Università di Trento

Dipartimento di Matematica

Laurea Triennale in Matematica

~ · ~

Anno Accademico 2023–2024

Una panoramica sulle tecniche di classificazione in Machine Learning

Relatore
Prof. Luigi Amedeo Bianchi

Candidato
Daniele Capelli
226723

Data della Discussione: Luglio 2024

Alle mie nonne Ancilla e Maria

Ringraziamenti

Innanzitutto desidero esprimere il mio più sentito ringraziamento al mio relatore, il Prof. Luigi Amedeo Bianchi, per l'aiuto e il supporto che mi ha fornito durante questi mesi. I suoi consigli e le sue dritte sono stati fondamentali sia per la realizzazione di questa tesi che per la buona uscita del mio primo seminario.
Vorrei poi ringraziare i miei genitori, Clelia e Roberto, per essermi sempre stati vicini e avermi sostenuto nel mio percorso di studi triennale. Un ringraziamento va anche ai miei fratelli, Stefano, Matteo e Lorenzo, con una menzione speciale per il mio gatto e il mio pappagallo, ormai pronti a sostenere un percorso di studi tutto loro dopo avermi ascoltato ripetere per gli esami.
Desidero inoltre ringraziare sia i miei compagni del Collegio Clesio che i miei colleghi di corso, che durante questi tre anni mi hanno aiutato a superare le difficoltà incontrate, riuscendo a strapparmi sempre un sorriso.
Infine ringrazio tutti i professori e le persone che ho avuto modo di conoscere e che mi hanno aiutato a crescere sia personalmente che accademicamente.

Indice

Introduzione

Il Machine Learning è una branca dell'intelligenza artificiale che si pone come obiettivo quello di addestrare i computer ad apprendere dai dati e migliorare le proprie performance senza essere esplicitamente programmati per farlo: in particolare, le tecniche che rientrano all'interno di questo campo sono utili ad analizzare grandi set di dati per trovarne relazioni interne. Possiamo dividere gli algoritmi che rientrano all'interno del Machine Learning in due grandi categorie: tecniche di apprendimento supervisionato e tecniche di apprendimento non supervisionato. Nello specifico, l'apprendimento supervisionato si occupa di studiare modelli che vengono addestrati su un insieme di dati etichettati, ovvero forniti tramite una coppia input-output, con l'obiettivo di trovarne uno che restituisca previsioni su nuovi dati che siano il più accurate possibili. Ci verranno quindi forniti una serie di dati $(X_1, Y_1), \ldots, (X_N, Y_N)$ dove X_i è un vettore che contiene le informazioni riguardanti l'i-mo campione del dataset, mentre Y_i ne rappresenta il valore output. All'interno di questa categoria troviamo tecniche quali la regressione lineare e logistica o gli alberi di classificazione. Al contrario, l'apprendimento non supervisionato raggruppa quelle tecniche in cui il modello viene addestrato su un insieme di dati non etichettati, ovvero sprovvisti di un output, con l'obiettivo di trovare strutture non note a priori e sottostanti ai dati forniti: avremo a disposizione dunque semplicemente una serie di osservazioni $X_1, \ldots, X_N$. Ne sono un esempio il clustering, ovvero la divisione in gruppi senza che questi siano noti a priori, e la riduzione della dimensionalità (tramite tecniche quali la Principal Component Analysis).

Lo scopo di questo elaborato consiste nello studio dei problemi di classificazione, ovvero problemi in cui l'obiettivo consiste nell'assegnare un oggetto a una determinata classe conoscendone una serie di caratteristiche intrinseche. Questo è un problema piuttosto ricorrente in diversi ambiti: ad esempio, in contesto medico può risultare utile fornire modelli che permettano di stabilire se un paziente è affetto da una determinata malattia oppure no, mentre in ambito finanziario possiamo utilizzare queste tecniche per calcolare il rating di uno Stato. Nello specifico, se interpretiamo la classe di appartenenza dell'oggetto come l'output del nostro modello, ci accorgiamo che siamo all'interno di un contesto di addestramento supervisionato: lavoreremo quindi sotto l'ipotesi di avere a disposizione un dataset contenente dati di cui conosciamo già la classe di appartenenza. Nel dettaglio, studieremo i seguenti metodi.

- Nel Capitolo 2 vedremo come affrontare il problema senza la formulazione a priori di nessuna ipotesi sulla reale forma delle classi di risposta.

- Nel Capitolo 3 vedremo come utilizzando la regressione logistica si può affrontare il problema quando abbiamo un modello lineare sottostante alle classi.
- Successivamente, nel Capitolo 4, vedremo i modelli generativi per la classificazione, che permettono di migliorare i risultati ottenuti dalla regressione logistica sotto determinate ipotesi di partenza.
- Come ultimo spunto, nel Capitolo 5, vedremo un metodo di classificazione che risulta facilmente interpretabile a livello grafico tramite gli alberi di classificazione. Concluderemo introducendo alcuni metodi ensemble, nello specifico bagging e foreste randomiche.

Nel corso della nostra analisi studieremo da un punto di vista matematico alcuni degli aspetti principali di questi metodi, per poi andare a presentare degli esempi concreti che ci aiutino a capire meglio quanto esposto a livello teorico. Per l'analisi degli esempi pratici utilizzeremo concetti, molto diffusi all'interno del Machine Learning, quali training e test error, k-fold cross validation, accuratezza e matrici di confusione: questi sono spiegati in dettaglio nel Capitolo 1.

Nel corso dell'elaborato utilizzeremo [7] e [5] come riferimenti bibliografici.

Capitolo 1

Divisione dei dati durante l'addestramento e misure dell'errore

1.1 Introduzione

I metodi che rientrano all'interno del Machine Learning si pongono come obiettivo di restituire un modello che, a partire da una serie di dati iniziali forniti in fase di addestramento, possa fornire delle predizioni che risultino il più accurato possibile. Introduciamo quindi una serie di concetti, come la divisione tra training e test error e la definizione di una misura dell'errore, che saranno fondamentali nel seguito, concentrandoci nello specifico sulle tecniche di apprendimento supervisionato e sui problemi di classificazione.

1.2 Training e Test Error

Dato un metodo di apprendimento supervisionato, denotiamo con $\mathcal{H}$ l'insieme delle sue possibili istanze, ovvero l'insieme che contiene le diverse forme che il metodo può assumere al variare dei suoi coefficienti. In generale, se indichiamo con $E(X_i, Y_i, h)$ una misura che quantifica l'errore che risulta dall'utilizzare il predittore h per classificare l'oggetto i, che presenta attributi X_i e classe reale Y_i, vogliamo andare a risolvere

$$\min_{h \in \mathcal{H}} \frac{1}{N} \sum_{i=1}^{N} E(X_i, Y_i, h) \tag{1.1}$$

Chiamiamo training data (in quanto sono i dati utilizzati in fase di addestramento del nostro algoritmo) gli N dati (X_i, Y_i), al variare di $i = 1, \ldots, N$, mentre indichiamo la quantità

$$\frac{1}{N} \sum_{i=1}^{N} E(X_i, Y_i, h) \tag{1.2}$$

come training error, in quanto restituisce l'errore medio ottenuto sui dati utilizzati in fase di addestramento: risolvendo (1.1) otteniamo quindi un predittore h^* che ci permette di avere buone predizioni sui dati che abbiamo usato. Questo approccio presenta

una problematica di base, nota come overfitting: infatti può capitare che, applicando h^* per classificare nuovi dati che non avevamo utilizzato nella fase di addestramento del nostro metodo, otteniamo delle pessime predizioni in quanto (1.1) può restituire dei minimi che sono solo locali. Per risolvere questo problema un modo semplice di procedere consiste nel considerare una nuova serie di M dati (X'_j, Y'_j), indipendenti da quelli utilizzati precedentemente, da utilizzare per testare il nostro metodo attraverso la seguente quantità

$$\frac{1}{M}\sum_{j=1}^{M} E(X'_j, Y'_j, h^*) \tag{1.3}$$

In particolare, (1.3) è nota come test error e i dati utilizzati per calcolarla vengono detti test data. Quindi, per ottenere un modello h^* che sia ben performante, vogliamo combinare lo studio di training e test error: nello specifico, non saremo interessati a trovare necessariamente il minimo valore possibile per (1.2), ma semplicemente cercheremo istanze $\bar{h}$ che forniscano valori di test e training error bassi.
A questo punto rimane un ultimo problema da affrontare: nella realtà non abbiamo a disposizione due dataset diversi da utilizzare come training e test data. Dobbiamo quindi trovare un modo di lavorare sulle osservazioni che ci sono state fornite per dividerle opportunamente nei due sottoinsiemi che abbiamo introdotto precedentemente: tra i possibili modi di procedere vedremo il Validation Set Approach e la validazione incrociata a k-fold. Inoltre introdurremo anche la tecnica del bootstrap, utile per l'implementazione dei metodi ensemble.

1.3 Validation Set Approach

Un primo modo per affrontare la divisione di un dataset in training e test set è dato dalla tecnica del validation set approach: questa consiste nel dividere in maniera casuale i dati che ci sono stati forniti in due sottoinsiemi, detti training e validation set. Il primo insieme, così come suggerisce il nome, viene utilizzato per calcolare il training error da utilizzare per ottenere una stima dei coefficienti del modello: in corrispondenza dei valori che minimizzano (1.2), si predicono le risposte alle osservazioni presenti nel secondo insieme per determinare una stima del test error, detta validation error, data semplicemente dalla media della misura degli errori calcolate utilizzando queste predizioni.
Questo approccio, seppur molto semplice, presenta due problemi: il primo consiste nel fatto che la stima del validation error è molto variabile, in quanto dipende fortemente dalla divisione effettuata. Inoltre con questo metodo istruiamo il modello su poche osservazioni, soluzione generalmente deprecata in quanto i metodi statistici tendono a performare meglio quando costruiti su un numero grande di dati. Per queste ragioni si introduce una nuova tecnica, nota come validazione incrociata a k-fold.

1.4 Validazione incrociata a k-fold

La validazione incrociata a k-fold, o k-fold cross-validation, è una delle tecniche più utilizzate per ottenere una buona stima del test error: in particolare questo approccio ci permette di combinare i dati in modo che, sia in fase di addestramento che in fase di testing, i dati utilizzati siano in un numero ragionevole. Questo metodo consiste nel dividere il dataset di partenza in k insiemi, detti appunto folds, che abbiano approssimativamente la stessa dimensione, e procedere poi per step. Al primo passo, ragionando in maniera analoga a quanto fatto con il validation set approach, si considera il primo insieme come validation set e si istruisce il modello sull'unione degli altri folds: in questo modo si ottiene una stima del test error data dal validation error, che possiamo chiamare $E_{cv,1}$. A questo punto si ripete la stessa operazione k volte, andando a considerare al passo j come validation set il j-mo fold e istruendo il metodo sull'unione degli altri insiemi: si ottiene quindi il validation error dato da $E_{cv,j}$. Conclusa l'operazione possiamo combinare le quantità ottenute per ottenere una stima dell'errore totale data da

$$CV(k) = \frac{1}{k} \sum_{i=1}^{k} E_{cv,i} \tag{1.4}$$

A seconda del numero k di folds utilizzati il risultato ottenuto potrebbe variare: nella pratica, per tenere conto del costo computazionale dell'operazione, si ricorre solitamente all'uso di $k = 5$ oppure $k = 10$ folds.

1.5 Bootstrap

Come vedremo nella Sezione 5.4 alcune tecniche di Machine Learning si basano sul combinare i risultati ottenuti addestrando il modello su diversi dataset: il problema di questo approccio consiste nel fatto che nella pratica ne abbiamo a disposizione solo uno. Un'idea potrebbe quindi essere quella di dividere casualmente i dati all'interno di questo insieme in K gruppi, che interpretiamo come dataset distinti: tuttavia, come già detto, i metodi statistici performano meglio quando addestrati su un maggior numero di dati, quindi questo approccio risulta poco funzionale. Si introduce quindi la tecnica del bootstrap, che permette di simulare nuovi dataset con un elevato numero di osservazioni al loro interno. Supponiamo di avere a disposizione N campioni nel dataset originale: per crearne uno nuovo si procede nel seguente modo. Si crea un insieme contenente N dati, in cui ogni osservazione è scelta in modo casuale e indipendentemente dalle altre tra le N disponibili nel dataset originale: può succedere che qualche osservazione venga considerata più volte, così come può accadere che altre non compaiano. Ripetendo questo processo B volte, otteniamo B diversi dataset che possono quindi essere utilizzati per stimare i parametri e i coefficienti del metodo che stiamo studiando.

1.6 Addestramento su nuovi dati

Quando andiamo ad implementare concretamente delle tecniche di Machine Learning può capitare che ci vengano forniti dei dati in momenti diversi: ad esempio, se stiamo studiando l'incidenza di una malattia, è frequente ricevere informazioni su diversi gruppi di pazienti man mano che si raccolgono nuovi dati. Ovviamente siamo interessati a trovare un modo di tenere in considerazione tutto quello che ci viene fornito, anche se questo avviene in tempi successivi a un primo addestramento: vediamo diverse strategie con cui possiamo realizzare ciò.

- Retraining periodico: questa tecnica consiste nel riaddestrare interamente il nostro modello ogni volta che abbiamo a disposizione nuovi dati, utilizzando come training data osservazioni prese sia dal vecchio dataset che dal nuovo. Nel complesso è una buona via ma richiede di conservare i dati storici e può risultare computazionalmente oneroso.
- Addestramento incrementale: con questa tecnica si addestra il modello solo sui nuovi dati a disposizione.
- Apprendimento online: l'aggiornamento del modello avviene man mano che arrivano dei nuovi dati attraverso tecniche di apprendimento online, come suggerisce il nome.
- Fine-tuning: in questo approccio utilizziamo i nuovi dati a nostra disposizione per andare a modificare solo alcuni dei parametri del modello, mantenendo invariati quelli non considerati.
- Metodi ensemble: nel caso di metodi ensemble solitamente si utilizzano i nuovi dati per addestrare nuovi modelli da tenere in considerazione per la previsione finale (ad esempio nelle foreste randomiche si creano nuovi alberi utilizzando le nuove osservazioni) oppure si utilizzano per modificare i pesi utilizzati.

La scelta della strategia di utilizzare va compiuta tenendo in considerazione tre aspetti, oltre ovviamente allo scopo finale del problema specifico che stiamo studiando: la natura del modello (in quanto non tutte le tecniche viste possono essere applicate a priori), il costo computazionale della tecnica utilizzata e la quantità e la natura dei nuovi dati forniti.

1.7 Misure per i problemi di classificazione

Come ultimo spunto vediamo come testare la bontà e l'efficacia di un metodo di classificazione: solitamente si ricorre a una misura detta accuratezza, definita nel seguito.

Definizione 1.7.1 (Accuratezza)**.** Dato un metodo di classificazione, indicato con C il numero di predizioni corrette su un totale di T dati si definisce l'*accuratezza* come segue:

$$\text{Acc} = \frac{C}{T} \tag{1.5}$$

L'accuratezza assume valori compresi tra 0 ed 1: un modello perfetto assume accuratezza 1, mentre un modello che assegna in maniera totalmente casuale un oggetto a una classe assume accuratezza pari a 0.5. Nella pratica quindi si tende a scartare quei metodi che restituiscono una accuratezza sufficientemente lontana da 1.
Un altro strumento utile per stabilire l'efficacia di un metodo di classificazione è rappresentato dalle matrici di confusione.

Definizione 1.7.2 (Matrice di confusione)**.** Sia dato un metodo di classificazione statistico in cui abbiamo un numero K di classi come risposta e sia dato un insieme di osservazioni di cui conosciamo la classe di appartenenza. La *matrice di confusione* del metodo (o tabella di errata classificazione) è una matrice di dimensione $K \times K$ in cui l'elemento di posizione (i, j) è dato dal numero di elementi dell'insieme che appartengono (nella realtà) alla i-esima classe e che vengono classificati dal nostro modello nella j-ma classe.

Studiamo il caso dei problemi di classificazione binaria, che possiamo codificare tramite l'uso di una risposta positiva e una negativa. Indichiamo con TN il numero di osservazioni che vengono correttamente classificate come appartenti alla classe negativa (True Negative), con FP il numero di osservazioni (realmente) negative che vengono erroneamente classificate come positive (False Positive), con FN il numero di osservazioni (realmente) positive che vengono erroneamente classificate come negative (False Negative) e con TP il numero di osservazioni che vengono correttamente classificate come appartenti alla classe positiva (True Positive). La matrice di confusione di questo metodo è quindi la matrice M di dimensione 2×2 data da

$$M = \begin{pmatrix} TN & FP \\ FN & TP \end{pmatrix} \tag{1.6}$$

Capitolo 2

Classificatore Bayesiano e K-Nearest Neighbor

2.1 Introduzione

Dopo aver introdotto nel capitolo precedente una serie di nozioni base, come training e test error, cross-validation e accuratezza, iniziamo a studiare più nel dettaglio alcuni metodi di classificazione. Come accennato nell'Introduzione, in questo capitolo vedremo come affrontare il problema senza assumere ipotesi sulla forma sottostante le varie classi: otterremo il Classificatore Bayesiano, che costituisce la base degli altri metodi che studieremo, e K-Nearest Neighbor. Nel dedurre questi due metodi introdurremo in parallelo il problema della modifica del livello di soglia.

2.2 Classificatore Bayesiano

Cominciamo lo studio dei problemi di classificazione supervisionata presentando un metodo molto semplice ma al contempo molto potente, che non richiede ipotesi sulla forma dei predittori e delle classi: il Classificatore Bayesiano.
Indichiamo con G la variabile che restituisce l'output del nostro problema, ovvero $G(x)$ rappresenta la classe di appartenenza del campione x, con $\mathcal{G}$ l'insieme di tutte le possibili classi che può assumere il nostro output e con $\hat{G}$ la predizione che risulta dal nostro modello. L'idea alla base di una tecnica di Machine Learning consiste nel definire una funzione che vada a stimare l'errore del metodo, per poi andare a minimizzarla usando i dati a nostra disposizione, opportunamente divisi tra training e test (validation) data così come mostrato nella Sezione 1.2. Definiamo quindi una funzione che possa stimare l'errore commesso da un metodo di classificazione: si può ricorrere ad esempio all'uso di una matrice di perdita, che viene definita come segue.

Definizione 2.2.1. Sia K la cardinalità di $\mathcal{G}$. Una *matrice di perdita* $\mathbf{L}$ è una matrice di dimensione $K \times K$ costruita in modo che l'elemento $L(k, l)$ rappresenti il costo di classi-

ficare un oggetto che appartiene realmente alla classe $\mathcal{G}_k$ nella classe $\mathcal{G}_l$. Per convenzione si pone che gli elementi della matrice siano non-negativi.

Dalla Definizione 2.2.1 risulta immediato che la diagonale principale di $\mathbf{L}$ sarà costituita da zeri, in quanto la nostra predizione in questo caso risulta corretta. Dobbiamo trovare ora un modo di esprimere una funzione $L(k, l)$ che soddisfi le richieste della Definizione: tra le varie possibilità, che possono prendere in considerazione la possibilità di pesare maggiormente certe misclassificazioni, la scelta più comune consiste nell'uso della "zero-one loss function", funzione che assegna a ogni misclassificazione valore 1. Costruita la matrice $\mathbf{L}$, possiamo stimare l'errore previsto atteso del nostro metodo come segue.

Definizione 2.2.2. Data una matrice di perdita $\mathbf{L}$ si definisce *errore previsto atteso* la quantità

$$\mathrm{EPE} = \mathbb{E}\left[L(G, \hat{G}(X))\right] \tag{2.1}$$

dove il valore atteso è calcolato rispetto alla distribuzione congiunta $\mathbb{P}(G, X)$.

Per ottenere un buon classificatore cerchiamo quindi di minimizzare la quantità appena definita: conoscendo a priori la probabilità condizionata $\mathbb{P}(G|X)$ un metodo che risolve questo problema è il Classificatore Bayesiano, di seguito enunciato.

Metodo 1 (Classificatore Bayesiano)**.** Supponiamo di avere un problema di classificazione e di scegliere come misura di perdita la 0-1 loss function. Data un'osservazione x, la assegniamo alla classe in corrispondenza della quale la probabilità calcolata utilizzando la distribuzione condizionata (discreta) $\mathbb{P}(G|X)$ risulta massima. In formule:

$$\hat{G}(x) = \mathcal{G}_k \quad \text{dove } \mathbb{P}(\mathcal{G}_k|X = x) = \max_{g \in \mathcal{G}} \mathbb{P}(g|X = x) \tag{2.2}$$

Vale il seguente Teorema.

Teorema 2.2.1. *Il Classificatore Bayesiano è il metodo di classificazione in corrispondenza del quale l'errore previsto atteso risulta minimo.*

Dimostrazione. In primo luogo riscriviamo (2.1) calcolando il valore atteso condizionato a X. Indicando con $\mathbb{E}_X = \mathbb{E}[X]$ si ottiene che

$$\mathrm{EPE} = \mathbb{E}_X \sum_{k=1}^{K} L(\mathcal{G}_k, \hat{G}(X))\mathbb{P}(\mathcal{G}_k|X) \tag{2.3}$$

Per ottenere un minimo che sia globale, possiamo minimizzare (2.3) puntualmente: quindi a ogni input x assegniamo il valore $\hat{G}(x)$ tale che

$$\hat{G}(x) = \operatorname*{argmin}_{g \in \mathcal{G}} \sum_{k=1}^{K} L(\mathcal{G}_k, g)\mathbb{P}(\mathcal{G}_k|X = x) \tag{2.4}$$

Consideriamo il caso in cui utilizziamo la "0-1 loss function". Fissato g, il valore $L(\mathcal{G}_k, g)$ vale 0 solo quando g appartiene alla classe $\mathcal{G}_k$ nella realtà: quindi, supponendo che g nella realtà appartenga alla classe $\mathcal{G}_{i_g}$, possiamo riscrivere la sommatoria sopra come

$$\sum_{k=1}^{K} L(\mathcal{G}_k, g)\mathbb{P}\left(\mathcal{G}_k|X=x\right) = \sum_{k=1, k\neq i_g}^{K} \mathbb{P}\left(\mathcal{G}_k|X=x\right) = 1 - \mathbb{P}\left(\mathcal{G}_{i_g}|X=x\right)$$

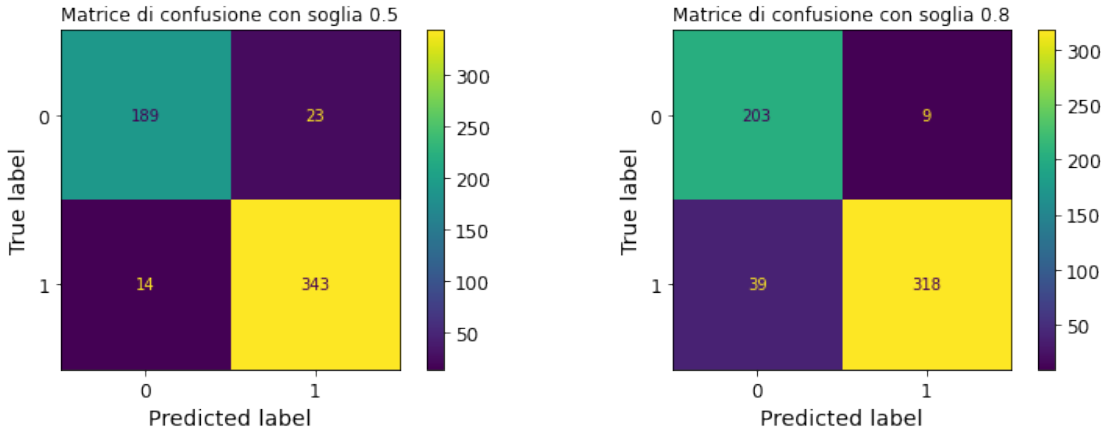
Il nostro problema diventa quindi equivalente a cercare

$$\hat{G}(x) = \underset{g\in\mathcal{G}}{\operatorname{argmin}} \left(1 - \mathbb{P}\left(g|X=x\right)\right) = \underset{g\in\mathcal{G}}{\operatorname{argmax}} \mathbb{P}\left(g|X=x\right)$$

che corrisponde al Classificatore Bayesiano. □

2.3 Modifica del livello di soglia

Studiamo ora il caso di un problema binario, ovvero un problema di classificazione con due possibili risposte: per semplicità le indicheremo con 1 e 2. Il classificatore Bayesiano consiste nell'assegnare all'osservazione x la classe 1 se $\mathbb{P}\left(1|X=x\right)$ è maggiore di $\mathbb{P}\left(2|X=x\right)$: siccome queste due probabilità devono sommare a 1, assegniamo alla prima classe il nostro input quando $\mathbb{P}\left(1|X=x\right) > 0.5$.
Questo approccio minimizza l'errore atteso totale, ma non tiene in considerazione la forma delle predizioni scorrette: in particolari contesti potremmo essere interessati al fatto che il numero di oggetti non correttamente assegnati alla prima classe sia molto basso, al costo di aumentare il numero di misclassificazioni nella seconda. Forniamo un esempio concreto: un creditore vuole analizzare se un cliente riuscirà a ripagare o meno un debito per decidere se concedergli un prestito: sarà quindi interessato al fatto che, effettuando la sua decisione sfruttando un metodo di classificazione, il numero di clienti che aveva predetto che sarebbero riusciti a ripagarlo, quando invece non riusciranno, sia molto basso, a scapito di non concedere denaro a chi nella realtà riuscirebbe a restituirlo, contrariamente a quanto predetto. Un modo per tenere in considerazione questo aspetto nel Classificatore Bayesiano consiste nel modificarne il livello di soglia, ovvero assegnare un dato x alla prima classe quando $\mathbb{P}\left(1|X=x\right) > \alpha$, dove $0 < \alpha < 1$ è chiamato appunto livello di soglia.
Vediamone una applicazione a dei dati reali: supponiamo di lavorare con il "Breast Cancer Dataset" [1] contenuto all'interno della libreria Python Scikit-Learn [8]. Questo dataset contiene i dati relativi a uno studio clinico effettuato su pazienti affette da cancro al seno, raccogliendone alcune features, quali la dimensione del tumore, e fornendo come categoria output la distinzione tra cancro benigno e maligno. Ci poniamo come obiettivo quello di fornire un modello che ci possa predirre questa distinzione: questo si può raggiungere ad esempio modellizzando la probabilità di malignità attraverso una regressione logistica (si veda il Capitolo 3 per maggiori dettagli su questo metodo) per poi applicare il Classificatore Bayesiano assegnando la classe a cui corrisponde probabilità maggiore. In particolare, vogliamo essere sicuri che la probabilità di comunicare a una paziente che è



(a) Matrice di confusione in corrispondenza di un valore di soglia pari a 0.5.

(b) Matrice di confusione in corrispondenza di un valore di soglia pari a 0.8.

Figura 2.1: Confronto sulle matrici di confusione derivanti dall'applicazione di una regressione logistica sul Breast Cancer Dataset in corrispondenza di diversi valori di soglia.

affetta da un tumore maligno, quando nella realtà ne presenta uno benigno, sia bassa, in modo da evitare di sottoporla a cure eccessivamente aggressive che potrebbero risultare dannose: in termini più formali, vogliamo tenere il numero di falsi positivi basso. Un modo di ottenere ciò consiste nel lavorare sul livello di soglia del Classificatore Bayesiano: utilizzando il semplice livello dato da una soglia pari a 0.5 otteniamo un numero di falsi positivi pari a 23 (si veda la Figura 2.1a). Andando ad aumentare il livello di soglia a 0.8 si ottiene un numero di falsi positivi pari a 9 (si veda la Figura 2.1b), ottenendo un risultato più in linea con quello che desideriamo. Osserviamo in conclusione che andare a modificare il livello di soglia non modifica sensibilmente l'accuratezza del nostro modello: in entrambi i casi abbiamo che è pari a circa 0.9, il che indica buone capacità predittive (si veda la Sezione 1.7 per ulteriori dettagli).

2.4 K-Nearest Neighbor

Come accennato in precedenza, il Classificatore Bayesiano, per essere applicato, richiede di conoscere la distribuzione condizionale $\mathbb{P}(G|X)$: generalmente questa non è una quantità nota a priori, quindi bisogna trovare un modo di modellizzarla. Nei capitoli successivi vedremo diversi modi per realizzare ciò: ad esempio la regressione lineare assume che la probabilità sia funzione di una funzione lineare nelle variabili fornite, mentre i metodi generativi assumono che abbia una particolare forma, nello specifico che sia una distribuzione normale.
Esiste però una strada ancora più semplice, data dal K-nearest neighbors (KNN), che stima la distribuzione condizionale senza nessuna ipotesi particolare. Dato un intero positivo K e una osservazione x_0, indichiamo con $\mathcal{N}_0$ l'insieme dei K training data che

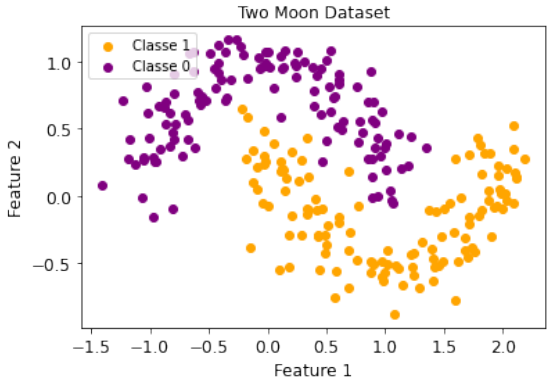


Figura 2.2: Two Moon Dataset generato a partire dalla funzione `make_moons` con parametri `n_samples` $= 250$, `noise` $= 0.15$ e `random_state` $= 0$

sono più vicini ad x_0 stesso. A questo punto stimiamo la probabilità condizionata di appartenenza alla j-ma classe come

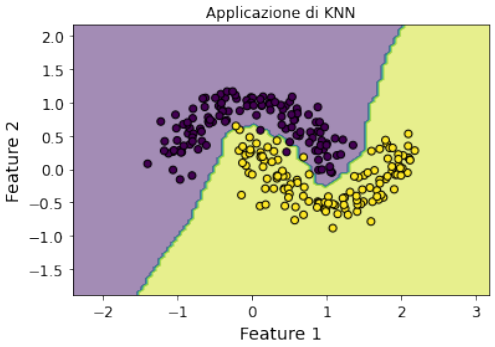
$$\mathbb{P}(G = j|X = x_0) = \frac{1}{K} \sum_{i \in \mathcal{N}_0} I(y_i = j) \tag{2.5}$$

ovvero tramite la frequenza relativa della j-ma classe calcolata rispetto ai K oggetti più vicini ad x_0. Il metodo KNN consiste nell'assegnare a x_0 la classe per cui (2.5) risulta massima.
Restano due problemi da analizzare. Il primo consiste nella scelta di una opportuna misura che quantifichi la distanza tra gli oggetti: solitamente si ricorre semplicemente al quadrato della distanza euclidea, ma questa si può modificare in base alle esigenze specifiche dei dati a disposizione. Il secondo problema consiste nell'individuare il numero di oggetti K da considerare come facenti parte dell'intorno della nostra osservazione: generalmente questo viene scelto tramite una validazione incrociata a 5 (o a 10) fold. Nello specifico, si individua una serie di valori $K_1, \dots, K_{\max}$ che può assumere il nostro parametro, si stima il test error di K_iNN tramite la validazione incrociata e si sceglie come valore per il parametro quello per cui si ottiene una migliore accuratezza finale. Osserviamo che generalmente utilizzare $K = 1$ porterà al migliore risultato in termini di training error: tuttavia, nel momento in cui vogliamo andare a testare il modello su nuovi dati, a livello di test error troveremo risultati che in generale non saranno molto buoni. Per questo motivo si tende a scartare a priori questo valore di K.

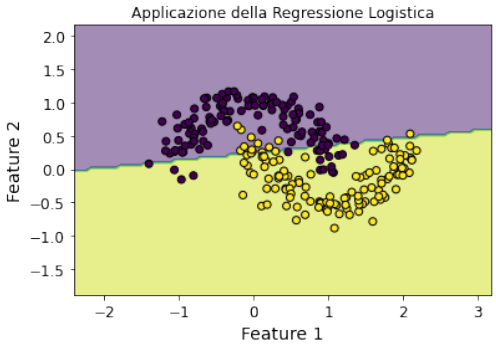
2.4.1 Una applicazione del metodo KNN

Vediamo ora un esempio di applicazione del metodo KNN. Lavoriamo con la libreria di Python "Scikit-learn" [8] e generiamo un dataset giocattolo a partire dalla funzione `make_moons` [9] (si veda la Figura 2.2 per una visualizzazione grafica dei dati). Sfruttando funzioni implementate in questa libreria, possiamo addestrare un modello KNN su questi

dati, usando come metrica la distanza euclidea e come numero di osservazioni nell'intorno $K = 5$. Otteniamo i risultati nella Figura 2.3. Per ottenere un paragone possiamo confrontare questo modello con quello ottenuto utilizzando una regressione logistica: spezzando il nostro dataset in training e test data otteniamo una accuratezza pari a 0.987 con KNN e 0.86 con la regressione logistica. Si può notare come la tecnica qui presentata sia un ottimo modo di procedere quando le ipotesi per gli altri metodi non sono soddisfatte: infatti, nell'esempio appena visto, le due classi non sono linearmente separabili, quindi l'ipotesi che costituisce la base della regressione logistica viene meno (si veda il Capitolo 3 per capire cosa questo significa). Come ulteriore aspetto sottolineiamo che KNN non rappresenta sempre il miglior metodo da utilizzare come alternativa ad altre tecniche: infatti, come accennato nella Sezione precedente, dobbiamo avere a disposizione dei dati la cui rappresentazione in $\mathbb{R}^n$, corredata da metrica euclidea, risulti coerente con quanto vogliamo studiare. Ad esempio, questa tecnica non risulta efficace nel cogliere le differenze tra le varie classi quando analizziamo problemi che presentano una elevata dimensionalità, ovvero un grande numero di predittori.
Vediamo infine un ultimo spunto, ovvero il problema della rinormalizzazione dei dati: questa è una tecnica molto utilizzata nella fase di implementazione ed esecuzione di un modello di Machine Learning, e consiste nello scalare i dati in modo che abbiano media 0 e varianza 1 oppure in modo che assumano valori in un determinato intervallo, solitamente $[0, 1]$. In generale viene utilizzata in quanto avere dati normalizzati consente agli algoritmi di essere eseguiti più velocemente e di ridurre il bias delle stime restituite. Osserviamo che questo strumento è particolarmente utile in tecniche che si basano sulla distanza, come ad esempio KNN: infatti, se nel problema che stiamo studiando compaiono features con scale diverse, quelle che assumono valori più grandi risulteranno molto più influenti sul risultato finale e questo potrebbe andare a restituire un modello poco performante. Nonostante possa sembrare che sia sempre utile eseguirla, la rinormalizzazione presenta anche dei lati negativi: infatti, nel caso in cui la differenza di scala tra le diverse features sia significativa, scalare i dati risulta dannoso. Per esempio, se stiamo lavorando con dei dati che fanno riferimento a qualche prezzo (espresso nella stessa valuta), il fatto che qualche feature assuma valori significativamente più elevati di altre può essere frutto di una caratteristica intrinseca del problema, che quindi non va eliminata. Inoltre, in tecniche come Naive Bayes (si veda la Sezione 4.5), rinormalizzare i dati può portare a modificare la condizione di indipendenza necessaria per la validità di questo particolare modello.



(a) Implementazione del metodo KNN con $K = 5$.



(b) Implementazione di una regressione logistica.

Figura 2.3: Confronto di KNN e regressione logistica sul Two Moon dataset eseguito a partire dai valori `n_samples`=250, `noise`=0.15, `random_state`=0. La divisione in training e test data è stata eseguita utilizzando la funzione `train_test_split` con parametri `test_size` = 0.7 e `random_state` = 10. Le due regioni che si ottengono dai metodi sono colorate con due colori diversi.

Capitolo 3

Regressione Logistica

3.1 Introduzione

Come già accennato nella Sezione 2.4, per poter applicare il Classificatore Bayesiano abbiamo bisogno di conoscere la distribuzione di probabilità delle classi condizionata rispetto ai predittori: uno dei metodi più utilizzati per stimarla è dato dalla regressione logistica, che andiamo a studiare in questo capitolo.

3.2 Il Modello Logistico

3.2.1 Caso di un solo predittore

Iniziamo la nostra analisi studiando il caso in cui abbiamo solamente due classi come risposta e un unico predittore X. Al posto di predire direttamente la classe di appartenenza di un input x, la regressione logistica (e in generale una buona parte dei metodi che studieremo) consiste nel fornire un modello che predica la probabilità che l'output Y appartenga a una determinata classe: calcolato questo valore in corrispondenza di ognuna delle classi del problema, applicando il Classificatore Bayesiano assegniamo all'osservazione la classe che è risultata essere più probabile. La prima idea che potremmo utilizzare per approcciare questo problema è modellizzare la probabilità usando una funzione lineare $h(X)$, ovvero

$$p(X) = h(X) = \beta_0 + \beta_1 X$$

Questo approccio presenta tuttavia una problematica di base: con questo modello possiamo predirre quantità negative o maggiori di 1, e volendole interpretare come probabilità questo risulta un controsenso. Un modo per risolvere questo inconveniente consiste nell'applicare alla nostra funzione lineare $h(X)$ una funzione che assuma valore compreso tra 0 ed 1: la scelta più comune ricade su una funzione, che è nota come funzione logistica. Questa è una funzione di classe $\mathcal{C}^\infty$, e quindi facilmente derivabile più volte, che, come richiesto, assume valori compresi tra 0 e 1: in particolare, questi non vengono mai

raggiunti, ma la funzione vi tende asintoticamente (rispettivamente per valori di t che tendono a $-\infty$ e $+\infty$). L'espressione di questa funzione è data da

$$\sigma(t) = \frac{e^t}{1 + e^t} \tag{3.1}$$

Seguendo questo ragionamento modellizziamo quindi la probabilità tramite una funzione data dalla composizione della funzione logistica con una funzione lineare, ovvero

$$p(X) = h(\sigma(X)) = \frac{e^{\beta_0+\beta_1 X}}{1 + e^{\beta_0+\beta_1 X}} \tag{3.2}$$

Nel seguito, in particolare nella Sezione 4.6, sarà utile fare riferimento alle seguenti quantità, ottenibili tramite un conto molto semplice, date da

$$\frac{p(X)}{1 - p(X)} = e^{\beta_0+\beta_1 X}$$

che vengono dette comunemente odds: un valore degli odds vicino a 0 è frutto di una probabilità $p(X)$ molto bassa, mentre un valore che tende a ∞ indica una probabilità $p(X)$ molto alta, pari a 1. Inoltre, quando siamo in presenza di un caso di equilibrio, ovvero nel caso in cui $p(X) = 0.5$, otteniamo un valore degli odds pari a 1. Calcolandone il logaritmo otteniamo quantità che variano tra $-\infty$ e $+\infty$, i cosiddetti log-odds o logit, che sono dati dalla formula

$$\log \frac{p(X)}{1 - p(X)} = \beta_0 + \beta_1 X \tag{3.3}$$

Queste quantità possono tornare utili nello studio di come il predittore influenza la probabilità finale di appartenenza a una classe.

3.2.2 Caso di più predittori

Supponiamo ora di avere p predittori $X = (X_1, \ldots, X_p)$. Per analogia con il caso unidimensionale possiamo generalizzare l'equazione (3.2) al seguente modello:

$$p(X) = \frac{e^{\beta_0+\beta_1 X_1+\cdots+\beta_p X_p}}{1 + e^{\beta_0+\beta_1 X_1+\cdots+\beta_p X_p}} \tag{3.4}$$

3.2.3 Caso di più classi

Analizziamo infine il caso in cui abbiamo $K > 2$ classi come risposta e p predittori: ci sono diversi modi di estendere i due modelli visti finora. Una delle tecniche che viene utilizzate a tale scopo è la regressione logistica multinomiale: dopo aver selezionato arbitrariamente una classe che funga da classe base, solitamente la classe K, si modellizzano per ogni $k = 1, \ldots, K-1$ le probabilità di appartenenza come

$$\mathbb{P}(Y = k|X = x) = \frac{e^{\beta_{k0}+\beta_{k1}x_1,+\cdots+\beta_{kp}x_p}}{1 + \sum_{l=1}^{K-1} e^{\beta_{l0}+\beta_{l1}x_1+\cdots+\beta_{lp}x_p}}$$

mentre per la classe base possiamo utilizzare

$$\mathbb{P}(Y = K|X = x) = \frac{1}{1 + \sum_{l=1}^{K-1} e^{\beta_{l0} + \beta_{l1}x_1 + \cdots + \beta_{lp}x_p}}$$

A questo punto svolgendo semplici calcoli si ottiene che

$$\log \frac{\mathbb{P}(Y = k|X = x)}{\mathbb{P}(Y = K|X = x)} = \beta_{k0} + \beta_{k1}x_1, + \cdots + \beta_{kp}x_p \tag{3.5}$$

che può essere utilizzato come punto di partenza per ricavare i coefficienti del modello. Un'alternativa a questa via è data dal softmax coding, che consiste nel non selezionare nessuna classe base ma assumere per ogni $k = 1, \ldots, K$ un modello della forma

$$\mathbb{P}(Y = k|X = x) = \frac{e^{\beta_{k0} + \beta_{k1}x_1, + \cdots + \beta_{kp}x_p}}{\sum_{l=1}^{K-1} e^{\beta_{l0} + \beta_{l1}x_1 + \cdots + \beta_{lp}x_p}}$$

In questo modello otterremo log-odds pari a

$$\log \frac{\mathbb{P}(Y = k|X = x)}{\mathbb{P}(Y = k'|X = x)} = (\beta_{k0} - \beta_{k'0}) + (\beta_{k1} - \beta_{k'1})x_1, + \cdots + (\beta_{kp} - \beta_{k'p})x_p$$

3.3 Fare predizioni

Introdotto il modello logistico da un punto di vista formale, dobbiamo trovare delle tecniche che ci permettano di stimarne i parametri e assegnare un dato input a una certa classe. Studiamo inizialmente il secondo problema: supponiamo quindi di aver già ricavato i parametri del modello, ad esempio tramite i metodi che introdurremo nella Sezione 3.4: vediamo come fornire un adeguato output al nostro problema. Il modo che risulta più semplice e intuitivo è quello di seguire il principio alla base del classificatore Bayesiano che abbiamo studiato nel Capitolo 2: si stimano le probabilità di appartenenza a ciascuna classe utilizzando le formule viste nelle Sezioni precedenti e i parametri ottenuti, per poi scegliere la classe in corrispondenza della quale questo valore è massimo, eventualmente modificando il livello di soglia come già visto in precedenza.

3.4 Stima dei coefficienti della regressione logistica

Vediamo a questo punto come possiamo stimare i coefficienti del modello logistico: per realizzare ciò, solitamente si ricorre al principio di massima verosimiglianza, calcolata utilizzando la log-verosimiglianza della variabile output G condizionata ai predittori X. Per semplicità studieremo solo il caso in cui abbiamo due classi come risposta: in questo setting, se abbiamo N osservazioni la funzione di log-verosimiglianza è data da

$$l(\theta) = \sum_{i=1}^{N} \log p_{g_i}(x_i; \theta) \tag{3.6}$$

dove indichiamo con $p_k(x_i; \theta) = \mathbb{P}(G = k|X = x_i; \theta)$. In generale, se abbiamo a disposizione K classi, avremo semplicemente più parametri in (3.6) nel modellizzare la quantità $p_{g_i}(x_i; \theta)$.

3.4.1 Soluzione con il Metodo di Newton

Consideriamo il caso in cui abbiamo due classi, che codificheremo con una risposta $y_i = 0$ per la prima e $y_i = 1$ per la seconda. Indichiamo con $p(x, \theta) = p_1(x, \theta)$ e quindi con $1 - p(x, \theta) = p_0(x, \theta)$. Studiamo il modello logistico (3.4): consideriamo β nella forma vettoriale $\beta = (\beta_0, \dots, \beta p)$ e assumiamo che i vettori x_i contengano una costante 1 al primo termine per tenere in considerazione la presenza dell'intercetta nella regressione. Vale il seguente risultato.

Proposizione 3.4.1. Sia dato $b \in \mathbb{R}$. La stima dei coefficienti di (3.4) attraverso il Metodo di Newton è data da:

$$\begin{cases} \beta_0 = b \\ \beta_{k+1} = \beta_k - \left(\frac{\partial l(\beta)}{\partial \beta \partial \beta^T}\right)^{-1}\Big|_{\beta_k} \cdot \frac{\partial l(\beta)}{\partial \beta}\Big|_{\beta_k} \end{cases} \tag{3.7}$$

Dimostrazione. In primo luogo possiamo notare che

$$1 - p(x_i, \beta) = 1 - \frac{e^{\beta^T x_i}}{1 + e^{\beta^T x_i}} = \frac{1}{1 + e^{\beta^T x_i}}$$

Sfruttando ciò e le proprietà dei logaritmi possiamo riscrivere (3.6) come segue:

$$\begin{aligned} l(\beta) &= \sum_{i=1}^{N} \left[y_i \log p(x_i, \beta) + (1 - y_i) \log\left(1 - p(x_i, \beta)\right)\right] \\ &= \sum_{i=1}^{N} \left[y_i \log p(x_i, \beta) - y_i \log\left(1 - p(x_i, \beta)\right) + \log\left(1 - p(x_i; \beta)\right)\right] \\ &= \sum_{i=1}^{N} \left[y_i \log \frac{p(x_i, \beta)}{1 - p(x_i, \beta)} + \log \frac{1}{1 + \exp\{\beta^T x_i\}}\right] \\ &= \sum_{i=1}^{N} \left[y_i \beta^T x_i - \log\left(1 + \exp\{\beta^T x_i\}\right)\right] \end{aligned} \tag{3.8}$$

A questo punto condizione necessaria per ottenere un massimo di (3.8) è che la derivata sia nulla, ovvero

$$\frac{\partial l(\beta)}{\partial \beta} = \sum_{i=1}^{N} x_i (y_i - p(x_i, \beta)) = 0 \tag{3.9}$$

Otteniamo quindi $p + 1$ equazioni non lineari in β. Vogliamo risolverle numericamente attraverso il metodo di Newton-Raphson: calcoliamone quindi le derivate (che corrispondono alla matrice Hessiana della funzione log-verosimiglianza)

$$\frac{\partial l(\beta)}{\partial \beta \partial \beta^T} = -\sum_{i=1}^{N} x_i x_i^T p(x_i, \beta)(1 - p(x_i, \beta)) \tag{3.10}$$

Dato un passo β^{old} possiamo aggiornare la stima dei nostri parametri come segue:

$$\beta^{\text{new}} = \beta^{\text{old}} - \left(\frac{\partial l(\beta)}{\partial\beta\partial\beta^T}\right)^{-1}\Big|_{\beta^{\text{old}}} \cdot \frac{\partial l(\beta)}{\partial\beta}\Big|_{\beta^{\text{old}}} \tag{3.11}$$

Otteniamo quindi quanto enunciato nella tesi. □

A questo punto può tornare utile scrivere il tutto in forma matriciale per agevolarne l'implementazione in un linguaggio di programmazione. Siano $\mathbf{y}$ il vettore contenente come elementi i dati y_i, X la matrice di dimensione $N \times (p+1)$ contenente i vettori x_i come colonne, $\mathbf{p}$ il vettore contenente le probabilità fittate $p(x_i, \beta^{\text{old}})$ e W la matrice diagonale $N \times N$ con i-mo elemento (lungo la diagonale) dato dalla quantità $p(x_i, \beta^{\text{old}})(1 - p(x_i, \beta^{\text{old}}))$. Possiamo quindi riscrivere le quantità in (3.9) e (3.10) come segue:

$$\frac{\partial l(\beta)}{\partial\beta} = X^T(\mathbf{y} - \mathbf{p})$$

$$\frac{\partial l(\beta)}{\partial\beta\partial\beta^T} = -X^T W X$$

Quindi (3.7) diventa

$$\begin{aligned}\beta^{\text{new}} &= \beta^{\text{old}} + (X^T W X)^{-1} X^T(\mathbf{y} - \mathbf{p}) \\ &= (X^T W X)^{-1} X^T W (X\beta^{\text{old}} + W^{-1}(\mathbf{y} - \mathbf{p})) \\ &= (X^T W X)^{-1} X^T W \mathbf{z}\end{aligned} \tag{3.12}$$

dove con $\mathbf{z} = X\beta^{\text{old}} + W^{-1}(\mathbf{y} - \mathbf{p})$ indichiamo la "adjusted response". Questo metodo è noto come "iteratively reweighted least squares" (o IRLS) in quanto ogni step risolve il problema

$$\beta^{\text{new}} \leftarrow \underset{\beta}{\text{argmin}}\, (\mathbf{z} - X\beta)^T W (\mathbf{z} - X\beta)$$

In generale si sceglie di utilizzare come punto di partenza per la risoluzione il valore $\beta_0 = 0$: a livello teorico potrebbe capitare di non arrivare a convergenza, ma da un punto di vista pratico si è osservato che questo algoritmo converge praticamente sempre, essendo la log-verosimiglianza una funzione concava. Questo procedimento può essere esteso al caso in cui abbiamo $K \geq 3$ classi, ma il tutto risulta molto complicato.

3.4.2 Proprietà della stima ricavata con Newton

Come notato in precedenza la stima data da (3.12) risulta essere frutto di un "least square" pesato: da questa connessione derivano delle proprietà interessanti.

- In primo luogo abbiamo che se il modello è corretto la nostra stima $\hat{\beta}$ è consistente (ovvero converge al vero valore di β).
- Si può mostrare tramite un teorema limite centrale che la distribuzione di $\hat{\beta}$ converge a una normale di media β e varianza $(X^T W X)^{-1}$.

- Essendo la procedura computazionalmente costosa, utilizzando questo collegamento possiamo applicare preventivamente test come il "Rao Score Test" o il "Wald test" che ci permettono di escludere efficientemente parametri che possono essere considerati trascurabili. In questo modo non abbiamo bisogno di addestrare il modello più volte.
- Infine, si possono sfruttare alcune delle proprietà del metodo "least sqare" per ottenere implementazioni della regressione logistica più veloci ed efficienti.

3.4.3 Soluzione con Stochastic Gradient Descent

Vediamo ora un diverso approccio risolutivo al problema. Consideriamo ancora il caso in cui abbiamo due classi come risposta, ma questa volta le codificheremo con una risposta $y_i = 1$ per la prima classe e $y_i = -1$ per la seconda. Studiamo nuovamente il modello logistico (3.4) e, come prima, consideriamo β nella forma vettoriale $\beta = (\beta_0, \dots, \beta p)$. Supponiamo di avere a disposizione N training data della forma (X_i, Y_i) per $i = 1, \dots, N$ che siano indipendenti tra loro (questa richiesta non è forte: basta avere un dataset in cui ogni osservazione è misurata in modo indipendente dalle altre). Assumiamo infine che i vettori X_i contengano una costante 1 al primo termine per tenere in considerazione la presenza dell'intercetta nella regressione e che siano della forma $X_i = (1, X_{i1}, \dots, X_{ip})$. Vale il seguente risultato.

Proposizione 3.4.2. Nel setting appena illustrato, risolvere (3.4) equivale a risolvere il seguente problema:

$$\min_{\beta \in \mathbb{R}^{p+1}} \sum_{i=1}^{N} \log\left(1 + e^{y_i(\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip})}\right) \tag{3.13}$$

Dimostrazione. Sia $p(x, \theta) = p_{-1}(x, \theta)$ e quindi $p_1(x, \theta) = 1 - p(x, \theta)$. A questo punto la probabilità congiunta totale è data da:

$$\mathbb{P}\left(Y_1 = y_1, \dots, Y_N = y_N | X_1, \dots, X_N\right) = \prod_{i=1}^{N} p_i^{\frac{1}{2}(1-y_i)} (1 - p_i)^{\frac{1}{2}(y_i+1)} \tag{3.14}$$

dove abbiamo indicato con p_i la quantità

$$\begin{aligned} p_i = \mathbb{P}\left(Y_i = -1 | X_i\right) &= \frac{e^{\beta_0 + \beta_1 X_{i1} + \cdots + \beta_p X_{ip}}}{1 + e^{\beta_0 + \beta_1 X_{i1} + \cdots + \beta_p X_{ip}}} \\ &= \frac{1}{1 + e^{-(\beta_0 + \beta_1 X_{i1} + \cdots + \beta_p X_{ip})}} \end{aligned}$$

Sfruttando questa uguaglianza e (3.15) possiamo calcolare la log-verosimiglianza (3.6) come segue:

$$l(\beta) = \sum_{i=1}^{N} \frac{1}{2}(1 - y_i) \log\left(p_i\right) + \frac{1}{2}(y_i + 1) \log\left(1 - p_i\right) \tag{3.15}$$

A questo punto osserviamo che per ogni $i = 1, \dots, N$ vale che

$$\begin{aligned} 1 - p_i &= 1 - \frac{1}{1 + e^{-(\beta_0 + \beta_1 X_{i1} + \cdots + \beta_p X_{ip})}} \\ &= \frac{e^{-(\beta_0 + \beta_1 X_{i1} + \cdots + \beta_p X_{ip})}}{1 + e^{-(\beta_0 + \beta_1 X_{i1} + \cdots + \beta_p X_{ip})}} \\ &= \frac{1}{1 + e^{\beta_0 + \beta_1 X_{i1} + \cdots + \beta_p X_{ip}}} \end{aligned}$$

e quindi possiamo riscrivere

$$\begin{aligned} &\frac{1}{2}(1 - y_i)\log(p_i) + \frac{1}{2}(y_i + 1)\log(1 - p_i) \\ &= \begin{cases} \log\left(\frac{1}{1+e^{\beta_0 + \beta_1 X_{i1} + \cdots + \beta_p X_{ip}}}\right) & \text{se } y_i = 1 \\ \log\left(\frac{1}{1+e^{-(\beta_0 + \beta_1 X_{i1} + \cdots + \beta_p X_{ip})}}\right) & \text{se } y_i = -1 \end{cases} \\ &= -\log\left(1 + e^{y_i(\beta_0 + \beta_1 X_{i1} + \cdots + \beta_p X_{ip})}\right) \end{aligned}$$

La log-verosimiglianza (3.15) è quindi data da

$$l(\beta) = -\sum_{i=1}^{N} \log\left(1 + e^{y_i(\beta_0 + \beta_1 X_{i1} + \cdots + \beta_p X_{ip})}\right) \tag{3.16}$$

Applicare lo stimatore di massima verosimiglianza equivale quindi a massimizzare (3.16), ovvero a minimizzare la quantità

$$L(\beta) = \sum_{i=1}^{N} \log\left(1 + e^{y_i(\beta_0 + \beta_1 X_{i1} + \cdots + \beta_p X_{ip})}\right) \tag{3.17}$$

In conclusione, avendo a disposizione delle realizzazioni delle osservazioni $(x_1, y_1), \dots, (x_N, y_N)$ il problema può essere espresso come

$$\min_{\beta \in \mathbb{R}^{p+1}} \sum_{i=1}^{N} \log\left(1 + e^{y_i(\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip})}\right) \qquad \square$$

Grazie a questa Proposizione possiamo risolvere (3.13) per ottenere una stima dei coefficienti della regressione logistica: siccome la funzione da minimizzare è convessa e β-regolare[1], si applica un metodo di risoluzione noto come stochastic gradient descent, metodo che consente di trovare un vettore β^*, che corrisponde a un argomento del minimo della funzione obiettivo che vogliamo ottimizzare. In generale questa è una buona tecnica risolutiva, ma presenta come problema il fatto che rischia di restituire dei minimi che sono solo locali e non globali. A prima vista quindi questa riscrittura sembra inutile: la sua forza viene in campo nell'applicazione delle reti neurali, strumento molto utile quando abbiamo a disposizione molti dati e un modello sottostante difficilmente catturabile tramite un modello lineare (o in generale polinomiale). Per maggiori informazioni si vedano [5], [7] e [4].

[1] Una funzione $f : \mathbf{R}^d \to \mathbf{R}$ si dice β-regolare se il gradiente di f è una funzione β-Lipschitz, ovvero $\|\nabla f(w) - \nabla f(v)\| \leq \beta \|w - v\|$

3.5 Comportamento rispetto all'uso di nuovi dati

Nello studio e implementazione di alcuni problemi reali può capitare di ricevere nuovi dati in una fase successiva all'addestramento iniziale, che era stato eseguito su dei dati diversi: ad esempio, nello studio di un modello medico potremmo ricevere una serie di informazioni iniziali con cui creare un modello grezzo e, in una fase successiva, una nuova serie di osservazioni su dei nuovi pazienti, che vogliamo utilizzare per aggiornare le stime precedentemente ottenute per avere un modello finale con migliore capacità predittiva.
Vediamo come si comporta la regressione logistica rispetto all'introduzione di nuovi dati da utilizzare in fase di addestramento: in riferimento alle tecniche accennate nella Sezione 1.6, possiamo ricorrere a retraining periodico, addestramento incrementale, apprendimento online o fine-tuning. Tuttavia la tecnica che si preferisce utilizzare, specialmente nel caso in cui usiamo il metodo di Newton come risolutore, è il retraining periodico: combiniamo i dati esistenti con quelli nuovi per andare ad addestrare nuovamente l'intero insieme dei coefficienti. La scelta può comunque essere cambiata in base alle esigenze specifiche del problema che stiamo studiando o alla forma dei nuovi dati che ci vengono forniti.
Osserviamo infine che, generalmente, il riaddestramento con le tecniche che abbiamo elencato risulta particolarmente oneroso da un punto di vista computazionale: ci chiediamo quindi se abbia senso eseguirlo. Abbiamo che la regressione logistica è un metodo abbastanza sensibile ai dati utilizzati nella fase di addestramento: quindi l'aggiunta di nuove osservazioni porta alla creazione di modelli che, sfruttando un numero maggiore e più vario di osservazioni, presentano una migliore capacità predittiva. Si crea quindi un effetto che mitiga il costo computazionale dell'operazione.

3.6 Implementazione ed esempi pratici

Giunti a questo punto risulta interessante vedere a livello pratico qualche esempio per capire come funzionano effettivamente i metodi presentati nel corso di questo capitolo: utilizzando come linguaggio di programmazione Python risulta molto semplice implementare il modello presentato nella sezione 3.4.1 (si veda [2] per i dettagli). Inoltre all'interno delle librerie Python ne esiste una, Scikit-Learn [8], che contiene funzioni per addestrare modelli di Machine Learning: questa risulterà utile per valutare la bontà dei risultati ottenuti.
Come primo spunto andiamo ad analizzare come si comporta il nostro codice rispetto a quello fornito da Scikit-Learn: utilizziamo come dataset di riferimento Iris Dataset [6], di cui prendiamo in considerazione la classe "Iris Virginica" in corrispondenza di larghezza e spessore del petalo, si veda la Figura 3.1. Osserviamo subito che i risultati ottenuti tramite il nostro modello sono identici a quelli ottenuti con quello preimplementato: l'unico punto a sfavore consiste nei tempi di esecuzione, in quanto la nostra implementazione viene eseguita in un tempo pari a un numero compreso tra 10 e 50 volte quello preimplementato. Siccome stiamo lavorando in un contesto con due predittori e due classi possiamo vedere graficamente il risultato: in particolare possiamo tracciare la retta che

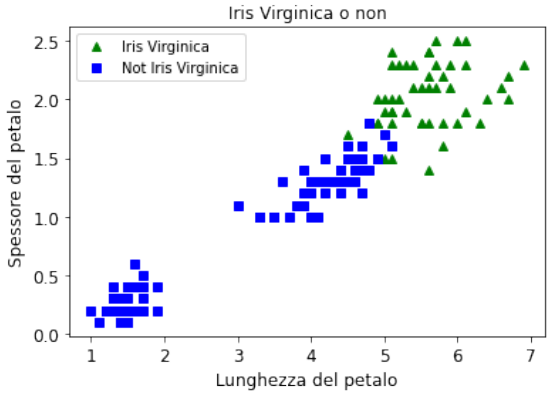


Figura 3.1: Iris Dataset.

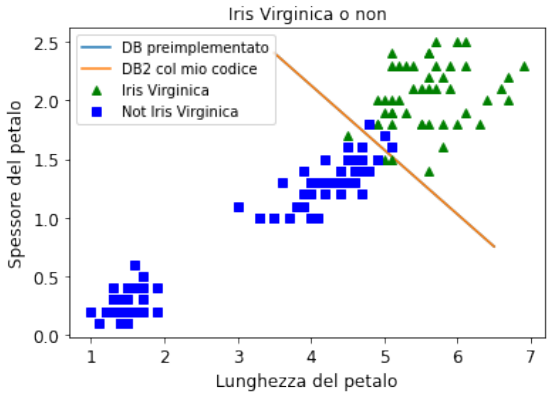


Figura 3.2: Confronto sul Decision Boundary ottenuto dalla regressione logistica implementata in Scikit-Learn con quello ottenuto usando l'implementazione in [2].

separa le due regioni dello spazio in cui assegniamo le due classi diverse. Otteniamo i risultati nella Figura 3.2: vediamo che effettivamente i due bordi di decisione coincidono, fatto che è confermato anche dall'uguaglianza dei coefficienti ottenuti eseguendo i due codici.
Passiamo ora ad analizzare un altro aspetto della regressione logistica: nello specifico vogliamo vedere come si comporta in presenza di un dataset sbilanciato. Supponiamo che, nello studio di un problema di classificazione binario, per ragioni che possono essere note o meno, ci venga fornito un dataset contenente una classe che risulta essere molto più numerosa dell'altra, quando nella realtà le due si equivalgono in termini di frequenze relative: per vederne un risultato concreto abbiamo creato un dataset fittizio usando la funzione `make_classification` di Scikit-Learn, si veda la Figura 3.3. Istruiamo dapprima la regressione logistica utilizzando un training set ottenuto tramite la funzione `train_test_split`, che divide in modo causale i dati a disposizione: la frequenza delle due classi in questo insieme è sostanzialmente la stessa. A questo punto, con una funzione

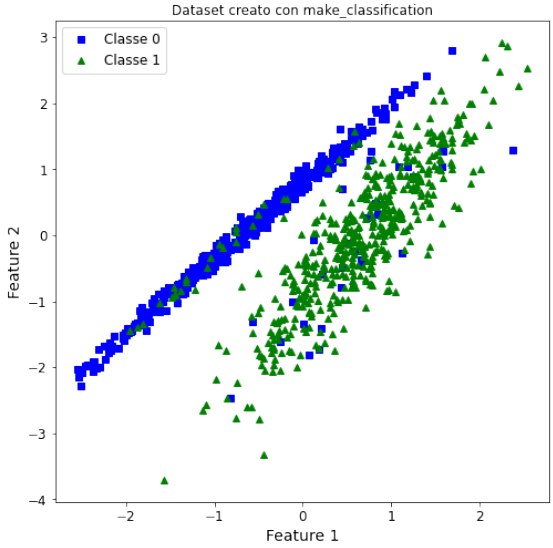
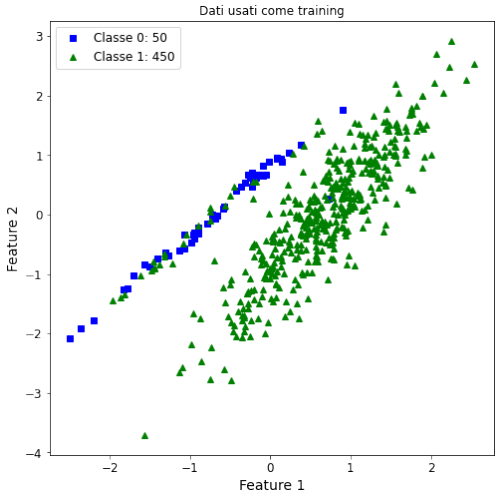
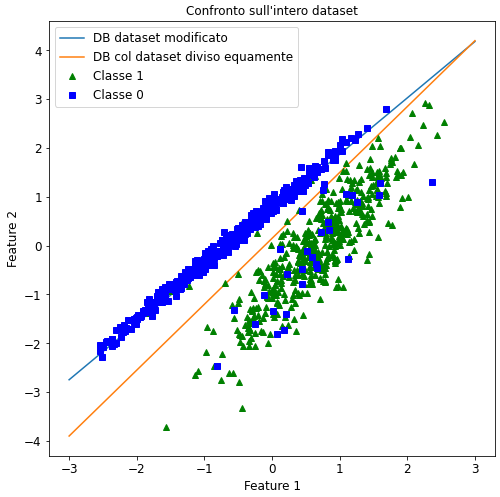


Figura 3.3: Dataset fittizio creato con la funzione `make_classification` con parametri `n_samples`=1000, `n_features`=2, `n_redundant`=0, `n_informative`=2, `n_repeated`=0, `n_clusters_per_class`=1, `flip_y`=0.1 e `random_state`=15.

che abbiamo creato appositamente a questo scopo, possiamo vedere cosa succede utilizzando come training dataset uno che contiene il 90% di oggetti appartenenti a una classe e il 10% dell'altra: otteniamo i risultati nelle Figure 3.4 e 3.5. Andando ad osservare in termini numerici i coefficienti ottenuti e a livello grafico la retta che separa le due regioni dello spazio a cui si assegnano le due classi possiamo quindi notare che la regressione logistica restituisce risultati che sono molto sensibili alle frequenze relative delle classi nei dati utilizzati in fase di addestramento: ecco quindi che, come avevamo già notato nella Sezione 3.5, quando entriamo in possesso di nuovi dati diventa fondamentale aggiornare il nostro modello, in quanto le osservazioni fornite inizialmente potrebbero non essere un buon campione rappresentativo dell'intera popolazione. Per i dettagli del codice e dei risultati ottenuti si veda ancora [2].

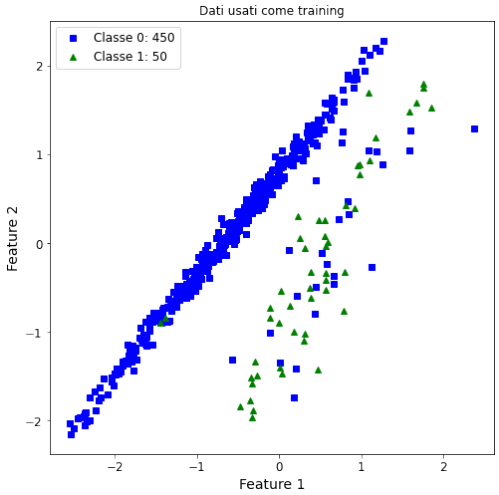


(a) Punti utilizzati in fase di training.

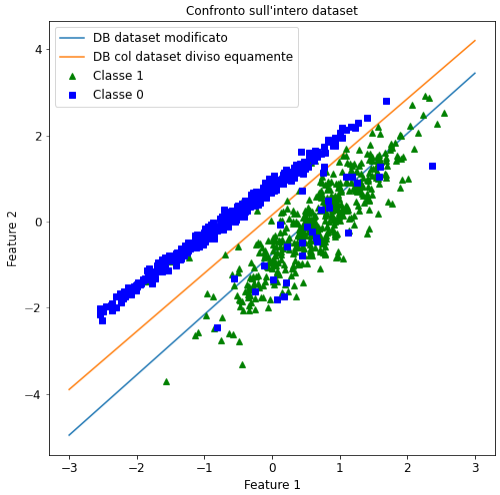


(b) Confronto dei Decision Boundary: in arancione quello ottenuto usando training data equidistribuiti, in azzurro quello usando training data sbilanciati.

Figura 3.4: Addestramento della regressione logistica sul dataset in Figura 3.3 usando 50 punti della classe 0 e 450 della classe 1.



(a) Punti utilizzati in fase di training.



(b) Confronto dei Decision Boundary: in arancione quello ottenuto usando training data equidistribuiti, in azzurro quello usando training data sbilanciati.

Figura 3.5: Addestramento della regressione logistica sul dataset in Figura 3.3 usando 450 punti della classe 0 e 50 della classe 1.

Capitolo 4

Modelli generativi per la classificazione

4.1 Perché introdurre dei nuovi metodi

Come abbiamo visto nel capitolo precedente, la regressione logistica si pone come obiettivo di modellizzare direttamente la distribuzione condizionale della variabile output Y dati i predittori contenuti nel vettore input X tramite una funzione sostanzialmente lineare: questo metodo presenta però dei problemi, vediamone i principali.

- Talvolta la stima dei parametri ottenuti all'interno di una regressione logistica risulta instabile: come abbiamo visto in un esempio nella Sezione 3.6, i coefficienti ottenuti eseguendo il metodo possono variare quando i dati forniti per istruirlo non sono estratti in modo randomico dalla popolazione di origine. Inoltre, l'approccio che abbiamo presentato nella Sezione 3.4.1 presenta difficoltà nel momento in cui le due classi sono distinte e ben separate, in quanto risulta particolarmente problematico invertire la matrice $X^T W X$.
- Estendere la regressione logistica al caso di risposta con più classi è risultato artificioso e poco intuitivo: le tecniche che studieremo in questo capitolo risultano essere più semplici da estendere oltre il caso binario.
- Infine, se la distribuzione del vettore X è approssimativamente normale, l'approccio che analizzeremo in seguito risulta essere più accurato della regressione logistica.

Presentiamo quindi un nuovo approccio al problema della classificazione supervisionata, che si basa sul modellizzare la distribuzione di X per ogni classe. Supponiamo di avere K classi come risposta e indichiamo con π_k la probabilità a priori che una osservazione arbitraria appartenga alla classe k. Sia poi $f_k(X) = \mathbb{P}(X|Y = k)$ la funzione densità di X per un'osservazione che appartiene alla classe k. Applicando il Teorema di Bayes possiamo affermare che

$$p_k(x) := \mathbb{P}(Y = k|X = x) = \frac{\pi_k f_k(x)}{\sum_{l=1}^{K} \pi_l f_l(x)} \tag{4.1}$$

La quantità $p_k(x)$ viene detta probabilità a posteriori che il dato $X = x$ appartenga alla classe k. Questa quantità viene stimata direttamente nella regressione logistica: il nostro approccio consiste ora nel ribaltare il punto di vista fornendo una stima delle quantità π_k e $f_k(x)$, che servono come punto di partenza per calcolare la probabilità a posteriori a cui siamo veramente interessati.
Stimare π_k risulta semplice nel caso in cui abbiamo delle osservazioni casuali: basta utilizzare la frazione di osservazioni che arrivano dal training dataset e che appartengono alla classe k. Al contrario, fornire una stima di $f_k(x)$ risulta un compito arduo: analizzeremo nel seguito alcuni modi di farlo.
Una volta che abbiamo questi valori, possiamo utilizzarli per calcolare le quantità in (4.1) e, attraverso il Classificatore Bayesiano, possiamo predirre la classe di appartenenza di una data osservazione assegnando quella in corrispondenza della quale la probabilità a posteriori risulta maggiore.

4.2 Linear Discriminant Analysis con un unico predittore

Studiamo il caso in cui abbiamo un solo predittore, ovvero $p = 1$, e K classi come risposta. Supponiamo che $f_k(x)$ segua una legge normale, o Gaussiana, ovvero

$$f_k(x) = \frac{1}{\sqrt{2\pi\sigma_k^2}} \exp\left\{ -\frac{1}{2\sigma_k^2}(x - \mu_k)^2 \right\} \tag{4.2}$$

dove μ_k, σ_k^2 indicano la media e la varianza dei parametri nella classe k. A questo punto assumiamo anche che la varianza sia la stessa per ognuna delle classi, ovvero $\sigma_1^2 = \cdots = \sigma_K^2$. Otteniamo una stima delle probabilità a posteriori della seguente forma

$$p_k(x) = \frac{\pi_k \frac{1}{\sqrt{2\pi}\sigma} \exp\{-\frac{1}{2\sigma^2}(x - \mu_k)^2\}}{\sum_{l=1}^{K} \pi_l \frac{1}{\sqrt{2\pi}\sigma} \exp\{-\frac{1}{2\sigma^2}(x - \mu_l)^2\}} \tag{4.3}$$

Come spiegato precedentemente vogliamo utilizzare il Classificatore Bayesiano per assegnare un'osservazione alla classe più probabile: vogliamo quindi studiare quando la quantità $p_k(x)$ risulta massima. In primo luogo possiamo andare a semplificare il termine $1/\sqrt{2\pi}\sigma$; inoltre, essendo il denominatore di (4.3) comune a tutte le quantità $p_k(x)$, possiamo studiarne solo il numeratore per ottenere il massimo. A questo punto, essendo il logaritmo una funzione crescente, possiamo assegnare la classe in corrispondenza della quale $\log\left(\pi_k \exp\{-\frac{1}{2\sigma^2}(x - \mu_k)^2\}\right)$ risulta massimo. Utilizzando le proprietà dei logaritmi si ottiene facilmente che ciò equivale a cercare il k che rende massima la quantità

$$\delta_k(x) = x \cdot \frac{\mu_k}{\sigma^2} - \frac{\mu_k^2}{2\sigma^2} + \log \pi_k \tag{4.4}$$

Da un punto di vista pratico rimane ora il problema di fornire un modo per stimare le quantità $\mu_1, \ldots, \mu_K$, $\sigma^2, \pi_1, \ldots, \pi_K$, che a priori non sono note: una possibile soluzione è enunciata nel Metodo 2. Quanto visto prende il nome di "Linear Discriminant Analysis" o LDA.

Metodo 2 (LDA con un predittore)**.** Sia dato un problema di classificazione con K classi totali e un unico predittore e siano N il numero totale di training data e n_k il numero di osservazioni appartenenti alla classe k. Nelle ipotesi elencate in questa sezione, si possono stimare come segue le medie e la varianza:

$$\hat{\mu}_k = \frac{1}{n_k} \sum_{i:y_i=k} x_i \quad k = 1, \ldots, K \tag{4.5}$$

$$\hat{\sigma}^2 = \frac{1}{N-K} \sum_{k=1}^{K} \sum_{i:y_i=k} (x_i - \hat{\mu}_k)^2 \tag{4.6}$$

Per quanto riguarda i valori delle probabilità a priori $\pi_1, \ldots, \pi_K$ in assenza di ulteriori informazioni si usano semplicemente le stime date dalle proporzioni

$$\hat{\pi}_k = \frac{n_k}{N} \tag{4.7}$$

Il metodo LDA consiste nell'assegnare un dato x alla classe per cui (4.4) calcolato con queste stime risulta massimo.

4.3 Linear Discriminant Analysis con più predittori

Studiamo ora il caso in cui abbiamo p predittori, che raccogliamo all'interno del vettore $X = (X_1, \ldots, X_p)$, e K classi di risposta.
Per la nostra analisi faremo uso delle distribuzioni Gaussiane multidimensionali: vediamone un piccolo riassunto. Una Gaussiana multidimensionale è un vettore aleatorio X di dimensione p la cui funzione di densità è data da

$$f(x) = \frac{\exp\left\{-\frac{1}{2}(x-\mu)^T \mathbf{\Sigma}^{-1}(x-\mu)\right\}}{(2\pi)^{\frac{p}{2}}\sqrt{\det(\mathbf{\Sigma})}} \tag{4.8}$$

dove μ è un vettore di dimensione p e $\mathbf{\Sigma}$ è una matrice di dimensione $p \times p$ simmetrica e definita positiva (in questo modo ammette inversa e determinante non nullo). In particolare, il vettore μ rappresenta il vettore media di X mentre $\mathbf{\Sigma}$ ne rappresenta la matrice di covarianza: nel seguito scriveremo che $X \sim N(\mu, \mathbf{\Sigma})$. Possiamo osservare facilmente la seguente proprietà.

Proposizione 4.3.1. Dato un vettore aleatorio Gaussiano X, ogni sua componente X_k segue una distribuzione gaussiana (unidimensionale) di media μ_k e varianza $\mathbf{\Sigma}_{kk}$.

Dimostrazione. Questa proprietà segue dal fatto che, in generale, dato un vettore aleatorio Gaussiano X di media μ e matrice di covarianza $\mathbf{\Sigma}$ ogni suo sottovettore rimane un vettore Gaussiano, così come ogni combinazione lineare delle sue componenti lo è. Infatti se andiamo a calcolare la funzione caratteristica di X otteniamo che questa è data da

$$\Phi_X(t) = \mathbb{E}\left[e^{i\langle t, X\rangle}\right] = e^{i\langle t,\mu\rangle} e^{-\frac{1}{2}\langle t, \mathbf{\Sigma} t\rangle}$$

A questo punto è noto che se L è una matrice di dimensione $n \times p$ allora il vettore aleatorio $Y = LX$ assume funzione caratteristica $\Phi_Y(t) = \Phi_X(L^T t)$: dunque se X è un vettore aleatorio Gaussiano vale che

$$\Phi_Y(t) = \Phi_X(L^T t) = e^{i\langle t, L\mu\rangle} e^{-\frac{1}{2}\langle t, L\mathbf{\Sigma}L^T t\rangle}$$

ovvero il vettore Y è un vettore Gaussiano di media $L\mu$ e matrice di covarianza $L\mathbf{\Sigma}L^T$. Presa quindi come matrice L quella che rappresenta la proiezione sulla k-esima componente si ottiene quanto abbiamo affermato. □

Tornando alla nostra analisi, nel caso generale di p predittori e K classi di risposta, la Linear Discriminant Analysis assume che le osservazioni nella classe k siano osservate da una distribuzione Gaussiana multidimensionale $N(\mu_k, \mathbf{\Sigma})$, dove μ_k indica il vettore media della classe k mentre la matrice di covarianza $\mathbf{\Sigma}$ è comune a tutte le classi. Ragionando in maniera analoga a quanto fatto per il caso con un unico predittore si può estendere il Metodo 2 al caso con p predittori, ottenendo il seguente metodo.

Metodo 3 (LDA con p predittori)**.** Nelle ipotesi del Metodo 2, ma con p predittori, il metodo LDA assegna a un'osservazione x la classe per cui la seguente quantità risulta massima:

$$\delta_k(x) = x^T \mathbf{\Sigma}^{-1}\mu_k - \frac{1}{2}\mu_k^T \mathbf{\Sigma}^{-1}\mu_k + \log \pi_k \tag{4.9}$$

Per stimare medie, matrice di covarianza e probabilità a priori si utilizzano formule analoghe a quelle mostrate nel Metodo 2, modificando (4.6) come segue:

$$\hat{\mathbf{\Sigma}} = \sum_{k=1}^{K} \sum_{i:y_i=k} \frac{(x_i - \hat{\mu}_k)(x_i - \hat{\mu}_k)^T}{N - K} \tag{4.10}$$

Come osservato in precedenza, una volta ottenute queste stime, possiamo applicare il Classificatore Bayesiano e assegnare un'osservazione alla classe in corrispondenza della quale otteniamo probabilità a posteriori massima. Per quanto visto nel Capitolo 2 otteniamo in questo modo la classificazione con l'errore previsto atteso (totale) più basso possibile: rimane però un problema, in quanto non abbiamo ulteriori informazioni sulle classi nelle quali gli errori sono accaduti nè tantomeno sulla tipologia di errore commesso (potrebbero essere avvenuti tutti all'interno della stessa classe, potrebbe essere stato assegnato a tutte le misclassificazioni lo stesso valore e così via). Come già notato nella Sezione 2.3 ci sono alcuni casi in cui risulta molto importante tenere conto anche di questi dettagli: in analogia a quanto visto in quella stessa sezione, possiamo andare a tenere conto di questo aspetto andando a modificare il livello di soglia utilizzato per assegnare una osservazione a una data classe.

4.4 Quadratic Discriminant Analysis

Nella Linear Discriminant Analysis abbiamo sempre supposto che la matrice di covarianza fosse comune alle varie classi di risposta: ovviamente questa assunzione non può essere

sempre rispettata nella realtà. Presentiamo ora un metodo che cerca di rilassare questa ipotesi.

Metodo 4 (QDA)**.** Supponiamo di lavorare sotto le ipotesi del Metodo 3, con l'unica differenza che assumiamo che ogni classe segua una distribuzione gaussiana $N(\mu_k, \mathbf{\Sigma}_k)$ in cui sia il vettore media μ_k che la matrice di covarianza $\mathbf{\Sigma}_k$ risultano essere diversi da classe a classe. La Quadratic Discriminant Analysis (QDA) consiste nell'assegnare un'osservazione $X = x$ alla classe per cui il seguente valore risulta massimo:

$$\begin{aligned}\delta_k(x) &= -\frac{1}{2}(x-\mu_k)^T\mathbf{\Sigma}_k^{-1}(x-\mu_k) - \frac{1}{2}\log|\mathbf{\Sigma}_k| + \log\pi_k \\ &= -\frac{1}{2}x^T\mathbf{\Sigma}_k^{-1}x + x^T\mathbf{\Sigma}_k^{-1}\mu_k - \frac{1}{2}\mu_k^T\mathbf{\Sigma}_k^{-1}\mu_k - \frac{1}{2}\log|\mathbf{\Sigma}_k| + \log\pi_k\end{aligned} \tag{4.11}$$

A questo punto basta fornire una stima delle medie, delle matrici di covarianza e delle probabilità a priori per applicare il metodo.

Osserviamo che, nel caso in cui abbiamo p predittori, il metodo QDA richiede di stimare $\frac{Kp(p+1)}{2}$ parametri, mentre LDA richiede di stimarne "solo" Kp: vediamo perchè in certi contesti risulta più ragionevole utilizzare la Quadratic Discriminant Analysis al posto della Linear, nonostante sia a livello computazionale più costosa. Da un lato, avendo meno parametri liberi da calcolare, il metodo LDA risulta avere una varianza più contenuta del metodo QDA, ovvero ripetendo l'addestramento su diversi sottoinsiemi estratti casualmente dal nostro dataset i risultati tenderanno a essere molto vicini tra loro: tuttavia, nel momento in cui l'ipotesi di matrice di covarianza comune alle varie classi viene meno, la Linear Discriminant Analysis restituisce risultati che presentano un bias elevato, ovvero una distanza dai parametri reali considerevole. Per questo motivo preferiamo utilizzare la Quadratic Discriminant Analysis o in presenza di un dataset di dimensioni considerevoli (che riduce l'effetto negativo dell'alta varianza) o se l'assunzione di una matrice di covarianza comune è debole. Inoltre, siccome la QDA presenta una natura quadratica nei predittori (si veda la Sezione 4.6 per capire in che senso), nel caso in cui ci troviamo di fronte a un modello quadratico implementare questo metodo risulta più efficiente che implementare una versione di LDA opportunamente modificata per superare la linearità.

4.5 Naive Bayes

Vediamo ora un ultimo modo per stimare le densità $f_k(x)$ in (4.1): il Naive Bayes.

Metodo 5 (Naive Bayes)**.** Sia dato un problema di classificazione in cui abbiamo K classi come risposta. Il Metodo Naive Bayes consiste nell'assumere che all'interno della classe k i p predittori siano indipendenti: quindi

$$f_k(x) = f_{k1}(x) \times f_{k2}(x) \times \cdots \times f_{kp}(x) \tag{4.12}$$

dove f_{kj} indica la funzione densità del j-mo predittore all'interno della classe k. Possiamo quindi assegnare un'osservazione x alla classe per cui (4.1) risulta massima, ovvero alla classe che massimizza

$$\mathbb{P}(Y = k|X = x) = \frac{\pi_k \times f_{k1}(x) \times \cdots \times f_{kp}(x)}{\sum_{l=1}^{K} \pi_l \times f_{l1}(x) \times \cdots \times f_{lp}(x)} \tag{4.13}$$

Rimane ora da definire un modo per stimare le densità a livello dei predittori: vediamone alcune alternative.

- Se la variabile X_j è quantitativa, possiamo assumere che, al variare della classe k, $X_j|Y = k \sim N(\mu_{jk}, \sigma^2_{jk})$ e fornire una stima di media e varianza a partire dai nostri dati, seguendo formule analoghe a quelle viste nel caso della Quadratic Discriminant Analysis.

- Sempre nel caso in cui X_j sia una variabile quantitativa, si possono sfruttare gli istogrammi per stimarne la densità. In particolare, realizziamo degli istogrammi per ognuna delle classi in corrispondenza di ciascuno dei predittori. Fissata la classe k, avendo a disposizione una osservazione x_j a livello del j-mo predittore, stimiamo $f_{kj}(x_j)$ come la frazione di osservazioni nella classe k che ricadono nello stesso istogramma in cui cade x_j.

- Infine se lavoriamo con una variabile X_j di tipo qualitativo possiamo stimare f_{kj} tramite la proporzione di osservazioni usate in fase di training che appartengono alla classe k al variare delle categorie del predittore.

Osserviamo che l'ipotesi di indipendenza risulta computazionalmente molto efficiente in quanto elimina la necessità di dover cercare di quantificare l'associazione tra due diversi predittori: infatti, richiedendo questa proprietà andiamo a imporre che la distribuzione congiunta dei due predittori sia "nulla". Nel caso di una Gaussiana multidimensionale, questo vuol dire che gli elementi che non stanno sulla diagonale della matrice di covarianza $\mathbf{\Sigma}$ siano nulli: nel caso di p predittori avremo quindi bisogno di stimare solo i p elementi diagonali di questa matrice, e ciò rende le operazioni richieste per addestrare il modello computazionalmente più veloci. Sebbene questo fatto sia molto utile, nella realtà questa assunzione non è quasi mai verificata: nonostante ciò, si è notato che spesso questo metodo porta comunque a buoni risultati, soprattutto quando il numero di osservazioni totali N è relativamente vicino a quello dei predittori p. Questo avviene perchè assumere l'indipendenza dei predittori porta a una diminuzione della varianza del problema, in quanto diminuiscono i parametri da stimare: tuttavia così facendo aumentiamo il bias della nostra stima. Dobbiamo quindi valutare attentamente cosa siamo interessati a studiare.

4.6 Confronto tra i vari metodi presentati

Vediamo ora quali sono le differenze tra i vari metodi che abbiamo presentato, nell'ipotesi in cui stiamo studiando un problema con K classi come risposta. Volendo utilizzare il Classificatore Bayesiano, andiamo ad assegnare ogni osservazione alla classe

che massimizza $\mathbb{P}(Y = k|X = x)$: tale classe corrisponde quindi anche alla classe che massimizza

$$\log\left(\frac{\mathbb{P}(Y = k|X = x)}{\mathbb{P}(Y = K|X = x)}\right)$$

Vediamo metodo per metodo come si comporta questa quantità. In primo luogo per la regressione logistica abbiamo già calcolato in precedenza questo valore, ottenendo come risultato (3.5).
Per quanto riguarda la Linear Discriminant Analysis possiamo mostrare che

$$\begin{aligned}
&\log\left(\frac{\mathbb{P}(Y = k|X = x)}{\mathbb{P}(Y = K|X = x)}\right) \\
&= \log\frac{\pi_k f_k(x)}{\pi_K f_K(x)} \\
&= \log\frac{\pi_k \exp\{-\frac{1}{2}(x-\mu_k)^T\mathbf{\Sigma}^{-1}(x-\mu_k)\}}{\pi_K \exp\{-\frac{1}{2}(x-\mu_K)^T\mathbf{\Sigma}^{-1}(x-\mu_K)\}} \\
&= \log\frac{\pi_k}{\pi_K} - \frac{1}{2}(x-\mu_k)^T\mathbf{\Sigma}^{-1}(x-\mu_k) + \frac{1}{2}(x-\mu_K)^T\mathbf{\Sigma}^{-1}(x-\mu_K) \\
&= \log\frac{\pi_k}{\pi_K} - \frac{1}{2}(\mu_k+\mu_K)^T\mathbf{\Sigma}^{-1}(\mu_k-\mu_K) + x^T\mathbf{\Sigma}^{-1}(\mu_k-\mu_K) \\
&= a_k + \sum_{j=1}^{p} b_{kj}x_j
\end{aligned} \tag{4.14}$$

dove abbiamo posto $a_k = \log\frac{\pi_k}{\pi_K} - \frac{1}{2}(\mu_k+\mu_K)^T\mathbf{\Sigma}^{-1}(\mu_k-\mu_K)$ e b_{kj} come il j-mo elemento di $\mathbf{\Sigma}^{-1}(\mu_k - \mu_K)$.
Per la Quadratic Discriminant Analysis si mostra invece che

$$\log\left(\frac{\mathbb{P}(Y = k|X = x)}{\mathbb{P}(Y = K|X = x)}\right) = a_k + \sum_{j=1}^{p} b_{kj}x_j + \sum_{j=1}^{p}\sum_{l=1}^{p} c_{kjl}x_j x_l \tag{4.15}$$

dove a_k, b_{kj}, c_{kjl} sono opportune funzioni di $\pi_k, \pi_K, \mu_k, \mu_K, \mathbf{\Sigma}_k$ e $\mathbf{\Sigma}_K$. Osserviamo quindi che la Quadratic Discriminant Analysis, come suggerito dal nome, presenta un modello sottostante quadratico.

Infine abbiamo che per quanto riguarda il Naive Bayes possiamo calcolare

$$\begin{aligned}
&\log\left(\frac{\mathbb{P}\left(Y=k|X=x\right)}{\mathbb{P}\left(Y=K|X=x\right)}\right)\\
&=\log\frac{\pi_k f_k(x)}{\pi_K f_K(x)}\\
&=\log\frac{\pi_k\prod_{j=1}^{p} f_{kj}(x_j)}{\pi_K\prod_{j=1}^{p} f_{Kj}(x_j)}\\
&=\log\frac{\pi_k}{\pi_K}+\sum_{j=1}^{p}\log\frac{f_{kj}(x_j)}{f_{Kj}(x_j)}\\
&=a_k+\sum_{j=1}^{p} g_{kj}(x_j)
\end{aligned}\tag{4.16}$$

dove abbiamo posto $a_k=\log\frac{\pi_k}{\pi_K}$ e $g_{kj}=\log\frac{f_{kj}(x_j)}{f_{Kj}(x_j)}$.
Sfruttando queste quantità possiamo andare a fornire alcune analogie e differenze tra i vari metodi presentati: elenchiamone le principali.

- In primo luogo, come facilmente intuibile, notiamo che la Linear Discriminant Analysis rappresenta un caso particolare di Quadratic Discriminant Analysis in cui poniamo tutti i coefficienti $c_{kjl}=0$ per ogni scelta di k,j,l.

- Un generico classificatore che possieda un bordo di decisione lineare è un caso speciale di Naive Bayes in cui si pongono le funzioni $g_{kj}(x_j)=d_{kj}x_j$ per opportuni coefficienti d_{kj}: questo significa che LDA rappresenta una particolare istanza di Naive Bayes, contrariamente a quanto potrebbe sembrare da uno sguardo superficiale.

- Se modellizziamo $f_{kj}(x_j)$ in (4.16) con una distribuzione normale $N(\mu_{kj},\sigma_j^2)$, ovvero teniamo la stessa varianza al variare delle classi, otteniamo $g_{kj}=b_{kj}x_j$ dove $b_{kj}=\frac{\mu_{kj}-\mu_{Kj}}{\sigma_j^2}$: questo non è altro che un caso particolare di Linear Discriminant Analysis.

- La Quadratic Discriminant Analysis e il Naive Bayes non sono l'uno il caso particolare dell'altro.

- Infine, sia la regressione logistica che la Linear Discriminant Analysis danno luogo a una stima che risulta lineare rispetto alle variabili x_j, con l'unica differenza che LDA parte dall'assunzione che i predittori $X_1,\ldots,X_p$ seguano distribuzioni normali. Per questo motivo ci aspettiamo che questo metodo performerà meglio quando questa ipotesi è vera, mentre si preferisce utilizzare la regressione logistica (o altri metodi) nel caso in cui non si è certi di ciò.

4.7 Comportamento rispetto all'arrivo di nuovi dati

Come già fatto nella Sezione 3.5, vediamo brevemente come gestire l'arrivo di nuovi dati dopo una prima fase iniziale di addestramento dei modelli generativi: per tenere conto delle nuove quantità, possiamo utilizzare una qualsiasi delle tecniche elencate nella Sezione 1.6, scegliendo quella che preferiamo in base alle esigenze specifiche del problema in considerazione e alla forma delle nuove osservazioni. Nel contesto dei modelli generativi è interessante osservare che i nuovi dati possono essere utilizzati per due scopi: il primo, utilizzabile solo quando stiamo considerando LDA e QDA, consiste nell'aggiornamento dei parametri dei modelli relativi alle medie e alle matrici di covarianza, mentre il secondo, l'unico possibile per Naive Bayes, consiste nell'aggiornamento delle stime delle probabilità a posteriori. Nel caso della Discriminant Analysis possiamo quindi giostrare queste due alternative per cercare di ottenere un risultato finale migliore.

4.8 Esempi di modelli generativi

Per concludere la nostra panoramica sui modelli generativi risulta interessante portare alcuni esempi pratici per capire come effettivamente si comportano questi metodi. In generale possiamo osservare che, nel caso in cui abbiamo a disposizione un gran numero di osservazioni e un modello sottostante lineare, i risultati di una regressione logistica e dei metodi finora visti sono abbastanza simili. Prendiamo come esempio i dati relativi a Iris Dataset che abbiamo visto nella Sezione 3.6 e addestriamo su questo dataset i modelli che abbiamo studiato: a livello di accuratezza otteniamo i risultati nella Tabella 4.1. In particolare sono tutti molto vicini ad 1, quindi per quanto visto nella Sezione 1.7 possiamo affermare che tutti e quattro i modelli presi in considerazione sono molto efficaci nel cogliere la differenza tra le due classi. Inoltre, possiamo notare che i risultati della regressione logistica, di LDA e di QDA sono molto simili: infatti il modello presenta due classi linearmente separabili, e probabilmente l'ipotesi di gaussianità sottostante alle classi è ragionevole. Infine anche il fatto che l'accuratezza dedotta utilizzando Naive Bayes sia molto alta suggerisce la presenza di indipendenza tra i predittori: osserviamo che, in generale, qualora questa condizione venga meno l'accuratezza ottenuta da questo metodo risulta avere valori intorno a 0.5.
Analizziamo cosa succede invece quando siamo in presenza di un problema che presenta una natura non lineare: come facilmente intuibile da quanto abbiamo dedotto nella Sezione 4.6, la regressione logistica e la Linear Discriminant Analysis produranno un modello che possiederà una bassa capacità predittiva. Volendo comunque utilizzare una di queste due tecniche, ci sono diversi modi per superare il limite imposto dalla linearità, uno dei quali consiste nel creare nuovi predittori andando a combinare quelli già presenti (in questo modo possiamo creare ad esempio un modello quadratico andando a considerare il quadrato di un predittore o in generale le combinazioni date dal prodotto di due predittori qualsiasi). Questo procedimento può risultare computazionalmente lento, specialmente nel caso in cui ci venga fornito un dataset con un numero elevato di dati e predittori: per questo motivo si preferisce ricorrere all'uso di modelli che comprendono

Metodo	**Accuratezza**
Regressione Logistica	0.9556
LDA	1.0
QDA	1.0
Naive Bayes	0.9333

Tabella 4.1: Confronto delle accuratezze ottenute addestrando una serie di metodi su Iris Dataset, di cui si trova una rappresentazione grafica nella Figura 3.1. I quattro metodi sono stati istruiti sul dataset diviso in training e test data tramite la funzione `train_test_split` di Scikit-Learn con parametri `train_size` = 0.7 e `random_state` = 1, e l'accuratezza è stata calcolata sui test data. Inoltre tutti e quattro i metodi sono stati eseguiti sfruttando le funzioni pre-implementate in Scikit-Learn: per maggiori dettagli sui parametri utilizzati si veda la funzione `confronta_soglie` in [2].

già per loro costruzione la possibilità di superare la soglia della linearità. Ad esempio, come già accennato la Quadratic Discriminant Analysis permette di tenere in considerazione la possibilità di aggiungere componenti che siano frutto di combinazioni quadratiche dei predittori: vediamone una applicazione pratica. Costruiamo un dataset fittizio, che chiameremo Two Circles dataset, generato a partire dalla funzione `make_circles` di Scikit-Learn (si veda [3] per ulteriori informazioni sulla funzione generatrice e la Figura 4.1 per una rappresentazione grafica). Chiaramente questo dataset presenta due classi la cui divisione non può essere colta tramite un modello lineare: questo fatto viene confermato analizzando i risultati delle accuratezze calcolate istruendo i modelli visti finora, che sono riportati nella Tabella 4.2. La regressione logistica e la LDA, istruite sui dati forniti senza ulteriori combinazioni, restituiscono pessimi risultati, ovvero accuratezze intorno a 0.5, che corrispondono a un modello privo di capacità predittorie. Al contrario la QDA, come intuibile, è molto efficace nel cogliere le varie differenze tra i due gruppi: infatti l'accuratezza calcolata con questo metodo risulta molto vicina ad 1. Infine, anche Naive Bayes risulta ottenere una buona accuratezza (anche se minore di quella ottenuta tramite QDA): infatti le due classi sono generate dalla funzione in modo che le due features che caratterizzano i dati siano indipendenti l'una dall'altra, quindi le assunzioni che ne costituiscono la base sono rispettate. Osserviamo come ultimo aspetto che potremmo aggiungere le combinazioni quadratiche dei due predittori per poter utilizzare la regressione logistica e la Linear Discriminant Analysis: effettivamente, così facendo, i risultati che si ottengono sono ottimi, e quasi migliori di quelli ottenuti precedentemente. Il problema è che aggiungere queste features ci ha portato a impiegare un tempo pari a circa 5 volte quello impiegato per addestrare la QDA: nel nostro specifico esempio non è un problema, in quanto abbiamo solamente due predittori e un numero relativamente

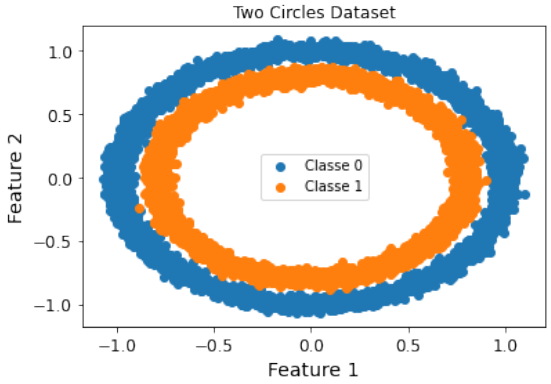


Figura 4.1: Two Circles Dataset generato a partire dalla funzione `make_circles` con parametri `n_samples` = 10000, `noise` = 0.03 e `random_state` = 1.

basso di dati. Tuttavia, andando a considerare un problema con più predittori e/o dati questo potrebbe portare a una eccessiva lentezza del procedimento in fase di istruzione.

Metodo	**Accuratezza**
Regressione Logistica	0.4873
LDA	0.4873
QDA	0.9973
Naive Bayes	0.9967

Tabella 4.2: Confronto delle accuratezze ottenute addestrando una serie di metodi sui dati del Two Circles Dataset presentato in Figura 4.1. I quattro metodi sono stati istruiti sul dataset diviso in training e test data tramite la funzione `train_test_split` di Scikit-Learn con parametri `train_size` = 0.7 e `random_state` = 1, e l'accuratezza è stata calcolata sui test data. Inoltre tutti e quattro i metodi sono stati eseguiti sfruttando le funzioni pre-implementate in Scikit-Learn: per maggiori dettagli sui parametri utilizzati si veda la funzione `confronta_soglie` in [2].

Capitolo 5

Alberi di classificazione

5.1 Introduzione

I metodi visti finora sono molto interessanti da un punto di vista matematico, ma possono risultare scomodi da interpretare, specialmente nel caso in cui abbiamo più di due predittori. Uno strumento particolarmente utile per venire incontro all'esigenza di una maggiore interpretabilità è rappresentato dagli alberi: queste sono costruzioni molto intuitive da interpretare, al punto che risultano molto efficaci anche nel caso in cui si vogliano ottenere risultati che siano fruibili da persone non esperte del settore. Vediamone la costruzione.

5.2 Costruzione di un albero di classificazione

Supponiamo che ci venga fornito un problema di classificazione che ammette p predittori. La costruzione di un albero di classificazione passa attraverso due step: il primo passo consiste nel dividere lo spazio dei predittori $X_1, \dots, X_p$ in J regioni $R_1, \dots, R_J$ contenute in $X_1 \times \cdots \times X_p$ che siano distinte, separate tra loro e tali che ricoprano l'intero spazio (ovvero $\cup_{n=1}^{J} R_n = X_1 \times \cdots \times X_p$), mentre il secondo passo consiste nel costruire le predizioni, che richiediamo essere uniformi all'interno di una data regione R_j. Questo secondo step risulta quello più facile da risolvere: data una regione R_j, al variare delle classi contiamo quanti training data cadono all'interno della regione stessa. A questo punto assegniamo a ogni osservazione contenuta in questa regione la classe che risulta essere più frequente.
Dobbiamo quindi trovare un modo per costruire le regioni in cui dividere lo spazio dei predittori. A priori le regioni potrebbero avere forma arbitraria, ma per semplicità assumiamo che siano dei rettangoli pluri-dimensionali: data una misura dell'errore E, cerchiamo delle regioni che minimizzino l'errore complessivo, ovvero

$$\min_{R_1,\dots,R_J} \sum_{j=1}^{J} \sum_{i \in R_j} E(y_i, \hat{y}_i) \tag{5.1}$$

dove abbiamo indicato con $E(y_i, \hat{y}_i)$ l'errore che risulta dall'assegnare all'osservazione i, con classe reale y_i, la classe $\hat{y}_i$. Discuteremo in seguito la scelta di questa misura.
Considerare ogni possibile divisione risulta estremamente oneroso da un punto di vista computazionale: è per questo che si considera un procedimento noto come "recursive binary splitting", che viene comunemente indicato come "avido", in quanto considera a ogni passo una divisione che ottimizza la procedura a quel dato passo e non necessariamente nel futuro. Partiamo considerando il caso in cui tutti i predittori assumono valori continui (nella pratica reali). Siano $X_1, \ldots, X_p$ i predittori del nostro problema: per ogni $j = 1, \ldots, p$ consideriamo un cutpoint s, e al variare di s consideriamo i due semispazi

$$R_1(j, s) = \{X | X_j \leq s\}, \quad R_2(j, s) = \{X | X_j > s\} \tag{5.2}$$

A questo punto la divisione avverrà in corrispondenza dei valori di j ed s che minimizzano la seguente quantità:

$$\sum_{i: x_i \in R_1(j,s)} E(y_i, \hat{y}_i) + \sum_{i: x_i \in R_2(j,s)} E(y_i, \hat{y}_i) \tag{5.3}$$

Questo approccio risulta chiaramente non utilizzabile nel caso in cui abbiamo a disposizione una variabile di tipo qualitativo: vediamo come procedere in questo caso. Semplicemente al posto di dividere lo spazio con una disuguaglianza possiamo considerare un taglio che corrisponde ad assegnare alcuni dei valori della variabile a una delle due regioni e il resto all'altra.
A questo punto, indipendentemente dal tipo di variabili con cui stiamo lavorando, abbiamo diviso il nostro spazio di partenza in due regioni. Per procedere nella costruzione dell'albero, si ripete il procedimento sulle due regioni separatamente: si considera la divisione migliore della prima regione, la migliore della seconda e tra le due si sceglie di tenere la divisione che porta alla maggiore diminuzione dell'errore complessivo. Abbiamo quindi ottenuto tre regioni. L'algoritmo procede quindi ricorsivamente: in particolare, al passo n divideremo una delle n regioni ottenute per ottenerne $n + 1$.
Vediamo infine come introdurre un criterio di stop che può essere utile sia per velocizzare la costruzione da un punto di vista computazionale che per evitare di ottenere come risultato finale regioni che contengano un solo elemento, una soluzione che è chiaramente poco auspicabile in quanto estremamente legata ai dati utilizzati in fase di addestramento: tra i vari esempi a nostra disposizione quello più utilizzato consiste nel continuare la divisione fino a quando ogni regione identificata dall'albero conterrà al massimo un numero prefissato di osservazioni, solitamente cinque. Si sceglie di utilizzare questo numero perchè nella pratica si è osservato essere ideale nel ridurre la possibilità di realizzare overfitting senza intaccare eccessivamente le capacità predittive del modello, ma chiaramente si possono utilizzare regioni più o meno grandi a seconda delle caratteristiche del nostro problema: se ad esempio abbiamo a disposizione poche osservazioni può essere conveniente utilizzare regioni più piccole. Al contrario, se il nostro dataset di partenza contenesse molti dati, il risultato finale potrebbe essere poco chiaro o dare problemi di overfitting: dobbiamo perciò trovare un modo di diminuire la dimensione dell'albero, ovvero il numero di nodi e foglie (i nodi terminali), che diventa quindi un nuovo parametro

del problema. Per fare ciò si ricorre solitamente al "cost-complexity pruning": questo metodo consiste nel costruire un albero molto grande T_0 (ad esempio con massimo 5 osservazioni per nodo), per poi "potarlo" nel seguente modo. Sia $T \subset T_0$ un sottoalbero ottenuto potando T_0, ovvero ottenuto unendo alcuni dei nodi (interni) di T_0. Indicato con $|T|$ il numero di nodi terminali di T, per ogni nodo m, a cui corrisponde la regione R_m, siano

$$N_m = \#\{x_i \in R_m\}$$

$$Q_m(T) = \frac{1}{N_j} \sum_{x_i \in R_m} E(y_i, \hat{y}_i)$$

Preso $\alpha \geq 0$ definiamo la seguente quantità:

$$C_\alpha(T) = \sum_{m=1}^{|T|} N_m Q_m(T) + \alpha |T| \tag{5.4}$$

Scelto il valore di α utilizzeremo l'albero $T_\alpha \subset T_0$ che minimizza (5.4): in particolare vale il seguente risultato.

Proposizione 5.2.1. Fissato $\alpha \geq 0$, esiste ed è unico un albero T che minimizza (5.4).

Resta quindi da trovare un criterio per determinare il parametro α: solitamente si ricorre alla five o alla tenfold cross-validation, scegliendo il valore di $\hat{\alpha}$ che minimizza l'errore trovato usando la cross validation. L'albero finale sarà quindi $T_{\hat{\alpha}}$, costruito utilizzando i dati dell'intero dataset.
A questo punto per completare la nostra analisi dobbiamo fornire una misura dell'errore per poter calcolare (5.1): vediamo tre possibili alternative, ovvero classification error rate, Gini ed entropia, definiti come segue.

Definizione 5.2.1. Dato un metodo di classificazione su K classi sia $\hat{p}_{mk}$ la proporzione di osservazioni training nella regione R_m che appartengono alla classe k. Il *classification error rate* è dato da

$$E = 1 - \max_k (\hat{p}_{mk}) \tag{5.5}$$

Definiamo l'*indice di Gini* come

$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}) \tag{5.6}$$

Infine definiamo l'*entropia* come

$$D = -\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk} \tag{5.7}$$

Analizziamoli più nel dettaglio: il "classification error rate" restituisce, in corrispondenza di ogni regione R_m, la frazione di osservazioni usate per istruire l'albero che non appartengono alla classe più frequente nella regione stessa. Per costruire un albero senza potatura questa misura risulta talvolta poco precisa: si preferisce quindi utilizzare l'indice di Gini o l'entropia. L'indice di Gini misura la varianza totale sulle K classi: in particolare, assume valori piccoli quando le quantità $\hat{p}_{mk}$ risultano vicine a 0 oppure 1. Per questo motivo si dice che misura la purezza dei nodi: G risulta piccolo quando i nodi contengono prevalentemente osservazioni da una singola classe. Infine, per quanto riguarda l'entropia, osserviamo che essendo $0 \leq \hat{p}_{mk} \leq 1$ segue che $0 \leq -\hat{p}_{mk} \log \hat{p}_{mk}$ e quindi D risulta non-negativo. Si può notare che, come G, D assume valori vicini a zero quando le quantità $\hat{p}_{mk}$ risultano vicine a 0 oppure 1. Si osserva che nella pratica Gini ed entropia assumono valori molto simili, e risultano entrambi molto efficaci nel catturare la purezza dei nodi. Per questo motivo si ricorre a questi indici nella costruzione dell'albero, mentre solitamente si utilizza il classification error rate per eseguire il cost-complexity pruning.
Una piccola nota finale: a volte capita che, nel costruire un albero, alcune divisioni portino a nodi che presentano la stessa predizione. Nonostante possa sembrare controintuitivo, questo serve a migliorare la purezza dei nodi: solitamente a una delle due regioni corrisponde una purezza che si avvicina molto a 1. Questo vuol dire che a un dato che cade all'interno di questa regione può essere assegnata con maggiore sicurezza la classe predetta, mentre magari nell'altra regione si è meno sicuri.

5.3 Vantaggi e svantaggi degli alberi e Metodi Ensemble

Gli alberi di classificazione (e di regressione, non trattati in questa tesi) presentano alcuni vantaggi: innanzitutto, risultano facilmente interpretabili grazie alla rappresentazione grafica, che risulta immediata; inoltre non richiedono l'introduzione di variabili dummy per modellizzare i predittori qualitativi.
Tuttavia presentano anche alcuni svantaggi: in primo luogo generalmente non hanno lo stesso livello di accuratezza dei modelli presentati precedentemente. Infatti dipendono fortemente dalla dimensione del dataset utilizzato in fase di addestramento: un numero troppo ridotto di osservazioni iniziali può portare a non cogliere in modo esauriente le strutture sottostanti il problema, così come un numero troppo elevato può portare a trovare relazioni che nella realtà non esistono, mentre nei metodi visti nei precedenti capitoli la dimensione del dataset di partenza non era particolarmente influente per determinare la bontà del modello. In aggiunta risultano poco robusti, in quanto piccoli cambiamenti nei dati portano solitamente a cambiamenti significativi a livello della predizione. Per migliorare questo aspetto si possono utilizzare dei metodi ensemble, che consistono in generale nel combinare metodi più semplici per ottenere modelli più performanti: nel caso specifico dei classificatori ad albero, vedremo come esempi il bagging e le foreste randomiche.

5.4 Bagging

Gli alberi di classificazione costruiti come mostrato nella Sezione 5.2 sono soggetti ad alta varianza: questo significa ad esempio che, se dividiamo in due parti il dataset che ci viene fornito e istruiamo il nostro albero prima su uno dei due gruppi e poi sull'altro, otterremo risultati che generalmente sono abbastanza diversi. Per risolvere questo problema uno dei metodi che si può applicare è il "bootstrap aggregation" o "bagging".
Ricordiamo innanzitutto che, quando abbiamo diverse stime per una stessa quantità, alla loro media campionaria è associata una varianza minore rispetto alle singole stime: date n osservazioni indipendenti $Z_1, \dots, Z_n$ di varianza comune σ^2, la varianza della loro media è infatti pari a $\frac{\sigma^2}{n}$, minore quindi della varianza a livello della singola osservazione. Sfruttiamo questo fatto: per diminuire la varianza della nostra stima possiamo calcolare B predizioni $\hat{f}_1(x) \dots, \hat{f}_B(x)$, utilizzando B diversi dataset per istruire il metodo, e andare poi ad utilizzare come stima finale una quantità che rappresenti la media delle nostre predizioni.
Il problema principale di questa strategia è che nella realtà disponiamo di un unico dataset da cui prendere i dati: per superarlo possiamo ricorrere all'uso del bootstrap, tecnica che permette di costruire B diversi dataset a partire da uno solo (si veda la Sezione 1.5 per maggiori dettagli). Nel caso specifico degli alberi, questa soluzione si traduce nel creare B alberi (senza potarli) per poi andare a fare una predizione che sia data dalla media delle predizioni che risultano dai singoli alberi.
Vediamo come possiamo costruire la media nel caso in cui abbiamo un problema di classificazione: tra i vari modi di fare ciò il più semplice è il seguente. Inizialmente creiamo B alberi utilizzando i B bootstrapped dataset per istruirli: successivamente, data un'osservazione ne registriamo la classe predetta da ognuno degli alberi e assegniamo a questo dato la classe che risulta occorrere il maggior numero di volte tra le B predizioni.
Osserviamo infine che, a livello teorico, il numero di alberi B è un parametro che andrebbe determinato volta per volta, in base ai dati a nostra disposizione: tuttavia nella pratica si sceglie quasi sempre di usare $B = 100$ bootstrapped dataset, in quanto si è osservato che tendenzialmente restituisce ottimi risultati.
Inoltre il bagging permette di fornire una stima dell'importanza dei predittori: fissato un predittore, si calcola per ogni albero la somma delle diminuzioni nell'indice di Gini che comporta ogni divisione associata al predittore in questione e se ne registra poi la media su tutti i B alberi. A questo punto si possono ordinare i risultati ottenuti: variabili con diminuzioni medie meggiori indicano predittori più significativi.

5.4.1 Stima dell'errore Out-of-Bag

Giunti a questo punto è interessante osservare che esiste un modo diretto di stimare il test error di un metodo bagging senza passare dalla divisione in training e validation set. Vale il seguente risultato.

Proposizione 5.4.1. Supponiamo di avere a disposizione un dataset con un numero elevato di osservazioni. Allora, in media, ogni albero costruito con il metodo bagging

utilizza circa due terzi dei dati a disposizione.

Dimostrazione. Supponiamo che il nostro dataset contenga N osservazioni e costruiamo un dataset B con il metodo boostrap. Per ogni $i = 1, \dots, N$ definiamo la variabile aleatoria X_i in modo che assuma valore 1 se l'i-esimo dato compare nel dataset B e 0 altrimenti. Vale che

$$\mathbb{P}(X_i = 1) = 1 - \mathbb{P}(X_i = 0) = 1 - \mathbb{P}(\{i \text{ non compaia nel bootstrap}\})$$

Indichiamo con A_{ik} l'evento che corrisponde al fatto che l'i-mo dato non compaia come k-mo elemento del bootstrapped dataset: questi sono eventi indipendenti in quanto ogni scelta è indipendente dalle altre. Inoltre è immediato che $\mathbb{P}(A_{ik}) = 1 - \frac{1}{N}$. Quindi

$$\mathbb{P}(X_i = 0) = \mathbb{P}\left(\bigcup_{k=1}^{N} A_{ik}\right) = \prod_{k=1}^{N} \mathbb{P}(A_{ik}) = \prod_{k=1}^{N} \left(1 - \frac{1}{N}\right) = \left(1 - \frac{1}{N}\right)^N$$

In conclusione abbiamo che

$$\mathbb{P}(X_i = 1) = \left(1 - \frac{1}{N}\right)^N$$

A questo punto, indicata con $S = \sum_{i=1}^{N} X_i$ la variabile aleatoria che conta il numero di osservazioni utilizzate nel bootstrapped dataset otteniamo che

$$\mathbb{E}[S] = \sum_{i=1}^{N} \mathbb{E}[X_i] = \sum_{i=1}^{N} \left(1 - \frac{1}{N}\right)^N = N\left(1 - \frac{1}{N}\right)^N$$

Ora, supponendo di avere N abbastanza grande, la quantità $\left(1 - \frac{1}{N}\right)^N$ è approssimabile come $\frac{1}{e} \sim \frac{2}{3}$. Quindi in conclusione abbiamo che

$$\mathbb{E}[S] \sim \frac{2}{3}N \qquad \square$$

Quindi ogni volta che costruiamo un albero circa un terzo dei nostri dati rimane escluso dall'addestramento: questi vengono detti osservazioni out-of-bag (OOB). Di conseguenza per ciascuno dei B alberi costruiti usando un bootstrapped dataset possiamo predire la classe di appartenenza delle osservazioni che ne risultano OOB. Fissata una osservazione (presa dal dataset di partenza), questa risulterà in media essere un'osservazione OOB per circa $B/3$ degli alberi costruiti: per assegnarle una classe finale, ne prediciamo la classe di appartenenza in corrispondenza di ognuno degli alberi per cui risulta essere un dato OOB e prendiamo come classe finale quella che risulta più frequente. Calcolando infine l'errore ottenuto utilizzando queste come predizioni, per ognuno dei dati nel dataset originale, otteniamo una quantità che viene definita errore OOB: questa è una stima del test error complessivo che risulta computazionalmente meno onerosa e più efficiente della classica cross-validation.

5.5 Foreste Randomiche

Il metodo bagging presentato precedentemente è molto utile per diminuire la varianza degli alberi di classificazione, tuttavia presenta un problema: gli alberi costruiti a partire da un bootstrapped dataset risultano altamente correlati tra loro. Supponiamo ad esempio che ci sia un predittore che risulta estremamente più significativo degli altri: di conseguenza, la maggior parte degli alberi costruiti con il metodo bagging (se non tutti) presenteranno una divisione iniziale che considera questo predittore. In questo modo gli alberi risulteranno altamente correlati tra loro, e il risultato finale della nostra predizione potrebbe esserne falsato.
Per superare questo problema sono state introdotte le foreste randomiche: così come il bagging, consistono nel costruire una serie di alberi istruiti su un boostrapped dataset. L'unica differenza è la seguente: per ogni albero, a ogni passo si considera una divisione che possa riguardare esclusivamente m dei p predittori, che vengono scelti ogni volta in modo casuale. Questo m è un nuovo parametro: solitamente si sceglie di utilizzare come valore $m \approx \sqrt{p}$, anche se si possono considerare valori diversi per migliorare le performance del nostro metodo. In particolare, la scelta della radice quadrata risulta essere un buon compromesso tra due problematiche: infatti, da un lato diminuire il numero di predittori considerati porta ad avere una costruzione molto più veloce da un punto di vista computazionale. Tuttavia, utilizzare un numero troppo ridotto di predittori porta a un elevato bias nell'output restituito, in quanto tenere in considerazione meno possibilità per la divisione comporta chiaramente una diminuzione della capacità di cogliere la vera relazione sottostante i dati: anche semplicemente a livello intuitivo si capisce che usare $m = 1$ oppure 2 non risulta una scelta condivisibile. Osserviamo infine che prendere $m = p$ corrisponde a eseguire bagging.
A questo punto tornando all'esempio in cui abbiamo un predittore molto significativo in media, per ogni albero, $(p - m)/p$ delle divisioni non lo potranno tenere in considerazione: come risultato gli alberi così costruiti risulteranno meno correlati rispetto al bagging puro.

5.6 Addestramento su nuovi dati

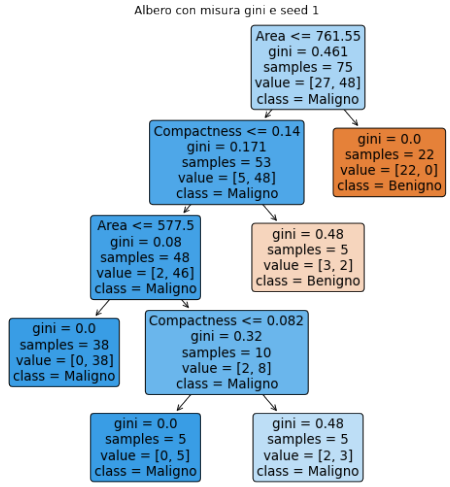
Vediamo a questo punto come si comportano gli alberi di classificazione rispetto all'introduzione di nuovi dati in una fase successiva al primo addestramento. A livello teorico si potrebbero utilizzare retraining periodico, addestramento incrementale e apprendimento online (si veda la Sezione 1.6): tuttavia queste tecniche risultano particolarmente costose da un punto di vista computazionale se usate in maniera superficiale. In questo contesto si mostra la potenza dei metodi ensemble: infatti, quando usiamo le foreste randomiche o il metodo bagging, un modo di incorporare le informazioni portate dai nuovi dati all'interno del nostro modello consiste nell'utilizzarli per addestrare una serie di nuovi alberi, che può essere più o meno grande in base alla quantità di nuove osservazioni a disposizione, che insieme ai B originari contribuiscano a fornire una adeguata predizione su una data osservazione.

Possiamo quindi confrontare come si comporta l'aggiornamento di un albero di classificazione rispetto all'aggiornamento di un modello di regressione logistica: in generale, come già visto, nella regressione logistica si riaddestrano tutti i parametri del modello su una combinazione tra nuovi e vecchi dati. Questo può risultare particolarmente costoso da un punto di vista computazionale, specialmente nel caso in cui abbiamo un numero elevato di variabili. Al contrario quando si aggiorna un albero di classificazione si cerca di localizzare in quale punto troviamo cambiamenti nei nuovi dati, per andare a modificare solo i nodi (e in generale le parti) dell'albero che ne vengono effettivamente interessati: in questo modo l'operazione diventa molto più veloce da un punto di vista computazionale. Inoltre la regressione logistica tende a esssere molto sensibile al cambiamento di dati: può capitare che l'aggiunta di nuove osservazioni porti a risultati molto diversi rispetto a quelli originali, mentre gli alberi riescono ad assorbire meglio ciò grazie alla loro flessibilità. Nonostante ciò risulta più semplice interpretare quanto i nuovi dati modificano la regressione logistica rispetto agli alberi: infatti risulta molto leggibile il cambiamento a livello di importanza delle variabili, in quanto è facilmente deducibile dal valore dei coefficienti associati, mentre negli alberi può risultare difficile coglierne la variazione, in quanto questa potrebbe riguardare parti dell'albero nascoste o interne. Questo è forse uno dei pochi aspetti sui quali lo strumento fornito dagli alberi di classificazione risulta poco interpretabile.

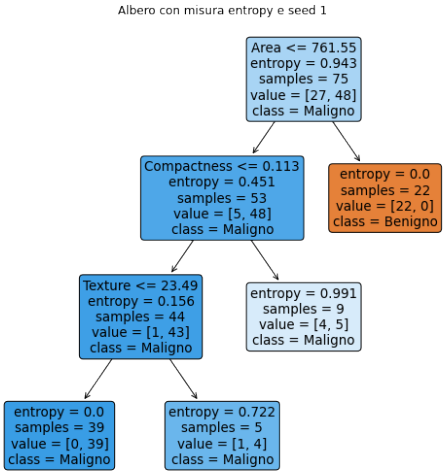
5.7 Applicazione al Breast Cancer Dataset

Per concludere la nostra trattazione sugli alberi di classificazione vediamo un esempio pratico che ci faccia capire meglio quanto abbiamo affermato nel corso di questo capitolo: studiamo ad esempio il Breast Cancer Dataset, che abbiamo già introdotto nella sezione 2.3. Per una implementazione dettagliata si veda [2].
Vediamo innanzitutto come si traduce il fatto che gli alberi di classificazione sono soggetti ad alta varianza: prendiamo inizialmente in considerazione i dati relativi a Texture, Perimeter, Area, Smoothness e Compactness del tumore e costruiamo degli alberi di classificazione che ci permettano di distinguere tra cancro benigno e maligno. Ne costruiremo sei, utilizzando ogni volta come training data 75 osservazioni, estratte casualmente tramite la funzione `sample` della libreria `random` di Python eseguita a partire da tre semi diversi, e utilizzando come misura l'indice di Gini e l'entropia. Otteniamo i risultati nelle Figure 5.1, 5.2 e 5.3: analizzando da un punto di vista grafico i nostri risultati osserviamo due aspetti. In primo luogo, i risultati ottenuti considerando lo stesso insieme di addestramento ma misure dell'errore diverse sono molto simili: questo ci conferma il fatto che l'indice di Gini e l'entropia misurano sostanzialmente le stesse quantità, ma sotto due punti di vista diversi. Al contrario, alberi costruiti con dati diversi sono molto diversi tra loro: questo è in linea con quanto ci aspettavamo dallo studio teorico effettuato precedentemente, in particolare ci conferma il fatto che gli alberi di classificazione, costruiti in modo "base", sono soggetti ad alta varianza.

Per superare questi difetti abbiamo introdotto i metodi ensemble: vediamo come questi si comportano sul Breast Cancer Dataset. Consideriamo ora l'insieme di tutti i



(a) Costruzione dell'albero con indice di Gini.



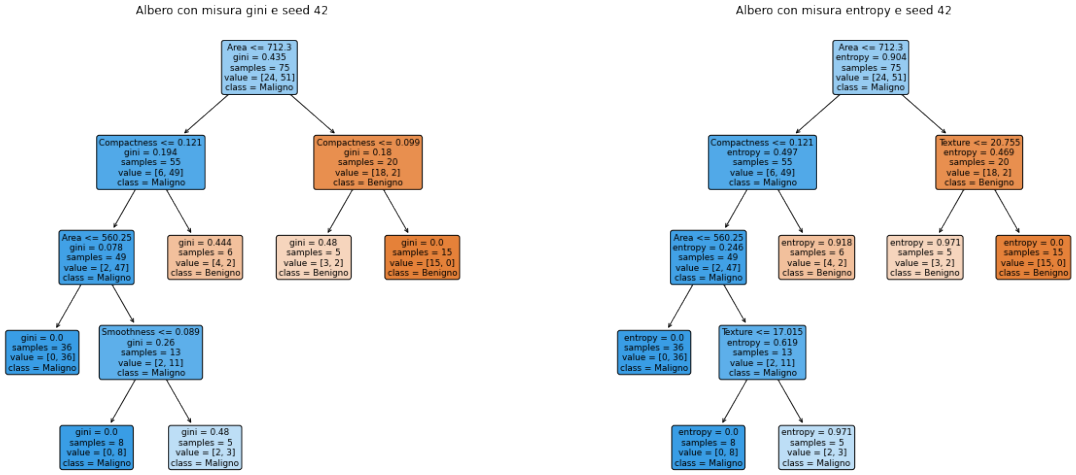
(b) Costruzione dell'albero con entropia.

Figura 5.1: Confronto di due alberi di classificazione costruiti su 75 osservazioni del Breast Cancer Dataset prese in maniera casuale dalle osservazioni totali usando come valore del seme `seed = 1`.

(a) Costruzione dell'albero con indice di Gini.

(b) Costruzione dell'albero con entropia.

Figura 5.2: Confronto di due alberi di classificazione costruiti su 75 osservazioni del Breast Cancer Dataset prese in maniera casuale dalle osservazioni totali usando come valore del seme `seed = 10`.



(a) Costruzione dell'albero con indice di Gini.

(b) Costruzione dell'albero con entropia.

Figura 5.3: Confronto di due alberi di classificazione costruiti su 75 osservazioni del Breast Cancer Dataset prese in maniera casuale dalle osservazioni totali usando come valore del seme `seed = 42`.

predittori forniti nel dataset e costruiamo tre diverse divisioni dei dati in training e validation set effettuate tramite la funzione `train_test_split` della libreria Scikit Learn con tre valori diversi del seme iniziale: in corrispondenza di ognuna di queste divisioni, istruiamo un albero di classificazione e una foresta randomica costruita con un numero di alberi pari a 100. Un primo parametro che possiamo utilizzare per confrontare i due approcci consiste nella stima dell'accuratezza totale eseguita calcolandola sulle osservazioni non utilizzate in fase di addestramento: eseguendo predizioni nei tre insiemi otteniamo i valori nella Tabella 5.1. Nonostante i valori delle foreste randomiche siano leggermente migliori, questa differenza non è particolarmente marcata: il tutto sembrerebbe inutile, considerando il fatto che le foreste impiegano circa tra 20 e 30 volte il tempo di implementazione degli alberi. Prendiamo in considerazione quindi un altro aspetto: utilizziamo i modelli creati per fornire una predizione su tutti i dati del dataset e andiamo a confrontare quanti dati vengono predetti nello stesso modo dai tre alberi e dalle tre foreste randomiche. Otteniamo che i tre alberi forniscono la stessa predizione su 523 dati, che corrispondono al 92% circa sul totale, mentre le foreste randomiche predicono nella stessa maniera 560 dati, che corrispondono al 98% circa del totale: questi dati possono essere riletti indirettamente come un indicatore della varianza dei nostri modelli. Abbiamo quindi una conferma della riduzione della varianza data dalle foreste randomiche.
Infine, come ultimo aspetto, possiamo andare a paragonare i risultati ottenuti da un albero di classificazione, una foresta randomica e K-nearest neighbors: implementandoli sempre sul dataset che stiamo studiando possiamo ottenere i risultati nella Tabella 5.2. In termini di accuratezza si osserva che i risultati sono molto simili, mentre a livello di tempi

Valore del seme	**Accuratezza dell'albero di classificazione**	**Accuratezza della foresta randomica**
5	0.947	0.947
10	0.930	0.982
42	0.971	0.965

Tabella 5.1: Valori dell'accuratezza degli alberi di classificazione e delle foreste randomiche costruite dividendo il dataset tramite la funzione `train_test_split` con valore del parametro `random_state` pari a quello nella prima colonna e `train_size` pari a 0.7. In particolare tutti e tre i modelli di alberi di classificazione sono stati costruiti utilizzando la funzione `DecisionTreeClassifier` con parametro `min_sample_size` pari a 5, mentre le foreste randomiche sono state costruite utilizzando la funzione `RandomForestClassifier` con parametri `n_estimators` pari a 100 e `oob_score` pari a `True`.

di esecuzione osserviamo che le foreste risultano computazionalmente un po' più lente da addestrare, mentre alberi di classificazione e KNN presentano tempi di addestramento molto simili.

Metodo	**Accuratezza**	**Tempo di esecuzione (secondi)**
Albero di classificazione	0.971	0.027
Foresta Randomica	0.965	0.791
5-NN	0.959	0.062

Tabella 5.2: Valori dell'accuratezza dell'albero di classificazione, della foresta randomica e di (5)NN costruiti dividendo il dataset tramite la funzione `train_test_split` con valore del parametro `random_state` pari a 42 e `train_size` pari a 0.7. In particolare l'albero di classificazione è stato costruito utilizzando la funzione `DecisionTreeClassifier` con parametro `min_sample_size` pari a 5, mentre la foresta randomica è stata costruita utilizzando la funzione `RandomForestClassifier` con parametri `n_estimators` pari a 100 e `oob_score` pari a `True`. Possiamo osservare che, a livello di accuratezze, i risultati ottenuti siano molto simili, mentre a livello di tempi di esecuzione la foresta randomica viene implementata in un tempo che è pari a circa 30 volte quello impiegato per addestrare un semplice albero.

Bibliografia

[1] *Breast Cancer Dataset*. URL: https://scikit-learn.org/stable/datasets/toy_dataset.html#breast-cancer-dataset.

[2] Daniele Capelli. *Codice per la tesi*. URL: https://github.com/dcapelli02/Codice-Tesi-Triennale.

[3] *Circles*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_circles.html.

[4] Sjoerd Dirksen. «Introduction to Machine Learning (WISB365) - Lecture Notes». Note del corso 'Introduction to Machine Learning' (codice identificativo WISB365) tenuto presso l'Università di Utrecht nell'A.A. 2023/24.

[5] Trevor Hastie, Robert Tibshirani e Jerome Friedman. *The elements of statistical learning. Data mining, inference, and prediction*. English. Springer Ser. Stat. New York, NY: Springer, 2001. ISBN: 0-387-95284-5.

[6] *Iris Dataset*. URL: https://scikit-learn.org/stable/auto_examples/datasets/plot_iris_dataset.html#sphx-glr-auto-examples-datasets-plot-iris-dataset-py.

[7] Gareth James et al. *An introduction to statistical learning. With applications in Python*. English. Springer Texts Stat. Cham: Springer, 2023. ISBN: 978-3-031-38746-3; 978-3-031-39189-7; 978-3-031-38747-0. DOI: 10.1007/978-3-031-38747-0.

[8] *Scikit Learn*. URL: https://scikit-learn.org/stable/.

[9] *Two Moon*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_moons.html.

Elenco delle figure

Elenco delle tabelle