Università degli Studi di Padova
Dipartimento di Scienze Statistiche

Corso di Laurea Magistrale in
Scienze Statistiche

Un approfondimento sul problema della classificazione in presenza di una classe rara

Relatore: Prof.ssa Giovanna Menardi
Dipartimento di Scienze Statistiche

Laureanda: Martina Dossi
Matricola N. 1130649

Anno Accademico 2017/2018

Indice

Introduzione **1**

1 Classificazione in presenza di una classe rara **5**
1.1 Introduzione 5
1.2 Natura del problema 6
1.3 Valutazione di un modello 9
1.3.1 Stima dell'errore condizionato 10
1.3.2 Criteri di valutazione 11

2 Alcuni rimedi al problema dello sbilanciamento **17**
2.1 Introduzione 17
2.2 Metodi di ricampionamento 18
2.3 Metodi che agiscono sull'algoritmo 20
2.4 Metodi basati sul *boosting* 22
2.4.1 L'algoritmo *AdaBoost* 22
2.4.2 Il *gradient boosting* 26
2.4.3 *Cost-sensitive boosting* 28
2.4.4 Metodi di *boosting* per classi sbilanciate 30

3 Delle proposte basate sull'AUC **33**
3.1 Introduzione 33
3.2 Una variante dell'albero di classificazione 34
3.3 Il *gradient boosting* con funzione di perdita basata sull'AUC 38

4 Analisi empirica **43**
4.1 Introduzione 43
4.2 Studio di simulazione 43
4.2.1 Definizione degli scenari 44
4.2.2 Modelli concorrenti 46
4.2.3 Risultati 49
4.3 Applicazione a dati reali 56

Conclusione **60**

A Codice R **61**

Bibliografia **73**

Introduzione

Un modello di classificazione è una funzione matematica che determina univocamente la classe di appartenenza di un'unità statistica, sulla base dei valori osservati per le variabili di interesse. La sua accuratezza previsiva dipende dall'abilità di classificare correttamente nuove unità, qualsiasi sia la classe da cui provengono. Tuttavia, quando i dati presentano un forte sbilanciamento nella distribuzione delle classi, la qualità di molti modelli di classificazione rischia di essere compromessa, con un'accuratezza asimmetrica fra le classi che favorisce la più numerosa. Un'assunzione implicita, quando si fa riferimento ad un problema di dati sbilanciati, è che la classe più rilevante per l'analisi sia invece quella rara e che sia pertanto fondamentale riconoscerla correttamente. In questo molti modelli falliscono perché sono sopraffatti dalla classe prevalente, assumendo una distribuzione equa fra le classi e costi di errata classificazione uguali, mentre commettere un errore su un'unità della classe rara ha tipicamente costi maggiori nel contesto in questione. L'obiettivo è pertanto quello di recuperare l'accuratezza previsiva sulla classe minoritaria, con adeguati accorgimenti che ne rafforzino il peso all'interno delle analisi.

Il fallimento della maggior parte dei modelli di classificazione può dipendere dal fatto che, in fase di stima, sia prevista l'ottimizzazione di un criterio di accuratezza globale, che tende per costruzione a favorire la classe prevalente. Il principale contributo di questa tesi nasce dall'idea di sfruttare i benefici offerti dall'AUC in quanto metrica di valutazione indipendente dalla distribuzione sottostante delle classi ed estenderli anche alla fase di stima di un modello. In questa prospettiva, sono stati appositamente modificati due modelli di classificazione non specificatamente progettati per la gestione di classi rare ed è stato predisposto il codice necessario alla loro implementazione.

Il primo metodo proposto trae spunto dal classico albero di classificazione, un modello frequentemente impiegato nei processi decisionali perché semplice e di immediata interpretabilità. Nella sua versione originaria, l'albero cresce minimizzando una qualche misura di impurità, come l'entropia o l'indice di Gini, con l'obiettivo di ridurre progressivamente l'eterogeneità fra le classi. Intuitivamente, quando lo sbilanciamento fra due classi è molto accentuato, il contributo di quella prevalente rischia di sovrastare, se non annullare, quello della classe minoritaria, costringendo l'albero ad adattarsi troppo meticolosamente ai dati dell'insieme di stima pur di riconoscerla. Sulla base di queste considerazioni, si introduce un criterio alternativo per la crescita dell'albero basato sull'AUC, in grado di tenere conto simultaneamente degli errori di entrambe le classi senza trascurare quella rara. Come risulterà chiaro in seguito, l'albero così costruito incorpora un naturale criterio di arresto che dipende anch'esso dall'AUC.

Il secondo metodo è invece basato sul *gradient boosting*, un algoritmo di classificazione più recente ed estremamente efficace in molti contesti diversi. La procedura prevede l'ottimizzazione di una qualche funzione di perdita e la specificazione di un modello di base, elementi che possono essere scelti arbitrariamente e che ne determinano una grande flessibilità. La nuova proposta è costruita derivando una funzione di perdita che ottimizza direttamente l'AUC da inserire all'interno del *gradient boosting*, di cui sarà considerata una particolare formulazione. La funzione è un'approssimazione del complemento a uno della statistica di Wilcoxon-Mann-Whitney, equivalente all'AUC, appositamente adattata per assicurarne la differenziabilità e il suo impiego all'interno dell'algoritmo. Infatti, mentre l'approssimazione fa riferimento a coppie di unità con una duplice sommatoria, la funzione di perdita che si richiede nell'algoritmo considera singolarmente ogni osservazione, rendendo quindi necessario questo ulteriore adeguamento.

Nel primo capitolo si introduce il problema della classificazione binaria, restringendo il campo al caso di interesse in cui la classe positiva è rara. Si spiegano le ragioni per cui molti classici metodi di classificazione soffrono della presenza di una classe rara, approfondendo la natura del problema e descrivendo quelle caratteristiche dei dati che possono interagire con esso e aggravarne le conseguenze. Lo sbilanciamento dei dati è problematico non soltanto in fase di stima di un modello, ma anche nella valutazione della sua accuratezza previsiva. Le usuali metriche di valutazione che si basano su criteri di accuratezza globale possono portare a conclusioni

fuorvianti e sono pertanto sostituite da misure in grado di valutare separatamente gli errori provenienti dalle due classi. Si dedica particolare attenzione alla descrizione della curva ROC e dell'AUC, la metrica maggiormente adottata nel contesto in questione.

Il secondo capitolo presenta vari approcci che possono essere perseguiti per gestire adeguatamente il problema dello sbilanciamento. Tra questi vi sono i metodi di ricampionamento, che consentono di alterare preliminarmente la distribuzione delle classi per ridurne lo sbilanciamento prima di stimare un qualche modello di classificazione, e le modifiche che incidono direttamente sull'algoritmo, finalizzate ad orientarne l'attenzione verso la classe minoritaria. A seguire, viene introdotta una panoramica dei metodi di *boosting*, di cui si approfondiscono nel dettaglio i due algoritmi più diffusi, *AdaBoost* e il *gradient boosting*, e se ne presentano alcune varianti, sia quelle che inglobano specifici costi di errata classificazione per le unità delle due classi, sia altre che integrano la procedura con tecniche di ricampionamento, sviluppate specificatamente per far fronte al problema dello sbilanciamento.

Nel terzo capitolo vengono spiegate dettagliatamente le due nuove proposte, di cui si sono già introdotte le principali motivazioni e i tratti distintivi.

Nel quarto e ultimo capitolo si presenta un confronto empirico fra alcuni dei più rappresentativi modelli basati sul *boosting* e i nuovi metodi proposti, per valutarne la qualità previsiva in contesti caratterizzati dalla presenza di una classe rara. In particolare, nella prima parte si riporta uno studio di simulazione, con strutture di dati prestabilite e di complessità variabile su cui stimare tutti i modelli, mentre nella seconda parte si restringe l'analisi alle sole nuove proposte e ai modelli direttamente confrontabili con esse per verificarne l'adeguatezza in contesti reali.

È riportata in Appendice l'implementazione dei nuovi metodi e di altre procedure a cui sarà fatto riferimento nella tesi.

Capitolo 1
Classificazione in presenza di una classe rara

1.1 Introduzione

Un problema di classificazione consiste nell'individuazione di una regola che assegni univocamente ogni osservazione ad una particolare categoria, sulla base di un insieme di dati per i quali è nota a priori la classe di appartenenza di ciascuno.

Sia $\mathcal{S}_n$=$\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$ l'insieme dei dati disponibili, denominato campione o insieme di stima, che include n coppie $(\mathbf{x}, y)$, dove $\mathbf{x}$ è un vettore di variabili esplicative e y una variabile di interesse, su cui si assume che le esplicative abbiano una certa influenza. Il vettore $\mathbf{x}$ è definito in uno spazio d-dimensionale $\mathcal{X}$ che è l'unione di domini continui, discreti e categoriali, mentre y assume valori in uno spazio categoriale $\mathcal{Y} = \{\mathcal{Y}_0, \ldots, \mathcal{Y}_{\mathcal{G}-1}\}$ che comprende $\mathcal{G}$ possibili etichette di classe. Si assume che le n coppie di osservazioni siano unità indipendenti e identicamente distribuite, provenienti da una distribuzione di probabilità congiunta non nota $\mathcal{F}$ definita su $\mathcal{X} \times \mathcal{Y}$. In particolare, nel lavoro che segue l'attenzione sarà circoscritta al caso in cui la variabile risposta è dicotomica, ossia $y \in \mathcal{Y} = \{\mathcal{Y}_0, \mathcal{Y}_1\}$, dove, per convenzione, $\mathcal{Y}_0$ indica la classe negativa e $\mathcal{Y}_1$ quella positiva, una notazione di evidente richiamo medico che implica un diverso grado di importanza per le due categorie, a favore della seconda.

Un modello di classificazione, o classificatore, costruito sull'insieme $\mathcal{S}_n$, produce una regola $\mathcal{H} : \mathcal{X} \mapsto \mathbb{R}$ che consente di elaborare previsioni per future unità di cui sono stati osservati solo i valori delle variabili esplicative. Le quantità $\mathcal{H}(\mathbf{x})$ sono dei

punteggi che misurano il grado di confidenza con cui le osservazioni appartengono ad una certa classe, che, per convenzione, si assume essere quella di maggior interesse. Per alcuni modelli, $\mathcal{H}(\mathbf{x})$ può rappresentare una stima della probabilità $\mathbb{P}[y = \mathcal{Y}_1|\mathbf{x}]$ e appartenere quindi all'intervallo più ristretto $[0, 1]$. Si stabilisce la categoria prevista $\hat{y}$ introducendo un'opportuna soglia di classificazione $k \in \mathbb{R}$ tale che:

$$\hat{y} = \begin{cases} \mathcal{Y}_0 & \text{se } \mathcal{H}(\mathbf{x}) < k, \\ \mathcal{Y}_1 & \text{altrimenti.} \end{cases} \tag{1.1}$$

Quando $\mathcal{H}(\mathbf{x})$ è la stima di una probabilità, la scelta più naturale per la soglia è 0.5.

Il problema sarà ulteriormente ristretto al contesto in cui la classe positiva, oltre ad essere di maggiore interesse, non è sufficientemente rappresentata e si caratterizza per una numerosità fortemente inferiore rispetto all'altra classe. Quando le unità positive sono rare il modello può facilmente interpretarle come degli errori ed ignorarle, con la conseguenza di produrre molte più previsioni errate sulla classe minoritaria piuttosto che su quella prevalente e di minor interesse.

Il problema della classificazione in presenza di una classe rara ha attirato sempre più attenzione negli ultimi anni (Yang e Wu, 2006) sia perché pervade molti ambiti applicativi sia in quanto molti fra i più popolari modelli di classificazione risultano inadeguati nell'individuare correttamente la classe rara. Tra i lavori più significativi che offrono un quadro generale del problema si citano quelli di Sun *et al.* (2009) e Weiss (2004).

1.2 Natura del problema

I domini reali che si caratterizzano per la presenza di classi fortemente sbilanciate sono i più svariati e ciascuno è stato, in varia misura, oggetto di specifici studi mirati a gestire il problema così come si manifesta in ogni ambito. Tra gli esempi più ricorrenti ci sono l'individuazione di comportamenti fraudolenti nel settore finanziario (García *et al.*, 2012) e in quello delle telecomunicazioni (Olszewski, 2012), la diagnosi di malattie rare (Mazurowski *et al.*, 2008), il riconoscimento di espressioni geniche tumorali (Yu *et al.*, 2012) e di sequenze di proteine (Al-Shahib *et al.*, 2005).

La distribuzione sbilanciata fra le classi può essere un aspetto intrinseco dei dati, e accadere quindi naturalmente, oppure estrinseco, ossia imposto da cause esterne.

Il primo caso include quei contesti in cui la classe rara rappresenta un concetto ben circoscritto, mentre l'altra ne è la controparte e come tale può includere un numero indefinitamente maggiore di osservazioni. Il secondo caso emerge invece quando i dati per natura non sono sbilanciati, ma lo diventano nel momento in cui risulta troppo oneroso raccogliere osservazioni di una classe, in termini di costi economici, di tempo o per via di aspetti legati alla *privacy*. Inoltre, lo sbilanciamento fra le classi può essere assoluto oppure relativo, intendendo con quest'ultimo che la cardinalità di una classe è molto più grande di quella dell'altra, ma che sono stati comunque raccolti sufficienti dati per entrambe (per un ulteriore approfondimento, si veda He e Garcia, 2009).

Molti dei più comuni modelli di classificazione, sia parametrici che non parametrici, come la regressione logistica, il metodo dei vicini più prossimi, gli alberi di classificazione e le reti neurali, si rivelano inadeguati a trattare dati per cui la distribuzione delle classi è particolarmente sbilanciata. A titolo esemplificativo, si pensi al metodo dei vicini più prossimi, esso tenderà a classificare erroneamente molti casi rari per il semplice fatto che la maggior parte dei vicini di un'unità appartenente alla classe rara avrà più probabilità di appartenere alla classe prevalente (Mani e Zhang, 2003). Un albero di classificazione, invece, dovrà crescere molto in profondità per cogliere la classe rara, rischiando di sovra-adattarsi ai dati, o, altrimenti, in seguito alla fase di potatura, potrà finire per trascurare completamente le unità positive (Chawla, 2003). In generale, questi e altri metodi falliscono perché vengono sopraffatti dalla classe prevalente finendo così per ignorare gli esempi della classe rara. Alcuni modelli di classificazione sono derivati ottimizzando una qualche funzione obiettivo sui dati dell'insieme di stima che tipicamente si basa sull'accuratezza globale. In questi casi la classe favorita sarà inevitabilmente la più numerosa, cioè quella che contribuisce maggiormente a tale metrica. Banalmente, in un insieme di dati dove per ogni unità positiva ce ne sono 99 della classe negativa, una regola di classificazione potrebbe raggiungere un'accuratezza del 99% etichettando tutti i soggetti come negativi. In apparenza ottimo, un simile risultato è invece inutile se l'interesse dello studio è quello di identificare la classe positiva. Si nota quindi la presenza di un'assunzione implicita quando si fa riferimento ad un problema di classi sbilanciate. Infatti, se la classe più numerosa fosse anche quella di maggiore interesse, non sarebbe così grave assegnare ad essa tutti i soggetti dello studio; viceversa, la dominanza della classe maggiore diventa un serio ostacolo al problema della clas-

sificazione quando la classe più importante è quella rara, la cui errata classificazione comporta costi maggiori. Nel Capitolo 2 saranno presentati alcuni possibili rimedi per gestire il problema, tali da orientare l'attenzione dell'algoritmo verso la classe minoritaria.

Varie ricerche sia teoriche che empiriche mostrano come, ad inficiare la qualità di un modello di classificazione in presenza di una classe rara, non sia soltanto il grado di sbilanciamento fra le classi, ma che, piuttosto, siano da accusare una serie di altre condizioni concomitanti che interagiscono con lo sbilanciamento, di cui si discute, ad esempio, nei lavori di Japkowicz e Stephen (2002) e di López *et al.* (2013). Come evidenziano tali studi, non vi è un'esplicita e chiara relazione fra il grado di sbilanciamento che caratterizza un insieme di dati e la *performance* di un modello. Infatti, una distribuzione equa fra le classi non garantisce che un modello sia poi accurato (Visa e Ralescu, 2005) e dati che esibiscono uno sbilanciamento anche meno severo possono essere più problematici di altri con una classe estremamente rara. Questo suggerisce quindi l'esistenza di altre potenziali aggravanti che possono rendere complicato un problema di dati sbilanciati. In qualsiasi contesto è risaputo come la numerosità campionaria giochi un ruolo cruciale nel determinare la qualità di un modello. Se esigua, infatti, può impedire di cogliere regolarità nei dati, ma, quando una classe è già rara, avere ancora meno osservazioni complica notevolmente il problema. In varie ricerche, come quella sostenuta da Japkowicz e Stephen (2002), è stato verificato che un classificatore può non risentire di un livello di sproporzione anche molto elevato fra le classi se questo viene compensato da un numero sufficientemente grande di unità nell'insieme di stima. Viceversa, se la classe rara non è adeguatamente rappresentata, il modello per riconoscerla rischia di adattarsi troppo minuziosamente ai dati. Il secondo elemento rilevante riguarda la possibilità di sovrapposizione fra le classi. Quanto più questo si verifica, tanto più arduo diventa per il classificatore determinare una regola discriminatoria fra le due classi. Nel lavoro di Prati *et al.* (2004) vengono costruiti dei *dataset* artificiali con varie combinazioni di sbilanciamento e sovrapposizione fra le due classi, per poi stimare e confrontare dei modelli e giungere alla conclusione che proprio il livello di sovrapposizione gioca il ruolo più decisivo sulla loro *performance*. In questi scenari, i tradizionali modelli che massimizzano l'accuratezza globale, oltre a non distinguere correttamente le classi, tendono ad assegnare le aree di sovrapposizione alla più numerosa, assimilando le unità positive che vi cadono all'interno a rumore. Vi è

infine un ultimo fattore che, in aggiunta allo sbilanciamento delle classi, può notevolmente compromettere i risultati di un modello, ossia quando il concetto spiegato dalla classe rara si manifesta in più *sotto-concetti* (si citano, ad esempio, i lavori di Japkowicz, 2003 e Weiss, 2010). Il fatto che una classe possa essere costituita da più sottoinsiemi è comune a molti problemi, ma quando le unità che vi appartengono sono già rare allora questo può rendere veramente insidioso distinguerle da rumore perché si creano degli insiemi disgiunti di unità positive sia lontani nello spazio che scarsamente rappresentati. Per le questioni presentate, si evince come insiemi di dati con lo stesso grado di sbilanciamento fra le classi non possano di fatto essere equiparati, dato che sulla complessità di ciascuno incidono in varia misura diversi altri elementi.

1.3 Valutazione di un modello

In un contesto di dati sbilanciati, non solo la stima di un modello di classificazione è problematica, ma lo è anche la valutazione della sua accuratezza. Per poter scegliere la migliore regola di classificazione entro un insieme di modelli concorrenti è fondamentale poter fare affidamento su una metrica di valutazione adeguata, che tenga opportunamente conto della presenza di una classe rara all'interno dell'insieme di dati.

La qualità di un modello di classificazione riflette la sua abilità di prevedere correttamente la classe di appartenenza di una nuova unità $(\mathbf{x}_0, y_0)$ ed è, per questo, indicata anche come *capacità di generalizzazione*. Avendo stimato un modello $\mathcal{H}$ sui dati dell'insieme di stima $\mathcal{S}_n$, si misura la discrepanza fra il vero valore y_0 e quello previsto $\hat{y}_0$ tramite una funzione di perdita $\mathrm{L}(y_0, \hat{y}_0)$ che è, per definizione, non negativa. A seconda dei casi, la funzione di perdita può anche essere specificata in funzione del punteggio $\mathcal{H}(\mathbf{x}_0)$ assegnato dal classificatore alla nuova unità. L'*errore di generalizzazione* del modello $\mathcal{H}$, o *errore condizionato*, viene definito come:

$$\mathrm{Err}_{\mathcal{S}} = \mathbb{E}_{\mathcal{F}(\mathbf{x}_0, y_0)}[\mathrm{L}(y_0, \mathcal{H}(\mathbf{x}_0))|\mathcal{S}_n]. \tag{1.2}$$

Il nuovo punto $(\mathbf{x}_0, y_0)$ rappresenta la quantità aleatoria nell'espressione (1.2) e proviene dalla stessa distribuzione di probabilità congiunta $\mathcal{F}$ dei dati usati per la stima del modello. Si assume, invece, che sia fissato l'insieme $\mathcal{S}_n$, rispetto a cui è condizionato il valore atteso. Per valutare l'accuratezza di un classificatore $\mathcal{H}$,

quindi, è necessario calcolare una stima del suo errore di generalizzazione (1.2) e, secondariamente, specificare una funzione di perdita opportuna.

1.3.1 Stima dell'errore condizionato

La qualità di un classificatore può essere valutata in modo affidabile solo a patto di aver ottenuto una stima realistica del suo vero errore $\mathrm{Err}_{\mathcal{S}}$. Tipicamente, l'*errore di stima*, ossia l'errore calcolato come perdita media sulle n unità del campione di stima:

$$\overline{\mathrm{err}} = \frac{1}{n} \sum_{i=1}^{n} \mathrm{L}(y_i, \mathcal{H}(\mathbf{x}_i)), \tag{1.3}$$

è minore del vero errore $\mathrm{Err}_{\mathcal{S}}$. Questo accade perché vengono usati gli stessi dati sia per stimare il modello che per misurarne l'errore. Quanto più un modello è complicato, tanto più esso si adatta alle unità dell'insieme di stima, seguendo anche articolate strutture sottostanti. La sua distorsione si riduce, ma ne aumenta la varianza, intendendo con distorsione la capacità di riprodurre il fenomeno non noto di interesse e con varianza la sensibilità a variazioni nel campione di stima. Si dice, equivalentemente, che il modello si *sovra-adatta* ai dati. Quando si valuta e si sceglie il grado di complessità di un modello è fondamentale ricercare il bilanciamento ottimale fra distorsione e varianza; se troppo complicato, un modello rischia di fornire previsioni inaffidabili, essendosi adattato anche alle irregolarità casuali del campione di stima (si veda, per un approfondimento, Azzalini e Scarpa, 2012, Cap. 3). L'errore di stima $\overline{\mathrm{err}}$ diminuisce invece all'aumentare della complessità del modello, producendo una stima sempre più distorta e ottimistica dell'errore di generalizzazione. Questa discrepanza può essere anche intuita notando che nell'espressione (1.2) del vero errore, la nuova unità $(\mathbf{x}_0, y_0)$ rispetto a cui è espresso il valore atteso è aleatoria e non appartiene necessariamente all'insieme $\mathcal{S}_n$. Alcune possibilità, quindi, più coerenti per stimare l'errore di generalizzazione sono il metodo *holdout* e la convalida incrociata. Il primo consiste nel dividere casualmente il campione di dati in due parti, utilizzando la prima come insieme di stima e la seconda come insieme di verifica, su cui valutare l'accuratezza del modello e calcolare una stima del suo errore. Il metodo della convalida incrociata, invece, non soffre dell'arbitrarietà associata ad un'unica bipartizione dei dati, ma li suddivide in più sottoinsiemi, usandone a rotazione uno come insieme di verifica e i restanti per la stima. L'errore viene poi

ricavato come media degli errori sui sottoinsiemi. Per una trattazione esaustiva di questi e altri metodi si rimanda a Hastie *et al.* (2001, Cap. 7). Sia il metodo *hold-dout* che quello della convalida incrociata vengono frequentemente usati nelle analisi empiriche, soprattutto il primo essendo legato a costi computazionali inferiori, ma possono emergere dei problemi in presenza di dati sbilanciati. Può accadere, infatti, che non vi siano sufficienti unità della classe rara da poter essere ragionevolmente suddivise nei due o più insiemi richiesti dalla procedura e la scarsità dei dati può causare stime dell'errore altamente variabili.

1.3.2 Criteri di valutazione

L'errore di generalizzazione e la sua stima dipendono, naturalmente, dalla specificazione di un particolare criterio che metta in relazione valori previsti e valori osservati. Nel contesto della classificazione, una scelta comune è considerare la funzione di perdita *zero-uno*, definita come $\mathrm{L}(y, \hat{y}) = \mathbb{I}(y \neq \hat{y})$, essendo $\mathbb{I}(\cdot)$ una funzione indicatrice. Assumendo che $\mathcal{H}(\mathbf{x})$ sia una stima della probabilità $\mathbb{P}[y = \mathcal{Y}_1|\mathbf{x}]$, un'altra tipica funzione è l'*entropia incrociata*, chiamata anche funzione di perdita *logaritmica*: $\mathrm{L}(y, \mathcal{H}(\mathbf{x})) = -\mathbb{I}(y = \mathcal{Y}_1)\log\mathcal{H}(\mathbf{x}) - \mathbb{I}(y = \mathcal{Y}_0)\log(1 - \mathcal{H}(\mathbf{x}))$.

Ci sono, in generale, molti criteri diversi per misurare l'accuratezza di un modello di classificazione, alcuni più indicati di altri quando è presente un problema di classi sbilanciate. Tipicamente, il confronto tra i valori osservati y e le categorie previste $\hat{y}$ si basa su una matrice di confusione, o tabella di errata classificazione, come quella rappresentata nella Figura 1.1, in cui le quattro celle contengono le frequenze assolute dei veri positivi (VP), ossia i casi positivi correttamente classificati, dei falsi positivi (FP), i casi negativi previsti come positivi, dei veri negativi (VN), i casi negativi correttamente classificati e, infine, dei falsi negativi (FN), che sono i casi positivi erroneamente classificati come negativi. Scegliere di utilizzare la funzione di perdita zero-uno per valutare la qualità del modello è equivalente a calcolare il tasso di errata classificazione a partire dai dati riportati nella tabella, pari alla somma di falsi positivi e falsi negativi sulla numerosità totale. Tuttavia, come è stato ampiamente dimostrato (Weiss e Provost, 2001), l'utilizzo di una simile misura in presenza di classi fortemente sbilanciate porta a risultati inaffidabili e conclusioni fuorvianti. Oltre a essere distorto in favore della classe più numerosa, il tasso di errata classificazione considera ugualmente importanti entrambe le classi, e quindi anche gli

		Classe reale		
		$\mathcal{Y}_0$	$\mathcal{Y}_1$	Totale
Classe	$\mathcal{Y}_0$	VN	FN	VN+FN
Prevista	$\mathcal{Y}_1$	FP	VP	FP+VP
	Totale	VN+FP	FN+VP	

Figura 1.1: Matrice di confusione per un problema di classificazione binario.

errori di classificazione che ne derivano, mentre in questo contesto non riconoscere un'unità positiva può avere implicazioni molto più gravi che commettere errori sulla classe negativa. Per capire l'entità del problema, si consideri un semplice esempio in cui il campione di verifica comprende 100 osservazioni, di cui 90 appartenenti alla classe negativa e 10 a quella positiva. Si supponga, inoltre, che un classificatore binario commetta 8 errori, producendo un tasso di errata classificazione pari all'8%, qualsiasi siano le unità mal classificate. Proprio a causa di questo, tale misura rende indistinguibili due scenari diametralmente opposti: se gli 8 errori riguardassero la classe negativa, il tasso di falsi positivi (esempi negativi classificati come positivi) sarebbe pari al 9% e quello di falsi negativi (esempi positivi classificati come negativi) allo 0%, se invece gli errori colpissero la classe positiva, il tasso di falsi positivi sarebbe nullo, ma quello di falsi negativi salirebbe all'80%.

Alla luce di quanto spiegato, è evidente la necessità di una metrica di valutazione in grado di considerare separatamente gli errori associati ad ogni classe. Dalla matrice di confusione possono essere recuperate diverse misure:

- tasso di veri positivi (TVP), o *sensibilità*, o *recall* (R): VP/(FN+VP),
- tasso di falsi positivi (TFP): FP/(VN+FP),
- tasso di veri negativi (TVN), o *specificità*: VN/(VN+FP),
- tasso di falsi negativi (TFN): FN/(FN+VP),
- precisione, o *precision* (P): VP/(FP+VP).

Nessuna è tuttavia sufficiente per valutare un modello se interpretata singolarmente, ma ne vanno considerate congiuntamente più d'una. Per questo, sono state introdotte alcune loro utili combinazioni, come, ad esempio, la misura F, che unisce *precision*

e *recall* tramite la loro media armonica, oppure la media geometrica dell'accuratezza su entrambe le classi.

Come è stato specificato nel paragrafo 1.1, una regola di classificazione produce originariamente un punteggio che esprime il grado di confidenza o la probabilità stimata per ogni unità di appartenere alla classe positiva. Di conseguenza, per poter riassumere i risultati in una matrice di confusione, è necessario specificare una soglia k che renda l'output discreto. Per non dover condizionare la valutazione di un modello ad un unico valore della soglia, che è, in aggiunta, problematico definire quando i dati presentano un forte sbilanciamento fra le classi, una valida alternativa è offerta dalla curva ROC (*Receiving Operating Characteristic*), uno strumento che permette di studiare analiticamente il risultato di un modello di classificazione indipendentemente dalla soglia (Provost e Fawcett, 1997). La curva, di cui si riporta un esempio nella Figura 1.2, è ottenuta ponendo in ascissa il tasso di falsi positivi e in ordinata quello di veri positivi, calcolati al variare della soglia. Una delle particolarità legate all'utilizzo della curva ROC è che essa permette di valutare l'ordinamento delle unità imposto dal classificatore, valutandone in questo modo l'abilità nel discriminare tra unità positive e negative.

Nel grafico riportato, la linea rossa coincide con la strategia di assegnare casualmente le unità ad una o all'altra classe, ci si muove lungo la bisettrice al variare della frequenza con cui il classificatore casuale assegna le unità alla classe positiva. Per potersi spostare verso la regione triangolare superiore, il classificatore deve estrarre un qualche tipo di informazione utile dai dati. Il modello ideale, che prevede correttamente tutte le osservazioni, produce una curva ROC passante per il punto di

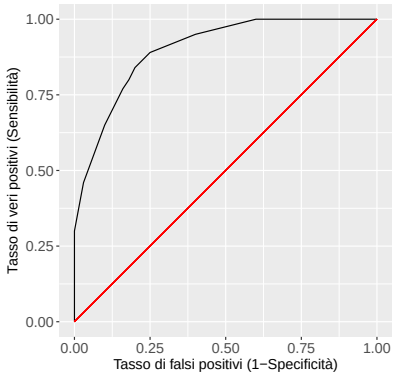

Figura 1.2: Un esempio di curva ROC.

coordinate (1,0), ossia tasso di veri positivi pari a 1 e di falsi positivi pari a 0, a prescindere dalla soglia. Un modello invece che giace al di sotto della bisettrice ha una *performance* peggiore della classificazione casuale. Si noti che lo spazio decisionale è comunque simmetrico rispetto alla linea rossa, così che ogni classificatore al di sotto di essa possa essere invertito per posizionarsi nella zona superiore.

Mettendo in luce il *trade-off* esistente fra i benefici (TVP) e i costi (TFP) di un modello stimato al variare della soglia, la curva ROC è quindi un'ottima espressione bidimensionale della sua *performance* e permette di selezionare il miglior modello entro un insieme di modelli concorrenti come quello la cui curva ROC sovrasta tutte le altre. Sfortunatamente, questo caso si verifica di rado, perché in generale le curve si incrociano e non ve n'è una che domina uniformemente le altre. Per questo motivo, è più comune fare riferimento ad un indicatore sintetico, chiamato AUC (*Area Under the Curve*, Bradley, 1997), pari all'area sottostante la curva. Trattandosi della porzione di un'area unitaria, l'AUC varia tra 0 e 1, ma dato che il classificatore casuale associato alla diagonale ha un'area pari a 0.5, nessun classificatore realistico dovrebbe avere un valore dell'AUC minore di 0.5.

L'AUC ha anche un'interpretazione particolarmente interessante dal punto di vista statistico. Siano $\mathcal{S}^+ = \{(\mathbf{x}_i^+, \mathcal{Y}_1)|i = 1, \ldots, n_+\}$ e $\mathcal{S}^- = \{(\mathbf{x}_j^-, \mathcal{Y}_0)|j = 1, \ldots, n_-\}$, rispettivamente, l'insieme delle unità positive e l'insieme di quelle negative, dove n_+ e n_- indicano la cardinalità di ciascuno. Di conseguenza, si denotano con $\mathcal{H}(\mathbf{x}_i^+)$ i punteggi assegnati dal classificatore alle unità positive e con $\mathcal{H}(\mathbf{x}_j^-)$ quelli per le unità negative. L'AUC è equivalente (Yan *et al.*, 2003) al valore della statistica di Wilcoxon-Mann-Whitney (Mann e Whitney, 1947):

$$AUC = \frac{1}{n_+ n_-} \sum_{i=1}^{n_+} \sum_{j=1}^{n_-} \mathbb{I}_{0.5}(\mathcal{H}(\mathbf{x}_i^+) - \mathcal{H}(\mathbf{x}_j^-)), \tag{1.4}$$

dove $\mathbb{I}_{0.5}(t)$ è una particolare funzione indicatrice definita come:

$$\mathbb{I}_{0.5}(t) = \begin{cases} 0 & \text{se } t < 0, \\ 0.5 & \text{se } t = 0, \\ 1 & \text{se } t > 0. \end{cases} \tag{1.5}$$

Pertanto, l'AUC è interpretabile come una misura basata sul confronto fra coppie di osservazioni appartenenti a classi diverse. In particolare, per il significato della

statistica di Wilcoxon-Mann-Whitney, l'AUC di un classificatore è la stima della probabilità che esso dia un punteggio più alto ad una unità positiva piuttosto che ad una negativa ed è pari a 1 quando tutte le unità positive ricevono un punteggio maggiore di quelle negative, caso in cui esiste una soglia che discrimina perfettamente le osservazioni fra le due classi. Se da una parte è ragionevole utilizzare l'AUC per valutare la *performance* di un classificatore, sarà comunque necessario definire una soglia opportuna per allocare le unità ad una o all'altra classe.

In conclusione, si cita un altro strumento molto simile alla curva ROC e anch'esso ampiamente utilizzato in questo contesto: la curva PR (*Precision-Recall*), che ha in ascissa i valori di *recall* e in ordinata quelli di *precision* al variare della soglia (Manning e Schütze, 1999). Mentre la scelta di un classificatore sulla base della curva ROC ricade su quella più vicina all'angolo in alto a sinistra, nel caso della curva PR si cerca la più vicina all'angolo in alto a destra. Lo studio di Davis e Goadrich (2006) mette in luce la relazione fra le due curve, dimostrando come fra esse intercorra una corrispondenza biunivoca, poiché ogni punto di ciascuna è determinato dalla stessa matrice di confusione, a patto che il valore di *recall* sia non nullo. Nonostante questo, algoritmi che massimizzano l'AUC non necessariamente ottimizzano anche l'area sottostante la curva PR.

Capitolo 2

Alcuni rimedi al problema dello sbilanciamento

2.1 Introduzione

Le strategie che vengono tipicamente perseguite per trattare insiemi di dati con un problema di classi sbilanciate si possono suddividere in due ampie categorie. Un primo approccio prevede di modificare preliminarmente i dati prima di sottoporli ad un qualche modello di classificazione, con l'obiettivo di attenuare il grado di sproporzione fra le classi, principalmente adottando delle tecniche di ricampionamento. La seconda strada, invece, coinvolge direttamente l'algoritmo di classificazione e prevede l'inserimento al suo interno di opportune modifiche che ne orientino l'attenzione verso la classe minoritaria. Tra queste soluzioni rivestono uno spazio rilevante gli algoritmi che inglobano diversi costi di errata classificazione per le unità delle due classi, assegnandone uno maggiore alle osservazioni più rare. Altri algoritmi, seppur non necessariamente progettati con lo scopo di gestire il problema dello sbilanciamento, sono quelli che nascono dalla combinazione di più modelli per rafforzarne la *performance* individuale, fra i quali, nello specifico, saranno approfonditi alcuni algoritmi di *boosting*, sia quelli classici che altri appositamente sviluppati per questo contesto.

2.2 Metodi di ricampionamento

Le soluzioni basate sui metodi di ricampionamento consistono in un pre-trattamento dei dati, che, come tale, ha il vantaggio di essere indipendente da qualsiasi modello di classificazione e adattabile così a molti contesti diversi. L'obiettivo è quello di alterare preliminarmente la distribuzione delle classi così da alleviarne il grado di sbilanciamento. Quale nuova proporzione imporre alle classi è un aspetto variabile che deve essere esplorato in relazione ad ogni specifico problema. I metodi di ricampionamento risultano equivalenti a quelli che impongono diversi costi di errata classificazione o che spostano la soglia di decisione (Breiman *et al.*, 1984b), per i quali, analogamente, non vi è una regola universale per definire i costi o la soglia più opportune.

Le tecniche più semplici che seguono questo approccio si basano sul ricampionamento casuale delle unità del campione di stima. Il sottocampionamento casuale (senza reinserimento) dei casi della classe prevalente potenzialmente esclude informazione rilevante dal campione, perché si basa su una rimozione casuale dei dati, mentre il sovracampionamento casuale (con reinserimento) delle unità della classe rara, oltre ad incrementare i tempi computazionali, porta tipicamente il modello a sovra-adattarsi ai dati, in quanto produce repliche esatte delle unità rare senza di fatto aggiungere nuova informazione sulla classe. Sun *et al.* (2009) spiegano inoltre come il sovracampionamento possa dare luogo ad altri problemi quando le unità della classe rara si collocano in gruppi disgiunti. Infatti, in questi casi, può accadere che in alcuni gruppi le unità vengano molto più ricampionate che in altri, con la conseguenza di enfatizzare alcune aree e oscurarne delle altre, senza alcuna motivazione. Un procedimento più corretto dovrebbe quindi prima individuare i *sotto-concetti* che formano la classe minoritaria e poi sovracampionare da ciascuno fino al raggiungimento del bilanciamento desiderato, con lo svantaggio inevitabile di appesantire il costo dell'analisi (Jo e Japkowicz, 2004).

Per superare i limiti di questi metodi, negli ultimi anni sono state proposte varie soluzioni in cui il ricampionamento delle unità viene guidato da uno schema predefinito. Due tecniche di sottocampionamento che hanno mostrato buoni risultati sono quelle presentate nello studio di Liu *et al.* (2009), chiamate *EasyEnsemble* e *BalanceCascade*. La motivazione alla base di queste proposte è proprio quella di rimediare alla perdita di informazione propria dell'usuale sottocampionamento casuale. L'al-

goritmo *EasyEnsemble* combina i risultati di diversi classificatori ottenuti su insiemi di dati che uniscono ogni volta la classe rara ad un campione selezionato indipendentemente dalla classe prevalente. La seconda procedura, *BalanceCascade*, si appoggia ancora ad un'unione di più classificatori, ma opera una selezione sistematica di quali esempi della classe maggiore rimuovere prima di stimare ogni modello. Altri metodi, come *one-sided selection* (Kubat *et al.*, 1997) e il *link di Tomek* (Tomek, 1976) cercano di individuare un campione rappresentativo della classe prevalente rimuovendo quegli esempi vicini al confine con la classe rara oppure associati a rumore, accentuando la separazione fra i gruppi.

Per quanto riguarda il sovracampionamento, strategie più rigorose prevedono di generare nuove unità statistiche che siano *simili* a quelle rare, ma non esattamente delle loro copie, in modo da allontanare il rischio che il modello si sovra-adatti ai dati. Infatti, quando vengono aggiunte delle repliche esatte delle unità rare, la regione decisionale per quella classe tende ad essere sempre più specifica, riducendo la capacità di generalizzazione del classificatore. Se, invece, la numerosità di quella classe viene arricchita di nuove osservazioni, la regione ad essa associata diventa più ampia e meno specifica, migliorando così la qualità previsiva del modello. Per questo motivo, la tecnica SMOTE (*Synthetic Minority Over-sampling TEchnique*), proposta nel 2002 da Chawla *et al.*, ha guadagnato negli anni sempre più popolarità. La classe minoritaria viene sovracampionata generando nuovi esempi lungo il segmento che collega ogni unità rara ad alcuni dei suoi k vicini più prossimi, scelti casualmente entro la stessa classe. Il numero di vicini da considerare varia in relazione al grado di sovracampionamento richiesto. Per creare un nuovo esempio artificiale, SMOTE calcola la differenza nel vettore delle esplicative fra l'unità rara e uno dei suoi vicini, la moltiplica per un numero casuale appartenente a $[0, 1]$ e infine somma il risultato all'osservazione rara. Questo determina l'aggiunta di una nuova unità lungo la linea che unisce due punti della classe rara, forzando la relativa regione decisionale ad essere più generale. Un limite di questo approccio, dovuto al fatto che per ogni unità rara viene generata una nuova osservazione senza considerarne la posizione, è la possibilità di rimarcare la zona di sovrapposizione fra le classi. L'algoritmo *borderline*-SMOTE (Han *et al.*, 2005) rimedia a questo ostacolo suddividendo le unità della classe rara in tre categorie a seconda della loro posizione prima di applicare la tecnica SMOTE. Un altro contributo significativo in questa categoria, in grado di offrire buoni risultati, è la procedura ROSE (Menardi e Torelli, 2014), che prevede di

generare nuove osservazioni sintetiche a partire da una funzione di densità stimata con il metodo del nucleo condizionatamente alla classe di appartenenza.

Per una rassegna esaustiva sia dei metodi di sottocampionamento che di sovracampionamento disponibili in letteratura si rimanda a He e Garcia (2009), in cui vengono approfonditi i più significativi.

2.3 Metodi che agiscono sull'algoritmo

L'adattamento specifico di un algoritmo al problema di dover riconoscere una classe rara rappresenta una soluzione meno versatile e più sofisticata rispetto alla precedente, che richiede non soltanto una conoscenza approfondita dell'algoritmo, ma anche del problema applicativo che in particolare si intende affrontare.

Un primo approccio consiste nell'introdurre alcune modifiche nel classificatore che compensino per lo sbilanciamento dei dati e rafforzino la sua attenzione verso la classe più rara. Questa strategia è tipicamente applicata a quei modelli che si basano sull'ottimizzazione di una qualche funzione legata all'accuratezza globale. Sostituendola quindi con funzioni alternative che siano indipendenti dalla distribuzione delle classi è possibile migliorare la *performance* del modello. Esempi in questa direzione si trovano nei lavori di Cieslak e Chawla (2008) e Riddle *et al.* (1994) per quanto concerne gli alberi di decisione, oppure in Barandela *et al.* (2003) relativamente al metodo dei k vicini più prossimi.

Altri rimedi modificano l'algoritmo assegnando dei costi di errata classificazione diversi per le unità delle due classi, forzando anche in questo modo il processo a concentrarsi sui casi più rari. È un approccio che viene seguito quando la distribuzione asimmetrica delle classi è associata a diversi costi di errata classificazione e, tipicamente, quello relativo alle unità rare è molto più alto del corrispondente costo per le unità della classe negativa. Un costo non è necessariamente da intendersi in termini monetari e può, ad esempio, indicare anche uno spreco di tempo o la gravità di una malattia. Tuttavia, nella realtà molto spesso i costi non sono disponibili in fase di stima di un modello. Il costo c_i di classificare erroneamente l'i-esima unità dipende dalla sua vera classe di appartenenza ed è definito come:

$$c_i = c(y_i) = \begin{cases} c_{FN} & \text{se } y_i = \mathcal{Y}_1, \\ c_{FP} & \text{se } y_i = \mathcal{Y}_0, \end{cases} \tag{2.1}$$

dove $c_{FN} > 0$ è il costo di un falso negativo e $c_{FP} > 0$ quello associato ad un falso positivo. Coerentemente con quanto spiegato prima, si assume che $c_{FN} > c_{FP}$. La corretta classificazione di un'unità non comporta invece alcun costo, quindi $c_{VP} = c_{VN} = 0$ (Elkan, 2001). Nello specifico, si fa riferimento a questo modo di definire i costi come *asimmetria a livello di classe* (Landesa-Vázquez e Alba-Castro, 2015). L'obiettivo diventa allora quello di ottenere una regola di classificazione che minimizzi il costo atteso complessivo, alterando l'algoritmo in modo che possa inglobare i costi. Alcuni spunti interessanti sono presenti in Breiman *et al.* (1984b) e Kukar *et al.* (1998) con riferimento, rispettivamente, agli alberi di decisione e alle reti neurali.

Si dimostra facilmente che assegnare dei costi alle unità durante la fase di stima di un modello è equivalente a modificare il valore della soglia di classificazione rispetto all'usuale 0.5, spostandola verso la classe meno costosa. Data la probabilità $\mathbb{P}[y = \mathcal{Y}_1|\mathbf{x}]$, l'unità $\mathbf{x}$ viene assegnata alla classe positiva se e soltanto se il costo atteso per la previsione positiva, $c_{VP}\mathbb{P}[y = \mathcal{Y}_1|\mathbf{x}] + c_{FP}\mathbb{P}[y = \mathcal{Y}_0|\mathbf{x}]$, è minore di quello per la previsione negativa, $c_{VN}\mathbb{P}[y = \mathcal{Y}_0|\mathbf{x}] + c_{FN}\mathbb{P}[y = \mathcal{Y}_1|\mathbf{x}]$, che, per come sono stati definiti i costi, si formalizza nel modo seguente:

$$\begin{aligned} &c_{FN}\mathbb{P}[y = \mathcal{Y}_1|\mathbf{x}] > c_{FP}\mathbb{P}[y = \mathcal{Y}_0|\mathbf{x}] \iff \\ &c_{FN}\mathbb{P}[y = \mathcal{Y}_1|\mathbf{x}] > c_{FP}(1 - \mathbb{P}[y = \mathcal{Y}_1|\mathbf{x}]) \iff \\ &\mathbb{P}[y = \mathcal{Y}_1|\mathbf{x}] > \frac{c_{FP}}{c_{FN} + c_{FP}}. \end{aligned} \tag{2.2}$$

Ponendo $c_{FN} > c_{FP}$, allora la soglia $k = c_{FP}/(c_{FN} + c_{FP})$ è minore di 0.5, rendendo più probabile che le unità siano classificate come positive. L'accuratezza previsiva rispetto alla classe rara aumenta, al costo di un incremento anche del tasso di falsi positivi.

Per concludere, un'ulteriore categoria di possibili soluzioni a livello di algoritmo è data dalle combinazioni di classificatori, dette anche tecniche di *ensemble learning*, che seguono la logica del *boosting*, del *bagging* o delle *random forest*. Sebbene questi metodi non siano stati originariamente sviluppati per affrontare problemi di dati sbilanciati, portano anche in questo contesto a miglioramenti spesso significativi della *performance* del singolo classificatore. Essi condividono l'idea di base che l'unione tramite *voto di maggioranza* di più regole, ottenute apportando ogni volta delle modifiche ai dati dell'insieme di stima, porti ad una riduzione nella varianza del singolo modello e renda le previsioni più stabili. Il *boosting*, in particolare,

si è verificato empiricamente essere responsabile anche di una diminuzione nella distorsione del modello (Friedman *et al.*, 2000). Per rendere questi metodi ancora più specifici verso il problema in questione sono stati sviluppati numerosi algoritmi che integrano le diverse procedure con i metodi di ricampionamento visti nella sezione precedente. Per una rassegna generale si vedano, ad esempio, gli studi di Díez-Pastor *et al.* (2015) e Galar *et al.* (2012).

2.4 Metodi basati sul *boosting*

Il *boosting* affonda le sue radici nella teoria dell'apprendimento automatico e ha avuto origine nello studio teorico di Schapire (1990) e di Freund (1995), inserendosi così fra una delle tecniche di *ensemble learning* più potenti. L'intuizione sottostante è che ottenere regole di classificazione mediocri sia molto più semplice e immediato che trovare un'unica, altamente accurata, regola di previsione e che, d'altra parte, la loro combinazione possa contribuire a un sensibile miglioramento dell'accuratezza di ogni singola regola. La strategia consiste quindi nel scegliere un qualsiasi classificatore *debole*, o *di base*, la cui *performance* non sia inferiore ad una classificazione casuale delle unità, applicarlo sequenzialmente a delle varianti del campione di stima per un prefissato numero di iterazioni, e infine combinare con un voto di maggioranza pesato le previsioni ottenute in un'unica regola di classificazione. Si assume, a seguire, che le classi della variabile risposta y siano definite come $\{\mathcal{Y}_0, \mathcal{Y}_1\} = \{-1, +1\}$.

2.4.1 L'algoritmo *AdaBoost*

L'algoritmo *AdaBoost*, forma più concisa per *Adaptive Boosting*, è stato sviluppato nel 1997 da Freund e Schapire per affrontare problemi di classificazione binaria, con lo scopo di convertire un classificatore debole in uno *forte*, ed è stato poi presto esteso a molti altri contesti, come a problemi di classificazione multipla e di regressione, tutti esaustivamente trattati da Schapire e Freund (2012). Si cita inoltre, fra i tanti, il lavoro di Friedman *et al.* (2000), che offre una significativa spiegazione del metodo in chiave statistica.

L'Algoritmo 2.1 presenta lo pseudo-codice dell'algoritmo *AdaBoost*. Dopo aver ricevuto in input i dati dell'insieme di stima $\mathcal{S}_n$ e avendo definito una famiglia di funzioni $h(\mathbf{x}; \gamma)$ per il classificatore debole, il cui argomento γ identifica un partico-

Algoritmo 2.1 *AdaBoost.*

1. Input: insieme di stima $\mathcal{S}_n = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$, con $\mathbf{x} \in \mathcal{X}$, $y \in \{-1, +1\}$; famiglia di funzioni $h(\mathbf{x}; \gamma)$.
2. Inizializzazione: $D_i^{(1)} = 1/n$, per $i = 1, \ldots, n$.
3. Per $t = 1, \ldots, T$:
   (a) Stima il classificatore debole $h(\mathbf{x}; \gamma^{(t)})$ sulle unità con distribuzione $D^{(t)}$.
   (b) Ottieni le previsioni: $\{\hat{y}_1^{(t)}, \ldots, \hat{y}_n^{(t)}\}$.
   (c) Calcola l'errore pesato:
   $$\varepsilon^{(t)} = \mathbb{P}_{i \sim D^{(t)}}[\hat{y}_i^{(t)} \neq y_i].$$
   (d) Assegna: $\alpha^{(t)} \leftarrow \frac{1}{2}\ln\Big(\frac{1-\varepsilon^{(t)}}{\varepsilon^{(t)}}\Big)$.
   (e) Aggiorna la distribuzione:
   $$D_i^{(t+1)} = \frac{D_i^{(t)}}{Z^{(t)}} \times \begin{cases} e^{-\alpha^{(t)}} & \text{se } \hat{y}_i^{(t)} = y_i \\ e^{\alpha^{(t)}} & \text{se } \hat{y}_i^{(t)} \neq y_i \end{cases} = \frac{D_i^{(t)} \exp\{-\alpha^{(t)} y_i \hat{y}_i^{(t)}\}}{Z^{(t)}}, \tag{2.3}$$
   dove $Z^{(t)}$ è una costante di normalizzazione scelta affinché $D^{(t+1)}$ sia una distribuzione.
4. Output: $\hat{y} = \text{sign}(\mathcal{H}(\mathbf{x}))$, dove $\mathcal{H}(\mathbf{x}) = \sum_{t=1}^{T} \alpha^{(t)} \hat{y}^{(t)}$.

lare insieme di parametri non noto, l'algoritmo procede chiamando iterativamente il modello di base T volte. Ad ogni passo, i dati da fornire al classificatore debole vengono prima alterati aggiornando la distribuzione su di essi. Il peso $D_i^{(t)}$ di tale distribuzione per l'unita i al passo t riflette l'importanza di classificare correttamente quell'unità al passo corrente. Si inizializza allora l'algoritmo ponendo uguali tutti i pesi. Con il procedere delle iterazioni, vengono incrementati quelli delle unità mal classificate al passo precedente, forzando il classificatore debole verso quelle unità più ardue da riconoscere. Intuitivamente, questo modo di procedere rende l'algoritmo particolarmente appropriato per gestire il problema dello sbilanciamento, per il fatto

che le unità della classe rara, più difficili da classificare correttamente, riceveranno un peso sempre crescente.

Ottenute le previsioni $\{\hat{y}_1, \ldots, \hat{y}_n\}$, la qualità del modello di base $h(\mathbf{x}; \gamma^{(t)})$ è misurata dal suo errore pesato rispetto alla distribuzione $D^{(t)}$:

$$\varepsilon^{(t)} = \mathbb{P}_{i \sim D^{(t)}}[\hat{y}_i^{(t)} \neq y_i] = \sum_{i : \hat{y}_i^{(t)} \neq y_i} D_i^{(t)}, \tag{2.4}$$

dove $\mathbb{P}_{i \sim D^{(t)}}[\cdot]$ denota la probabilità di selezionare casualmente un esempio, come specificato dall'indice i, rispetto alla distribuzione $D^{(t)}$. Quindi, l'errore pesato $\varepsilon^{(t)}$ è la probabilità che il modello debole stimato al passo t classifichi erroneamente un'unità estratta casualmente secondo la distribuzione $D^{(t)}$, o, equivalentemente, la somma dei pesi sugli esempi mal classificati. L'assunzione che il classificatore di base debba essere debole si traduce più formalmente nella condizione che ogni $\varepsilon^{(t)}$ sia al più pari a $1/2 - \delta^{(t)}$, per $\delta^{(t)}$ arbitrariamente piccolo.

Il passo successivo consiste nel calcolo del parametro $\alpha^{(t)}$, un indicatore proporzionale all'accuratezza del classificatore di base stimato alla t-esima iterazione. Dall'espressione riportata nello pseudo-codice, si nota che $\alpha^{(t)} > 0$ se $\varepsilon^{(t)} < 1/2$ e che $\alpha^{(t)}$ cresce al diminuire di $\varepsilon^{(t)}$, assicurando così che, quanto più è accurato il classificatore di base, tanto maggiore sarà il suo contributo finale. Si dimostra che l'errore di stima commesso da *AdaBoost* è limitato superiormente dalla funzione di perdita esponenziale e $\alpha^{(t)}$ viene scelto in modo da minimizzare questa quantità.

La distribuzione $D^{(t)}$ viene aggiornata secondo la Regola 2.3, in cui i pesi delle osservazioni correttamente classificate vengono moltiplicati per $e^{-\alpha^{(t)}} < 1$ e gli altri per $e^{\alpha^{(t)}} > 1$. Questo aggiornamento può essere espresso in forma più concisa moltiplicando ogni peso $D_i^{(t)}$ per $\exp\{-\alpha^{(t)} y_i \hat{y}_i^{(t)}\}$, sfruttando il fatto che etichette e previsioni assumono valori in $\{-1, +1\}$. I valori risultanti vengono poi normalizzati, così da assicurare che la nuova distribuzione $D^{(t+1)}$ abbia effettivamente somma unitaria. L'effetto di questa operazione è di aumentare i pesi delle unità mal classificate dal modello di base e diminuire quelli delle unità correttamente classificate.

Al termine, *AdaBoost* combina i T classificatori deboli in una regola finale $\mathcal{H}(\mathbf{x})$, tramite un voto di maggioranza pesato in cui le previsioni calcolate ad ogni passo $\hat{y}^{(t)}$ ricevono il corrispondente peso $\alpha^{(t)}$. Le unità vengono assegnate alle due classi $\{-1, +1\}$ in base al segno della combinazione lineare così calcolata.

Algoritmo 2.2 Generalizzazione di *AdaBoost* tramite la procedura di stima *forward stagewise* di un modello additivo.

1. Input: insieme di stima $\mathcal{S}_n = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$, con $\mathbf{x} \in \mathcal{X}$, $y \in \{-1, +1\}$; famiglia di funzioni $h(\mathbf{x}; \gamma)$.
2. Inizializzazione: $\mathcal{H}^{(0)}(\mathbf{x}) = 0$.
3. Per $t = 1, \ldots, T$:
   (a) Risolvi: $(\beta^{(t)}, \gamma^{(t)}) = \arg\min_{(\beta,\gamma)} \sum_{i=1}^{n} \mathrm{L}(y_i, \mathcal{H}^{(t-1)}(\mathbf{x}_i) + \beta h(\mathbf{x}_i; \gamma))$.
   (b) Aggiorna: $\mathcal{H}^{(t)}(\mathbf{x}) = \mathcal{H}^{(t-1)}(\mathbf{x}) + \beta^{(t)} h(\mathbf{x}; \gamma^{(t)})$.
4. Output: $\mathcal{H}(\mathbf{x}) = \sum_{t=1}^{T} \beta^{(t)} h(\mathbf{x}; \gamma^{(t)})$.

Una caratteristica particolare di *AdaBoost* riguarda la sua stretta connessione con i modelli additivi, di cui si dimostra essere un caso particolare (Friedman *et al.*, 2000). Si supponga di voler approssimare una generica funzione $\mathcal{H}(\mathbf{x})$ definita come sommatoria di funzioni di base $h(\mathbf{x}; \gamma)$:

$$\mathcal{H}(\mathbf{x}) = \sum_{t=1}^{T} \beta^{(t)} h(\mathbf{x}; \gamma^{(t)}), \tag{2.5}$$

dove con $\beta^{(t)}$ si indica il t-esimo coefficiente della sommatoria. Questi modelli vengono tipicamente stimati minimizzando la media di una qualche funzione di perdita sulle unità di stima:

$$(\beta^{(t)}, \gamma^{(t)}) = \underset{\{\beta^{(l)}, \gamma^{(l)}\}_1^T}{\arg\min} \sum_{i=1}^{n} \mathrm{L}\Big(y_i, \sum_{l=1}^{T} \beta^{(l)} h(\mathbf{x}_i; \gamma^{(l)})\Big), \quad t \in 1, \ldots, T, \tag{2.6}$$

che approssima il valore atteso. Nella quasi totalità dei casi pratici, tuttavia, le tecniche di ottimizzazione numerica richieste risultano computazionalmente troppo intensive e si preferisce semplificare il problema stimando una singola funzione di base alla volta. La procedura di stima *forward stagewise* prevede quindi di approssimare l'equazione (2.5) tramite minimizzazioni successive, così come descritto in forma di pseudo-codice nell'Algoritmo 2.2. L'approssimazione viene migliorata sequenzialmente, grazie all'aggiunta, ad ogni ciclo dell'algoritmo, di una funzione e il relativo coefficiente, senza aggiustare quelli precedentemente stimati. Si dimostra che se la funzione di perdita è esponenziale, cioè se $\mathrm{L}(y, \mathcal{H}(\mathbf{x})) = \exp\{y\mathcal{H}(\mathbf{x})\}$, allora questa procedura coincide con l'algoritmo *AdaBoost*, avendo posto come funzione di

base un certo classificatore debole che restituisce le etichette previste $\{\hat{y}_1, \ldots, \hat{y}_n\}$, per cui la regola di classificazione finale è data da $\hat{y} = \text{sign}(\mathcal{H}(\mathbf{x}))$. Si può così concludere che l'algoritmo *AdaBoost* minimizza la funzione di perdita esponenziale nella procedura di stima *forward stagewise* di un modello additivo.

2.4.2 Il *gradient boosting*

L'algoritmo *gradient boosting* rappresenta una formulazione del *boosting* molto versatile, presentata da Friedman nel 2001, che unisce l'idea del *boosting* ad una procedura di ottimizzazione numerica per stimare la funzione di perdita che si basa sulla discesa del gradiente. L'estrema flessibilità di questo metodo deriva proprio dalla possibilità di poter scegliere arbitrariamente la funzione di perdita, a seconda della natura del problema.

Friedman sfrutta la connessione fra *AdaBoost* e la sua generalizzazione come modello additivo stimato con la procedura *forward stagewise* come punto di partenza per una formulazione ancora più generale del *boosting*. L'intuizione sottostante è che il passo di minimizzazione 3(a) dell'Algoritmo 2.2 possa essere risolto tramite una procedura di ottimizzazione numerica basata, in particolare, sulla discesa del gradiente. Allora, nella classica procedura di discesa del gradiente, al passo t la funzione dovrebbe essere aggiornata secondo la regola seguente:

$$\mathcal{H}^{(t)}(\mathbf{x}) = \mathcal{H}^{(t-1)}(\mathbf{x}) - \rho \nabla \mathrm{L}(y, \mathcal{H}^{(t-1)}(\mathbf{x})), \tag{2.7}$$

dove $\rho > 0$, il *tasso di apprendimento*, indica l'ampiezza del passo e $\nabla \mathrm{L}(y, \mathcal{H}^{(t-1)}(\mathbf{x}))$ è il gradiente della funzione di perdita al passo corrente. Il gradiente è calcolato nello spazio di dimensione n rispetto ai valori che sono stati stimati al passo precedente $\mathcal{H}^{(t-1)}(\mathbf{x})$ ed è definito come segue:

$$\begin{aligned} \nabla \mathrm{L}(y, \mathcal{H}^{(t-1)}(\mathbf{x})) &= [r_1^{(t)}, \ldots, r_n^{(t)}] \\ &= \left[\frac{\partial \mathrm{L}(y_1, \mathcal{H}^{(t-1)}(\mathbf{x}_1))}{\partial \mathcal{H}^{(t-1)}(\mathbf{x}_1)}, \ldots, \frac{\partial \mathrm{L}(y_n, \mathcal{H}^{(t-1)}(\mathbf{x}_n))}{\partial \mathcal{H}^{(t-1)}(\mathbf{x}_n)} \right], \end{aligned} \tag{2.8}$$

dove $r_i^{(t)}$, $i = 1, \ldots, n$, indica la componente i-esima del vettore, chiamata anche pseudo-residuo o residuo generalizzato. Il segno negativo nell'equazione (2.7) dipende dal fatto di considerare il gradiente negativo della funzione di perdita, che esprime la direzione locale in cui decresce maggiormente. Si noti l'analogia con l'espressione

Algoritmo 2.3 *Gradient boosting.*

1. Input: insieme di stima $\mathcal{S}_n = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$, con $\mathbf{x} \in \mathcal{X}$, $y \in \{-1, +1\}$; famiglia di funzioni $h(\mathbf{x}; \gamma)$.
2. Inizializzazione: $\mathcal{H}^{(0)}(\mathbf{x}) = 0$; $\rho > 0$.
3. Per $t = 1, \ldots, T$:
   (a) Calcola punto per punto il gradiente della funzione di perdita al passo corrente:
   $$r_i^{(t)} = \left[\frac{\partial \mathrm{L}(y_i, \mathcal{H}(\mathbf{x}_i))}{\partial \mathcal{H}(\mathbf{x}_i)}\right]_{\mathcal{H}(\mathbf{x}_i)=\mathcal{H}^{(t-1)}(\mathbf{x}_i)}, \text{ per } i = 1, \ldots, n.$$
   (b) Approssima il gradiente negativo con un classificatore di base $h(\mathbf{x}; \gamma^{(t)})$ tramite il criterio dei minimi quadrati:
   $$\gamma^{(t)} = \arg\min_{\gamma} \sum_{i=1}^{n} [-r_i^{(t)} - h(\mathbf{x}_i; \gamma)]^2.$$
   (c) Aggiorna: $\mathcal{H}^{(t)}(\mathbf{x}) = \mathcal{H}^{(t-1)}(\mathbf{x}) + \rho h(\mathbf{x}; \gamma^{(t)})$.
4. Output: $\mathcal{H}(\mathbf{x}) = \sum_{t=1}^{T} \rho h(\mathbf{x}; \gamma^{(t)})$.

3(b) dell'Algoritmo 2.2, dove il parametro ρ e il gradiente negativo sono sostituiti dal coefficiente della sommatoria e da una funzione di base opportunamente stimati al passo precedente.

Il problema legato all'utilizzo della regola di aggiornamento (2.7) è che non permette di valutare la funzione in punti diversi dagli n valori osservati. Per questa ragione, si cerca un'approssimazione del gradiente negativo che possa essere valutata ovunque. Si stima, più precisamente, il classificatore debole maggiormente correlato con il gradiente negativo della funzione di perdita. Questo ragionamento conduce alla definizione del *gradient boosting*, il cui pseudo-codice è presentato nell'Algoritmo 2.3. Ad ogni iterazione, l'algoritmo determina la direzione, definita dal gradiente negativo, in cui è più opportuno migliorare la stima e seleziona il modello, entro una classe di funzioni prestabilita, più in accordo con quella direzione. Il classificatore

di base $h(\mathbf{x}; \gamma^{(t)})$ al passo t è quindi stimato tramite il criterio dei minimi quadrati:

$$\gamma^{(t)} = \arg\min_{\gamma} \sum_{i=1}^{n} [-r_i^{(t)} - h(\mathbf{x}_i; \gamma)]^2, \tag{2.9}$$

e viene aggiunto al modello previa moltiplicazione per il tasso di apprendimento. La regola finale $\mathcal{H}(\mathbf{x})$, che esprime un punteggio, può essere opportunamente ricondotta alla stima di una probabilità o alle etichette di classe, a seconda di come viene scelta la funzione di perdita. Due comuni alternative per i problemi di classificazione sono la funzione di perdita esponenziale e quella binomiale.

Il parametro ρ determina la velocità con cui l'algoritmo si adatta ai dati e gioca il ruolo di parametro di lisciamento. La sua scelta va condotta in relazione al numero massimo di iterazioni T, con cui si influenza reciprocamente. In generale, un valore di ρ piccolo richiede più iterazioni dell'algoritmo per raggiungere un grado soddisfacente di adattamento ai dati, ma allontana il rischio del sovra-adattamento.

Si cita, per concludere, il metodo *stochastic gradient boosting* (Friedman, 2002), un ulteriore affinamento dell'algoritmo con il duplice vantaggio di migliorare la capacità previsiva del modello riducendone i tempi computazionali di stima. Ad ogni iterazione, non vengono utilizzate tutte le unità dell'insieme di stima originario, ma ne viene estratto casualmente un sotto campione, senza reinserimento.

2.4.3 *Cost-sensitive boosting*

I metodi di *cost-sensitive boosting* mantengono le stesse linee guida dell'algoritmo *AdaBoost*, ma introducono delle modifiche nella regola di aggiornamento dei pesi (2.3) in modo da includere dei costi per le osservazioni del campione di stima. Tipicamente, questo può avvenire in tre modi diversi: inserendo i costi all'interno della funzione esponenziale, all'esterno di essa, oppure adottando entrambe le soluzioni. Seguendo queste indicazioni, sono stati sviluppati tre algoritmi *AdaC1*, *AdaC2* e *AdaC3* (Sun *et al.*, 2007), in cui, come naturale conseguenza ad una nuova regola di aggiornamento, cambia anche il calcolo del parametro $\alpha^{(t)}$, che deve includere i diversi costi c_i associati all'errata classificazione dell'unità i-esima.

Un algoritmo simile a *AdaC1* è *AdaCost* (Fan *et al.*, 1999), che si distingue dal primo per il fatto di sostituire i costi c_i con una funzione $\beta(\cdot)$ di aggiustamento dei costi, che dipende dall'esito della classificazione per l'unità i-esima e dal suo costo c_i. La funzione può essere scelta arbitrariamente purché rispetti il principio di

aumentare in modo più aggressivo i pesi per un'osservazione che ha costo maggiore quando viene mal classificata, e diminuirli in modo più conservativo altrimenti, rispetto ad un'unità che ha costo inferiore. Il parametro $\alpha^{(t)}$ viene calcolato di conseguenza.

Altri due algoritmi, *CSB1* e *CSB2* (Ting, 2000), modificano la regola di aggiornamento dei pesi in modo analogo a *AdaC2*, quindi inserendo un nuovo termine per i costi fuori dall'esponenziale, ma calcolando i valori $\alpha^{(t)}$ in modo alternativo.

Per concludere la rassegna dei principali metodi di *cost-sensitive boosting* si cita l'algoritmo *RareBoost* (Joshi *et al.*, 2001), che distingue il calcolo di $\alpha^{(t)}$ per le previsioni positive e per quelle negative così da scalare i pesi per i falsi positivi in modo proporzionale a quanto bene si distinguono dai veri positivi, e i pesi per i falsi negativi in relazione ai veri negativi. A differenza dei metodi precedenti, *RareBoost* non richiede che siano specificati preliminarmente dei costi, ricavandoli in modo automatico ad ogni passo. Tuttavia, questa soluzione soffre di un limite tanto vincolante da sconsigliarne l'utilizzo proprio quando i dati presentano una classe rara, nonostante sia questo il contesto per cui l'algoritmo è stato ideato. Il motivo dipende da una condizione molto forte che deve essere soddisfatta affinché sia calcolabile $\alpha^{(t)}$ per le unità classificate come positive, ossia che, fin dalla prima iterazione, il classificatore debole trovi almeno un falso positivo, altrimenti l'algoritmo collassa.

I metodi presentati propongono delle strategie alternative per pesare le unità dell'insieme di stima ad ogni nuovo passo di *AdaBoost*, in modo da distinguere le osservazioni non soltanto in base al risultato del classificatore debole, ma anche in relazione alla loro classe di appartenenza. In questo senso, tutti gli algoritmi di *cost-sensitive boosting* aumentano i pesi dei falsi negativi più dei falsi positivi, ricordando che il costo per un'unità positiva è sempre più alto che per un'unità negativa. Le regole di aggiornamento dei pesi differiscono invece per quanto riguarda le osservazioni che vengono correttamente classificate ad ogni passo, i veri positivi e i veri negativi. Gli algoritmi *AdaC2* e *AdaCost* vengono tipicamente preferiti perché preservano più peso sui veri positivi, mentre *AdaC1*, *CSB1* e *CSB2* mostrano un comportamento opposto e per *AdaC3* non è certo cosa accada essendo una combinazione di *AdaC1* e *AdaC2*. Inoltre, uno studio empirico condotto da Sun *et al.* (2007) rivela la superiorità di *AdaC2* rispetto ai suoi rivali, essendo in grado di accumulare un peso sempre superiore sulle unità della classe rara.

2.4.4 Metodi di *boosting* per classi sbilanciate

In vari lavori è stato puntualizzato come la strategia di derivazione dei pesi propria dell'algoritmo *AdaBoost* sia equivalente ad un ricampionamento delle unità statistiche che combina, ad ogni passo, un sovracampionamento di quelle mal classificate al passo precedente e un sottocampionamento di quelle correttamente classificate. Da questo punto di vista, *AdaBoost* potrebbe rientrare in una delle soluzioni che operano a livello di dati, con il vantaggio aggiuntivo di ricampionare le unità automaticamente senza il costo di dover esplorare la distribuzione ottimale delle classi o ricercare le osservazioni più rappresentative da conservare o escludere dal campione. Inserendosi in questa logica, alcuni metodi introducono delle particolari tecniche di ricampionamento all'interno dell'algoritmo, come quelle presentate nel paragrafo 2.2, in modo da enfatizzare maggiormente la presenza delle unità positive. L'effetto implicito di queste strategie, specificatamente sviluppate per far fronte al problema dello sbilanciamento, è quindi quello di aumentare il peso della classe rara ad ogni iterazione dell'algoritmo.

L'algoritmo *SMOTEBoost* (Chawla *et al.*, 2003) combina la tecnica SMOTE con il classico algoritmo di *boosting*, introducendola ad ogni passo per indurre il classificatore di base ad individuare regioni decisionali più ampie per la classe rara. In particolare, l'algoritmo proposto si basa su una versione del *boosting* chiamata *AdaBoost.M2* (Freund *et al.*, 1996), che si riduce a *AdaBoost* in problemi di classificazione binaria, ma permette anche di gestire insiemi di dati con più classi. All'iterazione t dell'algoritmo, quindi, vengono generati nuovi esempi sintetici dalla classe rara con la tecnica SMOTE e la distribuzione $D^{(t)}$ è aggiornata di conseguenza. Le nuove unità non vengono aggiunte al campione di stima, ma sono rimosse prima dell'iterazione successiva. Gli autori osservano come l'introduzione della procedura SMOTE nel *boosting* accentui la diversità fra i classificatori di base, dato che ad ogni passo vengono introdotti nuovi e diversi esempi sintetici. Così come è stata ideata la tecnica *borderline*-SMOTE per perfezionare SMOTE, nel 2009, Hu *et al.* hanno proposto il metodo *MSMOTEBoost*, che tiene conto della diversa posizione delle unità della classe rara prima di generarne di nuove, appesantendo però il costo computazionale.

In modo analogo, ma adottando una tecnica di ricampionamento inversa, l'algoritmo *RUSBoost* (Seiffert *et al.*, 2008) altera la distribuzione dei dati con un

sottocampionamento casuale delle unità della classe prevalente ad ogni passo dell'algoritmo di *boosting*. Il principale limite del sottocampionamento casuale, che risiede nella perdita di informazione, viene superato proprio grazie all'integrazione con il *boosting*. Infatti, se alcuni dati rilevanti possono essere assenti in certe iterazioni, è probabile che siano inclusi successivamente. Oltre ad essere una soluzione computazionalmente meno intensiva rispetto a *SMOTEBoost*, un confronto empirico fra le due proposte condotto dagli stessi autori suggerisce come i risultati siano comparabili, se non migliori con *RUSBoost*. A partire da questo algoritmo, è stato recentemente proposto *EUSBoost* (Galar *et al.*, 2013), che cerca di accentuare la diversità fra i classificatori di base usando per ciascuno un diverso sotto-insieme delle unità della classe maggiore.

Capitolo 3

Delle proposte basate sull'AUC

3.1 Introduzione

La curva ROC e l'AUC sono tra le metriche di valutazione più diffuse per misurare la qualità di un modello di classificazione, soprattutto in presenza di una classe rara, essendo indipendenti dalla distribuzione sottostante delle classi e non vincolate ad un unico valore per la soglia di classificazione. I metodi proposti in questo capitolo nascono dalla volontà di adattare modelli preesistenti e basati sull'ottimizzazione di un criterio di accuratezza globale al problema più specifico della classificazione in presenza di una classe rara tramite l'introduzione dell'AUC nella loro fase di stima.

Il primo metodo che sarà presentato prevede delle modifiche al classico albero di classificazione, in modo da guidarne la crescita ottimizzando localmente l'AUC ad ogni nodo. Il primo studio in questa direzione è stato condotto da Ferri *et al.* (2002), con l'introduzione di un criterio per la selezione della variabile e del punto di *split* basato sull'AUC. Un altro valido contributo è anche quello di Hossain *et al.* (2008), per aver aggiunto un criterio di potatura che fosse anch'esso guidato dall'AUC.

Per quanto concerne il secondo metodo, invece, sarà derivata una funzione di perdita basata sull'AUC da inserire nell'algoritmo di *gradient boosting*, così da rafforzarne l'accuratezza previsiva verso la classe minoritaria. La funzione di perdita sarà appositamente adattata e modificata per garantirne la differenziabilità e il suo impiego all'interno dell'algoritmo. Relativamente al *gradient boosting*, sarà presa in considerazione una sua particolare formulazione, nota come *XGBoost* (*Extreme Gradient Boosting*, Chen e Guestrin, 2016), che introduce delle lievi modifiche alla proposta originaria di Friedman (2001) ed è computazionalmente molto efficiente.

Il codice relativo ad entrambe le procedure è stato sviluppato nell'ambiente di programmazione R (R Core Team, 2014) ed è presentato in Appendice.

3.2 Una variante dell'albero di classificazione

Gli alberi rappresentano uno degli strumenti di classificazione più frequentemente utilizzati, in grado di ridurre efficacemente problemi complessi e di diversa natura in un numero limitato di decisioni più semplici, fornendo soluzioni di immediata interpretazione. Un albero procede tramite partizionamenti successivi dello spazio $\mathcal{X}$ in cui giacciono le variabili esplicative. Le componenti di un albero binario sono disuguaglianze, del tipo $\mathbf{x}_j < t$, ed è chiamata *radice* la prima. Si tratta di un metodo ricorsivo che sceglie, ad ogni passo, la variabile e il punto di suddivisione, o di *split*, che ottimizzano un qualche criterio, dividendo così lo spazio con un iper-piano parallelo all'asse della variabile selezionata. Si chiamano *foglie*, o *nodi terminali*, le regioni finali in cui viene suddiviso lo spazio. Tipicamente, nella fase di costruzione vengono minimizzate misure di impurità come l'entropia o l'indice di Gini, in modo da avere la massima riduzione di eterogeneità fra le classi ad ogni nodo, preferendo usare il tasso di errata classificazione nella fase di potatura. Nei casi in cui le classi si caratterizzano per un grado di sbilanciamento molto elevato, tuttavia, la classe più numerosa gioca un ruolo dominante nel calcolo di queste metriche e l'albero può aver bisogno di crescere molto in profondità per riconoscere la classe rara. Talvolta la crescita viene fermata anche prima che essa sia riconosciuta, mentre in altri casi può essere la fase di potatura a rimuovere proprio le foglie che la identificano. L'idea è quindi quella di sostituire alle metriche usuali una misura, l'AUC precisamente, più sensibile alla presenza di una classe rara, in modo da non trascurarla.

L'aspetto principale da definire nella fase di costruzione di un albero binario riguarda il criterio con cui i dati devono essere suddivisi in due sottoinsiemi ad ogni nodo. La selezione della variabile e poi quella del relativo punto di *split* vengono svolte in due passi separati. L'utilizzo della curva ROC come strumento per la selezione della variabile esplicativa più discriminante rispetto alla variabile risposta è stato proposto e studiato nel lavoro di Mamitsuka (2006). Seguendo questa strada, quindi, ad ogni *split* dell'albero viene scelta la variabile x_j^* che massimizza l'AUC(x_j, y), per $j = 1, \ldots, d$. Si considera, in altri termini, che i valori assunti da

ogni variabile esprimano un grado di confidenza per la specifica unità di appartenere ad una certa classe di y e che l'AUC ne rifletta in questo senso il potere discriminatorio. Scelta la variabile, viene calcolata una griglia di $(n-1)$ potenziali valori per lo *split*, pari ai punti intermedi fra due valori successivi della variabile ordinata $x_j^{*,ord}$, l'insieme $\{(x_{i,j}^{*,ord} + x_{i+1,j}^{*,ord})/2\}_{i=1}^{n-1}$. Ogni volta che i dati vengono divisi in due gruppi, quindi per ogni valore della griglia, ci sono due possibili modi di assegnare ad essi una classe tra $\{\mathcal{Y}_0, \mathcal{Y}_1\}$, associati a due punti distinti nello spazio della curva ROC simmetrici rispetto alla bisettrice e a due valori dell'AUC complementari. È possibile calcolare l'AUC anche per un solo punto come l'area sottostante la curva spezzata $(0,0)$-(TFP,TVP)-$(1,1)$, pari a (TVP-TFP+1)/2. Chiaramente, una sola delle due opzioni di classificare i gruppi è ragionevole, quella per cui il valore così calcolato dell'AUC è più grande di 0.5. Dopo aver isolato questi valori, si procede allora alla selezione del punto della griglia associato al più grande di essi come valore ottimale per lo *split*.

I passi descritti vengono ripetuti in modo ricorsivo fino a che tutte le foglie non sono pure o viene incontrata una qualche regola di arresto. Prima di descrivere alcuni criteri per la potatura dell'albero, va notato che dei vincoli sulla sua crescita sono già imposti per costruzione. Si stabilisce, infatti, che le variabili da confrontare ad ogni nodo prima di scegliere il punto di *split* debbano avere un valore dell'AUC rispetto ad y maggiore di 0.5. Valori inferiori indicano che il potere discriminatorio di una certa esplicativa rispetto alla variabile risposta è minore di quello di un classificatore casuale e non c'è motivo di considerarla. Come suggeriscono Hossain *et al.* (2008), un possibile criterio per la potatura basato sull'AUC consiste nel fissare un limite superiore, in aggiunta a quello inferiore di 0.5, oltre cui non considerare una variabile per lo *split*. Questo impedisce all'albero di crescere per quei sottoinsiemi di dati in cui tutte le variabili hanno un valore dell'AUC con y molto grande, in modo da arginare il rischio del sovra-adattamento ai dati. In generale, per impedire che l'albero si sovra-adatti ai dati possono essere valutate le usuali alternative. È quindi possibile fissare un limite sul numero minimo di osservazioni ad ogni nodo prima di procedere con la ricerca del punto di *split*, oppure sul numero massimo di osservazioni in ogni nodo terminale, o ancora sulla profondità dell'albero. Altrimenti, si possono considerare i metodi della convalida incrociata o di divisione dei dati in insieme di stima e verifica per individuare la dimensionale ottimale dell'albero.

Esempio

Per rendere più chiaro il metodo proposto, si illustra il procedimento con cui cresce l'albero attraverso un esempio molto semplice in due dimensioni, (x_1, x_2). Viene generato un campione di n=30 unità, 25 appartenenti alla classe negativa provenienti da una normale standard bivariata e 5 alla classe positiva generate da una normale centrata in (1,1) e con matrice di varianza e covarianza diagonale con elementi entrambi pari a 0.5. I dati sono riportati nella Figura 3.1.

Dal confronto fra AUC(x_1,y)=0.864 e AUC(x_2,y)=0.832 si seleziona la variabile x_1 come radice dell'albero e si calcola una griglia di 29 possibili punti per lo *split* dopo averla ordinata. Ciascuno determina una partizione dicotomica dei dati e ogni volta le due foglie, quella di sinistra e di destra, possono essere classificate in due possibili modi, essendo $y \in \{-1, +1\}$.

Per capire come avviene la scelta del punto di *split*, si osservano per semplicità solo quattro valori significativi della griglia, come mostrato nel grafico (b) della Figura 3.1. Per $x_1 = 0.49$, ad esempio, i dati a sinistra possono essere classificati con l'etichetta $\{-1\}$ e quelli a destra con $\{+1\}$, oppure il contrario. Ad ogni situazione è associata una matrice di confusione, su cui calcolare il tasso di falsi positivi e quello di veri positivi. Le due possibili matrici di confusione determinano punti simmetrici rispetto alla bisettrice nello spazio della curva ROC, l'uno essendo la negazione dell'altro. Nel grafico (c) della Figura 3.1 sono rappresentate, a colori, le curve ROC relative ai quattro valori della griglia che generano punti superiori alla bisettrice. Quelli che giacciono al di sotto, in grigio, vengono scartati prima di procedere con il confronto. Questa rappresentazione, completa, include 29 curve ROC e la scelta del punto di *split* ricade sul valore della griglia che massimizza l'AUC tra le varie curve. Nell'esempio, l'AUC maggiore si ha per la curva ROC azzurra ed è pari a 0.82, determinando come punto di *split* $x_1 = 1.08$. Definiti la radice dell'albero e il suo punto di *split*, il procedimento continua ricorsivamente per ogni sottoinsieme dei dati, massimizzando localmente l'AUC ad ogni nuovo nodo.

Le foglie dell'albero si caratterizzano per essere pure oppure perché nessuna variabile, su quei dati, ha un valore di AUC con y maggiore di 0.5. Il modello finale e la partizione che esso induce sui dati sono illustrati nella Figura 3.2.

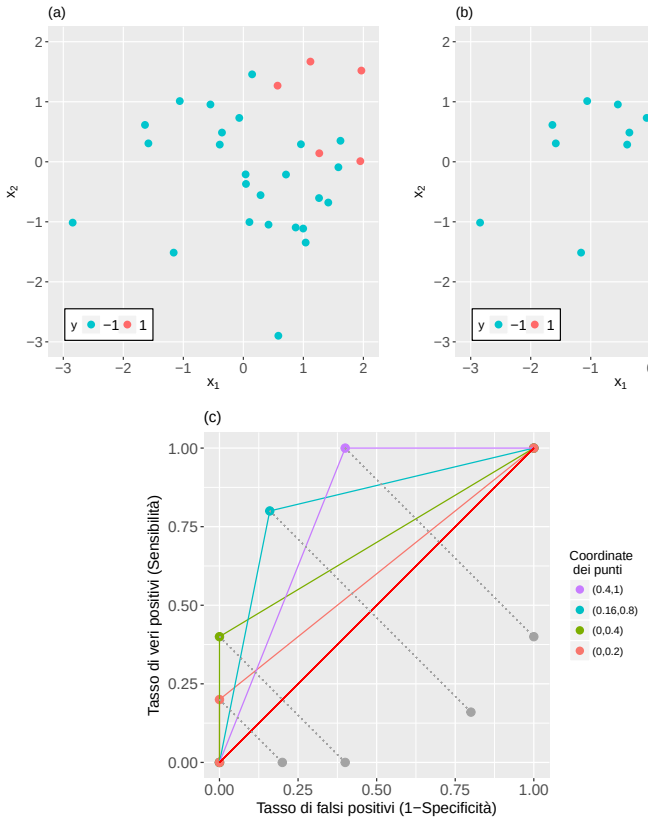

Figura 3.1: (a) Grafico di dispersione per i dati dell'esempio. (b) Partizioni dicotomiche dei dati per quattro valori della griglia: $x_1 = t$, per $t \in \{0.49, 1.08, 1.78, 1.96\}$. (c) Curve ROC relative ai valori della griglia considerati, ottenute per i punti superiori alla bisettrice; in grigio sono segnati i punti simmetrici, inferiori alla diagonale.

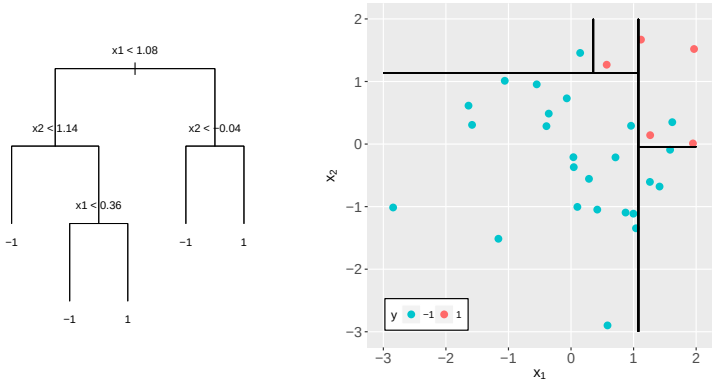

Figura 3.2: Albero completo e relativo partizionamento dei dati.

3.3 Il *gradient boosting* con funzione di perdita basata sull'AUC

Il *gradient boosting* rappresenta un algoritmo estremamente versatile ed efficace anche per affrontare problemi di classificazione con forte sbilanciamento dei dati. Nel metodo che si propone in questa sezione se ne considera la recente versione sviluppata da Chen e Guestrin (2016), chiamata *XGBoost*, e si definisce una particolare funzione di perdita basata sull'AUC da inserire al suo interno. *XGBoost* è un'accurata implementazione del classico algoritmo di *gradient boosting*, efficiente da un punto di vista computazionale e disponibile per molti ambienti di programmazione diversi.

La formulazione del *gradient boosting* proposta da Chen e Guestrin si discosta dall'originario algoritmo di Friedman (2001) e si rivolge specificatamente a due possibili scelte per la famiglia di modelli di base: l'albero di classificazione e regressione (CART, Breiman *et al.*, 1984a) e il modello lineare, con un interesse predominante verso il primo. Il primo accorgimento prevede l'introduzione di una penalizzazione per la complessità del modello di base ad ogni iterazione dell'algoritmo, con l'obiettivo di ridurre il rischio del sovra-adattamento ai dati. Come conseguenza, la funzione di perdita viene inglobata entro una funzione obiettivo che include anche un termine di penalizzazione da specificare in relazione alla famiglia di modelli scelta. La seconda variazione riguarda, invece, la minimizzazione della funzione obiettivo. Nel classico algoritmo di *gradient boosting* la funzione di perdita viene minimizzata tramite approssimazioni consecutive del suo gradiente negativo con un qualche modello di base. Diversamente, in *XGBoost* ad ogni iterazione viene prima approssimata la funzione obiettivo e successivamente stimato il modello che la minimizza. L'approssimazione è ottenuta con un'espansione in serie di Taylor della funzione di perdita arrestata al secondo ordine. In questo modo, per adattare l'algoritmo a diverse funzioni di perdita, risulta sufficiente specificarne la derivata prima e la derivata seconda. Gli autori sostituiscono poi la generica espressione del modello di base con quella di un CART per derivare analiticamente la struttura dell'albero che minimizza la funzione obiettivo approssimata ad ogni passo.

Il metodo proposto ripercorre i passi di *XGBoost*, scegliendo come modello di base uno *stump*, ossia un particolare caso dei CART con un solo *split* e due foglie.

Si tratta di una scelta frequentemente adottata negli algoritmi di *boosting*, essendo un modello semplice e computazionalmente efficiente. Inoltre, definendo a priori questa struttura per il modello di base, non è necessario aggiungere un termine di regolarizzazione alla funzione di perdita. Siano R_1 e R_2 le regioni disgiunte di $\mathcal{X}$ rappresentate dalle due foglie dello *stump* e w_1 e w_2 i relativi punteggi. Uno *stump* è definito allora formalmente nel seguente modo:

$$h(\mathbf{x};\gamma) = \sum_{j=1,2} w_j \mathbb{I}(\mathbf{x} \in \mathrm{R}_j), \tag{3.1}$$

con parametri $\gamma = \{w_1, w_2, \mathrm{R}_1, \mathrm{R}_2\}$.

Nella procedura di ottimizzazione della funzione di perdita, al generico passo t l'obiettivo è stimare il modello $h(\mathbf{x};\gamma^{(t)})$ in modo tale che:

$$\gamma^{(t)} = \arg\min_{\gamma} \sum_{i=1}^{n} \mathrm{L}(y_i, \mathcal{H}^{(t-1)}(\mathbf{x}_i) + h(\mathbf{x}_i;\gamma)). \tag{3.2}$$

Diversamente, l'approccio di Chen e Guestrin (2016) prevede di valutarne un'espansione in serie di Taylor arrestata al secondo ordine come approssimazione. Richiamando quindi:

$$f(x+\Delta x) \simeq f(x) + f'(x)\Delta x + \frac{1}{2} f''(x)\Delta x^2, \tag{3.3}$$

si riformula il problema di minimo come:

$$\gamma^{(t)} = \arg\min_{\gamma} \sum_{i=1}^{n} \left[\mathrm{L}(y_i, \mathcal{H}^{(t-1)}(\mathbf{x}_i)) + r_i^{(t)} h(\mathbf{x}_i;\gamma) + \frac{1}{2} e_i^{(t)} h^2(\mathbf{x}_i;\gamma) \right], \tag{3.4}$$

dove $r_i^{(t)}$, $i = 1, \ldots, n$, è l'elemento i-esimo del vettore delle derivate prime (2.8), mentre $e_i^{(t)}$, $i = 1, \ldots, n$, è l'elemento i-esimo del vettore delle derivate seconde, definito come segue:

$$e_i^{(t)} = \frac{\partial^2 \mathrm{L}(y_i, \mathcal{H}^{(t-1)}(\mathbf{x}_i))}{\partial \mathcal{H}^{(t-1)}(\mathbf{x}_i)}. \tag{3.5}$$

Rimuovendo il primo termine in (3.4), da cui non dipende il problema di minimo, e sostituendo al modello di base l'espressione dello *stump*, il problema diventa, più specificatamente:

$$(w_j^{(t)}, R_j^{(t)}) = \arg\min_{(w_j, \mathrm{R}_j)} \sum_{i=1}^{n} \left[r_i^{(t)} \sum_{j=1,2} w_j \mathbb{I}(\mathbf{x}_i \in \mathrm{R}_j) + \frac{1}{2} e_i^{(t)} \sum_{j=1,2} w_j^2 \mathbb{I}(\mathbf{x}_i \in \mathrm{R}_j) \right], \tag{3.6}$$

per $j = \{1, 2\}$. A questo punto, si esprime il problema in funzione della struttura dell'albero, cioè scrivendo la funzione di perdita in modo da raggruppare le varie quantità ad ogni foglia, come segue:

$$\begin{aligned}(w_j^{(t)}, R_j^{(t)}) &= \arg\min_{(w_j, \mathrm{R}_j)} \sum_{j=1,2} \left(w_j \sum_{i:\mathbf{x}_i \in \mathrm{R}_j} r_i^{(t)} + \frac{1}{2} w_j^2 \sum_{i:\mathbf{x}_i \in \mathrm{R}_j} e_i^{(t)} \right) \\ &= \arg\min_{(w_j, \mathrm{R}_j)} \sum_{j=1,2} \left(w_j \mathcal{R}_j^{(t)} + \frac{1}{2} w_j^2 \mathcal{E}_j^{(t)} \right),\end{aligned} \tag{3.7}$$

per $j = \{1, 2\}$, dove $\mathcal{R}_j^{(t)} = \sum_{i:\mathbf{x}_i \in \mathrm{R}_j} r_i^{(t)}$ e $\mathcal{E}_j^{(t)} = \sum_{i:\mathbf{x}_i \in \mathrm{R}_j} e_i^{(t)}$.

La stima dei parametri che definiscono l'albero avviene in due fasi successive. Assumendone fissata la struttura, ossia la bipartizione dello spazio in due regioni R_1 e R_2, i valori finali delle foglie w_1 e w_2 si ricavano risolvendo il problema di minimo come definito in (3.7). Trattandosi della somma di due funzioni quadratiche indipendenti, si ottengono le due soluzioni distinte:

$$w_1^{(t)} = -\frac{\mathcal{R}_1^{(t)}}{\mathcal{E}_1^{(t)}}, \qquad w_2^{(t)} = -\frac{\mathcal{R}_2^{(t)}}{\mathcal{E}_2^{(t)}}. \tag{3.8}$$

La funzione di perdita al passo t è quindi minima in:

$$\mathrm{L}_{\min} = -\frac{1}{2} \sum_{j=1,2} \frac{(\mathcal{R}_j^{(t)})^2}{\mathcal{E}_j^{(t)}}, \tag{3.9}$$

avendo sostituito al suo interno le espressioni delle quantità $w_1^{(t)}$ e $w_2^{(t)}$.

Si determina la struttura ottimale dello *stump* confrontando i possibili punti di *split* di tutte le variabili e calcolando, per ciascuno, i punteggi ottimali per le foglie, $w_1^{(t)}$ e $w_2^{(t)}$ (3.8). Le regioni ottimali $\mathrm{R}_1^{(t)}$ e $\mathrm{R}_2^{(t)}$ derivano quindi dal punto di *split* che raggiunge il valore più piccolo della funzione di perdita (3.9).

Un'ultima precisazione riguarda il tasso di apprendimento ρ, omesso dalla trattazione soltanto per semplicità espositiva, essendo una quantità fissata non da stimare. Infatti, come nell'usuale *gradient boosting*, il suo ruolo è quello di regolare la velocità dell'algoritmo e il grado di adattamento ai dati e viene quindi moltiplicato per ogni *stump* prima che esso sia aggiunto al modello.

La modifica dell'algoritmo proposta in questo lavoro prevede di ricavare una funzione di perdita basata sull'AUC, in modo da poter inserire le sue derivate, prima e seconda, all'interno del metodo presentato e completare così la procedura.

Per ottimizzare direttamente l'AUC una possibilità è massimizzare la statistica di Wilcoxon-Mann-Whitney (1.4), presentata al termine del Capitolo 1. Questa quantità, tuttavia, oltre a non essere differenziabile, fa riferimento a coppie di osservazioni, mentre la funzione di perdita espressa nella (3.2) è relativa alle singole unità statistiche. Per renderla trattabile devono quindi essere risolti entrambi i problemi. Relativamente al primo punto, Yan *et al.* (2003) propongono due quantità per approssimare direttamente la statistica (1.4) e una ulteriore per il suo complementare, tutte differenziabili. Massimizzare l'AUC o minimizzare il suo complementare è equivalente, ma si sceglie la seconda via per coerenza con la precedente trattazione, dato che la funzione di perdita L nell'espressione (3.2) è convessa. Si introduce allora la seguente funzione differenziabile:

$$\mathrm{S}(\mathcal{H}(\mathbf{x}_i^+), \mathcal{H}(\mathbf{x}_j^-)) = \begin{cases} (-(\mathcal{H}(\mathbf{x}_i^+) - \mathcal{H}(\mathbf{x}_j^-) - \tau))^p & \text{se } \mathcal{H}(\mathbf{x}_i^+) - \mathcal{H}(\mathbf{x}_j^-) < \tau, \\ 0 & \text{altrimenti,} \end{cases} \tag{3.10}$$

dove $\tau \in (0, 1]$ e $p > 1$, da inserire nella duplice sommatoria:

$$\mathrm{U_S} = \frac{1}{n_+ n_-} \sum_{i=1}^{n_+} \sum_{j=1}^{n_-} \mathrm{S}(\mathcal{H}(\mathbf{x}_i^+), \mathcal{H}(\mathbf{x}_j^-)). \tag{3.11}$$

La quantità $\mathrm{U_S}$ è, quindi, un'approssimazione del complemento a uno della (1.4) e deve pertanto essere minimizzata. Quando un'unità positiva riceve un punteggio maggiore di una negativa di una quantità τ, allora quella coppia di unità non contribuisce alla funzione di perdita. L'influenza delle osservazioni è aggiustata in modo adattivo durante il processo di stima sulla base di confronti a coppie. Per quanto riguarda la scelta di τ e p, gli autori suggeriscono per τ un valore compreso fra 0.1 e 0.7, mostrando come i risultati non siano in realtà particolarmente sensibili a questo parametro, mentre per p l'indicazione fornita ricade su $p = 2$ o $p = 3$.

Per esprimere $\mathrm{U_S}$ in funzione di ogni singola unità, invece, si segue quanto Zhao *et al.* (2011) propongono per trattare un problema analogo, tramite l'introduzione di funzioni indicatrici. La quantità $\mathrm{U_S}$ può essere quindi riformulata come segue:

$$\mathrm{U_S} = \frac{1}{n_+ n_-} \sum_{i=1}^{n} \left[\mathbb{I}_{(y_i=1)} \sum_{i'=1}^{i-1} \mathrm{S}_{i'}^+ + \mathbb{I}_{(y_i=-1)} \sum_{i'=1}^{i-1} \mathrm{S}_{i'}^- \right], \tag{3.12}$$

dove:

$$\mathrm{S}_{i'}^+ = \mathbb{I}_{(y_{i'}=-1)} \mathrm{S}(\mathcal{H}(\mathbf{x}_i), \mathcal{H}(\mathbf{x}_{i'})), \quad \mathrm{S}_{i'}^- = \mathbb{I}_{(y_{i'}=1)} \mathrm{S}(\mathcal{H}(\mathbf{x}_{i'}), \mathcal{H}(\mathbf{x}_i)). \tag{3.13}$$

Una volta definiti τ e, soprattutto, l'esponente p, è immediato ricavare le derivate prima e seconda da inserire nell'algoritmo di *gradient boosting* presentato in precedenza.

Capitolo 4
Analisi empirica

4.1 Introduzione

In questo capitolo vengono analizzati e confrontati empiricamente alcuni dei principali metodi basati sul *boosting* e le due proposte descritte nel Capitolo 3, per valutarne la capacità discriminante in contesti caratterizzati dalla presenza di una classe rara. In particolare, l'analisi empirica si suddivide in due parti. La prima è dedicata ad uno studio di simulazione, in cui i vari metodi sono confrontati rispetto a scenari di cui è possibile controllare e modificare la struttura. Nella seconda parte, invece, l'analisi si restringe alle sole nuove proposte e ai modelli direttamente confrontabili con esse per verificarne l'adeguatezza in contesti reali.

4.2 Studio di simulazione

La finalità principale di questo studio è verificare in che misura diversi modelli di classificazione risultino compromessi dal problema dello sbilanciamento. L'idea è quella di predisporre strutture di dati con un diverso grado di complessità per scoprire se esistono, ed eventualmente quali sono, quelle combinazioni di elementi più problematiche per i modelli coinvolti. Nonostante ogni contesto reale sia un caso a sé, da questo studio possono emergere delle indicazioni generali utili a capire quali metodi preferire o scartare rispetto a possibili casi pratici, avendone osservato il comportamento globale negli scenari costruiti. È inoltre di particolare interesse esplorare la qualità previsiva dei nuovi modelli proposti, tramite, soprattutto, il confronto con la loro versione originaria.

4.2.1 Definizione degli scenari

Lo studio prende in considerazione quei fattori, descritti nel primo capitolo, che è noto possano influenzare la *performance* di un modello di classificazione quando è presente un problema di classi fortemente sbilanciate. Sebbene il grado di sbilanciamento rappresenti il fattore più caratterizzante di questi scenari, di per sé esso non rende un problema necessariamente complicato, basti pensare al caso in cui le due classi sono distanti nello spazio e quindi perfettamente riconoscibili anche da una semplice regola di classificazione.

Per esplorare scenari di complessità diversa si fanno variare il grado di sbilanciamento, la numerosità campionaria e la dimensione del problema per tre diverse strutture dei dati, in cui è sempre presente un certo grado di sovrapposizione fra le due classi. Il livello di sbilanciamento è espresso in termini di percentuale di unità della classe rara sulla numerosità complessiva ed è quindi un valore fra 0 e 100. I livelli considerati sono i seguenti: $\{0.6, 1, 5\}$. Per la numerosità campionaria si valutano i casi $n \in \{1000, 10000\}$, mentre per la dimensione d del campione, ossia il numero di variabili esplicative, si considera l'insieme $\{2, 5, 10\}$. Per quanto riguarda la struttura dei dati, si propongono tre possibili scenari, ognuno con caratteristiche diverse e di cui si riporta un esempio in due dimensioni nella Figura 4.1. Si presentano, a seguire, le descrizioni dettagliate di ogni configurazione dei dati.

- *Gruppi disgiunti.* Questa particolare rappresentazione delle due classi segue la proposta di Napierała *et al.* (2010), rifacendosi al problema che gli autori chiamano *subclus*. Lo spazio in cui giacciono i dati è un quadrato di lato $[0, 1]$ in $\mathbb{R}^2$, e diventa un ipercubo al crescere della dimensione. Le unità vengono estratte da una distribuzione uniforme rispetto ad alcune aree specifiche, in modo tale che gli esempi della classe minoritaria siano equamente collocati entro cinque rettangoli (segnati in nero nella figura) e che le unità negative si distribuiscano nello spazio contingente. Inoltre, negli estremi di ogni rettangolo, le unità delle due classi si sovrappongono, con una predominanza di unità negative che varia in base al grado di sbilanciamento e alla numerosità campionaria. In aggiunta, se la sproporzione fra le classi è modesta, nella regione centrale della figura, dove si trovano i rettangoli, gli esempi della classe negativa fra due rettangoli successivi possono essere sotto-rappresentati rispetto alle unità positive, dando origine, così, a gruppi disgiunti per entrambe le classi.

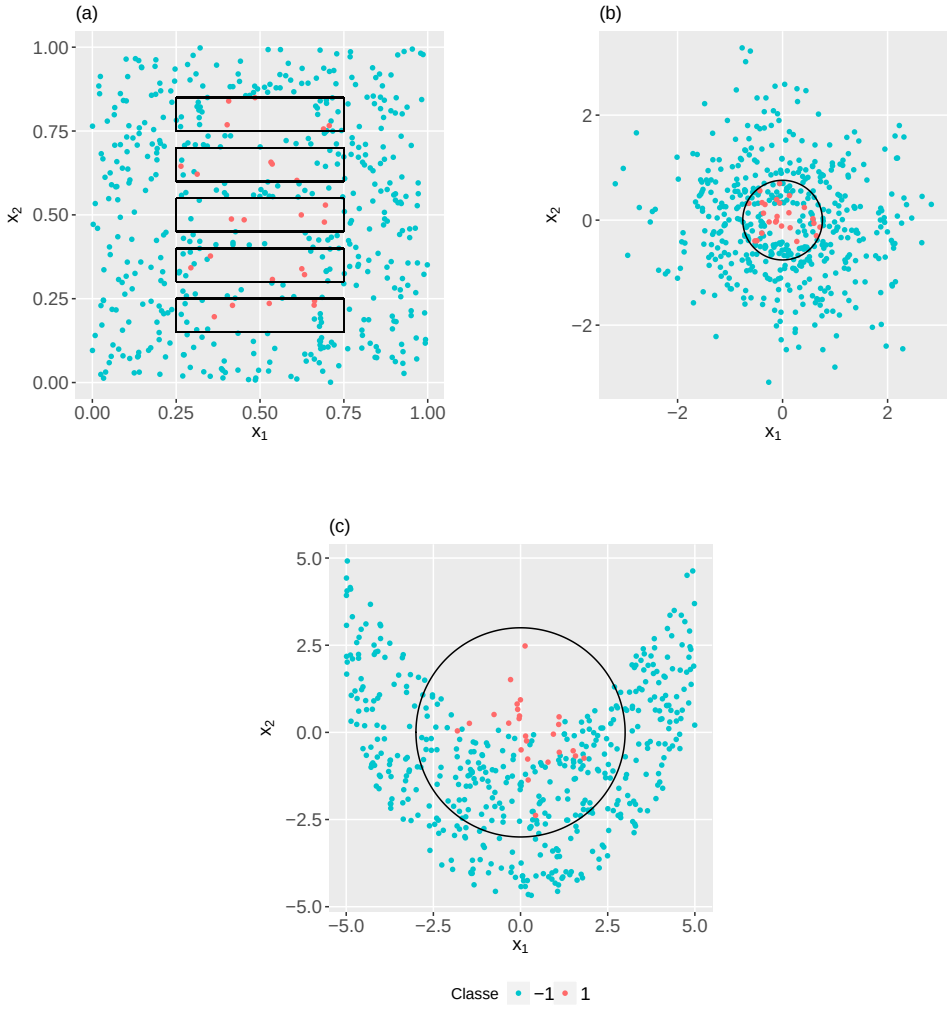

Figura 4.1: (a) Gruppi disgiunti. (b) Sfere annidate. (c) Gruppi con parabola. Ciascun campione ha 500 osservazioni, di cui il 5% costituito dalla classe rara, che giace nella regione delimitata in nero.

- *Sfere annidate.* La costruzione di questo insieme di dati trae spunto da Hastie *et al.* (2001, Cap. 10). Le variabili esplicative sono generate da variabili casuali normali standard indipendenti e le sfere sono costruite in modo che le unità più vicine all'origine appartengano alla classe rara. Per definire la frontiera decisionale che separa le due classi si calcola il raggio al quadrato $r^2 = \sum_{j=1}^{d} x_j^2$, dove d è il numero delle variabili esplicative. Gli esempi positivi sono quindi estratti in modo che r_i^2 sia inferiore ad una certa soglia. In particolare, si

fissa tale soglia pari al quantile 0.25 di una χ^2_d, per sfruttare la relazione che intercorre tra il quadrato di v.a. normali standard e v.a. chi-quadro e ricavare così una misura per il raggio coerente all'aumentare della dimensionalità dello spazio. Per creare una zona di sovrapposizione fra le classi, le unità di quella negativa soddisfano la condizione: $r^2 = \sum_{j=1}^{d} x_j^2 > \chi^2_d(0.05)$.

- *Gruppi con parabola.* In quest'ultima configurazione dei dati le unità della classe rara sono generate da una normale standard di dimensione d, mentre quelle della classe prevalente occupano una regione concava, con una sovrapposizione fra le due nella zona centrale. La regione della classe negativa è delimitata superiormente da una parabola di equazione $x_2 = 0.2x_1^2$ e inferiormente dalla stessa traslata di -5, quindi $x_2 = 0.2x_1^2 - 5$. La variabile x_1 assume valori in $[-5, 5]$ e, agli estremi, x_2 vale 5, quindi in $\mathbb{R}^2$ i dati occupano un quadrato di lato [-5,5] e le unità negative sono generate uniformemente entro la regione concava. L'estensione a più dimensioni può essere ottenuta in vari modi. Nel generico spazio $\mathbb{R}^d$ la regione per la classe rara rimane un ipersfera, mentre per la forma parabolica si aggiungono variabili da normali standard indipendenti.

4.2.2 Modelli concorrenti

Gli scenari descritti vengono utilizzati per confrontare la qualità previsiva di diversi modelli che, principalmente, si basano sul *boosting* e sono rappresentativi dei vari rimedi al problema dello sbilanciamento presentati nel paragrafo 2.4. Nel confronto si inseriscono anche altri modelli, tra cui le due proposte trattate nel Capitolo 3: l'albero di classificazione modificato con l'AUC e il *gradient boosting* con funzione di perdita basata sull'AUC.

La metrica scelta per la valutazione e il confronto dei modelli stimati è l'AUC, calcolata su insiemi di verifica di 100000 unità generati replicando le stesse caratteristiche del campione di stima. L'accuratezza previsiva di molti dei modelli coinvolti nello studio dipende, tuttavia, dalla selezione di quei parametri che ne impediscono il sovra-adattamento ai dati, individuando il compromesso ideale fra varianza e distorsione del modello. Per via dei numerosi casi in esame, non è stato operativamente possibile attuare delle scelte che fossero ovunque superiori ad altre, ma si è trattato, nella maggior parte dei casi, di decisioni arbitrarie, motivate da considerazioni pratiche. Quando possibile, i parametri sono stati selezionati valutando preliminarmente

i modelli su alcuni campioni, così che potesse delinearsi un'indicazione di quali scelte fossero ottimali in più contesti.

Per rendere equo il confronto fra gli algoritmi basati sul *boosting* vengono posti uguali sia il classificatore di base che il numero totale delle iterazioni. Tra le varie alternative disponibili per il modello di base, si sceglie quella più ricorrente nei vari studi e applicazioni pratiche, ossia lo *stump*. Relativamente al numero di iterazioni, poiché in molti casi è stato verificato che la *performance* del modello tende a stabilizzarsi da un certo numero in poi, nelle analisi svolte si è fissato un numero di iterazioni pari a 200, ritenendo sia sufficientemente grande.

I due principali modelli di *boosting* che vengono stimati sono quelli descritti nella prima parte della Sezione 2.4: l'algoritmo *AdaBoost* e il *gradient boosting*. Per quest'ultimo, è stata scelta la funzione di perdita esponenziale per analogia con *AdaBoost*, mentre il parametro di *shrinkage* è stato selezionato confrontando tre valori: $\{0.1, 0.3, 0.5\}$ su 50 campioni per ogni struttura dei dati ed alcune combinazioni di numerosità, dimensione e grado di sbilanciamento. Le differenze a favore di uno o dell'altro si sono rivelate esigue e non è emersa, soprattutto, una netta predominanza. Si è scelto quindi il valore 0.3, che in molti contesti ha portato ai migliori risultati.

Tra i metodi che combinano *AdaBoost* con tecniche di ricampionamento sono stati scelti *SMOTEBoost* e *RUSBoost*. Entrambi richiedono che sia specificata una nuova proporzione fra le classi da raggiungere ad ogni iterazione dell'algoritmo tramite, rispettivamente, sovracampionamento e sottocampionamento. Non necessariamente un bilanciamento perfetto fra le due classi è la soluzione ottimale, perché la distribuzione ideale dipende dai dati in esame e, non essendo nota a priori, andrebbe esplorata ogni volta. Inoltre, imporre una distribuzione equa fra le classi implicherebbe quasi raddoppiare la numerosità campionaria con *SMOTEBoost*, aggravando notevolmente i costi computazionali, e perdere, invece, una quota rilevante dell'informazione disponibile con *RUSBoost*. Si è quindi scelto di alterare la distribuzione dei dati imponendo che, ad ogni iterazione, la classe rara raggiunga un'occorrenza del 30% rispetto alla numerosità campionaria.

L'algoritmo di *cost-sensitive boosting* che si è scelto di includere nello studio è *AdaC2*, per le ragioni spiegate nel paragrafo di riferimento 2.4.3. L'assegnazione di costi di errata classificazione diversi per le unità che appartengono alle due classi è un modo alternativo, rispetto al ricampionamento, di attribuire più peso alle unità

positive, definendo per esse un costo maggiore. Raramente i costi sono noti a priori e il problema è analogo alla ricerca della distribuzione ottimale per i metodi di ricampionamento. Una possibilità, che viene spesso adottata, è fissare c_{FN} pari a 1 e confrontare alcuni valori di c_{FP}, minori di 1, per scegliere quello ideale. L'approccio seguito invece nello studio corrente può risultare semplicistico da un certo punto di vista, ma consente di confrontare in maniera equa gli scenari coinvolti sfruttando un'informazione nota come il grado di sbilanciamento ed evitando il costo operativo associato alla ricerca del valore di c_{FP} ottimale nei vari casi. Una prassi comune quando i dati sono sbilanciati è fissare la soglia di classificazione k pari alla frequenza relativa di casi della classe minoritaria. Assumendo allora $c_{FN} = 1$ e la definizione $k = c_{FP}/(c_{FN} + c_{FP})$ (2.2) si ha che $c_{FP} = k/(1-k)$. Quindi, con il 5% di unità rare risulta $c_{FP} = k/(1-k) = 0.05/0.95 \simeq 0.05$ e per gli altri casi il calcolo è analogo.

Per quanto concerne le due nuove proposte, analogamente ai modelli descritti finora, si sono dovute prendere delle decisioni che ne ultimassero la struttura. In particolare, si è lasciato crescere l'albero di classificazione modificato tramite l'AUC senza imporre ulteriori limiti rispetto a quello stabilito in fase di costruzione, ossia che, ad ogni *split*, siano valutate solo le variabili esplicative con un AUC rispetto alla variabile risposta maggiore di 0.5. Per avere un termine di confronto diretto rispetto a questo metodo, si è valutato anche l'albero di classificazione usuale, con criterio di *split* dato dall'indice di Gini e nessun vincolo sulla crescita se non quelli imposti per *default*. Il *gradient boosting* con funzione di perdita basata sull'AUC richiede invece la specificazione di diversi parametri, oltre al numero di iterazioni e al modello di base, scelti in conformità con quanto precedentemente indicato. Relativamente al grado di *shrinkage*, si è operato come per il *gradient boosting* con funzione di perdita esponenziale, confrontando i tre valori e scegliendo, in questo caso, $\rho = 0.1$. Nella definizione della funzione di perdita, invece, i parametri p e τ sono stati scelti seguendo le indicazioni di Zhao *et al.* (2011), quindi $p = 2$ e $\tau = 0.7$.

Infine, fra i metodi analizzati, si include anche la regressione logistica, uno dei più tradizionali modelli parametrici di classificazione. Le sue proprietà teoriche rispetto a dati fortemente sbilanciati sono state discusse in vari lavori, fra i quali si cita quello di King e Zeng (2001), dove è stato verificato come le probabilità stimate di appartenere alla classe positiva siano in realtà sottostimate per le unità di quella stessa classe. Si ritiene interessante, quindi, osservare empiricamente il comportamento del modello rispetto agli scenari costruiti, trattandosi di un metodo

tipicamente efficace nei problemi di classificazione più comuni.

Da un punto di vista strettamente operativo, le analisi sono state condotte con il linguaggio di programmazione R. In particolare, l'albero di classificazione è stato costruito con il pacchetto rpart (Therneau *et al.*, 2017), l'algoritmo *AdaBoost* nella sua versione originale si trova implementato nel pacchetto ada (Culp *et al.*, 2016), il *gradient boosting* è stato stimato tramite le funzioni del pacchetto gbm (Ridgeway, 2017) e gli algoritmi *SMOTEBoost* e *RUSBoost* sono disponibili nel pacchetto ebmc (Hao e Chen, 2017), sebbene in questo caso le funzioni siano state in parte ridefinite. Si sono introdotte delle modifiche affinché, come negli altri algoritmi di *boosting* nello studio, i pesi venissero inglobati direttamente dal classificatore di base senza ricampionare le unità ad ogni passo. Inoltre, sono stati cambiati il pacchetto e, di conseguenza, la funzione da cui *SMOTEBoost* attinge per la tecnica SMOTE, per avere più flessibilità nel definire la nuova proporzione desiderata tra le classi. L'algoritmo *AdaC2* è stato implementato appositamente, non essendo attualmente disponibile in alcun pacchetto di R, ed è presentato in Appendice. Infine, tra i molti pacchetti predisposti al calcolo dell'AUC, si è scelto di utilizzare ROCR (Sing *et al.*, 2005).

4.2.3 Risultati

I risultati sono riportati nelle Tabelle 4.1, 4.2 e 4.3, distinguendo per le tre configurazioni di dati e seguendo l'ordine con cui sono state presentate. Per ogni scenario sono stati calcolati la media e lo scarto quadratico medio dell'AUC per 300 simulazioni. Gli unici valori assenti sono quelli del *gradient boosting* con funzione di perdita basata sull'AUC nel caso in cui la numerosità del campione di stima sia pari a 10000. L'onere computazionale dell'algoritmo si aggrava fortemente al crescere della numerosità campionaria, perché la funzione di perdita, essendo originariamente formulata per coppie di unità, confronta ogni osservazione con tutte quelle che la precedono nel campione. Nell'analisi ci si limita pertanto al solo caso in cui $n = 1000$, computazionalmente sostenibile.

Osservando globalmente i risultati ottenuti si possono trarre alcune prime considerazioni relative ad andamenti comuni. L'aumento della dimensione dello spazio implica, generalmente, un deterioramento dei risultati, legato al fenomeno della *maledizione della dimensionalità*. Considerare più dimensioni significa, in altri termini,

aumentare il grado di sparsità dei dati, con la creazione di spazi vuoti che rendono sempre più complicato individuare la regione della classe rara. Per compensare il crescente spazio tra i punti, la numerosità campionaria dovrebbe crescere con un ordine di grandezza di n^d, una strada operativamente non percorribile per la quasi totalità dei casi pratici. Il fenomeno della maledizione della dimensionalità è particolarmente problematico per i metodi non parametrici, mentre è meno vincolante per quelli parametrici, che assumono una qualche struttura per il modello da cui provengono i dati e che è di interesse stimare. Aumentare la numerosità campionaria, sebbene non dell'ordine teoricamente richiesto, è sicuramente un modo di rimediare al problema e alleviarne le conseguenze, rendendo più popolata la regione della classe rara. Anche agire direttamente sulla distribuzione delle classi è una soluzione, aumentando la percentuale di unità positive sulla numerosità complessiva e rendendone quindi più rappresentativa la regione di appartenenza. Avere a disposizione più dati, con una dimensione modesta e classi meno sbilanciate è una combinazione di fattori che semplifica il problema della classificazione per qualsiasi metodo, come è, d'altra parte, intuitivo. Sulla base di queste osservazioni, si comprende allora come il caso in assoluto più problematico dello studio sia quello in cui $n = 1000$, $d = 10$ e la percentuale di unità positive pari allo 0.6%.

La regressione logistica, nella maggior parte dei casi, non offre risultati migliori di quelli che si avrebbero con una banale classificazione casuale delle unità. L'AUC medio è, infatti, pari a 0.5 in qualsiasi scenario relativo alle prime due strutture di dati, mentre migliora nell'ultima configurazione, in cui il gruppo per la classe maggiore ha la forma, in due dimensioni, di una parabola. La *performance* rimane comunque modesta, soprattutto al crescere del numero di variabili esplicative, e non sembra risentire particolarmente della percentuale di unità positive nel campione così come della numerosità campionaria.

La *performance* di *AdaBoost* è molto soddisfacente in tutti gli scenari considerati. Lo sbilanciamento fra le classi non sembra essere un ostacolo rilevante, soprattutto quando la numerosità campionaria è pari a 10000, situazione in cui il valore medio dell'AUC è spesso maggiore di 0.9. L'aumento della numerosità campionaria o della frequenza assoluta di osservazioni positive implicano netti miglioramenti nel valore medio dell'AUC. Come è naturale attendersi, lo scarto quadratico medio è generalmente maggiore quando la numerosità è inferiore o lo sbilanciamento è più accentuato, ossia quando si complica il problema della classificazione.

Gli algoritmi *SMOTEBoost* e *RUSBoost* alterano entrambi *AdaBoost* tramite un ricampionamento delle unità dal campione di stima ad ogni iterazione, ma utilizzano strategie opposte e conducono a risultati molto diversi. Il metodo *SMOTEBoost* mostra la *performance* peggiore tra gli algoritmi basati sul *boosting* inclusi nello studio, in certi casi superiore di poco ad una classificazione casuale delle unità e con un valore medio dell'AUC quasi sempre minore di 0.7. Questo metodo risulta scarsamente sensibile a variazioni nella struttura dei dati e mostra, in qualche caso, un comportamento anomalo rispetto agli altri algoritmi, dimostrando di peggiorare con una numerosità maggiore o di migliorare all'aumentare del numero di esplicative. Sarebbe interessante approfondire l'analisi di questo metodo rispetto, ad esempio, ad altri gradi di bilanciamento fra le classi. I risultati prodotti dall'algoritmo *RUSBoost* sono decisamente migliori, per quanto rimangano anch'essi inferiori all'originario *AdaBoost*. Si nota, in particolare, una maggiore sensibilità alla dimensione campionaria nei casi con $n = 1000$ e, soprattutto, una percentuale dello 0.6% per le unità della classe rara. Altrimenti, fatta eccezione per le sfere annidate, con $n = 10000$ non sembra esserci alcuna differenza al variare della dimensione. I valori dello scarto quadratico medio osservati sia per *SMOTEBoost* che per *RUSBoost* sono, tipicamente, più alti che per gli altri algoritmi di *boosting*. Probabilmente, il modo con cui *SMOTEBoost* genera nuove unità sintetiche e, nel caso di *RUSBoost*, il fatto di usare una porzione esigua e variabile dell'informazione totale ad ogni passo provocano maggiore instabilità nella procedura.

L'algoritmo di *cost-sensitive boosting*, *AdaC2*, ha una *performance* generalmente migliore di *RUSBoost*, più comparabile con *AdaBoost* ma ancora inferiore nella maggior parte dei casi. Nel dettaglio, *AdaC2* offre risultati migliori con le sfere annidate, quando $n = 1000$ e la percentuale di unità della classe rara è dello 0.6% e dell'1%. Lo scarto quadratico medio è vicino ai corrispettivi valori di *AdaBoost*, risultando maggiore in alcuni casi con $d = 10$.

Il *gradient boosting* con funzione di perdita esponenziale è un'implementazione del classico *AdaBoost* che segue l'algoritmo di Friedman (2001, Algoritmo 2.3). In molti casi, con il *gradient boosting* i valori medi dell'AUC tendono ad essere superiori e quelli dello scarto quadratico medio inferiori rispetto ad *AdaBoost*, soprattutto quando la numerosità campionaria è elevata.

La variante del *gradient boosting* con funzione di perdita ricavata a partire dall'AUC mostra risultati spesso migliori rispetto al *gradient boosting* con funzione di

perdita esponenziale. La differenza è evidente soprattutto quando il grado di sbilanciamento è più forte e tende ad affievolirsi quando la percentuale di casi rari è del 5%. Lo scarto quadratico medio è approssimativamente uguale fra i due algoritmi.

I risultati dell'albero di classificazione proposto nel Capitolo 3 vengono confrontati con quelli di un albero ottenuto con i parametri di *default* del pacchetto rpart, come spiegato nel paragrafo precedente. La nuova proposta mostra una *performance* superiore quando, in due dimensioni, il grado di sbilanciamento è più estremo. All'aumentare della numerosità campionaria o della percentuale di casi positivi sul totale, il classico albero basato sull'indice di Gini ha un valore medio dell'AUC generalmente superiore e, in certi casi, comparabile con gli algoritmi basati sul *boosting*. Dall'altro lato, il suo scarto quadratico medio è talvolta elevato, mentre quello dell'albero che si basa sull'AUC rimane sempre ridotto. Dal confronto emerge che, potenzialmente, la nuova formulazione dell'albero potrebbe condurre a risultati migliori di quello classico quando il problema dello sbilanciamento è più accentuato, ma sembra non sostenere il confronto con dimensionalità elevate. In fase di costruzione, la scelta di selezionare prima la variabile e successivamente il miglior punto di *split* ha un vantaggio naturale in termini di tempo computazionale, perché limita e vincola la ricerca della miglior suddivisione ad una sola variabile. Di fatto, però, i risultati di questo studio suggeriscono l'eventualità di implementarne una versione che, analogamente a rpart, ottimizzi simultaneamente le due scelte, selezionando lo *split* ottimale sulla base di un confronto fra tutti i possibili punti di *split*.

%	n	d	*Regressione logistica*	*Albero Gini*	*AdaBoost*	*Gradient boosting*	*SMOTE-Boost*	*RUSBoost*	*AdaC2*	*Albero AUC*	*Gradient boosting AUC*
0.6	1000	2	0.500 (0.004)	0.500 (0.000)	0.772 (0.047)	0.782 (0.043)	0.602 (0.059)	0.777 (0.058)	0.781 (0.048)	0.632 (0.068)	0.790 (0.041)
		5	0.500 (0.012)	0.500 (0.500)	0.721 (0.041)	0.712 (0.040)	0.563 (0.059)	0.667 (0.073)	0.703 (0.045)	0.527 (0.048)	0.736 (0.042)
		10	0.500 (0.012)	0.500 (0.500)	0.676 (0.033)	0.644 (0.040)	0.540 (0.056)	0.561 (0.045)	0.663 (0.033)	0.501 (0.011)	0.684 (0.039)
	10000	2	0.500 (0.005)	0.757 (0.156)	0.916 (0.013)	0.934 (0.011)	0.663 (0.056)	0.803 (0.018)	0.878 (0.012)	0.684 (0.078)	–
		5	0.499 (0.012)	0.690 (0.169)	0.904 (0.013)	0.921 (0.011)	0.672 (0.065)	0.802 (0.020)	0.847 (0.022)	0.603 (0.063)	–
		10	0.501 (0.012)	0.627 (0.164)	0.895 (0.012)	0.900 (0.012)	0.665 (0.063)	0.802 (0.017)	0.799 (0.035)	0.514 (0.022)	–
1	1000	2	0.500 (0.003)	0.501 (0.008)	0.830 (0.034)	0.838 (0.030)	0.632 (0.059)	0.753 (0.080)	0.821 (0.034)	0.647 (0.056)	0.833 (0.030)
		5	0.499 (0.008)	0.501 (0.014)	0.786 (0.032)	0.777 (0.030)	0.609 (0.067)	0.750 (0.058)	0.743 (0.048)	0.544 (0.057)	0.790 (0.031)
		10	0.501 (0.009)	0.500 (0.007)	0.741 (0.027)	0.723 (0.028)	0.590 (0.067)	0.684 (0.064)	0.684 (0.039)	0.504 (0.017)	0.754 (0.030)
	10000	2	0.500 (0.004)	0.858 (0.116)	0.921 (0.010)	0.946 (0.008)	0.664 (0.055)	0.796 (0.008)	0.886 (0.009)	0.693 (0.082)	–
		5	0.500 (0.009)	0.849 (0.122)	0.918 (0.010)	0.940 (0.007)	0.677 (0.063)	0.797 (0.007)	0.868 (0.015)	0.626 (0.067)	–
		10	0.501 (0.009)	0.787 (0.165)	0.912 (0.010)	0.929 (0.007)	0.685 (0.068)	0.797 (0.007)	0.842 (0.024)	0.520 (0.024)	–
5	1000	2	0.500 (0.002)	0.781 (0.151)	0.914 (0.012)	0.931 (0.011)	0.670 (0.064)	0.811 (0.026)	0.879 (0.013)	0.686 (0.077)	0.910 (0.012)
		5	0.500 (0.004)	0.787 (0.136)	0.898 (0.013)	0.909 (0.012)	0.679 (0.071)	0.809 (0.024)	0.861 (0.020)	0.604 (0.064)	0.897 (0.013)
		10	0.500 (0.004)	(0.791) (0.124)	0.888 (0.013)	0.886 (0.013)	0.685 (0.072)	0.809 (0.031)	0.838 (0.028)	0.517 (0.018)	0.880 (0.013)
	10000	2	0.500 (0.002)	0.958 (0.008)	0.931 (0.007)	0.965 (0.004)	0.669 (0.055)	0.795 (0.001)	0.901 (0.008)	0.694 (0.090)	–
		5	0.500 (0.004)	0.957 (0.008)	0.930 (0.006)	0.964 (0.003)	0.688 (0.069)	0.795 (0.001)	0.901 (0.008)	0.667 (0.077)	–
		10	0.500 (0.004)	0.958 (0.007)	0.931 (0.007)	0.963 (0.003)	0.689 (0.069)	0.795 (0.001)	0.900 (0.008)	0.540 (0.024)	–

Tabella 4.1: Valore medio dell'AUC per i gruppi disgiunti, calcolato su 300 campioni e, tra parentesi, il corrispondente scarto quadratico medio.

%	n	d	*Regressione logistica*	*Albero Gini*	*AdaBoost*	*Gradient boosting*	*SMOTE-Boost*	*RUSBoost*	*AdaC2*	*Albero AUC*	*Gradient boosting AUC*
0.6	1000	2	0.500 (0.005)	0.501 (0.012)	0.780 (0.045)	0.782 (0.036)	0.562 (0.042)	0.751 (0.070)	0.782 (0.051)	0.619 (0.069)	0.790 (0.040)
		5	0.500 (0.008)	0.500 (0.000)	0.724 (0.038)	0.707 (0.039)	0.521 (0.026)	0.579 (0.051)	0.764 (0.034)	0.523 (0.036)	0.729 (0.042)
		10	0.500 (0.009)	0.500 (0.002)	0.641 (0.034)	0.612 (0.039)	0.507 (0.019)	0.490 (0.035)	0.720 (0.039)	0.501 (0.014)	0.650 (0.045)
	10000	2	0.500 (0.005)	0.664 (0.162)	0.898 (0.010)	0.899 (0.008)	0.642 (0.049)	0.866 (0.022)	0.875 (0.008)	0.690 (0.074)	–
		5	0.500 (0.008)	0.502 (0.021)	0.868 (0.012)	0.884 (0.008)	0.582 (0.032)	0.733 (0.039)	0.833 (0.009)	0.619 (0.034)	–
		10	0.500 (0.009)	0.500 (0.000)	0.809 (0.016)	0.853 (0.010)	0.553 (0.023)	0.646 (0.031)	0.802 (0.011)	0.539 (0.024)	–
1	1000	2	0.500 (0.004)	0.510 (0.039)	0.826 (0.030)	0.824 (0.031)	0.592 (0.046)	0.751 (0.060)	0.820 (0.039)	0.645 (0.061)	0.830 (0.029)
		5	0.500 (0.006)	0.500 (0.003)	0.778 (0.027)	0.767 (0.028)	0.540 (0.028)	0.611 (0.045)	0.806 (0.024)	0.542 (0.035)	0.778 (0.031)
		10	0.500 (0.007)	0.500 (0.001)	0.703 (0.028)	0.692 (0.032)	0.518 (0.020)	0.545 (0.029)	0.774 (0.027)	0.505 (0.016)	0.706 (0.037)
	10000	2	0.500 (0.004)	0.732 (0.166)	0.903 (0.006)	0.906 (0.005)	0.650 (0.054)	0.872 (0.014)	0.877 (0.006)	0.697 (0.075)	–
		5	0.500 (0.006)	0.502 (0.018)	0.879 (0.009)	0.895 (0.005)	0.587 (0.035)	0.750 (0.037)	0.827 (0.008)	0.634 (0.037)	–
		10	0.500 (0.008)	0.500 (0.000)	0.825 (0.013)	0.870 (0.007)	0.559 (0.025)	0.665 (0.030)	0.787 (0.011)	0.553 (0.022)	–
5	1000	2	0.500 (0.002)	0.777 (0.122)	0.895 (0.010)	0.897 (0.007)	0.663 (0.064)	0.861 (0.024)	0.873 (0.012)	0.681 (0.067)	0.897 (0.008)
		5	0.500 (0.003)	0.613 (0.088)	0.863 (0.011)	0.874 (0.009)	0.603 (0.041)	0.723 (0.037)	0.834 (0.014)	0.613 (0.032)	0.865 (0.010)
		10	0.500 (0.003)	0.537 (0.047)	0.803 (0.014)	0.839 (0.011)	0.577 (0.028)	0.642 (0.033)	0.778 (0.037)	0.533 (0.021)	0.818 (0.014)
	10000	2	0.500 (0.002)	0.852 (0.109)	0.910 (0.003)	0.913 (0.001)	0.675 (0.066)	0.873 (0.011)	0.878 (0.005)	0.706 (0.076)	–
		5	0.500 (0.003)	0.500 (0.000)	0.894 (0.004)	0.909 (0.002)	0.613 (0.046)	0.777 (0.032)	0.821 (0.018)	0.663 (0.053)	–
		10	0.500 (0.003)	0.500 (0.000)	0.864 (0.006)	0.896 (0.002)	0.590 (0.034)	0.699 (0.025)	0.704 (0.051)	0.595 (0.018)	–

Tabella 4.2: Risultati per le sfere annidate. Cfr. Tabella 4.1.

%	n	d	*Regressione logistica*	*Albero Gini*	*AdaBoost*	*Gradient boosting*	*SMOTE-Boost*	*RUSBoost*	*AdaC2*	*Albero AUC*	*Gradient boosting AUC*
0.6	1000	2	0.640 (0.046)	0.518 (0.059)	0.797 (0.056)	0.824 (0.048)	0.621 (0.051)	0.815 (0.061)	0.807 (0.059)	0.651 (0.069)	0.829 (0.043)
		5	0.564 (0.034)	0.504 (0.027)	0.778 (0.046)	0.789 (0.044)	0.599 (0.068)	0.711 (0.076)	0.745 (0.049)	0.578 (0.061)	0.799 (0.041)
		10	0.536 (0.022)	0.500 (0.002)	0.752 (0.043)	0.743 (0.050)	0.572 (0.072)	0.618 (0.062)	0.719 (0.040)	0.512 (0.025)	0.766 (0.044)
	10000	2	0.647 (0.006)	0.844 (0.035)	0.928 (0.012)	0.931 (0.010)	0.672 (0.043)	0.861 (0.068)	0.903 (0.016)	0.740 (0.083)	–
		5	0.621 (0.019)	0.841 (0.033)	0.926 (0.011)	0.927 (0.010)	0.680 (0.047)	0.865 (0.059)	0.879 (0.023)	0.682 (0.047)	–
		10	0.596 (0.019)	0.846 (0.036)	0.922 (0.010)	0.921 (0.009)	0.681 (0.050)	0.862 (0.066)	0.839 (0.033)	0.525 (0.029)	–
1	1000	2	0.647 (0.019)	0.633 (0.133)	0.843 (0.046)	0.863 (0.034)	0.645 (0.055)	0.834 (0.050)	0.847 (0.040)	0.679 (0.066)	0.862 (0.036)
		5	0.577 (0.031)	0.562 (0.111)	0.826 (0.040)	0.840 (0.028)	0.635 (0.063)	0.794 (0.072)	0.789 (0.042)	0.611 (0.056)	0.840 (0.035)
		10	0.549 (0.023)	0.515 (0.055)	0.809 (0.035)	0.812 (0.032)	0.621 (0.069)	0.746 (0.074)	0.734 (0.045)	0.513 (0.025)	0.821 (0.036)
	10000	2	0.647 (0.005)	0.861 (0.028)	0.937 (0.007)	0.938 (0.006)	0.678 (0.047)	0.873 (0.044)	0.903 (0.015)	0.739 (0.084)	–
		5	0.630 (0.013)	0.860 (0.029)	0.935 (0.007)	0.935 (0.006)	0.685 (0.050)	0.874 (0.043)	0.890 (0.020)	0.705 (0.048)	–
		10	0.611 (0.016)	0.858 (0.030)	0.933 (0.006)	0.932 (0.006)	0.691 (0.055)	0.873 (0.044)	0.874 (0.027)	0.530 (0.025)	–
5	1000	2	0.647 (0.003)	0.851 (0.036)	0.924 (0.013)	0.928 (0.009)	0.690 (0.059)	0.839 (0.088)	0.904 (0.016)	0.724 (0.079)	0.925 (0.011)
		5	0.620 (0.018)	0.853 (0.035)	0.920 (0.012)	0.920 (0.009)	0.707 (0.069)	0.841 (0.085)	0.892 (0.019)	0.681 (0.049)	0.920 (0.011)
		10	0.594 (0.018)	0.855 (0.038)	0.916 (0.011)	0.914 (0.008)	0.705 (0.065)	0.843 (0.084)	0.873 (0.027)	0.533 (0.025)	0.915 (0.011)
	10000	2	0.646 (0.002)	0.891 (0.014)	0.947 (0.002)	0.949 (0.002)	0.686 (0.053)	0.861 (0.054)	0.897 (0.010)	0.748 (0.083)	–
		5	0.642 (0.004)	0.890 (0.014)	0.947 (0.002)	0.947 (0.002)	0.691 (0.053)	0.862 (0.047)	0.897 (0.010)	0.738 (0.048)	–
		10	0.636 (0.006)	0.891 (0.015)	0.947 (0.002)	0.947 (0.002)	0.688 (0.049)	0.863 (0.051)	0.897 (0.009)	0.557 (0.025)	–

Tabella 4.3: Risultati per i gruppi con parabola. Cfr. Tabella 4.1.

4.3 Applicazione a dati reali

L'analisi empirica si conclude applicando i nuovi metodi proposti e i modelli da cui sono stati derivati a dei dati reali che presentano il problema dello sbilanciamento, in modo da poter esplorare ulteriormente la loro accuratezza previsiva, valutata come prima in termini di AUC.

Diversamente dallo studio di simulazione, dove è stato inevitabile prendere alcune decisioni in modo arbitrario che non fossero necessariamente ottimali per ogni scenario, è ora ragionevole stimare ciascun modello sulla base di una valutazione accorta dei parametri che entrano in gioco. Si è scelto quindi di dividere casualmente l'insieme di dati a disposizione in tre parti, circa della stessa numerosità. La prima parte costituisce l'insieme di stima, da usare per stimare i diversi modelli concorrenti. La seconda, l'*insieme di convalida*, serve a determinare i parametri di regolazione per ogni modello. Infine, l'ultima parte è l'insieme di verifica, che consente di verificare l'accuratezza previsiva soltanto dei modelli scelti nella precedente fase. Come è stato spiegato nel Capitolo 1, questo modo di procedere nei contesti reali in cui le classi sono fortemente sbilanciate può presentare un inconveniente, legato alla potenziale scarsità, o assenza, dei casi rari nei vari sottoinsiemi. Quindi, sebbene questo non risolva completamente il problema, i tre sottoinsiemi sono stati ottenuti estraendo le osservazioni dall'insieme originario in modo stratificato rispetto alle classi, così da garantire, almeno, la presenza delle unità più rare in ciascuno.

Per quanto riguarda il *gradient boosting*, si è deciso di mantenere fissato il numero di iterazioni a 200, per via del costo computazionale legato all'utilizzo della funzione di perdita basata sull'AUC, e si sono valutati tre valori per il parametro di *shrinkage*: $\rho = \{0.1, 0.3, 0.5\}$. Per l'albero, invece, si sono considerati quattro possibili livelli di profondità: $\mathrm{p} = \{1, 2, 4, 30\}$, dove con 1 si indica lo *stump*, e tre valori per il numero minimo di osservazioni da considerare ad ogni nodo per provare lo *split*: $\mathrm{ms} = \{2, 3 \times n/100, 5 \times n/100\}$, essendo n la numerosità del campione di stima.

I *dataset* utilizzati per le analisi e le relative fonti sono descritti a seguire.

- *Abalone* (Dheeru e Karra Taniskidou, 2017) è un insieme di 4177 osservazioni e 7 variabili esplicative numeriche relative a diverse misurazioni fisiche degli abaloni. La variabile risposta esprime la loro età e assume valori da 1 a 29. Essendo rari gli esemplari che superano i 20 anni, si è scelto di usare questa

soglia per rendere y dicotomica. La classe positiva conta quindi 62 unità, con una percentuale dell'1.48% sulla numerosità campionaria complessiva.

- *Phoneme* (ELENA[1]) è un insieme di 5404 unità e 5 variabili esplicative numeriche relative a misurazioni di suoni. La variabile risposta è dicotomica, la classe positiva rappresenta i suoni orali e quella negativa i suoni nasali. È già presente uno sbilanciamento fra le classi del 29%, ma per mettere alla prova i modelli con una classe rara si costruisce l'insieme di stima in modo che la classe positiva ne costituisca il 3%.

Nella Tabella 4.4 si riportano i risultati delle analisi per i *dataset* presentati. In particolare, per ciascuno sono indicati i valori dei parametri di regolazione scelti sull'insieme di convalida e il valore dell'AUC calcolato sull'insieme di verifica. I risultati sono coerenti con quelli emersi nello studio di simulazione. Per entrambi i *dataset*, il *gradient boosting* con funzione di perdita basata sull'AUC raggiunge risultati migliori che con la funzione di perdita esponenziale. L'albero di classificazione basato sull'AUC, invece, è superiore alla variante classica che usa l'indice di Gini nel caso di *Phoneme*, ma non con *Abalone*. Quando il numero di variabili esplicative è più ridotto il nuovo albero tende a fornire buoni risultati perché è più probabile che il miglior punto di *split* in assoluto dipenda proprio dalla variabile che è stata selezionata per la sua scelta, altrimenti, come si è già avuto modo di notare, questa procedura dovrebbe essere ulteriormente ottimizzata.

		Gradient boosting	*Gradient boosting AUC*	*Albero Gini*	*Albero AUC*
Abalone	Parametri	$\rho = 0.3$	$\rho = 0.1$	p = 2, ms = $5 \times n/100$	p = 2, ms = 2
	AUC	0.794	0.839	0.809	0.762
Phoneme	Parametri	$\rho = 0.1$	$\rho = 0.1$	p = 4, ms = $3 \times n/100$	p = 2, ms = 2
	AUC	0.812	0.830	0.753	0.771

Tabella 4.4: Parametri selezionati e valore dell'AUC per i modelli stimati e i *dataset* reali.

[1] https://www.elen.ucl.ac.be/neural-nets/Research/Projects/ELENA/elena.htm

Conclusione

In questa tesi sono stati approfonditi alcuni aspetti rilevanti legati al problema della classificazione in presenza di una classe rara, con l'obiettivo finale di proporre delle valide soluzioni al problema. Dopo una prima contestualizzazione del fenomeno, volta a chiarirne la natura e le implicazioni sulla qualità dei modelli di classificazione tradizionali, si è fornita una rassegna generale dei principali approcci che possono essere perseguiti per gestire adeguatamente il problema, riservando particolare attenzione ai metodi di *boosting*. A seguire sono state introdotte le due nuove proposte, nate da alcune considerazioni relative all'AUC: un albero di classificazione con criterio di crescita basato sull'AUC e un algoritmo di *gradient boosting* con modello di base dato dallo *stump* e una funzione di perdita appositamente ricavata per ottimizzare direttamente l'AUC.

I risultati emersi dalle analisi empiriche hanno permesso di confrontare i diversi metodi coinvolti, tra cui le nuove proposte, in modo da esplorarne la capacità discriminante in contesti sia simulati che reali in cui è presente una classe rara. Sorprendentemente, lo studio di simulazione ha messo in luce come il classico e più semplice algoritmo *AdaBoost* sia preferibile non solo al suo adattamento che ingloba costi di errata classificazione maggiori per le unità positive, ma anche alle versioni basate sul ricampionamento specificatamente derivate per gestire la presenza di una classe rara. Sarebbe interessante approfondire ulteriormente questo risultato, ricercando criteri non euristici di assegnazione dei costi e per l'individuazione della distribuzione ottimale. In particolare, quando i costi reali sono disponibili, sembra comunque il caso di valutare un algoritmo di *cost-sensitive boosting*, avendo notato come lo scarto fra *AdaC2* e *AdaBoost* negli scenari analizzati sia di fatto esiguo e, inoltre, possa dipendere da una scelta dei costi non ottimale.

Il *gradient boosting* è, in definitiva, il modello che porta ai migliori risultati nella quasi totalità dei casi considerati, dimostrandosi efficace nell'individuazione della classe rara anche nelle situazioni più problematiche. La versione che è stata sviluppata avendo derivato una nuova funzione di perdita che ottimizza l'AUC si è rivelata ancora più soddisfacente, soprattutto quando lo sbilanciamento è particolarmente marcato. Un comportamento che ha ricevuto conferma anche dall'applicazione ai due insiemi di dati reali. A fronte di queste ottime prestazioni, l'algoritmo potrebbe essere ulteriormente ottimizzato da un punto di vista computazionale, per alleggerirne i tempi di stima rispetto a dati con una numerosità campionaria elevata, un limite che dipende, come si è visto, dalla particolare forma della funzione di perdita. Anche il nuovo albero di classificazione basato sull'AUC risulta più efficace di un comune albero nei casi di maggior sbilanciamento fra le classi. È tuttavia emerso un *trend* negativo all'aumentare della dimensione campionaria più forte che con l'albero usuale, che lascia supporre un certo margine di miglioramento, suggerendo, in particolare, di tentare con un'ottimizzazione simultanea della variabile per lo *split* e del punto di *split* stesso.

Appendice A
Codice R

Il seguente codice permette di stimare l'albero di classificazione modificato tramite l'AUC (paragrafo 3.2). Le funzioni presentate sono, in ordine, le seguenti:

- mini.tree, richiede in input X, il *data frame* contenente le variabili esplicative, e y, la variabile risposta, di tipo *factor* con valori $\{0, 1\}$. La funzione seleziona l'esplicativa con il maggiore AUC rispetto ad y e individua su quella il punto di *split* ottimale, considerando l'AUC come descritto nel paragrafo di riferimento.
- cond, richiede in input *data*, un *data frame* che contiene sia le esplicative che la variabile risposta, e *minsplit*, il numero minimo di osservazioni da considerare ad ogni nodo per provare lo *split*. La funziona stabilisce se un nodo è una foglia controllando innanzitutto il vincolo imposto da *minsplit* e verificando poi se il nodo è puro o, in caso contrario, se su quei dati tutte le variabili hanno un valore dell'AUC con y inferiore o uguale a 0.5.
- tree.auc.list, richiede in input X, y e *minsplit* come già definiti. La funzione stima l'albero richiamando in modo ricorsivo mini.tree sui nodi che non sono foglie secondo le condizioni espresse in cond. Restituisce l'albero nella forma di una lista annidata.
- LinearizedNestedList è stata importata da *GitHub*[2]. La funzione implementa un algoritmo ricorsivo per linearizzare gli elementi di una lista annidata. Riceve in input la lista *NList*, *NameSep*, per separare i livelli della lista, e altri parametri per cui si rimanda alla documentazione fornita dall'autore, non essendo necessari in questo contesto. La lista restituita in output non è più annidata,

[2] https://gist.github.com/akhilsbehl/5990864

ma ogni elemento è un elenco dei numeri dei precedenti livelli, separati da *NameSep*, in modo da tenere memoria della gerarchia originaria.

- nomi.nodi, richiede in input X, y e *minsplit* come già definiti. La funzione assegna un numero ai nodi dell'albero, assumendo che la radice valga 1 e che, in un albero completamente cresciuto, i successivi numeri siano assegnati per riga da sinistra a destra. Per ottenere questa numerazione, la funzione parte dall'albero in forma di lista e la linearizza con LinearizedNestedList.
- tree.auc.data, richiede in input X, y e *minsplit* come già definiti. La funzione stima l'albero in modo analogo a tree.auc.list, ma restituisce i risultati nella forma di un *data frame*.
- AUCtree, richiede in input X, y, *minsplit*, come già definiti, e *profondità*, il livello desiderato di profondità massima dell'albero, dove con 1 si indica uno *stump*. Per *default*, *minsplit* e *profondità* sono pari, rispettivamente, a 2 e 30, per lasciare che l'albero cresca senza vincoli. La funzione richiama tutte le precedenti funzioni e restituisce un oggetto della classe tree, con un output della stessa forma.

Le previsioni si calcolano con la funzione predict del pacchetto tree (Therneau *et al.*, 2017), essendo il modello stimato della relativa classe. Anche il grafico si può ottenere con lo stesso espediente, specificando l'opzione *uniform* per *type*, poiché quella di *default* prevede di far crescere i rami proporzionalmente alla decrescita in impurità.

```
## Funzione mini.tree() ##

mini.tree <- function(X, y) {
  require(ROCR)
  require(dplyr)
  colnames(X) <- paste("x", 1:ncol(X), sep = "")
  data <- data.frame(X, y)
  # Scelta della variabile per lo split:
  auc.variabili <- apply(X, 2,
                         function(x) performance(prediction(x, y),
                                                 measure="auc")@y.values[[1]])
  nome.x <- attributes(auc.variabili)$names[which.max(auc.variabili)]
  # Griglia di possibili valori per lo split (sulla variabile ordinata):
  griglia <- sort(unique(X[, nome.x]))[-length(unique(X[, nome.x]))] +
    diff(sort(unique(X[, nome.x])))/2
  xy <- data.frame(x = X[, nome.x], y)
  # Regola generale: condizione sempre < (a sx T, a dx F)
  f.griglia <- function(griglia) {
    xy.sx <- xy[xy$x < griglia, ]
    xy.dx <- xy[xy$x > griglia, ]
```

```
    xy.sx$yhat <- 1
    xy.dx$yhat <- 0
    unione <- bind_rows(xy.sx, xy.dx)
    tpr <- table(unione$yhat, unione$y)[2, 2]/
      sum(table(unione$yhat, unione$y)[, 2])
    fpr <- table(unione$yhat, unione$y)[2, 1]/
      sum(table(unione$yhat, unione$y)[, 1])
    auc1 <- (tpr - fpr + 1)/2
    auc2 <- (1 - auc1)
    if (auc1 > auc2)  { auc <- auc1 }
    if (auc1 <= auc2) { auc <- auc2 }
    return(auc)
  }
  auc.all <- unlist(lapply(griglia, f.griglia))
  auc.var <- as.numeric(auc.variabili[which.max(auc.variabili)])
  valore.split <- griglia[which.max(auc.all)]
  auc.split <- auc.all[which.max(auc.all)]
  data.s <- data[data[, nome.x] < valore.split, ] # Foglia a sx
  data.d <- data[data[, nome.x] > valore.split, ] # Foglia a dx
  return(list(nome.var = nome.x,
              auc.var = auc.var,
              valore.split = valore.split,
              auc.split = auc.split,
              data.s = data.s,
              data.d = data.d,
              prop.s = sum(data.s$y == 1)/nrow(data.s),
              prop.d = sum(data.d$y == 1)/nrow(data.d)))
}


## Funzione cond() ##

cond <- function(data, minsplit) {
  if (nrow(data) < minsplit)
    esito1 <- esito2 <- T
  if (nrow(data) >= minsplit) {
    if (length(unique(data$y)) == 1) {
      esito1 <- esito2 <- T # Il nodo è puro
    }
    else if (length(unique(data$y)) == 2) {
      esito1 <- F
    }
    if (esito1 == F) {
      X <- subset(data, select = -y)
      auc.variabili <- apply(X, 2,
                             function(x) performance(prediction(x, data$y),
                                                     measure = "auc")@y.values[[1]])
      if (sum(auc.variabili <= 0.5) == length(auc.variabili))  {
        esito2 <- T
      }
      else if (sum(auc.variabili > 0.5) >= 1) {
```

```
        esito2 <- F
      }
    }
  }
  if ((esito1 == T) | (esito1 == F & esito2 == T)) {
    foglia <- T
  }
  else if (esito1 == F & esito2 == F) {
    foglia <- F
  }
  return(foglia)
}


## Funzione tree.auc.list() ##

tree.auc.list <- function(X, y, minsplit) {
  part <- mini.tree(X, y)
  data.s <- part$data.s
  data.d <- part$data.d
  punto.split <- paste(part$nome.var, round(part$valore.split, 4), sep = "<")
  if (cond(data.s, minsplit) == FALSE) {
    foglia.s <- tree.auc.list(X = subset(data.s, select = -y),
                              y = data.s$y, minsplit)
  } else {
    foglia.s <- list(c(sum(data.s == 1)/nrow(data.s)))
  }
  if (cond(data.d, minsplit) == FALSE) {
    foglia.d <- tree.auc.list(X = subset(data.d, select = -y),
                              y = data.d$y, minsplit)
  } else {
    foglia.d <- list(c(sum(data.d == 1)/nrow(data.d)))
  }
  output <- list(punto.split, foglia.s, foglia.d)
}


## Funzione LinearizeNestedList() ##

LinearizeNestedList <- function(NList, LinearizeDataFrames = FALSE,
                                NameSep = "/", ForceNames = FALSE) {
  stopifnot(is.character(NameSep), length(NameSep) == 1)
  stopifnot(is.logical(LinearizeDataFrames), length(LinearizeDataFrames) == 1)
  stopifnot(is.logical(ForceNames), length(ForceNames) == 1)
  if (! is.list(NList)) return(NList)
  if (is.null(names(NList)) | ForceNames == TRUE)
    names(NList) <- as.character(1:length(NList))
  if (is.data.frame(NList) & LinearizeDataFrames == FALSE)
    return(NList)
  if (is.data.frame(NList) & LinearizeDataFrames == TRUE)
    return(as.list(NList))
```

```
  A <- 1
  B <- length(NList)
  while (A <= B) {
    Element <- NList[[A]]
    EName <- names(NList)[A]
    if (is.list(Element)) {
      if (A == 1) {
        Before <- NULL
      } else {
        Before <- NList[1:(A - 1)]
        }
      if (A==B) {
        After <- NULL
      } else {
        After <- NList[(A + 1):B]
        }
      if (is.data.frame(Element)) {
        if (LinearizeDataFrames == TRUE) {
          Jump <- length(Element)
          NList[[A]] <- NULL
          if (is.null(names(Element)) | ForceNames == TRUE)
            names(Element) <- as.character(1:length(Element))
          Element <- as.list(Element)
          names(Element) <- paste(EName, names(Element), sep = NameSep)
          NList <- c(Before, Element, After)
        }
        Jump <- 1
      } else {
        NList[[A]] <- NULL
        if (is.null(names(Element)) | ForceNames == TRUE)
          names(Element) <- as.character(1:length(Element))
        Element <- LinearizeNestedList(Element, LinearizeDataFrames, NameSep,
                                       ForceNames)
        names(Element) <- paste(EName, names(Element), sep = NameSep)
        Jump <- length(Element)
        NList <- c(Before, Element, After)
        }
    } else {
      Jump <- 1
      }
    A <- A + Jump
    B <- length(NList)
  }
  return(NList)
}
```

```
## Funzione nomi.nodi() ##

nomi.nodi <- function(X, y, minsplit) {
  alb.list <- tree.auc.list(X, y, minsplit)
  alb <- LinearizeNestedList(alb.list, NameSep = "")
  depth <- function(this , thisdepth = 0) {
    if(! is.list(this)) {
      return(thisdepth)
    } else {
      return(max(unlist(lapply(this, depth, thisdepth = thisdepth+1))))
    }
  }
  prof <- depth(alb.list) - 1
  if (prof == 1) {
    code <- c("1", "21", "31")
  }
  if (prof > 1) {
    v <- vector("list", prof)
    v[1] <- list(c("21", "31"))
    for (i in 2:prof) {
      v[i] <- list(paste(rep(substring(unlist(v[i - 1]), 1, i - 1), each = 2),
                         unlist(v[1]), sep = ""))
    }
    code <- c("1", unlist(v))
  }
  return(match(names(alb), code))
}


## Funzione tree.auc.data() ##

tree.auc.data <- function(X, y, minsplit) {
  part <- mini.tree(X, y)
  data.s <- part$data.s
  data.d <- part$data.d
  punto.split <- paste(part$nome.var, round(part$valore.split, 4), sep = "<")
  auc.split <- part$auc.split
  n.split <- nrow(data.s) + nrow(data.d)
  prop1.split <- (sum(data.s == 1) + sum(data.d == 1))/n.split
  if (cond(data.s, minsplit) == FALSE) {
    foglia.s <- tree.auc.data(X = subset(data.s, select = -y),
                              y = data.s$y, minsplit)
  } else {
    foglia.s <- data.frame(foglia.s = c(sum(data.s == 1)/nrow(data.s)),
                           nfoglia.s = nrow(data.s))
  }
  if (cond(data.d, minsplit) == FALSE) {
    foglia.d <- tree.auc.data(X = subset(data.d, select = -y),
                              y = data.d$y, minsplit)
  } else {
    foglia.d <- data.frame(foglia.d = c(sum(data.d == 1)/nrow(data.d)),
```

```
                              nfoglia.d = nrow(data.d)) }
  res.data <<- bind_rows(data.frame(punto.split, n.split, auc.split,
                                    prop1.split), foglia.s, foglia.d)
}


## Funzione AUCtree() ##

AUCtree <- function(X, y, minsplit = 2, profondita = 30) {
  options(warn = -1)
  require(tree)
  colnames(X) <- paste("x", 1:ncol(X), sep = "")
  data <- data.frame(X, y)
  data <- tree.auc.data(X, y, minsplit)
  # Riproduco lo stesso data.frame del pacchetto "tree":
  data$punto.split <- as.character(data$punto.split)
  var <- data$punto.split
  var[! is.na(var)] <- unlist(lapply(strsplit(var[! is.na(var)],
                                              '<', fixed = TRUE), '[', 1))
  var[is.na(var)] <- "<leaf>"
  n <- rowSums(data[c("n.split", "nfoglia.s", "nfoglia.d")], na.rm = TRUE)
  dev <- data$auc.split
  dev[is.na(dev)] <- 0
  prop.1 <- rowSums(data[c("prop1.split", "foglia.s", "foglia.d")], na.rm = TRUE)
  prop.0 <- 1 - prop.1
  yprob <- cbind(prop.0, prop.1)
  yval <- ifelse(prop.1 > 0.5, 1, 0)
  yval <- as.factor(yval)
  val.split <- unlist(lapply(strsplit(data$punto.split[! is.na(data$punto.split)],
                                      '<', fixed = TRUE), '[', 2))
  cutleft <- rep("", nrow(data))
  cutleft[! is.na(data$punto.split)] <- paste("<", val.split, sep = "")
  cutright <- rep("", nrow(data))
  cutright[! is.na(data$punto.split)] <- paste(">", val.split, sep = "")
  splits <- cbind(cutleft, cutright)
  frame <- data.frame(var, n, dev, yval)[1L:nrow(data), ]
  frame$splits <- array(c(cutleft, cutright), c(nrow(data), 2),
                        list(character(0L),
                             c("cutleft", "cutright")))[1L:nrow(data), ,
                                                        drop = FALSE]
  class(frame$yval) <- class(y)
  frame$yprob <- array(c(prop.0, prop.1), c(nrow(data), 2),
                       list(character(0L), c("0", "1")))[1L:nrow(data), ,
                                                         drop = FALSE]
  row.names(frame) <- nomi.nodi(X, y, minsplit)

  # Potatura sulla base della profondità indicata:
  frame <- frame[as.numeric(rownames(frame)) <= (2^(profondita + 1)) - 1, ]
  frame$var[(as.numeric(rownames(frame)) >= (2^profondita)) &
            (as.numeric(rownames(frame)) <= (2^(profondita + 1)) - 1)] <- "<leaf>"
  frame$dev[(as.numeric(rownames(frame)) >= (2^profondita)) &
```

```
            (as.numeric(rownames(frame)) <= (2^(profondita + 1)) - 1)] <- 0
  frame$splits[(as.numeric(rownames(frame)) >= (2^profondita)) &
                 (as.numeric(rownames(frame)) <= (2^(profondita + 1)) - 1), ] <- c(""
                     , "")
  # Oggetto della classe "tree":
  f <- as.formula(paste("y~", paste(names(X), collapse = "+")))
  albero.tree <- tree(f, data = data.frame(X, y))
  obj <- list(frame = frame,
              call = albero.tree$call,
              terms = albero.tree$terms,
              weights = albero.tree$weights,
              y = y)
  attr(obj, "ylevels") <- c("0", "1")
  attr(obj, "xlevels") <- attr(albero.tree, "xlevels")
  class(obj) <- "tree"
  return(obj)
}
```

Il codice seguente include due funzioni relative al *gradient boosting* con funzione di perdita basata sull'AUC (paragrafo 3.3). La prima, grad.auc, consente di stimare il modello. Richiede in input X, la matrice o il *data frame* delle variabili esplicative, la variabile risposta y, in formato *numeric* e con valori $\{-1, 1\}$, il numero massimo di iterazioni $Tmax$, il tasso di apprendimento *shrinkage*, e il parametro τ, *tau*, per la funzione di perdita. Per *default*, si considerano *shrinkage*=0.1 e *tau*=0.7. L'esponente p è invece fissato pari a 2, per ricavare le derivate di conseguenza.

La funzione pred.grad calcola le previsioni per una matrice di dati *newdata* e il modello stimato *object*. La funzione restituisce un valore continuo in $[0, 1]$ che rappresenta il grado di confidenza con cui ogni osservazione appartiene alla classe positiva.

```
## Funzione grad.auc() ##

grad.auc <- function(X, y, Tmax, shrinkage = 0.1, tau = 0.7) {
  # Modello di base:
  stump <- function(X, y, g, h) {
    cut.points <- apply(X, 2, function(x) list(sort(unique(x))[-length(unique(x))]
                                               + diff(sort(unique(x)))/2))
    objectives <- vector("list", dim(X)[2])
    for (j in 1:dim(X)[2]) {
      cut.singlevar <- unlist(cut.points[[j]])
      for (i in 1:length(cut.singlevar)) {
        split.point <- cut.singlevar[i]
        G1 <- sum(g[X[, j] < split.point]) # Nodo a sx
        H1 <- sum(h[X[, j] < split.point])
        G2 <- sum(g[X[, j] > split.point]) # Nodo a dx
```

```
      H2 <- sum(h[X[, j] > split.point])
      objectives[[j]][i] <- -((G1^2)/H1 + (G2^2)/H2)/2
      objectives[[j]][i][is.na(objectives[[j]][i])] <- 0
    }
  }
  x.sel <- paste("x", which.min(rapply(objectives, min)), sep = "")
  min.ind <- rapply(objectives, which.min)[which.min(rapply(objectives, min))]
  min.value <- unlist(cut.points[[x.sel]])[min.ind]
  G1 <- sum(g[X[, x.sel] < min.value])
  H1 <- sum(h[X[, x.sel] < min.value])
  G2 <- sum(g[X[, x.sel] > min.value])
  H2 <- sum(h[X[, x.sel] > min.value])
  y.hat <- ifelse(X[, x.sel] < min.value, -G1/H1, -G2/H2)
  return(result = list(y.hat = y.hat, x.sel = x.sel, min.value = min.value,
                       sc.sx = -G1/H1, sc.dx = -G2/H2))
}
final.result <- data.frame(Var = 0, Split = 0, Score.sx = 0, Score.dx = 0)
colnames(X) <- paste("x", 1:ncol(X), sep = "")
Tp <- sum(y == 1)
Tn <- sum(y == -1)
f.hat <- rep(0, length(y)) # Previsione inziale, costante = 0
for (t in 1:Tmax) {
  g <- rep(0, length(y))
  h <- rep(0, length(y))
  for (i in 2:length(y)) {
    sum1 <- 0 # Per g_i
    sum2 <- 0 # Per h_i
    j <- 1
    if (y[i] == 1) {
      for (j in 1:(i-1)) {
        sum1 <- sum1 + ifelse(y[j] == -1, ifelse((f.hat[i] - f.hat[j]) < tau,
                                                 2*(f.hat[i] - f.hat[j] - tau),
                                                 0), 0)/(Tp*Tn)
        sum2 <- sum2 + ifelse(y[j] == -1, ifelse((f.hat[i] - f.hat[j]) < tau,
                                                 2, 0), 0)/(Tp*Tn)
      }
    }
    if (y[i] == -1) {
      for (j in 1:(i-1)) {
        sum1 <- sum1 + ifelse(y[j] == 1, ifelse((f.hat[j] - f.hat[i]) < tau,
                                                -2*(f.hat[j] - f.hat[i] - tau),
                                                0), 0)/(Tp*Tn)
        sum2 <- sum2 + ifelse(y[j] == 1, ifelse((f.hat[j] - f.hat[i]) < tau,
                                                2, 0), 0)/(Tp*Tn)
      }
    }
    g[i] <- sum1
    h[i] <- sum2
  }
  f <- stump(X, y, g, h)
  f.hat <- f.hat + shrinkage*f$y.hat
```

```
    # Per le previsioni su nuovi dati:
    final.result[t, ] <- c(f$x.sel, f$min.value, f$sc.sx, f$sc.dx)
  }
  final.result[, 2] <- as.numeric(final.result[, 2])
  final.result[, 3] <- as.numeric(final.result[, 3])
  final.result[, 4] <- as.numeric(final.result[, 4])
  output <- list(f.hat = f.hat, final.result = final.result, shrinkage = shrinkage)
  return(output)
}


## Funzione pred.grad() ##

pred.grad <- function(object, newdata) {
  var <- object$final.result[, 1]
  split <- object$final.result[, 2]
  sum <- 0
  for (m in 1:length(var)) {
    y.hat <- rep(object$final.result[m, 3], dim(newdata)[1])
    y.hat[newdata[, var[m]] > split[m]] <- object$final.result[m, 4]
    sum <- sum + (object$shrinkage)*y.hat
  }
  pred <- (sum - min(sum))/(max(sum) - min(sum))
  # Nota: appartiene a [0,1] ma non è la stima di una probabilità, è solo un
  # punteggio che esprime il grado di confidenza con cui l'osservazione i-esima
  # appartiene alla classe positiva, che varia nel range [0,1]
  # (1 <=> massima confidenza)
  return(pred)
}
```

Nel codice seguente sono presentate le funzioni per generare i dati dello studio di simulazione (paragrafo 4.2.1). Seguendo l'ordine con cui sono stati introdotti, le funzioni si chiamano disg, sfere e par. Richiedono in input la numerosità campionaria n, la percentuale di unità per la classe rara, *perc*, espressa come un numero in $(0, 100)$, e la dimensione *dim* dei dati, ossia il numero di variabili esplicative. Ognuna restituisce un *data frame* in cui la variabile risposta, indicata con y, è di tipo *factor*, con valori $\{0, 1\}$.

```
## Fuzione disg() ##

disg <- function(n, perc, dim) {
  sizes <- function(n, k = 5) {
    sizes <- c()
    for (i in 1:k) {
      first <- 1+(((i-1)*n)%/%k)
      last <- ((i*n)%/%k)
      sizes <- append(sizes, last-first+1)
```

```
    }
    return(sizes)
  }
  n.min <- round(perc*n/100)
  n.disg <- sample(sizes(n = n.min), replace = F)
  data <- data.frame(matrix(runif(5*n*dim, 0, 1), ncol = dim), y = rep(0, 5*n))
  names(data) <- c(paste("x", 1:dim, sep = ""), "y")
  # Zona con sole unità della classe rara:
  data$y[(data$x1 > 0.35 & data$x1 < 0.65) &
           (data$x2 > 0.15 & data$x2 < 0.25)] <- 1
  data$y[(data$x1 > 0.35 & data$x1 < 0.65) &
           (data$x2 > 0.30 & data$x2 < 0.40)] <- 2
  data$y[(data$x1 > 0.35 & data$x1 < 0.65) &
           (data$x2 > 0.45 & data$x2 < 0.55)] <- 3
  data$y[(data$x1 > 0.35 & data$x1 < 0.65) &
           (data$x2 > 0.60 & data$x2 < 0.70)] <- 4
  data$y[(data$x1 > 0.35 & data$x1 < 0.65) &
           (data$x2 > 0.75 & data$x2 < 0.85)] <- 5
  # Zona di sovrapposizione:
  data$y[((data$x1 > 0.25 & data$x1 <= 0.35) | (data$x1 >= 0.65 & data$x1 < 0.75))
         & (data$x2 > 0.15 & data$x2 < 0.25)] <- 11
  data$y[((data$x1 > 0.25 & data$x1 <= 0.35) | (data$x1 >= 0.65 & data$x1 < 0.75))
         & (data$x2 > 0.30 & data$x2 < 0.40)] <- 22
  data$y[((data$x1 > 0.25 & data$x1 <= 0.35) | (data$x1 >= 0.65 & data$x1 < 0.75))
         & (data$x2 > 0.45 & data$x2 < 0.55)] <- 33
  data$y[((data$x1 > 0.25 & data$x1 <= 0.35) | (data$x1 >= 0.65 & data$x1 < 0.75))
         & (data$x2 > 0.60 & data$x2 < 0.70)] <- 44
  data$y[((data$x1 > 0.25 & data$x1 <= 0.35) | (data$x1 >= 0.65 & data$x1 < 0.75))
         & (data$x2 > 0.75 & data$x2 < 0.85)] <- 55
  # Creo la classe maggiore:
  data2 <- data[sample(which(data$y == 0 | data$y == 11 | data$y == 22 |
                               data$y == 33 | data$y == 44 | data$y == 55),
                       n-n.min, replace = F), ]
  data2$y[data2$y > 0] <- 0
  # Creo la classe rara:
  data2 <- rbind(data2, data[sample(which(data$y == 1 | data$y == 11),
                                    n.disg[1], replace = F), ])
  data2 <- rbind(data2, data[sample(which(data$y == 2 | data$y == 22),
                                    n.disg[2], replace = F), ])
  data2 <- rbind(data2, data[sample(which(data$y == 3 | data$y == 33),
                                    n.disg[3], replace = F), ])
  data2 <- rbind(data2, data[sample(which(data$y == 4 | data$y == 44),
                                    n.disg[4], replace = F), ])
  data2 <- rbind(data2, data[sample(which(data$y == 5 | data$y == 55),
                                    n.disg[5], replace = F), ])
  data2$y[data2$y > 0] <- 1
  data2$y <- factor(data2$y)
  return(data2[sample(nrow(data2), replace = FALSE), ])
}
```

```
## Funzione sfere() ##

sfere <- function(n, perc, dim) {
  n.min <- round(perc*n/100)
  data <- data.frame(matrix(rnorm(10*n*dim, 0, 1), ncol = dim), y = rep(0, 10*n))
  names(data) <- c(paste("x", 1:dim, sep = ""),"y")
  r <- apply((data[, 1:dim]^2), 1, sum)
  data2 <- data[sample(which(r <= qchisq(0.25, dim)), n.min, replace = F), ]
  data2$y <- 1
  data2 <- rbind(data2, data[sample(which(r > qchisq(0.05, dim)), n-n.min,
                                    replace = F), ])
  data2$y <- as.factor(data2$y)
  return(data2[sample(nrow(data2)), ])
}


## Funzione par() ##

par <- function(n, perc, dim) {
  n.min <- round(perc*n/100)
  data <- data.frame(matrix(rnorm(n*dim, 0, 1), ncol = dim),
                     y = as.factor(c(rep(1, n.min), rep(0, n-n.min))))
  names(data) <- c(paste("x", 1:dim, sep=""), "y")
  parabola <- function(x, a, b, d)
    a*(x-b)^2+d
  data$x1[(n.min+1):n] <- runif(n-n.min, -5, 5)
  data$x2[(n.min+1):n] <- parabola(data$x1[(n.min+1):n], 0.2, 0, 0) +
    runif(n-n.min, -5, 0)
  return(data[sample(nrow(data)), ])
}
```

Il codice che segue è stato sviluppato per stimare il modello *AdaC2* ed ottenerne le previsioni, usando come classificatore di base un albero costruito con rpart. La funzione AdaC2 richiede quindi in input sia l'insieme di stima *train* che l'insieme di verifica *test*. La variabile risposta y nell'insieme di stima si considera di tipo *factor* con valori $\{0, 1\}$. Vanno inoltre specificati una formula, inserendo i nomi della variabile risposta e delle esplicative in *formula*, il numero massimo di iterazioni *Tmax*, il parametro *control*, relativo alle opzioni che possono essere indicate per costruire l'albero con rpart, e il costo *c0* che si intende assegnare alle unità della classe negativa, ricordando che il costo per le unità positive è unitario. Le previsioni sono espresse in termini di punteggi in $[0, 1]$.

```
## Funzione AdaC2() ##

AdaC2 <- function(formula, train, test, Tmax, control, c0) {
```

```
  n <- dim(train)[1]
  w <- rep(1, n)
  w[train$y == 0] <- c0
  costi <- w # Vettore dei costi
  w <- w/sum(w) # I pesi iniziali non sono pari a 1/n ma ai costi normalizzati
  sum.test <- 0
  alpha.all <- rep(0,Tmax)
  for (t in 1:Tmax) {
    w <<- w
    fit.t <- rpart(formula = formula, data = train, weights = w, control = control)
    pr.t <- predict(fit.t, newdata = train, type= "class")
    ind <- as.numeric(train$y != pr.t)
    # Classificazione corretta -> ind = 0
    # Classificazione errata   -> ind = 1
    num <- sum(w[ind == 0]*costi[ind == 0])
    den <- sum(w[ind == 1]*costi[ind == 1])
    if (num > den) {
      alpha<-(log(num/den))/2
    }
    else if (num <= den) {
      alpha<-0
    }
    alpha.all[t] <- alpha
    # Aggiornamento dei pesi:
    w[ind == 0] <- costi[ind == 0]*w[ind == 0]*exp(-alpha)
    w[ind == 1] <- costi[ind == 1]*w[ind == 1]*exp(alpha)
    w <- w/sum(w) # Normalizzazione dei pesi
    pr.test <- predict(fit.t, newdata = test, type = "class")
    sum.test <- sum.test + alpha*(as.numeric(pr.test == 1))
  }
  out<-list(alpha = alpha.all, prev.test = sum.test/sum(alpha.all))
  return(out)
}
```

Bibliografia

Al-Shahib A.; Breitling R.; Gilbert D. (2005). Feature selection and the class imbalance problem in predicting protein function from sequence. *Applied Bioinformatics*, **4**(3), 195–203.

Azzalini A.; Scarpa B. (2012). *Data analysis and data mining: An introduction*. OUP USA.

Barandela R.; Sánchez J. S.; Garcıa V.; Rangel E. (2003). Strategies for learning in class imbalance problems. *Pattern Recognition*, **36**(3), 849–851.

Bradley A. P. (1997). The use of the area under the roc curve in the evaluation of machine learning algorithms. *Pattern recognition*, **30**(7), 1145–1159.

Breiman L.; Friedman J. H.; Olshen R. A.; Stone C. J. (1984a). 347 notation index. In *Classification And Regression Trees*, pp. 1–17. CRC Press.

Breiman L.; Friedman J.; Olshen R.; Stone C. (1984b). Classification and regression trees. monterey, calif., usa: Wadsworth.

Chawla N. V. (2003). C4. 5 and imbalanced data sets: investigating the effect of sampling method, probabilistic estimate, and decision tree structure. In *Proceedings of the ICML*, volume 3, p. 66.

Chawla N. V.; Bowyer K. W.; Hall L. O.; Kegelmeyer W. P. (2002). Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, **16**, 321–357.

Chawla N. V.; Lazarevic A.; Hall L. O.; Bowyer K. W. (2003). Smoteboost: Improving prediction of the minority class in boosting. In *European Conference on Principles of Data Mining and Knowledge Discovery*, pp. 107–119. Springer.

Chen T.; Guestrin C. (2016). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pp. 785–794. ACM.

Cieslak D. A.; Chawla N. V. (2008). Learning decision trees for unbalanced data. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 241–256. Springer.

Culp M.; Johnson K.; Michailidis G. (2016). *ada: The R Package Ada for Stochastic Boosting*. R package version 2.0-5.

Davis J.; Goadrich M. (2006). The relationship between precision-recall and roc curves. In *Proceedings of the 23rd international conference on Machine learning*, pp. 233–240. ACM.

Dheeru D.; Karra Taniskidou E. (2017). UCI machine learning repository.

Díez-Pastor J. F.; Rodríguez J. J.; García-Osorio C. I.; Kuncheva L. I. (2015). Diversity techniques improve the performance of the best imbalance learning ensembles. *Information Sciences*, **325**, 98–117.

Elkan C. (2001). The foundations of cost-sensitive learning. In *International joint conference on artificial intelligence*, volume 17, pp. 973–978. Lawrence Erlbaum Associates Ltd.

Fan W.; Stolfo S. J.; Zhang J.; Chan P. K. (1999). Adacost: misclassification cost-sensitive boosting. In *Icml*, volume 99, pp. 97–105.

Ferri C.; Flach P.; Hernández-Orallo J. (2002). Learning decision trees using the area under the roc curve. In *ICML*, volume 2, pp. 139–146.

Freund Y. (1995). Boosting a weak learning algorithm by majority. *Information and computation*, **121**(2), 256–285.

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

Freund Y.; Schapire R. E. *et al.* (1996). Experiments with a new boosting algorithm. In *Icml*, volume 96, pp. 148–156. Bari, Italy.

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

Friedman J. H. (2001). Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pp. 1189–1232.

Friedman J. H. (2002). Stochastic gradient boosting. *Computational Statistics & Data Analysis*, **38**(4), 367–378.

Galar M.; Fernandez A.; Barrenechea E.; Bustince H.; Herrera F. (2012). A review on ensembles for the class imbalance problem: bagging-, boosting-, and hybrid-based approaches. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, **42**(4), 463–484.

Galar M.; Fernández A.; Barrenechea E.; Herrera F. (2013). Eusboost: Enhancing ensembles for highly imbalanced data-sets by evolutionary undersampling. *Pattern Recognition*, **46**(12), 3460–3471.

García V.; Marqués A. I.; Sánchez J. S. (2012). Improving risk predictions by preprocessing imbalanced credit data. In *International Conference on Neural Information Processing*, pp. 68–75. Springer.

Han H.; Wang W.-Y.; Mao B.-H. (2005). Borderline-smote: a new over-sampling method in imbalanced data sets learning. In *International Conference on Intelligent Computing*, pp. 878–887. Springer.

Hao H.; Chen (2017). *ebmc: Ensemble-Based Methods for Class Imbalance Problem.* R package version 1.0.0.

Hastie T.; Tibshirani R.; Friedman J. (2001). *The elements of statistical learning.* Springer Series in Statistics Springer New York Inc., New York, NY, USA.

He H.; Garcia E. A. (2009). Learning from imbalanced data. *IEEE Transactions on knowledge and data engineering*, **21**(9), 1263–1284.

Hossain M. M.; Hassan M. R.; Bailey J. (2008). Roc-tree: A novel decision tree induction algorithm based on receiver operating characteristics to classify gene expression data. In *Proceedings of the 2008 SIAM International Conference on Data Mining*, pp. 455–465. SIAM.

Hu S.; Liang Y.; Ma L.; He Y. (2009). Msmote: improving classification performance when training data is imbalanced. In *Computer Science and Engineering, 2009. WCSE'09. Second International Workshop on*, volume 2, pp. 13–17. IEEE.

Japkowicz N. (2003). Class imbalances: are we focusing on the right issue. In *Workshop on Learning from Imbalanced Data Sets II*, volume 1723, p. 63.

Japkowicz N.; Stephen S. (2002). The class imbalance problem: A systematic study. *Intelligent data analysis*, **6**(5), 429–449.

Jo T.; Japkowicz N. (2004). Class imbalances versus small disjuncts. *ACM Sigkdd Explorations Newsletter*, **6**(1), 40–49.

Joshi M. V.; Kumar V.; Agarwal R. C. (2001). Evaluating boosting algorithms to classify rare classes: Comparison and improvements. In *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*, pp. 257–264. IEEE.

King G.; Zeng L. (2001). Logistic regression in rare events data. *Political analysis*, **9**(2), 137–163.

Kubat M.; Matwin S. *et al.* (1997). Addressing the curse of imbalanced training sets: one-sided selection. In *ICML*, volume 97, pp. 179–186. Nashville, USA.

Kukar M.; Kononenko I. *et al.* (1998). Cost-sensitive learning with neural networks. In *ECAI*, pp. 445–449.

Landesa-Vázquez I.; Alba-Castro J. L. (2015). Revisiting adaboost for cost-sensitive classification. part ii: Empirical analysis. *arXiv preprint arXiv:1507.04126*.

Liu X.-Y.; Wu J.; Zhou Z.-H. (2009). Exploratory undersampling for class-imbalance learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, **39**(2), 539–550.

López V.; Fernández A.; García S.; Palade V.; Herrera F. (2013). An insight into classification with imbalanced data: Empirical results and current trends on using data intrinsic characteristics. *Information Sciences*, **250**, 113–141.

Mamitsuka H. (2006). Selecting features in microarray classification using roc curves. *Pattern Recognition*, **39**(12), 2393–2404.

Mani I.; Zhang I. (2003). knn approach to unbalanced data distributions: a case study involving information extraction. In *Proceedings of workshop on learning from imbalanced datasets*, volume 126.

Mann H. B.; Whitney D. R. (1947). On a test of whether one of two random variables is stochastically larger than the other. *The annals of mathematical statistics*, pp. 50–60.

Manning C. D.; Schütze H. (1999). *Foundations of statistical natural language processing*. MIT press.

Mazurowski M. A.; Habas P. A.; Zurada J. M.; Lo J. Y.; Baker J. A.; Tourassi G. D. (2008). Training neural network classifiers for medical decision making: The effects of imbalanced datasets on classification performance. *Neural networks*, **21**(2-3), 427–436.

Menardi G.; Torelli N. (2014). Training and assessing classification rules with imbalanced data. *Data Mining and Knowledge Discovery*, **28**(1), 92–122.

Napierała K.; Stefanowski J.; Wilk S. (2010). Learning from imbalanced data in presence of noisy and borderline examples. In *International Conference on Rough Sets and Current Trends in Computing*, pp. 158–167. Springer.

Olszewski D. (2012). A probabilistic approach to fraud detection in telecommunications. *Knowledge-Based Systems*, **26**, 246–258.

Prati R. C.; Batista G. E.; Monard M. C. (2004). Class imbalances versus class overlapping: an analysis of a learning system behavior. In *Mexican international conference on artificial intelligence*, pp. 312–321. Springer.

Provost F.; Fawcett T. (1997). Analysis and visualization of classifier performance: comparison under imprecise class and cost distributions. In *Proceedings of the*

Third International Conference on Knowledge Discovery and Data Mining, pp. 43–48. AAAI Press.

R Core Team (2014). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

Riddle P.; Segal R.; Etzioni O. (1994). Representation design and brute-force induction in a boeing manufacturing domain. *Applied Artificial Intelligence an International Journal*, **8**(1), 125–147.

Ridgeway G. (2017). *gbm: Generalized Boosted Regression Models*. R package version 2.1.3.

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

Schapire R. E.; Freund Y. (2012). *Boosting: Foundations and algorithms*. MIT press.

Seiffert C.; Khoshgoftaar T. M.; Van Hulse J.; Napolitano A. (2008). Rusboost: Improving classification performance when training data is skewed. In *Pattern Recognition, 2008. ICPR 2008. 19th International Conference on*, pp. 1–4. IEEE.

Sing T.; Sander O.; Beerenwinkel N.; Lengauer T. (2005). Rocr: visualizing classifier performance in r. *Bioinformatics*, **21**(20), 7881.

Sun Y.; Kamel M. S.; Wong A. K.; Wang Y. (2007). Cost-sensitive boosting for classification of imbalanced data. *Pattern Recognition*, **40**(12), 3358–3378.

Sun Y.; Wong A. K.; Kamel M. S. (2009). Classification of imbalanced data: A review. *International Journal of Pattern Recognition and Artificial Intelligence*, **23**(04), 687–719.

Therneau T.; Atkinson B.; Ripley B. (2017). *rpart: Recursive Partitioning and Regression Trees*. R package version 4.1-11.

Ting K. M. (2000). A comparative study of cost-sensitive boosting algorithms. In *In Proceedings of the 17th International Conference on Machine Learning*. Citeseer.

Tomek I. (1976). Two modifications of cnn. *IEEE Trans. Systems, Man and Cybernetics*, **6**, 769–772.

Visa S.; Ralescu A. (2005). The effect of imbalanced data class distribution on fuzzy classifiers-experimental study. In *Fuzzy Systems, 2005. FUZZ'05. The 14th IEEE International Conference on*, pp. 749–754. IEEE.

Weiss G. M. (2004). Mining with rarity: a unifying framework. *ACM Sigkdd Explorations Newsletter*, **6**(1), 7–19.

Weiss G. M. (2010). The impact of small disjuncts on classifier learning. In *Data Mining*, pp. 193–226. Springer.

Weiss G. M.; Provost F. (2001). The effect of class distribution on classifier learning: an empirical study. *Rutgers Univ.*

Yan L.; Dodier R. H.; Mozer M.; Wolniewicz R. H. (2003). Optimizing classifier performance via an approximation to the wilcoxon-mann-whitney statistic. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pp. 848–855.

Yang Q.; Wu X. (2006). 10 challenging problems in data mining research. *International Journal of Information Technology & Decision Making*, **5**(04), 597–604.

Yu H.; Ni J.; Dan Y.; Xu S. (2012). Mining and integrating reliable decision rules for imbalanced cancer gene expression data sets. *Tsinghua Science and technology*, **17**(6), 666–673.

Zhao P.; Hoi S. C.; Jin R.; YANG T. (2011). Online auc maximization.