4)
4. state-of-the-art results on several datasets for both neutral and cost-sensitive setups (refer to Sec. 4.6)

4.2 Related Work

There are two main approaches to multi-class classification: combining multiple binary classifiers, and training multi-class classifiers directly, both of which already exist in the context of boosting.

A multi-class prediction can be achieved by compounding the outputs of multiple binary classifiers. AdaBoost.M2 [22] and AdaBoost.MR [36] train binary classifiers that maximize the margins between pairs of classes. AdaBoost.MH [36] concurrently trains M one-vs-all classifiers, sharing the same weak learners for each binary classifier. Reduction to multiple one-vs-all or one-vs-one classifiers requires augmenting the given datasets, and often results in sub-optimal joint classifiers [34]. ECOC [1] trains multiple binary classifiers and uses error-correcting codes for output classification. Selecting error-codes is often problem dependent and not straightforward; AdaBoost.ERP [28] attempts to find a good trade-off between error-correcting and base learning. JointBoost [40] trains shared binary classifiers, optimizing over all combinations of binary groupings of classes. CD-MCBoost [34] and CW-Boost [37] perform coordinate-descent on a multi-class loss function, thereby focusing each weak learner on a single class. HingeBoost.OC [24] uses output coding combined with a multi-class hinge loss, demonstrating improvements in performance on several datasets. In all of these approaches, the number of classifiers trained (and eventually evaluated) is super-linear in the number of classes, potentially slowing down classification [17].

On the other hand, strong boosted classifiers can be generated using a single chain of multi-class weak learners directly, avoiding the super-linear issue of the aforementioned methods. AdaBoost.M1 [22] is a simple adaptation of binary AdaBoost that makes use of multi-class learners, but has an unreasonably strong (often un-

achievable) weak-learning condition [30]. SAMME [45] uses a fixed set of codewords to additively generate a score for each class; however, [30] show that it does not satisfy the weak learning conditions required to be *boostable* (i.e. it is unable to reduce the training error in some situations). GD-MCBoost [34] also uses a fixed set of codewords. AOSO-LogitBoost [39] improves upon previous methods by adaptively separating pairs of classes at each iteration. Struct-Boost [38] combines weak structured learners into a strong structured classifier, applicable to multi-class classification. GLL-Boost [5] exhibits guess-aversion, attempting to avoid outputting equal scores over all classes. However, multi-class weak learners are not as easy to generate and are inherently more complex than their binary counterparts, potentially leading to worse overfitting of the training data. Further, these methods require a sum-to-zero constraint on their output codes, adding to the complexity of computation as well as restricting the classifier's output [29].

As discussed above, many varieties of boosting have been proposed over the years. However, most approaches do not properly deal with the case of cost-sensitive classification. As a workaround, several post-hoc methods exist that interpret classifier output scores as posterior distributions, enabling the estimation of the minimum-risk class as a second step [19, 23, 32]. Approximating posterior distributions is often inaccurate [31], and we claim is unnecessary. Our proposed method is distinguished from prior work, consolidating the following desirable properties into a single framework:

- a novel multi-class loss function that is easily implementable due to its simple, closed-form solutions (refer to Eq. 4.5)
- direct estimation of the minimum-risk class via cost-sensitive training; avoiding the need to (inaccurately) approximate a posterior distribution (refer to Sec. 4.3)
- a classifier composed of a single chain of binary (i.e. less complex) weak learners, sharing features among all classes; leading to less overfitting [34]
- vector-valued outputs without sum-to-zero constraints, simplifying optimization and producing classifiers that can be more expressive
- a novel method for growing binary trees as weak learners (refer to Sec. 4.4), leading to improved performance (refer to Table 4.1)

Unifying all the above features into a single framework translates into superior performance as demonstrated in Sec. 4.6, compared against both prior boosting and non-boosting approaches. Considering that boosting is one of the most widely used

supervised classification frameworks, we find that a simple, improved multi-class cost-sensitive classifier is an important contribution to the field.

Although this work focuses on developing a better boosting algorithm; for completeness, we also compare against several other state-of-the-art algorithms: 1-vs-All and multi-class SVMs [11], Structured SVMs [8] and Random Forests [9].

4.3 Approach

In this section, we review multi-class classification, formulate an appropriate upper-bounding loss function, and generalize to deal with cost-sensitive classification.

The goal of multi-class classification is to obtain a function F that correctly predicts the class of queried data points. A data point is represented as a feature vector $\mathbf{x}$ and is associated with a class label y. If there are a total of d features and K possible output classes, then:

$$\mathbf{x} \in \mathcal{X} \subseteq \mathbb{R}^d \qquad y \in \mathcal{Y} \equiv \{1,2,...,K\} \qquad F : \mathcal{X} \rightarrow \mathcal{Y}$$

For a specific classifier F, the expected misclassification error is:

$$\varepsilon_{\mathbb{E}} \equiv \mathbb{E}_{x,y}\{\mathbb{1}_{(F(\mathbf{x})\neq y)}\} \equiv \mathbb{E}_{x,y}\{\langle \mathbf{1}-\boldsymbol{\delta}_y, \boldsymbol{\delta}_{F(\mathbf{x})}\rangle\}$$

For F to output a discrete label, we first define a vector-valued *certainty function* $\mathbf{H} : \mathcal{X} \rightarrow \overline{\mathbb{R}}^K$, where the index of the maximal certainty is used as the output label:

$$F(\mathbf{x}) \equiv \arg\max_k \{\langle \mathbf{H}(\mathbf{x}), \boldsymbol{\delta}_k\rangle\}$$

In practice, we do not know the underlying distribution of the data; thus, we empirically approximate the expected misclassification error (as discussed in Sec. 2.1; see Eq. 2.2). Given a stochastically sampled training set of N points, the error is approximated as:

$$\varepsilon \approx \frac{1}{N}\sum_{n=1}^{N}\langle \mathbf{1}-\boldsymbol{\delta}_{y_n}, \boldsymbol{\delta}_{F(\mathbf{x}_n)}\rangle$$

However, as previously discussed, different errors may incur different costs (e.g. letting a knife through a security checkpoint versus a water bottle). Costs can be encoded as a matrix $\underline{\mathbf{C}}$, where each entry $c_{yk} \geq 0$ is the cost of classifying a sample as class k when its true class is y. We implicitly assume that correct classification incurs no cost; $c_{yy} = 0$. A cost vector $\mathbf{c}_y$ is simply the y^{th} row of this cost matrix.

Consequently, the empirical risk (i.e. expected cost) can be expressed as:

$$\mathcal{R} \;\equiv\; \mathbb{E}\big\{\langle \mathbf{c}_y, \boldsymbol{\delta}_{F(\mathbf{x})}\rangle\big\} \;\;\approx\;\; \frac{1}{N}\sum_{n=1}^{N}\langle \mathbf{c}_{y_n}, \boldsymbol{\delta}_{F(\mathbf{x}_n)}\rangle$$

Accordingly, given a hypothesis space $\mathcal{H}$ of possible certainty functions, we can formulate our problem as:

$$\mathbf{H}^* = \arg\min_{\mathbf{H}\in\mathcal{H}}\{\mathcal{R}\}$$

In this form, minimizing the error is infeasible, as discussed in Sec. 2.2. Without further constraining $\mathcal{H}$, the search space is intractably large; moreover, the error function is difficult to optimize as it is non-convex [29]. Instead, we propose a family of surrogate loss functions that reach the same optimum as the original error and that can tractably be minimized.

Coupled Sum

Consider any scalar function g on the extended real numbers $\overline{\mathbb{R}}$ that is convex, upper-bounds the unit step function, and whose minimal value, like the step function, is also 0 (e.g. $g(x) \equiv e^x$, $\log_2(1+e^x)$, $(x+1)^2$). Given a cost vector $\mathbf{c}_n$, by carefully choosing non-negative *subcosts* $\mathbf{c}_n^+$ and $\mathbf{c}_n^-$, we define the coupled sum $\mathcal{L}_n$ as:

$$\mathcal{L}_n(\mathbf{x}) \;\equiv\; \langle \mathbf{c}_n^+, \mathbf{g}[\mathbf{H}(\mathbf{x})]\rangle + \langle \mathbf{c}_n^-, \mathbf{g}[-\mathbf{H}(\mathbf{x})]\rangle - \langle \mathbf{c}_n^*, \mathbf{1}\rangle \tag{4.1}$$

$$\text{where:}\;\; c_{nk}^* \;\equiv \min_{h\in\overline{\mathbb{R}}}\{c_{nk}^+\, g(h) + c_{nk}^-\, g(-h)\} \quad \text{and:}\;\; \mathbf{c}_n^+, \mathbf{c}_n^- \geq \mathbf{0},$$

$$\text{(recall that:}\quad \mathbf{g}[\mathbf{H}] \equiv [g(H_1), g(H_2), ..., g(H_K)]\text{)}$$

Being a sum of convex functions, $\mathcal{L}_n$ is itself convex, and further, by construction, $\mathcal{L}_n$ has an minimal value of 0. By aptly selecting subcost vectors $\mathbf{c}_n^+$ and $\mathbf{c}_n^-$, we ensure that $\mathcal{L}_n$ upper-bounds the sample-wise misclassification cost.

Note that $\mathbf{H}(\mathbf{x})$ outputs a vector of K certainty scores, one for each class. Let $\mathbf{H}_n^*$ be the optimal certainty score given subcost vectors $\mathbf{c}_n^+$ and $\mathbf{c}_n^-$:

$$\mathbf{H}_n^* \;\equiv\; \arg\min_{\mathbf{H}\in\overline{\mathbb{R}}^K}\{\langle \mathbf{c}_n^+, \mathbf{g}[\mathbf{H}]\rangle + \langle \mathbf{c}_n^-, \mathbf{g}[-\mathbf{H}]\rangle\}$$

To guarantee that our loss function upper-bounds the misclassification cost, we can enforce the following implication:

$$\forall\, k \neq y_n,\;\; \langle \mathbf{H}(\mathbf{x}), \boldsymbol{\delta}_k\rangle \geq \langle \mathbf{H}(\mathbf{x}), \boldsymbol{\delta}_{y_n}\rangle \;\;\Rightarrow\;\; \mathcal{L}_n(\mathbf{x}) \geq \langle \mathbf{c}_n, \boldsymbol{\delta}_k\rangle \tag{4.2}$$

Equivalently, if $\mathbf{H}(\mathbf{x})$ is optimal, except for the k^{th} and y_n^{th} values which are equal to each other (i.e. $H_j(\mathbf{x}) = H^*_{nj} \;\; \forall\, j \neq k, y_n, \;\; H_k(\mathbf{x}) = H_{y_n}(\mathbf{x}) = h$), then Eq. 4.2 can be expressed as:

$$\forall\, k \neq y_n, \;\; \min_h \big\{ (c^+_{nk} + c^+_{ny_n})\, g(h) + (c^-_{nk} + c^-_{ny_n})\, g(-h) - (c^*_{nk} + c^*_{ny_n}) \big\} \;\geq\; c_{nk} \quad (4.3)$$

Depending on the choice of g, by satisfying Eq. 4.3 for each class label k, appropriate subcost vectors can be chosen[1] that ultimately imply:

$$\mathcal{L}_n(\mathbf{x}) \geq \langle \mathbf{c}_n, \boldsymbol{\delta}_{F(\mathbf{x})} \rangle \quad (4.4)$$

Accordingly, the risk $\mathcal{R}$ can be replaced with a convex, upper-bounding surrogate loss $\mathcal{L}$ (whose minimal value is also 0):

$$\boxed{\mathcal{R} \;\leq\; \mathcal{L} \;\equiv\; \frac{1}{N} \sum_{n=1}^{N} \big(\langle \mathbf{c}^+_n, \mathbf{g}[\mathbf{H}(\mathbf{x}_n)] \rangle + \langle \mathbf{c}^-_n, \mathbf{g}[-\mathbf{H}(\mathbf{x}_n)] \rangle \big) - c^*} \quad (4.5)$$

$$\text{where:} \;\; c^* \;\equiv\; \frac{1}{N} \sum_{n=1}^{N} \langle \mathbf{c}^*_n, \mathbf{1} \rangle$$

Therefore, minimizing the loss implies minimizing the risk as well.

Up to this point, we derived a new family of multi-class cost-sensitive loss functions. From hereon in, we explicitly use the function $g(x) = e^x$ as our convex upper-bound due to its simplicity and closed-form solutions. We call the resulting method *REBEL* since it uses this exponential loss.

Subcost Vectors

As discussed in Sec. 4.3, to ensure an upper-bounding loss as in Eq. 4.4, we need to enforce the inequality in Eq. 4.3, explicitly using $g(x) = e^x$:

$$\forall\, k \neq y_n, \;\; \min_h \big\{ (c^+_{nk} + c^+_{ny_n})\, e^{h} + (c^-_{nk} + c^-_{ny_n})\, e^{-h} - (c^*_{nk} + c^*_{ny_n}) \big\} \;\geq\; c_{nk} \quad (4.6)$$

Since our situation is under-constrained, we can greatly simplify the inequality in Eq. 4.6 by setting $\; c^+_{ny_n} = 0, \;\; c^-_{nk} = 0 \;\; \forall\, k \neq y_n$. Further, note that for $\; x_1, x_2 \geq 0$, $\min_h \{ x_1\, e^h + x_2\, e^{-h} \} = 2\sqrt{x_1 x_2}$; therefore, $\; \mathbf{c}^*_n = 2\sqrt{\mathbf{c}^+_n \odot \mathbf{c}^-_n} = \mathbf{0}$ (due to Eq. 4.1) Thus, the inequality in Eq. 4.6 drastically simplifies to:

$$\forall\, k \neq y_n, \;\; 2\sqrt{c^+_{nk} c^-_{ny_n}} \;\geq\; c_{nk}$$

[1] Sec. 4.3 below outlines explicit values of the subcost vectors using $g(x) = e^x$.

Accordingly, we parameterize subcost vectors $\mathbf{c}_n^+$ and $\mathbf{c}_n^-$ in terms of $\theta_n > 0$ as:

$$\mathbf{c}_n^+ = \frac{[\mathbf{c}_n]^2}{4\theta_n} \qquad \mathbf{c}_n^- = \theta_n \, \boldsymbol{\delta}_{y_n}$$

In an attempt to keep the subcost vector values as uniform as possible[2], to reduce their variance, θ_n is set to the mean of the values in $\mathbf{c}_n^+$:

$$\theta_n = \frac{1}{K-1} \Big\langle \frac{[\mathbf{c}_n]^2}{4\theta_n}, \mathbf{1} \Big\rangle \qquad \therefore \quad \theta_n^2 = \frac{\|\mathbf{c}_n\|^2}{4(K-1)} \qquad \therefore \quad \theta_n = \frac{\|\mathbf{c}_n\|}{2\sqrt{K-1}}$$

Finally, we use these θ_n to fix REBEL's subcost vectors as:

$$\mathbf{c}_n^+ = \frac{\sqrt{K-1}}{2\|\mathbf{c}_n\|} \, [\mathbf{c}_n]^2 \qquad \mathbf{c}_n^- = \frac{\|\mathbf{c}_n\|}{2\sqrt{K-1}} \, \boldsymbol{\delta}_{y_n}$$

Greedy Iterative Minimization

In this work, we model the certainty function $\mathbf{H}$ as a weighted sum of T binary weak learners. Recall that in standard binary boosting, a certainty score $h_{_T}$ is the sum of T weak learners, where $\alpha_t \in \overline{\mathbb{R}}$ are scalar weights and $f_t : \mathbb{R}^d \to \{\pm 1\}$ are binary weak learners (chosen from a hypothesis set $\mathcal{F}$), and the final classification is just the sign of $h_{_T}(\mathbf{x})$ [35]. In our model, the scalar weights α_t are extended into K-dimensional vectors $\mathbf{a}_t \in \overline{\mathbb{R}}^K$ as follows:

$$h_{_T}(\mathbf{x}) \equiv \alpha_{_0} + \sum_{t=1}^{T} f_t(\mathbf{x}) \, \alpha_t \quad \to \quad \mathbf{H}_{_T}(\mathbf{x}) \equiv \mathbf{a}_{_0} + \sum_{t=1}^{T} f_t(\mathbf{x}) \, \mathbf{a}_t \tag{4.7}$$

and the overall classifier outputs the index of the maximum-valued element in $\mathbf{H}_{_T}$.

With this additive form, after having trained the first I iterations; on iteration $I+1$, we have: $\mathbf{H}_{_{I+1}}(\mathbf{x}) = \mathbf{H}_{_I}(\mathbf{x}) + f_{_{I+1}}(\mathbf{x}) \, \mathbf{a}_{_{I+1}}$. Greedy iterative optimization may be carried out by fixing all parameters from the previous I iterations and minimizing the loss function with respect to the parameters in iteration $I+1$.

Accordingly, we expand Eq. 4.5 as:

$$\begin{aligned}
\mathcal{L}_f(\mathbf{a}) &= \frac{1}{N} \sum_{n=1}^{N} \big(\langle \mathbf{c}_n^+, \mathbf{exp}[\mathbf{H}_{_I}(\mathbf{x}_n) + f(\mathbf{x}_n) \, \mathbf{a}] \rangle + \langle \mathbf{c}_n^-, \mathbf{exp}[-[\mathbf{H}_{_I}(\mathbf{x}_n) + f(\mathbf{x}_n) \, \mathbf{a}]] \rangle \big) \\
&= \frac{1}{N} \sum_{n=1}^{N} \big(\langle \mathbf{w}_n^+, \mathbf{exp}[f(\mathbf{x}_n) \, \mathbf{a}] \rangle + \langle \mathbf{w}_n^-, \mathbf{exp}[-f(\mathbf{x}_n) \, \mathbf{a}] \rangle \big)
\end{aligned}$$

[2] Uniformity in subcost values yielded better results, albeit without extensive experimentation.

with *multi-class weights* $\mathbf{w}^{\pm}$ defined as:

$$\mathbf{w}_n^+ = \mathbf{c}_n^+ \odot \mathbf{exp}[\mathbf{H}_I(\mathbf{x}_n)], \quad \mathbf{w}_n^- = \mathbf{c}_n^- \odot \mathbf{exp}[-\mathbf{H}_I(\mathbf{x}_n)]$$

Given a binary weak learner f, we further define *weak learner weight sums* $\mathbf{s}_f^{\pm}$ as:

$$\mathbf{s}_f^{\pm} \equiv \frac{1}{N} \sum_{n=1}^{N} \left[\mathbb{1}\big(f(\mathbf{x}_n) = \pm 1\big)\, \mathbf{w}_n^+ + \mathbb{1}\big(f(\mathbf{x}_n) = \mp 1\big)\, \mathbf{w}_n^- \right] \tag{4.8}$$

leading to a compact expression for the overall loss:

$$\mathcal{L}_f(\mathbf{a}) = \langle \mathbf{s}_f^+, \mathbf{exp}[\mathbf{a}] \rangle + \langle \mathbf{s}_f^-, \mathbf{exp}[-\mathbf{a}] \rangle$$

To determine the optimal $\mathbf{a}^*$, we set the gradient to zero and solve:

$$\nabla_{\mathbf{a}} \mathcal{L}_f(\mathbf{a}^*) = \mathbf{0} \qquad \therefore \quad \mathbf{s}_f^+ \odot \mathbf{exp}[\mathbf{a}^*] = \mathbf{s}_f^- \odot \mathbf{exp}[-\mathbf{a}^*]$$

$$\therefore \quad \mathbf{a}^* = \frac{1}{2}\big(\mathbf{ln}[\mathbf{s}_f^-] - \mathbf{ln}[\mathbf{s}_f^+]\big) \tag{4.9}$$

Plugging in $\mathbf{a}^*$, the optimal loss given a weak learner f also has a compact, closed-form expression:

$$\mathcal{L}(f) \equiv \mathcal{L}_f(\mathbf{a}^*) = 2\langle \sqrt{\mathbf{s}_f^+ \odot \mathbf{s}_f^-}, \mathbf{1} \rangle \tag{4.10}$$

Also of interest is the rate of reduction in loss at a given iteration, i.e. the ratio between the optimal loss at the end of an iteration to the loss at the beginning:

$$\frac{\mathcal{L}_{I+1}}{\mathcal{L}_I} = \frac{\mathcal{L}_f(\mathbf{a}^*)}{\mathcal{L}_f(\mathbf{0})} = \frac{2\langle \sqrt{\mathbf{s}_f^+ \odot \mathbf{s}_f^-}, \mathbf{1} \rangle}{\langle \mathbf{s}_f^+ + \mathbf{s}_f^-, \mathbf{1} \rangle} \tag{4.11}$$

Finally, joint optimization of f and $\mathbf{a}$ can be accomplished by looping over candidate weak learners $f \in \mathcal{F}$ (the hypothesis space of possible weak learners), selecting the learner f^* that minimizes $\mathcal{L}(f)$ and its corresponding optimal vector $\mathbf{a}^*$.

Note that every element of $\mathbf{a}^*$ is dynamically optimized at every iteration. It is neither a fixed code-word, nor has a sum-to-zero constraint, enhancing the expressiveness of the classifier, ultimately leading to better accuracy (see Sec. 4.6). Further, in the binary (2-class) case, this framework exactly reduces to binary AdaBoost, LogitBoost, etc. (depending on the choice of $g(x)$; refer to Appendix 7.3 for details).

4.4 Decision Trees

In section 4.3, we introduced our loss function and an optimization procedure given a hypothesis space of binary weak learners. In this section, we describe our approach specifically for growing useful binary decision trees. Decision trees are simple white-box models, and in particular, shallow trees (i.e. depth ≤ 4) generalize well and have proven robust in practice [18].

A decision stump $f : \mathcal{X} \rightarrow \{\pm 1\}$ can be written as: $f(\mathbf{x}) \equiv p\,\text{sign}(\langle \mathbf{x}, \boldsymbol{\delta}_j \rangle - \tau)$ where $p \in \{\pm 1\}$ is a polarity, $\boldsymbol{\delta}_j$ selects the j^{th} feature in $\mathbf{x}$ (out of d possible features), and $\tau \in \mathbb{R}$ is a threshold. Let $\mathcal{H}_1$ denote the hypothesis space of all unique decision stumps on the training data. The input space $\mathcal{X} \subseteq \mathbb{R}^d$ has d dimensions, at times on the order of several thousands (see Sec. 4.6). Each dimension has a finite number of possible thresholds (at most equaling the number of training samples N). In practice, we only consider N_τ, a fixed number of thresholds[3]. Finally, there are two possible polarities; thus, $d \times N_\tau \times 2$ unique stumps in $\mathcal{H}_1$. Let $\mathcal{H}_D$ denote the space of unique depth-D trees. For each split in a depth-D tree, both child nodes are depth-$(D{-}1)$ trees. This leads to an exponential number of possible trees, totaling $(d \times N_\tau \times 2)^D$ unique trees in $\mathcal{H}_D$. Even on a GigaHertz computer, looping through each possible tree in $\mathcal{H}_D$ is only feasible for $D = 1$. To overcome this issue, we propose a greedy algorithm for using deeper trees as weak learners in our estimator.

Greedily Growing Trees

By greedily adding one depth-layer at a time, we can grow trees to any depth in a computationally tractable manner. Using this procedure, we guarantee that with each added depth-layer, the tree leads to a more accurate strong classifier. Let us assume that we have already jointly trained a depth-D tree $f^{(D)}$ and corresponding output vector $\mathbf{a}^{(D)}$; the following method efficiently grows the $D{+}1^{\text{th}}$ layer.

1. Replace each leaf node in $f^{(D)}$ with a stump having identical parameters as its parent node, thereby deepening the tree without changing its functionality.
2. Holding $\mathbf{a}^{(D)}$ fixed, optimize each of the 2^D newly added stumps. This operation is $O(2^D \times d \times N_\tau)$, resulting in a new tree $f^{(D+1)}$. In the worst case, all added stumps (and hence, overall accuracy) remain unchanged.
3. Fixing $f^{(D+1)}$, minimize $\mathcal{L}_f(\mathbf{a})$ with respect to $\mathbf{a}$, storing the optimal vector as $\mathbf{a}^{(D+1)}$.

[3] We find that $N_\tau \approx 200$ (evenly-spaced bins over the range of the training data) is sufficient to achieve maximal accuracy in practice resulting in $d \times N_\tau \times 2 \approx 10^6$ unique stumps in $\mathcal{H}_1$.

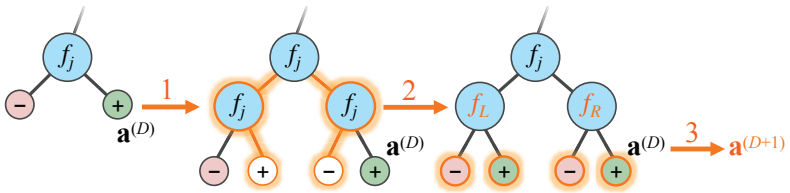


Figure 4.2: Growing a binary decision tree by one layer, as described above.

Figure 4.2 illustrates our tree-growing technique, reducing the computational complexity from 10^{6D} to $2^D \times d \times N_\tau$, making our classifier fit for use with shallow binary decision trees as weak learners.

4.5 Weak Learning Conditions

At each iteration, boosting uses a weak learner to improve upon classification in the previous iterations. Since each successive iteration can be viewed as being independent of all others, in order for a boosting algorithm to fully minimize the training error (via the training loss), the decrease in loss must always be exponential (i.e. $\exists\, \gamma > 0$ such that $\forall\, I, \quad \mathcal{L}_{I+1}/\mathcal{L}_I \leq 1-\gamma$).

If in some situation, no available weak learner is good enough, boosting cannot proceed and the algorithm terminates. Consequently, for a boosting algorithm to be able to exponentially minimize the training error in any situation (i.e. regardless of sample weights), it must always have access to an *adequate* weak learner.

"Adequate" could mean different things for different algorithms; each algorithm's particular adequacy requirement is encoded as its *weak learning condition*. For an algorithm to be *boostable*, its weak learning condition must be both necessary and sufficient for exponential error minimization [30].

In the case of binary boosting, the weak learning condition is intuitive; requiring that there exist a weak learner that achieves a weighted average accuracy marginally better than random guessing (i.e. at least $50\% + \gamma$, for some *edge* $\gamma > 0$).

REBEL's weak learning condition is:

$$\exists\, \gamma > 0 \quad \text{such that:} \quad \forall\, \mathbf{w}_n^+, \mathbf{w}_n^- \geq \mathbf{0}, \quad \exists\, f \in \mathcal{F} \;\; \text{satisfying:}$$

$$\left\langle \left| \sum_{n=1}^{N} [\mathbf{w}_n^+ - \mathbf{w}_n^-]\, f(\mathbf{x}_n) \right|, \mathbf{1} \right\rangle \;\geq\; \gamma \sum_{n=1}^{N} \langle \mathbf{w}_n^+ + \mathbf{w}_n^-, \mathbf{1} \rangle \tag{4.12}$$

Proposition 4.5.1: If REBEL's weak learning condition (Eq. 4.12) is met, then the training error is guaranteed to be driven to 0 at an exponential rate.

Proof: Note that rearranging the weight sums (i.e. $\mathbf{s}_f^{\pm}$ from Eq. 4.8), we get:

$$\mathbf{s}_f^+ - \mathbf{s}_f^- = \frac{1}{N}\sum_{n=1}^{N}[\mathbf{w}_n^+ - \mathbf{w}_n^-]f(\mathbf{x}_n) \qquad \mathbf{s}_f^+ + \mathbf{s}_f^- = \frac{1}{N}\sum_{n=1}^{N}[\mathbf{w}_n^+ + \mathbf{w}_n^-]$$

Accordingly, Eq. 4.12 can be compactly expressed as:

$$\langle|\mathbf{s}_f^+ - \mathbf{s}_f^-|, \mathbf{1}\rangle \;\geq\; \gamma\,\langle \mathbf{s}_f^+ + \mathbf{s}_f^-, \mathbf{1}\rangle$$

Since both sides are positive, squaring both sides maintains the inequality:

$$\langle|\mathbf{s}_f^+ - \mathbf{s}_f^-|, \mathbf{1}\rangle^2 \;\geq\; \gamma^2\,\langle \mathbf{s}_f^+ + \mathbf{s}_f^-, \mathbf{1}\rangle^2$$

Adding $\langle \mathbf{s}_f^+ + \mathbf{s}_f^-, \mathbf{1}\rangle^2$ to both sides and rearranging, we get:

$$(1-\gamma^2)\,\langle \mathbf{s}_f^+ + \mathbf{s}_f^-, \mathbf{1}\rangle^2 \;\geq\; \langle \mathbf{s}_f^+ + \mathbf{s}_f^-, \mathbf{1}\rangle^2 - \langle|\mathbf{s}_f^+ - \mathbf{s}_f^-|, \mathbf{1}\rangle^2 \;=\; \langle 2\mathbf{s}_f^+, \mathbf{1}\rangle\langle 2\mathbf{s}_f^-, \mathbf{1}\rangle$$

Swapping both sides and taking the square-root:

$$2\sqrt{\langle \mathbf{s}_f^+, \mathbf{1}\rangle\langle \mathbf{s}_f^-, \mathbf{1}\rangle} \;\leq\; \sqrt{1-\gamma^2}\,\langle \mathbf{s}_f^+ + \mathbf{s}_f^-, \mathbf{1}\rangle \tag{4.13}$$

Since the square-root function is concave and all $s_{fk}^{\pm} \geq 0$, then Jensen's inequality can be applied:

$$\frac{\sum_{k=1}^{K} s_{fk}^+ \sqrt{\frac{s_{fk}^-}{s_{fk}^+}}}{\sum_{k=1}^{K} s_{fk}^+} \leq \sqrt{\frac{\sum_{k=1}^{K} s_{fk}^+ \left(\frac{s_{fk}^-}{s_{fk}^+}\right)}{\sum_{k=1}^{K} s_{fk}^+}}$$

$$\therefore \;\; \sum_{k=1}^{K} s_{fk}^+ \sqrt{\frac{s_{fk}^-}{s_{fk}^+}} \leq \sqrt{\sum_{k=1}^{K} s_{fk}^+} \sqrt{\sum_{k=1}^{K} s_{fk}^+ \left(\frac{s_{fk}^-}{s_{fk}^+}\right)}$$

$$\therefore \;\; \sum_{k=1}^{K} \sqrt{s_{fk}^+ s_{fk}^-} \leq \sqrt{\Big(\sum_{k=1}^{K} s_{fk}^+\Big)\Big(\sum_{k=1}^{K} s_{fk}^-\Big)}$$

Expressed in vector form:

$$\langle \sqrt{\mathbf{s}_f^+ \odot \mathbf{s}_f^-}, \mathbf{1}\rangle \leq \sqrt{\langle \mathbf{s}_f^+, \mathbf{1}\rangle\langle \mathbf{s}_f^-, \mathbf{1}\rangle} \tag{4.14}$$

Combining Eq. 4.13 and Eq. 4.14, if the condition in Eq. 4.12 is met, then:

$$2\langle\sqrt{\mathbf{s}_f^+\odot\mathbf{s}_f^-},\mathbf{1}\rangle \;\leq\; \sqrt{1-\gamma^2}\,\langle\mathbf{s}_f^+ + \mathbf{s}_f^-,\mathbf{1}\rangle$$

Note from Eq. 4.11 that the loss ratio is equivalent to:

$$\frac{\mathcal{L}_{I+1}}{\mathcal{L}_I} \;=\; \frac{2\langle\sqrt{\mathbf{s}_f^+\odot\mathbf{s}_f^-},\mathbf{1}\rangle}{\langle\mathbf{s}_f^+ + \mathbf{s}_f^-,\mathbf{1}\rangle} \;\leq\; \sqrt{1-\gamma^2} \;\leq\; 1-\frac{\gamma^2}{2}$$

Therefore, if the condition in Eq. 4.12 is met for T iterations, then:

$$\frac{\mathcal{L}_T}{\mathcal{L}_0} \;\leq\; \Big(1-\frac{\gamma^2}{2}\Big)^T$$

and hence, REBEL minimizes the training error at an exponential rate in the number of iterations T. Q.E.D.

Proposition 4.5.2: If REBEL's weak learning condition (Eq. 4.12) cannot be met, then there is a situation in which the training error cannot be exponentially reduced.

Proof: If REBEL's weak learning condition is unmet, then its negation is true:

$$\forall\,\gamma>0,\;\; \exists\;\mathbf{w}_n^+,\mathbf{w}_n^-\geq\mathbf{0} \quad \text{such that:} \quad \forall\, f\in\mathcal{F}$$

$$\Big\langle\Big|\sum_{n=1}^{N}[\mathbf{w}_n^+-\mathbf{w}_n^-]f(\mathbf{x}_n)\Big|,\mathbf{1}\Big\rangle \;<\; \gamma\sum_{n=1}^{N}\langle\mathbf{w}_n^++\mathbf{w}_n^-,\mathbf{1}\rangle \tag{4.15}$$

Taking the limit as $\gamma\to 0$, Eq. 4.15 reduces to:

$$\exists\;\mathbf{w}_n^+,\mathbf{w}_n^-\geq\mathbf{0} \quad \text{such that:} \quad \forall\, f\in\mathcal{F}, \quad \Big\langle\Big|\sum_{n=1}^{N}[\mathbf{w}_n^+-\mathbf{w}_n^-]f(\mathbf{x}_n)\Big|,\mathbf{1}\Big\rangle \;=\; 0$$

Rewriting in terms of the weight sums $\mathbf{s}_f^{\pm}$ and solving for $\mathbf{a}^*$ using Eq. 4.9 leads to:

$$\langle|\mathbf{s}_f^+-\mathbf{s}_f^-|,\mathbf{1}\rangle=0 \qquad \therefore\;\; \mathbf{s}_f^+=\mathbf{s}_f^- \qquad \therefore\;\; \mathbf{a}^*=\frac{1}{2}\big(\mathbf{ln}[\mathbf{s}_f^-]-\mathbf{ln}[\mathbf{s}_f^+]\big)=\mathbf{0}$$

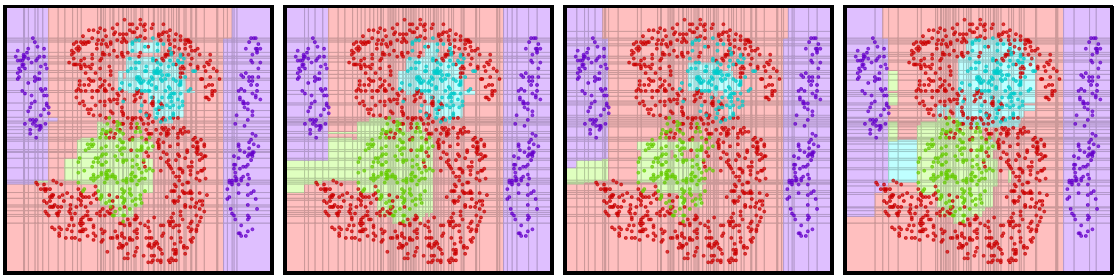
Since $\mathbf{a}=\mathbf{0}$, there is no change to the strong classifier, and hence the training error remains stationary, certainly not exponentially decreasing. Q.E.D.

Propositions 4.5.1 and 4.5.2 combine to validate that REBEL's weak learning condition (Eq. 4.12) is equivalent to boostability; thus, REBEL is indeed a true boosting method in the theoretical sense, as defined in [30].

4.6 Experiments

We benchmark our approach on both synthetic and real data, exploring its performance vis-a-vis other methods and demonstrating its functionality.

Experiments using synthetic data test REBEL's main properties; Fig. 4.3 demonstrates the behavior of classifiers trained using four different cost matrices: (a) uniform misclassification costs, (b) $10\times$ higher green false-negative costs, (c) $10\times$ higher red false-negative costs, and (d) $10\times$ higher red false-positive costs. As expected, REBEL adapts its behavior according to the input misclassification costs. Note from Fig. 4.3 that REBEL avoids missing a class with high false negative cost at the expense of missing other classes.



(a) Uniform Costs (b) High green F.N. (c) High red F.N. (d) High red F.P.

Figure 4.3: Effects of cost-sensitive training on synthetic data using REBEL with 100 depth-2 trees. Classes are color-coded, each panel is the result of a different cost matrix. (c) and (d) demonstrate the difference in output with a high false-negative cost versus a high false-positive cost. As encoded in the cost matrix, in (c), no red datapoint is misclassified, whereas in (d), no red classification is incorrect.

Fig. 4.4 traces the min-risk confidence score (i.e. $\mathbf{H}(\mathbf{x})$) over two paths through the input space, using either stumps or depth-2 trees as weak learners. The scores vary across class boundaries while staying fairly constant within class-specific regions.

Fig. 4.5 compares REBEL's performance (using cost-sensitive training) to the standard method (i.e. training on just the data, estimating the posterior distributions, and using the costs only at test-time). 10 random datasets and 20 random cost matrices were generated for a total of 200 trials.

Each dataset consists of 1000 training and 500 test points drawn from a mixture of multiple Gaussian clusters (datasets are plotted in Fig. 4.6 below). All cost matrices have zeros along the diagonals and (positive) normally-distributed off-diagonal entries, normalized such that random classification results in unit cost. For each trial, classifiers consist of 100 stumps. REBEL outperforms the standard method

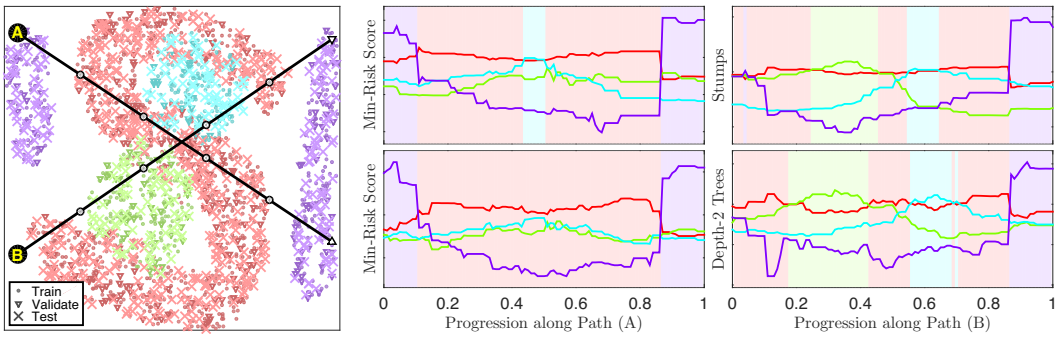


Figure 4.4: REBEL output on a synthetic 2D dataset, sampled along two paths labeled (A) and (B). Nibs along the two paths on the left-most panel are at 20% increments. Two classifiers were trained: the first using 100 stumps (top), the second using 100 depth-2 trees (bottom). Min-risk confidence scores (i.e. $\mathbf{H}(\mathbf{x})$) are plotted (the background colors correspond to the minimum-risk class).

in $\sim 90\%$ of the trials, especially on harder classification problems (i.e. when both methods incur relatively high costs).

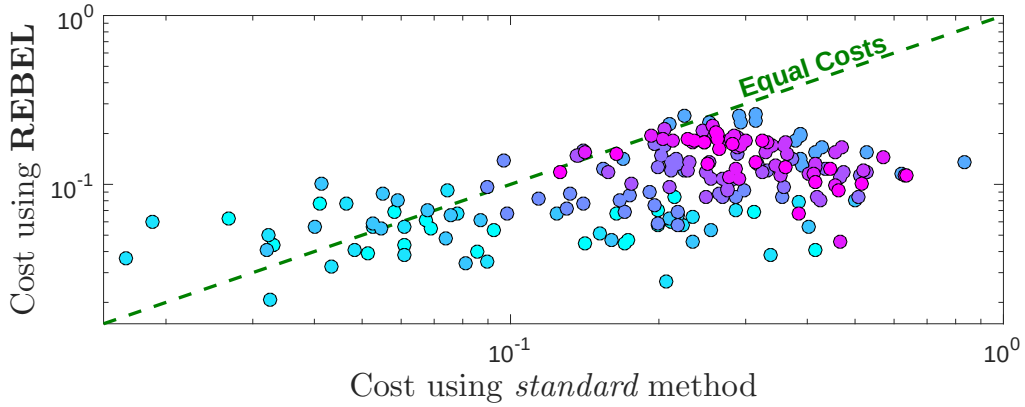


Figure 4.5: REBEL versus the standard (two-step) approach. $\sim 90\%$ of the trials lead to better results using REBEL, especially in more difficult problems (i.e. when final test misclassification costs are higher).

Standard multi-class classification on real data

We benchmark REBEL on several UCI datasets [4] and the MNIST handwritten digits [27], using a uniform (cost-neutral) metric. We compared our method against several competitive boosting methods: 1-vs-All AdaBoost and AdaBoost.MH [36], AdaBoost.ECC [16] Struct-Boost [38], CW-Boost [37], and A0S0-LogitBoost [39].

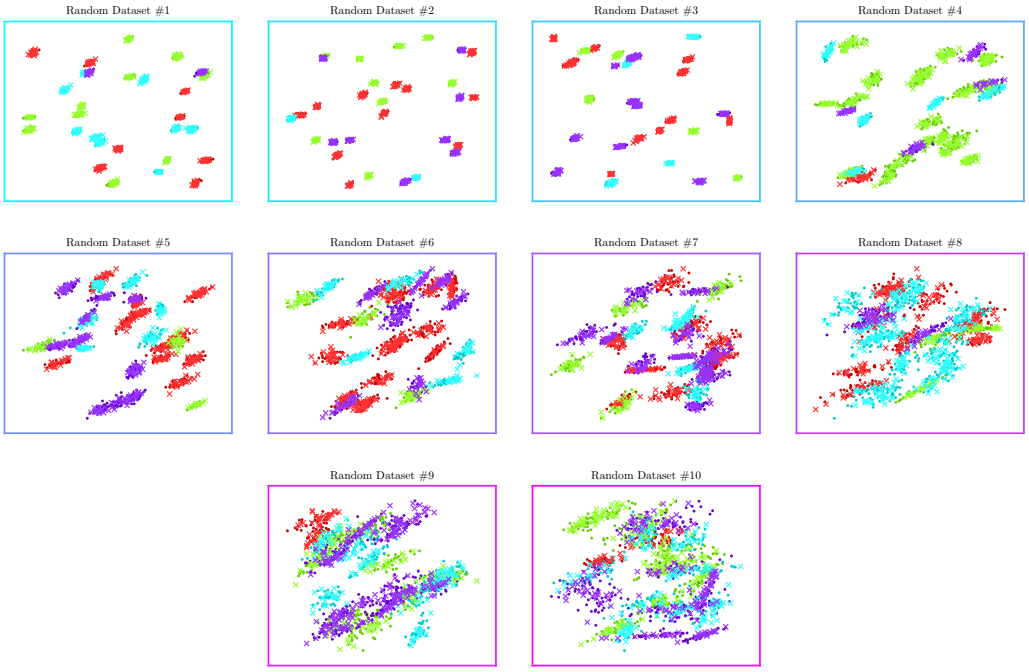


Figure 4.6: Random datasets generated for Fig. 4.5

Based on the experimental setup in [38], we use stumps as weak learners, each method having a maximum of 200 weak learners. Five versions of each dataset are generated, randomly split with 50% of the samples for training, 25% for validation (i.e. for finding the optimal number of weak learners), and the remaining 25% for testing. All misclassifications are equally costly; thus, we report percent error with the resulting means and standard deviations in Fig.4.7. REBEL is the best performing method on three of these five datasets, and is second best to CW-Boost on the two remaining datasets. Even in these commonly-used and *saturated* UCI datasets, we are able to show consistent accuracy compared to previous approaches.

Cost-sensitive classification

As discussed in the introduction, some scenarios require cost-sensitive classification. Among many existing datasets, we chose two current datasets that fall into this category: (1) SUN scene recognition [43] and (2) CUB200 fine-grained bird categorization [42]. Both of these datasets are organized into hierarchies; thus, an apt evaluation metric penalizes each misclassification according to its distance from the true class along the hierarchy.

We compare our method against state-of-the-art boosting methods as well as non-

boosting baselines: 1vsAll AdaBoost and AdaBoost.MH [36], AdaBoost.ECC [1], Struct-Boost [38], CW-Boost [37], A0S0-LogitBoost [39], multi-class and 1vsAll SVM [11], Struct-SVM [8], and Random Forests [9].

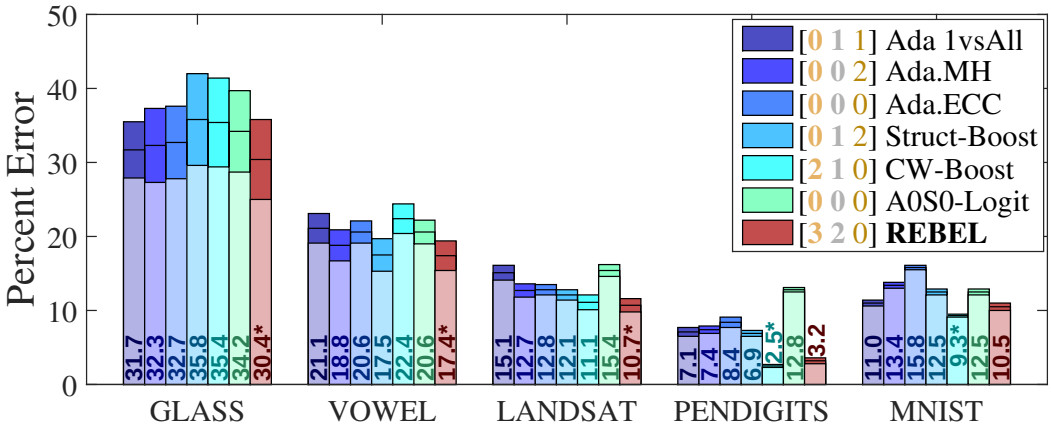


Figure 4.7: Popular boosting algorithms tested on UCI datasets. Bar caps indicate ± 1 std. from the mean. The number of times each method places [1st, 2nd, 3rd] is indicated in the legend. REBEL consistently places in the top two on every dataset.

For Struct-Boost [38], we report performance as published by their authors. For the remaining methods, we downloaded or re-implemented code, cross-validating using the training data to establish optimal parameters, specifically, the number of boosted weak learners (up to 200 on SUN and 2500 on CUB) and the tree depth (to a maximum of 4). For the SVMs, we performed grid search through C and γ as recommended in [11]. Finally, for the random forest, we performed grid search for tree depth (to a maximum of 7) and number of trees (between 1 and 1000) with combinations not exceeding the number of feature splits allotted to the boosted methods to ensure a fair comparison. For completeness, we report the performance of each method on both the hierarchical and uniform cost metrics.

SUN scene recognition

We benchmark REBEL on the same two subsets of the SUN database as used in [38], containing six and fifteen classes respectively, and using HOG features as input [12]. For each subset, five random splits were generated with 75% of the samples used for training and 25% for testing. In each of the SUN subsets, the corresponding cost matrices were normalized by the appropriate factor to ensure that random guessing

incurred the same cost ($\sim 83\%$ in the 6-class subset and $\sim 93\%$ in the 15-class subset). The mean and standard deviations over the five splits are shown in Fig. 4.8.

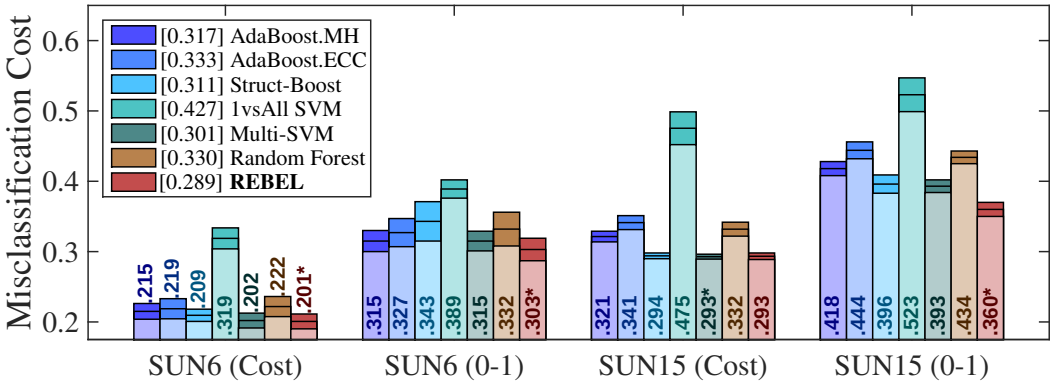


Figure 4.8: SUN scene recognition datasets, with state-of-the-art classifiers evaluated on a six-class (SUN6) and a fifteen-class subset (SUN15). Each method was trained and evaluated in both a (Cost) cost-sensitive and (0-1) cost-neutral way. REBEL is a consistent top-performing method across all of these subsets.

In cross-validating for optimal parameters on the SUN subsets, REBEL performed best using depth-2 trees as weak learners, see Table 4.1. REBEL is consistently the top-performing method on all of the subsets, edging out multi-class SVM and the other boosting methods. Other than the one-vs-all SVM, which consistently under-performs, most of the methods lead to similar accuracies.

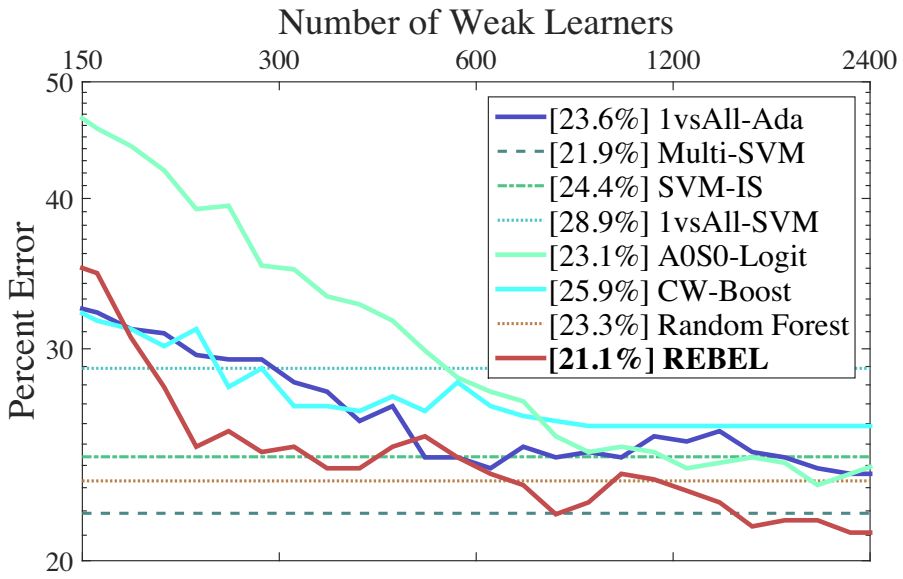
	Stumps	Depth-2	Depth-3	Depth-4
SUN6(0-1)	32.8%	**30.3%**	34.2%	35.0%
SUN6(cost)	0.363	**0.323**	0.327	0.325
CUB(0-1)	21.4%	**20.9%**	22.1%	22.9%
CUB(cost)	0.291	0.291	0.286	**0.26**

Table 4.1: REBEL misclassification errors on SUN-6 and CUB datasets using different tree depths. Deeper trees consistently outperform stumps, validating our tree-growing method of Sec. 4.4.

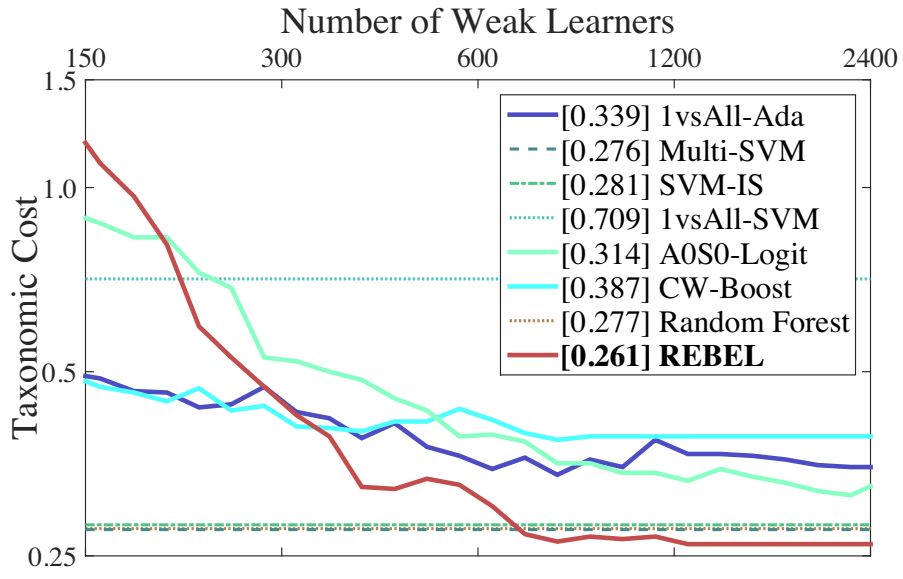
CUB fine-grained bird categorization

We further benchmark REBEL on the CUB200 fine-grained bird categorization dataset, using a fourteen-class *Woodpecker* subset as in [20, 6, 44, 13]. Each bird in

the subset is represented as a 4096-feature vector, the output of a pre-trained Convolutional Neural Network [25], consisting of 392 training and 398 test samples.



(a) Misclassification error (0-1)



(b) Cost-sensitive (Cost)

Figure 4.9: CUB200 fine-grained bird categorization results. Several state-of-the-art classifiers evaluated on a standard fourteen-class subset. All methods are trained and evaluated (a) cost-neutrally and (b) cost-sensitively, tree depths for each boosting method are chosen by cross-validation. The lowest cost of each method on the test set is indicated in square brackets. REBEL with decision trees is the best performing method in both cases.

Classification errors are shown in Fig. 4.9. Boosting methods are plotted as a function of the number of weak learners and non-boosting methods are plotted as horizontal lines. REBEL is the best performing method in both cases. All other boosting methods saturated between 20% and 40% higher costs than REBEL; significantly worse. REBEL also outperformed multi-class SVM, albeit by much smaller margins. Since this subset of birds has a particularly shallow taxonomy, its cost matrix is not that different from that of uniform errors; hence, we do not expect drastic improvements in performance and are satisfied with these *modest* gains.

In cross-validating for optimal parameters on CUB, REBEL performed best using depth-2 trees as weak learners for the cost-neutral experiment and depth-4 trees for the cost-sensitive experiment (see Table 4.1), validating the benefit of our decision tree growing approach introduced in Sec.4.4.

4.7 Discussion

Our experimental results indicate that REBEL is as good as *and even outperforms* other state-of-the-art classifiers. But (1) why does it work so well? And (2) why does it also outperform existing classifiers in cost-insensitive situations (in which REBEL *seemingly has no advantage*)?

(1) We are pleased with REBEL's performance and seek a better understanding. Our current explanation relates to the form of our model and agrees with empirical results, summarized as follows:

(a) In each iteration, REBEL determines a single binary split f_t (and the corresponding vector $\mathbf{a}_t$). This forces feature sharing across classes. Experiments show superior performance against baselines that do not require feature sharing (observed in Figs. 4.7,4.8: REBEL vs 1vsAll, Ada.ECC, CW-Boost). Feature sharing is a form of regularization [40, 34].

(b) Avoiding sum-to-zero constraints on the associated vector $\mathbf{a}_t$ is empirically advantageous. The additional degree of freedom lets each weak learner and vector pair be more expressive, leading to a steeper reduction in loss (observed in Figs. 4.7,4.8: REBEL vs AOSO, Fig. 4.9: REBEL's quick initial descent in error/cost).

Therefore (a) and (b) act in opposite directions and the balance of the two leads to beneficial results. This finding is also theoretically justified since REBEL employs the Binary Weak Learning Condition in [30], thereby neither overly overfitting nor overly underfitting the training data.

(2) The proposed surrogate loss function (Eq. 4.5) is set up to incorporate classification costs during training. However, misclassification cost is equivalent to classification error when the cost matrix has uniform costs. In this case, the min-risk class is the max-posterior class. Therefore, there is no reason to expect better or worse performance than other methods that minimize classification error. The fact that our our method performs slightly better on the datasets we tested (observed in Fig. 4.7) is a further indication that our balance of regularization and expressiveness (point (1) above) is advantageous.

4.8 Conclusions

We presented a multi-class cost-sensitive boosting framework with a novel family of simple surrogate loss functions. This framework directly models the minimum-risk class without explicitly approximating a posterior distribution. Training is based on minimizing classification costs, as specified by the user (e.g. following taxonomic distance). Specifically using an exponential-based loss function, we derived and implemented REBEL.

REBEL unifies the best qualities from a number of algorithms. It is conceptually simple, and optimizes feature sharing among classes. We found that REBEL is able to focus on important distinctions (those with costly errors) in exchange for more forgivable ones. We prove that REBEL employs the weak learning condition required to be a true Boosting algorithm in the theoretical sense.

We compared REBEL to several state-of-the-art boosting and non-boosting methods, showing improvements on the various datasets used. Being based on boosting, our method may be implemented as a cascade to yield very fast classification, useful in real-time applications. Our technique therefore holds the promise of producing classifiers that are as accurate and faster than the state-of-the-art.

References

[1] E. L. Allwein, R. E. Schapire, and Y. Singer. “Reducing multiclass to binary: a unifying approach for margin classifiers”. In: *JMLR* (2001).

[2] R. Appel, X. P. Burgos-Artizzu, and P. Perona. “Improved Multi-Class Cost-Sensitive Boosting via Estimation of the Minimum-Risk Class”. In: *arXiv* 1607.03547 (2016).

[3] R. Appel et al. “Quickly boosting decision trees-pruning underachieving features early”. In: *ICML*. 2013.

[4] K. Bache and M. Lichman. *UCI Machine Learning Repository (UC Irvine)*. 2013. URL: `http://archive.ics.uci.edu/ml`.

[5] O. Beijbom et al. “Guess-averse loss functions for cost-sensitive multiclass boosting”. In: *ICML*. 2014.

[6] T. Berg and P. N. Belhumeur. “POOF: Part-based one-vs.-one features for fine-grained categorization, face verification, and attribute estimation”. In: *CVPR*. 2013.

[7] L. Bourdev and J. Brandt. “Robust object detection via soft cascade”. In: *CVPR*. 2005.

[8] S. Branson, O. Beijbom, and S. Belongie. “Efficient Large-Scale Structured Learning”. In: *CVPR*. 2013.

[9] L. Breiman. “Random Forests”. In: *Machine Learning* (2011).

[10] X.P. Burgos-Artizzu et al. “Social Behavior Recognition in continuous videos”. In: *CVPR*. 2012.

[11] C. Chang and C. Lin. “LIBSVM: A library for support vector machines”. In: *Transactions on Intelligent Systems and Technology* (2011).

[12] N. Dalal and B. Triggs. “Histograms of Oriented Gradients for Human Detection”. In: *CVPR*. 2005.

[13] J. Deng, J. Krause, and L. Fei-Fei. “Fine-grained crowdsourcing for fine-grained recognition”. In: *CVPR*. 2013.

[14] J. Deng et al. “ImageNet: A Large-Scale Hierarchical Image Database”. In: *CVPR*. 2009.

[15] J. Deng et al. “What does classifying more than 10,000 image categories tell us?” In: *ECCV*. 2010.

[16] T. G. Dietterich and G. Bakiri. “Solving multiclass learning problems via error-correcting output codes”. In: *arXiv* 9501101 (1995).

[17] P. Dollár, R. Appel, and W. Kienzle. “Crosstalk Cascades for Frame-Rate Pedestrian Detection”. In: *ECCV*. 2012.

[18] P. Dollár et al. “Pedestrian Detection: An Evaluation of the State of the Art”. In: *PAMI* (2011).

[19] P. Domingos. “Metacost: A general method for making classifiers cost-sensitive”. In: *SIGKDD*. 1999.

[20] R. Farrell et al. “Birdlets: Subordinate categorization using volumetric primitives and pose-normalized appearance”. In: *ICCV*. 2011.

[21] Y. Freund. “Boosting a weak learning algorithm by majority”. In: *Information and Computation* (1995).

[22] Y. Freund and R. E. Schapire. “Experiments with a new boosting algorithm”. In: *Machine Learning International Workshop*. 1996.

[23] J. H. Friedman, T. Hastie, and R. Tibshirani. “Additive logistic regression: a statistical view of boosting”. In: *The Annals of Statistics*. 2000.

[24] T. Gao and D. Koller. “Multiclass boosting with hinge loss based on output coding”. In: *ICML*. 2011.

[25] Y. Jia et al. “Caffe: Convolutional Architecture for Fast Feature Embedding”. In: *arXiv* 1408.5093 (2014).

[26] M. Kabra et al. *JAABA: An interactive machine-learning tool for automatic annotation of animal behavior*. Tech. rep. 2012.

[27] Y. LeCun and C. Cortes. *The MNIST database of handwritten digits*. Tech. rep. 1998.

[28] L. Li. “Multiclass boosting with repartitioning”. In: *ICML*. 2006.

[29] Y. Mroueh et al. “Multiclass Learning with Simplex Coding”. In: *NIPS*. 2012.

[30] I. Mukherjee and R. E. Schapire. “A Theory of Multiclass Boosting”. In: *NIPS*. 2010.

[31] A. Niculescu-Mizil and R. Caruana. “Obtaining Calibrated Probabilities from Boosting”. In: *NIPS*. 2005.

[32] D. B. O’Brien, M. R. Gupta, and R. M. Gray. “Cost-sensitive Multi-class Classification from Probability Estimates”. In: *ICML*. 2008.

[33] S. Ramaswamy. “Multiclass cancer diagnosis using tumor gene expression signatures”. In: *National Academy of Sciences* (2001).

[34] M. Saberian and N. Vasconcelos. “Multiclass Boosting: Theory and Algorithms”. In: *NIPS*. 2011.

[35] R. E. Schapire. “The strength of weak learnability”. In: *Machine Learning* (1990).

[36] R. E. Schapire and Y. Singer. “Improved Boosting Algorithms Using Confidence-rated Predictions”. In: *Conference on Computational Learning Theory*. 1999.

[37] C. Shen and Z. Hao. “A direct formulation for totally-corrective multi-class boosting”. In: *CVPR*. 2011.

[38] G. Shen, G. Lin, and A. van den Hengel. “StructBoost: Boosting methods for predicting structured output variables”. In: *PAMI* (2014).

[39] P. Sun, M. D. Reid, and J. Zhou. “AOSO-LogitBoost: Adaptive One-Vs-One LogitBoost for Multi-Class Problem”. In: *arXiv* 1110.3907 (2011).

[40] A. Torralba, K. P. Murphy, and W. T. Freeman. “Sharing visual features for multiclass and multiview object detection”. In: *PAMI* (2007).

[41] P. Viola and M. J. Jones. “Robust Real-Time Face Detection”. In: *IJCV* (2004).

[42] P. Welinder et al. *Caltech-UCSD birds 200*. Tech. rep. 2010.

[43] J. Xiao et al. “Sun database: Large-scale scene recognition from abbey to zoo”. In: *CVPR*. 2010.

[44] N. Zhang, R. Farrell, and T. Darrell. “Pose pooling kernels for sub-category recognition”. In: *CVPR*. 2012.

[45] J. Zhu et al. “Multi-class AdaBoost”. In: *Statistics and its Interface* (2009).

Chapter 5

THEORETICALLY GUARANTEED LEARNERS

Corresponding paper: [2]

Powerful machine learning algorithms should also be accessible to users that don't have a lot of data, computational power, or expertise. Further, in many circumstances, an algorithm must be able to explain the logic behind its decisions. It is therefore desirable to have a simple yet accurate white-box learning system that trains quickly and with little data. To this end, we build upon the multiclass boosting method REBEL, and present a novel family of weak learners called *Localized Similarities*. Due to these learners, our framework is provably able to minimize the training error of *any* dataset at an exponential rate. We carry out experiments on a variety of synthetic and real datasets, demonstrating a tendency to avoid overfitting. We evaluate our method on several standard UCI datasets against other state-of-the-art methods, showing that our method performs amongst the best.

5.1 Introduction

The past couple of years have seen vast improvements in the performance of machine learning algorithms. Deep Neural Networks of varying architectures are achieving unprecedented performance in many fields [11]. However, as discussed in Ch. 1, there are several considerable drawbacks to employing such networks: their inner workings are complex and sometimes mysterious [18], and training them often requires expertise, processing power, computational time, and vast amounts of data not accessible to the average user [19]. Consequently, there is still a need for a simple but accurate "white-box" machine learning system that is able to train quickly and with little data.

As we have argued in previous chapters, boosted classifiers are easy and quick to train [3], produce very fast classifiers at test-time when computed as a cascade [5,

8], and can efficiently handle cost-sensitive multi-class data [1], exhibiting high empirical accuracies. Moreover, being a linear combination of weak learners, the complexity of a strong boosted classifier is dependent on its weak learners. Thus, when using simple learners such as shallow decision trees (Sec. 3.3), the inner workings of a boosted classifier are clearly interpretable; it is a white-box system.

A shallow decision tree partitions the input space using axis-aligned hyperplanes, i.e. by comparing a few input features to a few fixed thresholds. This simplicity enables very fast evaluation; however, there are distributions of data for which axis-aligned hyperplanes are unable to fully separate the classes. Furthermore, even when the training data can be perfectly classified, the resulting decision boundaries are often jagged and do not generalize well. This situation is exemplified in Fig. 5.1 (left), where a boosted shallow tree classifier is trained to perfectly classify the data.

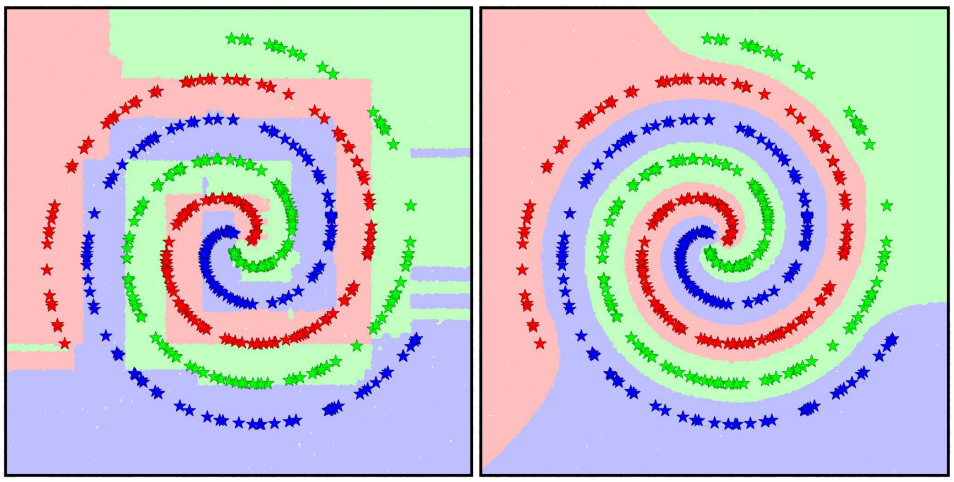


Figure 5.1: Boosted classification using shallow decision trees (left), leading to classification boundaries that are axis-aligned and not representative of the data. Although this method can lead to perfect training, it is overfitting. Our method (right) uses *Localized Similarities* (described in Sec. 5.4). Paired with a procedure that provably guarantees exponential loss minimization, our classifiers focus on well-generalizing boundaries that fit the data.

Note that the classifier on the left (using decision trees) does not accurately capture the pattern inherent in the data, whereas the classifier on the right (using our novel weak learners) leads to intuitive boundaries, properly generalizing the data.

In this chapter, we review the fundamentals of REBEL, derive an optimal binarization of the multi-class data at each iteration, and propose a novel family of weak learners which we call Localized Similarities.

The main contributions of our work are the following:

1. a derivation of the optimal binarization of multi-class data (see Sec. 5.3)
2. a proof that the training error is fully minimized within a finite number of iterations (see Sec. 5.4)
3. a simple family of weak learners (Localized Similarities) that facilitate the loss guarantees (see Sec. 5.4)
4. empirical state-of-the-art results on a range of datasets (see Sec. 5.6)

Background

Boosting is a mature method, originally formulated for binary classification (e.g. AdaBoost and similar variants) [13, 9, 10]. Although multi-class classification is more complex than its binary counterpart, many advances have been made in both performance and theory in the context of boosting (as discussed in Sec. 4.2).

In the noteworthy paper "A Theory of Multiclass Boosting" [12], many of the existing multi-class boosting methods were shown to be inadequate, either because they demand too much of their weak learners at each iteration, or because their loss function is unable to deal with some configurations of the training data. [12] outlines the appropriate *Weak Learning Condition* that a boosting algorithm must impose on its weak learners in order to guarantee training convergence. However, it does not prescribe a method with which to find an adequate set of weak learners.

The goal of our work is to propose a family of weak learners that are guaranteed to satisfy the weak learning condition and whose inner workings are easily interpretable. Using REBEL (proposed in Ch. 4) as our multi-class boosting method, our overall framework is meant to be as straightforward as possible so that it is accessible and practical to more users; we outline it in Sec. 5.2 below.

5.2 REBEL Revisited

In the multi-class setting, a datapoint is represented as a feature vector $\mathbf{x}$ and is associated with a class label y. Each sample is comprised of d features and belongs to one of K classes:

$$\mathbf{x} \in \mathcal{X} \subseteq \mathbb{R}^d \qquad y \in \mathcal{Y} \equiv \{1,2,...,K\}$$

A good classifier reduces the training error while generalizing well to potentially-unseen data. We use (cost-neutral) REBEL due to its support for binary weak learners, its mathematical simplicity (i.e. closed-form solution to loss minimization),

and its strong empirical performance [1]. REBEL returns a vector-valued output $\mathbf{H}$, the sum of T (binary weak learner f, accumulation vector $\mathbf{a}$) pairs:

$$\mathbf{H}(\mathbf{x}) \equiv \sum_{t=1}^{T} f_t(\mathbf{x})\, \mathbf{a}_t$$

$$\text{where: } \; f_t : \mathbb{R}^d \to \{\pm 1\}, \;\; \mathbf{a}_t \in \overline{\mathbb{R}}^K$$

The hypothesized class is simply the index of the maximal entry in $\mathbf{H}$:

$$F(\mathbf{x}) \equiv \arg\max_{y \in \mathcal{Y}} \{\langle \mathbf{H}(\mathbf{x}), \boldsymbol{\delta}_y \rangle\}$$

REBEL uses an exponential loss function to upper-bound the training error:

$$\mathcal{L} \equiv \frac{1}{2N} \sum_{n=1}^{N} \langle \mathbf{exp}[\mathbf{y}_n \circ \mathbf{H}(\mathbf{x}_n)], \mathbf{1} \rangle$$

$$\text{where: } \; \mathbf{y}_n \equiv \mathbf{1} - 2\,\boldsymbol{\delta}_{y_n} \;\; \text{(i.e. all } +1\text{s with a } -1 \text{ in the } y_n^{\text{th}} \text{ index)}$$

Being a greedy, additive model, all previously-trained parameters are fixed and each iteration amounts to jointly optimizing a new weak learner f and accumulation vector $\mathbf{a}$. To this end, the loss at iteration $I{+}1$ may be expressed as:

$$\mathcal{L}_{I+1} = \frac{1}{N} \sum_{n=1}^{N} \langle \mathbf{w}_n, \mathbf{exp}[f(\mathbf{x}_n)\, \mathbf{y}_n \circ \mathbf{a}] \rangle \tag{5.1}$$

$$\text{where:} \quad \mathbf{w}_n \equiv \frac{1}{2}\, \mathbf{exp}[\mathbf{y}_n \circ \mathbf{H}_I(\mathbf{x}_n)]$$

Given a weak learner f, we define the true and false weight sums ($\mathbf{s}_f^{\mathrm{T}}$ and $\mathbf{s}_f^{\mathrm{F}}$) as:

$$\mathbf{s}_f^{\mathrm{T}} \equiv \frac{1}{N} \sum_{n=1}^{N} \mathbb{1}[f(\mathbf{x}_n)\mathbf{y}_n < 0] \circ \mathbf{w}_n \qquad \mathbf{s}_f^{\mathrm{F}} \equiv \frac{1}{N} \sum_{n=1}^{N} \mathbb{1}[f(\mathbf{x}_n)\mathbf{y}_n > 0] \circ \mathbf{w}_n \tag{5.2}$$

Using these weight sums, the loss can be simplified as:

$$\mathcal{L}_{I+1} = \mathcal{L}_f \equiv \langle \mathbf{s}_f^{\mathrm{T}}, \mathbf{exp}[-\mathbf{a}] \rangle + \langle \mathbf{s}_f^{\mathrm{F}}, \mathbf{exp}[\mathbf{a}] \rangle$$

Expressed in this way, the optimal accumulation vector $\mathbf{a}^*$ has a simple closed-form solution (refer to Eq. 4.9), and the loss simplifies to:

$$\mathbf{a}^* = \frac{1}{2}\big(\mathbf{ln}[\mathbf{s}_f^{\mathrm{T}}] - \mathbf{ln}[\mathbf{s}_f^{\mathrm{F}}]\big) \qquad \therefore \;\; \mathcal{L}_f = 2\langle \sqrt{\mathbf{s}_f^{\mathrm{T}} \circ \mathbf{s}_f^{\mathrm{F}}}, \mathbf{1} \rangle \tag{5.3}$$

At each iteration, REBEL's tree-growing algorithm (Sec. 4.4) initially requires an exhaustive search through a pool of decision stumps (which is tractable but time-consuming), keeping the binary learner that best reduces the multi-class loss in Eq. 5.3. In some cases, no axis-aligned trees can further reduce the loss; hence, the training procedure stalls.

Instead, in our new framework, we first determine an adequate "binarization" of the multi-class data (i.e. a separation of the K-class data into two groups) and then train a weak learner guaranteed to reduce the loss for any possible distribution of data.

5.3 Binarizing Multi-Class Data

Note that Eq. 5.3 is upper-bounded by the following expression:

$$\mathcal{L}_f \;=\; 2\langle\sqrt{\mathbf{s}_f^{\mathrm{T}}\!\odot\mathbf{s}_f^{\mathrm{F}}},\mathbf{1}\rangle \;\leq\; \langle\mathbf{s}_f^{\mathrm{T}}\!+\mathbf{s}_f^{\mathrm{F}},\mathbf{1}\rangle - \frac{1}{2}\mathcal{U} \qquad \text{where: } \mathcal{U} \equiv \Big\langle\frac{[\mathbf{s}_f^{\mathrm{T}}\!-\mathbf{s}_f^{\mathrm{F}}]^2}{[\mathbf{s}_f^{\mathrm{T}}\!+\mathbf{s}_f^{\mathrm{F}}]},\mathbf{1}\Big\rangle \quad (5.4)$$

$$\text{which follows from: } \sqrt{x(1-x)} \;\leq\; \frac{1}{2} - \Big(\frac{1}{2}-x\Big)^2 \quad \text{and setting: } x = \frac{s^{\mathrm{T}}}{s^{\mathrm{T}}+s^{\mathrm{F}}}$$

$\mathcal{L}_f$ is minimized by maximizing $\mathcal{U}$:

$$\mathcal{U} \;=\; \Big\langle\frac{[\mathbf{s}_f^{\mathrm{T}}\!-\mathbf{s}_f^{\mathrm{F}}]^2}{[\mathbf{s}_f^{\mathrm{T}}\!+\mathbf{s}_f^{\mathrm{F}}]},\mathbf{1}\Big\rangle \;\equiv\; \Bigg\langle\frac{\Big[\sum_{n=1}^{N} f(\mathbf{x}_n)\,\mathbf{w}_n\!\odot\mathbf{y}_n\Big]^2}{\Big[\sum_{n=1}^{N}\mathbf{w}_n\Big]},\mathbf{1}\Bigg\rangle \;=\; \Big\|\sum_{n=1}^{N} f(\mathbf{x}_n)\,\mathbf{u}_n\Big\|^2 \quad (5.5)$$

$$\text{where: } \mathbf{u}_n \;\equiv\; \frac{\mathbf{w}_n\!\odot\mathbf{y}_n}{\sqrt{\Big[\sum_{n=1}^{N}\mathbf{w}_n\Big]}}$$

Eq. 5.5 can be expressed as a product of matrices by stacking all of the $\mathbf{u}_n$ as column vectors of a matrix $\underline{\mathbf{U}}$ and defining $\mathbf{f}$ as a row vector with elements $f(\mathbf{x}_n)$:

$$\mathcal{U} \;=\; \mathbf{f}\,[\underline{\mathbf{U}}^{\top}\underline{\mathbf{U}}]\,\mathbf{f}^{\top}$$

Let $\hat{\mathbf{v}}_n$ be the eigenvector of $\underline{\mathbf{U}}^{\top}\underline{\mathbf{U}}$ corresponding to the n^{th} largest eigenvalue λ_n. Consequently, $\mathbf{f}$ can be decomposed as:

$$\mathbf{f} \;=\; \langle\mathbf{f},\hat{\mathbf{v}}_1\rangle\,\hat{\mathbf{v}}_1 + \sum_{n=2}^{N}\langle\mathbf{f},\hat{\mathbf{v}}_n\rangle\,\hat{\mathbf{v}}_n \quad (5.6)$$

Since $\underline{\mathbf{U}}^\top\underline{\mathbf{U}}$ is positive semi-definite, its eigenvalues are all non-negative, thus:

$$\therefore \;\; \mathcal{U} \;=\; \lambda_1\langle \mathbf{f}, \hat{\mathbf{v}}_1\rangle^2 + \sum_{n=2}^{N} \lambda_n \langle \mathbf{f}, \hat{\mathbf{v}}_n\rangle^2 \;\geq\; \lambda_1 \langle \mathbf{f}, \hat{\mathbf{v}}_1\rangle^2$$

Further, the trace of a matrix is equal to the sum of its eigenvalues and $\underline{\mathbf{U}}^\top\underline{\mathbf{U}}$ has at most K non-zero eigenvalues (λ_1 being the largest), hence:

$$\lambda_1 \;\geq\; \frac{1}{K}\operatorname{tr}(\underline{\mathbf{U}}^\top\underline{\mathbf{U}}) \;=\; \frac{1}{K}\sum_{n=1}^{N}\|\mathbf{u}_n\|^2 \;=\; \frac{1}{K}\left\langle \frac{\sum_{n=1}^{N}[\mathbf{w}_n]^2}{\left[\sum_{n=1}^{N}\mathbf{w}_n\right]}, \mathbf{1}\right\rangle \;\geq\; \frac{\mathcal{L}_0}{KN} \tag{5.7}$$

$$\text{since:} \;\; \sum_{n=1}^{N} x_n^2 \;\geq\; \frac{1}{N}\Big(\sum_{n=1}^{N} x_n\Big)^2 \;\; \text{(by Jensen's inequality)} \;\; \text{and:} \;\; \sum_{n=1}^{N}\langle \mathbf{w}_n, \mathbf{1}\rangle = \mathcal{L}_0$$

Based on this formulation, optimal binarization is achieved by setting the binarized class b_n of each sample n as the sign of its corresponding element in $\hat{\mathbf{v}}_1$:

$$b_n \;\equiv\; \operatorname{sign}(\langle \hat{\mathbf{v}}_1, \boldsymbol{\delta}_n\rangle)$$

Accordingly, if $\mathbf{b}$ is the vector with elements b_n, then:

$$\langle \mathbf{b}, \hat{\mathbf{v}}_1\rangle^2 \;\equiv\; \langle \mathbf{sign}[\hat{\mathbf{v}}_1], \hat{\mathbf{v}}_1\rangle^2 \;=\; \langle |\hat{\mathbf{v}}_1|, \mathbf{1}\rangle^2 \;\geq\; 1 \tag{5.8}$$

(with the bound occurring for: $\hat{\mathbf{v}}_1 = \pm\boldsymbol{\delta}$; refer to App. 7.4 for proof)

Finally, by combining Eq. 5.4, Eq. 5.7, and Eq. 5.8, with perfect binarized classification (i.e. when the binary weak learner perfectly classifies the binarized data), the reduction in loss at any iteration is bounded by:

$$\frac{\mathcal{L}_{f^*}}{\mathcal{L}_0} \;\leq\; 1 - \frac{1}{2KN}$$

In general, there is no guarantee that any weak learner can achieve perfect binarized classification. However, what if we there was a weak learner that could *isolate* any single point in space (i.e. classify an inner point as $+1$ and all the rest as -1)?

5.4 Isolating Points

Let us assume that we have a weak learner f_i that can isolate a single point $\mathbf{x}_i$ in the input space $\mathcal{X}$. Accordingly, denote $\mathbf{f}_i = 2\boldsymbol{\delta}_i - \mathbf{1}$ as a vector of all -1s with a $+1$ in the i^{th} entry, corresponding to classification using the isolating weak learner $f_i(\mathbf{x}_n)$.

Assuming that $N \geq 4$, then for any unit vector $\hat{\mathbf{v}} \in \mathbb{R}^N$:

$$\max_i \{\langle \mathbf{f}_i, \hat{\mathbf{v}} \rangle^2\} \;\geq\; \frac{4}{N} \tag{5.9}$$

(Refer to App. 7.4 for proof)

Accordingly, the loss ratio at each iteration is upper-bounded:

$$\frac{\min\{\mathcal{L}_{f_1}, \mathcal{L}_{f_i}\}}{\mathcal{L}_0} \;\leq\; 1 - \frac{2}{KN^2}$$

Recall that the relative training error ε is the (discrete) fraction of incorrectly classified points in the training set:

$$\varepsilon \equiv \frac{1}{N}\sum_{n=1}^{N} \varepsilon_n \qquad \text{where:} \;\; \varepsilon_n \equiv \mathbb{1}_{\left(F(\mathbf{x}_n) \neq y_n\right)}$$

and that REBEL's loss always upper-bounds the training error:

$$\varepsilon_n \;\leq\; \frac{1}{2}\left\langle \exp[\mathbf{y}_n \circ \mathbf{H}(\mathbf{x}_n)], \mathbf{1} \right\rangle$$

Before the first iteration, the initial loss $\mathcal{L}_0 = K/2$. With each iteration, the loss decreases exponentially. Since the training error is discrete and is upper bounded by REBEL's loss, our framework is guaranteed to attain minimal training error on **any**[1] training set after a finite number of iterations:

$$T = \left\lceil \frac{\ln(2/KN)}{\ln\left(1 - \frac{2}{KN^2}\right)} \right\rceil \approx \left\lceil \frac{KN^2}{2} \ln\left(\frac{KN}{2}\right) \right\rceil \;\;\Rightarrow\;\; \frac{K}{2}\left(1 - \frac{2}{KN^2}\right)^T < \frac{1}{N} \;\;\Rightarrow\;\; \varepsilon = 0$$

Although this bound is too weak to be of practical use, it is a bound nonetheless (and can probably be improved). In the following section, we specify a family of weak learners with the ability to isolate single points.

[1] There may be situations in which multiple samples belonging to different classes are coincident in the input space. These cases can be dealt with (before or during training) either by assigning all such points as a special "mixed" class (to be dealt with at a later stage), or by setting the class labels of all coincident points to the single label that minimizes error (or cost).

One/Two-Point Localized Similarities

Classical decision stumps compare a single feature to a single threshold, outputting $+1$ or -1. Instead, our proposed family of weak learners (which we call *localized similarities*) compare points, and have two modes of operation. Let $k(\mathbf{x}_i, \mathbf{x}_j)$ denote a measure of similarity between vectors $\mathbf{x}_i$ and $\mathbf{x}_j$; specifically, we use negative squared Euclidean distance $-\|\mathbf{x}_i - \mathbf{x}_j\|^2$ due to its simplicity and effectiveness.

1. In one-point mode, given an anchor $\mathbf{x}_i$ and a thershold τ, the input space is classified as positive if it is more *similar* to $\mathbf{x}_i$ than τ and negative otherwise, ranging between $+1$ and -1:

$$f_i(\mathbf{x}) \equiv \frac{\tau - \|\mathbf{x}_i - \mathbf{x}\|^2}{\tau + \|\mathbf{x}_i - \mathbf{x}\|^2}$$

2. In two-point mode, given supports $\mathbf{x}_i$ and $\mathbf{x}_j$, the input space is classified as positive if it is more *similar* to $\mathbf{x}_i$ than to $\mathbf{x}_j$ (and vice-versa), with maximal absolute activations around $\mathbf{x}_i$ and $\mathbf{x}_j$, falling off away from the midpoint $\mathbf{m}$:

$$f_{ij}(\mathbf{x}) \equiv \frac{\langle \mathbf{d}, \mathbf{x} - \mathbf{m} \rangle}{4\|\mathbf{d}\|^4 + \|\mathbf{x} - \mathbf{m}\|^4}$$

$$\text{where: } \mathbf{d} \equiv \frac{1}{2}[\mathbf{x}_i - \mathbf{x}_j] \quad \text{and: } \mathbf{m} \equiv \frac{1}{2}[\mathbf{x}_i + \mathbf{x}_j]$$

One-point mode enables the isolation of any single datapoint, guaranteeing a baseline reduction in loss. However, it essentially memorizes the training data; mimicking a nearest neighbor classifier. Two-point mode adds the capability to generalize better by providing margin-style functionality. The combination of these two modes makes localized similarities flexible enough to tackle a wide range of classification problems. Furthermore, in either mode, the functionality of a localized similarity is easily interpretable: "which of these fixed training points is a given query point more similar to?"

Finding Adequate Localized Similarities

Given a dataset with N samples, there are about N^2 possible localized similarities. The following procedure selects an adequate localized similarities out of the many:

0. Using Eq. 5.3, calculate the base loss $\mathcal{L}_1$ for the *homogeneous* stump f_1 (i.e. the one-point stump with any $\mathbf{x}_i$ and $\tau \equiv \infty$, classifying all points as $+1$).
1. Compute the eigenvector $\hat{\mathbf{v}}_1$ (as in Eq. 5.6); group the points based on their binarized class labels b_n.

2. Find the optimal isolating Localized Similarity f_i (i.e. with $\mathbf{x}_i$ and appropriate τ, classifying point i as $+1$ and all other points as -1).
3. Using Eq. 5.3, calculate the corresponding loss $\mathcal{L}_i$. Of the two stumps f_1 and f_i, store the one with smaller loss as best-so-far.
4. Find point $\mathbf{x}_j$ most similar[2] to $\mathbf{x}_i$ among points of the opposite binarized class:

$$\mathbf{x}_j = \arg\max_{b_j=-b_i} \{k(\mathbf{x}_i, \mathbf{x}_j)\}$$

5. Calculate the loss achieved using the two-point localized similarity with $\mathbf{x}_i$ and $\mathbf{x}_j$ as supports. If it outperforms the previous best, store the newer learner and update the best-so-far loss.
6. Find all points that are *similar enough*[3] to $\mathbf{x}_j$ and remove them from consideration for the remainder of the current iteration. If all points have been removed, return the best-so-far stump; otherwise, loop back to step 4.

Upon completion of this procedure, the best-so-far stump is guaranteed to lead to an adequate reduction in loss, based on the derivation in Sec. 5.3 above.

5.5 Generalization Experiments

Our boosting method provably reduces the loss well after the training error is minimized. In this section, we demonstrate that the continual reduction in loss serves only to improve the decision boundaries and not to overfit the data.

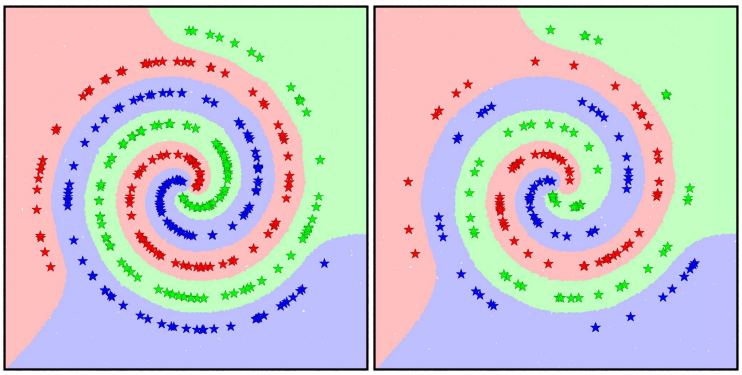


Figure 5.2: A 500-point 2-dimensional synthetic dataset with a $(2/3, 1/3)$ split of train data (left plot) to test data (right plot). Background shading corresponds to the hypothesized class using our framework.

[2] "most similar" need not be exact; approximate nearest neighbors also work.
[3] In our implementation, we remove all $\mathbf{x}_n$ where $f_{ij}(\mathbf{x}_n) \leq f_{ij}(\mathbf{x}_j)/2$.

We generated 2-dimensional synthetic datasets in order to better visualize and get an intuition for what the classifiers are doing. The results shown in this chapter are based on a dataset composed of 500 points belonging to one of three classes in a spiral formation, with a $(2/3, 1/3)$ train/test split. Fig. 5.2 shows the hypothesized class using a classifier trained for 1000 iterations.

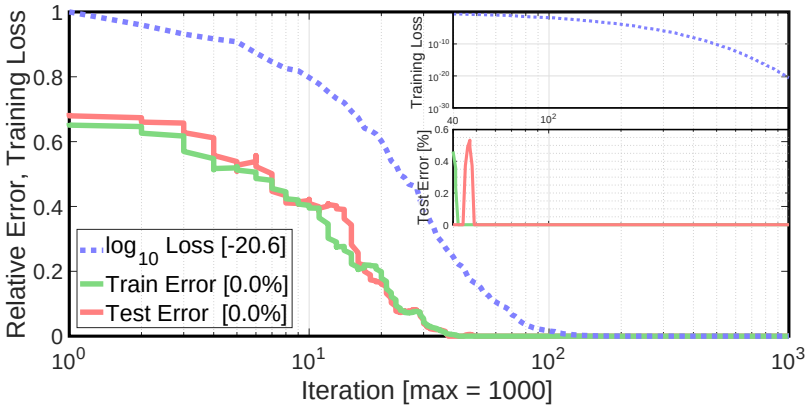


Figure 5.3: A plot of training loss, training error, and test error as a classifier is trained for 1000 iterations. Note that the test error does not increase even after the training error drops to zero. The lower inset is a zoomed-in plot of the train and test error, the upper inset is a plot of training loss using a log-scaled y-axis; both inset plots are congruous with the original x-axis.

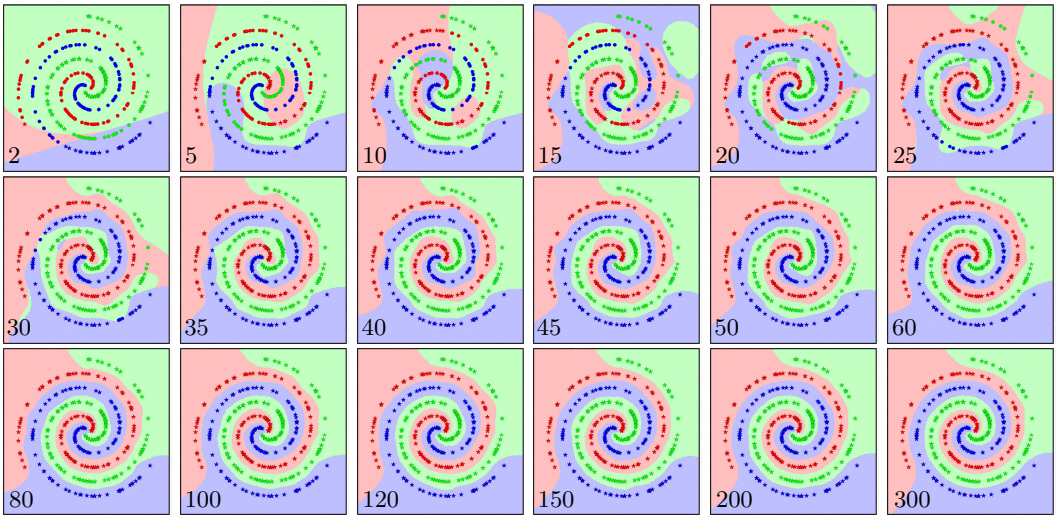


Figure 5.4: The progression of a classifier as it is trained (the corresponding iteration number is shown at the bottom left of each subplot).

Our classifier achieves perfect training (left) and test classification (right), producing a visually simple *well-generalizing* contour around the points. Training curves are given in Fig. 5.3, tracking the loss and classification errors per training iteration. Note that the test error does not increase even after the training error drops to zero.

In Fig. 5.4, we show the progression of a classifier as it is being trained. Once perfect training error is achieved (around iteration 35), the remaining iterations seem to focus on smoothening out classification boundaries.

The following experiments explore the functionality of our framework (i.e. REBEL using localized similarities) in two scenarios that could arise in practice: (1) varying sparsity of training data, and, (2) varying amounts of mislabeled training data.

Sparse Training Data

In this section of experiments, classifiers were trained using varying amounts of data, from $4/5$ to $1/5$ of the total training set. Fig. 5.5 shows the classification boundaries learned by the classifier (a_i,b_i), and the training curves (c_i). In all cases, the boundaries seem to aptly fit (and not *overfit*) the training data (i.e. being satisfied with isolated patches *without* overzealously trying to connect points of the same class together). This is more rigorously observed from the training curves; the test error does not increase after reaching its minimum (for hundreds of iterations).

Mislabeled Training Data

In this section of experiments, classifiers were trained with varying fractions of mislabeled data; from 1% to 30% of the training set. Fig. 5.6 shows the boundaries learned by the classifier (a_i,b_i), and the training curves (c_i). All classifiers seem to degenerate gracefully, isolating rogue points and otherwise maintaining relatively smooth boundaries. Even the classifier trained on 30% mislabeled data (which we would consider to be unreasonably noisy) is able to maintain smooth boundaries.

In all cases, the training curves still show that the test error is fairly stable once reaching its minimum value. Moreover, test errors approximately equal the fraction of mislabeled data, further validating the generalization properties of our method.

Real Data

Although the above observations are promising, they could result from the fact that the synthetic datasets are 2-dimensional. In order to rule out this possibility, we perform similar experiments on several UCI datasets [4] of varying input dimen-

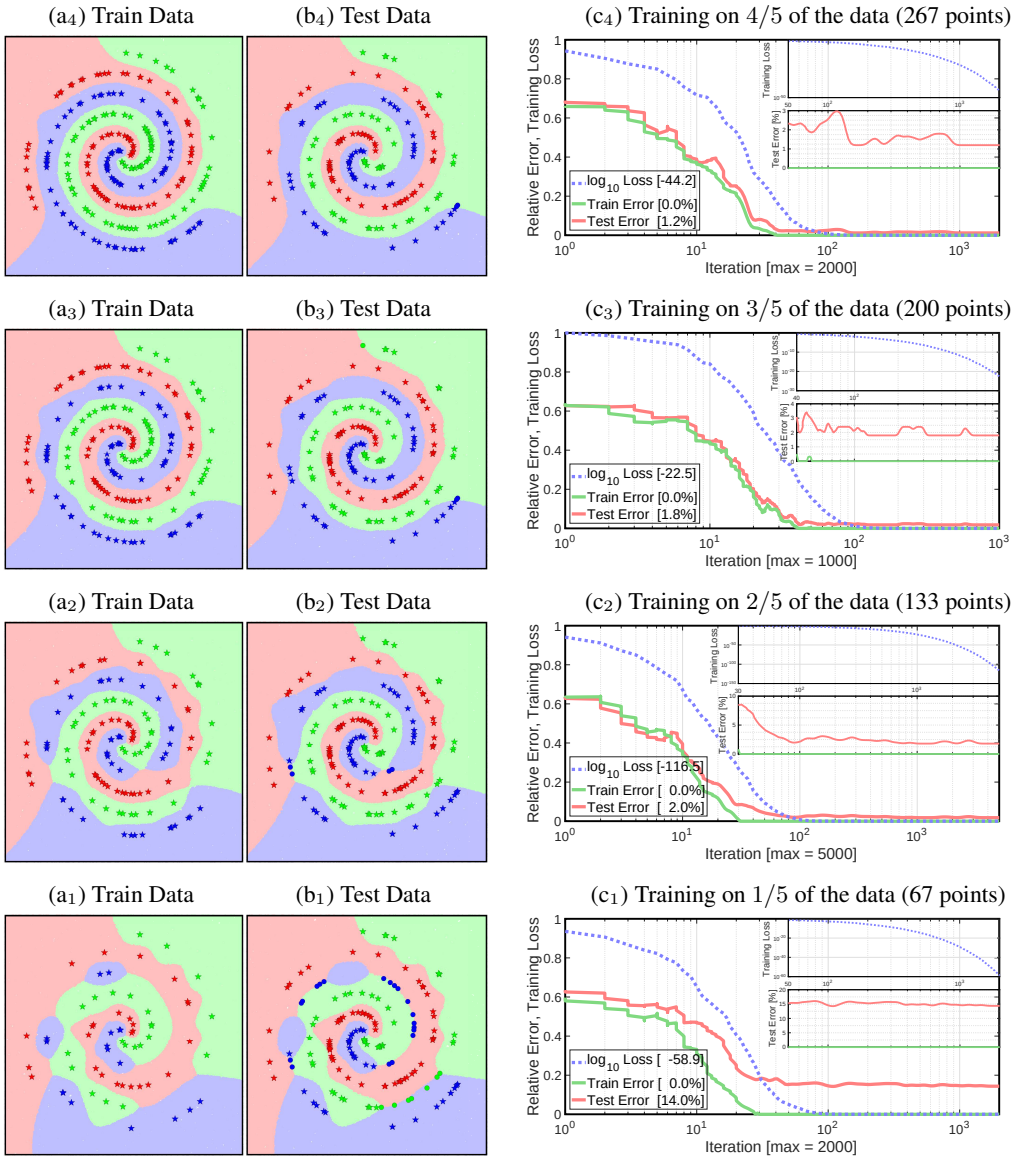


Figure 5.5: Classification boundaries (a,b), and training curves (c) when a classifier is trained on varying amounts of data. Stars are correctly-classified, circles are misclassified. In all cases, the test error is fairly stable once reaching its minimum.

sionalities (from 9 to 617). From the training curves in Fig. 5.7 (on the following page), we observe that once the test errors saturate, they no longer increase, even after hundreds of iterations.

In Fig. 5.8, we plot the training losses on a log-scaled y-axis. The linear trend signifies an exponential decrease in loss per iteration. Our proven bound predicts a much slower (exponential) rate than the actual trend observed during training. Note that within the initial $\sim 10\%$ of the iterations, the loss drops at an even faster rate,

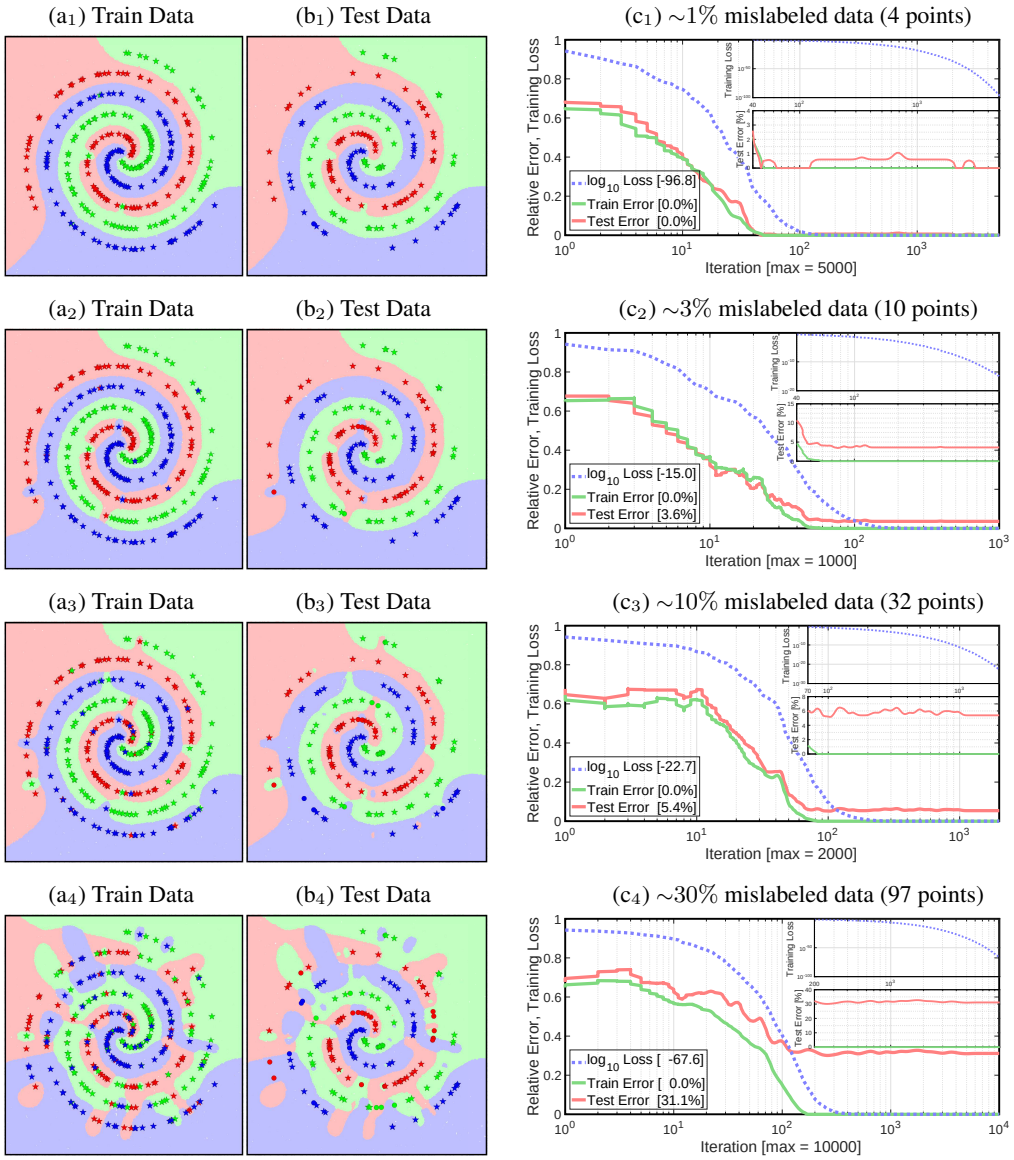


Figure 5.6: Classification boundaries (a,b), and training curves (c) when a classifier is trained on varying fractions of mislabeled data. In all cases, the test error is fairly stable once reaching its minimum. Even with 30% mislabeled data, the classification boundaries are reasonable given the training labels.

after which it settles down to a seemingly-constant rate of exponential decay. We have not yet determined the characteristics (i.e. the theoretically justified rates) of these observed trends, and relegate this endeavor to future work.

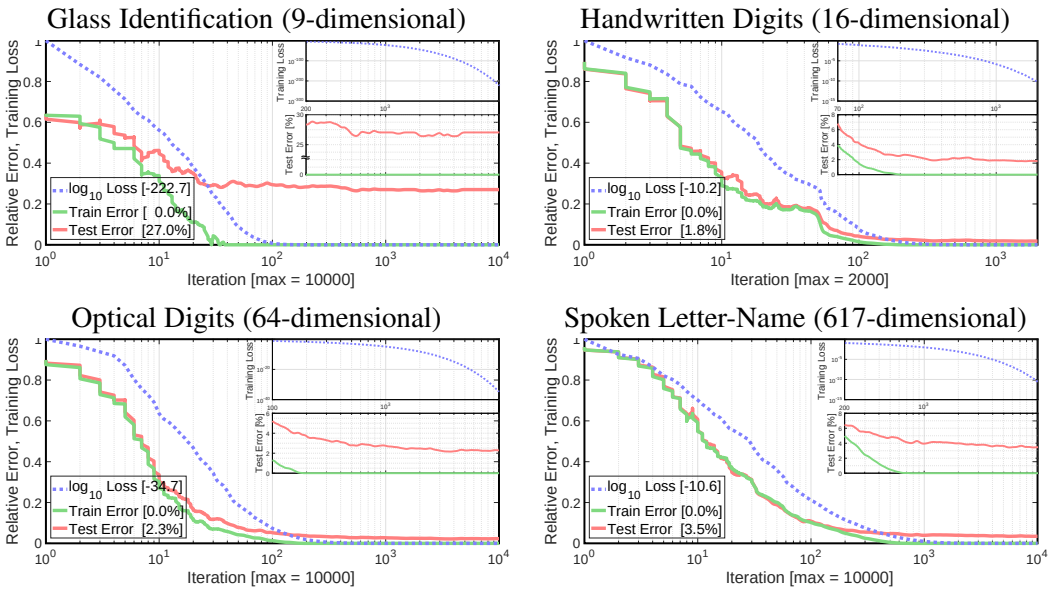


Figure 5.7: Training curves for classifiers trained on UCI datasets with a range of dimensionalities. In all cases, the test error is stable once it reaches its minimum.

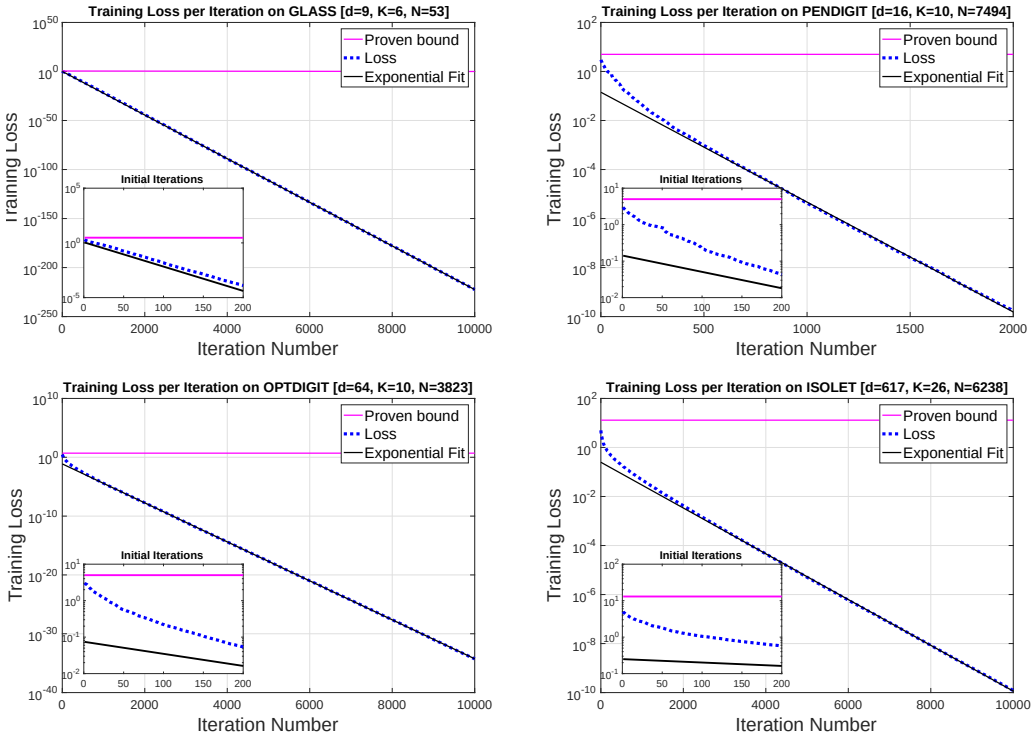


Figure 5.8: Training losses for classifiers trained on UCI datasets. The linear trend (visualized using a log-scaled y-axis) signifies an exponential decrease in loss, albeit at a much faster rate than established by our proven bound.

5.6 Comparison with Other Methods

In Sec. 5.4 we proved that our framework adheres to theoretical guarantees, and in Sec. 5.5 above, we showed that it has promising empirical properties. In this section, we compare our classifiers against several state-of-the-art boosting baselines. Specifically, we compare our method against 1-vs-All AdaBoost and AdaBoost.MH [14], AdaBoost.ECC [7], Struct-Boost [16], CW-Boost [15], AOSO-LogitBoost [17], and REBEL (using shallow decision trees as weak learners) [1].

Based on the same experimental setup as in [16, 1], each method is trained to a maximum of 200 weak learners. For each dataset, five random splits are generated, with 50% of the samples for training, 25% for validation (i.e. for setting hyperparameters where needed), and the remaining 25% for testing.

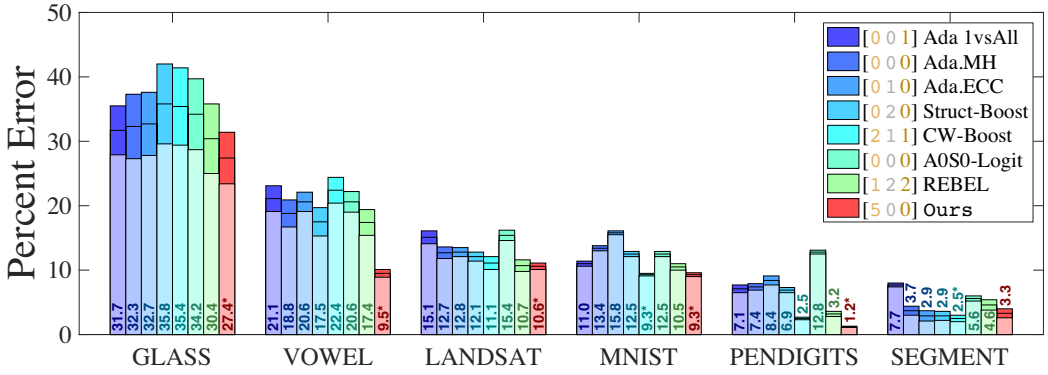


Figure 5.9: Test errors of various state-of-the-art and baseline classification methods on MNIST and several UCI datasets. All boosting methods are allowed to use 200 weak learners. Our method is the best on all but one dataset shown.

Our method is the most accurate on five of the six datasets tested. In the Vowel dataset, it achieves almost half of the error as the next best method. Note that although our framework uses REBEL as its boosting method, the Localized similarites add an extra edge, outperforming REBEL with decision trees in all runs. These results demonstrate the ability of our framework to produce easily interpretable classifiers that are also empirically proficient.

Comparison with Neural Networks

Complex neural networks are able to achieve remarkable performance on large datasets, but they require an amount of training data proportional to their complexity. In the regime of small to medium amounts of data (within which the UCI and MNIST datasets belong, i.e. $10 < N < 10^6$ training samples), such networks

cannot be too complex. Accordingly, in Fig. 5.10, we compare our method against fully-connected neural networks (with one hidden layer).

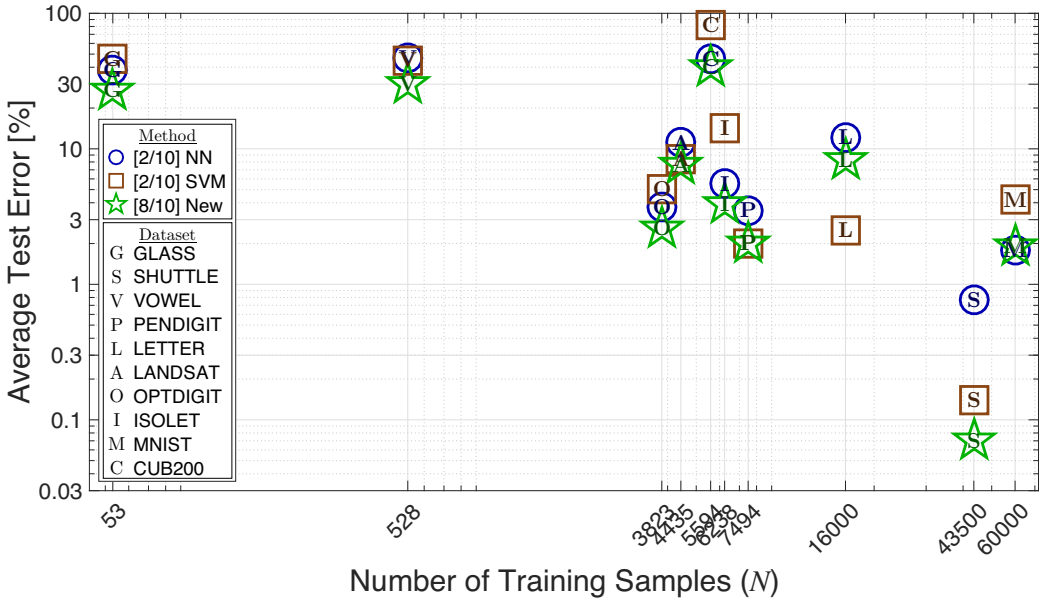


Figure 5.10: Comparison of our method versus Neural Networks and Support Vector Machines on ten datasets of varying sizes and difficulties. Our method is the most accurate on all but one dataset.

Four neural networks were implemented, each having one of the following architectures: $[d-4d-K]$, $[d-4K-K]$, $[d-2d-d-K]$, $[d-4K-2K-K]$. Only the one with the best test error is shown in the plot. A multi-class SVM [6] was validated using a 5×6 parameter sweep for C and γ. Our method was run until the training loss fell below $1/N$. Overall, our method achieves the best results on eight of the ten datasets, decisively marking it as the method of choice for this range of data.

Training times are plotted in Fig. 5.11. Our method's training times were comparable to those for neural networks (although our current code is not optimized so we expect substantial speed-ups). For the ISOLET, MNIST, and CUB200 datasets, the SVMs took much longer to train, whereas they were on-par with the other methods for the smaller datasets.

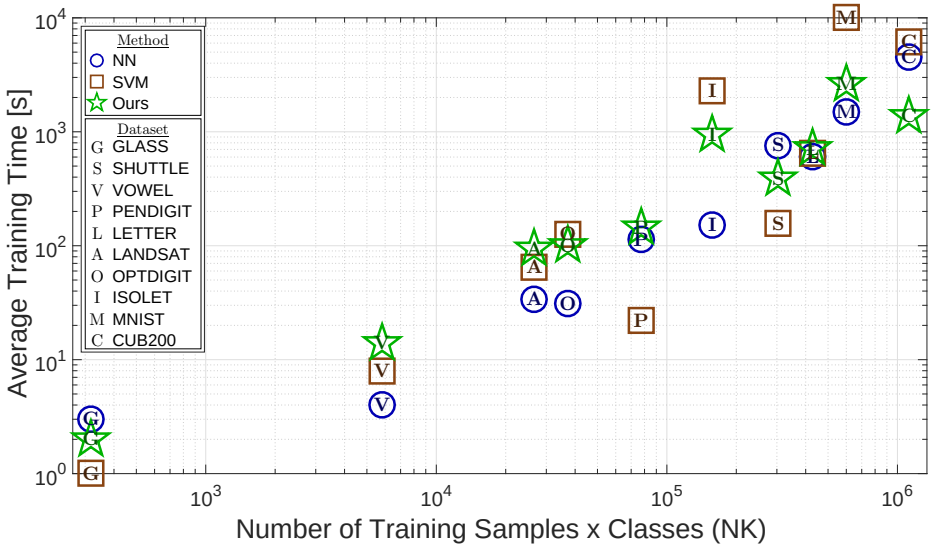


Figure 5.11: Comparison of the training times for our method, Neural Networks, and Support Vector Machines. At the time of this run, our code was not optimized. We expect substantial speed-ups in the future.

5.7 Discussion

In Sec. 5.5, we observed that our classifiers tend to smoothen the decision boundaries in the iterations beyond zero training error. In Fig. 5.12, we see that this is not the case with the typically-used axis-aligned decision stumps. Why does this happen with our framework?

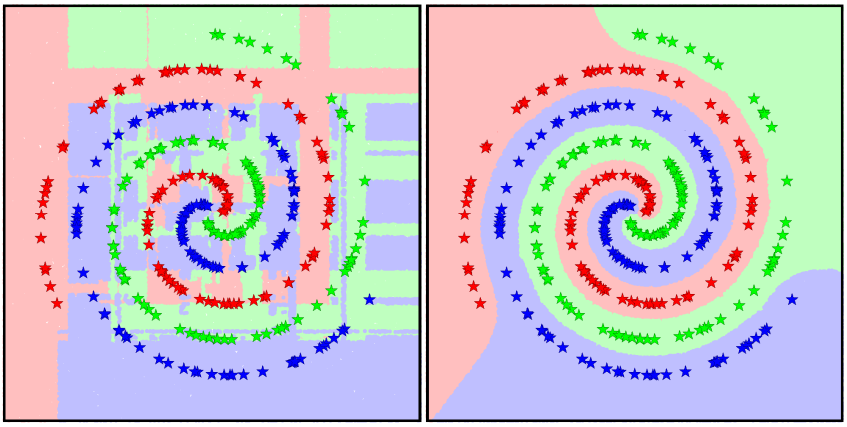


Figure 5.12: The contrasted difference between overtraining using (a) classical decision stumps and (b) localized similarities. (a) leads to massive overfitting of the training data, whereas (b) leads to smoothening of the decision boundaries.

First, we note that the largest-margin boundary between two points is the hyperplane that bisects them. Every two-point localized similarity is such a bisector. Therefore, it is not surprising that with only a pool of localized similarities, a classifier should have what it needs to place good boundaries. Further, since not all pairs need to be separated (many neighboring points belong to the same class), only a small subset of the $\sim N^2$ possible learners will ever need to be used.

Secondly, we note that if some point (either an outlier or an unfortunately-placed point) continues to increase in weight until it can no-longer be ignored, it can simply be isolated and individually dealt with using a one-point localized similarity, there is no need to combine it with other "innocent-bystander" points. This phenomenon is observed in the mislabeled training experiments in Sec. 5.5.

Together, the two types of localized similarities complement each other. With the guarantee that every step reduces the loss, each iteration focuses on either further smoothening out an existing boundary, or reducing the weight of a single unfit point.

5.8 Conclusions

We have presented a novel framework for multi-class boosting that makes use of a simple family of weak learners, called Localized Similarities. Each of these learners has a clearly understandable functionality; a test of similarity between a query point and some pre-defined samples.

We have proven that the framework adheres to theoretical guarantees: the training loss is minimized at an exponential rate, and since the loss upper-bounds the training error (which can only assume discrete values), our framework is therefore able to achieve perfect training on **any** dataset.

We further explored some of the empirical properties of our framework, noting that the combination of localized similarities and guaranteed loss reduction tend to lead to a non-overfitting regime, in which the classifier focuses on smoothing-out its decision boundaries. Finally, we compare our method against several state-of-the-art methods, outperforming all of the methods in most of the datasets.

Altogether, we believe that we have achieved our goal of presenting a simple multi-class boosting framework with theoretical guarantees and empirical proficiency.

References

[1] R. Appel, X. P. Burgos-Artizzu, and P. Perona. "Improved Multi-Class Cost-Sensitive Boosting via Estimation of the Minimum-Risk Class". In: *arXiv* 1607.03547 (2016).

[2] R. Appel and P. Perona. "A Simple Multi-Class Boosting Framework with Theoretical Guarantees and Empirical Proficiency". In: *ICML*. 2017.

[3] R. Appel et al. "Quickly boosting decision trees-pruning underachieving features early". In: *ICML*. 2013.

[4] K. Bache and M. Lichman. *UCI Machine Learning Repository (UC Irvine)*. 2013. URL: `http://archive.ics.uci.edu/ml`.

[5] L. Bourdev and J. Brandt. "Robust object detection via soft cascade". In: *CVPR*. 2005.

[6] C. Chang and C. Lin. "LIBSVM: A library for support vector machines". In: *Transactions on Intelligent Systems and Technology* (2011).

[7] T. G. Dietterich and G. Bakiri. "Solving multiclass learning problems via error-correcting output codes". In: *arXiv* 9501101 (1995).

[8] P. Dollár et al. "Fast Feature Pyramids for Object Detection". In: *PAMI* (2014).

[9] Y. Freund. "Boosting a weak learning algorithm by majority". In: *Information and Computation* (1995).

[10] Y. Freund and R. E. Schapire. "Experiments with a new boosting algorithm". In: *Machine Learning International Workshop*. 1996.

[11] Y. LeCun, Y. Bengio, and G. E. Hinton. "Deep learning". In: *Nature Research* (2015).

[12] I. Mukherjee and R. E. Schapire. "A Theory of Multiclass Boosting". In: *NIPS*. 2010.

[13] R. E. Schapire. "The strength of weak learnability". In: *Machine Learning* (1990).

[14] R. E. Schapire and Y. Singer. "Improved Boosting Algorithms Using Confidence-rated Predictions". In: *Conference on Computational Learning Theory*. 1999.

[15] C. Shen and Z. Hao. "A direct formulation for totally-corrective multi-class boosting". In: *CVPR*. 2011.

[16] G. Shen, G. Lin, and A. van den Hengel. "StructBoost: Boosting methods for predicting structured output variables". In: *PAMI* (2014).

[17] P. Sun, M. D. Reid, and J. Zhou. "AOSO-LogitBoost: Adaptive One-Vs-One LogitBoost for Multi-Class Problem". In: *arXiv* 1110.3907 (2011).

[18] C. Szegedy et al. "Intriguing properties of neural networks". In: *arXiv* 1312.6199 (2013).

[19] F. Yu et al. "LSUN: Construction of a Large-scale Image Dataset using Deep Learning with Humans in the Loop". In: *arXiv* 1506.03365 (2015).

Chapter 6

CONCLUSIONS

Boosting is an intuitive method for combining multiple weak (i.e. not very accurate) classifiers into a single strong (i.e. accurate) classifier. When implemented with cascaded evaluation, it is the amongst the fastest methods at test-time. Boosting had its shortcomings: decision trees commonly used as weak learners were slow to train and were unable to properly fit the data, and there was no simple unified framework for tackling cost-sensitive multi-class problems.

In this work, I have addressed all of these issues: when there is a need for an accurate, easy-to-use, white-box machine learning system that is able to train efficiently and with little data, I have shown that boosted classifiers are the tool for the job; specifically, REBEL using Localized Similarities as weak learners.

In Ch. 3, I presented a principled approach (called QuickBoost) for training identical classifiers at an order of magnitude faster than before. My approach is built on a bound on classification or regression error, guaranteeing that gains in speed do not come at a loss in classification performance. Experiments show that this method is able to reduce training cost by an order of magnitude or more, or given a computational budget, can train classifiers that reduce errors by two-fold or more.

In Ch. 4, I presented a multi-class cost-sensitive boosting framework with a novel family of simple surrogate loss functions. My framework directly models the min-risk class without explicitly approximating a posterior distribution. Training is based on minimizing user-specified classification costs (e.g. following taxonomic distance). Specifically using an exponential-based loss function, I derived and implemented REBEL.

REBEL unifies the best qualities from a number of algorithms. It is conceptually simple and optimizes feature sharing among classes. REBEL is able to focus on important distinctions (those with costly errors) in favor of more forgivable ones. I proved that REBEL employs the weak learning condition required to be a true boosting algorithm in the theoretical sense. I compared REBEL to several state-of-the-art methods, showing improvements on a range of datasets.

In Ch. 5, I presented a novel framework for multi-class boosting that makes use

of a simple family of weak learners, called Localized Similarities. Each of these learners has a clearly understandable functionality: a test of similarity between a query point and some pre-defined samples. I proved that the framework adheres to theoretical guarantees: the training loss is minimized at an exponential rate, and have shown that my framework is therefore able to achieve perfect training on any dataset within a finite number of iterations.

I also showcased the empirical properties of my framework, noting that the combination of Localized Similaritiesand guaranteed loss reduction tend to lead to a non-overfitting regime in which the classifier focuses on smoothing-out its decision boundaries. Finally, I compared my method against several state-of-the-art methods, outperforming all methods in all but one of the datasets.

Altogether, I have progressed the theory of cost-sensitive multi-class boosting and have shown that REBEL paired with Localized Similarities achieves state-of-the-art results. It can be run with the push of a button, no expertise necessary, its functionality is interpretable, and its performance is better than Neural Networks and other competing methods in the mid-sized data regime. It is the tool that everyone should have in their toolbox and it should be the first one they try.

Chapter 7

APPENDICES

7.1 Statistically Motivated Derivation of AdaBoost

In Ch. 2, we motivated AdaBoost from a practical standpoint by finding a suitable upper-bounding convex loss function of the misclassification error. Here, we give an alternative derivation, one that is statistically motivated, as in [1].

As discussed in Sec. 2.1, in discriminative classification, our goal is to learn the intrinsic posterior distribution of the data, $P_{y|x}$. Specifically, in the case of binary classification (i.e. $y \in \{\pm 1\}$), it would suffice to know $P(y = +1 \,|\, \mathbf{x})$.

We propose to estimate the posterior as $\tilde{p}(\mathbf{x}) \approx P(y = +1 \,|\, \mathbf{x})$ by expressing it as a logistic function of a confidence value $H : \mathcal{X} \to \overline{\mathbb{R}}$, as follows:

$$\tilde{p}(\mathbf{x}) \;=\; \frac{e^{H(\mathbf{x})}}{e^{-H(\mathbf{x})} + e^{H(\mathbf{x})}} \tag{7.1}$$

Note that this logistic form ensures that: $H \in [-\infty, \infty] \;\Leftrightarrow\; \tilde{p} \in [0, 1]$

To optimize our model, we minimize its expected negative log-likelihood:

$$\begin{aligned}
\mathcal{L} \;\equiv\; \mathbb{E}\{-l(\mathbf{x}, y)\} \;&=\; -\mathbb{E}\Big\{ \Big(\frac{1+y}{2}\Big) \ln \tilde{p}(\mathbf{x}) + \Big(\frac{1-y}{2}\Big) \ln\big(1-\tilde{p}(\mathbf{x})\big) \Big\} \\
&=\; -\mathbb{E}\Big\{ \Big(\frac{1+y}{2}\Big) \big(H(\mathbf{x}) - \ln(e^{-H(\mathbf{x})} + e^{H(\mathbf{x})})\big) \\
&\qquad + \Big(\frac{1-y}{2}\Big) \big(-H(\mathbf{x}) - \ln(e^{-H(\mathbf{x})} + e^{H(\mathbf{x})})\big) \Big\} \\
&=\; \mathbb{E}\{\ln(e^{-H(\mathbf{x})} + e^{H(\mathbf{x})}) - yH(\mathbf{x})\} \\
&=\; \mathbb{E}\Big\{ \ln\Big(\frac{e^{-yH(\mathbf{x})} + e^{yH(\mathbf{x})}}{e^{yH(\mathbf{x})}}\Big) \Big\} \\
&=\; \mathbb{E}\{\ln(1 + e^{-2yH(\mathbf{x})})\}
\end{aligned}$$

Solving for optimal H^*:

$$\frac{\partial \mathcal{L}}{\partial H} = 0 \;=\; \int_{\mathbf{x}} P(\mathbf{x}, y = +1) \Big(\frac{-2e^{-2H^*(\mathbf{x})}}{1 + e^{-2H^*(\mathbf{x})}}\Big) \;+\; \int_{\mathbf{x}} P(\mathbf{x}, y = -1) \Big(\frac{2e^{2H^*(\mathbf{x})}}{1 + e^{2H^*(\mathbf{x})}}\Big)$$

$$\therefore \quad \int_{\mathbf{x}} \frac{P(\mathbf{x}, y = +1)}{1 + e^{2H^*(\mathbf{x})}} = \int_{\mathbf{x}} \frac{P(\mathbf{x}, y = -1)}{1 + e^{-2H^*(\mathbf{x})}}$$

$$\therefore \quad H^*(\mathbf{x}) \;=\; \frac{1}{2}\ln\Big(\frac{P(\mathbf{x}, y=+1)}{P(\mathbf{x}, y=-1)}\Big) \;=\; \frac{1}{2}\ln\Big(\frac{P(y=+1\,|\,\mathbf{x})}{P(y=-1\,|\,\mathbf{x})}\Big)$$

Thus, the optimal confidence function H^* is half the log-ratio of the intrinsic posterior probabilities (also known as the *logit*), and plugging it in to Eq. 7.1, the optimal value of our estimator $\tilde{p}$ is indeed the intrinsic posterior that we seek.

Since $H \in \overline{\mathbb{R}}$, an additive model is a fitting choice (no pun intended):

$$H(\mathbf{x}) \;\equiv\; \sum_{t=1}^{T} \alpha_t f_t(\mathbf{x}) \qquad \text{where:}\;\; \alpha_t \in \overline{\mathbb{R}}, \;\; f_t : \mathcal{X} \to \{\pm 1\}$$

By fixing all model parameters from the first I iterations, we can greedily optimize the $I{+}1^{\text{th}}$ parameters as follows:

$$(\alpha_{I+1}, f_{I+1}) = \arg\min_{\alpha, f}\{\mathcal{L}_{I+1}\} = \arg\min_{\alpha, f}\Big\{\, \mathbb{E}\{\ln\big(1 + e^{-2yH_I(\mathbf{x})}\, e^{-2y\alpha f(\mathbf{x})}\big)\}\,\Big\}$$

The above procedure is aptly named LogitBoost, but as it turns out, it does not yield a closed-form solution. However, we can use a clever surrogate function to approximate the negative log-likelihood of our initial model:

$$\mathcal{L}' \;\equiv\; \mathbb{E}\{e^{-yH(\mathbf{x})}\} \;\approx\; \mathbb{E}\{\ln(1 + e^{-2yH(\mathbf{x})})\}$$

This surrogate function is the basis of AdaBoost, yielding a closed-form solution as shown in Sec. 2.2. Furthermore, the solution for optimal H^* remains unchanged:

$$\frac{\partial \mathcal{L}'}{\partial H} \;=\; 0 \;\;=\;\; -\int_{\mathbf{x}} P(\mathbf{x}, y=+1)\, e^{-H^*(\mathbf{x})} \;+\; \int_{\mathbf{x}} P(\mathbf{x}, y=-1)\, e^{H^*(\mathbf{x})}$$

$$\therefore \quad H^*(\mathbf{x}) \;=\; \ln\Big(\frac{P(\mathbf{x}, y=+1)}{P(\mathbf{x}, y=-1)}\Big) \;=\; \ln\Big(\frac{P(y=+1\,|\,\mathbf{x})}{P(y=-1\,|\,\mathbf{x})}\Big)$$

7.2 QuickBoost via Information Gain, Gini Impurity, and Variance

In Ch. 3, we prove a bound on the misclassification error on a preliminary subset of the data, given as Proposition 1:

$$r \leq m \quad \Rightarrow \quad Z_r \varepsilon_r \leq Z_m \varepsilon_m$$

Using this bound, we are able to prune features early and speed up the training of decision trees using the classification error criterion. In this document, we prove the same bound on other common types of stump splitting criteria (information gain, Gini purity, and variance minimization), extending our method for use with regression trees as well as binary or multi-class classification trees.

In a decision tree, an input propagates down the tree (based on the tree parameters) until it reaches a single leaf node. The optimal tree parameters are those that lead to the best error based on the training data. Recall that an m-subset is the set of m datapoints with the largest weights. We define ρ_j to be the set of elements in the m-subset that are assigned to leaf j using the optimal tree parameters (when trained on the n-subset). We define the sum of the weights of elements that belong to that leaf j as Z_{ρ_j} and the sum of the weights of elements in ρ_j with class y as $Z^y_{\rho_j}$

$$Z_m \equiv \sum_{n \leq m} w_n \qquad Z_{\rho_j} \equiv \sum_{n \in \rho_j} w_n \qquad Z^y_{\rho_j} \equiv \sum_{n \in \rho_j} w_n \, \mathbb{1}(y_n = y)$$

In regression, elements have values $\mathbf{y}_i \in \mathbb{R}^d$ and we define the weighted average value of a leaf as:

$$\tilde{\mathbf{y}}_{\rho_j} \equiv \frac{1}{Z_{\rho_j}} \sum_{n \in \rho_j} w_n \, \mathbf{y}_n$$

Given an m-subset of data (with $m \leq N$), the preliminary error is computed by summing the error in each leaf, proportionally weighted by the total mass of samples belonging to that leaf:

$$\varepsilon_m = \sum_j \frac{Z_{\rho_j}}{Z_m} \varepsilon_{\rho_j} \quad \text{which we reformulate as:} \quad Z_m \varepsilon_m = \sum_j Z_{\rho_j} \varepsilon_{\rho_j}$$

Our goal is to show that for each leaf j, the product of preliminary error and subset mass always exceeds that of a smaller subset. Let u_j be the elements in ρ_j that are in an r-subset and $\bar{u}_j$ be the elements that are *not*, where $r \leq m$:

$$u_j \equiv \{n \,|\, n \in \rho_j, n \leq r\}, \quad \bar{u}_j \equiv \{n \,|\, n \in \rho_j, n > r\} \qquad \text{[Note that: } \; u_j \cup \bar{u}_j = \rho_j \text{]}$$

$$Z_{u_j} \equiv \sum_{n \in u_j} w_n \quad Z^y_{u_j} \equiv \sum_{n \in u_j} w_n \, \mathbb{1}(y_n = y) \qquad Z_{\bar{u}_j} \equiv \sum_{n \in \bar{u}_j} w_n \quad Z^y_{\bar{u}_j} \equiv \sum_{n \in \bar{u}_j} w_n \, \mathbb{1}(y_n = y)$$

The optimal tree parameters for the r-subset might be different than those for the m-subset; hence, the use of potentially sub-optimal parameters may lead to a worse error. Accordingly, in the following section, we finalize our proof by showing: $Z_{\rho_j}\varepsilon_{\rho_j} \geq Z_{u_j}\varepsilon_{u_j} \; \forall j$

$$\Rightarrow \quad Z_m\varepsilon_m = \sum_j Z_{\rho_j}\varepsilon_{\rho_j} \geq \sum_j Z_{u_j}\varepsilon_{u_j} \geq Z_r\varepsilon_r$$

Q.E.D.

Note that this proof applies to trees of any depth and any number of leaves, not just binary stumps.

The following proofs are based on inequalities which are given at the end.

Misclassification

$$\varepsilon_m \equiv \sum_j \frac{Z_{\rho_j}}{Z_m}\Big(1 - \frac{Z^{y_j}_{\rho_j}}{Z_{\rho_j}}\Big) \qquad \therefore \quad Z_m\varepsilon_m = \sum_j \overbrace{(Z_{\rho_j} - Z^{y_j}_{\rho_j})}^{Z_{\rho_j}\varepsilon_{\rho_j}}$$

$$Z_{\rho_j}\varepsilon_{\rho_j} = Z_{\rho_j} - Z^{y_j}_{\rho_j} = (Z_{u_j} + Z_{\bar{u}_j}) - (Z^{y_j}_{u_j} + Z^{y_j}_{\bar{u}_j}) = \underbrace{Z_{u_j} - Z^{y_j}_{u_j}}_{Z_{u_j}\varepsilon_{u_j}} + \underbrace{Z_{\bar{u}_j} - Z^{y_j}_{\bar{u}_j}}_{Z_{\bar{u}_j}\varepsilon_{\bar{u}_j} \geq 0}$$

Information Gain

$$\varepsilon_m \equiv \sum_j \frac{Z_{\rho_j}}{Z_m}\Big(-\sum_y \frac{Z^y_{\rho_j}}{Z_{\rho_j}} \ln\Big(\frac{Z^y_{\rho_j}}{Z_{\rho_j}}\Big)\Big) \qquad \therefore \quad Z_m\varepsilon_m = \sum_j \overbrace{\Big(-\sum_y Z^y_{\rho_j} \ln\Big(\frac{Z^y_{\rho_j}}{Z_{\rho_j}}\Big)\Big)}^{Z_{\rho_j}\varepsilon_{\rho_j}}$$

$$Z_{\rho_j}\varepsilon_{\rho_j} = -\sum_y Z^y_{\rho_j} \ln\Big(\frac{Z^y_{\rho_j}}{Z_{\rho_j}}\Big) = -\sum_y (Z^y_{u_j} + Z^y_{\bar{u}_j}) \ln\Big(\frac{Z^y_{u_j} + Z^y_{\bar{u}_j}}{Z_{u_j} + Z_{\bar{u}_j}}\Big)$$

$$\geq \underbrace{\Big(-\sum_y Z^y_{u_j} \ln\Big(\frac{Z^y_{u_j}}{Z_{u_j}}\Big)\Big)}_{Z_{u_j}\varepsilon_{u_j}} + \underbrace{\Big(-\sum_y Z^y_{\bar{u}_j} \ln\Big(\frac{Z^y_{\bar{u}_j}}{Z_{\bar{u}_j}}\Big)\Big)}_{Z_{\bar{u}_j}\varepsilon_{\bar{u}_j} \geq 0}$$

The proof for Information Gain Ratio is a trivial adaptation of the proof above.

Gini Impurity

$$\varepsilon_m \equiv \sum_j \frac{Z_{\rho_j}}{Z_m} \sum_y \frac{Z^y_{\rho_j}}{Z_{\rho_j}} \left(1 - \frac{Z^y_{\rho_j}}{Z_{\rho_j}}\right) \qquad \therefore \quad Z_m \varepsilon_m = \sum_j \overbrace{\left(Z_{\rho_j} - \sum_y \frac{(Z^y_{\rho_j})^2}{Z_{\rho_j}}\right)}^{Z_{\rho_j}\varepsilon_{\rho_j}}$$

$$Z_{\rho_j}\varepsilon_{\rho_j} = Z_{\rho_j} - \sum_y \frac{(Z^y_{\rho_j})^2}{Z_{\rho_j}} = (Z_{u_j} + Z_{\bar{u}_j}) - \sum_y \frac{(Z^y_{u_j} + Z^y_{\bar{u}_j})^2}{Z_{u_j} + Z_{\bar{u}_j}}$$

$$\geq \underbrace{\left(Z_{u_j} - \sum_y \frac{(Z^y_{u_j})^2}{Z_{u_j}}\right)}_{Z_{u_j}\varepsilon_{u_j}} + \underbrace{\left(Z_{\bar{u}_j} - \sum_y \frac{(Z^y_{\bar{u}_j})^2}{Z_{\bar{u}_j}}\right)}_{Z_{\bar{u}_j}\varepsilon_{\bar{u}_j} \geq 0}$$

Variance Minimization

$$\varepsilon_m \equiv \sum_j \frac{Z_{\rho_j}}{Z_m} \sum_{n \in \rho_j} \frac{w_n}{Z_{\rho_j}} \|\mathbf{y}_n - \tilde{\mathbf{y}}_{\rho_j}\|^2 \qquad \therefore \quad Z_m \varepsilon_m = \sum_j \overbrace{\left(\sum_{n \in \rho_j} w_n \|\mathbf{y}_n\|^2 - \frac{\|Z_{\rho_j}\, \tilde{\mathbf{y}}_{\rho_j}\|^2}{Z_{\rho_j}}\right)}^{Z_{\rho_j}\varepsilon_{\rho_j}}$$

$$\begin{aligned} Z_{\rho_j}\varepsilon_{\rho_j} &= \sum_{n \in \rho_j} w_n \|\mathbf{y}_n\|^2 - \frac{\|Z_{\rho_j}\, \tilde{\mathbf{y}}_{\rho_j}\|^2}{Z_{\rho_j}} \\ &= \sum_{n \in u_j} w_n \|\mathbf{y}_n\|^2 + \sum_{n \in \bar{u}_j} w_n \|\mathbf{y}_n\|^2 - \frac{\|Z_{u_j}\, \tilde{\mathbf{y}}_{u_j} + Z_{\bar{u}_j}\, \tilde{\mathbf{y}}_{\bar{u}_j}\|^2}{(Z_{u_j} + Z_{\bar{u}_j})} \\ &\geq \underbrace{\left(\sum_{n \in u_j} w_n \|\mathbf{y}_n\|^2 - \frac{\|Z_{u_j}\, \tilde{\mathbf{y}}_{u_j}\|^2}{Z_{u_j}}\right)}_{Z_{u_j}\varepsilon_{u_j}} + \underbrace{\left(\sum_{n \in \bar{u}_j} w_n \|\mathbf{y}_n\|^2 - \frac{\|Z_{\bar{u}_j}\, \tilde{\mathbf{y}}_{\bar{u}_j}\|^2}{Z_{\bar{u}_j}}\right)}_{Z_{\bar{u}_j}\varepsilon_{\bar{u}_j} \geq 0} \end{aligned}$$

Inequalities

For positive scalars $a, b \geq 0$ and $\alpha, \beta > 0$, the following inequality holds:

$$\begin{aligned} (a+b)\ln\Big(\frac{a+b}{\alpha+\beta}\Big) &\leq a\ln\Big(\frac{a}{\alpha+\beta}\Big) + b\ln\Big(\frac{b}{\alpha+\beta}\Big) \\ &= a\ln\Big(\frac{a}{\alpha}\cdot\frac{\alpha}{\alpha+\beta}\Big) + b\ln\Big(\frac{b}{\beta}\cdot\frac{\beta}{\alpha+\beta}\Big) \\ &= a\ln\Big(\frac{a}{\alpha}\Big) + b\ln\Big(\frac{b}{\beta}\Big) - \underbrace{\Big(a\ln\Big(1+\frac{\beta}{\alpha}\Big) + b\ln\Big(1+\frac{\alpha}{\beta}\Big)\Big)}_{\geq 0} \end{aligned}$$

$$\therefore \quad (a+b)\ln\Big(\frac{a+b}{\alpha+\beta}\Big) \;\le\; a\ln\Big(\frac{a}{\alpha}\Big) + b\ln\Big(\frac{b}{\beta}\Big)$$

For any vectors (or scalars) $\mathbf{a}, \mathbf{b}$ and positive scalars $\alpha, \beta > 0$, the following inequality holds:

$$\begin{aligned}
\frac{\|\mathbf{a}+\mathbf{b}\|^2}{\alpha+\beta} &= \frac{\|\mathbf{a}\|^2 + \|\mathbf{b}\|^2}{\alpha+\beta} + \frac{2\langle \mathbf{a}, \mathbf{b}\rangle}{\alpha+\beta} \\
&= \frac{\|\mathbf{a}\|^2}{\alpha}\Big(1 - \frac{\beta}{\alpha+\beta}\Big) + \frac{\|\mathbf{b}\|^2}{\beta}\Big(1 - \frac{\alpha}{\alpha+\beta}\Big) + \frac{2\langle \mathbf{a}, \mathbf{b}\rangle}{\alpha+\beta} \\
&= \frac{\|\mathbf{a}\|^2}{\alpha} + \frac{\|\mathbf{b}\|^2}{\beta} - \underbrace{\frac{\|\beta\,\mathbf{a} - \alpha\,\mathbf{b}\|^2}{\alpha\beta(\alpha+\beta)}}_{\geq 0}
\end{aligned}$$

$$\therefore \quad \frac{\|\mathbf{a}+\mathbf{b}\|^2}{\alpha+\beta} \;\le\; \frac{\|\mathbf{a}\|^2}{\alpha} + \frac{\|\mathbf{b}\|^2}{\beta}$$

7.3 Reduction of REBEL to Binary AdaBoost

In Ch. 4, we propose a novel multi-class cost-sensitive boosting method, REBEL. In this section, we prove that REBEL reduces to binary boosting methods, depending on the convex upper-bounding function g. In the binary (two-class) cost-insensitive case, the surrogate loss reduces to:

$$\mathcal{L} = \frac{1}{2N}\sum_{y_n=1}\big(g(-H_{n1}) + g(H_{n2})\big) + \frac{1}{2N}\sum_{y_n=2}\big(g(H_{n1}) + g(-H_{n2})\big) \tag{7.2}$$

where we denote: $\displaystyle\sum_{y_n=k}(...) \equiv \sum_{n=1}^{N} \mathbb{1}_{(y_n=k)}(...)$ (i.e. summing over only class k)

and: $H_{n1} \equiv \langle \mathbf{H}_I(\mathbf{x}_n), \boldsymbol{\delta}_1 \rangle,\ H_{n2} \equiv \langle \mathbf{H}_I(\mathbf{x}_n), \boldsymbol{\delta}_2 \rangle,\ H^+_{n1} \equiv \langle \mathbf{H}_{I+1}(\mathbf{x}_n), \boldsymbol{\delta}_1 \rangle,\ H^+_{n2} \equiv \langle \mathbf{H}_{I+1}(\mathbf{x}_n), \boldsymbol{\delta}_2 \rangle$

Proposition: $H_{n1} = -H_{n2}\ \ \forall\, I$

Proof: (by induction)

Before any training: $\mathbf{H}_0(\mathbf{x}_n) \equiv \mathbf{0}\ \ \therefore\ \ H_{n1} = H_{n2} = 0\ \ \therefore\ \ H_{n1} = -H_{n2}\ \ \forall\, n$

Assume this holds for $\mathbf{H}_I(\mathbf{x})$. On the $I{+}1^{\text{th}}$ iteration, $\mathbf{H}_{I+1}(\mathbf{x}) = \mathbf{H}_I(\mathbf{x}) + f(\mathbf{x})\,\mathbf{a}$, hence:

$$\begin{aligned}
\mathcal{L}_{I+1} &= \frac{1}{2N}\Big(\sum_{y_n=1}\big(g(-H^+_{n1}) + g(H^+_{n2})\big) + \sum_{y_n=2}\big(g(H^+_{n1}) + g(-H^+_{n2})\big)\Big) \\
&= \frac{1}{2N}\sum_{y_n=1}\big(g(-H_{n1} - f(\mathbf{x}_n)a_1) + g(H_{n2} + f(\mathbf{x}_n)a_2)\big) \\
&+ \frac{1}{2N}\sum_{y_n=2}\big(g(H_{n1} + f(\mathbf{x}_n)a_1) + g(-H_{n2} - f(\mathbf{x}_n)a_2)\big)
\end{aligned}$$

Therefore, solving for optimal a_1 by setting: $\frac{\partial \mathcal{L}_{I+1}}{\partial a_1} = 0$

$$\therefore\ \sum_{y_n=1} f(\mathbf{x}_n)g'(-H_{n1} - f(\mathbf{x}_n)a_1) = \sum_{y_n=2} f(\mathbf{x}_n)g'(H_{n1} + f(\mathbf{x}_n)a_1) \tag{7.3}$$

Whereas solving for optimal a_2 by setting: $\frac{\partial \mathcal{L}_{I+1}}{\partial a_2} = 0$

$$\therefore\ \sum_{y_n=1} f(\mathbf{x}_n)g'(H_{n2} + f(\mathbf{x}_n)a_2) = \sum_{y_n=2} f(\mathbf{x}_n)g'(-H_{n2} - f(\mathbf{x}_n)a_2)$$

Since by assumption, $H_{n1} = -H_{n2}$, therefore:

$$\therefore\ \sum_{y_n=1} f(\mathbf{x}_n)g'\big(-H_{n1} - f(\mathbf{x}_n)(-a_2)\big) = \sum_{y_n=2} f(\mathbf{x}_n)g'\big(H_{n1} + f(\mathbf{x}_n)(-a_2)\big) \tag{7.4}$$

Note that if $a_1 = a$ solves Eq. 7.3, then $a_2 = -a$ solves Eq. 7.4; hence: $a_1 = -a_2$.

$$\therefore \quad H_{n1}^{+} = H_{n1} + f(\mathbf{x}_n)\, a_1 = -H_{n2} - f(\mathbf{x}_n)\, a_2 = -(H_{n2} + f(\mathbf{x}_n)\, a_2) = -H_{n2}^{+}$$

On the I+1$^{\text{th}}$ iteration, again: $H_{n1}^{+} = -H_{n2}^{+}$

Back to the original binary loss (Eq. 7.2):

$$\begin{aligned}\mathcal{L} &= \frac{1}{2N}\sum_{y_n=1}\big(g(-H_{n1}) + g(H_{n2})\big) + \frac{1}{2N}\sum_{y_n=2}\big(g(H_{n1}) + g(-H_{n2})\big) \\ &= \frac{1}{N}\Big(\sum_{y_n=1} g(-H_{n1}) + \sum_{y_n=2} g(H_{n1})\Big) = \frac{1}{N}\sum_{n=1}^{N} g(-y_n^{\star} H_{n1})\end{aligned}$$

$$\text{where: } \; y_n^{\star} \in \{+1, -1\} \;\; \text{for: } \; y_n \in \{1, 2\}$$

which is exactly the same form as the loss function for binary boosting. For instance, if $g(x) \equiv e^x$, REBEL reduces to AdaBoost, $g(x) \equiv \log_2(1 + e^x)$ reduces to LogitBoost, and $g(x) \equiv (1 + x)^2$ reduces to L_2Boost.

7.4 Unit Vector Bounds

Proposition 5.8: $\langle |\hat{\mathbf{v}}|, \mathbf{1} \rangle^2 \ \geq \ 1$

Proof: Reformulate as a constrained minimization problem, with $\mathbf{x} \in \mathbb{R}^N$:

$$\min_{\mathbf{x}} \{ \langle \mathbf{x}, \mathbf{1} \rangle \} \quad \text{such that:} \ \ \|\mathbf{x}\|^2 = 1, \ \mathbf{x} \geq \mathbf{0}$$

$$\therefore \ \ L = \langle \mathbf{x}, \mathbf{1} \rangle - \lambda(\|\mathbf{x}\|^2 - 1) - \sum_{n=1}^{N} \mu_n (\langle \mathbf{x}, \boldsymbol{\delta}_n \rangle - 0) \qquad \big[\mu_n \geq 0 \ \forall \, n\big]$$

$$\therefore \ \ \nabla_{\mathbf{x}} L = \mathbf{1} - 2\lambda \, \mathbf{x} - \sum_{n=1}^{N} \mu_n \, \boldsymbol{\delta}_n$$

$$\therefore \ \ 2\lambda \, \mathbf{x}^* = \sum_{n=1}^{N} (1 - \mu_n) \, \boldsymbol{\delta}_n \qquad \therefore \ \ \mathbf{x}^* \ = \ \frac{\sum\limits_{n=1}^{N} (1 - \mu_n) \, \boldsymbol{\delta}_n}{\sqrt{\sum\limits_{n=1}^{N} (1 - \mu_n)^2}} \qquad \big[\mu_n \leq 1 \ \forall \, n\big]$$

$$\therefore \ \ \langle \mathbf{x}^*, \mathbf{1} \rangle \ = \ \frac{\sum\limits_{n=1}^{N} (1 - \mu_n)}{\sqrt{\sum\limits_{n=1}^{N} (1 - \mu_n)^2}} \ \geq \ \frac{\sum\limits_{n=1}^{N} (1 - \mu_n)^2}{\sqrt{\sum\limits_{n=1}^{N} (1 - \mu_n)^2}} \ = \ \sqrt{\sum_{n=1}^{N} (1 - \mu_n)^2}$$

To have unit norm, $\mathbf{x}$ must contain at least one non-zero element. Thus, without loss of generality, we assume $x_1 > 0$; hence: $\mu_1 = 0$

$$\therefore \ \ \langle \mathbf{x}^*, \mathbf{1} \rangle \ \geq \ \sqrt{1 + \sum_{n=2}^{N} (1 - \mu_n)^2} \ \geq \ 1$$

Q.E.D.

Proposition 5.9: $\max_i\{\langle \mathbf{1}-2\boldsymbol{\delta}_i,\hat{\mathbf{v}}\rangle^2\} \;\geq\; \dfrac{4}{N}$ for $N \geq 4$

Proof: Reformulate as a constrained minimization problem with $\mathbf{x} \in \mathbb{R}^N$. Without loss of generality, assume that $\langle \mathbf{x}, \mathbf{1}\rangle \geq 0$ and that its first element x_1 is a minimal element (i.e. $x_1 \leq x_n \;\forall\, n$).

$$\min_{\mathbf{x}}\{\langle \mathbf{x}, \mathbf{1}-2\boldsymbol{\delta}_1\rangle\} \quad \text{such that: } \|\mathbf{x}\|^2 = 1, \;\; \mathbf{x} \geq x_1\,\mathbf{1}$$

$$\therefore \;\; L = \langle \mathbf{x}, \mathbf{1}-2\,\boldsymbol{\delta}_1\rangle - \lambda(\|\mathbf{x}\|^2 - 1) - \sum_{n=2}^{N} \mu_n(\langle \mathbf{x}, \boldsymbol{\delta}_n\rangle - x_1) \qquad \big[\mu_n \geq 0 \;\forall\, n\big]$$

$$\therefore \;\; \nabla_{\mathbf{x}} L = [\mathbf{1}-2\boldsymbol{\delta}_1] - 2\lambda\,\mathbf{x} - \sum_{n=2}^{N} \mu_n\,\boldsymbol{\delta}_n$$

$$\therefore \;\; 2\lambda\,\mathbf{x}^* = -\boldsymbol{\delta}_1 + \sum_{n=2}^{N}(1-\mu_n)\,\boldsymbol{\delta}_n \qquad \therefore \;\; \mathbf{x}^* \;=\; \frac{-\boldsymbol{\delta}_1 + \sum_{n=2}^{N}(1-\mu_n)\,\boldsymbol{\delta}_n}{\sqrt{1 + \sum_{n=2}^{N}(1-\mu_n)^2}}$$

$$\therefore \;\; \langle \mathbf{x}^*, \mathbf{1}-2\boldsymbol{\delta}_1\rangle \;=\; \frac{1 + \sum_{n=2}^{N}(1-\mu_n)}{\sqrt{1 + \sum_{n=2}^{N}(1-\mu_n)^2}}$$

Note that if $x_n > x_1$ then $\mu_n = 0$, and if $x_n = x_1$ then $(1-\mu_n) = -1$.

Let M be the number of unique indices $n \geq 2$ for which $x_n = x_1$.

$$\therefore \;\; \langle \mathbf{x}^*, \mathbf{1}-2\boldsymbol{\delta}_1\rangle \;=\; \frac{1 + \big((N-1) - M\big) - M}{\sqrt{N}} \;=\; \frac{N - 2M}{\sqrt{N}}$$

Since $\langle \mathbf{x}, \mathbf{1}\rangle \geq 0$, hence: $-1 + \big((N-1) - M\big) - M \geq 0 \;\;\therefore\; -2M \geq 2-N$

$$\therefore \;\; \langle \mathbf{x}^*, \mathbf{1}-2\boldsymbol{\delta}_1\rangle \;\geq\; \frac{2}{\sqrt{N}} \qquad \therefore \;\; \langle \mathbf{x}^*, \mathbf{1}-2\boldsymbol{\delta}_1\rangle^2 \;\geq\; \frac{4}{N}$$

Q.E.D.