Ottimizzazione dei parametri di un classificatore SVM non lineare con funzione di base radiale come kernel

Stefano Agostini

***Abstract*— Questo elaborato ha l'obiettivo di mettere a confronto vari algoritmi di ottimizzazione globale per funzioni valutabili facilmente in molti punti, applicandoli al caso d'uso della ricerca dei parametri migliori per un classificatore Support-Vector Machine (SVM).**

I. INTRODUZIONE

Le Support-Vector Machines (SVM) sono dei modelli di apprendimento supervisionato, associati ad algoritmi di regressione e classificazione. Dato un insieme di esempi per l'addestramento (chiamato training set), ognuno dei quali etichettato con la classe di appartenenza, un algoritmo di addestramento per le SVM costruisce un modello che assegna i nuovi esempi ad una delle classi. Gli esempi vengono mappati come punti nello spazio e vengono separati in modo tale che quelli appartenenti a diverse categorie siano divisi da uno spazio il più ampio possibile. Quando arrivano nuovi esempi non classificati, questi vengono mappati nello stesso spazio e vengono identificati con la classe relativa a quella regione dello spazio.
Oltre alla classificazione lineare è possibile fare uso delle SVM per svolgere la classificazione non-lineare utilizzando un kernel, mappando implicitamente gli input in uno spazio delle caratteristiche multi-dimensionale.
Per la classificazione non lineare con un kernel Radial Basis Function (RBF) i parametri fondamentali sono due:

- **C**, parametro di multa nella funzione di decisione per gli esempi non linearmente separabili. Un valore basso di questa costante, consente una classificazione con un margine di separazione più grande, ma influisce negativamente sull'accuratezza dell'addestramento. Un valore alto, invece, consente una accuratezza migliore sul training set, che può portare ad un overfitting.
- γ , parametro che rappresenta l'inverso della deviazione standard del kernel. È usata come misura di somiglianza tra due punti. Un valore basso di questo parametro può portare a considerare due punti simili anche se molto diversi, mentre un valore alto può portare a considerare due punti simili solo se uguali tra loro.

Risulta quindi importante usare valori consoni di tali variabili per avere una classificazione il più generale possibile.
Per ottimizzare questi parametri è possibile usare algoritmi di ottimizzazione globale per ottimizzare la funzione di accuratezza del classificatore su un certo esempio di valutazione. L'unico vincolo sulle variabili da imporre per questi algoritmi è quello di cercare nel solo dominio positivo.
Gli algoritmi di ottimizzazione globale utilizzati in questo elaborato sono tutti algoritmi per funzioni poco costose poiché calcolano il valore della funzione in molti punti e valutano grazie ad esso dove cercare un prossimo candidato in cui la funzione potrebbe essere migliore.

II. ESPERIMENTO

Per svolgere l'elaborato, si è deciso di utilizzare un SVM con kernel RBF sul database Iris e di ottimizzare i suoi parametri C e γ tramite tre algoritmi di ottimizzazione globale: Basin Hoppin, Differential Evolution e Dual Annealing. Il codice è stato scritto il linguaggio Python3. Come libreria per la parte relativa a SVM è stata usata *scikit-learn*, mentre gli algoritmi di ottimizzazione sono implementati nella libreria *scipy* (v1.2.1).

A. Dataset

Il dataset usato per l'esperimento è Iris, composto da 150 istanze di iris classificate secondo tre specie: Iris setosa, Iris virginica e Iris versicolor. Ogni istanza è composta da quattro caratteristiche che rappresentano la lunghezza e la larghezza del sepalo e del petalo. Le classi hanno tutte 50 istanze di esempio.

B. Algoritmi

- **Basin Hopping** - Questo è un algoritmo a due fasi che alterna un passo globale stocastico ad una ottimizzazione locale. Ogni iterazione è composta dai seguenti passi:
  1) perturbazione casuale delle coordinate iniziali;
  2) ottimizzazione locale;
  3) accettazione o rifiuto delle nuove coordinate in base al valore della funzione in esse.

  Come algoritmo di ottimizzazione locale, è possibile utilizzare vari metodi. In questo caso, è stato scelto l'algoritmo Limited-memory Broyden–Fletcher–Goldfarb–Shanno vincolato (L-BFGS-B). I vincoli sono stati imposti per la positività dei parametri da ottimizzare.
  Per quanto riguarda il test d'accettazione, invece, è stato usato il criterio di Metropolis degli algoritmi Monte Carlo. Tale criterio accetta il nuovo valore se la funzione nel nuovo punto è migliore di quella al psso precedente, sennò accetta un passo che peggiora la funzione con una certa probabilità p.
  Questo algoritmo esiste anche nella versione per popolazioni di soluzioni, ma non è incluso nella libreria *scipy* di riferimento.
- **Differential Evolution** - Questo è un metodo stocastico basato su popolazioni. Ad ogni passo, l'algoritmo

perturba ogni soluzione candidata in base alle altre soluzioni candidate nella popolazione. Nello specifico, la strategia utilizzata per permutare le soluzioni nell'elaborato è *best1bin*. In questo metodo di generazione viene creata una mutazione della miglior soluzione al passo precedente perturbata in base ad altri due candidati scelti casualmente; successivamente viene creato un vettore di test scegliendo i parametri dalla mutazione o dal vettore dei candidati iniziali in base ad una probabilità con distribuzione binomiale, ma l'ultimo elemento di tale vettore viene sempre scelto dal valore permutato. Successivamente viene testato se i nuovi valori di test sono migliori di quelli all'iterazione precedente e, nel caso, vengono sostituiti ad essi.

- **Dual Annealing** - Questo metodo stocastico è derivato dall'algoritmo *Generalized Simulated Annealing*, che combina una generalizzazione di due tecniche di Simulated Annealing ad una ricerca locale. Anche questo algoritmo sfrutta la differenza di energia tra due punti come Basin Hoppin ma accetta passi che vanno a peggiorare il valore della funzione con una probabilità che decresce all'aumentare del numero di iterazioni.

C. Esperimento

Nell'elaborato, il database è stato diviso in tre sezioni: training set (80% del totale), evaluetion set (10%) e test set (10%). La prima porzione è stata usata per fare il fitting del modello del classificatore SVM. La seconda, invece, è stata usata per valutare l'accuratezza del classificatore, ovvero la funzione da ottimizzare, al variare dei parametri da ottimizzare. Infine, le prestazioni del classificatore con parametri ottimizzati sono state valutate sul test set per valutare l'accuratezza della classificazione su valori nuovi.
Essendo una classificazione multi-classe, il classificatore utilizzato implementa l'approccio "one-against-one": per ogni coppia di classi, viene creato un classificatore diverso e la decisione sulla classe di nuovi valori viene presa aggregando i risultati dei singoli classificatori.
Tutti gli algoritmi usati per l'elaborato sono stati implementati per la minimizzazione di funzioni. Per questo, la funzione passata a tali algoritmi da ottimizzare è stata l'accuratezza cambiata di segno.
Gli algoritmi usati sono stati fatti partire tutti da punti casuali, ma mentre Dual Annealing e Different Evolution generano internamente causalmente i punti da cui partire, per Basin Hoppin è stato necessario passare dei valori iniziali dei parametri da ottimizzare. Per migliorare i risultati di questo algoritmo è stato eseguito a partire da una successione di punti iniziali diversi, in modo da rendere l'algoritmo per popolazioni di soluzioni. Nel caso specifico, i punti di partenza dell'algoritmo sono stati tre.

III. Risultati

Essendo algoritmi stocastici, eseguendoli più volte è possibile ottenere valori diversi dell'ottimo del problema, ma i comportamenti degli algoritmi rimangono pressoché analoghi ad ogni esecuzione.
In tabella I è possibile vedere le accuratezze ottenute dal classificatore sull'insieme di valutazione durante l'addestramento e sull'insieme di test nella run effettuata.
Tutti gli algoritmi riescono a trovare in poco tempo un valore della funzione da massimizzare (accuratezza sull'evaluation set) molto alto sempre pari a 0.9333. In tutti i casi, però, gli algoritmi si fermano in valori dei parametri diversi, visibili in tabella II.
Anche sul test set i risultati sono simili a quelli ottenuti sull'insieme di valutazione. Dai risultati, però, è possibile vedere come i primi due algoritmi sbaglino solo 1/15 delle classificazioni richieste. In particolare, l'unico esempio sbagliato nella classificazione è lo stesso in entrambi i casi. Questo errore avviene sul settimo esempio dell'insieme, che viene assegnato alla classe con indice 2 invece che alla classe con indice 1.
L'unico algoritmo che riesce a trovare dei parametri buoni per riconoscere anche tale esempio è l'algoritmo di dual-annealing che ottiene sull'insieme di test un'accuratezza del 100%.

TABLE I
RISULTATI GLI ALGORITMI

Algoritmo	Accuratezza sull'insieme di valutazione	Accuratezza sull'insieme di test
Basin-Hopping	0.9333	0.9333
Differential Evolution	0.9333	0.9333
Dual Annealing	0.9333	1.0

TABLE II
VALORI OTTIMALI TROVATI

Algoritmo	C	γ
Basin-Hopping	1.0920	0.3434
Differential Evolution	1.3908	2.2481
Dual Annealing	0.1988	0.9719

IV. CONCLUSIONI

Tutti gli algoritmi utilizzati si sono rivelati buoni per risolvere il problema posto. Quello che risulta trovare un ottimo leggermente migliore sembra essere dual annealing, attraverso il quale sono stati trovati valori per C e γ tali da permettere al classificatore di riconoscere meglio gli esempi non visti durante la fase di training. Gli altri due algoritmi trovano dei punti di ottimo che creano un modello di SVM che soffre leggermente di overfitting.