Kernelized Dual Perceptron

Luca Bindini

1 Introduzione

Questo elaborato verte sulla realizzazione di un algoritmo per la classificazione binaria: il **Perceptron**.
Nello specifico è stato realizzato nella sua forma duale come descritto nel libro *An Introduction to Support Vector Machines and other kernel-based learning methods*.
La forma duale ci consente di utilizzare particolari funzioni (dette funzioni *kernel*) al posto del canonico prodotto scalare.

2 Brevi cenni teorici

La forma duale del *Perceptron* consente di applicare l'algoritmo anche a dei datasets non linearmente separabili e grazie alle funzioni *kernel* riesce a trovare un iperpiano che separi i dati in modo più efficiente (a seconda del tipo di funzione kernel).
I pesi del *Perceptron* nella sua forma duale sono rappresentati da un vettore α (inizialmente uguale al vettore nullo) dove ogni posizione i indica il numero di volte che l'algoritmo ha sbagliato a classificare l'elemento y_i.
La nuova funzione di decisione nella forma duale sarà:

$$h(x) = sgn(\sum_{j=1}^{l} \alpha_j y_j < x_j \cdot x > + b)$$

dove l è la cardinalità del train-set e b è il *bias*.
La funzione di decisione darà come output i valori 1 oppure -1 (problema di classificazione binaria).

N.B una funzione kernel K deve rispettare alcune proprietà:

- *Simmetrica*: ossia deve valere che $K(u, v) = K(v, u)$
- *Definita Positiva*: ossia se la matrice $K_{ij} = K(x_i, x_j)$ non ha autovalori negativi.

3 Datasets utilizzati

I 3 dataset utilizzati per questo progetto sono reperibili dal sito **UCI Machine Learning Repository** e sono i seguenti:

1. **Banknote Authentication Data Set**: dataset per la classificazione di banconote autentiche o false.
2. **QSAR Biodegradation Data Set**: dataset per la classificazione della biodegradabilità di alcune molecole.
3. **QSAR Androgen Receptor Data Set**: dataset per la classificazione binaria di recettori nucleari.

4 Implementazione

L'intero progetto è stato realizzato in linguaggio *Python 3.8* con l'ausilio di alcune librerie esterne utili ai fini della gestione del dataset e alla visualizzazione e il salvataggio dei risultati in forma grafica.

4.1 Esecuzione

Il dataset viene inizialmente caricato (da file *.csv*) attraverso il metodo *load_single_dataset* e splittato in 3 parti *train*, *validation* e *test-set* (con la proporzione del 60-20-20 %). Successivamente verranno istanziati tre oggetti di tipo *KernelizedDualPerceptron* ognuno operante con kernel diverso.
Una volta fatto il *training* si prova a predirre i dati sul *test-set* e se ne valuta l'accuratezza attraverso il metodo *accuracy_rate* contenuto nel file *performances*.

4.2 Funzioni kernel utilizzate

Per questo progetto sono state utilizzate 3 diverse funzioni kernel:

1. **Kernel Lineare**: $K(u, v) = u \cdot v$
2. **Kernel Polinomiale**: $K(u, v) = (u \cdot v + 1)^p$ (polinomi aventi grado massimo p)
3. **Kernel RBF**: $K(u, v) = exp(-\frac{\|u-v\|^2}{2\sigma^2})$

4.3 Matrice di Gram

Nel *Dual Perceptron* nella fase di *training* è necessario il calcolo della funzione kernel per ogni coppia di esempi del *train-set*, per questo può essere conveniente calcolare a priori una matrice detta **Matrice di Gram** definita come:

$$G_{ij} = K(x_i, x_j)$$

questa matrice avrà dimensione $N \times N$, con N cardinalità del train-set, quindi se il dataset è molto grande essa può diventare decisamente onerosa in termini di memoria.

4.4 Problema dell'overfitting

Il *Perceptron* nella fase di training eseguirà l'apprendimento del *train-set* per un numero di volte "sufficientemente" grande, dove ogni iterazione è detta *epoca* (nel codice è presente una costante *MAX_ITERATION* che indica il numero massimo di epoche).
Tuttavia può accadere che da una certa iterazione in poi il *Perceptron* cominci ad imparare "troppo bene" sul *train-set* ma andando a perdere in accuratezza sui nuovi dati (problema dell'**overfitting**).
Per arginare questo problema il dataset è stato suddiviso in un'ulteriore parte detta *validation-set* e ad ogni iterazione (epoca), durante la fase di training, si calcola l'errore sul *validation-set* e se esso è maggiore rispetto all'iterazione precedente vuol dire che avrei dovuto fermarmi all'epoca precedente.
Questo meccanismo è detto **Early stopping**.

Figura 1: Come si può notare l'errore sul validation-set inizia ad aumentare a partire dall'iterazione 6 e raggiunge quindi il minimo alla quinta iterazione. Questo esempio è stato realizzato sul dataset Banknote utilizzzando un kernel polinomiale.

5 Risultati

Andando a confrontare i vari kernel sui vari dataset è stato possibile osservare come sostanzialmente nella quasi totalità dei casi il kernel *RBF* fornisce l'accuratezza migliore mentre il kernel *lineare* quella peggiore.
Qui di seguito è rappresentato un grafico a barre dove per ogni dataset e per ogni kernel viene valutata l'accuratezza.

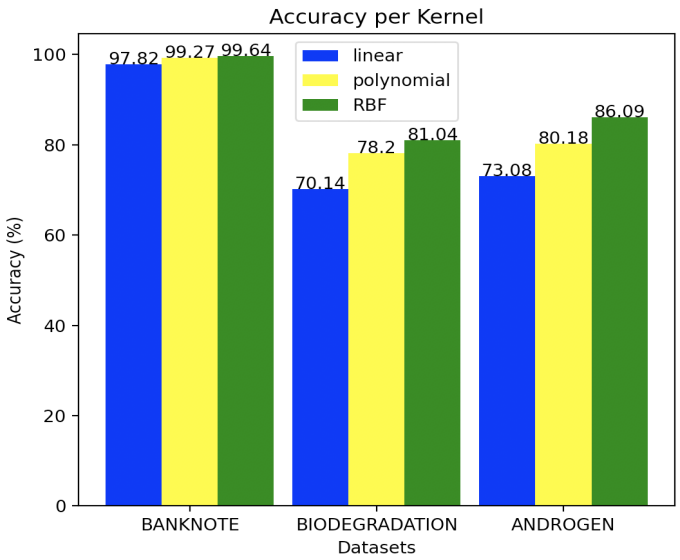

Figura 2: Confronto dell'accuratezza per i vari kernels

6 Riferimenti Bibliografici

S. Russel, P. Norvig. Artificial Intelligence: A Modern Approach. 3rd edition. Pearson, 2010

N. Cristianini, J. Shawe-Taylor. An Introduction to Support Vector Machines and other kernel-based learning methods. Cambridge, 2013

E. Fox. Kernelized Perceptron Support Vector Machines. University of Washington, 2017

L. Prechelt. Early Stopping. University of Karlsruhe