Tozzi Senesi Leonardo
7013640

Relazione dell'elaborato di Intelligenza Artificiale

Comando dell'esercizio assegnato

Riconoscimento di numeri civici

In questo esercizio si utilizzano implementazioni disponibili di Random Forest (p.es. scikit-learn in Python o Weka in Java) per classificare immagini di numeri civici, studiando l'andamento dell'errore di generalizzazione con il numero di esempi. Concretamente si utilizza il dataset SVHN (nel formato cropped e convertendo le immagini in grayscale). Si riporti l'errore di predizione sul training set e sul test set, usando dimensioni del training set di 2^k, per k crescente da 10 fino al massimo valore compatibile con le risorse di calcolo disponibili (eventualmente limitando il numero di iterazioni dell'algoritmo). Si presti attenzione a garantire che, per ogni k, le classi nel training set siano bilanciate.

Svolgimento

Nell'esercizio si utilizzano i dati presenti nel dataset SVHN fornito.
In particolare il dataset è composto da dei dati di training (X_train) , dei dati di test (X_test) e dei dati extra (X_extra). Ciascuno di questi è formato da valori raccolti in tensori di 4 dimensioni dove i primi due indici due specificano i pixel (32x32), il terzo il livello di colore considerato (0=r, 1=g, 2=b) e il quarto il numero dell'immagine considerata. Le immagini sono date in formato RGB e, come da comando, prima di procedere ad effettuare qualsiasi operazione devono essere trasformate in formato grayscale.
Per fare ciò ho definito una funzione *rgb2gray(Xa)* che prende in ingresso un tensore 4d e restituisce un tensore 3d con i valori dei pixel modificati per ottenere il formato grayscale, utilizzando la formula:

*Grayscale = 0.2989*r + 0.5870*g + 0.1140*b*
(Fonte: StackOverflow)

Successivamente ho salvato le nuove immagini per riutilizzarle senza dover necessariamente tutte le volte eseguire la funzione che, per terminare l'esecuzione, impiega diversi minuti. Di seguito si riporta la definizione della funzione sopracitata:

```
def rgb2gray(Xa):
    r=float(0)
    g=float(0)
    b=float(0)
    ris=float(0)
    dim=int(Xa.shape[3])
    Xb=np.zeros([32,32,dim])
    for i in range(dim):      #per ogni immagine
        for x in range(32):   #per ogni colonna
            for y in range(32):  #per ogni riga
                r=Xa[x,y,0,i]
                g=Xa[x,y,1,i]
                b=Xa[x,y,2,i]
                #print(r)
                #print(g)
                #print(b)
                ris= float(0.2989 * r + 0.5870 * g + 0.1140 * b)
                #print(ris)
                Xb[x][y][i]=ris
                #print(X_train[x,y,i])
            y=0
        x=0
    return Xb
```

Dopo aver adattato i vari datasets per soddisfare le specifiche e per renderli compatibili tra di loro, eventualmente effettuando anche la trasposizione, era necessario mostrare l'andamento dell'errore di predizione sul training set e sul test set al crescere della dimensione del training set. In particolare dovevo considerare il training set con dimensione pari a 2^K con K crescente da 10 a 18, in quanto la dimensione del dataset di train era 138610($\simeq$2^18).
Per fare ciò ho importato la libreria Sickit-learn utilizzando il classificatore Random Forest. Ho "allenato" il classificatore con il training set a dimensione crescente e poi ho calcolato la predizione utilizzando la funzione .predict prima su X_test e poi su X_train e calcolato l'accuracy del mio classificatore nei due casi e il conseguente errore. Per visualizzare meglio i risultati ho scelto di disegnarne il grafico che riporto:

Come possiamo notare dal grafico l'errore di predizione sul dataset di test ha un andamento decrescente, invece ha un andamento costante quello relativo al dataset di train. Appare subito chiaro come l'errore di predizione sul training set sia costantemente circa lo 0%, mentre quello sul test set inizialmente assuma valori abbastanza elevati che decrescono esponenzialmente. Siamo quindi in presenza di overfitting.

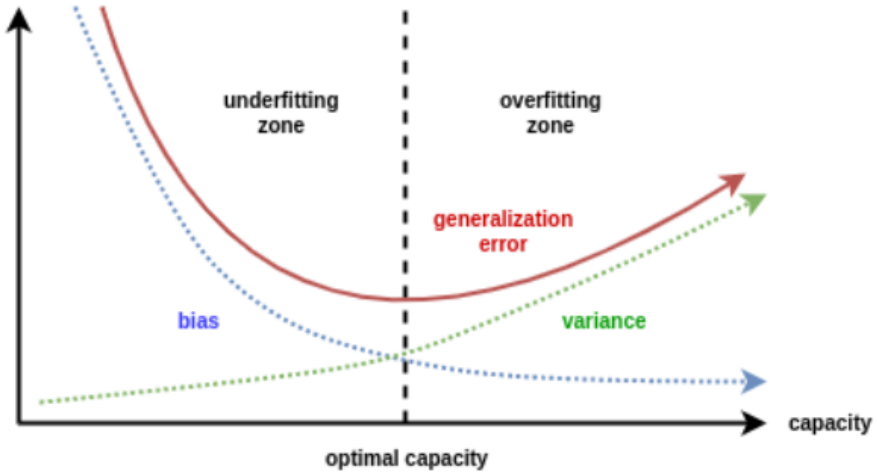
(Fonte immagine: AndreaMinini.com)

Commenti sul risultato ottenuto

In primo luogo, definiamo l'"errore di generalizzazione".

"Nelle applicazioni di apprendimento supervisionato nell'apprendimento automatico e nella teoria dell'apprendimento statistico, l'errore di generalizzazione (noto anche come errore fuori campione) è una misura della precisione con cui un algoritmo è in grado di prevedere i valori di risultato per dati precedentemente invisibili." Wikipedia

L'errore di generalizzazione è quindi una misura di quanto bene un modello di apprendimento automatico si comporta (cioè prevede) su dati precedentemente non visti. Quindi, più è piccolo, meglio è. Questo è anche chiamato errore fuori campione.
Come vediamo l'errore sul training set è quasi nullo, quindi l'algoritmo è in grado di imparare quasi alla perfezione i dati in ingresso, ma questo comporta che impari anche tutte le informazioni distorte (ovvero le cifre presenti ai lati delle immagini che distraggono dalla cifra di interesse) con la conseguenza di avere sì un errore basso sui dati già noti del training set ma anche quella di sviluppare una capacità predittiva bassa e quindi un errore di generalizzazione alto, che però diminuisce al crescere della dimensione del training set in quanto ci avviciniamo sempre di più ad un modello well-fitted.



(Fonte immagine: Ichi.pro)

Come vediamo dal grafico sopra riportato l'errore di generalizzazione dipende da due valori:

- Il bias che misura essenzialmente l'incapacità di un algoritmo di adattarsi sufficientemente bene, alla distribuzione dei dati di un set di training.
- La varianza che misura la sensibilità di un algoritmo rispetto a uno specifico set di dati di training (se la varianza è alta vi è un adattamento eccessivo).

Quindi il bias rappresenta l'accuracy sul training set, mentre la varianza rappresenta l'accuracy sul test set.
In particolare l'errore di generalizzazione è dato da:

$$Error = Var(x) + Bias^2(x) + \varepsilon^2$$

(ε = rumore ,Fonte: Ichi.pro)

Nel nostro caso l'errore di generalizzazione, quindi, è abbastanza elevato in quanto sui dati di addestramento abbiamo un errore di predizione quasi nullo, mentre sui dati ancora non visti, ovvero sul test set, l'errore di predizione ha un valore grande.