Università degli Studi di Genova

Facoltà di Scienze Matematiche,
Fisiche e Naturali

Corso di studi in
Statistica Matematica e trattamento Informatico dei Dati

Anno accademico 2013/2014

Tesi di Laurea

Metodi di Adaboost Multiclasse

Candidato:
D'Angelo Giacomo

Relatore:
Dott. Fabrizio Malfanti

Correlatore:
Prof.ssa Eva Riccomagno

"Sky's the limit"
The notorious B.I.G.

Indice

Capitolo 1

Introduzione

La tesi di laurea riguarda l'approfondimento delle teorie inerenti al mondo degli algoritmi di boosting: i procedimenti, le peculiarità e le differenze di utilizzo per le principali versioni di Adaboost, avendo io avuto un primo approccio durante lo svolgimento del tirocinio relativo a questi algoritmi.

1.1 Obiettivo

Obiettivo di questa tesi è mostrare l'algoritmo adaboost nella sua versione multi-classe e alcune varianti ed estensioni; infatti esistono diverse versioni per affrontare il problema multivariato ognuna delle quali differisce per una qualche caratteristica che verrà analizzata nei prossimi paragrafi.
Lo scopo del tirocinio svolto è stato quello di costruire un filtro Adaboost per la classificazione di previsioni sportive, in modo tale da ottenere un classificatore che risulti più efficace rispetto a quelli utilizzati per la costruzione del filtro; ovvero attraverso questo algoritmo si vuole combinare un numero finito di bookmakers per la costruzione di uno virtuale. Questo bookmaker virtuale si pone come obiettivo quello di predire correttamente l'esito di più partite rispetto ai bookmakers utilizzati per costruire il filtro.
Lo studio effettuato nella tesi è servito per capire se l'algoritmo Adaboost fosse realmente quello migliore, rispetto a tutti gli altri algoritmi, per il problema preso in considerazione. Si sono quindi analizzati a fondo le caratteristiche dell'Adaboost per verificare se quello scelto inizialmente fosse realmente la versione più efficiente per la risoluzione del problema riscontrato durante il tirocinio.

1.2 I dati

Il dataset considerato si riferisce agli ultimi cinque anni di previsioni sportive inerenti al campionato di calcio tedesco, la Bundesliga e a quello inglese, la Premier League.

I dati sono stati reperiti sul sito *Football-data.uk*, il quale riporta per ogni anno:

- DATA: data in cui è stata giocata la partita
- HOME_TEAM: nome squadra che gioca in casa
- AWAY_TEAM: nome squadra che gioca in trasferta
- RESULT: esito reale della partita
- QUOTAS_H: previsione dell'esito vittoria in casa
- QUOTAS_D: previsione dell'esito pareggio
- QUOTAS_A: previsione dell'esito vittoria in trasferta

Le tre quote sono prese per cinque bookmakers:

- BW: Bet&Win
- IW: Interwitten
- LB: Ladbrokes
- SJ: StanJames
- WH: WilliamHill

Date	HomeTeam	AwayTeam	FTR	BWH	BWD	BWA	IWH	IWD	IWA	LBH	LBD	LBA	WHH	WHD	WHA	SJH	SJD	SJA
07/08/09	Wolfsburg	Stuttgart	H	1,95	3,3	3,5	2	3,3	3,6	2	3,4	3,75	1,95	3,5	3,75	2	3,5	3,6
08/08/09	Dortmund	FC Koln	H	1,65	3,5	4,8	1,65	3,6	5,1	1,67	3,3	4,5	1,62	3,3	5	1,67	3,4	5
08/08/09	Hertha	Hannover	H	1,8	3,4	4	1,8	3,4	4,3	1,73	3,4	4	1,7	3,3	4,2	1,8	3,25	4,33
08/08/09	Hoffenheim	Bayern Munich	D	3,75	3,45	1,85	3,7	3,4	1,9	3,75	3,3	1,8	3,75	3,3	1,8	3,5	3,25	2
08/08/09	Mainz	Leverkusen	D	3,55	3,25	1,95	3,35	3,3	2,1	3,2	3,2	2	3	3,3	2,05	3,4	3,25	2,1
08/08/09	Nurnberg	Schalke 04	A	3,3	3,25	2,05	3,1	3,3	2,2	2,88	3,3	2,1	3,1	3,3	2	3	3,25	2,2
08/08/09	Werder Bremen	Ein Frankfurt	A	1,4	4,15	7	1,45	4	7	1,36	4	7	1,4	3,75	7	1,36	4,5	8
09/08/09	Bochum	M'gladbach	D	2,15	3,3	3,05	2,1	3,3	3,35	2,1	3,3	2,88	2,2	3,2	2,8	2,1	3,25	3,25
09/08/09	Freiburg	Hamburg	D	3,55	3,3	1,95	3,35	3,3	2,1	3,4	3,25	1,91	3,25	3,3	1,95	3,6	3,25	2
15/08/09	Bayern Munich	Werder Bremen	D	1,55	3,8	5,25	1,6	3,6	5,6	1,53	3,5	5,5	1,6	3,4	4,8	1,5	4,33	6
15/08/09	Ein Frankfurt	Nurnberg	D	2	3,3	3,4	2	3,3	3,6	1,91	3,25	3,4	1,95	3,25	3,3	2,05	3,25	3,75
15/08/09	FC Koln	Wolfsburg	A	4,15	3,3	1,8	4	3,4	1,85	3,6	3,3	1,83	3,75	3,3	1,8	4	3,5	1,9
15/08/09	Hamburg	Dortmund	H	2	3,3	3,4	2,1	3,3	3,35	2	3,2	3,2	2	3,2	3,2	2	3,5	3,6
15/08/09	Hannover	Mainz	D	1,75	3,35	4,35	1,75	3,5	4,6	1,8	3,3	3,75	1,83	3,3	3,6	1,91	3,4	4
15/08/09	Leverkusen	Hoffenheim	H	2	3,3	3,4	1,9	3,3	3,9	1,91	3,25	3,4	1,91	3,25	3,4	2,1	3,6	3,25
15/08/09	Stuttgart	Freiburg	H	1,45	3,8	6,8	1,45	4	7	1,4	4	6,5	1,4	4	6,5	1,44	4,33	7,5
16/08/09	M'gladbach	Hertha	H	2,3	3,2	2,85	2,35	3,15	2,95	2,38	3,2	2,6	2,25	3,2	2,75	2,38	3,4	2,88
16/08/09	Schalke 04	Bochum	H	1,45	3,8	6,8	1,4	4,3	7,5	1,4	3,75	7	1,36	4	7	1,44	4,33	7,5
21/08/09	Hoffenheim	Schalke 04	D	2,4	3,2	2,7	2,5	3,2	2,75	2,4	3,3	2,88	2,5	3,3	2,75	2,4	3,33	2,88
22/08/09	Dortmund	Stuttgart	D	2,25	3,2	2,95	2,25	3,2	3,05	2,2	3,2	2,8	2,15	3,25	2,88	2,25	3,4	3,1
22/08/09	FC Koln	Ein Frankfurt	D	2,1	3,25	3,2	2,2	3,2	3,2	2,1	3,2	3	2,1	3,2	3	2,1	3,4	3,5
22/08/09	Freiburg	Leverkusen	A	3,25	3,2	2,1	3,35	3,3	2,1	3	3,2	2,1	3,2	3,2	2	3,2	3,4	2,2
22/08/09	Mainz	Bayern Munich	H	6,8	3,8	1,45	6,5	3,9	1,5	6,5	3,6	1,44	6,5	4	1,4	6,75	3,8	1,53
22/08/09	Nurnberg	Hannover	A	1,9	3,25	3,75	2,15	3,25	3,25	2	3,2	3,2	2	3,2	3,2	1,95	3,4	3,9
23/08/09	Bochum	Hertha	H	2,5	3,2	2,6	2,4	3,2	2,85	2,38	3,2	2,6	2,4	3,1	2,6	2,6	3,4	2,6
23/08/09	Werder Bremen	M'gladbach	H	1,45	3,8	6,8	1,4	4,3	7,5	1,4	3,75	7	1,36	4,2	6,5	1,44	4,5	7
23/08/09	Wolfsburg	Hamburg	A	1,85	3,3	3,9	1,85	3,4	4	1,8	3,3	3,75	1,8	3,4	3,6	1,8	3,6	4,33
28/08/09	M'gladbach	Mainz	H	1,95	3,3	3,55	2,05	3,3	3,4	1,91	3,25	3,4	2	3,4	3,75	2,1	3,4	3,4
29/08/09	Bayern Munich	Wolfsburg	H	1,8	3,4	4	1,8	3,4	4,3	1,8	3,2	4	1,73	3,4	4	1,8	3,75	4,33
29/08/09	Ein Frankfurt	Dortmund	D	2,7	3,2	2,4	2,85	3,2	2,4	2,75	3,2	2,25	2,6	3,25	2,3	3,1	3,4	2,25
29/08/09	Hannover	Hoffenheim	A	2,55	3,2	2,55	2,55	3,3	2,55	2,5	3,2	2,5	2,5	3,1	2,5	2,7	3,25	2,63

Figura 1.1: Esempio file di dati a disposizione

I file a disposizione per il progetto erano nella forma rappresentata in Figura 1.1

1.3 Data description

Studiare i dati che avevamo a disposizione ha portato a capire come sia vasta la casistica nel mondo dei filtri di boosting. Per una migliore comprensione introduciamo i termini "due/multi-classe", ovvero gli stati d'uscita dell'evento, che possono essere di tipo dicotomico o multinomiale e "single/multi-label", ovvero la classificazione del determinato evento, che può essere sempre la stessa o variare. Quest'ultimo concetto è spiegato graficamente attraverso la tabella seguente:

Squadra A	Squadra B	Esito
Juventus	Chievo	1
Juventus	Chievo	1
Juventus	Chievo	1
Juventus	Chievo	1
Juventus	Chievo	X

L'oggetto partita Juventus-Chievo, ogni volta che sarà giocata, avrà la possibilità di essere classificata in maniera diversa, e quindi potrà assumere più di un tipo di label. Ogni combinazione "due/multi-classi" con "single/multi-label" viene affrontata da una versione dell'algoritmo, come vedremo nei prossimi capitoli.

Versione	Classe	Label
Alga Tossica	Due-Classi	Single-label

La tabella appena mostrata indica l'esempio della classificazione sulle alghe, se possono essere o meno dannose all'uomo. Questo viene considerato il caso "standard". Gli stati d'uscita della classificazione sono due: tossica/non tossica, e durante la fase di addestramento la loro etichetta (label) non cambia, il determinato oggetto alga sarà sempre tossica o sana anche se verrà controllato più volte.

Versione	Classe	Label
Tennis	Due-Classi	Multi-label

Il tennis prevede sempre un vincitore, i casi d'uscita sono, quindi, due: vittoria giocatore A o vittoria giocatore B. Il vincitore ad ogni partita disputata tra giocatore A e giocatore B può variare, ovvero l'oggetto partita può essere classificato in entrambi i modi a seconda del risultato della partita, in pratica nel corso delle

partite l'etichetta cambia.

Versione	Classe	Label
Calcio	Multi-Classi	Multi-label

Il calcio, avendo tre risultati possibili per la singola partita, ammette tre uscite: vittoria squadra A (quella che gioca in casa), pareggio e vittoria squadra B (la squadra che disputa la partita in trasferta). Come nel caso precedente, il risultato ogni volta che si giocherà quella specifica partita potrà variare, ecco come il calcio, ma come tutte le classificazioni sportive, sarà di tipo multi-label.

Sebbene ci siano state queste considerazioni, bisogna sottolineare che la distinzione tra etichetta singola e multipla è utile per l'addestramento dei classificatori che poi adaboost utilizzerà per migliorarne la performance; nel nostro caso i bookmakers che predicevano i risultati delle partite erano classificatori già istruiti, quindi il caso di riferimento per il nostro studio è stato comunque single-label. (È nato di conseguenza il problema di dare classificatori istruiti all'algoritmo, risolto nel corso del tirocinio).

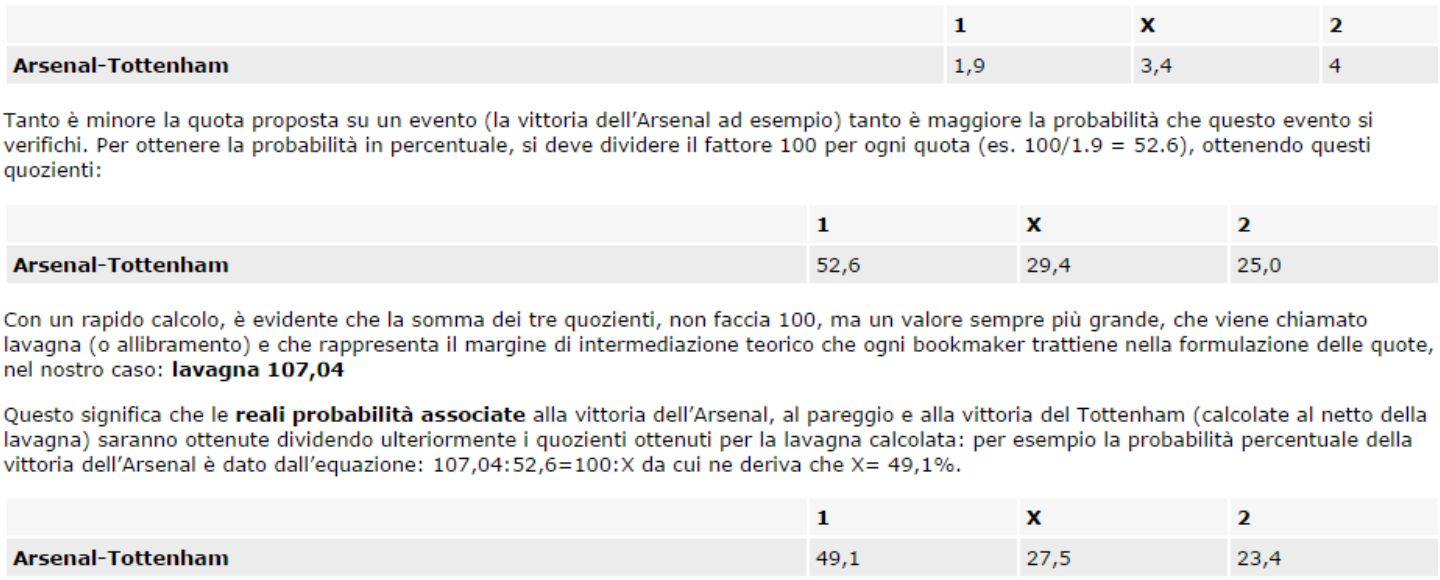
1.4 Data preparation

Considerando un classificatore con le previsioni per singola squadra, si è suddiviso il dataset in 18 file contenenti ciascuno solo le partite di quella squadra. A questo punto ogni file ha subito una ulteriore suddivisione isolando ogni bookmaker, in modo tale da avere cinque file contenenti le informazioni di una squadra con le relative quote della casa.

Si sono costruiti i seguenti file per ogni squadra e bookmakers:

- HISTORY: contenente le probabilità legate ai tre possibili eventi, nome della squadra sfidante, data, se la squadra di riferimento giocava in casa o trasferta e il risultato reale
- TRAIN: uguale al file History
- QUOTAS: contenente le probabilità legate ai tre possibili eventi, nome della squadra sfidante, data.
- TEST: contiene data e squadra avversaria

I primi due file vengono utilizzati dal programma nella fase di addestramento del filtro, rappresentano la storia delle previsioni dei siti e, a differenza di una procedura classica del boosting, sono i classificatori deboli già addestrati.



	1	X	2
Arsenal-Tottenham	1,9	3,4	4

Tanto è minore la quota proposta su un evento (la vittoria dell'Arsenal ad esempio) tanto è maggiore la probabilità che questo evento si verifichi. Per ottenere la probabilità in percentuale, si deve dividere il fattore 100 per ogni quota (es. 100/1.9 = 52.6), ottenendo questi quozienti:

	1	X	2
Arsenal-Tottenham	52,6	29,4	25,0

Con un rapido calcolo, è evidente che la somma dei tre quozienti, non faccia 100, ma un valore sempre più grande, che viene chiamato lavagna (o allibramento) e che rappresenta il margine di intermediazione teorico che ogni bookmaker trattiene nella formulazione delle quote, nel nostro caso: **lavagna 107,04**

Questo significa che le **reali probabilità associate** alla vittoria dell'Arsenal, al pareggio e alla vittoria del Tottenham (calcolate al netto della lavagna) saranno ottenute dividendo ulteriormente i quozienti ottenuti per la lavagna calcolata: per esempio la probabilità percentuale della vittoria dell'Arsenal è dato dall'equazione: 107,04:52,6=100:X da cui ne deriva che X= 49,1%.

	1	X	2
Arsenal-Tottenham	49,1	27,5	23,4

Figura 1.2: Lavagna. In questo esempio di una classica partita di Premier, 7.04 sarà la percentuale che il bookmer x prevede di guadagnare su quella partita. Si noti che questo valore, sebbene oscilli nello stesso intervallo, varia per ogni partita e soprattutto per ogni bookmaker.

I bookmakers sono considerati già addestrati in quanto essi esprimono già una predizione della quale si sa se è giusta o sbagliata; solitamente, invece, l'adaboost costruisce autonomamente diversi classificatori deboli che poi addestra.

I file QUOTAS e TEST sono invece utilizzati per verificare l'efficacia: vengono date al classificatore strong le quotas di partite di cui non conosce l'esito; esso restituirà le sue previsioni in probabilità dei tre eventi.

Le probabilità a cui si fa riferimento sono ricavate dalle quote dei bookmakers; per questa trasformazione è necessario introdurre il concetto di **lavagna** che verrà spiegato anche grazie all'utilizzo della Figura 1.2

1.5 Risultati

È stato quindi applicato l'algoritmo Adaboost, in particolare utilizzando la versione più inerente al problema trattato. Lo scopo era ovviamente non quello di predire tutti i risultati delle partite, poichè alcune giornate calcistiche avranno sempre una netta favorita e, nel caso questa non dovesse vincere, sarà altamente improbabile prevederlo, ma valutare le partite con esito più incerto.
Maggiore attenzione si è posta alle partite con esito più incerto, e che quindi avranno pronostici diversi tra diversi bookmakers. A prova di questo, considerando il classificatore della squadra Schalke 04 e le sue 17 partite utilizzate come test, solo una aveva pronostici diversi e le rimanenti erano predette giuste o sbagliate da tutti i classificatori. Il filtro adaboost che è stato costruito si comportava in maniera analoga ai weaks per quanto riguarda le 16 partite "standard". La partita Schalke 04 vs Stoccarda veniva predetta in modo corretto solo dal bookmaker InterWitten. Il procedimento è stato applicato per ogni squadra del campionato tedesco, in particolare si è consultato il classificatore inerente allo Stoccarda per combinare il risultato di questa partita. Entrambi danno le stesse probabilità sui tre eventi. Il nostro classificatore, pur sbagliando la predizione, commetteva un errore minimo rispetto agli altri quattro weaklearns (circa dell'1%).

Si è cercata quindi una partita predetta giustamente da due classificatori, di cui uno dava però dava l'esito 1 e 2 equiprobabili (Hannover-Schalke 04 del 24/08/2013); è stata rimossa dal file history e inserita nelle quotas. La partita in questione, al contrario del caso precedente, veniva classificata in maniera corretta ovvero predicendo come esito più probabile la vittoria in casa dell'Hannover. In particolare le probabilità restituite dal nostro classificatore erano: 36.9%, 26.9% e 36.2% rispettivamente per gli esiti 1, X e 2,

Avendo a disposizione pochi bookmakers e partite "particolari", ci si ritiene estremamente soddisfatti dei risultati avuti e ottimisti di ottenerne di migliori valutando più campionati e considerando più agenzie di scommesse.

Capitolo 2

Adaboost per classificazione binaria

Adaboost, diminutivo di Adaptive Boosting, è uno tra gli algoritmi di Data Mining più utilizzati. Fu proposto da Yoav Freund e Robert Schapire, i quali hanno vinto il prestigioso "Godel Prize" (premio assegnato ai migliori lavori in computer science) nel 2003 per il loro lavoro.
È molto accurato, semplice e con diverse applicazioni possibili, risulta meno suscettibile al rischio di overfitting rispetto ad altri algoritmi di apprendimento; può, inoltre, essere utilizzato insieme ad altri tipi di algoritmi di apprendimento per migliorare le prestazioni di questi algoritmi. I difetti che presenta questo filtro includono il fatto che risulta sensibile agli outliers e ad i "noisy" data, ovvero ai dati sporchi. Si basa sull'idea di creare una regola di predizione altamente accurata combinandone diverse dette "deboli". Risulta empiricamente che Adaboost riduce significativamente l'errore di qualsiasi classificatore debole.

Si denota con X lo spazio delle istanze mentre con Y quello delle possibili etichette (ovvero i possibili modi in cui l'istanza viene classificata). Si assume $Y = \{-1, +1\}$. Dato un weak learner ("classificatore debole") e un training set $D = (x_1, y_1), (x_2, y_2), .., (x_m, y_m)$, dove $x_i \in X$ e $y_i \in Y$ (i=1,..,m), l'algoritmo Adaboost funziona come segue.

Inizialmente assegna pesi uguali a tutti gli esempi di training (x_i, y_i) con (i=1,..,m). Si denota come D_t, la distribuzione dei pesi al t-esimo passo di apprendimento.

Dal training set D e dalla distribuzione dei pesi D_t, l'algoritmo genera un weak learner $h_t : X \to Y$, richiamando il weak learner di partenza. Successivamente usa gli esempi di training per testare h_t e i pesi degli esempi classificati in modo scorretto vengono incrementati. Quindi, si ottiene una distribuzione dei pesi aggiornata, D_{t+1}.
Dal training set e da D_{t+1}, Adaboost genera un altro classificatore debole richiamando nuovamente il weak learner di partenza. Il processo è così ripetuto per T volte. Il modello finale è la combinazione lineare dei T weak learners in base ai pesi determinati durante il processo di training.

Di seguito si riporta uno schema delle varie fasi:

- Sia $D = (x_1, y_1), (x_2, y_2), .., (x_m, y_m)$
- Si inizializza il vettore dei pesi: $D_1(i) = 1/m$
- Il processo viene effettuato per T iterazioni
- Sia $\mathcal{L}$ weak learner iniziale una funzione di due variabili (D, D_t) a valori in Y

(a) $h_t = \mathcal{L}(D, D_t)$ viene fatto il train di un weak learner h_t da D usando la distribuzione D_t

(b) $\varepsilon_t = Pr_i \sim D_i[h_t(x_i \neq y_i)]$ misura l'errore di h_t

(c) $\alpha_t = \frac{1}{2} \ln \frac{1-\varepsilon_t}{\varepsilon_t}$ determina il peso di h_t

(d) $D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} exp(-\alpha_t) & seh_t(x_i) = y_i \\ exp(\alpha_t) & seh_t(x_i) \neq y_i \end{cases} = \frac{D_t(i) exp(-\alpha_t y_i h_t(x_i))}{Z_t}$

Aggiorna la distribuzione, dove Z_t è un fattore di normalizzazione che permette a D_{t+1} di essere una distribuzione.

L'output sarà:

$$H(x) = sign(\sum_{t=1}^{T} \alpha_t h_t(x))$$

Viene riportato un breve esempio di applicazione dell'algoritmo: abbiamo a disposizione un dataset di 10 elementi, appartenenti a due classi differenti, e 3 classificatori deboli.

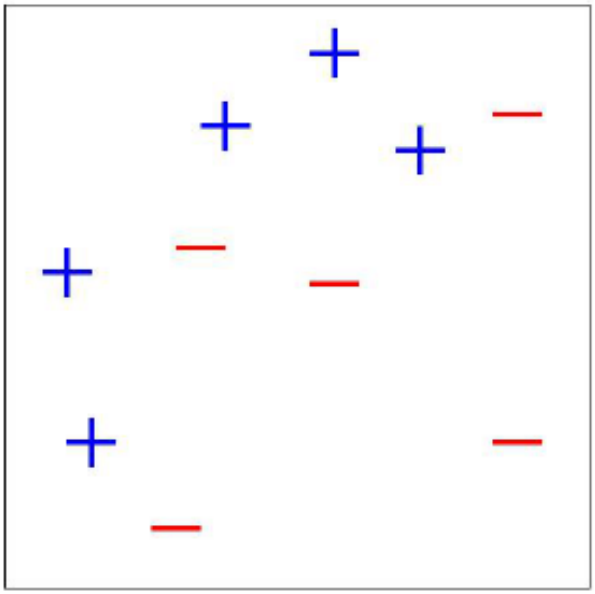


Figura 2.1: *Disposizione dei dati.*

Inizialmente entrambe le popolazioni hanno lo stesso peso. La "frontiera" tra le due popolazioni è evidente ma non è lineare.

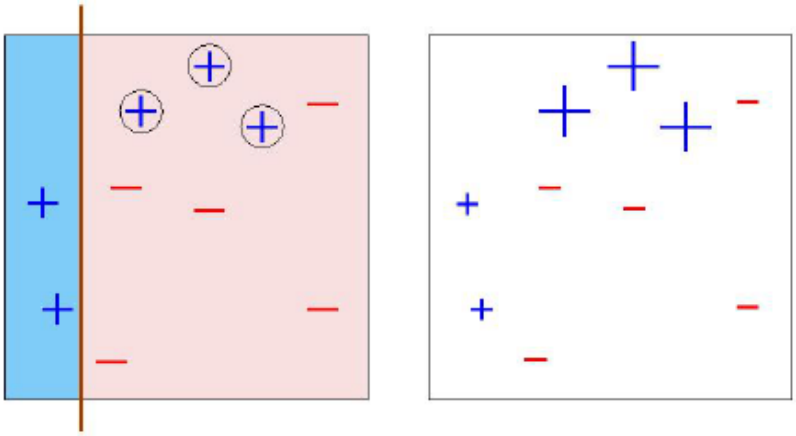


Figura 2.2: *Esito del primo classificatore e conseguente aggiornamento dei pesi.*

Individuato il "primo" classificatore, possiamo notare come classifica correttamente la classe "-", ma commette 3 errori, tali osservazioni verranno pesate maggiormente al passo successivo. Dalle formule per il calcolo dei coefficienti, otteniamo $\alpha_1 = 0.42$.

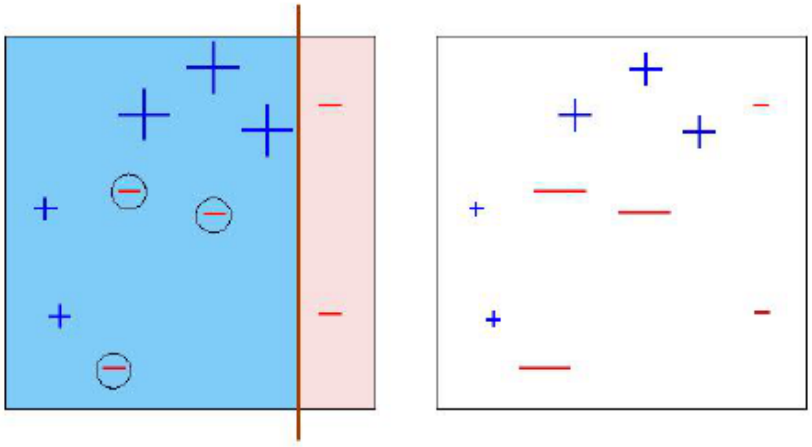


Figura 2.3: *Esito del secondo classificatore e conseguente aggiornamento dei pesi.*

I risultati del secondo classificatore sono analoghi a quelli del primo, è di grande importanza, però, notare come variano i pesi delle osservazioni. Il valore del coefficiente di tale classificatore è $\alpha_2 = 0.66$.

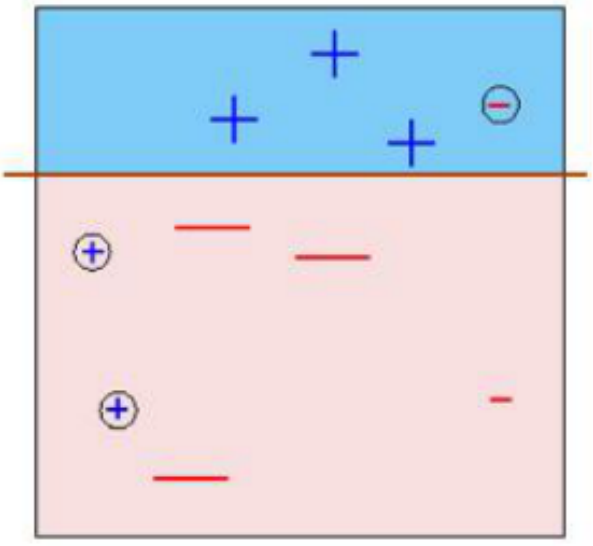


Figura 2.4: *Esito del terzo classificatore.*

Introduciamo il terzo classificatore. Questo commette 3 errori, ma i loro pesi sono nettamente inferiori rispetto agli altri, perchè tali osservazioni avevano già ottenuto un esito corretto dai precedenti classificatori, otteniamo così un valore per il coefficiente $\alpha_3 = 0.91$. Osserviamo i risultati così ottenuti:

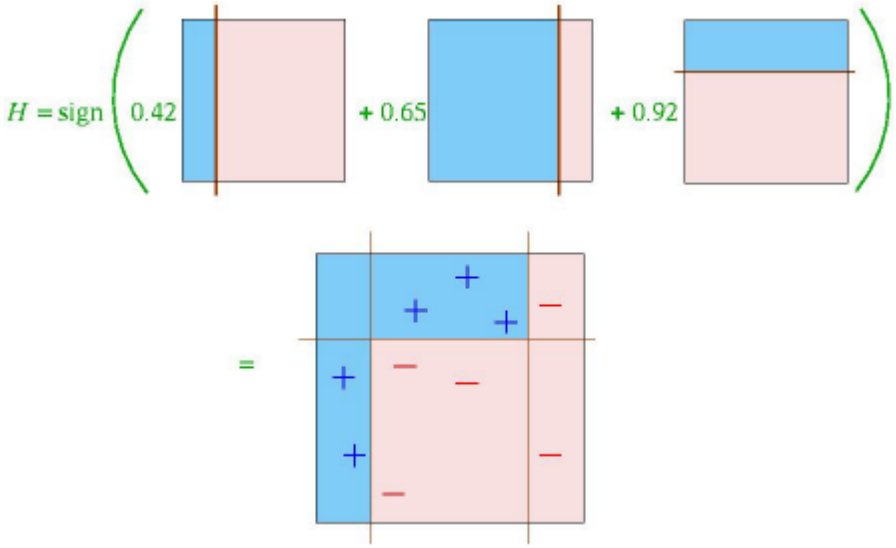


Figura 2.5: *Esito del classificatore finale.*

Capitolo 3

Adaboost Multiclasse

L'algoritmo AdaBoost si può considerare una grande innovazione per risolvere problemi di Data Mining. Questo filtro viene applicato per casi dicotomici, ovvero dove le possibili uscite (i possibili risultati) possono essere due (es. vero/falso, sì/no, 1/0, ecc.). In questo capitolo si analizzeranno le estensioni che permettono di affrontare il caso multivariato ideate dagli stessi creatori del precedente filtro. Il primo chiamato Adaboost.M1, è il ponte necessario per arrivare all'algoritmo che è stato fondamentale per le ricerche svolte nel tirocinio, ovvero Adaboost.M2. Esso infatti parte dalle considerazioni del suo antecedente per migliorarne la struttura matematica e la performance. In questo capitolo non viene preso ancora in considerazione il concetto di multi-label, infatti entrambi gli algoritmi hanno, implicitamente, ancora un approccio single-label.

3.1 Adaboost.M1

È la versione più semplice tra gli adaboost multiclasse. L'algoritmo prende come input un training set di m esempi $D = ((x_1, y_1), ..., (x_m, y_m))$ dove x_i è una istanza disegnata da qualche spazio X e rappresentata in qualche maniera (generalmente un vettore di valori attributi), e $y_i \in Y$ è la classe etichetta associata con x_i. Si assuma che il set di possibili etichette Y sia di cardinalità k, dove k è sempre un numero intero positivo.

Come nel caso dicotomico, l'adaboost ha la possibilità di interagire con un altro algoritmo non specificato (chiamato WeakLearn), il quale verrà chiamato ripetutamente per una serie di iterazioni. Al passo t il booster provvede il WeakLearn con una distribuzione D_t sul training set D. In risposta, il WeakLearn calcola un classificatore o una ipotesi $h_t : X \times Y \rightarrow [0, 1]$ che dovrebbe classificare correttamente una frazione di training set che ha grande probabilità in relazione alla

distribuzione D_t. Lo scopo del weak learner è quello di trovare un'ipotesi h_t che minimizzi l'errore $\varepsilon_t = Pr_i \sim D_t[h_t(x_i \neq y_i)]$. Questo errore viene misurato rispetto alla distribuzione D_t che è stata fornita dal weak learner stesso. Il processo continua per T volte e, per ultimo, il booster combina le ipotesi $h_1, ..., h_T$ in una singola ipotesi finale h_{fin}.

Non sono ancora stati specificati il metodo attraverso il quale D_t è calcolata ad ogni iterazione e come h_{fin} viene calcolata. Esistono differenti modi di agire, Adaboost.M1 si comporta come segue:

- Per una sequenza di m esempi di training $(x_1, y_1), ...(x_m, y_m)$ con etichette $y_i \in Y = \{1, ..., k\}$
- Si inizializza il vettore dei pesi: $D_1(i) = 1/m$
- Il processo viene effettuato per T iterazioni
- Sia $\mathcal{L}$ weak learner iniziale

(a) Viene chiamato il weak learner fornendolo della distribuzione D_t

(b) In risposta, il weak learner genere un'ipotesi $h_t = \mathcal{L}(D, D_t)$

(c) $\varepsilon = \sum_{i:h_t(x_i)\neq y_i} D_t(i)$ si misura l'errore di h_t
Se ε >1/2, allora T=t-1 e il programma abortisce

(d) $\alpha_t = \frac{\varepsilon_t}{1-\varepsilon_t}$ determina il peso di h_t

(e) $D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} \alpha_t & seh_t(x_i) = y_i \\ 1 & altrimenti \end{cases}$

Aggiorna la distribuzione, dove Z_t è un fattore di normalizzazione (scelto in modo tale che D_{t+1} sia una distribuzione).

L'output sarà:

$$h_{fin}(x) = \underset{y \in Y}{\text{argmax}} \sum_{t:h_t(x)=y} log\frac{1}{\alpha_t}$$

La distribuzione iniziale D_1 è scelta uniforme su D, quindi D_1(i)=$1/m$ per tutte le *i = 1, ..., m*. Per calcolare la distribuzione D_{t+1} da D_t e l'ultima ipotesi weak h_t, si moltiplica il peso dell'esempio i per $\alpha_t \in [0, 1)$ se h_t classifica x_i correttamente, altrimenti il peso rimane invariato. I pesi sono poi rinormalizzati dividendo per la costante di normalizzazione Z_t. Con efficacia, esempi "facili" che sono stati classificati correttamente prendono dalle precedenti ipotesi weak un peso minore, mentre esempi "difficili" che tendono ad essere spesso classificati in maniera errata prendono un peso maggiore. Perciò, adaboost focalizza un peso

maggiore sugli esempi che sembrano più difficili per il WeakLearn.

Il valore α_t è calcolato come funzione dell'errore ε secondo il punto (c). L'ipotesi finale h_{fin} è un voto pesato delle ipotesi weak. Per una istanza data x, h_{fin} emette l'etichetta y che massimizza la somma dei pesi delle ipotesi weak predicendo quella specifica etichetta. Il peso delle ipotesi h_t è definito come $\log(\frac{1}{\alpha_t})$, ovvero viene dato peso più grande alle ipotesi con errore più basso.

Il Teorema 1 dimostra l'importanza teorica di questo algoritmo. Esso mostra che se le ipotesi weak in modo compatibile hanno errore solo leggermente migliore di 1/2, allora l'errore di training dell'ipotesi finale h_{fin} si abbassa a zero in maniera esponenziale (per il caso di classificazione binaria, questo significa che le ipotesi weak debbano essere solo lievemente migliori della casualità).

Teorema 1. Si supponga che l'algoritmo weak, ovvero WeakLearn, quando chiamato dall'Adaboost.M1, generi ipotesi con errori $\varepsilon_1, ..., \varepsilon_T$, dove ε è definito come illustrato precedentemente. Si assuma ogni ε <1/2 e si ponga $\gamma_t = 1/2 - \varepsilon_t$. Allora l'ipotesi finale h_{fin} è maggiorata come segue.

$$\frac{|\{i:h_{fin}(x_i)\neq y_i\}|}{m} \leq \prod_{t=1}^{T} \sqrt{1-4\gamma_t^2} \leq exp\left(-2\sum_{t=1}^{T}\gamma_t^2\right).$$

Il Teorema 1. implica che l'errore di training[1] dell'ipotesi finale generato da M1 è piccolo. Questo però non implica necessariamente che l'errore di test[2] sia altrettanto piccolo. In pratica l'errore di test tende ad essere migliore rispetto a quanto suggerisce la teoria, indicando un chiaro difetto in questa visione teorica.

Il principale svantaggio dell'Adaboost.M1 è quello di essere incapace di trattare le weak ipotesi con errore superiore a 1/2. L'errore previsto dall'ipotesi che genera casualmente l'etichetta è 1-1/k. Quindi, per k=2, le ipotesi weak hanno bisogno di essere leggermente migliori della scelta casuale, ma quando k>2 (ovvero tutti i casi multidimensionali, per cui l'M1 è stato creato), la richiesta che l'errore sia meno di 1/2 è abbastanza forte e spesso può essere difficile da conoscere.

[1] L'errore di training o errore empirico è basato sui dati di training, ovvero è quello che si commette durante la fase di apprendimento.

[2] L'errore di test è l'errore di generalizzazione che si stima sui nuovi dati.

3.2 Adaboost.M2

Questa versione tenta di superare la difficoltà trovata nella versione M1 "aumentando la comunicazione" tra l'algoritmo di boosting e il weak learner. Per prima cosa, viene permesso al weak learner di generare più ipotesi dove, piuttosto che identificare una singola etichetta in Y, sceglie un set di "plausibili" etichette. Questo può essere spesso più facile che sceglierne solo una. Per esempio, nello scenario OCR, può esser difficile dire se una particolare immagine è un "7" o un "9", ma è facile eliminare tutte le altre possibilità. In questo caso, piuttosto che decidere tra 7 e 9, il set delle ipotesi può essere (7,9), indicando entrambe le etichette plausibili.

Si permette inoltre al weak learner di dare un grado di plausibiltà, così ogni ipotesi weak darà un vettore $[0,1]^k$, dove le componenti con valori vicini a 0 o a 1 corrispondono a quelle etichette considerate rispettativamente implausibili o plausibili. Questo vettore non è costituito da probabilità, la somma quindi non necessita di eguagliare a 1.

Mentre l'algoritmo weak acquista maggior potenza espressiva, si mette un requisito sulle performance delle ipotesi weak più complesso. Si introduce quindi il concetto di pseudo-loss, che è una misura di errore più sofisticata. Essa, a differenza degli errori che sono calcolati rispetto alle distribuzioni sugli esempi di training, viene calcolata rispetto alla distruzione sull'insieme di tutte le coppie di esempi e etichette sbagliate.

Vedremo che Adaboost.M2 compie il boosting se ogni ipotesi weak ha la pseudoloss leggermente migliore del calcolo randomizzato.

Per formalità si introduca questo concetto, una *mislabel* è una coppia (i,y) dove i è l'indice dell'esempio di training e y è un'etichetta sbagliata associata all'esempio i. Sia B l'insieme di tutte le *mislabel* $B = \{(i,y) : i \in \{1,...,m\}, y \neq y_i\}$. Una distribuzione mislabel è una distribuzione definita su B.

Il procedimento del ciclo Adaboost.M2 è il seguente:

- Per una sequenza di m esempi $(x_1, y_1), ...(x_m, y_m)$
  con etichette $y_i \in Y = \{1, ..., k\}$
- Si inizializza il vettore dei pesi: $D_1(i) = 1/|B| per (i,y) \in B$
- Il processo è svolto per T volte specificando il numero di iterazioni

- Sia $\mathcal{L}$ weak learner iniziale
- $B = \{(i, y) : i \in \{1..m\}, y \neq y_i\}$

Per t =1, 2, ..., T:

(a) Viene chiamato il weak learner, fornendolo di una distribuzione D_t

(b) In risposta, il weak learner genera un'ipotesi $h_t : X \times Y \rightarrow [0, 1]$

(c) Viene calcolata la pseudo-loss di h_t:

$$\varepsilon = \frac{1}{2} \sum_{(i,y) \in B} D_t(i, y)(1 - h_t(x_i, y_i) + h_t(x_i, y)).$$

(d) $\alpha_t = \frac{\varepsilon_t}{1-\varepsilon_t}$ determina il peso di h_t

(e) Si aggiorna D_t:

$$D_{t+1}(i, y) = \frac{D_t(i,y)}{Z_t} \alpha_t^{(1/2)(h_t(x_i,y_i) - h_t(x_i,y))}$$

dove Z_t è un fattore di normalizzazione che rende D_{t+1} una distribuzione.

L'output sarà:

$$h_{fin}(x) = \underset{y \in Y}{\operatorname{argmax}} \sum_{t=1}^{T} log \frac{1}{\alpha_t} h_t(x, y)$$

Per ogni passo t di boosting, Adaboost.M2 fornisce il weak learner con la distribuzione mislabel D_t. In risposta, il weak learner calcola un'ipotesi h_t della forma $h_t : X \times Y \rightarrow [0, 1]$. Non ci sono restrizioni su $\sum_y h_t(x, y)$. In particolare, il vettore predizione non deve essere una distribuzione di probabilità.

Intuitivamente, si interpreta ogni mislabel (i,y) come rappresentazione di una domanda binaria del tipo: "Predici che l'etichetta associata all'esempio x_i sia y_i (ovvero l'etichetta corretta) o y (una tra le etichette sbagliate)?". Con questa interpretazione, il peso $D_t(i, y)$ assegnato a questa mislabel rappresenta l'importanza di distinguere l'etichetta sbagliata y dall'esempio x_i.

Un'ipotesi weak h_t viene poi interpretata nella maniera seguente. Se $h_t(x_i, y_i) = 1$ e $h_t(x_i, y) = 0$ allora $h_t(i, y)$ ha (correttamente) predetto che l'etichetta di x_i sia y_i e non y (h_t ritiene y_i essere "plausibile" e y "non plausibile". In maniera analoga, $h_t(x_i, y_i) = 0$ e $h_t(x_i, y) = 1$ allora $h_t(i, y)$ ha fatto la predizione sbagliata. Se $h_t(x_i, y_i) = h_t(x_i, y)$ allora la predizione di h_t è fatta arbitrariamente.

Questa interpretazione porta a definire la psudo-loss dell'ipotesi h_t rispetto alla distribuzione mislabel D_t come:

$$\varepsilon = \frac{1}{2}\sum_{(i,y)\in B} D_t(i,y)(1 - h_t(x_i, y_i) + h_t(x_i, y)).$$

Può essere verificato che la pseudo-loss è minimizzata quando le etichette corrette y_i sono assegnate al valore 1 e le etichette scorrette $y \neq y_i$ al valore 0.

Lo scopo del weak learner è quello di trovare un'ipotesi h_t con una pseudo-loss piccola. Perciò, gli algoritmi "standard" è possibile che abbiano bisogno di qualche modifica per essere usati in questa maniera. Dopo avere ricevuto h_t, la distribuzione mislabel è aggiornata usando una regola simile a quella utilizzata nell'Adaboost.M1. L'ipotesi finale h_{fin} restituisce come output, per un'istanza data x, l'etichetta y che massimizza la media pesata delle ipotesi weak $h_t(x, y)$.

Il seguente teorema pone un limite superiore all'errore di training dell'ipotesi finale. Si noti che questo teorema necessita solo che le ipotesi weak abbiano la pseudo-loss minore di 1/2, senza considerare il numero delle classi. Nonostante le ipotesi weak siano state stimate rispetto alla pseudo-loss, si è certamente stimata l'ipotesi finale utilizzando l'errore di misura ordinario.

Teorema 2. Si supponga un classificatore debole WeakLearn, che quando viene chiamato dall'Adaboost.M2, generi ipotesi con la pseudo-loss $\varepsilon_1, ..., \varepsilon_T$, dove ε è definita come nello schema precedente. Sia $\gamma_t = \frac{1}{2} - \varepsilon_t$. Allora il margine superiore dell'ipotesi finale h_{fin}:

$$\frac{|\{i:h_{fin}(x_i)\neq y_i\}|}{m} \leq (k-1)\prod_{t=1}^{T}\sqrt{1-4\gamma_t^2} \leq (k-1)exp\left(-2\sum_{t=1}^{T}\gamma_t^2\right).$$

dove k è il numero di classi.

Capitolo 4

Generalizzazioni per affrontare il problema multi-label

In questo capitolo verranno analizzati due modelli che tengono conto del concetto multi-label. Il primo, chiamato SAMME, aggiunge una costante additiva nel calcolo della funzione dell'errore per la creazione di un nuovo modello; il secondo, chiamato Adaboost.MH, parte dalle considerazioni del filtro M2 per sviluppare un nuovo concetto di errore.

4.1 Forward Stagewise Additive Modeling

Prima di arrivare ad enunciare gli algoritmi che verranno trattati in questo capitolo, si introduce il concetto di Forward Stagewise Additive Modeling, che in italiano potrebbe essere tradotto come modello addittivo svolto per passaggi in avanti, dove un modello addittivo generale, si ottimizza attraverso la forward stagewise.

Molti modelli di classificazione e regressione possono essere scritti come combinazione lineare di modelli più semplici:

$$f(x) = \sum_{m=1}^{M} \beta_m b_m(x, \gamma_m)$$

Dove:

- x è un dato di imput
- $\{\beta_m, \gamma_m\}$ sono parametri del modello
- $b_m(x, \gamma_m)$ sono altre funzioni arbitrarie di x

Generalmente,$\{\beta_m, \gamma_m\}$ sono stimate per minimizzare alcune loss function L, che misurano l'errore di predizioni sui dati di training. Spesso questo procedimento risulta essere piuttosto complicato, se però si ottimizza la funzione su una funzione base del tipo:

$$min \sum_{i=1}^{N} L(y_i, \beta b_m(x_i, \gamma)$$

può essere risolto il problema. In altri termini l'idea è, aggiungere sequenzialmente nuove funzioni base per l'espansione della funzione $f(x)$ senza cambiare i parametri che sono stati aggiunti.

L'adaboost fitta un modello addittivo, attraverso il modello forward stagewise, dove:

- La funzione base b_m è un classificatore binario
- La funzione oggetto è la loss esponenziale

$$L(y, f) = e^{-yf(x)}$$

4.2 SAMME

Prima di commentare le particolarità di questa variante, si espone un breve schema riguardante i passaggi principali dell'algoritmo:

- Si inizializza il vettore dei pesi: $D_1(i) = 1/m$
- Il processo viene effettuato per T iterazioni
- Sia $\mathcal{L}$ il classificatore weak iniziale

(a) Viene chiamato il weak learner $\mathcal{L}$

(b) In risposta, esso genera h_t per i dati di training usando i pesi

(c) $\varepsilon_t = \sum_{i=1}^{m} D_t(i)\mathbb{1}(c_i \neq h_t(x_i))/\sum_{i=1}^{m} D_t(i)$ misura l'errore

(d)

$$\alpha_t = \frac{1}{2} \ln \frac{1 - \varepsilon_t}{\varepsilon_t} + log(K - 1) \tag{4.1}$$

(e) Si assegna:

$$D_t(i) \leftarrow D_t(i)\ \exp(\alpha_t \mathbb{1}(c_i \neq h_t(x_i)),\ \mathrm{i} = 1,\ ...,\ \mathrm{m}$$

(f) Ri-normalizza $D_i(t)$

L'output sarà:

$$C(x) = \underset{k}{\operatorname{argmax}} \sum_{(t=1)}^{T} \alpha_t \mathbb{1}(h_t(x) = k)$$

SAMME assomiglia molto alle versioni Adaboost, con la principale differenza in (4.1). Adesso in modo tale da avere α_t positivo, si ha necessità solo che $1-\varepsilon_t > 1/\mathrm{K}$, o che l'accuratezza di ogni classificatore weak sia migliore della classificazione casuale piuttosto che 1/2. Come conseguenza, questo algoritmo pone più peso sulle osservazioni mal classificate in due dimensioni rispetto agli adaboost standard, e inoltre esso combina i classificatori weak in maniera leggermente diversa, ovvero per $\log(K-1)\sum_{t=1}^{T} \mathbb{1}(h_t(x) = k)$.

Nel caso di k=2, ovvero il caso a due classi, la costante addittiva di SAMME diventa un valore pari a 0 (essendo il logaritmo di 1), l'algoritmo si riduce quindi al "classico" adaboost. Questo termine extra, come sarà spiegato in maniera esaustiva nella prossima sezione, non è artificiale e crea un nuovo algoritmo equivalente a fittare un modello addittivo in forward stagewise attraverso la loss function esponenziale. La differenza tra SAMME e Adaboost (quando k=3) è anche illustrata in Figura 1.

Come è già stato detto, Adaboost si arresta dopo che l'errore supera 1/2. Questo non riguarda SAMME: anche se l'errore diventasse maggiore (o uguale) a 1/2, il valore α sarebbe ancora positivo; da ciò, gli esempi di training classificati in maniera errata prendono più peso, e l'errore di test si mantiene decrescente anche dopo 600 iterazioni.

4.2.1 Giustificazione teorica

Come accennato in precedenza la costante additiva $\log(K-1)$ in (4.1) non è artificale, crea un filtro equivalente a fittare un modello addittivo attraverso la forward stagewise. Friedman sviluppò una prospettiva statistica sull'originale algoritmo adaboost a due classi, che induce a vedere l'adaboost a due classi come un modello addittivo forward stagewise attraverso la funzione loss esponenziale:

$$L(y, f) = e^{-yf(\mathbf{x})}$$

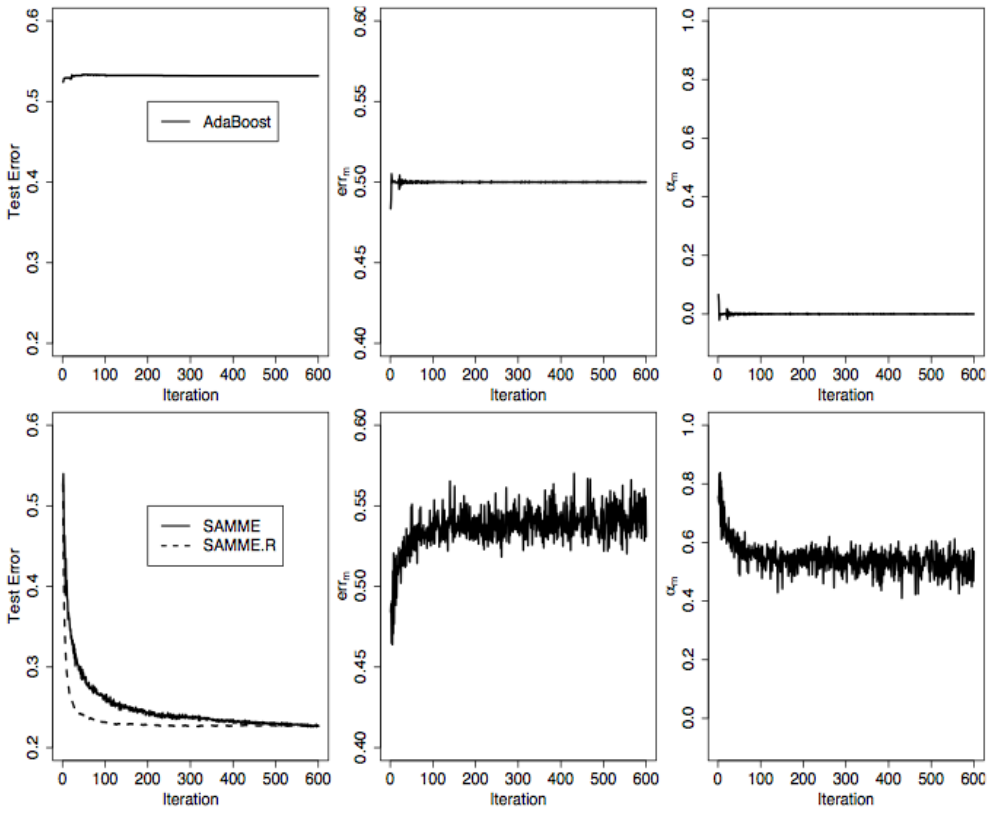


Figura 4.1: Confronto grafico fra Adaboost (prima riga) e SAMME (seconda riga) su una semplice simulazione a tre classi. La taglia di training è 3000, mentre la taglia di test è 10000. Sono stati utilizzati come classificatori weak dieci alberi di decisione.

dove $y = (\mathbb{1}(y = 1) - \mathbb{1}(y = 2)) \in \{-1, 1\}$ in un ambiente di classificazione a due classi. Non è difficile mostrare che l'argmin della funzione loss esponenziale è la metà della trasformazione logit:

$$f^*(\mathbf{x}) = \underset{f(\mathbf{x})}{\operatorname{argmin}} L(y, f) = \frac{1}{2} \frac{(Prob(y)=1|\mathbf{x})}{(Prob(y)=2|\mathbf{x})}$$

La regola di classificazione ottimale di Bayes concorda col segno di $f^*(x)$, ovvero

$$\underset{y}{\operatorname{argmax}} \text{ Prob}(Y = y|\mathbf{X}{=}\mathbf{x}) = \text{sign}(f^*(\mathbf{x}))$$

4.2.2 Nuova loss function multiclasse

In un ambiente di classificazione multiclasse, si può ricodificare l'output y con un vettore $\mathbf{y}$ K-dimensionale, dove tutte le entrate sono uguali a $-\frac{1}{K-1}$ eccetto a 1 in posizione k se $c{=}k$, cioè $\mathbf{y} = (y_1, ..., y_K)^t$ e:

$$y_k = \begin{cases} 1 & y = k \\ -\frac{1}{K-1} & y \neq k \end{cases} \tag{4.2}$$

La stessa architettura è stata utilizzata in altri ambienti come ad esempio nella versione multiclasse di support vector machine (SVM).

La generalizzazione della funzione loss esponenziale al caso multivariato segue naturalmente:

$$L(\mathbf{y},\mathbf{f}) = exp(-\frac{1}{K}(y_1 f_1 + ... + y_K f_K)) = (-\frac{1}{K}\mathbf{y^t f}) \tag{4.3}$$

dove f_k corrisponde alla classe k.

Si noti che c'è bisogno di alcuni vincoli su $\mathbf{f}$ affinchè (4.3) sia stimabile, d'altra parte, aggiungendo qualsiasi costante a tutto f_k non cambia la perdita (loss). Viene scelto di usare il vincolo di simmetria:

$$f_1 + ... + f_K = 0$$

In questo modo, quando K=2 la funzione loss esponenziale multinomiale si riduce alla funzione loss esponenziale classica biclasse.

In maniera simile al caso a due classi, si è interessati a scoprire cosa questa funzione loss esponenziale multiclasse stimi. A questo si può rispondere cercando il suo argmin. In particolare, si è interessati a:

$$\underset{\mathbf{f(x)}}{\text{argmin}}\ \mathrm{E}_{\mathbf{Y}|\mathbf{x}}\exp\ (-\tfrac{1}{K}(Y_1 f_1 + ... + Y_K f_K))$$

$$\text{soggetto al vincolo } f_1 + ... + f_K = 0$$

Segue dai calcoli:

$$exp(-\tfrac{1}{K}(f_1(\mathbf{x}) - \tfrac{f_2(\mathbf{x})}{K-1} - \cdots - \tfrac{f_K(\mathbf{x})}{K-1})Prob((y) = 1|\mathbf{x}) + \ldots$$

$$+exp(-\tfrac{1}{K}(\tfrac{f_1(\mathbf{x})}{K-1} - \tfrac{f_2(\mathbf{x})}{K-1} - \cdots - f_K(\mathbf{x})Prob((y) = K|\mathbf{x}) =$$

$$exp(-\tfrac{1}{K}(F_1(\mathbf{x}) + \tfrac{f_1(\mathbf{x})}{K-1}))Prob((y) = K|\mathbf{x})$$

$$+exp(-\tfrac{1}{K}(f_K(\mathbf{x}) + \tfrac{f_K(\mathbf{x})}{K-1}))Prob((y) = K|\mathbf{x})$$

Attraverso il metodo dei moltiplicatori di Lagrange questo problema di ottimizzazione vincolata può esser riscritto come:

$$exp(-\tfrac{f_1(\mathbf{x})}{K-1}Prob((y) = 1|\mathbf{x})) + \cdots + exp(-\tfrac{f_K(\mathbf{x})}{K-1}Prob((y) = K|\mathbf{x}))+$$

$$-\lambda(f_1(\mathbf{x}) + \cdots + f_K(\mathbf{x})$$

dove λ è il moltiplicatore di Lagrange. Calcolando le derivate rispetto a f_k e λ, troviamo:

$$-\tfrac{1}{K-1}exp(-\tfrac{f_1(\mathbf{x})}{K-1}Prob((y) = 1|\mathbf{x}) - \lambda = 0$$

$$\vdots$$

$$-\tfrac{1}{K-1}exp(-\tfrac{f_K(\mathbf{x})}{K-1}Prob((y) = K|\mathbf{x}) - \lambda = 0$$

$$f_1 + ... + f_K = 0$$

Risolvendo questo sistema di equazioni, si ottiene:

$$f_k^*(\mathbf{x}) = (K-1)(logProb(c = k|\mathbf{x}) - \frac{1}{K}\sum_{k'=1}^{K} logProb(y = k'|\mathbf{x})) \tag{4.4}$$

con k=1,...,K; perciò:

$$\underset{k}{\text{argmax}} f_k^*(\mathbf{x}) = \underset{k}{\text{argmax}} Prob(c = k|\mathbf{x})$$

che è la regola di classificazione ottimale di Bayes per l'errore. Questo giustifica l'uso della funzione loss esponenziale multiclasse. L'equazione (4.4) trova anche un modo per recuperare $Prob(y = k|\mathbf{x})$ una volta che $f_k^*(\mathbf{x})$ sono stimate:

$$Prob(c = k|\mathbf{x}) = \frac{exp(\frac{1}{K-1}f_k^*(\mathbf{x}))}{exp(\frac{1}{K-1}f_1^*(\mathbf{x}))+...+exp(\frac{1}{K-1}f_k^*(\mathbf{x}))} \text{ per k =1, ..., K}$$

4.2.3 SAMME e FSAM

Adesso si mostrerà che l'algoritmo SAMME è equivalente al modello addittivo per forward stagewise utilizzando la loss function esponenziale multiclasse trovata nel paragrafo precedente (4.3). Inizialmente si lavorerà con la loss function canonica, successivamente verrà applicata quella multiclasse.

Avendo dei dati di training, ci auguriamo di trovare $f(\mathbf{x}) = (f_1(\mathbf{x}) + ... + f_K(\mathbf{x}))^t$ tale che:

$$\min_{f(\mathbf{x})} \sum_{i=i}^{m} L(\mathbf{y}_i, f(\mathbf{x}_i)) \tag{4.5}$$

soggetto al vincolo

$$f_1(\mathbf{x}) + ... + f_K(\mathbf{x}) = 0 \tag{4.6}$$

Si consideri $f(\mathbf{x})$ che ha la seguente forma:

$$f(\mathbf{x}) = \textstyle\sum_{t=1}^{T} \beta_t g_t(\mathbf{x})$$

dove $\beta_t \in \mathbb{R}$ sono coefficienti e $g_t(\mathbf{x})$ sono funzioni base. Ordiniamo a $g_t(\mathbf{x})$ di soddisfare il vincolo $g_1(\mathbf{x}) + ... + g_K(\mathbf{x}) = 0$.

Per esempio,la funzione $g_t(\mathbf{x})$ che si considera prende valore in uno dei K possibili vettori K-dimensionali; precisamente, dato $\mathbf{x}$, $\mathbf{g(x)}$ mappa $\mathbf{x}$ su $\mathcal{Y}$

$$g : \mathbf{x} \in \mathbb{R}^p \rightarrow \mathcal{Y}$$

dove $\mathcal{Y}$ contiene K vettori K-dimensionali.

Il modello forward stagewise approssima la soluzione delle richieste (4.5)- (4.6) sequenzialmente aggiungendo nuove funzioni base all'espansione senza adattare i parametri e coefficienti di quelli che erano già stati aggiunti. Specificatamente, l'algoritmo parte con $f_0(\mathbf{x}) = 0$ sequenzialmente seleziona nuove funzioni base da un dizionario e le aggiunge:

- Inizializza $f_0(x) = 0$
- Per $t = 1...T$ iterazioni

  Calcola:

$$(\beta_t, g_t(\mathbf{x})) = \min_{\beta, g} \textstyle\sum_{i=1}^{m} L(\mathbf{y}_i, f_{t-1}(\mathbf{x}_i) + \beta g(\mathbf{x}_i))$$

Assegna:

$$f_t(\mathbf{x}) = f_{(t-1)}(\mathbf{x}) + \beta_t g_t(\mathbf{x})$$

Il passo cruciale di questo studio avviene al calcolo di (β_t, g_t), infatti, usando la funzione loss esponenziale multivariata:

$$(\beta_t, g_t) = \underset{\beta,g}{\operatorname{argmin}} \sum_{i=i}^{m} exp(-\frac{1}{K}\mathbf{y}_i^t(f_{t-1}(\mathbf{x}_i) + \beta g(\mathbf{x}_i)) \tag{4.7}$$

$$= \underset{\beta,g}{\operatorname{argmin}} \sum_{i=1}^{m} D_t(i) exp(-\frac{1}{K}\beta \mathbf{y}_i^t g(\mathbf{x}_i)) \tag{4.8}$$

dove $D_t(i) = \exp\, -\frac{1}{K}\mathbf{x}_i^t(f_{t-1}(\mathbf{x}_i)$ sono i pesi non normalizzati.

Si noti che ogni funzione $g(\mathbf{x})$ nei K vettori K-dimensionali ha una corrispondenza con un classificatore multiclasse $h(x)$ come segue:

$$h(x) = k, \tag{4.9}$$

$$\text{se } g_k(\mathbf{x}) = 1$$

e vice versa.

Quindi, risolvere $g_t(\mathbf{x})$ è equivalente a trovare il classificatore multi-classe $h_t(x)$ che può generare $g_t(\mathbf{x})$. Di seguito si procederà quindi con lo svolgimento della soluzione:

$$h_t(\mathbf{x}) = argmin \sum_{i=1}^{m} D_t(i)\mathbb{1}(y_i \neq h(\mathbf{x}_i)) \tag{4.10}$$

$$\beta_t = \frac{(K-1)^2}{K}(log\frac{1-\varepsilon_t}{\varepsilon_t} + log(K-1) \tag{4.11}$$

Dove ε_t è definito come :

$$\varepsilon_t = \textstyle\sum_{i=1}^{m} D_t(i)\mathbb{1}(c_i \neq h(\mathbf{x}_i))/\sum_{i=1}^{m} D_t(i)$$

Adesso, per ogni valore fissato $\beta > 0$, utilizzando la definizione (4.9)
si può esprimere il criterio in (4.8) come:

$$\sum_{y_i=h(x_i)} D_t(i)\text{exp } (-\tfrac{\beta}{K-1} - \sum_{y_i \neq h(x_i)} D_t(i)\text{exp } (\tfrac{\beta}{(K-1)^2}) =$$

$$exp(-\frac{\beta}{K-1}\sum_i D_t(i) + (exp(\frac{\beta}{(K-1)^2}) - exp(\frac{\beta}{K-1}))\sum_i^m D_t(i) \qquad (4.12)$$

Solo l'ultima somma dipende dal classificatore $h(\mathbf{x})$, noi prendiamo quella che contiene la (4.10). A questo punto, inserendo (4.10) in (4.8) e risolvendo per β, si ottiene (4.11).

Il modello viene poi aggiornato:

$$f_t(\mathbf{x}) = f_{t-1}(\mathbf{x}) + \beta_t g_t(\mathbf{x})$$

e i pesi per la prossima iterazione saranno:

$$D_t(i) \leftarrow D_t(i)\ \text{exp } (-\tfrac{1}{K}\beta_t y_i^t g_t(\mathbf{x}_i))$$

Questo equivale a:

$$D_t(i)exp(-\frac{(K-1)^2}{K^2}\alpha_t y_i^t g_t(\mathbf{x}_i)) = \begin{cases} D_t(i)exp(-\frac{(K-1)^2}{K}\alpha_t) & y_i = h(x_i) \\ D_t(i)exp(\frac{1}{K}\alpha_t) & y_i \neq h(x_i) \end{cases} \qquad (4.13)$$

Dove α_t è definita in (4.1) col termine extra log(K-1), e il nuovo peso è equivalente al peso dell'algoritmo SAMME aggiornato dopo la normalizzazione.

4.3 Adaboost.MH

Adaboost.MH è una generalizzazione del filtro M2. Potendo la singola osservazione x appartenere a diverse etichette (concetto multi-label), si introduce un nuovo algoritmo che, con diversi concetti di funzione loss, affronta la possibilità di cambiamento delle label. Si noti, come si mostrerà in questo paragrafo, che nel caso ogni osservazione appartenga a una sola classe, questo filtro si riduce ad Adaboost.M2.

Sia Y un set finito di etichette, e sia $k = |Y|$. In un ambiente di classificazione tradizionale, ogni esempio $x \in X$ alla singola classe $y \in Y$, quindi gli esempi etichettati sono coppie (x, y). Lo scopo poi, generalmente, è di trovare un'ipotesi $H : X \to Y$ che minimizza la probabilità che $y \neq H(x)$ su un nuovo esempio

osservato (x, y).

Nel caso multi-label, ogni istanza $x \in X$ può appartenere a diverse classi in Y. Quindi, un esempio etichettato è una coppia $(x, \mathcal{Y})$ dove $\mathcal{Y} \subseteq Y$ nel set di etichette assegnate a x. Il caso single-label è chiaramente un caso speciale quando $|Y| = 1$ per tutte le osservazioni.
Non è chiaro in questo ambiente precisamente come formalizzare lo scopo dell'algoritmo di apprendimento, e, in generale, la "giusta" formalizzazione può dipendere dal problema. Una possibilità è di cercare un'ipotesi che prova a predire solo una delle etichette assegnate a un esempio. In altre parole, lo scopo è di trovare $H : X \to Y$ che minimizza la probabilità che $H(x) \notin \mathcal{Y}$ su una nuova osservazione $(x, \mathcal{Y})$. Chiamiamo questa misura "un errore" delle ipotesi H siccome misura la probabilità di non ottenere anche una tra le etichette corrette. Si denota "un errore" di un'ipotesi h rispetto alla distribuzione D sulle osservazioni $(x, \mathcal{Y})$ in questo modo:

$$\text{one-err}_{D}(H) = Pr_{(x,Y)\tilde{}D}\left[H(x) \notin \mathcal{Y}\right]$$

Si noti che, per un problema di classificazione single-label, "un errore" è il corrispondente dell'errore ordinario. Nel paragrafo seguente, si introdurrà una nuova misura di perdita (loss) che può essere utilizzata in questo ambiente multiclasse, chiamata Hamming loss (o ranking loss). Si analizzeranno inoltre le modifiche appropriate all'Adaboost per via di questo nuovo concetto di perdita (loss).

4.3.1 Utilizzo della Hamming loss per i problemi multiclasse

Si supponga adesso che lo scopo sia di predire tutte e solo le etichette corrette. Ovvero, l'algoritmo di apprendimento genera un'ipotesi che predice sets di etichette, e la loss dipende su come queste predizioni differiscono da una che è stata osservata. Perciò, $H : X \to 2^Y$ e, rispetto alla distribuzione D la loss è:

$$\tfrac{1}{k} E_{(x,Y)\tilde{}D}\left[|h(x) \triangle \mathcal{Y}|\right]$$

dove $\triangle$ denota la differenza simmetrica (il valore 1/k è utilizzato solo per assicurarsi che il valore stia in $[0, 1]$). Chiamiamo questa misura *Hamming loss* di H, e la denotiamo come $hloss_D(H)$. Per minimizzare la Hamming loss, si può, in maniera naturale, decomporre il problema in k problemi di classificazione binaria ortogonali. Si pensa a $\mathcal{Y}$ come k etichette binarie (dipendendo da che un'etichetta y sia o non sia inclusa in $\mathcal{Y}$). Similmente, *h(x)* può essere vista come k predizioni binarie. La Hamming loss poi può essere considerata come una media dell'errore di h su questi k problemi binari.
Per $\mathcal{Y} \subseteq Y$, si definisce $\mathcal{Y}[l]$ per $l \in Y$ essere:

$$\mathcal{Y}[l] = \begin{cases} 1 & l \in \mathcal{Y} \\ -1 & l \notin \mathcal{Y} \end{cases}$$

Per semplificare la notazione, si identifica anche qualsiasi funzione $H : X \to 2^Y$ con una corrispondente funzione a due argomenti $H : X \times Y \to \{-1, 1\}$ definita da $H(x, l) = H(x)[l]$

Con la riduzione alla classificazione binaria in mente, è piuttosto inequivocabile vedere come utilizzare il boosting per minimizzare la Hamming loss. L'idea principale della riduzione è semplicemente ripetere ogni esempio di training (x_i, Y_i) per k esempi $((x_i, l), Y_i[l])$ per $l \in \mathcal{Y}$. Il risultato è un algoritmo di boosting chiamato proprio Adaboost.MH che mantiene una distribuzione sugli esempi i e etichette l. Come per ogni classificatore, si riporta un breve schema con i passaggi essenziali dell'algoritmo.

- Dati: $(x_1, Y_1), ..., (x_m, Y_m)$ dove $x_i \in X, \mathcal{Y}_i \subseteq Y$
- Si inizializza $D_1(i, l) = 1/(mk)$
- Per t = 1, ..., T

(a) Viene chiamato il weak learner utilizzando la distribuzione D_t

(b) Il weak learner, in risposta, genera un'ipotesi $h_t : X \times Y \to \mathbb{R}$

(c) Viene calcolato $\alpha_t \in \mathbb{R}$

(d) Viene aggiornata:

$$D_{t+1}(i, l) = \frac{D_t(i,l) exp(-\alpha_t Y_i[l] h_t(x_i, l))}{Z_t}$$

Dove Z_t è un fattore di normalizzazione (scelto in modo tale che D_{t+1} sia una distribuzione).
L'output sarà:

$$H(x, l) = sign(\sum_{t=1}^{T} \alpha_t h_t(x, l))$$

Al passo t, il weak learner accetta così una distribuzione D_t (in aggiunta al training set), e genera un'ipotesi weak $h_t : X \times Y \to \mathbb{R}$. Questa riduzione conduce anche alla scelta dell'ipotesi finale mostrata nello schema.

Teorema 3. Assumendo la notazione dello schema appena esposto, il seguente margine superiore per la Hamming loss di H sui dati di training:

$$\text{hloss(H)} \leq \prod_{t=1}^{T} Z_t$$

Si può ora applicare l'idea della sezione precedente in questo problema di classificazione binaria. Come prima, lo scopo è minimizzare:

$$Z_t = \sum i, l D_t(i,l) exp(-\alpha_t \mathcal{Y}_i\,[l]\, h_t(x_i, l))$$

per ogni iterazione.

Se si richiede che ad ogni passo h_t abbia range in $[-1, +1]$, poi bisognerebbe scegliere

$$\alpha_t = \tfrac{1}{2}\ln(\tfrac{1+r_t}{1-r_t})$$

dove:

$$r_t = \textstyle\sum_{i,l} D_t(i,l)\mathcal{Y}_i\,[l]\, h_t(x_i, l)$$

Questo porta a:

$$Z_t = \sqrt{1 - r_t^2}$$

E lo scopo del weak learner diventa la minimizzazione di $|r_t|$.
Si noti che (1-r_t)/2 equivale a:

$$Pr_{(i,l)\tilde{}D}\,[h_t(x_i, l) \neq \mathcal{Y}_i\,[l]]$$

Facciamo un esempio di come minimizzare $|r_t|$. Si supponga che lo scopo sia di trovare un'ipotesi weak h_t che "ignora" l'istanza x e predice soltanto sulla base dell'etichetta l. Si può quindi omettere l'argomento x e scrivere $h_t(x, l) = h_t(l)$; oltre che al pedice t . Per simmetria, minimizzare -r equivale a massimizzare r. Perciò, bisogna solo trovare h che massimizzi

$$r = \textstyle\sum_{i,l} D_t(i,l)\mathcal{Y}_i\,[l]\, h_t(l)$$

$$= \textstyle\sum_l \text{sign}\sum_i D(i,l)\mathcal{Y}_i\,[l]$$

4.4 Conclusioni

Abbiamo analizzato a fondo le principali varianti per l'adaboost multiclasse. Dagli studi fatti, il problema riscontrato nel corso del tirocinio ha come soluzione migliore il filtro Adaboost.M2: infatti, nonostante le previsioni sportive abbiano un concetto di multi-etichetta (label) questo viene implementato nella fase di addestramento, una fase che noi non abbiamo dovuto far svolgere all'algoritmo in quanto avevamo i bookmakers che rappresentano classificatori già istruiti. È stata presa in considerazione inizialmente di partire dal programma SAMME e modificarlo per arrivare ad un filtro che soddisfasse le nostre richieste. Il programma di SAMME, però, risulta implimentativamente molto complicato, e quindi è stato un motivo in più utilizzare il programma dell'algoritmo Adaboost.M2 e modificarlo per far sì che il filtro riconoscesse i bookmakers come i propri classificatori addestrati.

Capitolo 5

Appendice

5.1 Loss Function

Nell'ottimizzazione matematica, statistica, teoria delle decisioni e machine learning, la loss function (funzione di perdita) o cost function (funzione di costo) è una funzione che elabora un evento o un valore di una o più variabili in un numero reale intuitivamente rappresentando qualche "costo", associato all'evento. Un problema di ottimizzazione cerca di ridurre la loss function. Una funzione oggetto è o una loss function o la sua negativa (chiamata qualche volta regard function o utility function).

In statistica, tipicamente una loss function è utilizzata per stimare parametri, e l'evento in questione è qualche funzione della differenza tra valori stimati e valori reali per un'istanza di dati. Il concetto, fu introdotto in statistica da A. Wald. In economia, per esempio, questa è solitamente l'economic costo (o regret). Nella classificazione è la pena per una classificazione sbagliata di un esempio.

5.2 Logit Function

La funzione logit è una funzione usata in matematica, specialmente in statistica. Quando il parametro di una funzione rappresenta una probabilità p , la funzione logit fornisce il log-odds, ovvero il logaritmo delle odds p/(1-p).
Il logit di un numero p compreso tra 0 e 1 è dato dalla formula:

$$\text{logit } (p) = \log(\tfrac{p}{1-p}) = -\log(\tfrac{1}{p} - 1)$$

5.3 Moltiplicatori di Lagrange

Nella giustificazione della costante addittiva nell'algoritmo Adaboost SAMME, viene utilizzato un processo di ricerca dei massimi e minimi di una funzione conosciuto col nome di "moltiplicatori di Lagrange".
Per una comprensione migliore si deve, per prima cosa, esaminare la teoria elementare meglio conosciuta. Concerne col problema di trovare, per una funzione $f(x, y, \dots)$ in una regione data G , il punto $x_0, y_0, \dots$ in G che la funzione $f(x, y, \dots)$ ha come massimo o minimo (estremi) rispetto a tutti i punti di G nelle vicinanze di $x_0, y_0, \dots$. Questo problema ha sempre una soluzione, in accordo con il teorema di Weiestrass che dice:
Teorema di Weiestrass: Ogni funzione che è continua in un dominio chiuso G delle variabili possiede un valore più grande e uno più piccolo all'interno o al margine del dominio.
Se la funzione $f(x, y, \dots)$ è differenziabile in G e se un valore estremo è raggiunto da un punto interno $x_0, y_0, \dots$, allora le derivate di $f(x, y, \dots)$ rispetto ad ogni variabili si annullano in $x_0, y_0, \dots$, in altre parole, il gradiente di f è uguale a zero.
Ma questa condizione necessaria non è condizione sufficiente, come può essere visto dall'esistenza dei punti di flesso e quelli di sella (per esempio $f(x) = x^3$ a $x_0 = 0$; $f(x, y) = xy$ a $x_0 = 0, y_0 = 0$). In generale, punti di cui tutte le derivate si annullano, ovvero che $\partial f = 0$, sono conosciuti come *punti stazionari*. Punti stazionari che forniscono un massimo e un minimo relativi ad una vicinanza appropriata sono chiamati "estremi". Se le variabili non sono indipendenti, ma soggette a restrizioni $g_1(x, y, \dots) = 0, g_2(x, y, \dots) = 0, \dots, g_h(x, y, \dots) = 0$, si ottengono condizioni necessarie per un punto, estremo o stazionario per mezzo dei moltiplicatori di Lagrange.
Questo metodo consiste nel seguente procedimento: Per trovare un punto all'interno del dominio di variabili indipendenti di cui $f(x, y, \dots)$ ha un punto estremo o solamente uno stazionario, si introducono h +1 nuovi parametri, i "moltiplicatori", $\lambda_0, \lambda_1, \dots, \lambda_h$ e si costruisce la funzione

$$F = \lambda_0 f + \lambda_1 g_1 + \lambda_2 g_2 + \dots, \lambda_h g_h$$

A questo punti si determinano le quantità $x_0, y_0, \dots$, e i rapporti di $\lambda_0, \lambda_1, \dots, \lambda_h$ attraverso le equazioni

$$\frac{\partial F}{\partial x} = 0, \frac{\partial F}{\partial y} = 0, \dots$$

$$\frac{\partial F}{\partial \lambda_1} = g_1 = 0, \dots, \frac{\partial F}{\partial \lambda_h} = g_h = 0$$

Queste equazioni rappresentano le condizioni desiderate per il carattere stazionario di $f(x, y, \dots)$ o l'estremo di f sotto le restrizioni date. Se $\lambda_0 \neq 0$ si potrebbe (e dovrebbe) mettere $\lambda_0 = 1$ perchè F è omogeneo nelle quantità λ_i. Il metodo

di Lagrange è semplicemente uno strumento che, preservando la simmetria, evita l'eliminazione esplicita di h nelle variabili dalla funzione $f(x, y, \dots)$, mediante restrizioni sussidiarie. Si considerino ora due istruttivi, sebbene elementari, esempi.

(a) Di tutti i triangoli, dati la base e il perimetro, il triangolo isoscele ha l'area più grande. Di tutti i triangoli, dati la base e l'area, il triangolo isoscele ha il perimetro più lungo. Queste considerazioni possono essere risolte senza calcoli considerando l'ellissi per il quale la base data è la linea che connette i due fuochi.

(b) Problema di Steiner. Dati tre punti A_1, A_2, A_3, i quali formano un angolo acuto, un quarto punto P deve essere trovato affinchè la somma delle distanze $PA_1 + PA_2 + PA_3$ sia la più piccola possibile. Si consideri un cerchio attraverso P con centro in A_3; allora P deve essere posto sul cerchio in modo tale che $PA_1 + PA_2$ sia la più piccola possibile. Lo stesso ragionamento deve essere fatto intercambiando i punti 1, 2, 3; tutti i tre angoli $A_1PA_2, A_2PA_3, A_3PA_1$ devono, perciò, essere equivalenti e pari a $2\pi/3$. Il problema è quindi risolto.

Riferimenti

- [1] Xindong Wu, Vipin Kumar, J. Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda, Geoffrey J. McLachlan, Angus Ng, Bing Liu, Phillip S. Yu, Zhi-Hua Zhou, Michael Steinbach, David J. Hand, Dan Steinberg. *Top 10 Algorithms in Data Mining.* Knowledge and Information Systems, 14(1):1-37, 2008.

- [2] Yoav Freund, Robert E.Schapire. *Experiments with a New Boosting Algorithm.* Machine Learning: Proceedings of the Thirteenth International Conference, 1996.

- [3] Ji Zhu, Saharon Rosset, Hui Zou, Trevor Hastie (January 12, 2006). *Multi-class Adaboost.*

- [4] Robert E. Schapire, Yoram Singer. *Improved Boosting Algorithms Using Confidence-rated Predictions.* Machine Learning, 37(3):297-336, 1999.

- [5] Richard Courant, David Hilbert. *METHODS OF MATHEMATICAL PHYSICS.* Volume I, pp 164-166.

- [6] Wikipedia contributors, *Adaboost.* Wikipedia, The Free Encyclopedia, 10 December 2008, 22:42 UTC, Online: accessed 20-November-2014.

- [7] Wikipedia contributors, *Logit.* Wikipedia, The Free Encyclopedia, 19 March 2014, 20:28 UTC, Online: accessed 20-November-2014.

- [8] Wikipedia contributors, *Loss Function.* Wikipedia, The Free Encyclopedia, 14 November 2014, 06:40 UTC, Online: accessed 20-November-2014.

Ringraziamenti

Ringrazio particolarmente il Prof. Malfanti per avermi dato la possibilità di lavorare per tre mesi con lui e ad avere dato un grosso contributo per lo sviluppo della mia tesi di laurea. Ringrazio la Prof.ssa Riccomagno per l'aiuto fornito durante l'approfondimento delle tematiche affrontate durante il corso del mio tirocinio.
Voglio ringraziare inoltre il collega del Prof. Fabrizio Malfanti, Andrea Cabella, per l'aiuto nella creazione del programma e per le giornate trascorse insieme.
Voglio ringraziare i miei splendidi genitori, Lorella e Renzo, per essere stati sempre al mio fianco in ogni decisione della mia vita, per dimostrarmi ogni giorno il bene che provano e per avermi dato la possibilità di continuare i miei studi. Grazie a mio nonno che mi ha insegnato ad apprezzare la vita anche per le piccole cose; un pensiero va anche alle mie nonne Emma e Marisa, e a mio zio Remo, siete i miei Angeli.
Durante questi tre anni ho conosciuto persone fantastiche, ma il primo ringraziamento va ad Andrea, con il quale ho condiviso la maggior parte dei miei giorni dalla quinta elementare a oggi.
Grazie ai miei compagni di corso, che si sono rivelati persone stupende e amici veri: Beniamino, Psyco, Massi, Fex, Michela, Sara e Serena; grazie anche alle persone conosciute grazie a questa esperienza: Dala, LaDoppiaL, Lo Zio e Petrosyan.
Grazie ai miei amici che sono una vera e propria famiglia: Federico, Feo, Davide, Nicolo, Jacopo, Enri, Giulio e Beis. Infine grazie ai miei amici di Vignale, in particolare Michele, Edoardo, Ghibo e Bebeto e alle due mie amiche Perla e Chiarella.