Perceptron vs Decision Tree

Francesco Amorosini

April 13, 2019

1 Introduzione

Perceptron e Decision Tree sono due dei modelli più utilizzati per effettuare machine learning, collocandosi nella categoria dell'apprendimento supervisionato. Questa categoria è basata su agenti che, dopo aver osservato un insieme di coppie input-output, producono una funzione che mappa dai primi ai secondi. Tale funzione potrà poi essere utilizzata per fare previsioni su dati non ancora osservati.
Questo progetto ha lo scopo di confrontare le curve di apprendimento dei due modelli sul problema della classificazione, utlizzando come dataset un insieme di immagini di articoli di moda messo a disposizione da Zalando[1].

2 Dalle Immagini ai Dati in Ingresso

Durante la prima esecuzione del codice verrà utilizzato il package *gitpython* per scaricare direttamente da GitHub[1] i dati e lo script necessario alla loro estrazione, che verranno collocati in un'apposita cartella *data*. L'esecuzione dello script scaricato ritorna due insiemi di coppie input-output, rispettivamente uno per il training dei modelli e l'altro per il test. Il training set e il test set sono composti rispettivamente da 60000 e 10000 immagini in scala di grigio di dimensione 28×28 pixel, che dovranno essere classificati in 10 possibili tipologie di vestiti.

Standardizzazione dei dati. Per quanto detto sopra, ogni immagine da classificare è rappresentata con un vettore di 784 ($28 \times 28 = 784$) valori tra 0 e 255. Utilizzare dati di questo tipo per addestrare un modello può talvolta portare a risultati insoddisfacenti, soprattutto per quei modelli che utilizzano la distanza Euclidea per calcolare la distanza tra due punti[2]. Per questo motivo, prima di effettuare gli esperimenti, sia il training set che il

test set sono stati standardizzati tramite l'oggetto *StandardScaler* presente nella libreria *Sci-Kit Learn*. Dopo la standardizzazione, ogni immagine da classificare verrà rappresentata da un vettore di 784 elementi a media nulla e varianza unitaria.

3 I modelli della libreria Sci-Kit Learn

Perceptron. Perceptron è un classificatore lineare, e nel caso binario funziona tracciando un iperpiano che cerca di separare i punti appartenenti a una classe da quelli che non ne fanno parte. Nel nostro caso, poichè dobbiamo classificare le immagini in 10 classi diverse, verranno generati 10 iperpiani. La classe di appartenenza di un'immagine verrà scelta a seconda di quale piano è "più sicuro" di averla etichettata correttamente. Ogni iperpiano è definito da un vettore $\overrightarrow{w}$, che ne descrive l'orientamento, e da un valore b, che ci permette di traslarlo nello spazio. In fase di training, Perceptron cerca di predire la classe di un'immagine tramite questi iperpiani, correggendo i parametri $\overrightarrow{w}$ e b in caso di errore.
Il Perceptron della libreria *Sci-Kit Learn* è implementato in modo tale da iterare sul training set un numero molto grande di volte (*epochs*, di default 1000), dove la correzione dei parametri degli iperpiani è moltiplicata per un fattore molto piccolo ad ogni iterazione (α, di default 10^{-4}). Per evitare che la fase di train si prolunghi inutilmente, è possibile specificare una soglia minima di correzione (*tol*, di default 10^{-3}): se la correzione dei parametri è più piccola di *tol* si assume che il modello sia arrivato a convergenza, e di conseguenza si interrompe il training.

Decision Tree. Decision Tree è un metodo di apprendimento supervisionato non parametrico che funziona separando i dati in subsets via via sempre più piccoli e omogenei dal punto di vista delle categorie presenti. Ogni volta che i dati vengono separati secondo un certo criterio (viene effettuata una decisione), tale decisione viene aggiunta all'interno di una struttura ad albero. Un Decision Tree ottimale riesce a classificare dati correttamente utilizzando il minor numero possibile di decisioni. Un albero con troppi nodi, infatti, è un modello eccessivamente complesso che predice perfettamente ogni immagine del training set ma ottiene scarsi risultati sui dati di test (overfitting). Per questo motivo dobbiamo scegliere una misura che per ogni subset ci permetta sia di individuare la migliore separazione possibile, sia di capire quando non è più conveniente continuare a generare ulteriori subset (pruning).

Nel Decision Tree della libreria *Sci-Kit Learn* la misura che viene utilizzata di default è l'indice di impurità di Gini. Tramite quest'ultima è bene specificare il parametro *min_impurity_decrease* come criterio di arresto: se la differenza di impurità a seguito di una decisione è minore di tale parametro non verranno effettuate ulteriori separazioni in quel subset.

4 Esperimenti

Dopo aver preprocessato i dati come descritto nel paragrafo 2, procediamo al tracciamento delle cruve di apprendimento di Perceptron e Decision Tree tramite i seguenti passaggi:

1. Tramite la libreria *Sci-Kit Learn* si instanziano i modelli desiderati. In questa fase l'utente è tenuto a specificare i macroparametri che ritiene più appropriati per ogni modello. Nel nostro caso abbiamo scelto una $tol = 10^{-4}$ per Perceptron e $min\_impurity\_decrease = 10^{-3}$ per il Decision Tree, lasciando gli altri parametri invariati.

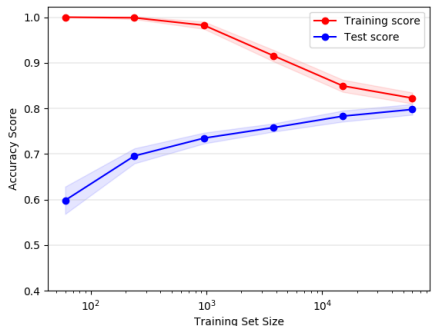
2. Viene specificato il numero di iterazioni per ogni esperimento, nel nostro caso 30. É molto importante randomizzare l'ordinamento del training set tra un'iterazione e l'altra: questo passaggio infatti previene l'introduzione di un bias dovuto all'ordinamento dei dati.

3. Per ogni iterazione viene invocata la funzione *learning_curve* presente nella libreria *Sci-Kit Learn*. Tale funzione prende in entrata i seguenti oggetti:
   - Il modello di cui tracciare la curva;
   - I dati su cui addestrare e testare il modello;
   - Un criterio secondo cui effettuare lo split train/test dei dati. Poichè i nostri dati sono stati estratti già suddivisi in training set e test set, abbiamo utilizzato un oggetto *PredefinedSplit* (anch'esso presente nella libreria) per descrivere la suddivisione dei dati;
   - Una lista contenente le dimensioni di training set su cui effettuare l'esperimento (*training sizes*). Nel nostro caso abbiamo utilizzato per ogni esperimento 6 training sizes (che corrisponderanno ai 6 punti della curva) la cui distanza cresce in modo esponenziale;
   - Una metrica con cui misurare le prestazioni del modello, nel nostro caso l'*accuracy* (cioè il rapporto tra le immagini classificate correttamente e le immagini totali);

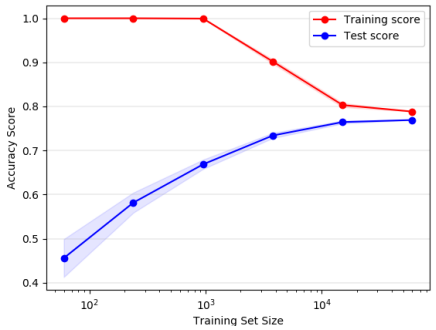
La funzione ritorna la lista di training sizes e due vettori (uno per il training set e uno per il test set) contenenti la misura delle accuratezze per ogni training size.

4. Al termine dell'esperimento avremo un numero di coppie di vettori pari al numero di iterazioni effettuate: da questi calcoliamo altri due vettori contenenti le medie aritmetiche e le deviazioni standard delle accuratezze ottenute per ogni training size, e utilizziamo la libreria *matplotlib* per tracciare le curve con i risultati ottenuti.

5 Analisi dei risultati



(a) Curva di apprendimento di Perceptron



(b) Curva di apprendimento di Decision Tree

Dai risultati ottenuti possiamo trarre le seguenti conclusioni: Nel primo tratto di curva possiamo notare un *training score* vicino ad 1 e un *test score* piuttosto basso con una varianza molto alta. Questo è un chiaro segno di overfitting, dovuto al fatto che l'addestramento dei modelli con le specifiche scelte porta alla modellazione del rumore quando si hanno pochi esempi di train. La varianza in particolare è dovuta al fatto che, quando si addestra un modello con pochi esempi, le sue prestazioni dipendono fortemente da quali esempi vengono selezionati dall'intero training set. Per questo motivo in generale la varianza va riducendosi con l'aumento degli esempi di train utilizzati. La varianza di Perceptron tuttavia non sembra diminuire particolarmente, ed anzi è presente anche nel *training score*. Questo è dovuto al fatto che Perceptron è un modello lineare, ma un dataset in cui il numero di campioni è molto maggiore del numero di attributi per campione (784) ha poche possibilità di essere linearmente separabile.

References

[1] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017. Also available as: https://github.com/zalandoresearch/fashion-mnist.

[2] Sudharsan AsaithambiFollow. Why, how and when to scale your features, 2017. https://medium.com/greyatom/why-how-and-when-to-scale-your-features-4b30ab09db5e,.