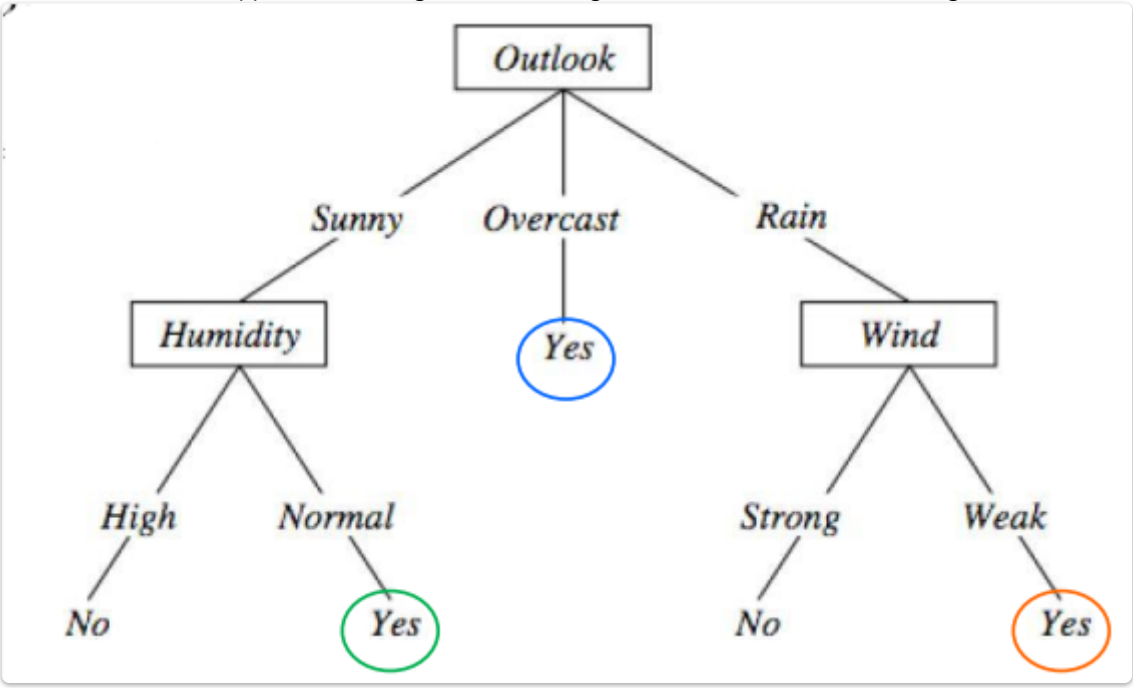
Potere espressivo

L'albero di decision rappresenta un disgiunzione di congiunzioni di costrutti sul valore degli attributi.



Lo spazio delle ipotesi H è capace di esprimere qualunque funzione finita a valori discreti

Top-Down

ID3 è un algoritmo base per l'apprendimento dei Decision Tree. Dato un insieme di esempi di training, l'algoritmo esegue la *ricerca nello spazio* del DT. La costruzione dell'albero è top-down, l'algoritmo esegue una ricerca greedy:

1. Seleziona il miglior attributo
2. Viene poi creato un nodo discendente per ogni possibile valore dell'attributo e gli esempi sono partizionati in base a quel valore
3. Il processo si ripete per ogni nodo successore finché tutti gli esempi sono classificati bene o finché non rimangono più attributi

```
ID3 (X, T, Attrs)
    create Root node
    IF tutti gli X sono + RETURN Root con classe +
    IF tutti gli X sono - RETURN Root con classe -
    IF Attrs è vuoto RETURN Root con la classe più comune di T in X
    ELSE
        A <-- miglio attributo, attributo di decisione per Root <-- A
        FOR EACH possibile valore v_i di A
            aggiungi un nuovo ramo sotto Root, per testare a = v_i
            X_i <-- sottoinsieme di X con A = k
            IF X_i è vuoto THEN aggiungi una nuova fogli acon la classe più comune di T in X
            ELSE aggiungi il sottoalbero generato da ID3(X_i, T, Attrs - {A})
    RETURN Root
```

dove `X` sono gli esempi di training, `T` il target attribute e `Attrs` gli altri attributi (inizialmente tutti).

Selezione del miglio attributo

ENTROPIA

Usiamo la nozione di entropia, che misura l'impurità di un insieme di esempi e dipende dalla distribuzione della variabile random p.

Entropia

$$Entropy(S) = p_+ log_2 p_+ - p_- log_2 p_-$$
$$\rightarrow Entropy([14+, 0-]) = -14/14 log_2(14/14) - 0 log_2(0) = 0$$

S è l'insieme di esempi di training, p_+ e p_- sono la porzione di esempi positivi e negativi in S.
Il guadagno di informazioni è la riduzione attesa di entropia causata dalla partizione degli esempi su un attributo. Ci sono $Values(A)$ valori possibili per A.

$$Gain(S, A) = Entropy(S) - \sum_{v \in Values(S)} \frac{|S_V|}{|S|} Entropy(S_V)$$

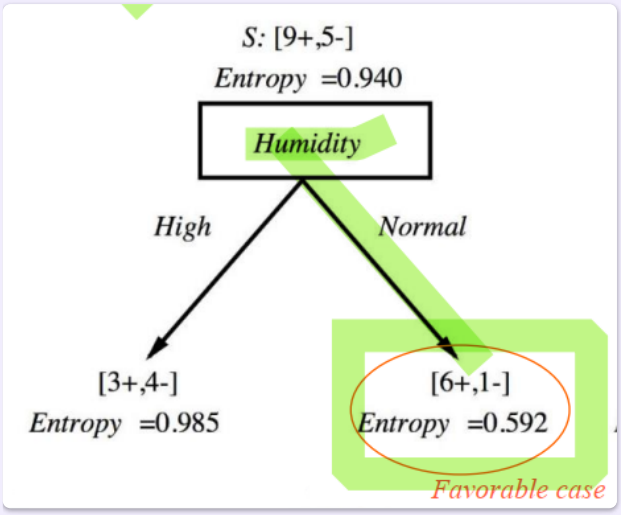
Information Gain

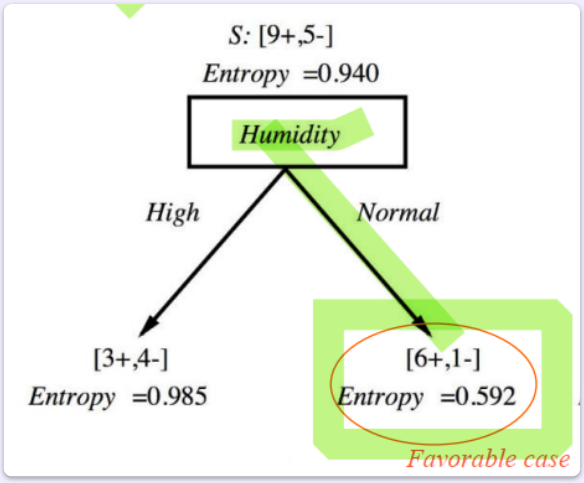
S_V è un sottoinsieme di esempi di S per i quali A ha valore v. Maggiore è il guadagno, maggiore è l'efficienza dell'attri buto nella classificazione dei dati di training

PERCHÉ SI USA

L'entropia misura l'omogeneità (anzi l'impurità) della classe del sottoinsieme di esempi, quindi si *seleziona un* A che *massimizzi* $Gain(S, A)$. Dopo la suddivisione ci sono valuri più bassi (target più omogenei) in ogni sottoinsieme (**maggior guadagno**).
Lo scopo è di *separare gli esempi* sulla base del target, trovare gli attributi che discriminano gli esempi che appartengono a diversi target (ad es. tutti i positivi a sinistra, tutti i negativi a destra).

Example





$$Gain(S, Humidity) = $$
$$= 9.40 - (\tfrac{7}{14}).985 - (\tfrac{7}{14}).592 = \boxed{.151}$$

$$Gain(S, Wind) = $$
$$.940 - (\tfrac{8}{14}).811 - (\tfrac{6}{14})1.0 = 0.48$$

PROBLEMI

Il guadagno di informazioni fornisce gli attributi con molti valori possibili. Si consideri *ad esempio* gli attributi $Date$ nell'esempio $PlayTennis$:
$\rightarrow$ $Date$ ha il massimo guadagno di informazioni: ogni giorno corrisponde a un diverso sottoinsieme puro: $[1+, 0-]$ o $[0+, 1-]$ $\implies$ 0 entropy
$\rightarrow$ Fit (separazione) perfetta dei dati di training
Non è significativo: *inutile per i dati non visti*

Misura alternativa: Gain Ratio

Gain ratio

$$GainRatio(S,A) = \frac{Gain(S,A)}{SplitInformation(S,A)}$$

$$SplitInformation(S,A) = -\sum_{i=1}^{c} \frac{|S_i|}{|S|} log_2 \frac{|S_i|}{|S|}$$

S_i sono gli insiemi ottenuti dalla partizione sui valori v_i di A fino a c valori

$SplitInformation$ musira l'**entropia** di S rispetto ai *valori di* A. Più la dispersione dei dati è uniforme, maggiore è il suo valore,.
$GainRatio$ penalizza gli attributi che separano gli esempi in molte classi minori come $Date$

Regolarizzazione

Problema $SplitInformaition(S,A)$ può essere vuoto ($\emptyset$) o molto piccolo quadno $|S_i| \approx |S|$ per qualche valore i $[log1 = 0]$.
Per mitigare questo effetto viene usata la seguente **euristica**:

1. Calcolare $GainRatio$ per ogni attributo
2. Applicare $GainRatio$ solo per gli attributi con $Gain$ sopra la media

Ricerca nello spazio delle ipotesi in DT Learning

La ricerca in *HP Space*, ricerca (con *Hill-climbing*) nello spazio dei possibili DT dal più semplice fino al più complesso, riferito all'algoritmo precedente (*Cand.Elimin*):

- Lo spazio delle ipotesi è completo (rappresenta tutte le funzioni o valori discreti)
- La ricerca mantiene una singola ipotesi alla volta
- Non torna indietro e non garantisce l'ottimalità
- Usa tutti gli esempi disponibili
- Può terminare prima
- Accetta classi disturbate

Bias induttivo in DT learing

"Shortest trees are prefered over longer trees"

Grazie alla ricerca dal semplice al complesso, incrementale. Inoltre questo è il *bias* esibito da un semplice *breadh first* che genera tutto il DT e seleziona quello consistente più corto.

"Preferes trees that place high information gain attributes close to the root"

I DT sono limitati dal rappresentare tutte le informazioni possibili, la *restrizione* non è sull'ipotesu ma sulla *strategia di ricerca*.

2 tipi di Bias

Preferenza o search bias (dato dalla strategia di ricerca)	Restrizione o lenguage bases
ID3 cerca uno spazio delle ipotesi completo, la strategia di ricerca è incompleta	Si fa la ricerca del tipo candidate-elimination che cerca uno *spazio delle ipotesi incompleto*. La strategia è completa

Nell'apprendimento di un modello lineare si usa una *combinazione di bias*.

Il search bias è preferibilmente al lenguage bias

- Nel ML si usano tipicamente approcci flessibili (capability universale) dei modelli), senza escludere a priori la funzione target sconosciuta
- Flessibilità → tener cura dei propri problemi di overfitting

Ockham's Razor Prefer shorter hypoteses

```
- <span style="color:#ff82b2"><i>Controllo della complessità del modello</i></span> dalla regoralizzazione dei sistemi lineari
```

- Nel ML è importante razionalizzare il modello alla fine

Problemi nell'apprendimento dei DT

Overfitting:

- interrompere presto
- ridurre gli errori di pruning
- regole post-pruning

Extensions:

- misure alternative per selezionare gli attributi
- attributi continuamente valutati
- gestione degli esempi con valorre degli attributi mancanti
- gestione degli attributi con costi differenti
- miglirare l'efficienza di computazione

Overfitting

Definition

Costruire alberi che "si adattano troppo" agli esempi di training può portare a overfitting. Consideriamo l'errore dell'ipotesi h su i dati di training ($error_D(h)$) e sull'intera distribuzione X di dati ($error_X(h)$).

> Overfitting
>
> L'ipotesi h fa overfitting sui dati di training se esiste un'ipotesi $h' \in H$ t.c.
>
> $$error_D(h) < error_D(h')$$
> $$AND$$
> $$error_X(h) < error_X(h')$$
>
> Quindi h' si comporta peggio sui dati di training, ma meglio sui dati sconosciuti

Gli approcci flessibili possono facilmente andare in contro a overfitting se non usati con particolare cura

> *image pag.28 of 3.4*

> Example
>
> [image pag.29 of 3.4]
> Immaginiamo che questo esempio disturbato causi la divisione del secondo nodo foglia: il DT cresce come la sua complessità

Evitare l'overfit nei DT

Ci sono 2 strategie:

1. Fermare presto la crescita dell'albero, prima della classificazione perfetta
2. Permettere che l'albero faccia overfit sui dati, poi fare *post-prune*
   Valutazione degli effetti:
Training & validation set	Altri approcci
Divide il training set in due (training & validation) e usa il secondo per valutare i punti 1. e 2.	Si usa un test statisstico per stimare gli effetti di espansione / pruning, oppure *minimum description lenght principle*

Pruning a errore ridotto

- *Ogni nodo è candidato* per il pruning
- Il pruning consiste nel **rimuovere sottoalberi** radicati in un nodo: il nodo diventa una foglia ed è assegnato al classificatore più comune
- Il nodo viene *rimosso* solo *se l'albero risultante non performa peggio* sul validation set
- I nodi vengono *potati iterativamente*: a ogni iterazione il nodo la cui rimozione incrementa di più l'accuratezza sul validation set viene potato
- Il pruning si ferma quando nessun pruning incrementa l'accuratezza
  Effetti del pruning a errore ridotto:

> *image pag.32 of 3.4*

REGOLE POST-PRUNING

1. Creare il DT dal training set
2. Convertire il DT in un insieme di regole equivalenti
   - Ogni *path* corrisponde a una *regola*
   - Ogni *nodo* lungo il path corrisponde a una *pre-condizione*
   - Ogni *classificazione di foglia* corrisponde a una *post-condizione*

$$(Outlook = Sunny) \wedge (Humidity = High) \implies (PlayTennis = No)$$

3. Potare (generalizzare) ogni regola rimuovendo le precondizioni che non migliorano l'accuratezza
4. Ordinare le regole nell'*ordine dell'accuratezza stimata*, e considerarle *in sequenza* quando si classificano nuove istanze

PERCHÉ SI CONVERTONO IN REGOLE

Dove un attributo continuo A, crea dinamicamente un *nuovo attributo* A_c

$$A_c = true \text{ if } A < c, false \text{ otherwise}$$

Come si determina il *valore soglia* c? (**Esempio:** $Temperature$ nell'esempio $PlayTennis$)

- Si ordinano gli esempi secondo $Temperature$

- Si determinano le soglie dei candidati *facendo la media* dei valori consegutivi che sono in classifica di cambiamento

$$\frac{48+60}{2} = 54 \qquad \frac{80+90}{2} = 85$$

- Si valuta *la soglia dei candidati* (attributi) secondo il guadagno di informazioni *(* Miglior $temperature > 54$ *)*. Ora il nuovo attributo compete con gli altri

Gestire i dati di training incompleti (imputation)

1. (*più comune*) Assegnare il valore più in comune tra tutti i dati di training negli esempi al nodo o nella stessa classe
2. Assegnare una probabilità p_i a ogni valore v_i, basata sulla frequenza, e assegnare i valori agli attributi mancanti, secondo questa distribuzione di pobabilità (si aggiungono più esempi pesati con la probabilità) $\overset{quindi}{\rightarrow}$ Si assegna una *frazione* p_i dell'esempio ad ogni albero discendente
3. Classificare il nuovo esempio allo stesso modo (pesandolo): Viene scelta la *classificazione più probabile*

Gestire attributi con costi differenti

Gli attributi dell'istanza possono avere un *costo associato*: prefiamo i DT chge usano attributi a **basso costo**. ID3 può essere modificato per *tenere conto del costo*:

1. Tan & Schlimmer (1990): $\frac{Gain^2(S,A)}{Cost(A)}$
2. Numez (1988): $\frac{2^{Gain(S,A)-1}}{(Cost(A)+1)^w}$ $\text{con } w \in [0,1]$

> *image pag.40 of 3.4*

Conclusione

- I DT sono un approccio popolare per la *classificazione* in un discreto numero di classi
  - *Spazio delle ipotesi espressivo* nell'area delle preposizione (approccio *rule-based*)
  - Robusti sui dati disturbati
  - *facile da capire* (regole $if - then$ esplicite)
  - Molte estensioni allo schema base
- **Language** e **search bias**: ID3 crea uno spazio delle ipotesi completo, con strategia greedy incompleta
- **Approccio flessibile**: L'overfitting è un problema importante trovato con *early stopping*, *post-pruning* e *generalizzazione* delle regole indotte