Machine Learning

UniShare

Davide Cozzi
@dlcgold

Indice

Capitolo 1

Introduzione

Questi appunti sono presi a lezione. Per quanto sia stata fatta una revisione è altamente probabile (praticamente certo) che possano contenere errori, sia di stampa che di vero e proprio contenuto. Per eventuali proposte di correzione effettuare una pull request. Link: `https://github.com/dlcgold/Appunti`.
Si segnala che le immagini sono tratte dalle slide del corso.

Capitolo 2

Introduzione al ML

Il **Machine Learning (*ML*)** è sempre più diffuso nonostante sia nato diversi anni fa.
Un **sistema di apprendimento automatico** ricava da un *dataset* una conoscenza non fornita a priori, descrivendo dati non forniti in precedenza. Si estrapolano informazioni facendo assunzioni sulle informazioni sistema già conosciute, creando una **classe delle ipotesi H**. Si cercano ipotesi coerenti per guidare il sistema di apprendimento automatico. Bisogna però mettere in conto anche eventuali errori, cercando di capire se esiste davvero un'ipotesi coerente e, in caso di assenza, si cerca di approssimare. In quest'ottica bisogna mediare tra **fit** e **complessità**. Ogni sistema dovrà cercare di mediare tra questi due aspetti, un *fit* migliore comporta alta *complessità*. Si ha sempre il rischio di **overfitting**, cercando una precisione dei dati che magari non esiste. Si ha un **generatore di dati** ma il sistema non ha conoscenza della totalità degli stessi.
Definiamo alcuni concetti base:

- **task (*T*)**, il compito da apprendere. È più acile apprendere attraverso esempi che codificare conoscenza o definire alcuni compiti. Inoltre il comportamento della macchina in un ambiente può essere diverso da quello desiderato, a causa della mutabilità dell'ambiente ed è più semplice cambiare gli esempi che ridisegnare un sistema

- **performance (*P*)**, la misura della bontà dell'apprendimento (e bisognerà capire come misurare la cosa)

- **experience (*E*)**, l'esperienza sui cui basare l'apprendimento. Il tipo di esperienza scelto può variare molto il risultato e il successo dell'apprendimento

In merito alle parti "software" distinguiamo:

- **learner**, la parte di programma che impara dagli esempi in modo automatico
- **trainer**, il *dataset* che fornisce esperienza al *learner*

Durante l'**apprendimento** si estrapolano dati da **istanze di addestramento o test**. Quindi:

- si ricevono i dati di addestramento
- il sistema impara ad estrapolare partendo da quei dati
- si ricevono dati di test su cui si estrapola

L'ipotesi da apprendere viene chiamata **concetto target** (tra tutte le ipotesi possibili identifico quella giusta dai dati di addestramento).
Approfondiamo il discorso relativo all'*esperienza*. Innanzitutto nel momento della scelta bisogna valutare la rappresentatività esperienza. SI ha inoltre un controllo dell'esperienza da parte del *learner*:

- l'esperienza può essere fornita al learner senza che esso possa interagire
- il learner può porre domande su quegli esempi che non risultano chiari

L'esperienza deve essere presentata in modo causale.
Si hanno due tipi di esperienza:

1. **diretta**, dove i learner può acquisire informazione utile direttamente dagli esempi o dover inferire indirettamente da essi l'informazione necessaria (può essere chiaramente più complicato)
2. **indiretta**

Il tipo di dato che studieremo comunemente sarà il **vettore booleano** e la risposta sarà anch'essa di tipo booleano. In questo contesto l'ipotesi è una **congiunzione di variabili**.
Per ogni istanza di addestramento cerchiamo una risposta eventualmente corrispondente al nostro *target* (ovvero 1), qualora esista.
Si hanno tre tipi di apprendimento:

1. **apprendimento supervisionato**, dove vengono forniti a priori esempi di comportamento e si suppone che il *trainer* dia la risposta corretta per ogni input (mentre il learner usa gli esempi forniti per apprendere). L'esperienza è fornita da un insieme di coppie:

   $$S \equiv \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$$

   e, per ogni input ipotetico x_i l'ipotetico trainer restituisce il corretto y_i

2. **apprendimento non supervisionato**, dove si riconosce *schemi* nell'input senza indicazioni sui valori in uscita. Non c'è target e si ha *libertà di classificazione*. Si cerca una *regolarità* e una *struttura* insita nei dati. In questo caso si ha:

   $$S \equiv \{x_1, x_2, \ldots, x_n\}$$

   Il clustering è un tipico problema di apprendimento non supervisionato. Non si ha spesso un metodo oggettivo per stabilire le prestazioni che vengono quindi valutate da umani

3. **apprendimento per rinforzo**, dove bisogna apprendere, tramite il *learner* sulla base della risposta dell'ambiente alle proprie azioni. Si lavora con un*addestramento continuo*, aggiornando le ipotesi con l'arrivo dei dati (ad esempio per una macchina che deve giocare ad un gioco). Durante la fase di test bisogna conoscere le prestazioni e valutare la correttezza di quanto appreso. Il learner viene addestrato tramite *rewards* e quindi apprende una strategia per massimizzare i *rewards*, detta **strategia di comportamento** e per valutare la prestazione si cerca di massimizzare "a lungo termine" la ricompensa complessivamente ottenuta

Possiamo inoltre distinguere due tipi di apprendimento:

1. **attivo**, dove il *learner* può "domandare" sui dati disponibili

2. **passivo**, dove il *learner* apprende solo a partire dai dati disponibili

Si parla di **inductive learning** quando voglio apprendere una funzione da un esempio (banalmente una funzione target f con esempio $(x, f(x))$, ovvero una coppia). Si cerca quindi un'ipotesi h, a partire da un insieme d'esempi di apprendimento, tale per cui $h \approx f$. Questo è un modello semplificato dell'apprendimento reale in quanto si ignorano a priori conoscenze e si assume

di avere un insieme di dati.Viene usato un approccio che sfrutta anche il *Rasoio di Occam*.

Terminologia:

- X, **spazio delle istanze**
- D, **training set**
- c, **concetto**, $c \subseteq X$
- h, **ipotesi**, $h \subseteq X$
- $(x, f(x))$, **esempio**, tale per cui:

$$f(x) = \begin{cases} 1 & \text{se } x \subseteq X \\ 0 & \text{altrimenti} \end{cases}$$

- $\{(x'_1, f(x'_1)), \ldots, (x'_n, f(x'_n))\}$, **test**
- $\{(x_1, f(x_1)), \ldots, (x_n, f(x_n))\}$, **training set**
- **cross validation**, ovvero ripeto m volte la validazione su campioni diversi di input per evitare che un certo risultato derivi dalla fortuna
- **ipotesi H**, ovvero una congiunzione $\wedge$ di vincoli sugli attributi. Tale ipotesi è **consistente**, ovvero è coerente con tutti gli esempi

Si avrà, in realtà, a che fare con dati, di target e ipotesi, booleani e questo ambito è propriamente chiamato **concept learning**. In questo contesto si cerca di capire quale funzione booleana è adatta al mio addestramento. In altre parole si cerca di apprendere un'ipotesi booleana partendo da esempi di training composti da input e output della funzione. Qualora nel concept learning si abbia a che fare con più di due possibilità si aumentano i bit usati. Nel concept learning un'ipotesi è un insieme di valori di attributi e ogni valore può essere:

- specificato
- non importante e si indica con ?
- nullo e si indica con $\emptyset$

Quindi, dato un training set D, cerco di determinare un'ipotesi $h \in H$ tale che:

$$h(x) = c(x), \forall x \in X$$

Si ha la teoria delle **ipotesi di apprendimento induttivo** che dice che se la mia h approssima bene nel *training set* allora approssima bene su tutti gli esempi non ancora osservati.
Il concept learning è quindi una ricerca del *fit* migliore.

Definizione 1. *Date $h_j, h_k \in H$ booleane e definite su X. Si ha che h_j è **più generale o uguale a** h_k (e si scrive con $h_j \geq h_k$) sse:*

$$(h_k(x) = 1) \longrightarrow (h_j(x) = 1), \ \forall x \in X$$

Si impone quindi un ordine parziale.
*Si ha che h_j è **più generale di** h_k (e si scrive con $h_j > h_k$) sse:*

$$(h_j \geq h_k) \wedge (h_k \ngeq h_j)$$

*Riscrivendo dal punto di vista insiemistico si ha che h_j è **più generale o uguale a** h_k sse:*

$$h_k \supseteq h_j$$

*e che è **più generale di** h_k sse:*

$$h_k \supset h_j$$

*Dal punto di vista logico si ha che h_j è **più generale di** h_k sse impone meno vincoli di h_k*

Lo spazio delle ipotesi è descritto da una congiunzione di attributi.
Parliamo ora algoritmo **Find-S**. Questo algoritmo permette di partire dall'ipotesi più specifica (attributi nulli, ovvero$\emptyset, \ldots, \emptyset$) e generalizzarla, trovando ad ogni passo un'ipotesi più specifica e consistente con il training set D. L'ipotesi in uscita sarà anche consistente con gli esempi negativi dando prova che il target è effettivamente in H. Con questo algoritmo non si può dimostrare di aver trovato l'unica ipotesi consistente con gli esempi e, ignorando gli esempi negativi non posso capire se D contiene dati inconsistenti. Inoltre non ho l'ipotesi più generale.

Algorithm 1 Algoritmo Find-S

function FINDS
  $h \leftarrow$ l'ipotesi più specifica in H
  for *ogni istanza di training positiva* x **do**
    for *ogni vincolo di attributo* a_i *in* h **do**
      if il vincolo di attributo a_i in h è soddisfatto da x **then**
        non fare nulla
      else
        sostituisci a_i *in* h *con il successivo vincolo più generale che è soddisfatto da* x
  return *ipotesi* h

Definizione 2. *Si dice che* h *è* ***consistente*** *con il training set* D *di concetti target sse:*

$$Consistent(h, D) := h(x) = c(x),\ \forall \langle x, c(x) \rangle \in D$$

Definizione 3. *Si definisce* ***version space****, rispetto ad* H *e* D*, come il sottoinsieme delle ipotesi da* H *consistenti con* D *e si indica con:*

$$VS_{H,D} = \{h \in H | \, Consistent(h, D)$$

Vediamo quindi algoritmo **List-Then Eliminate**:

Algorithm 2 Algoritmo List-Then Eliminate

function LTE
  $vs \leftarrow$ *una lista connettente tutte le ipotesi di* H
  for *ogni esempio di training* $\langle x, c(x) \rangle$ **do**
    rimuovi da vs *ogni ipotesi* h *non consistente con l'esempio di training, ovvero* $h(x) \neq c(x)$
  return *la lista delle ipotesi in* vs

Questo algoritmo è irrealistico in quanto richiese un numero per forza esaustivo di ipotesi.

Definizione 4. *Definiamo:*

- G *come il confine generale di* $VS_{H,D}$*, ovvero l'insieme dei membri generici al massimo. È l'insieme delle ipotesi più generali:*

$$G = \{g \in H | \, g \text{ è consistente con } D \wedge$$

$$(\nexists g' \in H \ t.c \ g' \geq q \wedge g' \text{ è consistente con } D\})$$

  Possiamo dire che $G = \langle ?, ?, ?, \ldots ? \rangle$

- *S come il confine specifico di $VS_{H,D}$, ovvero l'insieme dei membri specifici al massimo. È l'insieme delle ipotesi più specifiche:*

$$S = \{s \in H |\, s \text{ è consistente con } D \wedge$$

$$(\nexists s' \in H \text{ t.c } s' \geq s \wedge s' \text{ è consistente con } D\})$$

Possiamo dire che $S = \langle \emptyset, \emptyset, \emptyset, \ldots \emptyset \rangle$

Ogni elemento di $VS_{H,D}$ si trova tra questi confini:

$$VS_{H,D} = \{h |\, (\exists s \in S)\ (\exists g \in G)\ (g \geq h \geq s)\}$$

con $\geq$ che specifica che è più generale o uguale

Vediamo quindi algoritmo **candidate eliminate**

Algorithm 3 Algoritmo Candidate Eliminate

```
function CE
    G ← insieme delle ipotesi più generali in H
    S ← insieme delle ipotesi più specifiche in H
    for ogni esempio di training d = ⟨x, c(x)⟩ do
        if d è un esempio positivo then
            rimuovi da G ogni ipotesi inconsistente con d
            for ogni ipotesi s in S inconsistente con d do
                rimuovi s da S

                aggiungi a S tutte le generalizzazioni minime h di s
                tali che h sia consistente con d e qualche membro di G
                sia più generale di h
            rimuovi da S ogni ipotesi più generale di un'altra in S
        else
            rimuovi da S ogni ipotesi inconsistente con d
            for ogni ipotesi g in G inconsistente con d do
                rimuovi g da G

                aggiungi a G tutte le generalizzazioni minime h di g
                tali che h sia consistente con d e qualche membro di S
                sia più generale di h
            rimuovi da G ogni ipotesi più generale di un'altra in G
    return la lista delle ipotesi in vs
```

Questo algoritmo ha alcune proprietà:

- converge all'ipotesi h corretta provando che non ci sono errori in D e che $c \in H$
- se D contiene errori allora l'ipotesi corretta sarà eliminata dal *version space*
- si possono apprendere solo le congiunzioni
- se H non contiene il concetto corretto c, verrà trovata l'ipotesi vuota

Il nostro spazio delle ipotesi non è in grado di rappresentare un semplice concetto di target disgiuntivo, si parla infatti di **Biased Hypothesis Space**. Studiamo quindi un **unbiased learner**. Si vuole scegliere un H che esprime ogni concetto insegnabile, ciò significa che H è l'insieme di tutti i possibili sottoinsiemi di X. H sicuramente contiene il concetto target. S diventa l'unione degli esempi positivi e G la negazione dell'unione di quelli negativi. Per apprendere il concetto di target bisognerebbe presentare ogni singola istanza in X come esempio di training.
Un learner che non fa assunzioni a priori in merito al concetto target non ha basi "razionali" per classificare istanze che non vede.
Introduciamo quindi il **bias induttivo** considerando:

- un algoritmo di learning del concetto L
- degli esempi di training $D_C = \{\langle x, c(x)\rangle\}$

Si ha che $L(x_i, D_c)$ denota la classificazione assegnata all'istanza x_1, da L, dopo il training con D_c.

Definizione 5. *Il* ***bias induttivo*** *(con* ***bias*** *che normalmente denota una* distorsione *o un* scostamento *dei dati) di L è un insieme minimale di asserzioni B tale che, per ogni concetto target c e D_c corrispondente si ha che:*

$$[B \wedge D_c \wedge x_i] \vdash L(x_i, D_c),\ \forall x_i \in X$$

con $\vdash$ che rappresenta l'implicazione logica

Possiamo quindi distinguere:

- **sistema induttivo**, dove si hanno in input gli esempi di training e la nuova istanza, viene usato l'algoritmo *candidate eliminate* con H e si ottiene o la classificazione della nuova istanza nulla

- **sistema deduttivo** equivalente al sistema induttivo sopra descritto dove in input si aggiunge l'asserzione "H contiene il concetto target" e si produce lo stesso output tramite un **prover di teoremi**

Abbiamo quindi visto tre tipi di *learner*:

1. il **rote learner**, dove si ha classificazione sse x corrisponde ad un esempio osservato precedentemente. Non si ha *bias induttivo*

2. l'algoritmo **candidare eliminate** con **version space**, dove il *bias* corrisponde al fatto che lo spazio delle ipotesi contiene il concetto target

3. l'algoritmo **Find-S**, dove il *bias* corrisponde al fatto che lo spazio delle ipotesi contiene il concetto target e tutte le istanze sono negative a meno che il target opposto sia implicato in un altro modo

2.1 Alberi decisionali

Vediamo come sfruttare una struttura dati discreta, l'**albero di decisione**, per affrontare problemi di *concept learning*. Su questa struttura implementeremo l'algoritmo chiamato **ID3** che tra le ipotesi sceglie il risultato dell'apprendimento tramite esempi di addestramento. La lista delle ipotesi in questo caso è enorme e la scelta è guidata dal cosiddetto *information gain*.
Possiamo quindi scegliere, al posto delle classiche funzioni booleane, **alberi di decisione** per rappresentare un modello che applicato ad esempi non visti ci dirà se applicare in output un'etichetta vera o falsa in base a quanto appreso. Siamo in ambito di **apprendimento supervisionato**.
Si hanno attributi che hanno anche più di due valori. Per ogni ipotesi si ha un albero di decisione, come quello in figura 2.1. In rosso si hanno gli attributi, in blu i valori degli attributi e in arancione le foglie coi risultati. Le foglie sono le risposte booleane.
Avanzando nell'albero cerchiamo una risposta (che ci deve essere).
La flessibilità nella costruzione dell'albero sta nel scegliere gli attributi e i valori di ognuno. Con l'algoritmo **ID3** si costruiscono alberi decisionali in base alle istanze che ricevo (per avere un albero coerente con le istanze ricevute).
Formule booleane possono essere rappresentate in un albero decisionale (figura 2.2 e figura 2.3), costruendo un albero che sia *yes* solo nei casi la formula booleana sia vera.

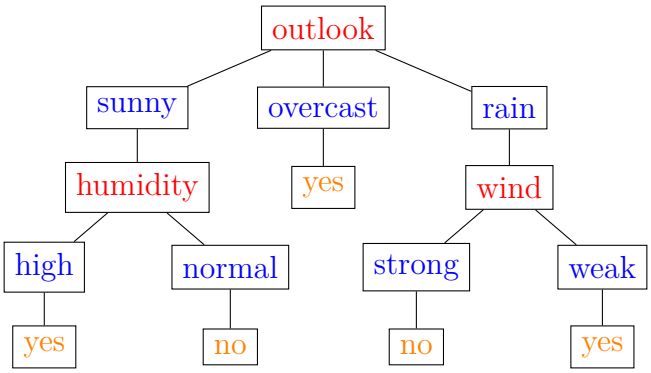


Figura 2.1: Esempio di albero decisionale

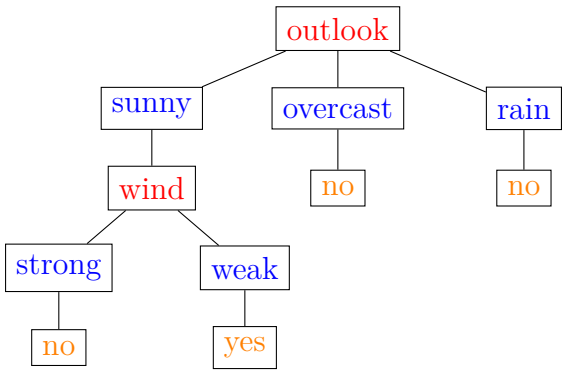


Figura 2.2: Esempio di albero decisionale per la formula $(Outlook = Sunny) \wedge (Wind = Weak)$

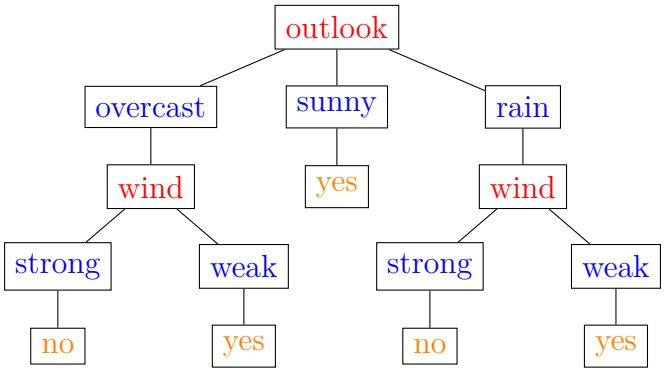


Figura 2.3: Esempio di albero decisionale per la formula $(Outlook = Sunny) \vee (Wind = Weak)$

Notiamo a questo punto come l'albero decisionale in figura 2.1 è la rappresentazione di:

$$(Outlook = Sunny \wedge Humidity = Normal)$$

$$\vee(Outlook = Overcast)$$

$$\vee(Outlook = Rain \wedge Wind = Weak)$$

Possiamo dire che gli alberi decisionali descrivono tutte le funzioni booleane. Avendo n funzioni booleane avremo un numero distinto di tabelle di verità (e quindi di alberi decisionali), ciascuna con 2^n righe, pari a 2^{2^n}.
Riassumiamo alcune caratteristiche degli alberi decisionali:

- abbiamo attributi con valori discreti
- abbiamo un target di uscita discreto, le foglie hanno valori precisi
- posso costruire ipotesi anche con disgiunzioni
- può esserci "rumore" nel training dei dati
- possono esserci attributi di cui non ho informazioni

2.1.1 Algoritmo ID3

Vista la difficoltà di scegliere l'albero si ha l'idea di scegliere un piccolo albero di partenza (o più piccoli) e ricorsivamente l'attributo più significativo (sia nei nodi rossi intermedi che nelle foglie) come radice per il sotto-albero. Si fanno quindi crescere in modo coerente gli alberi piccoli scelti in partenza. Si punta ad arrivare ad un albero valido per tutti gli esempi ricevuti e anche per quelli non visti.

Iniziamo a vedere l'algoritmo anche se saranno necessarie molte specifiche:

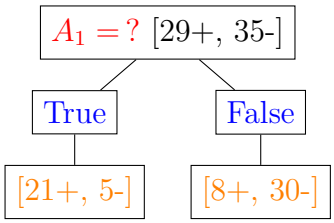
Algorithm 4 Algoritmo ID3

function ID3
  A ← il "miglior" attributo di decisione per il prossimo nodo
  assegno A come attributo di decisione per il nodo, che sarà rosso
  for *ogni valore dell'attributo A* **do**
    creo un discendente
  ordina gli esempi di training alla foglia
  in base al valore dell'attributo del branch
  if *ho classificato tutti gli esempi di training* **then**
    mi fermo
  else
    itero sulle foglie appena create

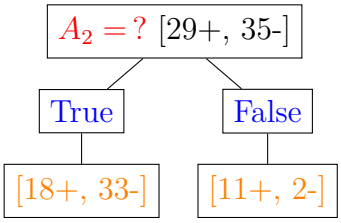
Bisogna in primis capire cosa si intende come **attributo migliore**. Per farlo introduciamo la seguente notazione:

$$[esempi\ \ positivi+,\ esempi\ \ negativi-]$$

entrambi rappresentati con un valore intero. Gli esempi positivi sono gli esempi che già mi hanno restituito *yes* mentre quelli negativi. In generale sono tutti esempi su cui devo ancora valutare l'attributo. E proseguo così assegnando etichette positive e negative. Vediamo quanto detto:



Se siamo nella situazione in cui dobbiamo confrontare l'attributo sopra con un altro, per esempio:



Entrambe le foglie del primo attributi ci parlano di valori sbilanciati (tra esempi positivi e negativi), a differenza delle due del secondo attributo, dove sono una sbilanciata e una no. Per ora stabiliamo ad occhio lo sbilanciamento. Il criterio di scelta ci porta a preferire lo sbilanciamento, verso un ideale "tutti positivi" o "tutti negativi". Quindi se un attributo ha divisioni sbilanciate è da ritenersi migliore.
Per essere ancora più precisi bisogna richiamare la matematica dell'**entropia**.

Definizione 6. *Dato un training set S con valori $v_i,\ i = 1 \dots n$. Se l'entropia di un insieme di bit misura più o meno la sua quantità di informazione (quanto è "speciale") noi possiamo richiamare una formula per l'entropia su S:*

$$I(P(v_1), \dots, P(v_n)) = \sum_{i=1} -P(v_i) \log_2 P(v_i)$$

Dove $I(x)$ indica il valore dell'entropia su x e $P(y)$ sta per la probabilità legata ad un valore y.
Nel caso booleano le istanze presenti in un certo insieme S sono associate ad un'etichetta, conteggiandole. Nella variabile p conto i valori di S con etichetta positiva e con n negativa (esempi positivi e negativi). Ottengo quindi in modo esplicito la sommatoria:

$$I\left(\frac{p}{p+n}, \frac{n}{p+n}\right) = -\frac{p}{p+n} \log_2 \frac{p}{p+n} - \frac{n}{p+n} \log_2 \frac{n}{p+n}$$

*Se inoltre diciamo che p_+ è la proporzione di esempi positivi e p_- di quelli negativi (saranno quindi tra 0 e 1) possiamo misurare l'**impurità** di S con l'entropia:*

$$Entropy(S) = -p_+ \log_2 p_+ - p_- \log_2 p_-$$

Avrò quindi alta entropia se positivi e negativi sono "metà e metà"

Parliamo quindi di **information gain** IG che viene calcolato su ogni attributo A e su S:

$$IG(S, A) = I\left(\frac{p}{p+n}, \frac{n}{p+n}\right) - remainder(A)$$

dove:

$$remainder(A) = \sum_{i=1}^{v} \frac{p_i + n_i}{p+n} I\left(\frac{p_i}{p_i+n_i}, \frac{n_i}{p_i+n_i}\right)$$

e quindi l'information gain è la riduzione aspettata nell'entropia per ordinare S sull'attributo A. Si sceglie l'attributo con il maggiore IG.
Possiamo riscrivere il conto come:

$$IG(S, A) = Entropy(S) - \sum_{v \in values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

Esempio 1. *Vediamo l'esempio di calcolo di entropia di A_1 con [29+,35-]:*

$$Entropy([29+, 35-]) = -\frac{29}{64}\log_2\frac{29}{64} - \frac{35}{64}\log_2\frac{35}{64} = 0.99$$

Calcolo anche l'information gain di A_1, sapendo che $Entropy([21+, 5-]) = 0.71$ e $Entropy([8+, 30-]) = 0.74$, e quindi:

$$IG(S, A_1) = 0.99 - \frac{26}{64} \cdot 0.71 - \frac{38}{64} \cdot 0.74 = 0.27$$

ugualmente calcolo $IG(S, A_2) = 0.12$.
Quindi so che devo scegliere A_1 in quanto $0.27 > 0.12$

Facciamo qualche osservazione finale sull'**algoritmo ID3**:

- lo spazio delle ipotesi è completo e sicuramente contiene il target
- ho in output una singola ipotesi
- non si ha backtracking sugli attributi selezionati, si procede con una ricerca greedy (ma trovo scelte buone localmente e non ottime)
- fa scelte basate su una ricerca statistica, facendo sparire incertezze sui dati
- il bias non è sulla classe iniziale, essendo lo spazio delle ipotesi completo, ma sulla scelta di solo alcune funzioni, preferendo alberi corti (e più semplici) e posizionando attributi ad alto information gain vicino alla radice. Il bias è quindi sulla preferenza di alcune ipotesi. Si usa il criterio euristico di *rasoio di Occam*
- H è l'insieme potenza delle istanze X

Viene introdotto però l'**overfitting**. Se misuro l'errore di una ipotesi h sul training set ($error_{traini}(h)$) e poi misuro l'errore di quella ipotesi sull'intero set delle possibili istanze D ($error_D(h)$) ho che l'ipotesi h va in **overfit** sul quel data set se:

$$error_{traini}(h) < error_{traini}(h') \ \wedge \ error_D(h) > error_D(h')$$

quindi se presa un'altra ipotesi questa è migliore della prima e ha un errore sull'intera distribuzione delle ipotesi inferiore vado in *overfit*. Il problema è che non posso sapere se esiste tale h'. Per evitare il problema uso sempre il rasoio di Occam scegliendo ipotesi semplici ed evitando di far crescere l'albero

quando lo "split" non è statisticamente significativo. Un altro modo è quello di togliere pezzi, all'albero, che toccano poche istanze o pure calcolare una *misura di complessità dell'albero*, minimizzando la grandezza dell'albero e gli errori del training set. Quest'ultima tecnica è detta **Minimum Description Length (*MDL*)** (???????)