UNIVERSITÀ DEGLI STUDI DI MILANO

FACOLTÀ DI SCIENZE E TECNOLOGIE

Corso di laurea triennale in Informatica

Titolo tesi

Relatore:
Prof. Anna Maria ZANABONI
Correlatore:
Prof. Dario MALCHIODI

Tesi di Laurea di:
Alessia LOMBARDA
Matricola: 908879

Anno Accademico 2020/2021

Contents

1 Introduzione

Prova [1]

2 Apprendimento supervisionato e induzione di insiemi fuzzy

2.1 Apprendimento supervisionato

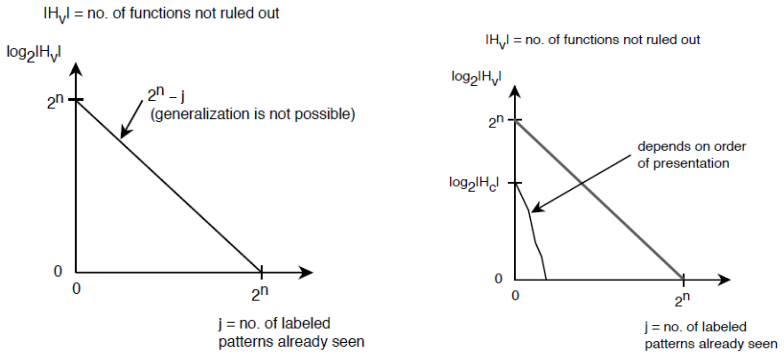
L'apprendimento automatico è una branca dell'intelligenza artificiale che studia algoritmi che utilizzano l'esperienza per migliorare la propria performance o per fare predizioni pi accurate, dove con esperienza intendiamo l'informazione passata disponibile al sistema che apprende, tipicamente una collezione di dati resi disponibili all'analisi. Questi dati possono essere insiemi di informazioni a cui sono state abbinate manualmente delle etichette oppure dati estratti direttamente dall'ambiente: gli algoritmi avranno l'obiettivo di effettuare predizioni sulla base dei dati appresi. L'apprendimento automatico si applica a svariati campi; alcune delle classi principali di problemi di apprendimento sono:

- **Classificazione:** l'assegnamento di ogni elemento ad una categoria; in questo caso di solito il numero di classi possibili è ridotto
- **Regressione:** la predizione di un valore reale per ogni elemento nel set; in questo caso l'errore associato ad una predizione errata dipende della differenza tra il valore predetto e quello reale
- **Ranking:** l'ordinamento di elementi in base a un criterio definito a priori
- **Clustering:** il partizionamento di elementi in regioni omogenee
- **Riduzione della dimensionalità:** la trasformazione di un set di elementi in una rappresentazione degli stessi a meno dimensioni, preservando alcune proprietà degli oggetti iniziali

L'apprendimento automatico può essere *supervisionato*, *non supervisionato* o *semi-supervisionato*; si tratterà la prima categoria di algoritmi.
L'apprendimento supervisionato è uno dei tipi di apprendimento automatico che definisce gli algoritmi in cui il learner riceve un set di dati con label associate (*training set*) per poi eseguire delle predizioni su dati non noti.
L'algoritmo in questione dovrà quindi apprendere una funzione, f, che cercherà di approssimare al meglio con una funzione h, dove sia f che h sono funzioni di un vettore di input $X = (x_1, x_2, ..., x_i, ..., x_n)$ di n componenti. Nel caso di apprendimento supervisionato si conoscono quindi (a volte solo in modo approssimato) i valori di f per gli m elementi del training set, Ξ. Assumiamo che, se possiamo trovare una funzione h che restituisce valori molto vicini a quelli di f per gli elementi di Ξ, allora questa funzione costituisce una buona predizione per f, specialmente se Ξ è grande.

Per quanto riguarda il vettore di input X, questo può essere costituito da valori reali, discreti o categorici, che a loro volta possono essere ordinati (ad esempio {*small, medium, large*}) o non ordinati; un caso a sè è l'uso di valori booleani, che può essere considerato un caso particolare dell'utilizzo di numeri discreti (1,0) o di variabili categoriche (*True*, *False*).
L'output può essere invece un numero reale, ovvero una stima, oppure un valore categorico, tipico dei classificatori. Nel caso di output booleani, invece, possiamo distinguere *istanze positive*, ovvero quelle con label 1, e *istanze negative*, quelle con label 0; se anche l'input è booleano il classificatore implementa una *funzione boleana*.

Definiamo ora il *bias*, ovvero un insieme di informazioni che devono essere note a priori perché l'algoritmo approssimi correttamente la funzione f: perché ciò avvenga, bisogna infatti limitare l'insieme di possibili funzioni, tra cui poi l'algoritmo sceglierà la migliore.
Se ad esempio si considera un algoritmo che deve approssimare una funzione booleana in n variablili, si potranno avere 2^n possibili input. Si supponga di non



(a) (b)

Figure 1: Relazione tra il numero di funzioni possibili e gli elementi del training set già esplorati in assenza e presenza di bias

avere bias, che sia $\mathcal{H}$ l'insieme delle 2^n possibili funzioni booleane e che non si abbiano preferenze tra quelle che danno corrispondenze sui dati del training set. In questo caso, dopo aver considerato un elemento del training set e la sua label si potrà suddividere a metà l'insieme delle funzioni possibili, ovvero distinguere quelle che classificherebbero correttamente quell'elemento dalle restanti. Presentando via via più elementi del training set, ad ogni passo del processo si dimezza il numero di funzioni considerabili come ipotesi di approssimazione per f. Non è possibile in questo caso generalizzare, perché i pattern già trovati non danno alcun indizio su quelli ancora da scoprire: è solo possibile memorizzare i primi, e ciò causa un apprendimento molto oneroso, come mostrato in Figura 1a.
Se invece si limitasse l'insieme $\mathcal{H}$ a un sottoinsieme $\mathcal{H}_c$, dipendentemente dal

sottoinsieme e dall'ordine di presentazione dei dati del training set, la curva che rappresenta il numero di possibili ipotesi di funzioni apparirebbe come in Figura 1b: potrebbe infatti accadere che dopo aver visto meno di 2^n campioni si arrivi già a selezionare una funzione h che ben approssimi f. L'introduzione del bias, quindi, ci permette di considerare solo alcune classi di funzioni, rendendo l'apprendimento più efficiente.

L'algoritmo di apprendimento viene quindi addestrato sul *training set* e successivamente valutato su un insieme distinto di dati, il *test set*. Si dice che una funzione *generalizza* se effettua predizioni corrette sul test set.
In questo contesto è anche necessario selezionare le features rilevanti da considerare per ogni campione, in quanto features significative possono guidare l'algoritmo di apprendimento correttamente, mentre altre povere di significato possono portare a significativi errori: prima di procedere con l'apprendimento è quindi necessario utilizzare un algoritmo di feature selection.
Altra questione da considerare è la possibile presenza di rumore nei vettori del training set: possiamo distinguere il *rumore di classe*, che altera il valore della funzione f, e quello di attributo, che altera in modo causale i valori del vettore di input X, rendendo imprecisa la corrispondenza tra i valori di input e quelli della funzione applicata su di essi.

Fondamentale è anche definire un metodo per la valutazione della performance dell'algoritmo: consideriamo a questo scopo l'*accuratezza* e le *funzioni di errore*. L'accuratezza consiste nel conteggiare il numero di predizioni corrette, e dividere questo valore per il la cardinalità del set di dati: si otterrà un valore compreso tra 0 e 1, che sarè maggiore se l'algoritmo predice in modo corretto. Uno stimatore pi robusto è invece l'*errore quadratico medio*, che si vuole minimizzare e rappresenta il rapporto tra la varianza entro i gruppi e la numerosità totale:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - h(x_i))^2$$

dove y_i è la label attesa e $h(x_i)$ quella predetta dall'algoritmo, e la sua radice quadrata, l'RMSE:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - h(x_i))^2}$$

Per migliorare la performance dell'algoritmo ed evitare che questo si specializzi sui dati (*overfitting*), è necessario svolgere una fase di training accurata; a causa della ridotta dimensione del training set disponibile, solitamente si è soliti usare una tecnica nota come *n-fold cross validation*, usata sia per la model selection (la selezione dei parametri liberi dell'algoritmo), sia per la fase di training.

Approfondiamo inizialmente l'uso della n-fold cross validation per la fase di model selection: detto θ il vettore dei parametri liberi dell'algoritmo, per un fissato valore di θ si partiziona in modo causale un set S di m elementi con label associate in n sottogruppi, o fold. L'i-esimo fold sarà quindi un campione $((x_{i1}, y_{i1}), ..., (x_{im}, y_{im}))$ di cardinalità m_i. Poi, per ogni $i \in [1, n]$ l'algoritmo viene addestrato su un *validation set* costituito da tutti i fold tranne l'i-esimo, per generare un'ipotetica h_i, e la performance di h_i viene testata sull'i-esimo fold. Il valore del parametro θ è scelto sulla base dell'errore medio di h_i, chiamato *cross-validation error*, denotato da $\widehat{R}_{CV}(\theta)$ e definito da

$$\widehat{R}_{CV}(\theta) = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{m_i} \sum_{j=m_i}^{n} L(h_i(x_{ij}), y_{ij})$$

I fold sono generalmente definiti della stessa taglia, ovvero $m_i = m/n$ per ogni $i \in [1, n]$. La scelta di n è molto importante ai fini dell'utilizzo di questo metodo: con un n grande ogni fold avrà taglia $m - m/n = m(1 - 1/n)$, che è vicino a m, la taglia dell'intero dataset, ma i fold saranno piuttosto simili, e quindi il metodo avrà un bias ridotto ed una grande varianza. Viceversa, valori bassi di n portano a training set differenziati, ma la loro taglia è significativamente pi bassa di m e quindi il metodo tenderà ad avere una varianza minore ma un bias maggiore, come si può vedere in Figura 3. Solitamente si scelgono come valori per n 5 o 10.

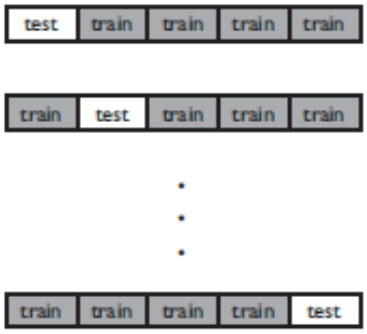


Figure 2: n-fold cross validation

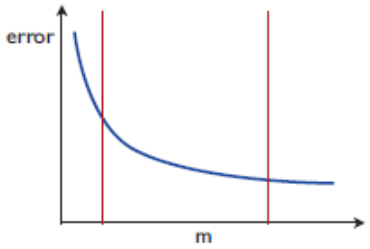


Figure 3: Grafico dell'errore di predizione di un classificatore in funzione della dimensione del training set

Per la model selection, quindi, la cross validation è utilizzata in questo modo: l'intero dataset viene suddiviso inizialmente in training e test set, dove il training set ha taglia m. Questo viene poi utilizzato per computare il cross validation error $\widehat{R}_{CV}(\theta)$ per un certo numero di possibili valori di θ; viene considerato come θ_0 il valore per cui $\widehat{R}_{CV}(\theta)$ è minore, e l'algoritmo viene addestrato con i parametri di θ_0 sull'intero training set di taglia m. La sua performance è poi valutata sul test set. Il metodo si può applicare allo stesso modo per quanto riguarda la fase di training: si otterrà un array di valutazioni (accuratezze o

errori), di cui possiamo calcolare la media per avere un'indicazione della bontà dell'apprendimento.

2.2 Insiemi fuzzy e induzione della funzione di membership

3 I diversi approcci al problema

3.1 Metodi basati sulla percezione

3.2 Metodi euristici

3.3 Clustering

3.4 Metodi basati su probabilità e possibilità

3.5 Support Vector Machines

3.6 Reti neurali

3.7 Nearest neighbour

3.8 Algoritmi genetici

4 Implementazione ed esperimenti

4.1 Algoritmi basati su clustering

4.2 Algoritmi basati su support vector machines

4.3 Risultati

5 Conclusioni

6 Riferimenti bibliografici

References

[1] Autore. Titolo. *Rivista*.