

Machine Learning

UniShare

Davide Cozzi
@dlcgold

Indice

1	Introduzione	2
2	Introduzione al ML	3

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/dlccgold/Appunti>.

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 facile 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 il 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 **coniunzione 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:

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

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:

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

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:

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

- $\{(x'_1, f(x'_1)), \dots, (x'_n, f(x'_n))\}$, **test**
- $\{(x_1, f(x_1)), \dots, (x_n, f(x_n))\}$, **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. 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:

$$(h_j \geq h_k) \wedge (h_k \not\geq h_j)$$

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 $\emptyset, \dots, \emptyset$) 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:

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

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:

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

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 richiede 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:

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

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

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

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

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

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

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

con \geq che specifica che è più generale o uguale

Vediamo quindi algoritmo **candidate eliminate**

Algorithm 3 Algoritmo Candidate Eliminate

```
function CE
   $G \leftarrow$  insieme delle ipotesi più generali in  $H$ 
   $S \leftarrow$  insieme delle ipotesi più specifiche in  $H$ 
  for ogni esempio di training  $d = \langle x, c(x) \rangle$  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_i , da L , dopo il training con D_c .

Definizione 5. Il **bias induttivo** 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 **candidate 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