ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

SCUOLA DI INGEGNERIA E ARCHITETTURA

Corso di Laurea in Elettronica, Informatica e Telecomunicazioni

APPRENDIMENTO DI ALBERI DI DECISIONE

Elaborata nel corso di: Fondamenti di Informatica B

Tesi di Laurea di:
FABIO MAROCCHI

Relatore:
Prof. ANDREA ROLI

ANNO ACCADEMICO 2013/2014
SESSIONE III

PAROLE CHIAVE

Alberi di Decisione

Apprendimento

Ensemble Learning

Machine Learning

Ai miei genitori, che mi hanno sempre supportato.
A mia sorella, che è stata per me un esempio da seguire.
Alla mia ragazza, che mi ha spronato a dare il meglio di me.
Ai miei amici, che mi hanno accompagnato in questo percorso.

Indice

Introduzione

Il settore dell'intelligenza artificiale è molto ampio e si può considerare costituito da diverse discipline, che vanno da problemi generici come l'apprendimento o la percezione a problemi specifici come il riconoscimento del linguaggio o di immagini. Poiché è difficile dare una definizione univoca di intelligenza, si può illustrare l'intelligenza artificiale seguendo diversi approcci, in base al punto di vista da cui la si guarda e all'obiettivo che si vuole ottenere.

L'esempio probabilmente più famoso è rappresentato dal test di Turing: si ritiene un computer intelligente se, interrogato per iscritto da un essere umano fornisce risposte indistinguibili da quelle che darebbe un altro essere umano. Secondo alcuni questo non è sufficiente a dimostrare l'intelligenza della macchina, in quanto essa si limita a simulare il comportamento umano, senza soffermarsi sui processi cognitivi che supportano questo comportamento. Se però l'obiettivo è quello di ottenere una macchina che si comporta e agisce come un essere umano, allora quello di Turing può essere considerato un test valido. Altri approcci si basano sul modo di "pensare" della macchina: basandosi sugli studi della disciplina denominata scienza cognitiva si può mirare a comprendere i processi cognitivi degli esseri umani e a modellare quindi le macchine in modo che "ragionino" come gli umani. Un altro possibile approccio è basato sulla logica e sul ragionamento "razionale", ovvero basandosi su regole di inferenza ben definite e sui dati disponibili, la capacità di ottenere un output corretto.

A prescindere da queste considerazioni, nel nostro contesto non parliamo di intelligenza artificiale in senso lato come nel caso dell'imitazione del pensiero umano, ma soltanto della capacità di svolgere un determinato task e, più in particolare, della capacità imparare, ovvero di migliorare le prestazioni del nostro sistema in quel determinato task.

Il machine learning è l'area dell'intelligenza artificiale che si occupa dello sviluppo di sistemi in grado di acquisire nuove conoscenze ed estrapolare

pattern da dati forniti in ingresso. In questa tesi effettuo una panoramica sull'apprendimento di alberi di decisione - un particolare metodo di apprendimento supervisionato - e sui vari metodi per poterne migliorare le prestazioni. Mostrerò poi i risultati di alcuni test effettuati allo scopo di quantificare empiricamente i miglioramenti apportati dalle tecniche descritte.

Nello specifico, nel primo capitolo viene introdotto l'apprendimento automatico e ne vengono descritte le categorie. Nel secondo capitolo vengono definite le caratteristiche degli alberi di decisione, i metodi per apprenderli e per affrontare situazioni particolari. Nel terzo capitolo vengono illustrate le tecniche utilizzate per l'incremento delle prestazioni. Nel quarto capitolo vengono descritti alcuni algoritmi e ne vengono confrontate le prestazioni. Infine, nel quinto capitolo, viene esposta un'analisi dei risultati ottenuti.

Capitolo 1

Machine Learning

Il machine learning è un settore fondamentale dell'intelligenza artificiale che si occupa dell'apprendimento automatico. Per poter definire una macchina intelligente, è necessario che questa sia in grado di imparare dalle proprie esperienze. Deve quindi poter effettuare delle scelte, valutare le conseguenze di queste scelte ed eventualmente modificare il proprio comportamento per fare in modo che le conseguenze siano quelle desiderate. Si richiede perciò che il software sia dotato almeno di un componente in grado di prendere decisioni, un componente in grado di interagire con l'ambiente per ottenere informazioni sulle conseguenze delle proprie azioni e un componente che permetta di modificare il componente che si occupa della strategia decisionale. Lo scopo finale di questo procedimento ciclico è di ottenere un programma in grado di migliorare le proprie prestazioni con l'esperienza.

In altre parole, secondo Mitchell (1997)[12]

> *"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."*

Traducendo:

> *"Si dice che un programma impara da un esperienza E rispetto ad un task T e ad una misura di prestazione P, se le sue prestazioni in T, misurate secondo P, migliorano con l'esperienza E."*

Per poter progettare un sistema di apprendimento è dunque necessario definire l'esperienza E, il task T e la misura di prestazione P.

L'idea alla base dell'apprendimento è quindi quella secondo cui le percezioni non devono essere utilizzate solo per scegliere l'azione da intraprendere, ma anche per permetter al software di migliorare le sue abilità decisionali per il futuro. Per poter ottenere questo risultato è necessario fornire al programma un qualche tipo di feedback che rappresenti il risultato delle sue azioni. Il tipo di apprendimento si differenzia solitamente in base al modo in cui questo feedback viene fornito. In particolare si possono distinguere tre categorie principali di apprendimento:

- supervisionato: con feedback diretto
- non supervisionato: nessun feedback esterno
- per rinforzo: con feedback indiretto

1.1 Apprendimento supervisionato

L'apprendimento supervisionato è un tipo di apprendimento in cui lo scopo consiste nell'estrapolare una funzione da un insieme di dati di addestramento detto training set. In questo insieme di addestramento ogni esempio consiste in una coppia, composta da un input e un output. Solitamente l'input viene espresso attraverso un vettore di attributi di diverso tipo mentre l'output è in genere un singolo valore. L'algoritmo di apprendimento supervisionato esamina gli esempi di addestramento con lo scopo di produrre una funzione in grado di attribuire il giusto output a nuove istanze non presenti nell'insieme di addestramento.

Ipotizzando che l'output corretto nei dati di addestramento sia prodotto attraverso quella che definiamo funzione obiettivo, lo scopo dell'apprendimento è quello di produrre una funzione che approssimi più verosimilmente possibile la funzione obiettivo. Questa funzione approssimata viene detta ipotesi e l'insieme delle ipotesi generabili dall'algoritmo di apprendimento viene detto spazio delle ipotesi. L'espressività dello spazio delle ipotesi influisce sulla qualità dell'ipotesi scelta e sul costo del calcolo dell'ipotesi. Se lo spazio è molto ampio sarà possibile generare molte ipotesi diverse, possibilmente complesse permettendo un'approssimazione più precisa anche di una funzione obiettivo complessa ma ad un costo elevato. Se, al contrario, lo spazio delle ipotesi è ridotto, una funzione obiettivo complessa sarà approssimata più grossolanamente ma il costo computazionale sarà ridotto.

Per poter risolvere correttamente un problema di apprendimento supervisionato è necessario stabilire quindi il formato dei dati di input e di output, ottenere i dati da incorporare nel training set, scegliere un modello di apprendimento e un algoritmo specifico, eseguire l'algoritmo sul training set e valutare la sua capacità di generalizzazione su un altro insieme di istanze detto test set.
Se la prestazione sul test set è soddisfacente si può utilizzare la funzione prodotta per elaborare l'output per nuove istanze di cui non conosciamo l'output corretto.

Ci sono alcuni aspetti da considerare nella risoluzione di un problema di apprendimento supervisionato che, come per l'ampiezza dello spazio di ipotesi, richiedono un compromesso tra diverse caratteristiche.

Ad esempio si rende necessario trovare un compromesso tra stabilità dell'algoritmo e adattabilità. Per stabilità si intende la propensione ad ottenere la stessa funzione o funzioni simili a partire da training set diversi mentre con adattabilità si intende la propensione a produrre funzioni diverse a partire da diversi training set. Una stabilità eccessivamente elevata porta a sbagliare sistematicamente la previsione degli output per determinate istanze. Un'adattabilità eccessiva porta invece ad ottenere risultati troppo dipendenti dal training set utilizzato. Con il giusto compromesso si possono ottenere risultati più vicini alla funzione obiettivo.

Un altro aspetto da considerare è la dimensione del training set in relazione alla complessità della funzione obiettivo. Se la funzione obiettivo è semplice, si otterranno risultati migliori utilizzando un training set relativamente ridotto mentre, se al contrario la funzione obiettivo è complessa, sarà necessario avere a disposizione un training set ampio, in modo che contenga molte delle possibili combinazioni di input.

Infine anche il numero degli attributi influisce sulla qualità dell'apprendimento. Se la funzione è complessa potrebbe richiedere un numero elevato di attributi ma se utilizziamo troppi attributi per una funzione semplice l'algoritmo faticherà a distinguere gli attributi importanti da quelli superflui e dovremo affrontare un problema che come vedremo più avanti viene definito di sovradattamento.

Volendo stabilire i componenti fondamentali di un problema di apprendimento supervisionato possiamo definire E come l'insieme di input e output corrispondenti appartenenti al training set, T come la produzione della funzione che approssima la funzione obiettivo e P come la capacità di associare il giusto output a nuove istanze di input.

1.2 Apprendimento non supervisionato

L'apprendimento non supervisionato consiste in quei problemi in cui si richiede di riconoscere pattern negli input senza avere a disposizione gli output corrispondenti. Naturalmente un apprendimento puramente non supervisionato non permette ad un software di imparare a prendere decisioni in quanto non ha a disposizione informazioni su quali siano gli output desiderabili.

Il procedimento avviene solitamente tramite metodologie statistiche con lo scopo di trovare pattern ricorrenti in un insieme di dati. Non viene fornita alcuna spiegazione sul significato di questi eventuali pattern e non si ricerca una corrispondenza con un output (da questo, il nome della categoria).

Una tecnica utilizzata nell'apprendimento non supervisionato è il clustering. Il clustering consiste nel raggruppare i dati considerati simili tra loro in sottoinsiemi detti cluster con lo scopo di determinare delle categorie di oggetti. Il processo può avvenire in modalità top-down partendo da un unico cluster e suddividendolo ricorsivamente in gruppi omogenei, oppure in modalità bottom-up, secondo cui ogni oggetto è un cluster e i cluster simili vengono uniti ricorsivamente. In entrambi i casi la condizione di arresto consiste solitamente nel raggiungimento di un numero prefissato di cluster.

Un'altra tecnica utilizzata è l'apprendimento di regole di associazione. Le regole di associazione consistono in relazioni di dipendenza tra i dati. Se si riescono a trovare queste relazioni è possibile estrapolare un pattern dai dati.

L'apprendimento non supervisionato viene utilizzato prevalentemente in ambito di data mining e gode di prestazioni migliori quando il dominio di lavoro è numerico poiché permette di utilizzare al meglio quelle tecniche derivate dalla statistica che possono essere utili in questo campo.

In un problema di apprendimento non supervisionato possiamo definire E come l'insieme dei dati in input, T come la ricerca di schemi ricorrenti nei dati, mentre P, che è difficile da definire dal momento che non abbiamo in diretto riscontro con un obiettivo, può essere rappresentato dal tipo di informazione che si riesce ad estrapolare.

1.3 Apprendimento per rinforzo

L'apprendimento per rinforzo si applica in quei casi in cui non si hanno indicazioni dirette su quale sia l'output desiderato, ma questo viene interpretato in base al cosiddetto rinforzo, ovverosia una ricompensa che determina quantitativamente se la scelta eseguita sia o meno positiva. La differenza principale tra apprendimento supervisionato e apprendimento per rinforzo consiste nel tipo di feedback che viene restituito: nel primo viene fornito l'output corretto, nel secondo viene soltanto valutato se e quanto la scelta eseguita sia desiderabile e porti ad avvicinarsi all'obiettivo finale.

Si può suddividere ulteriormente il tipo di apprendimento in passivo e attivo. Nell'apprendimento passivo la strategia è fissa e lo scopo è apprendere l'utilità dei vari stati e quindi la politica migliore. Nell'apprendimento attivo invece è necessario anche modificare la propria strategia per adattarla alle informazioni che vengono apprese durante l'esplorazione, ovvero la fase in cui vengono sperimentate di nuove scelte per valutarne l'efficacia.

Nell'apprendimento attivo spesso è necessario trovare un compromesso tra esplorazione e sfruttamento. Con sfruttamento si intende la strategia in cui si intraprendono le scelte che sappiamo essere le più redditizie in termini di rinforzo. Con esplorazione si intende prendere decisioni nuove, in modo da fare esperienza e capire tramite il rinforzo qual'è la politica migliore per raggiungere un obiettivo. Una strategia molto semplice è quella in cui si tende inizialmente a favorire l'esplorazione per avere a disposizione più informazioni e assicurarsi di non escludere subito scelte che potrebbero rivelarsi redditizie, poi si segue la politica ottima e si ritorna ad esplorare quando non è evidente quale sia la scelta migliore.

L'apprendimento per rinforzo viene spesso utilizzato quando l'apprendimento supervisionato è impraticabile ad esempio a causa dell'assenza di esempi di input e output. Inoltre può essere utile quando non è possibile conoscere il risultato della singola scelta ma si ricerca una strategia a lungo termine per raggiungere un obiettivo.

Nell'apprendimento per rinforzo possiamo definire E come l'insieme di scelte e relative ricompense, T come la ricerca del percorso che massimizza la ricompensa totale e P come la ricompensa stessa.

Capitolo 2

Apprendimento di alberi di decisione

In questo capitolo vengono introdotti gli alberi di decisione come meccanismo per effettuare scelte e classificazioni e viene descritto il funzionamento del corrispondente algoritmo di apprendimento. Vengono inoltre introdotti alcuni problemi che affliggono l'apprendimento di alberi di decisioni e suggerite le soluzioni più comuni. Le informazioni contenute in questo capitolo sono costituite prevalentemente da una rielaborazione di Russell-Norvig (2003)[17], Quinlan (1986)[14], e Rokach-Maimon (2015)[16]

2.1 Alberi di decisione

Gli alberi di decisione sono uno strumento che permette effettuare scelte che dipendono da diverse variabili e di rappresentarle in modo grafico ed intuitivo anche per un lettore umano.
Un albero di decisione consiste in un grafo ad albero in cui il cosiddetto nodo radice è l'unico nodo senza rami entranti, le foglie sono tutti i nodi senza rami uscenti e i nodi che hanno un solo ramo entrante e più rami uscenti sono detti nodi interni o nodi di test.
Un albero di decisione accetta in input un oggetto o istanza, caratterizzata da diversi attributi e fornisce in output una scelta o una classe. Alla radice e ad ogni nodo interno corrisponde un test su un attributo, e da ogni nodo fuoriescono tanti rami quanti sono i valori ammissibili dell'attributo testato. Ogni foglia infine, rappresenta la scelta finale per tutte le istanze che portano a quel nodo.

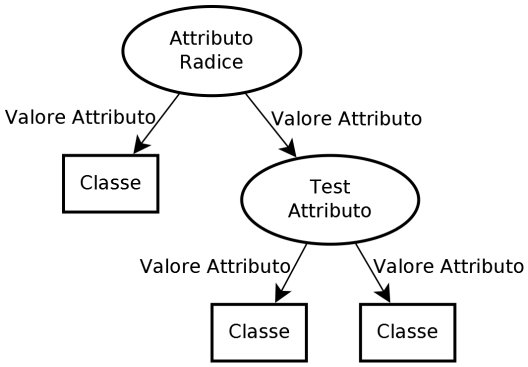

Figura 2.1: *Rappresentazione grafica di un albero di decisione*

Si possono definire due categorie principali di alberi di decisione: alberi di classificazione e alberi di regressione. Gli alberi di classificazione sono utilizzati quando l'output può assumere solo valori discreti (o classi) predeterminati, mentre quelli di regressione si usano quando l'output assume valori continui. Inizieremo descrivendo gli alberi di classificazione, mostrando poi anche i metodi che permettono di produrre alberi di regressione.

Il metodo più semplice per creare un albero decisionale per un determinato task consiste nel richiedere ad un esperto di quel settore quali sono i vari passi che devono essere seguiti per prendere una decisione. Questo approccio però è complicato da perseguire perché spesso è difficile reperire esperti disponibili e in alcuni casi è difficile persino per gli esperti definire formalmente quali siano i processi decisionali da seguire (ad esempio quando si tratta di settori ancora acerbi in cui la conoscenza umana è limitata). È in questo frangente che si rivela molto utile la possibilità di creare gli alberi automaticamente a partire dai dati attraverso un processo detto di induzione o apprendimento.

2.1.1 Utilizzo di alberi di decisione

Avendo a disposizione un albero di decisione (generato da un algoritmo di apprendimento o fornito da un esperto) è possibile classificare un oggetto seguendo un procedimento molto semplice, partendo dal nodo radice:

1. Si controlla il valore dell'attributo rappresentato dal nodo che si sta visitando e si segue il ramo corrispondente

2. Se il nuovo nodo è una foglia si prosegue al punto 3, altrimenti si torna al punto 1

3. Si assegna all'istanza la classe rappresentata dalla foglia e si termina l'esecuzione

In un albero decisionale ogni nodo suddivide l'insieme dei dati in un numero di sottoinsiemi equivalente al numero di rami uscenti dal nodo. Alla fine dell'albero si dovrebbe ottenere per ogni foglia un sottoinsieme contenente solo istanze caratterizzate dalla stessa classe. Tuttavia, è possibile decidere di interrompere in anticipo la produzione dell'albero (per i motivi che verranno spiegati in seguito) in modo tale da non riuscire ad associare ad ogni foglia una e una sola classe. In questo caso è possibile associare ad ogni foglia un vettore contenente le probabilità relative ad ogni classe[1].
Durante la classificazione, invece di associare all'istanza la classe rappresentata dalla foglia si può scegliere quella con la probabilità più alta, oppure assegnarla casualmente tenendo conto della probabilità di ogni classe in quella foglia.

È importante notare che in un determinato percorso non è sempre necessario che vengano controllati tutti gli attributi. Alcuni attributi potrebbero non essere necessari per determinare la classe dell'istanza corrente, altri, come vedremo più avanti, potrebbero essere superflui per l'intero problema.

2.2 Apprendimento

L'apprendimento automatico di alberi di decisione rientra nella categoria dell'apprendimento supervisionato, poiché consiste nel riuscire ad estrapolare un albero decisionale da un set di esempi costituiti da input e output. Questo set di esempi viene definito training set, in quanto viene utilizzato dall'algoritmo di apprendimento per addestrarsi a classificare tutte le possibili istanze. In questo senso, l'algoritmo di apprendimento di alberi decisionali prende come input un training set e fornisce in output un albero decisionale. Come accennato alla sezione 1.1, affinché questo tipo di apprendimento sia efficace è necessario che il training set abbia una dimensione adatta alla funzione obiettivo che si vuole approssimare. In caso contrario non sarebbe possibile classificare correttamente istanze non appartenenti al training set.

Una volta generato, l'albero di decisione viene solitamente testato su altri esempi (diversi da quelli presenti nel training set ma anch'essi etichettati con la classe corretta) in modo tale da verificare la sua capacità di classificare

[1] possono essere calcolate come il rapporto tra il numero di occorrenze di ogni classe e il numero totale di istanze nel sottoinsieme

oggetti non ancora visti. Questo nuovo insieme di istanze viene definito test set e la qualità dell'albero di decisione generato e quindi dell'algoritmo di apprendimento viene misurata in base alla percentuale di istanze classificate correttamente in questo test set.

Va inoltre sottolineato che,è possibile produrre alberi di decisione diversi a partire dallo stesso training set, tutti consistenti con i dati da cui sono stati prodotti. In linea generale si segue il principio del rasoio di Occam, secondo cui se più ipotesi sono consistenti con i dati, quella che ha più probabilità di essere corretta è quella che richiede meno ipotesi di partenza. In questo contesto significa che quando due o più alberi sono equivalenti[2] è da preferire quello di minor dimensione, poiché probabilmente avrà una maggiore capacità di generalizzazione, ovvero di classificare correttamente nuove istanze.
La dimensione degli alberi di decisione può essere valutata in termini di numero di nodi, numero di foglie, profondità (o numero di livelli) o numero di attributi usati. Oltre ad avere una capacità di generalizzazione maggiore, un albero di piccole dimensioni è anche meno oneroso da produrre e più leggibile da un utente umano.
Prendendo come esempio il caso limite in cui abbiamo un albero con un percorso diverso per ogni istanza del training set, questo classifica correttamente tutto il training set ma, poiché che non riesce ad estrarre alcuno schema dai dati e quindi ad estrapolare le relazioni chiave necessarie alla classificazione di nuove istanze, non sarà in grado di classificare esempi diversi da quelli del training set.

L'algoritmo di apprendimento avrà quindi anche lo scopo di trovare l'albero più piccolo, consistente con il training set. Un possibile approccio consiste nel generare tutti gli alberi decisionali consistenti con i dati e poi scegliere il più piccolo, ma questo metodo fa rientrare il problema nella classe di complessità NP-C (la categoria più complessa dei problemi non determinabili in tempo polinomiale). Perciò ci limiteremo a trovare un albero sufficientemente piccolo basandoci su algoritmi euristici. In questo modo potremmo non trovare il migliore degli alberi ma ne troveremo uno "soddisfacente" in tempi ragionevoli.

L'idea alla base dell'algoritmo è di trovare per ogni nodo l'attributo con il maggior impatto sulla classificazione e sceglierlo per suddividere l'albero. Una volta scelto il primo attributo, le possibilità per ogni sotto-albero determinato dal valore dell'attributo sono 2:

[2]Prodotti dallo stesso training set

- tutti gli esempi con quel valore hanno la stessa classe
- gli esempi con quel valore hanno classi diverse

Il primo caso è quello più auspicabile, in quanto la classe può già essere determinata. Nel secondo caso è possibile ripetere il procedimento trattando ogni sotto-albero come un nuovo problema di classificazione con un attributo in meno. Si può poi scegliere un nuovo "miglior attributo" tra quelli rimasti e proseguire così finché tutti gli esempi non siano stati classificati. Se ne deduce che la chiave per un buon algoritmo risiede in buona parte nella scelta dell'attributo migliore. La scelta dell'attributo non è l'unica componente che influisce sulla qualità dell'algoritmo ma di questo discuteremo più avanti.

La qualità di un albero prodotto da un determinato algoritmo dipende inoltre dal training set utilizzato. Per produrre un buon albero di decisione è necessario quindi disporre di un buon algoritmo e di un buon training set.

2.2.1 Training set

Il training set è l'insieme dei dati che costituisce l'esperienza su cui verrà addestrato l'albero di decisione. Solitamente viene rappresentato sotto forma di tabella in cui ad ogni riga corrisponde un'istanza e ad ogni colonna corrisponde un attributo di input, con l'aggiunta di una colonna per la classe corretta. Gli attributi di input possono essere di tre tipi:

- Numerici: hanno un valore numerico che può essere discreto o continuo.
- Ordinali: non hanno un valore numerico ma possono essere ordinati secondo un certo criterio (come ad esempio le medaglie olimpiche oro, argento e bronzo). A differenza dei numerici non sono caratterizzati da una nozione di distanza.
- Nominali: non hanno un valore numerico e non possono essere ordinati.

Differenti algoritmi di apprendimento possono essere in grado di trattare attributi di tipo diverso.

Per poter ottenere un albero con buona capacità di generalizzazione è buona norma che il training set contenga istanze che simboleggino il maggior numero di categorie di input possibile. In altre parole, è necessario che ci siano istanze appartenenti a tutte le classi previste e che per ogni attributo siano presenti tutti i valori.

Le istanze del training set devono inoltre essere coerenti e non ci devono essere gruppi di istanze con stessi valori degli attributi ma classi diverse. In caso contrario, durante la produzione dell'albero non sarebbe possibile assegnare univocamente una classe alla foglia raggiunta dalle istanze incoerenti.

Poiché spesso la dimensione del training set e la prestazione nella classificazione di nuove istanze è direttamente correlata, si tendono a preferire training set molto grandi. Nella pratica però è spesso necessario trovare un compromesso perché all'aumentare della dimensione del training set aumenta anche il tempo necessario per l'addestramento dell'albero. Inoltre spesso le classi per le istanze del training set sono determinate e registrate manualmente da esperti, pertanto produrre grandi training set richiede anche un lavoro umano non indifferente.

2.2.2 Scelta attributi

Ricordando che l'obiettivo dell'operazione di selezione dell'attributo è di trovare l'attributo che permette di raggiungere velocemente la classificazione del maggior numero di esempi, ne consegue che in prima analisi, l'attributo ideale è quello che da solo permette di classificare tutti gli esempi, mentre un attributo che lascia invariata la proporzione tra le classi all'interno dei suoi sotto-alberi può essere considerato superfluo.

Per poter determinare quali attributi siano utili e quanto, possiamo iniziare a pensare in termini di quantità di informazione, incertezza e probabilità. Per il concetto di incertezza faremo appello all'idea di entropia espressa da Shannon[19], secondo cui la quantità di informazione che otteniamo da un dato, dipende dalle conoscenze pregresse che si hanno a disposizione. In particolare, si può dire che dipenda dalla probabilità associata a quel dato.

Per tornare nel contesto dell'apprendimento di alberi di decisione, l'informazione da ottenere è la classe di una determinata istanza. La conoscenza pregressa risiede nella probabilità che l'attuale istanza appartenga ad una determinata classe. Il modo più intuitivo per calcolare questa probabilità consiste nel rapporto tra il numero di istanze che appartengono a quella classe e il numero totale di istanze nel training set.

Per semplicità immaginiamo un problema di apprendimento con attributi a valori discreti e due sole classi, denominate p e n per *Positivi* e *Negativi* Supponendo P_p e P_n rispettivamente come la probabilità che l'istanza appar-

tenga alla classe p e n, possiamo esprimere la quantità di informazione necessaria all'intero albero decisionale per determinare la classe corretta secondo la formula:

$$I(P_p, P_n) = -P_p log_2 P_p - P_n log_2 P_n$$

Ora possiamo scegliere casualmente un attributo e calcolare la quantità di informazione per ognuno dei sotto-alberi che vengono creati dalla suddivisione in base al valore dei suoi attributi.

Supponendo che l'attributo in questione (denominato A d'ora in avanti) abbia v possibili valori, possiamo ottenere la quantità di informazione necessaria dopo la suddivisione, attraverso la formula:

$$Resto(A) = \sum_{i=1}^{v} P_i I(P_{p_i}, P_{n_i})$$

dove P_i è la probabilità che un'istanza faccia parte dell' i-esimo sotto-albero e P_{p_i} e P_{n_i} indicano rispettivamente le probabilità P_p e P_n per l' i-esimo sotto-albero.
Detto questo è possibile trovare il **guadagno di informazione** ottenuto suddividendo l'albero in base all'attributo A, calcolando la differenza tra la quantità di informazione inizialmente necessaria e quella successiva alla suddivisione:

$$Guadagno(A) = I(P_p, P_n) - Resto(A)$$

Per trovare l'attributo migliore possiamo calcolare questa grandezza per ogni attributo e scegliere quello con il valore più alto.

In realtà però questo metodo ha un difetto evidente: nel caso di attributi con molti valori ammissibili la suddivisione produrrà sottoinsiemi molto piccoli (nel caso limite possiamo immaginare come attributo un identificatore univoco dell'istanza che produrrà sottoinsiemi contenenti un solo esempio) che hanno un alta probabilità di ottenere un guadagno di informazione elevato, mentre potrebbero non avere alcuna rilevanza ai fini della classificazione. Una possibile soluzione consiste nel limitare gli attributi a due soli valori ammissibili e a trattare gli attributi con più valori su più livelli. In questo modo si producono solo alberi binari, che però sono più difficili da interpretare da esperti umani in quanto tendono ad essere molto più profondi e si diramano più volte sugli stessi attributi.
Un'altra soluzione è quella di utilizzare una grandezza denominata rapporto di guadagno (gain ratio). Il rapporto di guadagno si basa anche sulla quantità di informazione che ci viene fornita da ogni valore di un attributo,

calcolata come:

$$I_V(A) = -\sum_{i=1}^{v} P_i log_2 P_i$$

Il rapporto di guadagno si calcola facendo il rapporto tra il guadagno e l'informazione per valore:

$$G_R(A) = \frac{Guadagno(A)}{I_V(A)}$$

All'interno di un algoritmo generalmente si usa il guadagno di informazione per un prima scrematura, poi si calcola il rapporto di guadagno degli attributi con guadagno superiore alla media per determinare il più adatto ed assicurarsi che non venga scelto un attributo con poco contenuto informativo.

Esistono anche altri metodi per calcolare l'utilità di un attributo come ad esempio l'indice di diverità Gini, usato nell'algoritmo CART[2] calcolato come

$$I_G(A) = 1 - \sum_{i=1}^{v} P_i^2$$

va a sostituire la quantità di informazione nel calcolo della qualità dell'attributo. Anche in questo caso si richiede una normalizzazione o uno sviluppo binario dell'albero per evitare di prediligere attributi con molti valori.

Secondo diversi studi[13] comunque, sembra che non ci sia un metodo migliore degli altri, semplicemente ogni metodo è il più adatto in determinate circostanze.

2.2.3 Dominio continuo

Tutti gli aspetti che abbiamo visto finora sono stati affrontati supponendo gli attributi di input e di output a valori discreti. Molto spesso però in problemi reali si richiede la capacità di gestire anche attributi numerici continui. Per poter inserire attributi di input continui all'interno delle tecniche viste è sufficiente definire delle soglie su cui i nodi effettueranno la suddivisione del training set. È stato dimostrato[8] che, ordinando le istanze in base ai valori dell'attributo continuo che si vuole valutare, le soglie migliori si presentano necessariamente tra due valori consecutivi le cui classi sono diverse. È possibile quindi scegliere i valori che presentano questa caratteristica come candidati, ed effettuare su questi il calcolo del guadagno di informazione o di

qualunque sia il criterio utilizzato per la scelta del migliore attributo. Una volta trovato il migliore lo si confronta con il guadagno di informazione relativo agli altri attributi (siano essi discreti o continui) e si seleziona il migliore. Questo metodo può essere esteso per consentire la suddivisione in più fasce di valori.
Questa tecnica fornisce buoni risultati in termini di correttezza degli alberi prodotti ma richiede un notevole incremento del tempo di calcolo, poiché la soglia deve essere calcolata nuovamente per ogni attributo in ogni nuovo sottoalbero prodotto.

Per quanto riguarda invece l'utilizzo di output a valori continui, occorre produrre un albero di regressione. Nella produzione di alberi di regressione si procede inizialmente con la suddivisione del training set come per gli alberi di classificazione, con la differenza che, non essendoci classi definite, si deve ricorrere ad un passaggio aggiuntivo per la scelta dell'attributo migliore. È necessario ordinare le istanze in base al valore dell'attributo di output e scegliere il punto o i punti di suddivisione che minimizzano l'errore quadratico medio. In base ai punti di suddivisione si producono delle categorie nell'attributo obiettivo che possono essere utilizzate come classi. La scelta dell'attributo migliore viene poi eseguita come per gli alberi di classificazione.

La produzione dell'albero però si ferma precocemente in base al criterio di arresto scelto (che verrà introdotto più avanti come tecnica di pruning) e si passa poi alla regressione lineare per trovare un valore da associare alle istanze appartenenti a ciascuna foglia. Il metodo più semplice è senza dubbio il calcolo della media dei valori dell'attributo obiettivo delle istanze appartenenti alla foglia e per foglie sufficientemente piccole fornisce una valutazione adeguata.

2.2.4 Attributi mancanti

Uno dei problemi che si possono incontrare in un caso di reale utilizzo di un sistema di apprendimento è la possibilità che per alcune istanze del training set non sia disponibile il valore di alcuni attributi. Questo incide sulla scelta dell'attributo migliore. È possibile seguire diversi approcci per affrontare questo problema.

Il più semplice consiste nell'ignorare, per il calcolo dell'indice di qualità di un attributo, tutte le istanze in cui quell'attributo non ha valore.

Altri metodi cercano di determinare il valore dell'attributo mancante in base al contesto.

Il più semplice di questi associa all'attributo il valore più frequente o, in alternativa, il valore più frequente per le istanze con la stessa classe.

Un altro metodo utilizza un approccio ad albero di decisione, ovvero l'attributo mancante viene considerato come attributo obiettivo e la classe come un normale attributo. Viene poi addestrato un albero usando come training set le istanze in cui quell'attributo è presente, e viene utilizzato per assegnare un valore agli attributi mancanti.

Tra i metodi che tentano di determinare il valore dell'attributo mancante, quello che sfrutta un albero di decisione sembra essere leggermente più preciso degli altri, ma la qualità complessiva non è comunque tale da consigliare questo metodo[14].

L'altro problema da affrontare quando mancano valori nel training set è la classificazione. Anche per classificare un'istanza con attributi mancanti è possibile assegnare all'attributo un valore secondo i metodi visti sopra. Solitamente però è preferibile seguire un metodo che riduce la probabilità di errore: quando ci si trova ad un nodo il cui attributo non è presente nell'istanza corrente è possibile seguire tutti i rami uscenti da quel nodo e proseguire fino alle foglie per poi scegliere la classe per maggioranza. È anche possibile modificare questo metodo per permettere di associare un peso ad ogni ramo uscente proporzionale alla distribuzione dei valori di quell'attributo. In questo modo la classe verrà scelta per maggioranza ma con il valore di ogni foglia moltiplicato per il peso assegnato.

2.2.5 Rumore

Un altro possibile problema da affrontare consiste nel avere a disposizione training set "rumorosi", ovvero in cui alcuni valori sono errati. Questo può portare ad avere coppie di istanze uguali negli attributi ma con diversa classificazione e rendere più difficile il compito di produrre un albero. Per poter addestrare alberi di decisione partendo da training set rumorosi è necessario che l'algoritmo possa accettare anche attributi inadeguati (ovvero con i quali si possono avere istanze incoerenti) e sia in grado di decidere di fermare preventivamente l'esplorazione quando si ritiene che le istanze rimaste siano soggette a rumore. Per l'arresto preventivo è possibile scegliere uno dei criteri di pruning proposti nella sezione 3.1.1, mentre per poter gestire le istanze in conflitto è possibile scegliere di associare alle foglie in cui giungono istanze uguali ma con classi diverse, la classe più presente tra le istanze che raggiungono la foglia.

2.2.6 Sovradattamento

Il principale difetto riscontrabile nell'apprendimento automatico di alberi di decisione è la suscettibilità al sovradattamento (overfitting). Il sovradattamento si presenta quando si hanno molti più attributi di quelli necessari e pochi esempi nel test set. Si palesa come un albero che trova relazioni inesistenti tra attributi e classi. Questo non solo produce alberi più estesi del necessario, ma queste relazioni inesistenti causeranno errori di classificazione nel test set in quanto attributi in realtà ininfluenti potrebbero essere scelti per la suddivisione.
È necessario inoltre trovare il giusto equilibrio quando si cerca di rimediare al sovradattamento per evitare di trovarsi di fronte al sottoadattamento (underfitting) in cui l'albero è troppo ridotto e di conseguenza aumenta l'errore di classificazione.

Alcune soluzioni al problema del sovradattamento vengono affrontate nel prossimo capitolo.

Capitolo 3

Tecniche avanzate

In questo capitolo vengono illustrate alcuni dei metodi che sono stati proposti per risolvere il problema del sovradattamento o, in generale, per migliorare le prestazioni degli algoritmi di apprendimento di alberi di decisione, in termini di capacità di generalizzazione.

3.1 Pruning

La soluzione più adottata per risolvere il problema del sovradattamento viene chiamata pruning (o potatura) e consiste nella riduzione delle dimensioni degli alberi per eliminare le scelte causate dal sovradattamento. È in genere possibile classificare i metodi di pruning in base al momento in cui avviene la potatura e alla direzione che si segue per eseguirla. Secondo il momento si possono suddividere gli algoritmi in due categorie:

- Pre-pruning, come metodo per evitare di esplorare i sottoalberi che risulterebbero sovradattati al training set, a tempo di produzione dell'albero
- Post-pruning, come metodo per rimuovere i sottoalberi errati o superflui, successivamente alla produzione dell'albero

Scegliendo di classificare in base alla direzione possiamo avere altre due categorie:

- Top-Down, quando si parte dalla radice e si seguono i rami dell'albero finché non si trova un sottoalbero da eliminare
- Bottom-Up, quando si parte dalle foglie e si risale finché non si trova il punto in cui si ritiene conveniente "potare" l'albero

È importante sottolineare che l'operazione di pruning comporta un peggioramento nella capacità di classificare le istanze del training set, nella speranza di migliorare la precisione nella classificazione di nuove istanze. Ciò significa che, nonostante produca un albero ridotto rispetto all'originale, è opportuno effettuare il pruning solo nei casi che lo richiedono.
In seguito sono discussi alcuni metodi di pruning proposti in Quinlan (1987)[15] e Rokach, Maimon (2015)[16].

3.1.1 Criterio di arresto

Il primo metodo che osserviamo è il più semplice tra quelli proposti, si tratta di un algoritmo di pre-pruning di tipo top-down. L'idea alla base è molto semplice: si interrompe l'espansione dell'albero quando si incontrano determinati criteri. Questi sono alcuni dei criteri comuni:

- È stata raggiunta la massima profondità consentita dell'albero
- Il numero di casi nei nodi terminali è inferiore al minimo consentito
- Se il nodo fosse espanso, il numero di casi nei nodi terminali sarebbe inferiore al minimo consentito
- Il criterio scelto per la selezione dell'attributo migliore restituisce un valore inferiore alla soglia minima

Queste soglie vanno decise a priori e influenzano direttamente la dimensione dell'albero. Per questo motivo, sebbene sia questo il metodo più semplice, risulta piuttosto inefficiente nell'effettuare un pruning adatto alla situazione e rischia di sfoltire più o meno del necessario, in base a quanto sia restrittivo o meno il criterio d'arresto.

3.1.2 Reduced-Error Pruning

Il Reduced-Error Pruning (a riduzione d'errore) può essere definito come un algoritmo di post-pruning di tipo bottom-up.
Il funzionamento è molto semplice: si parte dalle foglie e

1. si segue il ramo fino al nodo precedente
2. si elimina il nodo e si sostituisce con una foglia contenente una classe decisa in base alla maggioranza delle istanze che raggiungono quel nodo
3. si controlla l'errore di classificazione sul test set

- Se l'errore è diminuito si mantiene la modifica e si prosegue tornando al punto 1
- Se l'errore è aumentato si annulla la modifica

4. Se sono già state esplorate tutte le altre foglie ci si ferma, altrimenti si ritorna al punto 1 ripartendo da una foglia inesplorata

Questo algoritmo è molto rapido in quanto effettua un solo passaggio su ogni nodo ma necessita di un test set diverso dal training set e per questo motivo tende ad eliminare i sottoalberi che contribuiscono alla classificazione di particolari categorie di istanze presenti nel training set ma non nel test set.

3.1.3 Cost-Complexity Pruning

L'algoritmo di Cost-Complexity Pruning (costo-complessità) è un algoritmo di post-pruning. Il suo funzionamento si suddivide in due fasi. Nella prima fase si producono una serie di alberi $T_0, T_1, ..., T_k$, con T_0 come albero originale e ogni T_{i+1} generato sostituendo uno o più sottoalberi di T_i con le foglie adeguate, fino ad arrivare ad avere T_k come albero costituito da una sola foglia. Possiamo definirlo di tipo bottom-up in quanto si parte da un albero completo e si producono man mano alberi sempre più sfoltiti fino ad avere soltanto la radice. Nella seconda fase si valutano questi alberi e si seleziona uno di questi come soluzione finale.

Definiamo T l'albero decisionale prodotto da un training set composto da N casi, e definiamo E come il numero di errori di classificazione sul training set. Se $L(T)$ è il numero di foglie dell'albero T allora il costo-complessità è definito come

$$\frac{E}{N} + \alpha \cdot L(T)$$

per un parametro α.
Sostituendo ad un sottoalbero S la foglia più adatta, andremo ad incrementare gli errori di classificazione di un valore M ma avremo $L(S) - 1$ foglie in meno. In questo modo il nuovo albero avrebbe lo stesso costo-complessità dell'albero originale se

$$\alpha = \frac{M}{N \cdot (L(S) - 1)}$$

Per produrre T_{i+1} da T_i esaminiamo ogni sotto albero di T_i per trovare quello con il valore minimo di α. Il sottoalbero (o i sottoalberi) con quel valore di α viene sostituito dalla foglia corrispondente.

La seconda fase dell'algoritmo si basa sull'errore di classificazione nel test set. Supponiamo di disporre di un test set contenente N' casi e di utilizzarlo per testare ogni albero T_i. Chiamiamo E' l'errore minimo ottenuto e definiamo l'errore standard di E' come

$$se(E') = \sqrt{\frac{E' \cdot (N' - E')}{N'}}$$

Infine viene scelto come albero finale non quello con il minor numero di errori di classificazione, bensì il più piccolo albero in cui

$$E \leq E' + se(E')$$

3.1.4 Pessimistic Pruning

Il pessimistic pruning (a stima pessimistica) è un algoritmo di pre-pruning di tipo top-down.
Supponiamo di utilizzare l'albero originale T per classificare il training set composto da N istanze. Prendiamo un sottoalbero S di T e ad ogni foglia di S associamo le K istanze di cui quella foglia ha determinato la classe. Immaginiamo, poi, che J di queste K istanze siano state classificate erroneamente e definiamo ΣK e ΣJ le somme di K e J all'interno di S. Secondo una stima pessimistica, S potrà classificare in modo errato E' nuove istanze su ΣK nuovi casi, con

$$E' = \Sigma J + \frac{L(S)}{2}$$

Chiamiamo E il numero effettivo di errori che otteniamo sostituendo ad S la sua foglia migliore.
Il metodo di pruning pessimistico sostituisce ad S la sua foglia migliore quando

$$E + \frac{1}{2} \leq E' + se(E')$$

Questo metodo risulta molto rapido in quanto richiede di visitare al massimo una volta ciascun sottoalbero ed essendo di tipo top-down non richiede di visitare i nodi appartenenti ad un sottoalbero già eliminato. Inoltre non richiede un test set diverso dal training set.

3.1.5 Pruning chi quadro

Il χ^2 pruning è un metodo di pruning che può essere usato in modalità bottom up per il post pruning, oppure in modalità top down come pre pruning se utilizzato come criterio di arresto. In questo metodo si cerca di valutare

la dipendenza della classe da ogni attributo. Supponiamo di voler valutare l'attributo A partendo dal presupposto che questo attributo sia irrilevante ai fini della classificazione. Questo attributo suddivide l'insieme D di istanze che lo raggiungono, in v data set distinti che chiamiamo D_i. Immaginando per semplicità due sole classi di output, per ogni D_i ci saranno p_i istanze con classe p e n istanze con classe n. Se l'attributo è davvero irrilevante i valori attesi di p_i e n_i saranno rispettivamente p'_i e n'_i calcolati come

$$p'_i = P * \frac{p_i + n_i}{P + N} \qquad n'_i = P * \frac{p_i + n_i}{P + N}$$

con P e N rispettivamente i numeri di istanze con classe p e n nell'insieme D. Calcoliamo poi la distribuzione chi quadro a $v - 1$ gradi di libertà

$$\chi^2 = \sum_{i=1}^{v} \frac{(p_i - p'_i)^2}{p'_i} + \frac{(n_i - n'_i)^2}{n'_i}$$

Nel caso in cui l'attributo A sia puramente irrilevante i valori attesi corrispondono ai valori effettivi e $\chi^2 = 0$. All'aumentare del valore di χ^2 aumenta la deviazione dai valori attesi e aumenta la rilevanza dell'attributo. Ora non resta che scegliere una soglia c di deviazione massima e se $\chi^2 < c$ il sottoalbero prodotto dall'attributo A viene eliminato e sostituito da una foglia. Modificando il valore della soglia è possibile stabilire il livello di pruning desiderato: più è alta la soglia, più l'albero verrà potato.

3.1.6 Semplificazione delle regole di produzione

La semplificazione delle regole di produzione è un metodo diverso da quelli visti finora. Sfrutta la caratteristica degli alberi di decisione di poter essere tradotti in una serie di regole di produzione. In un albero infatti, ogni percorso dalla radice ad una foglia può essere tradotto in una regola in formato $X_1 \wedge X_2 \wedge \cdots \wedge X_n = Class_y$ dove ogni X_i è una condizione su un attributo. Nella prima fase di questo metodo si traduce l'albero in una serie di regole di produzione. Nella seconda fase si valuta la rilevanza di ogni condizione X_i quando tutte le altre condizioni in una regola sono soddisfatte. Per ogni regola, viene poi eliminata la condizione con la rilevanza minore, a patto che non sia superiore ad una soglia prestabilita.

Nella terza fase si valuta la qualità delle regole prodotte e si assegna ad ogni regola un valore proporzionale all'affidabilità della stessa.

Nella quarta fase si classificano le istanze del training set utilizzando la regola che combacia con ciascuna istanza. Se ci sono più regole conformi

con l'istanza, si sceglie quella con l'affidabilità più alta. Se non ci sono regole conformi con l'istanza, si assegna la classe più presente nel training set. Viene poi valutata la qualità generale della classificazione rimuovendo una regola. Se la qualità non peggiora la regola viene rimossa e si ripete.

Questo metodo non produce quindi un vero e proprio albero di decisione ma un insieme di regole. Anche in questo caso è possibile determinare quanto deve essere semplificato l'albero, modificando la soglia per l'eliminazione delle condizioni.

3.1.7 Altri

I metodi esposti fin qui sono solo alcuni dei più conosciuti e utilizzati. Esistono molti altri metodi e spesso nuovi algoritmi propongono nuovi metodi, che possono essere originali o basati su alcuni di questi ma implementati in un nuovo modo. Altri metodi conosciuti possono essere:

- Minimum Description Length Pruning: misura la dimensione dell'albero con il numero di bit necessari a codificarlo e sceglie i rami da eliminare cercando di ridurre la dimensione in bit.
- Error Based Pruning: è un evoluzione del pessimistic pruning implementata da Quinlan nell'algoritmo C4.5.

3.2 Validazione incrociata

La validazione incrociata è una tecnica che consiste nell'estrarre dal training set una frazione dei dati ed utilizzarli per verificare la capacità di predire eventi sconosciuti dell'albero prodotto. I risultati di questa verifica andranno poi a determinare quale sarà, tra quelli prodotti, l'albero finale.
Questo albero deve essere poi testato su un test set differente per evitare di incorrere nel problema del peeking, ovvero una situazione in cui le prestazioni dell'albero sul test set influiscono direttamente nella determinazione dell'albero, per cui quello che si verrebbe a creare non sarebbe l'albero che rispecchia meglio il training set ma quello che rispecchia meglio il test set (rendendolo di fatto parte del training set).

Una modalità molto diffusa di validazione incrociata è detta k-fold validation e si applica suddividendo il training set in k sottoinsiemi con la stessa dimensione, ripetendo il processo di validazione k volte, estraendo ogni volta un sottoinsieme diverso da utilizzare come test set. Viene infine scelto

l'albero con la migliore prestazione sul proprio test set. Un caso estremo è il cosiddetto *leave-one-out cross-validation* in cui si eseguono n esperimenti estraendo ogni volta un solo oggetto come test set.
La validazione incrociata richiede un training set ampio per evitare che la riduzione nel numero di istanze utilizzate per l'addestramento influisca negativamente sulla qualità di ciascun albero. È inoltre una tecnica che tende a migliorare le prestazioni comportando però un incremento del costo computazionale (ad esempio adottando un sistema leave-one-out si moltiplica il tempo di elaborazione per $n - 1$).

Un fattore da tenere in considerazione però è la ridondanza prodotta da questo metodo, in quanto generando k alberi basati su training set molto simili, ogni istanza del set viene utilizzata dall'algoritmo di apprendimento k volte (sempre pensando al caso limite del leave-one-out, ogni istanza viene ignorata una sola volta sulle n iterazioni).
Blockeel e Struyf (2002) [1] si sono occupati proprio di questo aspetto, con l'obiettivo di ridurre le ridondanze nel calcolo della qualità degli attributi in modo tale da diminuire anche l'impatto complessivo della validazione incrociata sul costo computazionale dell'algoritmo di produzione dell'albero di decisione.

Il metodo usato si inserisce nel momento del calcolo dell'indice di qualità degli attributi. Dal momento che un albero e il successivo avranno in comune la maggior parte degli esempi, la qualità degli attributi (calcolata attraverso uno dei metodi visti nella sezione 2.3) verrà determinata da data set molto simili, per cui è possibile sfruttare questa caratteristica per ignorare le sovrapposizioni.

Nel caso in cui si utilizzi per il calcolo della qualità degli attributi una grandezza additiva, ovvero in cui l'indice di qualità di un attributo su un data set può essere calcolato a partire dagli indici dell'attributo nei vari subset che compongono il data set di partenza (ad esempio il guadagno di informazione o il rapporto di guadagno), è possibile calcolare, per ogni attributo, gli indici $I(D_i)$ per ogni singolo data set D_i e sommarli per ottenere l'indice per l'intero training set $I(T) = \sum_{i=1}^{k} I(D_i)$. Si può poi calcolare l'indice per la partizione corrente come $I(T) - I(D_i)$ dove D_i è la partizione utilizzata come test set dell'esperimento corrente.
Siccome ogni esempio appare una sola volta in ciascuna partizione D_i, il calcolo degli indici deve avvenire solo n volte (n è il numero totale di istanze del training set) invece di $n \cdot k$.

È inoltre possibile utilizzare una tecnica simile anche quando l'indice di qualità degli attributi non viene calcolato tramite una grandezza additiva. In questo caso la riduzione in costo computazionale è limitata e dipende molto dalla distribuzione delle istanze all'interno delle partizioni. In particolare, trae vantaggio dai casi in cui le partizioni hanno distribuzioni simili delle istanze e di conseguenza gli alberi prodotti tendono a scegliere gli stessi attributi per la suddivisione.

Queste metodi permettono quindi di applicare la validazione incrociata riducendo l'incremento computazionale che ne deriva.

3.3 Ensemble learning

L'ensemble learning è una tecnica in cui vengono prodotti più alberi decisionali per lo stesso problema. Questi alberi vengono poi combinati per ottenere un classificatore che dovrebbe rivelarsi preciso almeno quanto il migliore tra gli alberi che lo costituiscono.
Questo tipo di strategia permette di diminuire la probabilità di classificare erroneamente una nuova istanza.

Per una definizione formale ci si può ricondurre al teorema della giuria di Condorcet[5], secondo cui in una giuria di votanti, in cui l'esito della votazione è binario e in cui siano p e M rispettivamente la probabilità che ha ogni votante di votare correttamente e la probabilità che il voto di maggioranza sia corretto, allora

- se $p > 0.5 \Rightarrow M > p$
- M tende a 1 se $p > 0.5$ e il numero di votanti tende ad infinito

Questo teorema si basa su due assunzioni fondamentali: i voti dei giurati sono indipendenti e ci sono solo due possibili risultati. Sebbene queste premesse siano piuttosto restrittive, questo teorema permette di concludere che, sotto determinate condizioni, è possibile assicurarsi una risposta corretta utilizzando un numero adeguato di votanti, la cui capacità di giudizio sia leggermente migliore di una decisione casuale.

I sistemi che abbiamo visto finora, che hanno un elevata probabilità di produrre un ipotesi che sia corretta su tutte le istanze tranne una frazione arbitrariamente piccola, sono detti *strong learner*. I sistemi che invece producono un ipotesi che opera solo leggermente meglio di una supposizione

casuale sono detti *weak learner*.
L'esempio più semplice di weak learner nell'apprendimento di alberi di decisione è rappresentato dai ceppi decisionali (decision stump). Si tratta di alberi composti dal solo nodo radice in cui ad ogni foglia è associata la classe più presente nel sottoinsieme corrispondente.

Per quanto riguarda l'apprendimento di alberi di decisione non possiamo ovviamente assumere che il risultato della classificazione da parte di diversi alberi prodotti dallo stesso training set sia completamente indipendente, ma utilizzando alcuni accorgimenti per produrre alberi semplici ed eterogenei, possiamo comunque ottenere risultati comparabili a quelli dimostrati dal teorema della giuria di Condorcet.

3.3.1 Caratteristiche

Un sistema di ensemble learning richiede i seguenti componenti:

- un training set;
- un algoritmo di induzione, che produce un albero di decisione a partire da un training set;
- un generatore di ensemble, che determina il metodo con cui vengono diversificati gli alberi, ad esempio, partizionando il training set;
- un combinatore, che prende l'output di diversi alberi sotto forma di classificazioni di un istanza e le combina per ottenere il risultato finale;

In questa sezione utilizzeremo la seguente notazione: T_i indica l'i-esimo albero di un ensemble, D_i indica l'i-esimo sottoinsieme del training set e I è l'algoritmo di induzione. In generale, poiché l'albero T_i è prodotto da I a partire dal data set D_i, si ha $T_i = I(D_i)$.
Le caratteristiche di ogni componente e le relazioni che li legano andranno a costituire il modello del sistema di apprendimento.
Le proprietà fondamentali del sistema possono essere sintetizzate nelle seguenti:

- Dipendenza tra classificatori: l'addestramento di un albero influisce sull'addestramento degli altri.
- Diversità: per ottenere un ensemble più efficace è necessario produrre alberi il più possibile diversi tra loro. La diversità può anche essere ottenuta a scapito dell'affidabilità dei singoli alberi.

- Dimensione dell'ensemble: il numero di classificatori dell'ensemble.

La dipendenza tra classificatori e la diversità sono in realtà spesso legate tra loro, poiché molti dei metodi più efficaci per generare diversità, inducono dipendenza tra i classificatori (esistono anche alcuni metodi per evitare questa conseguenza, come vedremo in seguito).
La dimensione dell'ensemble invece influisce principalmente sul tempo necessario a produrlo e sui problemi di apprendimento su cui sarà applicabile un determinato metodo.

È possibile classificare gli algoritmi di ensemble learning in base al metodo di combinazione. Esistono due categorie principali: metodi a votazione pesata e metodi di meta-apprendimento.
I metodi a votazione pesata utilizzano solitamente alberi prodotti in modo indipendente, poiché sulla produzione di un albero non influiscono gli alberi prodotti precedentemente. Gli alberi prodotti eseguono poi la classificazione in modo competitivo.
I metodi di meta-apprendimento invece producono alberi in modo dipendente, poiché producono nuovi alberi basati sulle prestazioni degli alberi già prodotti. In questo caso spesso gli alberi eseguono la classificazione in modo cooperativo.

3.3.2 Metodi a votazione pesata

I metodi a votazione pesata assegnano a ciascun classificatore un peso, che indica quanto il proprio risultato influisce sulla decisione finale. Qui sono descritti i principali metodi di combinazione a votazione pesata.

3.3.2.1 Votazione a maggioranza

In questo sistema, gli alberi vengono prodotti suddividendo il training set, ad esempio come per la validazione incrociata, e la classe finale viene scelta in base a quella che ha ottenuto il maggior numero di voti. È anche conosciuto come voto di pluralità o metodo base di ensemble.
Matematicamente può essere scritto come

$$class(x) = argmax(\sum_i g(y_i(x), c))$$

dove $y_i(x)$ è la classe scelta dall'albero T_i e $g(y, c)$ è una funzione definita come

$$g(y,c) = \begin{cases} 1 & \text{se } y = c \\ 0 & \text{se } y \neq c \end{cases}$$

3.3.2.2 Somma distribuzione

Anche in questo caso gli alberi vengono prodotti suddividendo il training set. La classe viene scelta in base al valore più alto all'interno del vettore probabilità condizionale ottenuto da ogni albero. Può essere scritto come

$$class(x) = argmax \sum_i P_{T_i}(y = c|x)$$

dove $P_{T_i}(y = c|x)$ è la probabilità che $y = c$ data l'istanza x per l'albero T_i. Può essere utilizzato quando alle foglie non sono associate singole classi ma vettori contenenti le probabilità per ogni classe.

3.3.2.3 Bagging

Il bagging[3] (Bootstrap Aggregating) è un metodo in cui vengono prodotti diversi alberi e la selezione dell'output avviene per votazione a maggioranza negli alberi di classificazione o come media negli alberi di regressione.

La peculiarità di questo metodo sta nel modo in cui vengono selezionati i training set dei singoli alberi. Per produrre ciascun training set, viene effettuato un campionamento casuale con ripetizione sul training set originale. Campionamento casuale con ripetizione significa che vengono selezionate casualmente un determinato numero di istanze, con la possibilità che la stessa istanza venga campionata più volte per lo stesso subset. In questo modo viene aumentata la diversità degli alberi prodotti. Per sfruttare al meglio questo metodo, sempre per promuovere la diversità negli alberi prodotti, si consiglia di utilizzare un algoritmo di induzione molto suscettibile alle variazioni del training set.

È stato verificato che le prestazioni, rispetto all'uso di un singolo albero, migliorano con l'utilizzo di algoritmi adattabili mentre possono anche peggiorare con l'utilizzo di algoritmi ad elevata stabilità.

3.3.2.4 Random Forest

Il metodo random forest[4] (o a foresta casuale) è un evoluzione del bagging che punta a differenziare ulteriormente gli alberi prodotti e quindi a

renderli ancora meno correlati tra loro. Per ottenere questo effetto, viene modificato l'algoritmo di induzione in modo tale da permettere, a tempo di produzione degli alberi, di selezionare casualmente un sottoinsieme di attributi e di scegliere per la suddivisione, l'attributo migliore tra questi. In questo modo si riduce la qualità dei singoli alberi ma, aumentando la diversità, si migliora la prestazione complessiva dell'aggregato.

Sono state testate anche implementazioni in cui il sottoinsieme di attributi contiene un solo elemento (ovvero l'attributo migliore viene scelto casualmente) e hanno mostrato prestazioni paragonabili agli algoritmi di boosting, richiedendo però un tempo notevolmente inferiore per la produzione degli alberi. È stato inoltre provato che non è soggetto a sovradattamento.

Il metodo a foresta casuale prende il nome dal fatto che viene solitamente utilizzato con un numero elevato di alberi, ma nonostante debba produrre più alberi di altri algoritmi rimane comunque più veloce, in quanto la parte più dispendiosa dell'algoritmo, ovvero la scelta del miglior attributo viene notevolmente semplificata se non addirittura eliminata.

3.3.3 Metodi di Meta-Apprendimento

Meta-apprendimento significa apprendere dagli alberi prodotti e dalle classificazioni che questi alberi effettuano sul training set. Si tratta quindi di un tipo di aggregato dipendente in cui vengono prodotti nuovi classificatori a partire dagli alberi di base. Seguono alcuni dei principali metodi di meta-apprendimento.

3.3.3.1 Stacking

Lo stacking è una tecnica utilizzata per ottenere un'alta precisione nella generalizzazione. Questo metodo cerca di valutare l'affidabilità degli alberi prodotti ed è solitamente usato per combinare alberi prodotti tramite algoritmi diversi.
L'idea consiste nel creare un meta-dataset contenente un'istanza per ogni istanza del dataset originale, in cui però si sostituiscono gli attributi originali con le classificazioni prodotte da ciascun albero, mentre l'output rimane la classe originale. Queste nuove istanze vengono poi utilizzate per produrre un meta-classificatore che combina le diverse previsioni in una sola.
Si suggerisce di dividere il training set originale in due subset. Il primo utilizzato per la creazione del meta-dataset e il secondo usato per produrre i

classificatori di base.
Le previsioni del meta-classificatore andranno quindi a riflettere le effettive prestazioni degli algoritmi di induzione di base.
Durante la classificazione di una nuova istanza, ogni albero di base produce la sua previsione. Queste previsioni andranno a costituire una nuova istanza che verrà classificata dal meta-classificatore.

Nel caso in cui gli alberi producano una classificazione a probabilità, è possibile incrementare le prestazioni dello stacking inserendo nelle nuove istanze le probabilità espresse da ogni albero per ogni classe.
È stato dimostrato come le prestazioni dello stacking siano almeno paragonabili a quelle del miglior classificatore scelto tramite validazione incrociata.

3.3.3.2 Arbiter Trees

Il metodo arbiter trees[6] produce un albero aggiuntivo detto arbitro e lo usa insieme agli alberi di base per scegliere la classe finale.
Per produrre l'arbitro si utilizzano gli alberi di base per classificare il training set. Si produce il training set da cui verrà indotto l'arbitro basandosi sulla regola di selezione, che ha il compito di scegliere quali istanze del training set iniziale andranno a costituire il training set dell'arbitro, in base alle classificazioni espresse dagli alberi base. L'arbitro viene poi prodotto utilizzando un algoritmo di induzione ed il nuovo training set così creato.

Le prestazioni di questo metodo dipendono quindi dalla regola di selezione e dalla regola di arbitraggio.
Le principali regole di selezione sono:

1. Le istanze in cui le classificazioni sono diverse (Gruppo 1)

$$D_{A_1} = \{x \in D | (T_i(x) \neq T_j(x), i \neq j\}$$

2. Come il gruppo 1 ma con l'aggiunta delle istanze in cui le classificazioni sono uguali ma errate (Gruppo 2)

$$D_{A_2} = D_{A_1} \cup \{x \in D | T_i(x) = T_j(x) \vee Class(x) \neq T_j(x), \forall i, i \neq j\}$$

Durante la classificazione di un'istanza si fa uso della regola di arbitraggio, che consiste nella votazione a maggioranza da parte degli alberi base e dell'arbitro, con i pareggi risolti a favore della classe scelta dall'arbitro.

Esiste anche una versione più evoluta di questo metodo, detta a multi-livello[7]. Si suddivide il training set in k sottoinsiemi disgiunti. Un arbitro viene indotto da ogni coppia di classificatori, e un nuovo arbitro viene creato ricorsivamente da due arbitri dello stesso livello. Di conseguenza, per k classificatori vengono generati $log_2(k)$ livelli di arbitri.

Il procedimento per la produzione degli arbitri è simile a quello del metodo base.

1. Data una coppia di classificatori, si uniscono i corrispondenti data set e si usano i due alberi per classificare l'unione dei set.

2. La regola di selezione confronta i risultati dei due alberi e seleziona le istanze dell'unione che andranno a formare il training set dell'arbitro.

3. L'arbitro viene prodotto dall'algoritmo usato per gli alberi di base partendo dal training set composto al punto 2.

Si ripetono i passi dall'1 al 3 finché non sono stati utilizzati tutti gli alberi di base, poi si ricomincia partendo da coppie di arbitri. L'algoritmo termina quando viene prodotto l'arbitro radice.

In questo modello si aggiunge una regola di selezione a quelle menzionate in precedenza:

3. Come il gruppo 2 ma con l'aggiunta delle istanze in cui le classificazioni sono uguali e corrette (Gruppo 3)

$$D_{A_3} = D_{A_2} \cup \{x \in D_i \cup D_{i+1} | T_i(x) = T_{i+1}(x) \vee Class(x) = T_i(x)\}$$

dove D_i e D_{i+1} sono i training set degli alberi T_i e T_{i+1} (siano essi alberi base o arbitri). Le istanze vengono però suddivise in base al gruppo a cui appartengono (discordi, concordi errate o concordi corrette). Vengono quindi suddivise in tre sottoinsiemi tramite cui vengono prodotti tre arbitri, A_c, A_{ce} e A_{cc}.

La classificazione in questo caso avviene seguendo un percorso bottom-up dai classificatori di base, all'arbitro radice. Ogni sistema composto da una coppia di classificatori e l'arbitro corrispondente sceglie in parallelo la classe per l'istanza. La regola di arbitraggio, sceglie qual'è la classe finale scelta da

ogni sistema. Questa classe verrà poi utilizzata insieme a quella dell'arbitro adiacente e all'arbitro prodotto da questa coppia, per decidere quale classe passerà al livello superiore. Il procedimento prosegue finché non si ottiene la classe scelta dal sistema che comprende l'arbitro radice.

La regola di arbitraggio varia in base alla regola di selezione utilizzata, in particolare

- Per le regole 1 e 2, la classe finale viene scelta come nel metodo base, ovvero tramite un voto di maggioranza espresso dalla coppia di alberi e dall'arbitro corrispondente

- Per la regola 3, se le classificazioni dei due alberi sono diverse, viene scelta la classificazione fatta dall'arbitro A_d. In caso contrario, se concordano con la classe scelta dall'arbitro A_{cc} viene utilizzata questa. Altrimenti viene scelta la classificazione di A_{ce}.

3.3.3.3 Combiner Trees

Il metodo denominato a combinazione[6], utilizza, come il metodo stacking, le previsioni effettuate dai classificatori di base per produrre un training set per l'algoritmo di meta-apprendimento. In questo caso il contenuto delle istanze viene determinato attraverso la regola di combinazione.
Durante la classificazione di una nuova istanza vengono innanzitutto generate le previsioni degli alberi di base, poi da queste previsioni viene generata una nuova istanza che viene classificata dal combinatore.
Lo scopo di questa strategia è di unire le previsioni degli alberi base imparando a conoscere le relazioni tra queste previsioni e quella corretta.

Sono state proposte tre strategie, ognuna basata su una differente regola di combinazione per la produzione del nuovo training set:

1. Le istanze sono composte dalla classe corretta e dalle classi scelte dagli alberi base

$$D_1 = \{Class(x), T_1(x), \ldots, T_n(x) | \forall x \in D\}$$

2. Le istanze sono simili a quelle del primo gruppo ma con l'aggiunta degli attributi originali

$$D_2 = \{Class(x), T_1(x), \ldots, T_n(x), attrvec(x) | \forall x \in D\}$$

3. Le istanze sono composte dalla classe corretta e da n set di m previsioni provenienti da classificatori binari, dove n è il numero di alberi di base e m è il numero di classi.

$$D_3 = \{Class(x), T_{1_1}, \ldots, T_{1_m}, \ldots, T_{i_j}, \ldots, T_{n_1}, \ldots, T_{n_m} | \forall x \in D\}$$

Ogni previsione T_{i_j} è prodotta da un classificatore binario addestrato su istanze in cui le classi sono denominate j e $\neg j$. In altre parole, si producono alberi di base specializzati per classe (ovviamente nel caso in cui ci siano due sole classi, questa regola non offre alcun vantaggio).

Una volta generato il nuovo training set, questo viene utilizzato per produrre il combinatore. Quando incontreremo una nuova istanza, gli alberi base genereranno le loro previsioni e verrà prodotta una nuova istanza seguendo la stessa regola di combinazione usata per produrre il training set del combinatore. Il combinatore infine deciderà la classe finale di questa istanza.

3.3.3.4 Grading

Il grading [18] è un metodo che produce un nuovo classificatore per ogni albero di base e lo usa per determinare se le previsioni di quest'ultimo siano corrette. Gli alberi di base vengono prodotti tramite validazione incrociata e vengono utilizzati per classificare tutte le istanze del training set. Viene creato un nuovo training set per ogni albero base, in cui la classe delle istanze è sostituita da una classe che indica se l'albero ha classificato correttamente l'istanza.
Da ciascuno di questi training set viene prodotto un nuovo albero, che ha l'obiettivo di prevedere quando l'albero di base sbaglierà.

Per classificare una nuova istanza si utilizzano gli alberi di base per definire le possibili classi e gli alberi derivati per decidere quali degli alberi (e quindi delle classi proposte) siano affidabili. Viene scelta la classe prevista dagli alberi ritenuti affidabili per quella istanza e in caso di conflitto (più alberi affidabili propongono classi diverse) si ricorre ad una votazione.

Questo algoritmo può essere interpretato come un evoluzione della validazione incrociata, in cui non si sceglie un solo albero ma si valuta istanza per istanza qual'è l'albero più adatto.

3.3.3.5 Boosting

Nel boosting vengono utilizzati training set con istanze pesate ovvero in cui durante l'apprendimento di un ipotesi, l'importanza di ogni istanza è proporzionale al proprio peso. Il boosting richiede richiede quindi algoritmi di induzione in grado di gestire queste istanze pesate. Ad ogni modo, se l'algoritmo che si ha a disposizione non può gestire istanze pesate è possibile comunque ricorrere al boosting replicando le istanze proporzionalmente al peso assegnato.
Il procedimento è il seguente:

1. Viene prodotto un albero con un processo che considera più rilevanti le istanze con peso maggiore
2. L'albero prodotto viene utilizzato per classificare il training set
3. Il peso delle istanze classificate correttamente viene ridotto, mentre quello delle istanze classificate erroneamente viene incrementato
4. Si ripetono i passi da 1 a 3 finché non sarà stato prodotto un determinato numero di alberi
5. Si assegna ad ogni albero prodotto un peso proporzionale alle sue prestazioni sul training set.

Per la classificazione di nuove istanze, si usa un sistema a votazione pesata (in genere votazione a maggioranza) da parte di tutti gli alberi. Lo scopo di questo algoritmo è di produrre alberi diversi, per coprire un insieme più ampio di tipi di istanze. Il difetto di cui viene spesso accusato il metodo di boosting consiste nella suscettibilità al rumore. Se ci sono dati errati nel training set l'algoritmo di boosting tenderà a dare un peso sempre maggiore alle istanze che li contengono, portando inevitabilmente ad un peggioramento delle prestazioni. Per ovviare a questo problema sono stati proposti anche algoritmi di boosting in cui le istanze che vengono ripetutamente classificate erroneamente vengono interpretate come contenenti dati errati e di conseguenza viene ridotto il loro peso.

3.4 Apprendimento basato sulla rilevanza

Un tipo di approccio diverso al problema del sovradattamento è rappresentato dall'apprendimento basato sulla rilevanza (Relevance Based Learning - Decision Tree Learning o RBLDTL)[17]. Invece di cercare di ignorare gli attributi superflui come nel pruning, questo metodo tenta di determinare a

priori le dipendenze tra attributi e di produrre quindi un training set contenente soltanto gli attributi ritenuti necessari all'apprendimento.
In particolare si ricerca la *determinazione consistente minima* ovvero il set più piccolo di attributi che da soli sono sufficienti a determinare la classe di un istanza.

L'approccio più semplice per trovare la determinazione consistente minima consiste nel valutare ogni sottoinsieme di attributi a partire dai singoli attributi, fino ad arrivare all'intero insieme, e verificare se il sottoinsieme corrente è consistente con il training set. Il primo sottoinsieme consistente sarà anche quello minimo. Questo approccio però ha una complessità computazionale elevata, poiché, ipotizzando una determinazione minima di dimensione p su n attributi totali, questa verrà trovata solo quando verranno analizzati i sottoinsiemi di dimensione p. Il numero di tali sottoinsiemi è $\binom{n}{p} = O(n^p)$. Ciò significa che si tratta di un problema NP-completo e, di conseguenza, non è utilizzabile quando n e p sono elevati.
Le prestazioni nell'apprendimento però, mostrano come questo metodo permetta ad un algoritmo che produce un albero di decisione a partire da un training set con attributi irrilevanti, di generare un albero molto più preciso, con un training set di dimensioni molto ridotte.

In altre parole, questo metodo è molto efficace nelle situazioni in cui la classe viene determinata soltanto da un numero ridotto di attributi, ma ha una complessità tale da renderlo impraticabile quando la classe viene determinata da molti attributi. Inoltre, va sottolineato che, nel caso in cui tutti gli attributi contribuiscano alla determinazione della classe, questo metodo non apporta alcun vantaggio.

Capitolo 4

Algoritmi e prestazioni

In questo capitolo vengono mostrati alcuni degli algoritmi più conosciuti e utilizzati, vengono evidenziate le rispettive capacità e analizzate le prestazioni su alcuni problemi di apprendimento.
Le prove sono state eseguite attraverso l'uso di Weka[10], un software open source che permette di applicare diversi algoritmi di apprendimento automatico. I training set sono stati reperiti alla UC Irvine Machine Learning Repository[11].

Per ogni algoritmo, le prestazioni in ciascun training set sono state valutate in tre modi: estraendo una porzione pari al 40% del training set da utilizzare come test set, tramite validazione incrociata 5-fold e tramite validazione incrociata 10-fold. Per ogni test eseguito viene mostrata la percentuale di istanze correttamente classificate. È stato inoltre registrato per ogni algoritmo il tempo medio di produzione del classificatore con il training set più ampio.

4.1 Training Set

Sono stati scelti per le prove 6 diversi problemi di classificazione, di cui 3 con soli attributi nominali, 2 con soli attributi numerici e uno con attributi numerici e nominali. Segue una descrizione dei vari training set utilizzati:

- iris: contiene 150 istanze con 4 attributi numerici. Ogni istanza rappresenta una pianta della famiglia degli iris. La classe indica a quale specie in particolare appartiene la pianta. Non presenta valori mancanti.

- diabetes: contiene 768 istanze con 8 attributi numerici. Ogni istanza rappresenta i risultati di diverse analisi su un paziente. La classe indica se il paziente soffra o meno di diabete. Non presenta attributi mancanti.

- zoo: contiene 101 istanze con 17 attributi, di cui 16 nominali e 1 numerico. Ogni istanza rappresenta un animale con le sue caratteristiche. La classe rappresenta la categoria a cui appartiene l'animale. Non presenta valori mancanti.

- agaricus-lepiota: contiene 8124 istanze con 22 attributi nominali. Ogni istanza rappresenta un fungo con le sue caratteristiche. La classe indica se il fungo sia o meno commestibile. Contiene 2480 valori mancanti, tutti per lo stesso attributo.

- breast-cancer: contiene 286 istanze con 9 attributi nominali. Ogni istanza rappresenta risultati delle analisi di un paziente affetto da cancro. La classe rappresenta se il cancro sia o meno recidivo. Contiene 9 valori mancanti tra gli attributi.

- tic-tac-toe: contiene 958 istanze con 9 attributi nominali. Ogni istanza rappresenta una situazione di end-game nel gioco del tris. La classe indica la vittoria del giocatore che ha iniziato. Non presenta valori mancanti.

4.2 ID3

L'algoritmo ID3[14] (Iterative Dichotomiser 3) è un algoritmo di induzione proposto da J. R. Quinlan e utilizzato come base per lo sviluppo di molti algoritmi prodotti in seguito, viene spesso usato a scopo didattico per la sua semplicità.

L'algoritmo ID3 sceglie per ogni nodo l'attributo con il più alto guadagno di informazione. Non produce direttamente l'albero finale ma, come suggerisce il nome, segue un processo iterativo. Dal training set originale viene selezionato un sottoinsieme di istanze che verrà utilizzato come training set per la produzione dell'albero. L'albero così prodotto viene poi testato sul resto del set originale. Se tutte le istanze vengono classificate correttamente l'albero viene considerato corretto e restituito come output. In caso contrario viene aggiunta al training set una selezione delle istanze classificate erroneamente e viene prodotto un nuovo albero. La procedura continua finché non

viene prodotto un albero corretto.

Non è abilitato all'utilizzo di attributi numerici e non è in grado di gestire valori mancanti. È soggetto a sovradattamento e sensibile al rumore poiché non prevede un procedimento di pruning. Inoltre, poiché utilizza il guadagno di informazione per valutare la qualità degli attributi, tende a favorire attributi con molti possibili valori.

4.2.1 Prestazioni

Le prestazioni dell'algoritmo ID3 sono state valutate solo sui training set contenenti attributi di tipo nominale. Inoltre, poiché l'algoritmo non è in grado di gestire valori mancanti, i training set che ne contengono sono stati modificati in modo da trattare ogni valore mancante come un nuovo valore. Il training set zoo testato con questo algoritmo è stato modificato, trasformando l'unico attributo numerico in un attributo nominale.

Training set	Estrazione	5-fold	10-fold
agaricus-lepiota[3]	100	100	100
breast-cancer[3]	46.5	56	57.7
tic-tac-toe	81.5	86.5	83.2
zoo	0	0	0

È importante notare come l'algoritmo non riesca a classificare correttamente nuove istanze con il training set zoo. Ciò è causato dal fatto che zoo contiene un attributo (il nome dell'animale) univoco per ogni istanza. Poiché l'algoritmo ID3 utilizza il guadagno di informazione per la scelta del miglior attributo, su questo training set sceglie sempre l'identificatore come attributo migliore e produce un albero con il solo nodo radice. Rimuovendo dal training set l'identificatore si ottengono questi risultati:

Training set	Estrazione	5-fold	10-fold
zoo	90	97	97

La media dei valori ottenuti può fornire una stima della precisione generale dell'algoritmo. La media, calcolata prendendo come risultato per il training set zoo quello senza l'identificatore, risulta 82.95 .
Il tempo medio per l'esecuzione con il training set agaricus-lepiota è di 0.11 secondi.

[3]Il training set è stato modificato per permetterne l'utilizzo da parte dell'algoritmo nonostante siano presenti attributi con valori mancanti

4.3 C4.5

L'algoritmo C4.5 è anch'esso stato proposto da J. R. Quinlan come evoluzione dell'algoritmo ID3.

Il C4.5 utilizza il rapporto di guadagno come indice di qualità degli attributi, eliminando la preferenza per gli attributi con molti valori.

Il C4.5 è in grado di gestire attributi numerici effettuando una suddivisione binaria. Può essere usato su training set con valori mancanti e attributi con costi diversi. Utilizza inoltre l'error based pruning come metodo di post-pruning.

4.3.1 Prestazioni

L'algoritmo C4.5 permette di gestire attributi numerici e valori mancanti, perciò non è necessario modificare i training set. L'algoritmo testato in questa sezione è più precisamente il J48, un'implementazione open source Java del C4.5.

Training set	Estrazione	5-fold	10-fold
agaricus-lepiota	100	100	100
breast-cancer	70.2	74.1	75.5
tic-tac-toe	80.4	84.4	85.1
iris	91.7	96	96
diabetes	73.6	71.2	73.8
zoo	95	92.1	92.1

La media dei valori è 86.18 .
Il tempo medio per l'esecuzione con il training set agaricus-lepiota è di 0.07 secondi.

4.4 CART

L'algoritmo CART, sviluppato e proposto in Classification and Regression Trees [2], produce alberi binari, ovvero ogni nodo interno ha esattamente due rami uscenti.
L'albero prodotto viene poi potato utilizzando il cost-complexity pruning.

Può gestire attributi numerici e valori mancanti ed è in grado di eseguire regressioni oltre a classificazioni.

4.4.1 Prestazioni

Training set	Estrazione	5-fold	10-fold
agaricus-lepiota	100	99.9	99.9
breast-cancer	68.4	68.5	69.2
tic-tac-toe	91.6	94.4	93.2
iris	91.7	95.3	95.3
diabetes	74.3	73.6	75.1
zoo	40	40.6	40.6

La media dei valori risulta 78.42 .

Anche in questo caso, come per l'algoritmo ID3, nel training set zoo le prestazioni sono molto basse perché viene scelto come attributo migliore l'identificatore. In questo caso però l'algoritmo esegue un post-pruning ed elimina anche il nodo radice, riducendo l'albero ad una sola foglia contenente la classe più presente nell'intero training set.
Eliminando l'identificatore si possono ottenere i seguenti risultati:

Training set	Estrazione	5-fold	10-fold
zoo	95	88.1	88.1

Considerando questi ultimi valori invece di quelli nella tabella precedente, la media diventa 86.7 .
Il tempo medio per l'esecuzione con il training set agaricus-lepiota è di 6.91 secondi.

4.5 AdaBoost

AdaBoost (Adaptive Boosting)[9] è l'algoritmo simbolo della tecnica di boosting. In AdaBoost viene associato un peso ad ogni istanza del training set. Inizialmente tutte le istanze hanno peso 1.

Viene prodotto un classificatore con un algoritmo di weak learning e viene testato su tutto il training set. Si incrementa il peso di ogni istanza classificata erroneamente e si riduce il peso di ogni istanza classificata correttamente.

Si ripete questo procedimento fino a che non sia stato raggiunto un numero di classificatori prestabilito.

Per classificare nuove istanze si esegue una votazione a maggioranza pesata, in cui il peso di ciascun classificatore è proporzionale alla sua precisione nella classificazione del training set.

L'algoritmo tende dunque a favorire ad ogni iterazione le istanze che vengono classificate erroneamente. Ciò porta però, in caso di training set rumoroso, a favorire proprio le istanze contenenti rumore.

Esistono diverse versioni dell'algoritmo AdaBoost e può essere utilizzato in combinazione con diversi algoritmi di weak learning.
La capacità dell'algoritmo di gestire attributi numerici e valori mancanti dipende dalle capacità del weak learner utilizzato.

4.5.1 Prestazioni

L'algoritmo utilizzato nel test utilizza come classificatore di base il ceppo decisionale ed esegue 10 iterazioni.

Training set	Estrazione	5-fold	10-fold
agaricus-lepiota	95.6	96.8	96.2
breast-cancer	73.7	72.4	70.3
tic-tac-toe	71.3	73.4	72.5
iris	95	95.3	95.3
diabetes	77.9	74.6	74.4
zoo	55	60.4	60.4

La media dei valori è 78.36 .
Le prestazioni su zoo sono pari a quelle del ceppo decisionale perché l'algoritmo non è stato in grado di applicare il boosting al suddetto training set.
Il tempo medio per l'esecuzione con il training set agaricus-lepiota è di 0.28 secondi.

4.6 Bagging

L'algoritmo di Bagging, è l'implementazione della tecnica omonima proposta da Breiman[3].

La capacità dell'algoritmo di gestire attributi numerici e valori mancanti dipende dalle capacità del weak learner utilizzato.

4.6.1 Prestazioni

In questa implementazione viene usato come algoritmo di base il ceppo decisionale e vengono eseguite 10 iterazioni.

Training set	Estrazione	5-fold	10-fold
agaricus-lepiota	89	88.7	88.7
breast-cancer	75.4	74.1	73.8
tic-tac-toe	71.8	69.9	69.9
iris	63.3	83.3	72
diabetes	76.2	73	71.9
zoo	55	60.4	61.4

La media dei valori è 73.21 .
Le prestazioni in zoo sono basse perché nonostante l'algoritmo utilizzi il campionamento con ripetizione, quasi tutti gli alberi di base scelgono lo stesso attributo per la suddivisione iniziale, ed essendo ceppi decisionali, risultano equivalenti. Ciò implica che la loro combinazione non apporta alcun vantaggio alla precisione generale.
Il tempo medio per l'esecuzione con il training set agaricus-lepiota è di 0.32 secondi.

4.7 Random Forest

L'algoritmo Random Forest rappresenta l'implementazione del metodo Random Forest proposto da Breiman[4].

L'algoritmo è in grado di gestire attributi numerici e valori mancanti.

4.7.1 Prestazioni

Le prove sono state fatte con due modalità: nella prima l'algoritmo sceglie gli attributi per la suddivisione in modo casuale, nella seconda sceglie il migliore in un sottoinsieme casuale di 5 attributi.
In entrambi i casi sono stati prodotti 100 alberi.

La tabella seguente mostra i valori relativi alla selezione casuale dell'attributo.

Training set	Estrazione	5-fold	10-fold
agaricus-lepiota	100	100	100
breast-cancer	71.9	72.4	72.4
tic-tac-toe	89.3	90.6	90.8
iris	95	94.7	94
diabetes	76.2	75.8	75.7
zoo	95	94.1	97

La media dei valori è 88.16 .
Il tempo medio per l'esecuzione con il training set agaricus-lepiota è di 1.2 secondi.

Quelli che seguono sono i valori relativi alla selezione in un sottoinsieme casuale di 5 attributi.

Training set	Estrazione	5-fold	10-fold
agaricus-lepiota	100	100	100
breast-cancer	68.4	65.4	68.2
tic-tac-toe	94.3	95.3	96.9
iris	95	96	95.3
diabetes	76.5	74.7	74.4
zoo	90	93.1	92.1

La media dei valori è di 87.87 .
Il tempo medio per l'esecuzione con il training set agaricus-lepiota è di 1.29 secondi.

4.8 Ceppo Decisionale

Per eseguire un confronto tra i precedenti metodi e un weak learner sono state eseguite le stesse prove anche con un ceppo decisionale, ovvero un albero che contiene il solo nodo radice. Solitamente il ceppo decisionale sceglie l'attributo che minimizza l'errore di classificazione.

Il ceppo decisionale è stato inoltre usato come base per gli algoritmi AdaBoost e Bagging. In questo modo è possibile verificare empiricamente il teorema della giuria di Condorcet.

L'algoritmo è in grado di gestire attributi numerici e valori mancanti.

4.8.1 Prestazioni

Training set	Estrazione	5-fold	10-fold
agaricus-lepiota	89	88.7	88.7
breast-cancer	68.4	69.9	68.5
tic-tac-toe	71.8	69.9	69.9
iris	63.3	66.7	66.7
diabetes	73.9	72.3	71.9
zoo	55	60.4	60.4

La media dei valori è di 70.86.
Il tempo medio per l'esecuzione con il training set agaricus-lepiota è di 0.2 secondi.

4.9 Confronto

Nella seguente tabella sono messi a confronto i risultati medi di ogni algoritmo su ogni training set. Sono inoltre evidenziati i migliori risultati per ogni set.

Training Set	ID3	C4.5	CART	ABoost	Bagging	RF(1)	RF(5)	CD
agaricus-lepiota	100	100	99.93	96.2	88.8	100	100	88.8
breast-cancer	53.4	73.27	68.7	72.13	74.43	72.23	67.33	68.93
tic-tac-toe	83.73	83.3	93.07	72.4	70.53	90.23	95.5	70.53
zoo	94.67[4]	93.07	90.4[4]	58.6	58.93	96	93.73	58.6
iris		94.57	94.1	95.2	72.87	94.57	95.43	65.57
diabetes		72.87	74.33	75.63	73.7	75.9	75.2	72.7

Sebbene l'algoritmo Random Forest sembri avere prestazioni nettamente superiori agli altri è bene notare che sono stati utilizzati due metodi diversi per lo stesso algoritmo, ognuno dei quali è più adatto a determinati training set. Ad esempio, eliminando il metodo Random Forest casuale (RF(1)) otteniamo

[4]I risultati sono stati ottenuti rimuovendo dal training set l'identificatore dell'istanza

Training Set	ID3	C4.5	CART	ABoost	Bagging	RF(5)	CD
agaricus-lepiota	100	100	99.93	96.2	88.8	100	88.8
breast-cancer	53.4	73.27	68.7	72.13	74.43	67.33	68.93
tic-tac-toe	83.73	83.3	93.07	72.4	70.53	95.5	70.53
zoo	94.67[4]	93.07	90.4[4]	58.6	58.93	93.73	58.6
iris		94.57	94.1	95.2	72.87	95.43	65.57
diabetes		72.87	74.33	75.63	73.7	75.2	72.7

Mentre rimuovendo il metodo Random Forest con l'attributo scelto tra 5 casuali (RF(5)) otteniamo

Training Set	ID3	C4.5	CART	ABoost	Bagging	RF(1)	CD
agaricus-lepiota	100	100	99.93	96.2	88.8	100	88.8
breast-cancer	53.4	73.27	68.7	72.13	74.43	72.23	68.93
tic-tac-toe	83.73	83.3	93.07	72.4	70.53	90.23	70.53
zoo	94.67[4]	93.07	90.4[4]	58.6	58.93	96	58.6
iris		94.57	94.1	95.2	72.87	94.57	65.57
diabetes		72.87	74.33	75.63	73.7	75.9	72.7

Si può dedurre da queste tabelle che non ci sia un algoritmo palesemente superiore agli altri.

Un confronto dei tempi di esecuzione dei diversi algoritmi fornisce questi risultati:

Algoritmo	Tempo(s)
CD	0.02
C4.5	0.07
ID3	0.11
AdaBoost	0.28
Bagging	0.32
RF(1)	1.20
RF(5)	1.28
CART	6.91

Mettendo da parte l'algoritmo CART, che impiega un tempo più di 5 volte superiore a quello impiegato dal Random Forest (nonostante quest'ultimo produca 100 alberi), i rapporti tra i valori sono in linea con quelli attesi. Per

[4]I risultati sono stati ottenuti rimuovendo dal training set l'identificatore dell'istanza

quanto riguarda i metodi di ensemble learning possiamo vedere come AdaBoost e Bagging impieghino, giustamente, un tempo leggermente superiore a quello necessario per produrre i dieci ceppi decisionali che li compongono, mentre i due metodi Random Forest, che producono 100 alberi casuali, mostrano come ciascuno degli alberi possa essere prodotto più velocemente di un ceppo decisionale, semplificando il processo di selezione dell'attributo.

Confrontando invece le prestazioni medie di ogni algoritmo otteniamo

Algoritmo	Prestazioni
RF(1)	88.16
RF(5)	87.87
CART	86.76*
C4.5	86.18
ID3	82.95*
AdaBoost	78.36
Bagging	73.21
CD	70.86

Si può concludere che nonostante il metodo Random Forest non sia il migliore in ogni training set, rimane comunque un metodo valido e in particolare permette di ottenere prestazioni consistenti in termine di precisione, senza richiedere la taratura di parametri o la modifica del training set.
Il C4.5 risulta invece l'algoritmo con il miglior compromesso tra precisione e tempo di produzione.
Si può verificare anche che l'algoritmo a Ceppo Decisionale risulta, come previsto, il classificatore con l'errore più elevato.
È inoltre degno di nota il fatto che i punteggi dei metodi AdaBoost e Bagging siano risultati particolarmente bassi a causa della loro incapacità di sfruttare i benefici apportati dall'ensemble learning nel training set zoo.
Ricordiamo infine che i valori contrassegnati dall'asterisco sono stati ottenuti modificando il training set zoo. Senza le modifiche i valori sarebbero stati 78.42 per CART e 66.36 per ID3. Ciò evidenzia ancora una volta come alcuni algoritmi siano efficaci solo sotto determinate circostanze.

Capitolo 5

Conclusioni

In un caso di utilizzo reale dell'apprendimento di alberi di decisione, è possibile che il contesto richieda una certa precisione, ma anche una certa velocità. Per questo motivo spesso è bene valutare le prestazioni di un algoritmo osservando queste due dimensioni.
Dai risultati esposti al termine del capitolo precedente possiamo concludere che non ci sia un algoritmo, tra quelli analizzati, che sia nettamente superiore agli altri. Ad ogni problema di apprendimento deve essere associato l'algoritmo più adatto.
Queste osservazioni non valgono soltanto per gli algoritmi finiti ma anche per ogni fase della produzione di un albero di decisione.

- Ogni metodo per il calcolo della qualità degli attributi ha pregi e difetti (sebbene sia risultato evidente dalle prove effettuate, che i metodi che prevedono la normalizzazione, permettono di affrontare training set differenti senza la necessità di modificare o rimuovere attributi).

- Ogni tecnica di pruning è più adatta ad un algoritmo che produce alberi più o meno sovradattati (e ad un training set che porti o meno l'algoritmo al sovradattamento).

- I metodi di ensemble learning devono essere scelti valutando il problema di apprendimento e il tipo di classificatore di base disponibile.

In definitiva, non esiste l'algoritmo perfetto, ma c'è un algoritmo adatto ad ogni problema.

Bibliografia

[1] H. Blockeel, J. Struyf - Efficient Algorithms for Decision Tree Cross-validation - Journal of Machine Learning Research (2002)

[2] L. Breiman, J. H. Friedman, R. A. Olshen, C. J. Stone - Classification and Regression Trees - Wadsforth International Group (1984)

[3] L. Breiman - Bagging Predictors - Machine Learning (1996)

[4] L. Breiman - Random Forests - Machine Learning (2001)

[5] N. de Caritat - Essai sur l'application de l'analyse á la probabilité des décisions rendues á la pluralité des voix - Imprimerie Royale (1785)

[6] P. K. Chan, S. J. Stolfo - Experiments on Multistrategy Learning by Meta-Learning - In Proceedings of the Second International Conference on Information and Knowledge Management (1993)

[7] P. K. Chan, S. J. Stolfo - Toward Parallel and Distributed Learning by Meta-Learning - In AAAI Workshop in Knowledge Discovery in Databases (1993)

[8] U. M. Fayyad, K. B. Irani - On the Handling of Continuous-Valued Attributes in Decision Tree Generation - Machine Learning (1992)

[9] Y. Freund, R. E. Schapire - A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting - Journal of Computer and System Sciences(1997)

[10] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, I. H. Witten - The WEKA Data Mining Software: An Update - SIGKDD Explorations, Volume 11, Issue 1 (2009)

[11] M. Lichman - UCI Machine Learning Repository [http://archive.ics.uci.edu/ml] - Irvine, CA: University of California, School of Information and Computer Science (2013)

[12] T. M. Mitchell - Machine Learning - McGraw-Hill (1997)

[13] S. K. Murthy - Automatic Construction of Decision Trees from Data: A Multi-Disciplinary Survey - Data Mining and Knowledge Discovery (1998)

[14] J. R. Quinlan - Induction of Decision Trees - Machine Learning (1986)

[15] J. R. Quinlan - Simplifying Decision Trees - International Journal of Man-Machine Studies (1987)

[16] L. Rokach, O. Maimon - Data Mining with Decision Trees: Theory and Applications - Second Edition - World Scientific Publishing (2015)

[17] S. J. Russell, P. Norvig - Artificial Intelligence: A Modern Approach - Second Edition - Prentice Hall (2003)

[18] A. K. Seewald, J. Fürnkranz - An Evaluation of Grading Classifiers - Proceedings of the 4th International Conference on Advances in Intelligent Data Analysis - Springer-Verlag (2001)

[19] C. E. Shannon – A Matematical Theory of Communication (1948)