**Decision Tree**

Ogni **istanza** è descritta da **coppie attributo- valore**, dove ognuna prende il nome di **letterale**.

I **nodi interni** sono etichettati con una delle **feature** del modello.

Ogni ramo che parte da un nodo interno è etichettato con un letterale. L'insieme dei letterali in corrispondenza di un nodo è chiamato **split**.

Ogni foglia è un’espressione logica, una congiunzione di letterali lungo il percorso radice- foglia. Segmento dello spazio delle istanze: Insieme delle istanze coperte da un percorso.

Ogni espressione logica può essere scritta in DNF.

Se il nostro linguaggio è abbastanza espressivo da rappresentare qualsiasi insieme di esempi positivi, dobbiamo assicurarci che l'algoritmo di apprendimento impieghi altri meccanismi per forzare la generalizzazione al di là degli esempi ed evitare l'overfitting - questo è il **bias induttivo** dell'algoritmo di apprendimento.

La maggior parte degli algoritmi di apprendimento che operano in spazi di ipotesi espressivi hanno un bias induttivo verso ipotesi meno complesse, sia implicitamente attraverso il modo in cui lo spazio di ipotesi viene cercato, sia esplicitamente incorporando una penalità di complessità nella funzione obiettivo.

**Tree Induction**: È una procedura ricorsiva, divide and conquer. Usa tre funzioni:

* **Homogeneous**(D), **true se le istanze sono omogenee** e possono essere etichettate con un’unica etichetta
* **Label**(D), **return l’etichetta** più appropriata
* **Best Split(**D,F), **return il migliore insieme di letterali** come **root** tree.

Procedura che divide i dati in sottoinsiemi e costruisce un albero per ognuno di essi, combinando i sottoalberi in uno unico. Funziona purché vi sia sempre un modo per fermare la ricorsione. Sono algoritmi greedy: Per ogni scelta si seleziona l’alternativa migliore partendo dalle info disponibili e questa non viene mai riconsiderata. Siamo interessati a trovare il più piccolo tra questi, che è np completo. Le euristiche che si usano sono: Entropia o Gini Index.

**Best Split**: Vuole trovare la migliore feature da mettere come root tree, (nel caso di 2 classi, si preferisce quella che separa il più possibile gli esempi di training).

Per attributi discreti, viene scelto il tipo di test, l’attributo, e la partizione. Per quelli continui, viene scelta una soglia.

Quello che si vuole misurare è la **purezza del nodo**. Una **divisione è pura** quando per tutti i rami, tutte le **istanze** che seguono quel ramo **appartengono alla stessa classe**.

**Empirical probability della classe +**: Tasso dei positivi all’interno del dataset.

**Minority class**: È il tasso di errore. Misura la proporzione di esempi classificati male se alla foglia è stata assegnata la classe di maggioranza.

**Entropia**: L’informazione attesa. Il #bit che devono essere usati per comunicare la classe di un esempio estratto a caso. Maggiore è la purezza del dataset, maggiore è la prevedibilità della classe, e quindi meno bit sono necessari.

**Gini Index**: Errore atteso se etichettiamo gli esempi nella foglia in modo casuale.

Per ogni attributo, di ogni tipo se numerico, calcolata l’impurità, si sceglie quello con valore minimo.

**Entropia**

È una **misura dell'incertezza** associata a una variabile casuale discreta.

, dove , record classificati con cj sul totale dei record. Entropia è un valore compreso tra [0,1]. H=1, quando il dataset è bilanciato, stessa probabilità per la classe + che per la classe –. H=0, quando non vi è impurità, tutti gli esempi appartengono ad un’unica classe, e quindi inutile per l’apprendimento.

**Information Gain**

Il guadagno informativo di un test è la **riduzione attesa dell'entropia causata dalla suddivisione degli esempi in base all'attributo utilizzato** nel test (cioè, conoscendo il valore dell'attributo). È equivalente al numero di bit necessari per codificare la classe di esempi che si risparmia quando si conosce anche il risultato del test X sugli esempi.

Hx(D) è l’entropia dopo il partizionamento di D per effetto del test basato su x. La somma di ciascuna entropia di ciascun sottoinsieme Di, pesata per la frazione di esempi che scendono per ciascun branch.

,

Viene scelto l’attributo con maggior information gain.

La **selezione di un nuovo attributo** e la **partizione degli esempi** viene ripetuta per ogni nodo discendente non terminale, usando solo gli esempi associati a quel nodo. Gli attributi che sono stati incorporati più in alto nell'albero vengono esclusi, in modo che un dato attributo possa comparire al massimo una volta lungo qualsiasi percorso dell'albero. Si ferma quando:

* Ogni attributo è già stato incluso lungo il percorso dell'albero
* Gli esempi di addestramento associati al nodo foglia corrente hanno tutti lo stesso valore dell'attributo target (cioè, la loro entropia è 0)

**Se** venissero usati **attributi con molti valori diversi** (l’attributo chiave). Si avrebbe |D|=n e |Di|=1. L'entropia di ogni sottoinsieme è nulla, infatti una chiave identifica l'istanza senza alcuna ambiguità. Hx(D) è nullo, quindi Gain(D,X)=H(D), sarebbe massimo. Quindi o si rimuovono le colonne che contengono informazioni non utili o si penalizzano in modo che non vengano scelte.

, da cui si calcola

**Gini Index**

Misura l**’impurità per k classi**.

L'impurità Gini è compresa tra 0 e 1, dove 0 rappresenta un dataset puro e 1 rappresenta quello completamente impuro.

**Test su attributi continui**

Quando si ha un certo attributo X che ammette *m* valori possibili e si considera

if (x>c): True; else: False, dove c è una soglia che produce il maggior information gain.

1. Si ordinano gli esempi secondo l’attributo x
2. Si identificano gli esempi adiacenti che differiscono nella classificazione
3. Si generano le soglie candidate a metà tra i valori
4. Queste soglie candidate possono essere valutate calcolando il guadagno di informazione associato a ciascuna di esse

**Esempio di Applicazione di Grow Tree (D,F) sul dataset Play tennis**:

* Partendo dal dataset, viene scelto “outlook” come root. Vengono generati 3 branch, per i 3 possibili valori che può ammettere outlook: Sunny, Overcast, Rain.
* Per Overcast, tutti gli esempi appartengono alla classe P. Viene generata una foglia.
* Ricorsione sugli altri due valori.
* Grow Tree (Sunny): Il test che viene scelto è humidity<75. 2 possibili branch (T, F). Caso true, entrambi appartengono a P (foglia), Caso false, tutti a N (foglia)
* Grow Tree (Rain): Il test è windy=?, 2 branch (T, F). Caso true, esempi tutti N, Caso false, esempi tutti P.

**Attributi con valori non noti**

Consideriamo F: insieme degli esempi con x noto; P(xi, cj): Probabilità di ciascuno dei possibili valori di x. Si assume che i valori non noti abbiano la stessa distribuzione rispetto all’attributo e alla classe. Il valore sconosciuto viene considerato come un valore a sé stante. C4.5 Penalizza gli attributi con valori non noti:

**Partizionamento**:

Ad ogni esempio viene assegnato un peso, inizialmente 1.

* Se quell’esempio ha valore noto, Allora viene inserito in Di con peso 1 e negli altri Dj con peso nullo.
* Se quell’esempio non ha valore, Viene messo in Di con peso wP(xi), probabilità di xi in F.

Nelle foglie si usa la nomenclatura A/B:

* A - Peso totale degli esempi associati alla foglia
* B - Peso totale degli esempi associati alla foglia, classificati male.

Esempio: N(3.4/0.4): 3.4 appartengono alla foglia, di cui 0.4 non appartengono alla classe N.

Un nuovo esempio, Percorre l’albero, quando si arriva alla foglia A/B, verrà classificato come appartenente ad A con probabilità = (a-b)/a e appartenente a B con P = b/a.

**C4.5**

* Ricerca di tipo Hill Climbing, di tutti i possibili alberi partendo dall’ipotesi più semplice e usando come misura l’information gain.
* Lo spazio dei possibili alberi è il power set di x. Include sicuramente il concetto target.
* Mantiene una sola ipotesi durante la ricerca. Non fa backtracking, e potrebbe ottenere una soluzione ottimale localmente.
* Usa tutti gli esempi di training. È meno sensibile agli errori nei singoli esempi.
* Cerca in uno spazio completo (in grado di esprimere qualsiasi funzione) e cerca in modo incompleto finché non è soddisfatta la terminazione.

**Bias induttivo**:

* Gli alberi più corti sono preferiti.
* Gli alberi che mettono più vicino alla root gli attributi con maggior information gain sono preferiti.

Overfitting:

c4.5 fa crescere ogni ramo dell'albero in profondità quanto basta per classificare perfettamente gli esempi di addestramento. **Un'ipotesi si adatta eccessivamente ai dati** quando esiste un'ipotesi più semplice, meno accurata sul set di addestramento ma più accurata su istanze diverse da quelle del set di addestramento. **L'overfitting** può verificarsi quando l'insieme di addestramento contiene errori (rumore) o quando è piccolo.

Metodi per evitare overfitting:

* Pre pruning: Interrompere la crescita prima di arrivare a un albero che classifica perfettamente gli esempi
* Post Pruning: Far crescere l’albero e potarlo dopo. 2 tecniche:
  + **Reduce Error Pruning**:
    - Potare un nodo = sostituirlo con una foglia con l’etichetta più frequente. Nodi rimossi solo se l’albero potato non è peggiore dell’originale. E si eliminano nodi che portano a aumentare l’accuratezza.
  + **Rule post Pruning**:
    - Si converte l’albero in un insieme di regole, con una regola per ogni percorso root-foglia. Si pota ogni regola rimuovendo precondizioni che migliorano l’accuratezza. E si ordinano le regole secondo l’accuratezza e si considerano in sequenza.

Come determinare la **DIM ottimale** dell’albero:

* Con un dataset separato per valutare post-pruning
* Con tutti gli esempi disponibili, ma con un test statistico per decidere se mantenere o no un nodo
* Con una misura della complessità di codifica dell’albero e degli esempi.

**Alberi di Regressione**

Quando la variabile target è continua. In questo caso ogni foglia è etichettata con un valore numerico. Ognuna definisce una regione nello spazio di input, dove le istanze che rientrano nella regione hanno risultati simili.

* Label (Y): return il valore medio
* Homogeneous (Y): return true se la varianza dei valori è nulla o minore di una soglia

**Obiettivo**: minimizzare la varianza su tutti i possibili split di un dato genitore. E si sceglie quello con varianza minima. Minimizzare questo, equivale a massimizzare la media ponderata delle medie quadratiche nei figli.