Dual Perceptron (dual form)

Lorenzo Ricci

Università degli studi di Firenze
Corso di Laurea in Ingegneria Informatica
Intelligenza Artificiale

A.S. 22/23

Sommario

1 Introduzione algoritmo **3**

2 Implementazione **3**

2.1 KernelFunctions 4

2.2 MyDualPerceptron 4

1 Introduzione algoritmo

Il primo algoritmo iterativo per l'apprendimento di classificazioni lineari è la procedura proposta da Frank Rosenblatt nel 1956 per il perceptron. E' una procedura *on-line* e *mistake-driven*, che inizia con un vettore peso iniziale **w0** (solitamente inizializzato tutto a 0, **w0=0**) e si adatta ogni volta che un punto, che sta venendo addestrato, viene malclassificato dai pesi attuali. L'algoritmo aggiorna il vettore peso e il bias direttamente. Inoltre questa procedura ha garantita la convergenza dall'esistenza di un iperpiano che classifica correttamente i punti su cui lo si sta facendo addestrare, e in questo caso si dice che i dati sono *linearmente separabili*. Quindi, viceversa, se non esiste un iperpiano i dati si dicono non separabili. Si definisce *margine funzionale di un esempio* ($\mathbf{x}_i$,y_i) con rispetto all'iperpiano ($\mathbf{w}$,b), la quantità:

$$\gamma_i = y_i(\langle x_i, x_j \rangle + b)$$

e si nota che se $\gamma > 0$ implica una corretta classificazione di ($\mathbf{x}_i$,y_i). L'algoritmo Perceptron lavora quindi, aggiungendo esempi di addestramento positivi classificati in modo errato o sottraendo quelli negarivi classificati in modo errato ad un vettore peso scelto all'inizio in modo aribitrario. Senza perdita di generalità, se si assume che il vettore peso iniziale è un vettore zero, e che l'ipotesi finale sarà quella di essere una combinazione lineare dei punti di addestramento, possiamo ridefinire il vettore peso:

$$\mathbf{w} = \sum_{i=1}^{l} \alpha_i y_i \mathbf{x}_i$$

dove, dal momento in cui il segno del coefficiente di $\mathbf{x}_i$ è dato dalla classificazione di y_i, gli α_i sono volori positivi proporzionali al numero di volte in cui una classificazione errata di $\mathbf{x}_i$ ha causato l'aggiornamento del peso. Punti che hanno causato pochi errori avranno un valore più piccolo di α_i, viceversa punti più difficili avranno questo valore più grande. Quindi fissato un set di addestramento S, si può pensare al vettore α come rappresentazione alternativa dell'ipotesi in coordinate diverse o duali.

2 Implementazione

Lo scopo del seguente progetto è quello di implementare l'algoritmo nella sua forma duale descritto in (Cristianini & Shawe-Taylor 1999), permettendo l'uso di funzioni kernel al posto del prodotto scalare ⟨xi, xj⟩.

Given training set S
$\boldsymbol{\alpha} \leftarrow \mathbf{0}$; $b \leftarrow 0$
$R \leftarrow \max_{1 \le i \le \ell} \|\mathbf{x}_i\|$
repeat
    for $i = 1$ to ℓ
        if $y_i \left(\sum_{j=1}^{\ell} \alpha_j y_j \langle \mathbf{x}_j \cdot \mathbf{x}_i \rangle + b \right) \le 0$ then
            $\alpha_i \leftarrow \alpha_i + 1$
            $b \leftarrow b + y_i R^2$
        end if
    end for
until no mistakes made within the *for* loop
return $(\boldsymbol{\alpha}, b)$ to define function $h(\mathbf{x})$ of equation (2.1)

Table 2.2: **The Perceptron Algorithm (dual form)**

2.1 KernelFunctions

La classe KernelFunctions fornisce tre metodi, uno per ogni tipologia di funzione kernel.

- Linear kernel: prende in ingresso una matrice X e ritorna il prodotto scalare tra X e X trasposta.
- Polynomial kernel: prende in ingresso una matrice X, un paramentro c, che viene sommato al prodotto scalare come nel kernel lineare e un ultimo paramento d che definisce il grado del polinomio.
- RBF kernel: prende in ingresso una matrice X e un parametro gamma e ritorna l'esponenziale della norma negativa moltiplicata per -gamma, il cui tutto è elevato al quadrato.

2.2 MyDualPerceptron

La mia implementazione dell'algoritmo consiste nella classe MyDualPerceptron. Il costruttore prende in ingresso la tipologia di kernel che si vuole utilizzare, ovvero un numero: 1 per il kernel lineare, 2 per il kernel polinomiale e 3 per l'rbf kernel. Lo pseudocodice mostrato in Table 2.2 viene implementato nel metodo *train()*, che prende in ingresso X, y, epochs. In particolare:

- X: Porzione di data set, ovvero i dati che verrano utilizzati per addestrare il vettore alpha e il bias.
- y: Porzione di data set, ovvero il vettore dei target di addestramento.
- epochs: il numero di epoche(iterazioni) del ciclo for sul quale si vuole addestrare l'algoritmo.

Inoltre all'interno del metodo *train()*, in base al tipo di kernel scelto dall'utente vengono mappati i dati di addestramento X e quindi creata la matrice K (nsamples,nsamples). Infine si utilizza un ciclo for per implementare il repeat dal

quale si uscirà quando all'interno del ciclo for interno non ci saranno errori. Una volta eseguita la funzione di *train()* che addestra i propri parametri alpha e b (bias), verrà chiamato il metodo *predict()* che prende anche lui in ingresso X e y, i quali però non saranno uguali a quelli del *train()*, ma saranno un'altra porzione di data set che viene utilizzata per testare l'accuracy dell'algoritmo nel predire i dati del target in output.