Confronto fra One-vs-One e One-vs-Rest Perceptron

Marco Ciccone

Lo scopo dell'elaborato è quello di realizzare un'implementazione del famoso algoritmo di apprendimento Perceptron, sia nella versione binaria che multiclasse.
Per quanto riguarda il classificatore binario semplice è stata implementata la versione "Voted Perceptron" descritta da Freund & Schapire (1998). Mentre per la versione multiclasse sono stati confrontati i risultati ottenuti con le strategie One-vs-One (All-Pairs) e One-vs-Rest.

In particolare si è applicato l'algoritmo al problema di riconoscimento di caratteri manoscritti.

Il linguaggio scelto per l'implementazione è Python. Sono state utilizzate le librerie NumPy e Matplotlib per le operazioni sui vettori (Norm, Dot Product) e fare il plot di alcuni semplici
esempi.

Introduzione

Sulla base del modello di neurone proposto da McCullough e Pitts (fig. 1), Rosenblatt introdusse nel 1962 il perceptron, un'unità funzionale a n ingressi reali $x_1, \ldots, x_n$ e una sola uscita binaria y. Se impieghiamo gli interi -1 e +1 per codificare i due stati binari il comportamento del perceptron è descritto dalla seguente relazione:

$$y = \text{sign}((\boldsymbol{w} \cdot \boldsymbol{x}) - b) = \text{sign}\left(\sum_{i=1}^{n} w_i x_i - b\right)$$

dove $(w \cdot x)$ è l'usuale prodotto interno in R^n tra i vettori $w = (w_1,\ldots,w_n)$ e $x = (x_1,\ldots,x_n) \in R^n$

$$(\boldsymbol{w} \cdot \boldsymbol{x}) = \sum_{i=1}^{n} w_i x_i$$

I numeri reali $w_1, \ldots, w_n$ contenuti nel vettore w vengono chiamati pesi del neurone, mentre lo scalare $b \in R$ è denominato soglia e decide quale valore della somma pesata $(w \cdot x)$ produce la commutazione dell'uscita dal valore -1 al valore +1. Queste quantità costituiscono i parametri del perceptron, ai quali viene assegnato un valore dal processo di addestramento. Se poniamo $w_0 = -b$, possiamo riscrivere l'equazione del perceptron nel modo seguente:

$$y = \text{sign}((\boldsymbol{w} \cdot \boldsymbol{x}) + w_0) = \text{sign}\left(\sum_{i=1}^{n} w_i x_i + w_0\right) = \text{sign}(\bar{\boldsymbol{w}} \cdot \bar{\boldsymbol{x}})$$

dove $w' = (w_0,w_1,\ldots,w_n)$ è un vettore di R^{n+1} contenente tutti i parametri del perceptron (indicato sempre con il nome vettore dei pesi), mentre $x = (1, x) = (1, x_1, \ldots, x_n)$ è un vettore di R^{n+1} ottenuto aggiungendo una prima componente fissata pari a 1 al pattern d'ingresso dato x. Il numero reale w_0 prende il nome di bias del perceptron.

Tale dispositivo può quindi essere impiegato esclusivamente per la soluzione di problemi di classificazione, nei quali vengono associati rispettivamente i valori di uscita y = -1 e y = +1 ad ognuna delle due classi.

L'addestramento del perceptron consiste nel trovare il vettore dei pesi piu` opportuno $w \in R^{n+1}$ per il problema di classificazione, a partire da un training set contenente m coppie ingresso-uscita $(x_1,y_1),\ldots,(x_m,y_m)$, con $x_i \in R^n$ e $y_i \in R$ per ogni $i = 1,\ldots,m$.

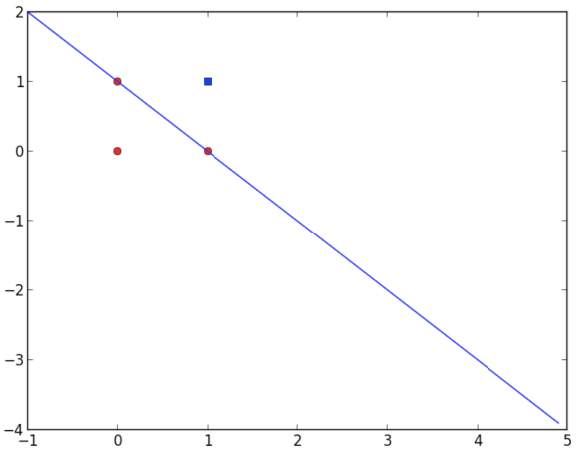
Voted Perceptron

La versione Voted Perceptron è leggermente differente dal Perceptron originale di Rosenblatt.
Il Voted Perceptron mantiene in memoria tutti gli iperpiani generati dall'algoritmo di apprendimento, e a ognuno viene dato un punteggio.
Se la predizione è giusta, allora viene aumentato il suo punteggio e si mantiene l'iperpiano per la prossima iterazione, altrimenti si aggiorna l'iperpiano con la regola di aggiornamento e si inizializza il punteggio a 1.
Alla fine dell'algoritmo di apprendimento avremo una lista contenente tutti gli iperpiani trovati e per ognuno di essi il numero di volte che ha fatto una previsione corretta.
Per il test si utilizza la seguente formula, dove ogni iperpiano trovato ha un diritto di voto, e il peso del voto è dato dal numero di volte che l'iperpiano "si è comportato bene" durante la fase di apprendimento.

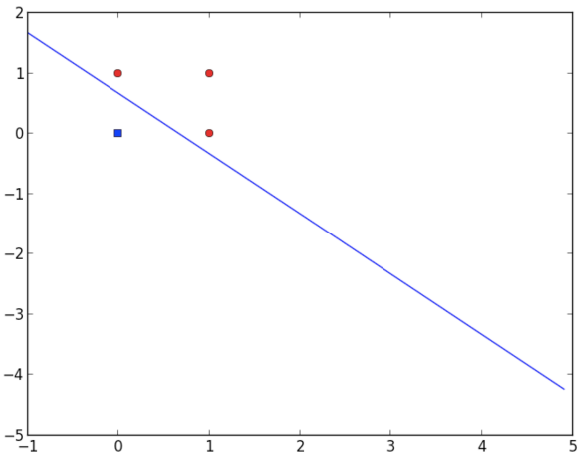
Sono state utilizzate strutture dati molto semplici per il Voted Perceptron.

E' stata realizzata una classe Perceptron che contiene il vettore dei pesi W e la soglia b correnti.
Inoltre viene mantenuta in memoria una copia di ogni W e b con il numero di volte in cui hanno avuto successo durante la fase di training, grazie a 3 liste wlist, blist, clist.

L'algoritmo è stato testato con alcuni dataset giocattolo, in modo da verificare la correttezza del classificatore binario.
Sono state testate le funzioni booleane AND e OR con i seguenti risultati.



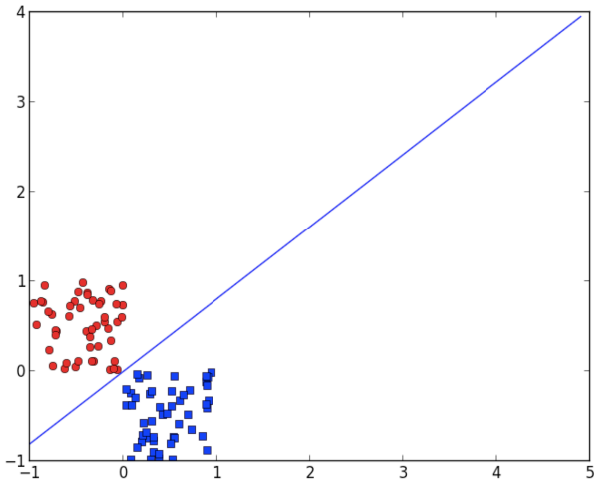
AND
w: [2.0, 2.0], b: 2.0



OR
w: [3.0, 3.0], b: 2.0

E' stato verificato che il perceptron non può trovare un iperpiano di separazione valido per la funzione XOR.

Inoltre è stato generato un dataset linearmente separabile con due feature x1,x2 per avere un'ulteriore conferma sulla correttezza dell'algoritmo.



Perceptron Multiclass

Per applicare l'algoritmo del Perceptron a un problema come il riconoscimento di caratteri manoscritti, occorre utilizzare una versione modificata dell'algoritmo per risolvere problemi di tipo multiclasse.
Il Perceptron infatti è un classificatore binario, ma possiamo combinare n Perceptron per trovare una soluzione al problema.

Sono state confrontate due strategie : One-vs-One (All Pairs) e One-vs-Rest.

One-vs-Rest
Dato K il numero di classi, la strategia One-vs-Rest prevede che il problema multiclasse venga suddiviso in K sottoproblemi binari.
Vengono istruiti separatamente K Perceptron binari, dove si trovano iperpiani di separazione fra una gli oggetti di una classe e tutti gli altri oggetti delle altre classi.
Al termine della fase di apprendimento si ottengono K iperpiani.
Nella fase di test la predizione viene effettuata seguendo la seguente regola:

per ogni classe si calcola :

$$f_k(x) = w'x + b_k$$

$$h(x) = argmax_{k=1...K} f_k(x)$$

One-vs-One
Dato K il numero di classi, la strategia One-vs-One prevede che il problema multiclasse venga suddiviso in K(K-1)/2 sottoproblemi binari.
Si utilizza così una sorta di "algoritmo del campionato di calcio" dove si fanno tutti gli abbinamenti possibili fra le classi.
Nella fase di apprendimento vengono istruiti K(K-1)/2 Perceptron binari differenti, in cui si confronta ogni classe con tutte le altre.
Alla fine si ottengono quindi K(K-1)/2 iperpiani e la predizione sul dataset da testare viene effettuato con ognuno di essi. La classe che ha ottenuto più punti è quella che viene predetta.

Risultati
Confrontiamo i risultati ottenuti con il dataset USPS:

	Digit 0	Digit 1	Digit 2	Digit 3	Digit 4	Digit 5	Digit 6	Digit 7	Digit 8	Digit 9
# Succ su Tot	354 su 359	252 su 264	187 su 198	121 su 166	117 su 200	118 su 160	139 su 170	139 su 147	125 su 166	59 su 177
% Succ	98.61%	95.45%	94.44%	72.89%	58.5%	73.75%	81.76%	94.56%	75.30%	33.33%

One-vs-One

Totale
1611 su 2007
80.27% di risultati predetti correttamente

	Digit 0	Digit 1	Digit 2	Digit 3	Digit 4	Digit 5	Digit 6	Digit 7	Digit 8	Digit 9
# Succ su Tot	334 su 359	185 su 264	198 su 198	106 su 166	92 su 200	131 su 160	136 su 170	139 su 147	70 su 166	31 su 177
% Succ	93.04%	70.08%	100.0%	63.86%	46.00%	81.88%	80.00%	94.56%	42.17%	17.51%

One-vs-Rest

Totale 1422 su 2007
70.85% di risultati predetti correttamente

I test sono stati effettuati con un limite al numero di epoche pari a 100. Il fatto di avere un numero di epoche prefissato ha reso inutile il learning rate, infatti i test effettuati con learning rate diversi danno sempre stessi risultati.
One-vs-One è molto più costoso della strategia One-vs-Rest perchè deve far apprendere K(K-1)/2 Perceptron contro i K del OvR, sebbene comunque i sottoproblemi siano di dimensione inferiore visto che l'apprendimento si fa solo con due classi alla volta e non con tutto il dataset come in OvR.

Con OvO si ottengono risultati precisi a discapito di un aumento di complessità. Le percentuali di successo sono più omogenee fra le classi.
OvR non riesce a ottenere previsioni corrette per le classi 9 (17.51%),4 (46.00%), 8 (42.17%).
Mentre invece predice correttamente il 100% dei test per la classe 2.

Con OvO abbiamo ottenuto dei miglioramenti su tutte le classi, in particolare su 9 (33.33%),4 (58.5%), 8 (75.30%), a discapito della classe 2 (94.44%)

I risultati fra le due strategie sono in accordo fra di loro.

Possiamo pensare che la classe 9 sia un test difficile per l'algoritmo del Perceptron.

Non essendo un dataset linearmente separabile questi risultati sono nella norma, per ottenere previsioni più corrette nel riconoscimento di caratteri manoscritti occorre utilizzare feature migliori dei pixel dell'immagine e funzioni kernel.