ELEC-E5430 Large-Scale Data Analysis

Esa Ollila

Contents

Chapter 1

Introduction

We have a set of *input* variables (or *features*)

$$X = (X_1, \ldots, X_p)$$

that are used to predict the *output* (or *outcome* or *response*) variable Y. In a typical scenario, we have an outcome measurement, usually quantitative (such as a stock price) or categorical (such as disease/no disease), that we wish to predict based on a set of features (such as diet or clinical measurements).

We assume there is an underlying function $f(\cdot)$ that captures the input-output relationship which we would like to estimate. We do not know f, but we get to observe example input-output pairs. This is the so called *supervised learning* problem, where a *training data*

$$\mathcal{T} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$$

is available for estimating f. Often f is parametrized, so we only need to learn a vector parameter that determines the function f.

The training data pair $(\mathbf{x}_i, y_i)$ is assumed to be generated at random via one of the following mechanism:

1. $(\mathbf{x}_i, y_i)$ is a realisation from an unknown $p+1$ variate joint distribution of (X, Y)
2. each input $\mathbf{x}_i$ is drawn independently from some unknown distribution and we get to observe the pair $(\mathbf{x}_i, f(\mathbf{x}_i) + \varepsilon_i)$, where ε_i represents a random error term.

Many machine learning problems can be posed as classification or regression tasks. These differ in output types and consequently in the prediction tasks (prediction or classification).

We use uppercase letters such as X or Y when referring to an abstract variable or when viewing X as a random variable or random vector. For example $X_j \in \mathbb{R}$ can represent a feature such as *Height of a person* (in a specific population), but $x_j \in \mathbb{R}$ is its observed value. Similarly $X \in \mathbb{R}^p$ can represent a vector of feature variables, and $\mathbf{x} = (x_1, \ldots, x_p)^\top \in \mathbb{R}^p$ observation of these variables for one measurement instance.

Thus for observed values (or non-random data points) we use the notations:

- *matrices* are represented by bold uppercase letters; E.g., $\mathbf{X} \in \mathbb{R}^{N \times p}$ denotes a fixed (known) $N \times p$ matrix.
- *vectors* are represented by bold lowercase letters. E.g., $\mathbf{x}_i \in \mathbb{R}^p$ denotes a fixed (known) $p \times 1$ vector. By a vector we always refer to a column vector.

The set of N input (observed) p-vectors $\mathbf{x}_i$, $i = 1, \ldots, N$, in the training data are often collected to an $N \times p$ matrix:

$$\begin{aligned} \mathbf{X} &= \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1p} \\ x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N1} & x_{N2} & \cdots & x_{Np} \end{pmatrix} = \begin{pmatrix} \mathbf{x}_1^\top \\ \mathbf{x}_2^\top \\ \vdots \\ \mathbf{x}_N^\top \end{pmatrix} \\ &= \begin{pmatrix} \boldsymbol{x}_1 & \boldsymbol{x}_2 & \cdots & \boldsymbol{x}_p \end{pmatrix} \end{aligned}$$

This convention distinguishes a p-vector of inputs $\mathbf{x}_i \in \mathbb{R}^p$ for the ith observation from the N-vectors $\boldsymbol{x}_j$ consisting of all the observations on variable (feature) X_j (e.g., "height"). Since all vectors are assumed to be column vectors, the ith row of $\mathbf{X}$ is $\mathbf{x}_i^\top$, the vector transpose of $\mathbf{x}_i = (x_{i1}, \ldots, x_{ip})^\top$. We collect the outputs, y_i, into a single vector

$$\mathbf{y} = (y_1, \ldots, y_N)^\top.$$

1.1 Classification task

Classification is a problem where two or more populations are known a priori and one or more new observations are classified into one of the known populations (classes) based on the measured characteristics. It is assumed that there are known number $K \geq 2$ *classes* (or populations or states), the output Y taking values in a finite set $\mathcal{G} = \{\mathcal{G}_1, \ldots, \mathcal{G}_K\}$. We would like to predict qualitative (categorical) outputs $Y \in \mathcal{G}$ representing class labels

Classification task can be often presented as a problem of partitioning the input vector space into disjoint (decision) regions. The task is to find the function (*discriminant rule*) $G(X) \mapsto Y$ that maps the inputs most accurately to their corresponding class labels.

It will be more convenient to encode the labels of the input variable Y with numeric values. We often use the convention

$$\mathcal{G} = \{1, \ldots, K\}$$

in the general $K > 2$ case and

$$\mathcal{G} = \{1, 2\} \quad \text{or} \quad \mathcal{G} = \{-1, 1\}$$

in the two-class ($K = 2$) case. Such numeric codes are sometimes referred to as *targets*.

Training data $\mathcal{T}$ consists of observations $\mathbf{x}_i \in \mathbb{R}^p$ from one of the K populations and $y_i \in \mathcal{G} = \{1, 2, \ldots, K\}$ is the associated known class label of the i^{th} observation. The goal in the classification task is to assign a new observation $\mathbf{x} = (x_1, \ldots, x_p)^\top \in \mathbb{R}^p$ to one of the $K \geq 2$ classes as accurately as possible.

A formal definition of a discriminant rule is given below.

Definition 1.1. A *discriminant rule* or *classifier* is a function $G(\mathbf{x}) : \mathbb{R}^p \to \mathcal{G} = \{1, \ldots, K\}$, i.e., a function that takes a data vector and returns a class label. Discriminant rule partitions the input space $\mathbb{R}^p$ into K *decision regions*,

$$\Gamma_k \equiv \Gamma_k(G) = \big\{\mathbf{x} \in \mathbb{R}^p : G(\mathbf{x}) = k\big\}$$

which are mutually disjoint and exhaustive sets, i.e.,

$$\Gamma_k \cap \Gamma_j = \emptyset \ \ \forall k \neq j \quad \text{and} \quad \bigcup_{k=1}^{K} \Gamma_k = \mathbb{R}^p.$$

Equivalently, $G(\mathbf{x})$ can be formulated as

$$G(\mathbf{x}) = \underset{k \in \{1,\ldots,K\}}{\arg\max}\ G_k(\mathbf{x}) \tag{1.1}$$

where $G_k(\mathbf{x}) \in \mathbb{R}$ is a *discriminant score* of $\mathbf{x}$ for class k, and

$$\Gamma_k = \big\{\mathbf{x} \in \mathbb{R}^p : G_k(\mathbf{x}) > G_j(\mathbf{x}) \quad \forall k \neq j \in \{1, \ldots, K\}\big\}$$

i.e., class k is chosen if it has the greatest discriminant score (and decide ties by random guesses).

In the binary classification problem ($K = 2$) and dimension $p = 2$, the Figure 1.1 illustrate the division of the sample space $\mathbb{R}^p$ into two disjoint decision regions.

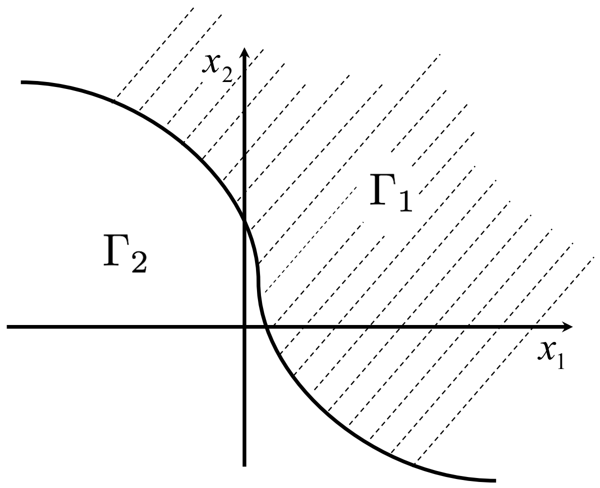


FIGURE 1.1: *Two grous $K = 2$ and $\mathbf{x} \in \mathbb{R}^2$ case: a discriminant rule distinguishes $\mathbb{R}^2$ into two disjoint decision regions Γ_1 and Γ_2 ($\mathbb{R}^2 = \Gamma_1 \cup \Gamma_2$).*

1.2 Regression (prediction) task

Refers to the case when we predict quantitative output $Y \in \mathbb{R}$, referred to as *response* variable, given the input $X = (X_1, \ldots, X_p)^T$ where $X_1, \ldots, X_p$ are referred to as *predictor* variables or *features*. The task is to find the predictor function $f(X) \mapsto Y$ that maps the inputs $X = (X_1, \dots, X_p)^\top$ most accurately to their corresponding outputs Y (for example by minimizing the MSE for training data: $\sum_{i=1}^N (y_i - f(\mathbf{x}_i))^2$). In high-dimensional cases, feature selection, so selecting significant features and getting rid of noisy features is essential part of the problem.

The regression model is

$$Y = f(X) + \varepsilon,$$

where ε is the zero mean random error term that account for modelling and measurement errors, and the predictor function is

$$f(X) = g\Big(\beta_0 + \sum_{j=1}^p X_j \beta_j\Big), \tag{1.2}$$

where $\boldsymbol{\beta} = (\beta_1, \dots, \beta_p)^\top \in \mathbb{R}^p$ is an unknown vector of *regression coefficients*, $\beta_0 \in \mathbb{R}$ is the *intercept* and g is a fixed *link function* such as linear function $g(x) = x$, yielding the *linear regression model.*

$$f(X) = \beta_0 + \sum_{j=1}^p X_j \beta_j. \tag{1.3}$$

Thus we observe the pairs $(\mathbf{x}_i, y_i)$ where $y_i = f(\mathbf{x}_i) + \varepsilon_i$, where $\{\varepsilon_i\}_{i=1}^N$ are i.i.d. unobserved zero mean random errors. Then using the training data $\mathcal{T} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$, the aim is to estimate the unknown regression coefficients $(\beta_0, \boldsymbol{\beta}) \in \mathbb{R} \times \mathbb{R}^p$ for a given function f.

Q: Let response Y be the house price of an apartment in Helsinki. What would be useful predictors $X_1, \dots, X_p$ in the linear regression model?

In terms of the training data, the linear input-output relationship can be represented by a set of N equations

$$\begin{aligned} y_1 &= \beta_0 + x_{11}\beta_1 + \dots + x_{1p}\beta_p + \varepsilon_1 \\ &\vdots \qquad\qquad\quad \vdots \\ y_N &= \beta_0 + x_{N1}\beta_1 + \dots + x_{Np}\beta_p + \varepsilon_N \end{aligned}$$

where ε_i-s represent i.i.d. random variables from the noise distribution ε. These are can be expressed using matrix-vector notations as

$$\begin{aligned} \mathbf{y} &= \beta_0 \mathbf{1} + \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon} \qquad (1.4) \\ &= \beta_0 \mathbf{1} + \boldsymbol{\beta}_1 \boldsymbol{x}_1 + \dots + \boldsymbol{\beta}_p \boldsymbol{x}_p + \boldsymbol{\varepsilon}. \end{aligned}$$

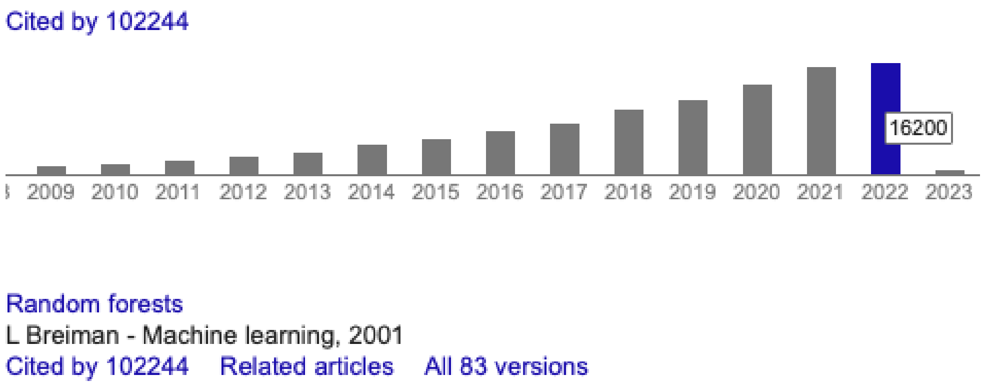
1.3 Discussion

The aim of these chapters are to provide a brief introduction to methods that are proven to be powerful in supervised learning tasks. Most data analysis languages such as python have put a lot of effort to provide powerful and computationally efficient implementations of these methods. We focus on 3 powerful ideas in supervised learning: a) Decision trees, Bagging and Random Forests, b) Boosting, and c) Lasso regression and its extensions.

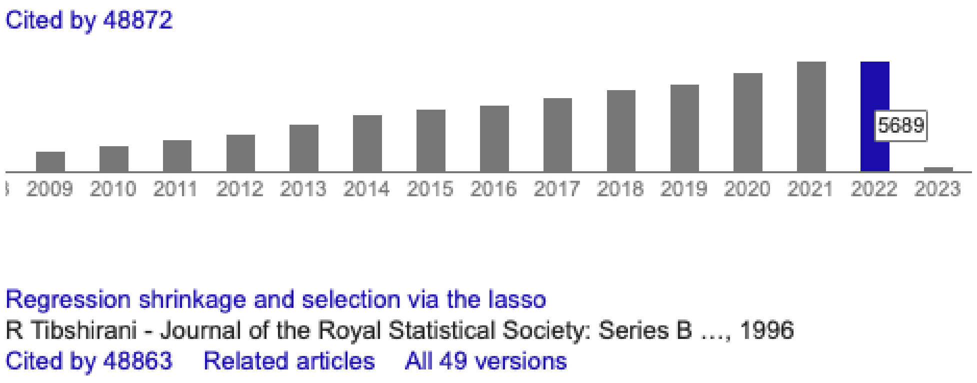
Paper	citations	citations / year
Lasso [Tibshirani, 1996]	48872	1879
Decision Trees [Breiman et al., 1984]	58493	1551
Random forests [Breiman, 2001]	102244	4868
Boosting [Freund and Schapire, 1997]	24957	998
Gradient Boosting [Friedman, 2001]	20429	972

TABLE 1.1: *Methods of the chapters. (Citation count: Jan 8, 2023)*

Citations time series for random forest and lasso (Jan 8, 2023):



Random forests
L Breiman - Machine learning, 2001
Cited by 102244 Related articles All 83 versions



Regression shrinkage and selection via the lasso
R Tibshirani - Journal of the Royal Statistical Society: Series B …, 1996
Cited by 48863 Related articles All 49 versions

Chapter 2

Classification: basic concepts

This chapter focuses on building the basic concepts of classification and the optimal Bayesian discriminant rule. The chapter covers Sections 10.5 and Section 10.6 from [Hastie et al., 2009] as well as some selected topics from Chapter 2, such as Section 2.4.

2.1 Optimal classifier

We assume that the joint $(p+1)$-variate distribution of input-output pair (Y, X) is known. The class label $Y \in \mathcal{G} = \{1, \ldots, K\}$ is a discrete random variable with a probability mass function (p.m.f.)

$$\pi_k = \Pr(Y = k) = \begin{cases} \text{"probability that a randomly selected} \\ \text{observation is from class } k\text{"} \end{cases}$$

These are the *a priori class probabilities* ($\sum_i \pi_i = 1$) which are assumed to be known. The prior probabilities can be significantly different between the classes.

For simplicity of exposition, assume that the p-variate feature vector contains measurements on a continuous p-variate random vector $X = (X_1, \ldots, X_p)$. The class conditional probability density function (p.d.f.)

$$f_{X|Y}(\mathbf{x} \mid k), \quad k \in \mathcal{G} \tag{2.1}$$

is the distribution of X when it is from class k. Note also that

$$\Pr(X \in \mathcal{X} \mid Y = k) = \int_{\mathcal{X}} f_{X|Y}(\mathbf{x} \mid k) \mathrm{d}\mathbf{x}.$$

The data conditional probability, i.e., the *a posteriori class probabilities*, are

$$p_k(\mathbf{x}) = \Pr(Y = k \mid X = \mathbf{x}) = \frac{f_{X|Y}(\mathbf{x} \mid k) \Pr(Y = k)}{f_X(\mathbf{x})} = \frac{f_{X|Y}(\mathbf{x} \mid k)\pi_k}{\sum_{k=1}^{K} f_{X|Y}(\mathbf{x} \mid k)\pi_k} \tag{2.2}$$

Note that $p_k(\mathbf{x}) \in [0, 1]$ and

$$\sum_{k \in \mathcal{G}} p_k(\mathbf{x}) = 1. \tag{2.3}$$

It is customary to take logarithm of p.d.f.-s as it often simplifies expressions. We express the log-posterior by

$$\log p_k(\mathbf{x}) = \ln f_{X|Y}(\mathbf{x} \mid k) + \ln \pi_k, \tag{2.4}$$

where we ignore the irrelevant additive constant ($= -\ln f_X(\mathbf{x})$) that does not depend on k.

The *classification loss* or *0-1 loss* is defined as

$$L_{0/1}(y, G(\mathbf{x})) = 1_{\{y \neq G(\mathbf{x})\}} \tag{2.5}$$

which is equal to 1 if the classifier G misclassifies $(\mathbf{x}, y)$, and 0 otherwise. Since we assume a full knowledge of the joint distribution, we can design an optimal classifier, which is the one that minimizes the risk (probability of error).

Definition 2.1. The (Bayes) *risk* of a discriminant rule G is

$$\mathrm{r}(G) = \Pr(G(X) \neq Y) = \mathbb{E}\big[1_{\{Y \neq G(X)\}}\big] \tag{2.6}$$

and it equals the probability that the rule makes an error (or the expected classification loss).

The risk measures how the estimated rule performs on average. It can also be written as

$$\mathrm{r}(G) = \sum_{k \in \mathcal{G}} \mathbb{E}_X\big[1_{\{G(X) \neq k\}} \big| Y = k\big]\pi_k = \sum_{k \in \mathcal{G}} \Pr(X \notin \Gamma_k(G) \mid Y = k)\pi_k \tag{2.7}$$

where $\Gamma_k(G)$ denote the classification regions (cf. Definition 1.1). Its empirical version, computed on the training sample,

$$\hat{\mathrm{r}}_N(G) = \frac{1}{N}\sum_{i=1}^{N} L_{0/1}(y_i, G(\mathbf{x}_i)) = \frac{1}{N}\sum_{i=1}^{N} 1_{\{y_i \neq G(\mathbf{x}_i)\}} \tag{2.8}$$

is called *empirical risk.*

The Bayes classifier is a rule that maximizes the posterior probability (2.2).

Definition 2.2. The *Bayes classifier* G^* assigns $\mathbf{x}$ to class k^* having maximum a posteriori probability, $G^*(\mathbf{x}) = k^*$, where[1]

$$\begin{aligned} k^* &= \arg\max_{k \in \mathcal{G}} p_k(\mathbf{x}) = \arg\max_{k \in \mathcal{G}} f_{X|Y}(\mathbf{x} \mid k)\pi_k \\ &= \arg\max_{k \in \mathcal{G}} \{\ln f_{X|Y}(\mathbf{x} \mid k) + \ln \pi_k\}, \end{aligned} \tag{2.9}$$

or in other words

$$p_{k^*}(\mathbf{x}) \geq p_j(\mathbf{x}) \;\Leftrightarrow\; \pi_{k^*} f_{X|Y}(\mathbf{x} \mid k^*) \geq \pi_j f_{X|Y}(\mathbf{x} \mid j) \qquad \forall k^* \neq j.$$

[1]Since the marginal distribution $f_X(\mathbf{x})$ of X (i.e., the unconditional likelihood over all populations that we could observe $\mathbf{x}$) is irrelevant constant that just scales the posterior probabilities in (2.2), and therefore, we can ignore it in (2.9)

Bayes rule is also the optimal classification rule:

Theorem 2.1. *The Bayes rule G^* is optimal in the sense of minimizing the error rate (risk), i.e., it verifies*

$$\mathrm{r}(G) \geq \mathrm{r}(G^*)$$

where $G(\cdot)$ can be any classifier.

Remark 2.1. In the binary classification problem ($K = 2$, $\mathcal{G} = \{0,1\}$), the Bayes classifier can be simply written as

$$G^*(\mathbf{x}) = \begin{cases} \text{Class } 0, & \text{if } p_1(\mathbf{x}) - p_0(\mathbf{x}) < 0 \\ \text{Class } 1, & \text{if } p_1(\mathbf{x}) - p_0(\mathbf{x}) > 0 \end{cases} \tag{2.10}$$

$$= \begin{cases} \text{Class } 0, & \text{if } p_1(\mathbf{x}) - \frac{1}{2} < 0 \\ \text{Class } 1, & \text{if } p_1(\mathbf{x}) - \frac{1}{2} > 0. \end{cases} \tag{2.11}$$

The case of ties, so when an observation falls in the decision boundary

$$f(\mathbf{x}) = p_1(\mathbf{x}) - p_0(\mathbf{x}) = p_1(\mathbf{x}) - 1/2 = 0$$

can be handled by a coin flip. Note that in the last identity we used the property that $p_0(\mathbf{x}) + p_1(\mathbf{x}) = 1$. Thus, encoding the classes as $\mathcal{G} = \{-1, 1\}$ instead of $\mathcal{G} = \{0,1\}$ allows us to write the Bayes rule concisely as

$$G^*(\mathbf{x}) = \mathsf{sign}[f(\mathbf{x})] \quad \text{with} \quad f(\mathbf{x}) = p_1(\mathbf{x}) - \frac{1}{2}. \tag{2.12}$$

Remark 2.2. When $K = 2$ and $\mathcal{G} = \{0,1\}$, we may express the decision regions as

$$\Gamma_0^* = \left\{ \mathbf{x} : \frac{f_{X|Y}(\mathbf{x} \mid 0)}{f_{X|Y}(\mathbf{x} \mid 1)} > \frac{\pi_1}{\pi_0} \right\} \qquad \text{and} \qquad \Gamma_1^* = \left\{ \mathbf{x} : \frac{f_{X|Y}(\mathbf{x} \mid 0)}{f_{X|Y}(\mathbf{x} \mid 1)} < \frac{\pi_1}{\pi_0} \right\}$$

and handling ties as random guessing. The detection rule can thus be expressed as

$$L(\mathbf{x}) = \frac{f_{X|Y}(\mathbf{x} \mid 0)}{f_{X|Y}(\mathbf{x} \mid 1)} \underset{1}{\overset{0}{\gtrless}} \frac{\pi_1}{\pi_0}$$

and notice that $L(\mathbf{x})$ is the *likelihood ratio.* If the a priori probabilities are identical, $\pi_0 = \pi_1 = 1/2$, then the decision regions are completely based on comparing the likelihoods of $\mathbf{x}$ for each class.

Example 2.1. Assume $K = 2$ classes with class conditional distributions following exponential distributions:

$$X|Y = 0 \sim \mathrm{Exp}(\lambda_0) \qquad \text{and} \qquad X|Y = 1 \sim \mathrm{Exp}(\lambda_1)$$
$$f_{X|Y}(x \mid 0) = \lambda_0 \exp\{-\lambda_0 x\} \qquad \text{and} \qquad f_{X|Y}(x \mid 1) = \lambda_1 \exp\{-\lambda_1 x\},\ x > 0$$

where $\lambda_k > 0$ is the rate parameter, $\lambda_k^{-1} = \mathbb{E}[X \mid Y = k]$, $k \in \{0,1\}$. Without loss of generality, assume $\lambda_0 < \lambda_1$ (this can be always done by simply relabelling the classes).

a) Derive the classification region Γ_0^* (that minimize the Bayes risk) in the case of uniform prior probabilities ($\pi_0 = \pi_1 = 1/2$).

b) Calculate the risk (so the probability of an error) $\mathsf{r}(G^*) = \Pr(Y \neq G^*(X))$ of this optimal Bayes classifier when $\lambda_0 = 1$ and $\lambda_1 = 3$.

If time permits, we go though this example on lectures... ■

2.2 Classification costs and Bayes risk

Discriminant rule should yield as few misclassifications as possible on average. In some applications one may also want to take into account possible costs caused by misclassifications.

Quantifying the consequences of the decisions can be formally expressed via a *cost function*:

$$C(k,j) : \mathcal{G} \times \mathcal{G} \to \mathbb{R}$$

which quantifies the cost of misclassifying an observation into class j when its true class is k. Commonly, one assumes that

$$C(k,k) = 0 \qquad \text{and} \qquad C(k,j) > 0, \quad \forall k \neq j \in \mathcal{G}$$

i.e., the cost is zero for correct classification and non-zero when an observation is incorrectly classified.

The simplest cost is the *uniform cost*

$$C(k,j) = \mathbb{1}_{\{k \neq j\}} = \begin{cases} 1, & k \neq j \\ 0, & k = j \end{cases} \tag{2.13}$$

as it gives unit cost to all errors. Our goal is then to find a rule $G(\mathbf{x})$ that minimizes the *expected cost of misclassification*, defined as

$$\begin{aligned} \mathrm{ECM}(G) &= \mathbb{E}_{X,Y}\big[C\big(Y, G(X)\big)\big] & (2.14) \\ &= \sum_{k=1}^{K} \mathbb{E}_X\big[C\big(k, G(X)\big) \mid Y = k\big]\pi_k, & (2.15) \end{aligned}$$

where $\mathbb{E}_X\big[C(k,G(X))\big|Y=k\big]$ is the expected cost of misclassification, when the observation is from class k. Notice that the Bayes risk $\mathrm{r}(G)$ coincides with ECM based on uniform costs.

Suppose we have two classes. The (expected) cost of classifying an observation from class 0 to class 1 is

$$\mathbb{E}_X\big[C(0,G(X)) \mid Y = 0\big] = \int C(0,G(\mathbf{x}))f_{X|Y}(\mathbf{x} \mid 0)\mathrm{d}\mathbf{x} = C(0,1)\int_{\Gamma_1(G)} f_{X|Y}(\mathbf{x} \mid 0)\mathrm{d}\mathbf{x}$$

and similarly

$$\mathbb{E}_X[C(1,G(X)) \mid Y = 1] = C(1,0)\int_{\Gamma_0(G)} f_{X|Y}(\mathbf{x} \mid 1)\mathrm{d}\mathbf{x}.$$

The ECM in (2.14) is

$$\pi_0 \cdot C(0,1)\int_{\Gamma_1(G)} f_{X|Y}(\mathbf{x} \mid 0)\mathrm{d}\mathbf{x} + \pi_1 \cdot C(1,0)\int_{\Gamma_0(G)} f_{X|Y}(\mathbf{x} \mid 1)\mathrm{d}\mathbf{x}.$$

We then have the following result.

Theorem 2.2. *The discriminant rule $G^*(\mathbf{x})$ that minimizes the ECM is based on discriminant scores*

$$G_0(\mathbf{x}) = \ln f_{X|Y}(\mathbf{x} \mid 0) + \ln \pi_0 + \ln C(0,1),$$
$$G_1(\mathbf{x}) = \ln f_{X|Y}(\mathbf{x} \mid 1) + \ln \pi_1 + \ln C(1,0),$$

where the decision regions are

$$\Gamma_0^* = \{\mathbf{x} : G_0(\mathbf{x}) \geq G_1(\mathbf{x})\} = \left\{ \mathbf{x} : \frac{f_{X|Y}(\mathbf{x} \mid 0)}{f_{X|Y}(\mathbf{x} \mid 1)} \geq \frac{C(1,0)}{C(0,1)} \cdot \frac{\pi_1}{\pi_0} \right\},$$
$$\Gamma_1^* = \{\mathbf{x} : G_0(\mathbf{x}) < G_1(\mathbf{x})\} = \left\{ \mathbf{x} : \frac{f_{X|Y}(\mathbf{x} \mid 0)}{f_{X|Y}(\mathbf{x} \mid 1)} < \frac{C(1,0)}{C(0,1)} \cdot \frac{\pi_1}{\pi_0} \right\}.$$

The optimal discriminant rule $G^*(\mathbf{x})$ is actionable if we know perfectly the conditional class p.d.f.'s $f_{X|Y}(\mathbf{x} \mid 0)$ and $f_{X|Y}(\mathbf{x} \mid 1)$, prior probabilities π_0 and π_1 and misclassification costs $C(0,1)$ and $C(1,0)$, or their ratios:

$$P = \frac{\pi_1}{\pi_0}, \quad C = \frac{C(1,0)}{C(0,1)}, \quad \text{and} \quad L(\mathbf{x}) = \frac{f_{X|Y}(\mathbf{x} \mid 0)}{f_{X|Y}(\mathbf{x} \mid 1)}. \tag{2.16}$$

Note that $L(\mathbf{x})$ is the likelihood ratio (*cf.* Remark 2.2). Choosing uniform costs, so $C(0,1) = C(1,0) = 1$, one obtain the Bayes rule which minimizes the risk $r(G)$.

Example 2.2. Suppose it is known that it is twice as costly to assign an observation from class 1 to class 0 than vice versa and that approximately 20% of observations belong to class 1. When an observation $\mathbf{x}$ receives values $f_{X|Y}(\mathbf{x} \mid 0) = 0.3$ and $f_{X|Y}(\mathbf{x} \mid 1) = 0.4$, then is it classified to class 1 or class 2? ■

2.3 Predictor and the loss functions

Consider the binary class problem with output variable $Y \in \mathcal{G} = \{-1, 1\}$. Then a classifier $G(\cdot)$ produces a prediction taking one of the two values, -1 or 1, and can be represented as

$$G(\mathbf{x}) = \mathsf{sign}[f(\mathbf{x})] \tag{2.17}$$

for some function $f : \mathbb{R}^p \to \mathbb{R}$, called the predictor function. We can write the misclassification loss in (2.5) as

$$L_{0/1}(y, f(\mathbf{x})) = 1_{\{y\mathsf{sign}[f(\mathbf{x})] \neq 1\}} = 1_{\{yf(\mathbf{x})<0\}} \tag{2.18}$$

where $m = yf(\mathbf{x})$ is called the *margin* of $(y, \mathbf{x})$. The 1-0 loss in (2.18) assigns unit penalty for negative margin values, and no penalty at all for positive ones. Consequently, the risk (2.6) and the empirical risk (2.8) can also be expressed as:

$$\mathrm{r}(f) = \mathbb{E}\left[1_{\{Yf(X)<0\}}\right] \tag{2.19}$$

$$\mathrm{r}_N(f) = \frac{1}{N}\sum_{i=1}^{N} 1_{\{y_i f(\mathbf{x}_i)<0\}}. \tag{2.20}$$

name	$L(y, f(\mathbf{x}))$	$f^*(\mathbf{x})$
misclassification loss	$1_{\{yf(\mathbf{x})<0\}}$	$p(\mathbf{x}) - 1/2$
exponential loss	$\exp(-yf(\mathbf{x}))$	$\frac{1}{2}\log\frac{p(\mathbf{x})}{1-p(\mathbf{x})}$
binomial deviance	$\log\left(1 + e^{-2y\,f(\mathbf{x})}\right)$	$\frac{1}{2}\log\frac{p(\mathbf{x})}{1-p(\mathbf{x})}$
support vector (hinge loss)	$[1 - yf(\mathbf{x})]_+$	$p(\mathbf{x}) - 1/2$

TABLE 2.1: *Popular loss functions for classification as well as the associated optimal predictor function solving* (2.21). *Notation* $[y]_+$ *means positive part of* y, *i.e.*, $[y]_+ = y1_{\{y>0\}} = \max(y, 0)$. *We use shorthand notation* $p(\mathbf{x})$ *for* $p_1(\mathbf{x}) = \Pr(Y = 1 \mid X = \mathbf{x})$.

Thus the risk empirical $\mathrm{r}_N(f)$ is simply the average of negative margins. An observation $(\mathbf{x}_i, y_i)$ with positive margin $y_i f(\mathbf{x}_i) > 0$ is classified correctly whereas those with negative margins $y_i f(\mathbf{x}_i) < 0$ are misclassified.

If would then be natural to find f that minimizes the empirical risk (2.8). However, this problem turns out to be NP-complete, meaning that no polynomial-time algorithm is believed to exist to solve it. Thus different loss function $L(y, f(\mathbf{x})) : \mathcal{G} \times \mathbb{R} \to \mathbb{R}$ are used instead of the misclassification loss $L_{0/1}(y, f(\mathbf{x}))$. In the two-class case, $L(y, f(\mathbf{x}))$ will be commonly a function of the margin $yf(\mathbf{x})$. In the regression set-up, where $f(\mathbf{x})$ expresses the dependence of output $y \in \mathbb{R}$ on input $\mathbf{x} \in \mathbb{R}^p$, the predictor function can be e.g., a linear regression fit, $f(\mathbf{x}) = \mathbf{x}^\top \boldsymbol{\beta}$.

The predictor function $f(\mathbf{x})$ that defines the classifier in (2.17) is chosen as the function that minimizes the (conditional) expected loss

$$f^*(\mathbf{x}) = \arg\min_{f(\mathbf{x})} \mathbb{E}_Y\left[L\big(Y, f(\mathbf{x})\big) \mid X = \mathbf{x}\right] \tag{2.21}$$

This is the optimal population rule, assuming knowledge of the conditional distribution $Y|X = \mathbf{x}$. Table 2.1 specifies the optimal predictors function for each considered loss function, where we have used shorthand notation $p(\mathbf{x})$ for $p_1(\mathbf{x}) = \Pr(Y = 1 \mid X = \mathbf{x})$. The goal in (2.21) is to produce positive margins as frequently as possible. Naturally, any loss criterion used to derive $f(\mathbf{x})$ should penalize negative margins more heavily than positive ones since positive margin observations are already correctly classified.

For misclassifcation los, the classifier is the Bayes classifier whose predictor function is $f^*(\mathbf{x}) = p(\mathbf{x}) - 1/2$ as pointed out in Remark 2.1. For exponential loss function the optimal rule equals one-half of log odds. As we shall see later, the exponential loss is related to AdaBoost classifier and $f^*(\mathbf{x})$ is the minimizer of the population version of the AdaBoost criterion. For binomial deviance loss, one obtains the same population minimizer $f^*(\mathbf{x})$ as for exponential loss function. For support vector loss, the optimal predictor function $f^*(\mathbf{x})$ is the same as for Bayes classifier. In this sense, support vector loss can be preferred. Comparison of loss functions are given in subsection 2.3.3.

Note that $1_{\{x\leq 0\}} \leq e^{-x}$ and thus we notice from (2.20) that the empirical risk can be upper bounded by

$$\frac{1}{N}\sum_{i=1}^{N} e^{-y_i f(\mathbf{x}_i)}.$$

Thus AdaBoost classifier that aims at minimizing the exponential loss can be interpreted as method that minimizes the upper bound of the empirical risk (training error). In fact, binomial deviance and SVM loss have the same interpretation as will become clear from Figure 2.1.

2.3.1 Logistic transformation

The binomial deviance loss function (*cf.* Table 2.1) equals the negative log-likelihood function of the binomial distribution, called the *deviance*, using the logit transformation defined below.

Definition 2.3. In 2-class problem, let $p(\mathbf{x}) = \Pr(Y = 1 \mid X = \mathbf{x})$ Then define the symmetric *logistic transformation* as

$$f(\mathbf{x}) = \frac{1}{2}\log\frac{p(\mathbf{x})}{1-p(\mathbf{x})} \Leftrightarrow p(\mathbf{x}) = \frac{e^{f(\mathbf{x})}}{e^{f(\mathbf{x})}+e^{-f(\mathbf{x})}} = \frac{1}{1+e^{-2f(\mathbf{x})}} \tag{2.22}$$

i.e., f equals half of the log-odds ratio.

Symmetric logistic transformation is related to conventional logistic or *sigmoid function*, commonly used to transform values on $(-\infty,\infty)$ into numbers on $(0,1)$, defined as

$$\sigma(z) = \frac{e^z}{1+e^z} = \frac{1}{1+e^{-z}} \Leftrightarrow \sigma^{-1}(z) = \log\frac{z}{1-z}. \tag{2.23}$$

Thus $p(\mathbf{x}) = \sigma(2f(\mathbf{x}))$ with $f(\mathbf{x}) = \frac{1}{2}\sigma^{-1}(p(\mathbf{x}))$.

In the two class case, we may model a random variable $Y' \in \{0,1\}$ as being generated from a Bernoulli distribution $\text{Ber}(p(\mathbf{x}))$, where $p(\mathbf{x})$ is defined in (2.22). The conditional probability mass function of $Y'|X = \mathbf{x}$ is thus

$$f_{Y'|X}(y'|\mathbf{x}) = p(\mathbf{x})^{y'}(1-p(\mathbf{x}))^{1-y'}, \quad y' \in \{0,1\}.$$

The Bernoulli log-likelihood function is then

$$l(y', p(\mathbf{x})) = y'\log p(\mathbf{x}) + (1-y')\log(1-p(\mathbf{x})) \tag{2.24}$$

which is also sometimes referred to as *cross entropy loss*. The *Binomial deviance*, i.e., the negative log-likelihood, $-l(y, p(\mathbf{x}))$, can be written using the relationship of $f(\mathbf{x})$ and $p(\mathbf{x})$ in (2.22) and output encoding $y' = (y+1)/2$, as

$$-l(y, f(\mathbf{x})) = \log(1+e^{-2yf(\mathbf{x})}), \quad y \in \{-1,1\}. \tag{2.25}$$

Note that $\exp(-yf(\mathbf{x}))$ itself is not a proper log-likelihood, as it does not equal the log of any probability mass function on plus or minus 1.

2.3.2 Loss functions for regression

In the regression the response is non-categorical, $Y \in \mathbb{R}$, and different loss functions are used. The most common loss functions are the *squared loss* (or L_2-loss)

$$L(y, f(\mathbf{x})) = \frac{1}{2}(y - f(\mathbf{x}))^2 \tag{2.26}$$

or the *absolute loss* (or L_1-loss)

$$L(y, f(\mathbf{x})) = |y - f(\mathbf{x})|^2. \tag{2.27}$$

They can also be used for binary-classification. In this case, they can be parametrized by margin value $yf(x)$ since for $y \in \{-1, 1\}$ one has that

$$\begin{aligned}
&.|y - f(\mathbf{x})| = |1 - yf(\mathbf{x})|, \\
&(y - f(\mathbf{x}))^2 = (1 - yf(\mathbf{x}))^2 = 1 - 2yf(\mathbf{x}) + (yf(\mathbf{x}))^2.
\end{aligned}$$

However, these functions are non-monotone functions of the margin value which is not optimal for classification setting as described in the next section.

For squared-error loss, the optimal predictor fucntion $f^*(\mathbf{x})$ is

$$\begin{aligned}
f^*(\mathbf{x}) &= \arg\min_{f(\mathbf{x})} \frac{1}{2}\mathbb{E}_Y\left[(Y - f(\mathbf{x}))^2 \mid X = \mathbf{x}\right] = \frac{1}{2}\mathbb{E}[Y \mid X = \mathbf{x}] \\
&= p(\mathbf{x}) - 1/2.
\end{aligned}$$

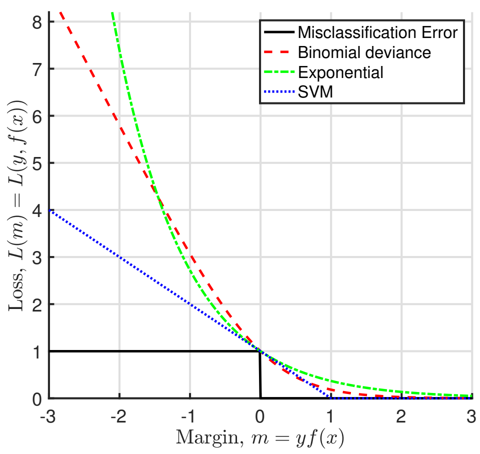
Thus we see that the population predictor function for the squared-error loss function is the same as for optimal Bayes classifier that minimizes the risk. However, squared-error loss function fails to penalize negative margins.

2.3.3 Comparison of loss criterions

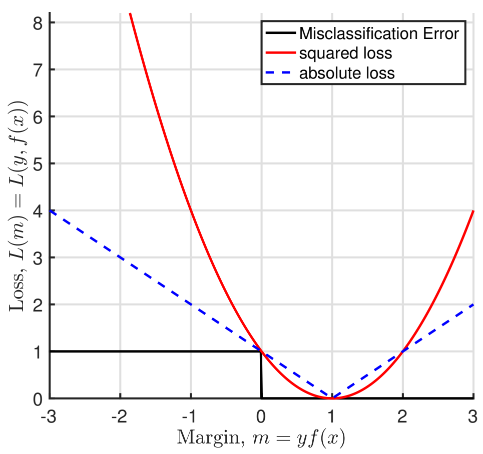
Figure 2.1 shows the misclassification, exponential, binomial deviance, support vector and squared-error loss function as a function of the margin $m = y \cdot f(\mathbf{x})$. Both the exponential, deviance and support vector loss can be viewed as monotone decreasing continuous approximations (surrogates) to misclassification loss or convex upper bounds for the misclassification error. Misclassification loss is non-differentiable and also non-convex as a function of the margin m and hence not suitable for boosting which essentially uses functional gradient descent for optimization.

The difference between them is in degree of penalizing negative margins. Binomial deviance can be considered to be *more robust* than exponential loss as it assigns relatively less influence on observations with large negative margins. Unlike exponential loss, it grows linearly as the margin value m tends to $-\infty$. Hence its performance is not as much affected if there are misspecification of the class labels in the training data. We also notice from Figure 2.1 that the squared-error loss is not a good surrogate for misclassification error.

When robustness is a concern, squared-error loss for regression and exponential loss for classification are not the best criteria from a statistical perspective. However, they both lead to the elegant modular boosting algorithms in the context of forward stagewise additive modeling.



(A) monotone loss functions



(B) non-monotone loss functions

Figure 2.1: *Loss functions $L(y, f(\mathbf{x}))$ for two-class classification is expressed as a function of margin $m = yf(\mathbf{x})$ as $L(y, f(\mathbf{x})) = L(m)$. The response is $y = \pm 1$; the prediction is $f(\mathbf{x})$, with class prediction $\mathsf{sign}[f(\mathbf{x})]$. The losses $L(m)$ in (a) are misclassification: $1_{\{m<0\}}$; exponential: $\exp(-m)$; binomial deviance: $\log(1 + \exp(-2m))$; and support vector: $(1 - m)_+$ and in (b) squared error: $(1 - m)^2$; and absolute $|1 - m|$. Each function has been scaled so that it passes through the point $(0, 1)$.*

2.4 Performance measures

Given the estimated rule $\hat{G}(\mathbf{x})$, its risk (cf. Definition 2.1), $\mathrm{r}(\hat{G}) = \Pr(\hat{G}(X) \neq Y)$ is difficult to calculate and hence one computes its empirical version. Consider binary classfication problem and encode the classes as negative ('$-$'), e.g., a negative result on covid-19 test, and positive ('$+$'), e.g., a positive result on covid-19 test. The *training error rate* or *empirical risk* is then defined by

$$\mathrm{r}_N(\hat{G}) = \frac{1}{N}\sum_{i=1}^{N} 1_{\{\hat{G}(\mathbf{x}_i)\neq y_i\}} = \frac{\mathrm{FP} + \mathrm{FN}}{\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}}$$

where $N = \mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}$, and

$$\begin{aligned} &\mathrm{FN} \text{ (aka } \textit{miss-detection}) \\ &\quad = \# \text{ of observations of class } + \text{ that are misclassified} \\ &\mathrm{FP} \text{ (aka } \textit{false alarm}) \\ &\quad = \# \text{ of observations of class } - \text{ that are misclassified} \end{aligned}$$

where FP and FN (TP and TN) stand for false (true) positive and negative, respectively. These numbers are often represented via 2×2 *confusion matrix*:

		Predicted class		
		+	−	sum
True	+	TP	FN	TP + FN
class	−	FP	TN	FP + TN

Usually normalized confusion matrix is reported which gives the numbers as percentages (per total number of instances in each class, i.e., figures are "row-normalized"). The *true positive rate* (TRP) and *true negative rate (TNR)* are

$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}} \qquad \text{and} \qquad \text{TNR} = \frac{\text{TN}}{\text{FP} + \text{TN}}.$$

Training error rate is far too optimistic and underestimates the true error rate (risk). This is because the data used to compute the discriminant rule is also used to evaluate it. Therefore, it is customary to report test error rate (explained next) which is as easy to calculate and does not require any distributional assumptions. Moreover, confusion matrices are reported for test data (not for training data).

2.4.1 Test accuracy/error rate

The *test error rate* (TER) is computed as follows:

1. *Hold-out*: Randomly split the available data into training set and test set using some split ratio[a]
2. Training set is used to construct the classifier $\hat{G}$ and the test set is used to evaluate a performance measure on a test set.[b]
3. Repeat steps 1 and 2 (using another random split), e.g., 100 times[c]
4. TER is computed as the average of obtained error rates while the test set accuracy is computed as $1 - \text{TER}$.

[a]e.g., ratio $2:1$ implies that $(2/3)^{rd}$ of observations in each class are used to build the training set and the remaining $(1/3)^{rd}$ are left for test set.

[b]The most commonly used performance measure is error rate, so the proportion of observations misclassified in the test set

[c]this is sometimes called repeated K-fold cross validations

In python, you can use `train_test_split` available at scikit-learn package for splitting the data to train and test sets.

Sometimes, the data size may not be sufficient for splitting the data into sufficiently large train and test set. Then less effective cross-validated TER can be computed as follows: the data set $\mathcal{T}$ is split into K disjoint subsets. For each subset of data, say $\mathcal{T}_i$, train on all but $\mathcal{T}_i$, then calculate the error rate on data $\mathcal{T}_i$ that was left out. Finally average the obtained errors rates.

Sometimes also stratified sampling is used in step 1 (hold-out): it may be helpful to sample so that proportions of $+/-$ observations is maintained in training/test data splits. This is the case especially when the data is *unbalanced* (i.e., the number of instances in classes vary significantly). Moreover, when data is unbalanced, then it is better to use some other performance measure than error rate, for example, the F1-score.

2.4.2 Other measures

Test error rate is not everything! If a dataset is unbalanced, the overall accuracy is not representative of the true performance of a classifier.

For example, consider an unbalanced binary classification problem with $\#'-' = 990$ and $\#'+' = 10$. Then simply labelling everything $'-'$ yields 99% accuracy and has no false alarms. But, this classifier misses all positive class observations, so has 100% miss rate.

Hence it is customary to report also precision and recall (which are often reported as a pair):

- *Recall* $= \dfrac{\text{TP}}{\text{TP}+\text{FN}}$ equals true positive rate (TPR)

  $= \#$found true objects / $\#$all true objects

  Recall is also sometimes called *Sensitivity.*
- *Precision* $= \dfrac{\text{TP}}{\text{TP}+\text{FP}}$ emphasizes TP-s and FP-s

  $= \#$found true objects / $\#$found all objects

Other commonly reported measures are F1-scores and specificity. The *F1-score* is computed as harmonic mean of precision and recall metrics, yielding

$$\textit{F1-score} = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}}.$$

Given its trade-off nature, the F1-score is quite commonly used and better adapted than the accuracy to unbalanced scenarios. There are also other metrics commonly reported such as

$$\textit{Specificity} = \frac{\text{TN}}{\text{TN}+\text{FP}}. \qquad \text{(equals true negative rate (TNR))}$$

We can generalize above measures to the multi-class case by summarizing over the rows and columns of the confusion matrix. Assuming that the confusion matrix, denoted by $\mathbf{M} = (M_{ij})$ is oriented as above, so each row corresponds to specific value for the "truth", we calculate

$$\text{precision}_i = \frac{M_{ii}}{\sum_j M_{ji}} \quad \text{and} \quad \text{recall}_i = \frac{M_{ii}}{\sum_j M_{ij}}.$$

Thus precision is the fraction of cases where the classifier correctly predicted class i out of all cases for which the algorithm predicted i (correctly and incorrectly). Recall is the fraction of cases where the classifier correctly predicted i out of all of the cases which are labelled as i.

2.5 Conclusions

The purpose of this chapter was to introduce the basic concepts of classification such as risk and the optimal Bayes classifier, different loss functions, and measures of accuracy.

Although the conceptes were illustrated in binary classification problem, they generalize in straightforward manner to multi-class case which we omit due to lack of time.

Chapter 3

Decision tree methods

Tree-based methods partition the feature space into a set of rectangles, and then fit a simple model (like a constant) in each one. They are conceptually simple yet powerful. We first describe a popular method for tree-based regression and classification called *CART* [Breiman et al., 1984] Since the set of splitting rules used to segment the predictor space can be summarized in a tree, these types of approaches are known as *decision-tree* methods. Good reading of these methods are Section 8.7 (Bagging), Section 9.2 (Tree-based methods) and Sections 15.2-15.3 (Random forests) of Hastie et al. [2009].

3.1 Decision tree

Decision trees can be applied to both regression (considered in Section 3.2) and classification (considered in Section 3.3) problems. We consider a regression problem with continuous response Y and inputs X_1 and X_2, each taking values in the unit interval. Figure 3.1 illustrates a simple *recursive binary tree* :

- split the space into 2 regions, and model the response by mean of Y in each region. Choose the variable and split-point to achieve the best fit.
- one or both of these regions are split into two more regions, and this process is continued, until some stopping rule is applied.

The partition in Figure 3.1 is obtained in 3 stages:

1. Split at $X_1 = t_1$.
2. Split the region $X_1 \leq t_1$ at $X_2 = t_2$ and $X_1 > t_1$ at $X_1 = t_3$.
3. Split the region $X_2 > t_2$ and $X_1 \leq t_1$ at $X_1 = t_4$.
4. Split the region $X_1 > t_3$ at $X_2 = t_5$.

$\Rightarrow$ yields a partition of (X_1, X_2)-space into regions $R_1, R_2, \ldots, R_6$.

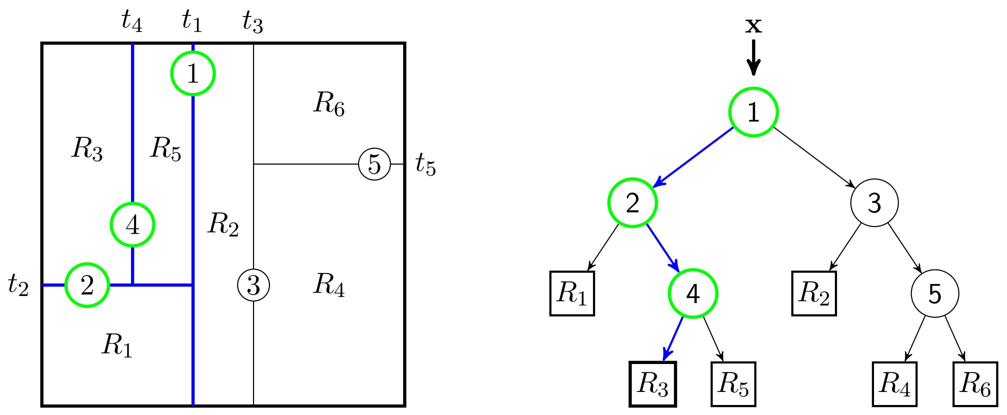


FIGURE 3.1: *Left panel shows partition of a 2-dimensional feature space by recursive binary splitting, as used in CART, applied to some fake data. Right panel shows the tree corresponding to the partition. The test data* $\mathbf{x}$ *fed into the tree belong to the region* R_3.

The corresponding regression model predicts Y with a constant c_m in region R_m, that is,

$$\hat{f}(X) = c_m \sum_{m=1}^{6} 1_{\{(X_1,X_2)\in R_m\}}.$$

The *terminal nodes* or *leaves* of the tree correspond to regions $R_1, R_2, \ldots, R_6$. The points along the tree where the predictor space is split are referred to as *internal nodes*.

3.2 Regression trees

Given the data of inputs and and a response, $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$, with $\mathbf{x}_i^\top = (x_{i1}, \ldots, x_{ip})$, the algorithm needs to decide:

1. *What is the best split?* (which variables and split points)
2. *How to split the variables*, i.e., what topology (shape) the tree should have? E.g., binary tree or multiway?
3. *How large to grow the tree?*

Suppose first that we have a partition $R_1, R_2, \ldots, R_M$, and we model the response as a constant c_m in each region:

$$f(\mathbf{x}) = c_m \sum_{m=1}^{M} 1_{\{\mathbf{x}\in R_m\}}.$$

If we adopt as our criterion minimization of sum of squares $\sum(y_i - f(\mathbf{x}_i))^2$, it is easy to see that the best $\hat{c}_m$ is

$$\hat{c}_m = \text{ave}(y_i|\mathbf{x}_i \in R_m) = \text{ average of } y_i \text{ in region } R_m.$$

Finding the best binary partition in terms of minimum sum of squares is generally computationally infeasible.

Hence one proceeds with a following *greedy algorithm*:

1. Starting with all of the data, split variable j at a split point s, and define the pair of half-planes:
$$R_1(j,s) = \{\mathbf{x} \mid x_j \le s\} \quad \text{and} \quad R_2(j,s) = \{\mathbf{x} \mid x_j > s\}$$
2. Determine the best splitting variable j and a split point s that solves
$$\min_{j,s} \Big[\min_{c_1} \sum_{\mathbf{x}_i \in R_1(j,s)} (y_i - c_1)^2 + \min_{c_2} \sum_{\mathbf{x}_i \in R_2(j,s)} (y_i - c_2)^2 \Big]$$
where for any j and s, the inner minimization is solved by
$$\hat{c}_1 = \text{ave}(y_i \mid \mathbf{x}_i \in R_1(j,s)) \quad \text{and} \quad \hat{c}_2 = \text{ave}(y_i \mid \mathbf{x}_i \in R_2(j,s))$$
3. Having found the best split, we partition the data into the two resulting regions and repeat the splitting process on each of the two regions. This process is continued until stopping criterion is met.

For each splitting variable, the determination of the split point s can be done very quickly and hence by scanning through all of the inputs, determination of the best pair (j,s) is feasible.

Q: What is a good stopping criterion - i.e., **how large we should grow the tree**? How do we know if we should split the tree node?

If we grow a tree deep enough, we end up in massive overfitting as we can usally fit the training data perfectly. The *tree size*, M, i.e., the number of terminal nodes, is a tuning parameter governing the model's complexity, and the optimal tree size should be adaptively chosen from the data, e.g., via pruning.

3.2.1 Pruning the tree

We define a *subtree* $T \subset T_0$ to be any tree that can be obtained by *pruning* T_0, that is, collapsing any number of its internal (non-terminal) nodes. We index terminal nodes (leaves) by m, with node m representing region R_m. Let $|T|$ denote the number of terminal nodes in T, and define

$$\begin{aligned} N_m &= \#\{\mathbf{x}_i \in R_m\}, & &(= \textit{node size}) \\ \hat{c}_m &= \frac{1}{N_m} \sum_{\mathbf{x}_i \in R_m} y_i, \\ Q_m(T) &= \frac{1}{N_m} \sum_{\mathbf{x}_i \in R_m} (y_i - \hat{c}_m)^2. & &(3.1) \end{aligned}$$

The *cost complexity criterion* is then defined as

$$C_\alpha(T) = \sum_{m=1}^{|T|} N_m Q_m(T) + \alpha |T|,$$

where $\alpha \geq 0$ is a tuning parameter that governs the tradeoff between tree size and its goodness of fit to the data:

- Large α results in smaller trees T (and vice versa).
- When $\alpha = 0$ the solution is the full tree T_0.

The **cost-complexity pruning** proceeds as follows:

1. Grow a large tree T_0, stopping the splitting process only when some minimum node size (say 5) is reached.
2. For each α, find the subtree T_α that minimizes $C_\alpha(T)$:

$$T_\alpha = \arg\min_{T \subseteq T_0} \sum_{m=1}^{|T|} N_m Q_m(T) + \alpha |T|,$$

   using *weakest-link pruning*[1]:

   2a) successively collapse the internal node that produces the smallest per-node increase in $\sum_m N_m Q_m(T)$

   2b) continue until one reaches the single-node (root) tree

   2c) this produces a sequence of subtrees which contains T_α.

3. Estimation of α is achieved by five- or tenfold cross-validation: $\hat{\alpha}$ minimizes the cross-validated sum of squares.
4. Final tree is $T_{\hat{\alpha}}$.

3.3 Classification trees

If the task is classification, so $Y \in \{1, 2, \ldots, K\}$, the only changes needed in the tree algorithm pertain to the criteria for splitting nodes and pruning the tree.

It is obvious that the node impurity measure $Q_m(T)$ in (3.1) needs to be redefined. Suppose there are N_m observations in a node m that represents a region R_m. Let $(\hat{p}_{m,1}, \ldots, \hat{p}_{m,K})$ denote the frequencies, that is,

$$\hat{p}_{m,k} = \frac{1}{N_m} \sum_{\mathbf{x}_i \in R_m} 1_{\{y_i = k\}} := \begin{array}{l} \text{proportion of class } k \\ \text{observations in node } m \end{array}.$$

We classify the observations in node m to class

$$k(m) = \arg\max_k \hat{p}_{m,k},$$

i.e., the majority class in node m.

Commonly used options for node impurity are listed in Table 3.1. Notice that minimizing the Gini-index will favour pure nodes, which is why it often the favoured option

[1]For each α one can show that there is a unique smallest subtree T that minimizes $C_\alpha(T)$.

Misclassification:	$\mathrm{Err}(m) = 1 - \hat{p}_{m,k(m)}$
Gini index:	$\mathrm{Gini}(m) = \sum_{k \neq k'} \hat{p}_{m,k}\hat{p}_{m,k'} = 1 - \sum_{k=1}^{K} \hat{p}_{m,k}^2$
- maximized when $\hat{p}_{m,k} = 1/K$ with value $1 - 1/K$ - minimized when all cases belong to a single class.	
Entropy/deviance:	$H(m) = -\sum_{k=1}^{K} \hat{p}_{m,k} \log \hat{p}_{m,k}$
- maximized when $\hat{p}_{m,k} = 1/K$ with value $\log K$. - minimized when one class has no cases in it.	

TABLE 3.1: *Different measures $Q_m(T)$ of node impurity*

for node impurity. For two classes, if p is the proportion in the second class, these three measures are:

$$1 - \max(p, 1-p), \quad 2p(1-p) \quad \text{and} \quad -p \log p - (1-p)\log(1-p),$$

respectively. They are shown in Figure 3.2. Either Gini index $\mathrm{Gini}(\cdot)$ or entropy $H(\cdot)$ are favoured when growing the tree:

- they are differentiable (better suited for numerical optimization)
- they are more sensitive to changes in the node probabilities.

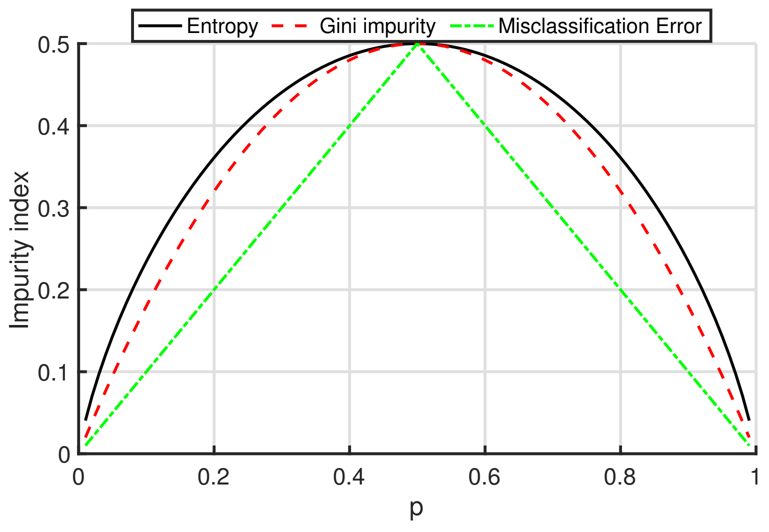


FIGURE 3.2: *Node impurity measures in the binary classification problem. The cross-entropy has been scaled by 1/2.*

3.3.1 Choosing the variable to split on and the split point

Suppose the parent node m is split into J new child nodes m_j, $j = 1, \ldots, J$. Each child has a count n_j and let

$$(\hat{p}_{m_j,1}, \ldots, \hat{p}_{m_j,K}), \quad j = 1, \ldots, J$$

denote the vector of class frequencies at child note m_j. The parent node has then a count $n = \sum_j n_j$ cases and the *gain* in Gini index is computed as

$$\mathsf{Gain}\big(G, m \to (m_j)_1^J\big) = G(m) - \sum_{j=1}^{J} \frac{n_j}{n} G(m_j).$$

(Similarly we can compute the gain for $H(m)$ or $\mathrm{Err}(m)$.)

One can then choose the best splitting node as the one that produces the biggest gain in G. Similarly a continuous variable can be discretized and then choose the splitting point as the point in the grid that produces the largest gain.

3.4 Bagging

A binary decision tree takes a majority vote within each cell of a partition of the feature space that has appearance as illustrated in Figure 3.1. Since the partition of the feature space is rough (highly non-smooth regardless of how large the tree is grown) and highly unstable/non-robust (i.e., the partition is learned with a lot of variance and a small change in the inputs can lead to large change in the output).

Ensemble methods aim to tackle these deficiencies of the decision trees. An *ensemble* (or committee) of classifiers is a classifier build upon some combination of learners. An ensemble method combines the following simple steps

1. Generate a number of classifiers $G_1(\cdot), \ldots, G_M(\cdot)$ based on the same data set but typically using some degree of randomness (e.g., in the selection of the features, or in the selection of the observations from the full data set).

2. Aggregate the classifiers to make the final prediction (take a majority vote).

3. In the case of ties (e.g., some classes have equal number of votes), then take a random pick of the classes.

Even though none of the classifier performs particularly well at its own, the aggregated result can be good. This is due to the wisdom of the masses.

Bagging aka *bootstrap aggregating* introduces randomness via bootstrap samples. A *bootstrap sample* $\mathcal{T}^* = \{(y_i^*, \mathbf{x}_i^*)\}_{i=1}^N$ is formed from the original sample by resampling **with replacement** from the original training data $\mathcal{T} = \{(y_i, \mathbf{x}_i)\}_{i=1}^N$. Suppose we have a learning algorithm (e.g., a tree), and suppose we have generated $\mathcal{T}_b^*$, for $b = 1, \ldots, B$ bootstrap samples. Let G_b be the classifier when applied to the boostrap sample $\mathcal{T}_b^*$. The predicted class of bagging classifier is then a majority vote over all base classifiers $G_1, \ldots, G_B$ or the one with highest mean probability estimate across the base classifiers. The latter option is nowadays the default choice while majority voting is used only when the base classifier does not provide the class probability estimates Thus one computes

regression: $\hat{f}(\mathbf{x}) = \frac{1}{B} \sum_{b=1}^{B} G_b(\mathbf{x})$ is the prediction of $\mathbf{x} \in \mathbb{R}^p$.

classification: Predicted class label for $\mathbf{x} \in \mathbb{R}^p$ is determined using majority voting:

$$\hat{G}(\mathbf{x}) = \arg\max_k \sum_{b=1}^{B} 1_{\{G_b(\mathbf{x}) = k\}}$$

or mean probability:

$$\hat{G}(\mathbf{x}) = \arg\max_k \sum_{b=1}^{B} \hat{\Pr}_b(Y = k \mid X = \mathbf{x})$$

where $p^{(b)}(\mathbf{x}) = \hat{\Pr}_b(Y = k | X = \mathbf{x})$ is the probability prediction for kth class obtained by $G_b(\cdot)$.

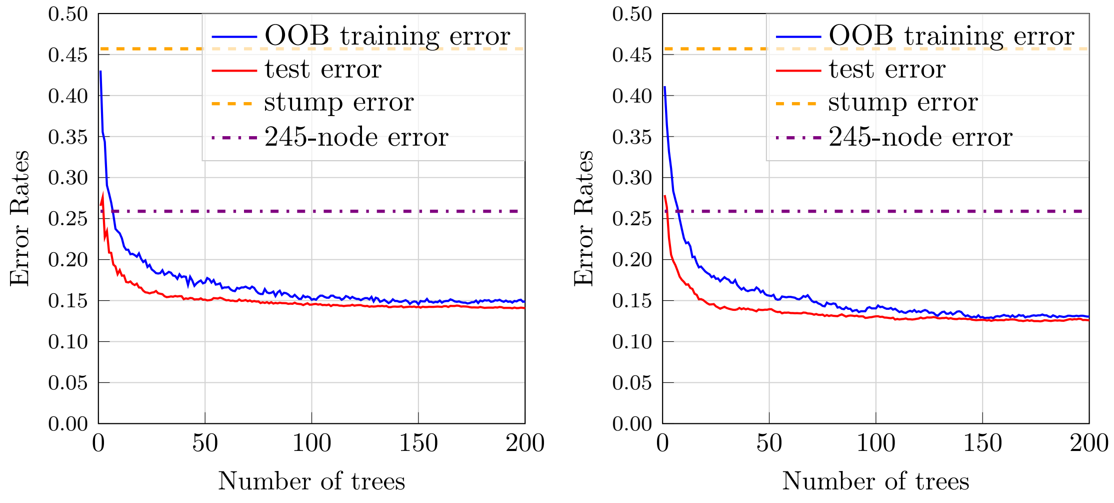
Algorithm 3.1 describes the Bagging procedure when base classifier is a tree. Note that the algorithm is general, and not in any means restricted to using trees. Basically, you can replace tree by any classifier.

Algorithm 3.1: `Bagging` algorithm for classification or regression using trees

```
    Input  : $\mathcal{T} = \{(y_i, \mathbf{x}_i^\top)\}_{i=1}^N$ (training data), $B$ (number of bootstrap
             samples)
    Output : Ensemble of trees $\{T_b\}_{b=1}^B$
1   for $b = 1$ to $B$ do
2       Draw a bootstrap sample $\mathcal{T}_b^*$ of size $N$ from the training data.
3       Build a tree $T_b(\cdot)$ on $\mathcal{T}_b^*$
    // To make a prediction at a new point x:
4   if classification task then
5       if not majority voting then
6           $\hat{G}(\mathbf{x}) = k$ having largest mean probability $\sum_{b=1}^B p_k^{(b)}(\mathbf{x})$
            // above $p^{(b)}(\mathbf{x}) = \hat{\Pr}_b(Y = k \mid X = \mathbf{x})$ based on $b$th tree $T_b(\cdot)$
7       else
8           $\hat{G}(\mathbf{x})$ = majority vote over $\{T_b(\mathbf{x})\}_{b=1}^B$
9   else
10      $\hat{f}(\mathbf{x}) = \frac{1}{B}\sum_{b=1}^B T_b(\mathbf{x})$.
```

An important feature of bagging (as well as of random forests) is its use of *out-of-bag (OOB)* samples: We can use cases not included in the Bootstrap sample (out-of-bag cases) as a test set. For each observation $\mathbf{z}_i = (\mathbf{x}_i, y_i)$, construct its bagging prediction by averaging only those trees which are formed using bootstrap samples in which $\mathbf{z}_i$ did not appear. Here averaging means either averaging the probability predictions or the class predictions in the case of majority voting. If an observation has no votes yet[2] one assigns the observation randomly to one of the classes. Ties can occur (namely, the case that two or more of the classes have same number of votes) when majority voting is used; in such case one again assigns the observation randomly to one of the classes. An OOB error estimate is almost identical to that obtained by N-fold cross-validation or to test error. Once the OOB error stabilizes, the training can be terminated. Thus OOB provides a simple means for choosing a good choice for B.

[2]this can happen only in the beginning of iterations. When iterations are ten or more, all observations usually have at least one prediction.



(A) Bagging (B) Random Forest

FIGURE 3.3: *Results for simulated data of Example 3.1. Error rates of (a) Bagging and (b) Random Forest (using $d = 2$ and $n_{min} = 3$) as a function of the number of trees. Also shown are the test error rate for a single stump, and a 245-node classification tree.*

Example 3.1. We consider the case that the features $X_1, \ldots, X_{10}$ are standard independent Gaussian, and the deterministic target Y is defined by

$$Y = \begin{cases} 1 & \text{, if } \sum_{j=1}^{10} X_j^2 > \chi^2_{10}(0.5) \\ -1 & \text{, otherwise} \end{cases},$$

where $\chi^2_{10}(0.5)$ is the median of a chi-squared random variable with 10 degrees of freedom (sum of squares of 10 standard Gaussians). We generate 2000 training observations $\{\mathbf{x}_i, y_i\}_{i=1}^N$ and 10,000 test observations. Hence we will have approximately 1000 (resp. 5000) cases in each class in the training (resp. test) data set.

Figure 3.3a illustrates the power of Bagging with threes. Applying stump (2 node tree) alone to the training data set yields a very poor test error rate of 46.0%, compared to 50.0% achieved by random guessing. Bagging achieves 14.05% test error rate which is significantly better than a single large classification tree (error rate 25.89%). Here we used the mean probability for class prediction. Figure shows the OOB misclassification error compared to the test error. Although 200 trees are averaged here, it appears from the plot that about 100 would be sufficient. The confusion matrix is shown in Figure 3.4. ■

3.5 Random forests

A random forest is an ensemble of decision trees where each decision tree is independently randomized. Bagging with decision trees is a simple example of a random forest.

Random forest as such suffer from the fact that bootstrap samples are highly correlated. As a consequence, different trees tend to select the same features and lead to

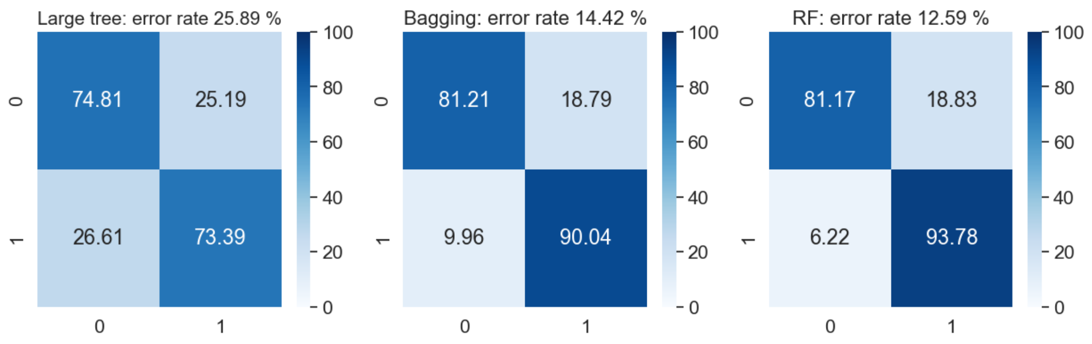


Figure 3.4: *Confusion matrices of Example 3.1 and Example 3.2. Large tree refers to a large tree with 245 nodes.*

(too) similar partitions of the feature space. Ideally, one would like to have partitions that are more independent. A simple cure is to incorporate *random feature selection*:

1. Generate classifiers by choosing random subsets of features and construct a decision tree on just the selected features. Random subsets are considered at each internal node so that the search space for each split is smaller.

2. Combine the approach with bagging.

Since the obtained partitions are less correlated, the obtained final aggregate prediction has reduced variance. Due to the reduced space at each split, the individual trees are built much faster than in bagging. As a rule of thumb one can use $d = \sqrt{p}$ features. The developer of random forests, Leo Breimann, called the random forests as the best "off-the-shelf" method for classification.[3] The algorithm give in Algorithm 3.2 follows the original publication Breiman [2001] but slight modifications to it are possible. Important parameters are d, i.e., the number of randomized features in each split) and n_{min}, i.e., minimum node size for ending splitting of nodes.

Example 3.2. We consider the same data set as in Example 3.1 and Figure 3.3b illustrates the results of random forest with threes. The parameters used were $d = 2$ (number of features used for splitting) and $n_{min} = 3$ (minimum node size) and $B = 200$. Random forest achieves error rate of 12.40% with 200 bags while bagging alone achieved 14.05% test error rate. Confusion matrix is shown in Figure 3.4.

Figure also shows the OOB training error compared to the test error. Although 200 trees are averaged, it appears that about 100 may suffice. The confusion matrices are shown in Figure 3.4b. ■

[3] An "off-the-shelf" method is one that can be directly applied to the data without requiring a great deal of time consuming data preprocessing or careful tuning of the learning procedure.

Algorithm 3.2: `RandomForest` algorithm for classification or regression

```
Input  : $\mathcal{T} = \{(y_i, \mathbf{x}_i)\}_{i=1}^N$ (training data), $B$ (number of bootstrap
         samples), $d$ (number of features in each split), $n_{min}$ (minimum
         node size)
Output : Ensemble of trees $\{T_b\}_{b=1}^B$
1  for $b = 1$ to $B$ do
2      Draw a bootstrap sample $\mathcal{T}_b^*$ of size $N$ from the training data.
       // Build a tree $T_b(\cdot)$ on $\mathcal{T}_b^*$
3      for each terminal node in tree do
4          Select $d$ variables at random from the $p$ variables.
5          Pick the best variable/split-point among the $d$ variables.
6          Split the node into two daughter nodes on the selected feature until the
             minimum node size $n_{min}$ is reached
   // To make a prediction at a new point $x$:
7  if classification task then
8      if not majority voting then
9          $\hat{G}(\mathbf{x}) = k$ having largest mean probability $\sum_{b=1}^B p_k^{(b)}(\mathbf{x})$
           // above $p^{(b)}(\mathbf{x}) = \hat{\Pr}_b(Y = k \mid X = \mathbf{x})$ based on $b$th tree $T_b(\cdot)$
10     else
11         $\hat{G}(\mathbf{x})$ = majority vote over $\{T_b(\mathbf{x})\}_{b=1}^B$
12 else
13     $\hat{f}(\mathbf{x}) = \frac{1}{B}\sum_{b=1}^B T_b(\mathbf{x})$.
```

Chapter 4

Boosting

Boosting is one of the most important recent developments in classification methodology. Boosting is an ensemble method that combines the outputs of many "weak" classifiers (such as simple trees) to produce a more powerful *committee* vote. A weak learner is a learning algorithm capable of producing classifiers with probability of error strictly (but not considerably) smaller than that of random guessing. A strong learner on the other hand is such that it is able to yield classifiers with arbitrarily small error probability given a sufficient amount of training data. Boosting is a fundamental concept in statistical learning as the idea is rather simple and can be modified and extended to many problems.

The first simple boosting procedure was developed by Schapire [1990] who showed that a weak learner could always improve its performance by training two additional classifiers on filtered versions of the input data stream. Later Freund and Schapire [1996, 1997] developed more adaptive and realistic AdaBoost, short for Adaptive Boosting, algorithm that combines many weak learners simultaneously. The techniques provably improved or "boosted" the performance of a single classifier by producing a majority vote of similar classifiers. The developers, Yoav Freund and Robert Schapire, won the 2003 Gödel Prize for their work on AdaBoost.

4.1 General ensemble scheme

Consider a weak learner or base procedure which constructs a function $G(\mathbf{x})$ based on input data:

$$\text{input } \{(\mathbf{x}_i, y_i)\}_{i=1}^N \longrightarrow \boxed{\text{weak learner}} \longrightarrow G(\cdot)$$

Boosting applies sequentially the weak classification/regression algorithm to repeatedly modified versions of the data. As an output one gets a sequence of weak learners $\{G_m(\mathbf{x})\}_{m=1}^M$.

The data modifications at each boosting step consist of applying weights $w_1, \ldots, w_N$ to each of the training observations $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$. Boosting can be applied to a regression or classification algorithm that accepts case weights, e.g.,

$$\underset{f}{\text{minimize}} \sum_{i=1}^{N} w_i L(y_i, f(\mathbf{x}_i)).$$

The predictions from all of them are combined through a weighted majority vote.

General boosting ensemble scheme:

initialize: $w_i^{(0)} = 1/N$, $i = 1, \ldots, N$

data 1 $\left\{(\mathbf{x}_i, y_i, w_i^{(0)})\right\}_{i=1}^{N} \longrightarrow \boxed{\text{weak learner}} \longrightarrow G_1(\cdot), \alpha_1$

data 2 $\left\{(\mathbf{x}_i, y_i, w_i^{(1)})\right\}_{i=1}^{N} \longrightarrow \boxed{\text{weak learner}} \longrightarrow G_2(\cdot), \alpha_2,$

$\vdots$

data M $\left\{(\mathbf{x}_i, y_i, w_i^{(M-1)})\right\}_{i=1}^{N} \longrightarrow \boxed{\text{weak learner}} \longrightarrow G_M(\cdot), \alpha_M$

output: Weighted majority vote $\hat{f}(\mathbf{x}) = \sum_{m=1}^{M} \alpha_m G_m(\mathbf{x})$, where α_m are classifier weights also estimated during the iterations.

For example, in the case of binary classification problem, the final prediction is

$$G(\mathbf{x}) = \mathsf{sign}(\hat{f}(\mathbf{x})) = \mathsf{sign}\left(\sum_{m=1}^{M} \alpha_m G_m(\mathbf{x})\right), \tag{4.1}$$

where the *classifier weights*, $\alpha_1, \ldots, \alpha_M$, computed by the boosting algorithm, are designed so that more accurate classifier $G_m(\mathbf{x})$ in the sequence obtains a larger weight and thus has a higher influence on $G(\mathbf{x})$. The *data weights* $w_i^{(m)}$, $i = 1, \ldots, N$ at each boosting iteration m depends on the accuracy of the previous classifiers, allowing the algorithm to focus its attention on those samples that are still incorrectly classified.

4.2 AdaBoost

The AdaBoost algorithm is the most well known boosting algorithm and it was originally developed for binary classification problem. Here the base classifier $G(\mathbf{x})$ attains values in $\{-1, 1\}$. Algorithm 4.1 shows the details of the *AdaBoost.M1* algorithm also known as the *discrete AdaBoost* [Friedman et al., 2000]. It is called discrete since the base classifier $G_m(\mathbf{x})$ returns a discrete class label.

The most important tuning parameter of AdaBoost is the number of weak learners M (or iterations) used in the aggregation. AdaBoost and other boosting procedures are quite resistant to overfitting when increasing the number of iterations M. This has been observed empirically, and is illustrated in Example 4.1 below. Nevertheless,

Algorithm 4.1: AdaBoost.M1 (alias Discrete AdaBoost) for binary classification

```
   Initialize: $w_i^{(0)} = 1/N$, $i = 1, \dots, N$
1  for $m = 1$ to $M$ do
2      Fit a classifier $G_m(\mathbf{x}) \in \{-1, 1\}$ to the training data using weights $w_i^{(m-1)}$.
3      Compute the weighted error rate
           $$\mathrm{err}_m = \frac{\sum_{i=1}^N w_i^{(m-1)} \mathbb{1}_{\{y_i \neq G_m(\mathbf{x}_i)\}}}{\sum_{i=1}^N w_i^{(m-1)}}$$
4      Compute $\alpha_m = \log((1 - \mathrm{err}_m)/\mathrm{err}_m)$.
5      Update the weights $w_i^{(m)} = w_i^{(m-1)} \cdot \exp[\alpha_m \cdot \mathbb{1}_{\{y_i \neq G_m(\mathbf{x}_i)\}}]$, $i = 1, \dots, N$.
   Output : $G(\mathbf{x}) = \mathsf{sign}\left(\hat{f}(\mathbf{x})\right)$ and $\hat{f}(\mathbf{x}) = \sum_{m=1}^M \alpha_m G_m(\mathbf{x})$
```

clear overfitting does occur for some datasets. Although early stopping is not necessary, AdaBoost and also other boosting algorithms are overfitting eventually, and early stopping, i.e., choosing a valid number of iterations M is necessary.

Some key points of AdaBoost.M1 are:

- Initially, $w_i = 1/N$, so at first step one trains the classifier on the data in the usual manner.
- For each successive iteration $m = 2, 3, \dots, M$ the observation weights are individually modified and the classification algorithm is reapplied to the weighted observations.
- At step m, observations that were misclassified by weak learner $G_{m-1}(\cdot)$ induced at the previous step have their weights *increased*.
- Thus, as iterations proceed, observations that are difficult to classify correctly receive ever-increasing influence.

A generalization of AdaBoost.M1 was proposed in Freund and Schapire [1996], and was further studied by Schapire and Singer [1999], that uses real-valued confidence-rated predictions rather than the $\{-1, 1\}$ of Discrete AdaBoost. Algorithm 4.2 presents this more general version of AdaBoost, referred to as *AdaBoost.R* (or *real AdaBoost*) in which the weak learner returns a class probability estimate $p^{(m)}(\mathbf{x}) = \hat{\Pr}_{\mathbf{w}}(Y = 1 \,|\, \mathbf{x}) \in [0, 1]$ at m^{th} iteration. Its contribution to the final classifier is one half the logit-transform of this probability estimate. In real AdaBoost, the base classifier returns a real-valued prediction, namely, a probability estimate of the class. Thus AdaBoost.R can use classifiers that can compute class prediction probability. For example, the predicted class probability for a decision tree is the fraction of samples of the same class in the terminal leaf.

The default algorithm in scikit-learn's `AdaBoostClassifier` is AdaBoost.R. In case you wish to use AdaBoost.M1, then choose option `algorithm='SAMME'`.

Example 4.1. Consider the same test and training data set as in Example 3.1. The performance of AdaBoost when the weak classifier is a stump (two terminal node

Algorithm 4.2: AdaBoost.R (alias Real AdaBoost)

```
  Initialize: $w_i^{(0)} = 1/N$, $i = 1, \dots, N$
1 for $m = 1$ to $M$ do
2     Fit a classifier $G_m(\mathbf{x})$ to obtain a class probability estimate
      $p^{(m)}(\mathbf{x}) = \hat{\Pr}_{\mathbf{w}}(Y = 1|\mathbf{x}) \in [0,1]$, using weights $w_i^{(m-1)}$ on the training data
3     Compute $f_m(\mathbf{x}) = \frac{1}{2} \log p^{(m)}(\mathbf{x})/(1 - p^{(m)}(\mathbf{x})) \in \mathbb{R}$.
4     Compute $w_i^{(m)} = w_i^{(m-1)} \exp\{-y_i f_m(\mathbf{x}_i)\}$, $i = 1, 2, \dots, N$ and renormalize so
      that $\sum_i w_i^{(m)} = 1$.
  Output : $G(\mathbf{x}) = \mathsf{sign}(\hat{f}(\mathbf{x}))$ and $\hat{f}(\mathbf{x}) = \sum_{m=1}^M f_m(\mathbf{x})$
```

classification tree) is shown in Figure 4.1a,b. Applying stump alone to the training data set yields a very poor test error rate of 46.0%, compared to 50.0% achieved by random guessing. However, as boosting iterations proceed the error rate steadily decreases, reaching 10.25% after $M = 600$ iterations when using AdaBoost.M1 and 5.63% using AdaBoost.R. Hence, boosting this weak classifier reduces its prediction error rate by a factor of 5 and 9, respectively. Both methods also outperform a single large classification tree (error rate 24.55%).

Figure 4.1c,d shows the performance when the weak learner is an 8-node decision tree. Initially, error rates for boosting eight-node trees decrease much more rapidly than for stumps. However, the error rates quickly level off and improvement is very slow after about 100 iterations. The error rate after 600 iteration is 6.86% and 7.19%, respectively. If we allow the AdaBoost.M1 with stumps to continue iterations, we can notice that after roughly 2000 iterations, it has achieved the same test error rate as AdaBoost.M1 using 8-node trees with $M = 600$ iterations. We also notice that (for this data set) boosting outperforms both random forest and bagging.

The confusion matrix of Adaboost based on stumps and $M = 600$ iterations are shown in Figure 4.2. These can be compared with Figure 3.4. ■

4.3 Forward Stagewise Additive Modeling

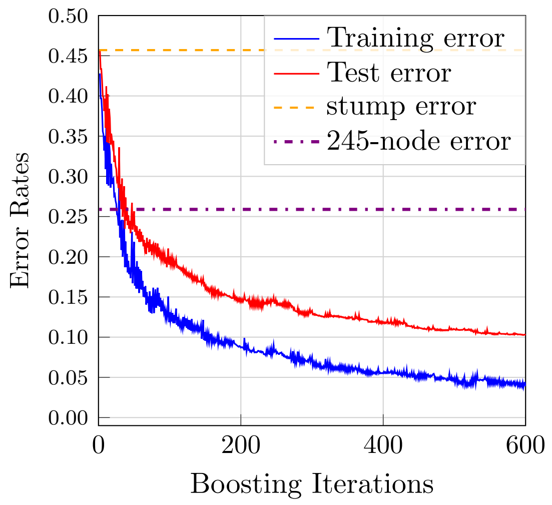
Consider a *basis function expansion* of the form

$$f(\mathbf{x}) = \sum_{m=1}^{M} \beta_m b(\mathbf{x}; \boldsymbol{\gamma}_m)$$

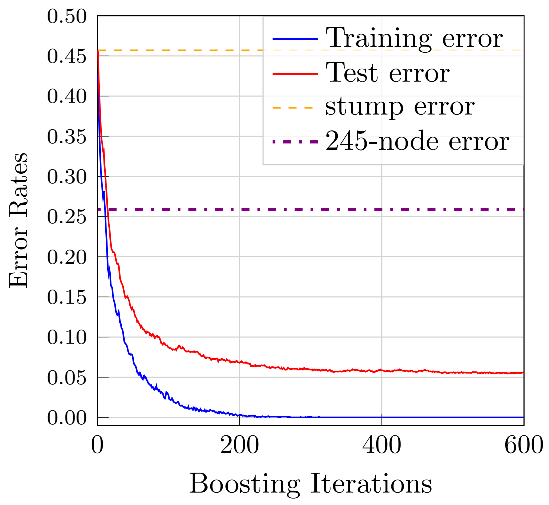
where β_m, $m = 1, 2, \dots, M$ are the expansion coefficients and $b(\mathbf{x}; \boldsymbol{\gamma}) \in \mathbb{R}$ are (usually simple) "basis" functions of the multivariate argument $\mathbf{x}$, characterized by a set of parameters $\boldsymbol{\gamma}$.

Typically these models are fit by minimizing empirical risk

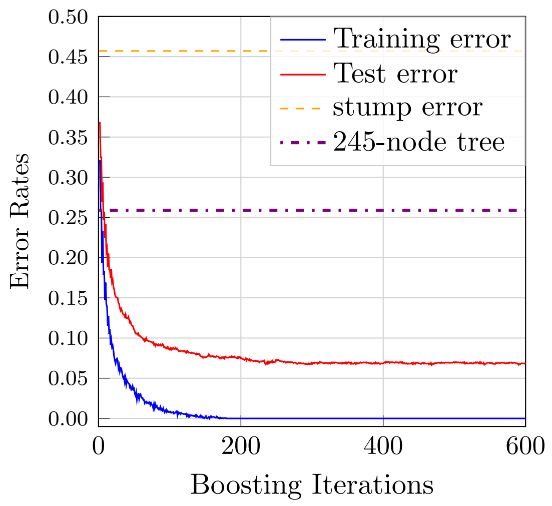
$$\underset{\{\beta_m, \boldsymbol{\gamma}_m\}_1^M}{\text{minimize}} \sum_{i=1}^{N} L\left(y_i, \sum_{m=1}^{M} \beta_m b(\mathbf{x}_i; \boldsymbol{\gamma}_m) \right). \tag{4.2}$$



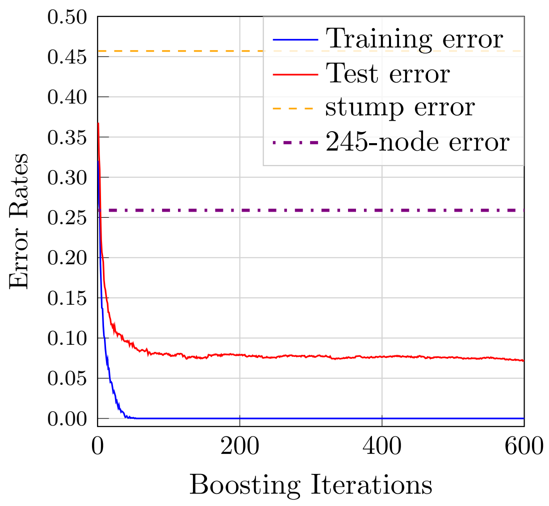
(A) Adaboost.M1 with stumps



(B) AdaBoost.R with stumps



(C) AdaBoost.M1 with 8-node tree



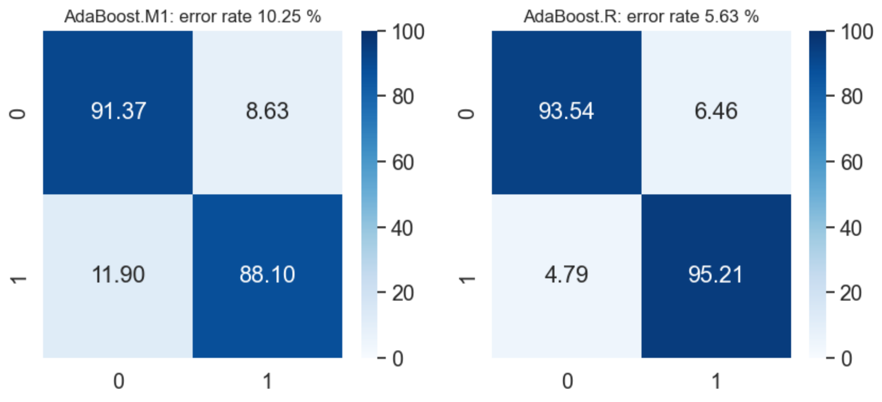
(D) AdaBoost.R with 8-node tree

FIGURE 4.1: *Results for simulated data of Example 4.1. Error rates as a function of the number of boosting iterations when using AdaBoost.M1 and AdaBoost.R when the weak learner is (a),(b) a stump and (c),(d) eight-node trees. Also shown are the test error rate for a single stump, and a 244-node classification tree*

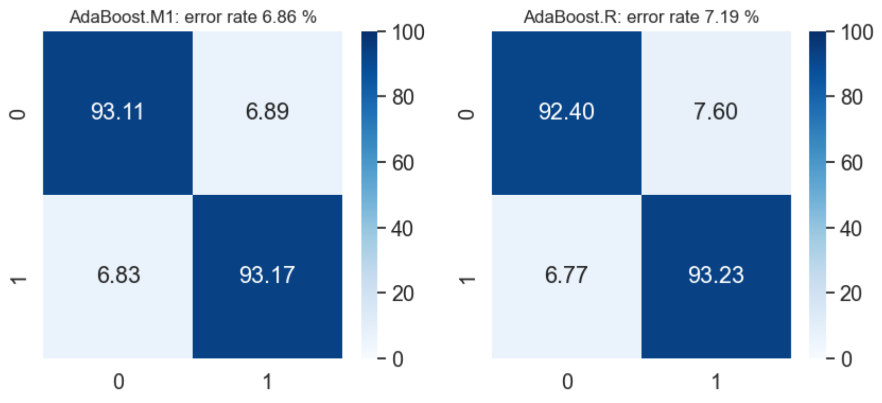
Solving this problem is often computationally infeasible. However, a simple approximate solution can be developed when it is feasible to rapidly solve the subproblem of fitting just a single basis function:

$$\underset{\beta,\gamma}{\text{minimize}} \sum_{i=1}^{N} L\Big(y_i, \beta b(\mathbf{x}_i;\gamma)\Big).$$

This is the idea in **forward stagewise modeling**, which *approximates* the solution to (4.2) by sequentially adding new basis functions to the expansion without adjusting the parameters and coefficients of those that have already been added.



(A) stumps



(B) 8-node trees

FIGURE 4.2: *Confusion matrices for AdaBoost.M and .R with stumps and 8 node trees ($M = 600$).*

Algorithm 4.3: Forward Stagewise Additive Modeling

```
  Initialize: $f_0(\mathbf{x}) = 0$
1 for $m = 1$ to $M$ do
2     Compute
          $(\beta_m, \boldsymbol{\gamma}_m) = \arg\min_{\beta,\boldsymbol{\gamma}} \sum_{i=1}^{N} L\big(y_i, f_{m-1}(\mathbf{x}_i) + \beta b(\mathbf{x}_i; \boldsymbol{\gamma})\big)$
3     Set $f_m(\mathbf{x}) = f_{m-1}(\mathbf{x}) + \beta_m b(\mathbf{x}; \boldsymbol{\gamma}_m)$.
```

This procedure, detailed in Algorithm 4.3, proceeds as follows:

- At each iteration m, one solves for the optimal basis function $b(\mathbf{x}; \boldsymbol{\gamma}_m)$ and corresponding coefficient $\boldsymbol{\gamma}_m$ to add to the current expansion $f_{m-1}(\mathbf{x})$.
- This produces $f_m(\mathbf{x})$, and the process is repeated. Previously added terms are not modified.

For squared-error loss one has

$$\begin{aligned} L(y_i, f_{m-1}(\mathbf{x}_i) + \beta b(\mathbf{x}_i; \boldsymbol{\gamma})) &= (\underbrace{y_i - f_{m-1}(\mathbf{x})}_{= r_{im}} - \beta b(\mathbf{x}_i; \boldsymbol{\gamma}))^2 \\ &= \big(r_{im} - \beta b(\mathbf{x}_i; \boldsymbol{\gamma})\big)^2. \end{aligned} \tag{4.3}$$

Thus, for squared-error loss, the term $\beta_m b(\mathbf{x}; \boldsymbol{\gamma}_m)$ that best fits the current residuals $\{r_{im}\}_{i=1}^{N}$ is added to the expansion at each step.

4.4 Exponential Loss + Stagewise additive modeling = AdaBoost.M1

Next we show that in binary classification problem, AdaBoost.M1 (Algorithm 4.1) is equivalent to forward stagewise additive modeling (Algorithm 4.3) that uses the exponential loss

$$L(y, f(\mathbf{x})) = \exp(-y f(\mathbf{x})) \tag{4.4}$$

and where the individual classifiers $G_m(\mathbf{x}) \in \{-1, 1\}$ take the role of the basis functions $b(\mathbf{x}; \boldsymbol{\gamma})$.

When $L(\cdot, \cdot)$ is the exponential loss in (4.4), the classifier G_m and corresponding coefficient β_m to be added at each step solve

$$\begin{aligned} (\beta_m, G_m) &= \arg\min_{\beta, G} \sum_{i=1}^{N} \exp\big\{ -y_i\big(f_{m-1}(\mathbf{x}_i) + \beta G(\mathbf{x}_i)\big)\big\} \\ &= \arg\min_{\beta, G} \sum_{i=1}^{N} w_i^{(m)} \exp\big(-\beta y_i G(\mathbf{x}_i)\big), \end{aligned} \tag{4.5}$$

where

$$w_i^{(m)} = \exp\big(-y_i\, f_{m-1}(\mathbf{x}_i)\big) \tag{4.6}$$

can be regarded as a weight that is applied to each observation.

The solution to (4.5) can be obtained in two steps. First, keeping β fixed, we can solve for G:

$$G_m = \arg\min_G \sum_{i=1}^N w_i^{(m)} \mathbf{1}_{\{y_i \neq G(\mathbf{x}_i)\}} \tag{4.7}$$

which is the classifier that minimizes the weighted error rate in predicting y. This follows by noting that the criterion in (4.5) may be written as

$$\begin{aligned}
&\sum_{i=1}^N w_i^{(m)} \exp\big(-\beta y_i G(\mathbf{x}_i)\big) \\
&\qquad = e^{-\beta} \sum_{y_i = G(\mathbf{x}_i)} w_i^{(m)} + e^{\beta} \sum_{y_i \neq G(\mathbf{x}_i)} w_i^{(m)} \\
&\qquad = (e^{\beta} - e^{-\beta}) \sum_{i=1}^N w_i^{(m)} \mathbf{1}_{\{y_i \neq G(\mathbf{x}_i)\}} + e^{-\beta} \sum_{i=1}^N w_i^{(m)}.
\end{aligned}$$

Then, after we plug in this G_m into (4.5) and solve for β we obtain

$$\beta_m = \frac{1}{2} \log \frac{1 - \mathrm{err}_m}{\mathrm{err}_m}, \tag{4.8}$$

where

$$\mathrm{err}_m = \frac{\sum_{i=1}^N w_i^{(m)} \mathbf{1}_{\{y_i \neq G_m(\mathbf{x}_i)\}}}{\sum_{i=1}^N w_i^{(m)}} \tag{4.9}$$

is the minimized weighted error rate (cf. Line 3 of Algorithm 4.1).

The approximation is then updated

$$f_m(\mathbf{x}) = f_{m-1}(\mathbf{x}) + \beta_m G_m(\mathbf{x})$$

which causes the weights in (4.6) for the next iteration to be

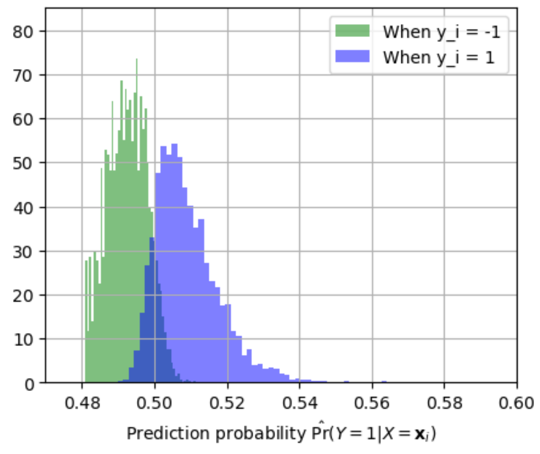
$$\begin{aligned}
w_i^{(m+1)} &= w_i^{(m)} \cdot e^{-\beta_m\, y_i\, G_m(\mathbf{x}_i)} \\
&= w_i^{(m)} \cdot e^{\alpha_m \mathbf{1}_{\{y_i \neq G_m(\mathbf{x}_i)\}}} \cdot e^{-\beta_m}
\end{aligned} \tag{4.10}$$

where $\alpha_m = 2\beta_m$ is exactly the quantity defined at line 4 of Algorithm 4.1. In obtaining (4.10) we used that

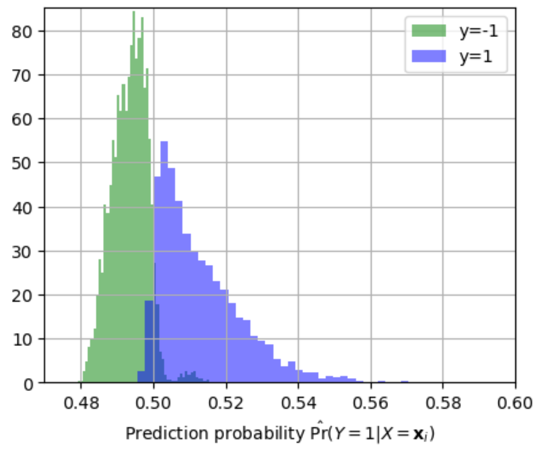
$$-y\, G(\mathbf{x}) = 2 \cdot \mathbf{1}_{\{y \neq G(\mathbf{x})\}} - 1.$$

The constant $e^{-\beta_m}$ in (4.10) multiplies all weights by the same value, so it can be ignored. Thus (4.10) is exactly equivalent to line 5 of Algorithm 4.1.

Remark 4.1. Recall from Table 2.1 that the optimum predictor function $f^*(\mathbf{x})$ that mininizes the exponential loss is one half of log odds. Thus we can interpret the additive expansion $\hat{f}(\mathbf{x}) = \sum_{m=1}^M \alpha_m G_m(\mathbf{x})$ of AdaBoost as an estimate of log-odds of $p(\mathbf{x}) = \Pr(Y = 1 | X = \mathbf{x})$. Note that it is not $\frac{1}{2}\times$ times log-odds since the actual



(A) AdaBoost.M1



(B) AdaBoost.R

FIGURE 4.3: *Results for simulated data of Example 4.1. Density histograms of prediction probabilities $\hat{p}(\mathbf{x})$ for test data. The density on light blue (resp. green) are test data cases with $y_i = 1$ (resp. $y_i = -1$).*

multiplier is $\beta_m = \alpha_m/2$ and the scaling by $1/2$ was ignored by convenience. This justifies using its sign as the classification rule in AdaBoost.M1. Moreover, due to (2.22) it allows us to define probability estimates $\hat{p}(\mathbf{x}) = \hat{\Pr}(Y = 1 \mid X = \mathbf{x})$ simply as

$$\hat{p}(\mathbf{x}) = \sigma(\hat{f}(\mathbf{x})) = \frac{1}{1 + e^{-\hat{f}(\mathbf{x})}}$$

where

$$\hat{f}(\mathbf{x}) = \sum_{m=1}^{M} \alpha_m G_m(\mathbf{x}) / \sum_{m=1}^{M} \alpha_m$$

is weighted mean predictions in the ensemble. This is what `.predict_proba` returns when using the `AdaBoostClassifier` with option `algorithm='SAMME'`. Figure 4.3 shows the normalized histograms of predictions probabilities $\hat{p}(\mathbf{x})$ for test data. The density on light blue (resp. red) are observations with $y_i = 1$ (resp. $y_i = -1$). These we would like to have prediction probabilities > 0.5 (resp. < 0.5). Thus we would like both histograms to be located above (resp. below) 0.5 vertical line with no overlap. As can be noted, AdaBoost.M1 has more unwanted overlaps than AdaBoost.R.

4.5 Discussion

We only touched a surface of vast literature on boosting in this chapter. There are tons of extensions of the boosting principle. Here we mention only three most well-known methods that your future boss may expect you to know when you apply to data / ML scientist position:

- *LogitBoost* [Friedman et al., 2000] fits an additive logistic regression models by stagewise optimization of the Bernoulli log-likelihood. Instead of minimizing the exponential loss as AdaBoost (which is an approximation of the Bernoulli log-likelihood), it minimizes the Bernoulli log-likelihood directly.

- *Gradient Boosting Machine (GBM)* developed by Friedman [2001] is a more general statistical framework for boosting that is based on its connection to **functional gradient descent (FGD)**, allowing to interpret boosting as a method for function estimation.
- *XGBoost* [Chen and Guestrin, 2016] is highly popular optimized distributed gradient boosting algorithm that builds on GBM framework. In mid 2010-s it was in many winning ML architectures in kaggle competitions.

Bibliography

Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.

Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. *Classification and regression trees*. CRC press, 1984.

Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997.

Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media, 2 edition, 2009.

Robert E Schapire. The strength of weak learnability. *Machine learning*, 5(2):197–227, 1990.

Yoav Freund and Robert E Schapire. Experiments with a new boosting algorithm. In *Icml*, volume 96, pages 148–156. Citeseer, 1996.

Jerome Friedman, Trevor Hastie, Robert Tibshirani, et al. Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). *The annals of statistics*, 28(2):337–407, 2000.

Robert E Schapire and Yoram Singer. Improved boosting algorithms using confidence-rated predictions. *Machine learning*, 37(3):297–336, 1999.

Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 785–794, 2016.