Fondamenti di Machine Learning

Laurea Triennale in Ingegneria delle Comunicazioni

4: k-nearest neighbors ed overfitting

Lecturer: S. Scardapane

SAPIENZA
UNIVERSITÀ DI ROMA

Instance-based learning

k-nearest neighbors

Nel caso del LS costruiamo un modello *globale* per tutto lo spazio $\mathbb{R}^d$. Quindi, modificare anche un singolo esempio di training (es., aggiungendo un **outlier**) influenza tutte le predizioni.

In un modello *locale*, invece, le predizioni sono influenzate solo dai punti di training ad esso vicine (**instance-based learning**).

Qui di seguito vediamo l'implementazione più semplice di questa idea, chiamata k-Nearest Neighbours (k-NN), che predice la media (o la mediana) dei k punti più simili.

Denotiamo con $\mathcal{N}_k(\mathbf{x})$ l'insieme degli indici dei k punti più simili ad $\mathbf{x}$ nel training set $\mathcal{S}$, ad esempio con la distanza Euclidea:

$$d(\mathbf{x}, \mathbf{z}) = \sqrt{\sum_i (x_i - z_i)^2}\,. \tag{1}$$

Dato un punto $\mathbf{x}$, il k-NN predice la media calcolata in $\mathcal{N}_k(\mathbf{x})$:

$$f(\mathbf{x}) = \frac{1}{k} \sum_{i \in \mathcal{N}_k(\mathbf{x})} y_i \tag{2}$$

Nel caso di problemi di classificazione è possibile sostituire la media con un voto di maggioranza (mediana).

Nel k-NN non è necessario allenare nulla, ma in fase di predizione è necessario avere l'intero training set in memoria, ed una procedura efficiente per calcolare i k valori più vicini. Questo può essere sconveniente per dataset molto grandi.

Il k-NN viene anche detto un modello **non parametrico**, nel senso che la sua complessità varia al variare del numero dei punti (intuitivamente, ogni esempio aggiunto al training set rende la decision boundary leggermente più complessa).

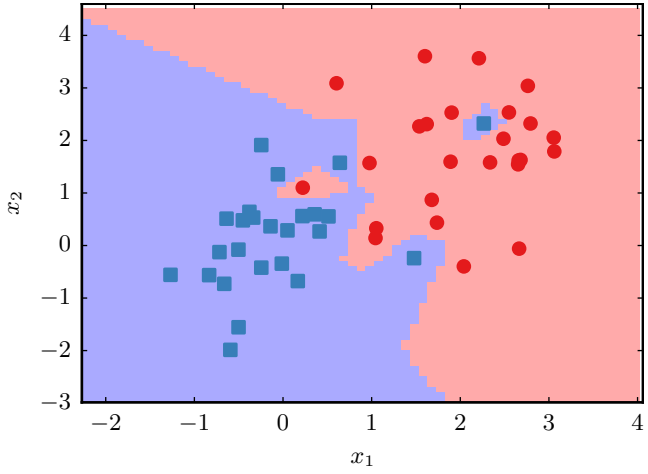

Figure 1: Decision boundary del k-NN con $k = 1$.

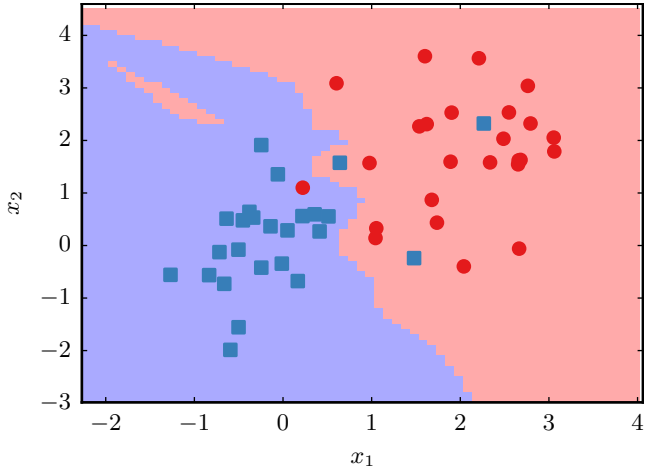

Figure 2: Decision boundary del k-NN con $k = 10$.

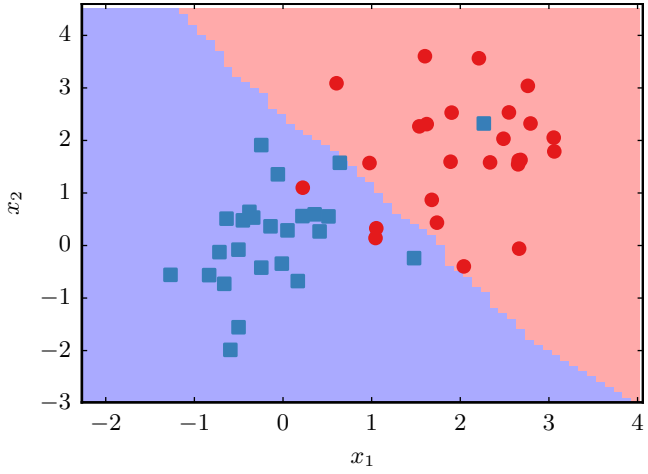

Figure 3: Decision boundary del k-NN con $k = 25$.

Possiamo generalizzare il k-NN supponendo di pesare diversamente i contributi da punti vicini o lontani:

$$f(\mathbf{x}) = \sum_{i \in \mathcal{N}_k(\mathbf{x})} d_i y_i \quad \text{con } d_i = \frac{d(\mathbf{x}, \mathbf{x}_i)}{\sum_{j \in \mathcal{N}_k(\mathbf{x})} d(\mathbf{x}, \mathbf{x}_i)}\,.$$

In questa forma, il k-NN viene detto uno stimatore di Nadaraya-Watson.

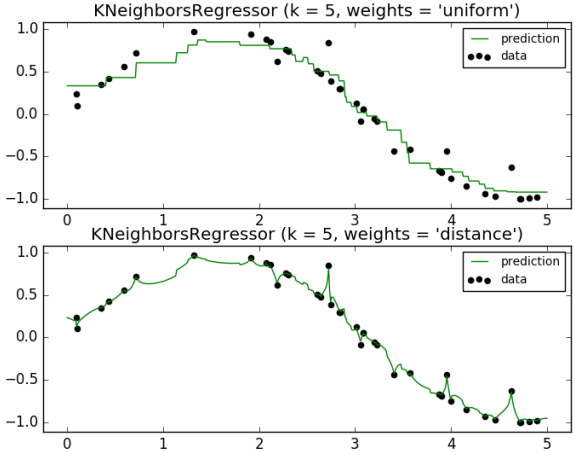

Figure 4: Image source: Nearest neighbors in scikit-learn

In alternativa alla distanza Euclidea possiamo considerare la cosiddetta distanza di **Manhattan** (o **taxicab**):

$$d(\mathbf{x}, \mathbf{z}) = \sum_{i=1}^{d} |x_i - z_i| \,.$$

I parametri di configurazione di un algoritmo (es., k nel k-NN, la funzione di distanza da usare, il learning rate) vengono detti gli **iper-parametri** del modello. Vedremo più avanti tecniche per la loro selezione.

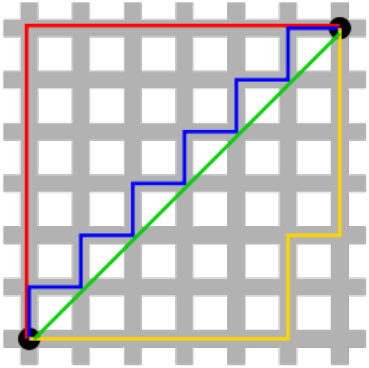

Figure 5: Visualizzazione della distanza di Manhattan [Wikipedia].

Topic aggiuntivi

Overfitting e complessità

Come già accennato, un modo di valutare le performance di un modello è tenere un dataset indipendente di valutazione (**holdout method**):

- Un **training set** per scegliere i parametri del nostro modello.
- Un **test set** (indipendente) per valutarne l'errore.

Vedremo più avanti tecniche (e metriche) più sofisticate per valutare gli algoritmi di apprendimento. In un problema di classificazione, se la proporzione di classi è mantenuta in entrambi i dataset, parliamo di **holdout stratificato**.

Per definizione, 1-NN ha sempre *zero* errore sul dataset di training. Le sue performance sul dataset di test sono però nettamente inferiori: in questo caso parliamo di **overfitting**.

All'opposto, il 25-NN ha errori su training e test simili, ma superiori a quelli del 10-NN. In questo caso parliamo invece di **underfitting** dell'algoritmo.

In generale, l'apprendimento supervisionato è molto diverso dalla pura *memorizzazione* degli esempi di training, in quanto ci interessa estrapolare (generalizzare) su dati futuri.

Il parametro k nel k-NN determina la **complessità** del modello: variandolo, possiamo passare da un modello molto semplice (costante su tutto il dataset) ad un dataset altamente non-lineare. In molti casi, l'overfitting dipende dalla scelta di una complessità troppo alta rispetto al dataset.

Più in generale, qualsiasi parametro che può essere scelto dall'utente (e non dalla procedura di apprendimento) viene detto un **iper-parametro**. La scelta degli iper-parametri ottimi per un problema è detto **model selection**.

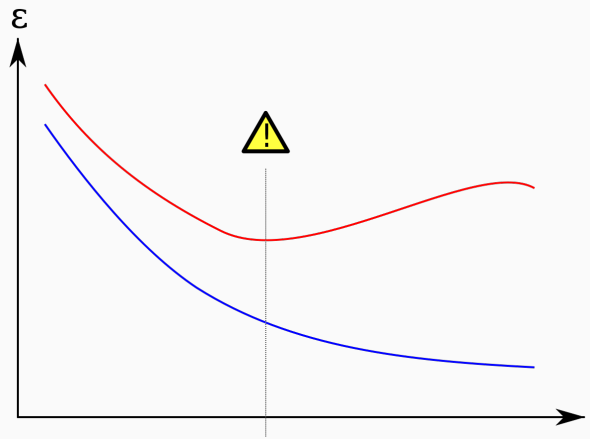

Figure 6: Da Wikipedia: errore (asse *y*) vs. una misura di complessità (asse *x*). Errore di training e test sono blu e rosso, rispettivamente.

Topic aggiuntivi

Bias variance decomposition

Diversi ragionamenti che abbiamo fatto in precedenza (es., sull'overfitting) possono essere formalizzati con la cosiddetta **bias-variance decomposition**. La sua derivazione è leggermente più tecnica di quanto visto finora, ma richiede solo di conoscere alcune proprietà del valore atteso:

$$\mathbb{E}[x + y] = \mathbb{E}[x] + \mathbb{E}[y] \qquad \texttt{Linearità} \tag{3}$$

$$\mathbb{E}[xy] = \mathbb{E}[x]\mathbb{E}[y] \text{ se } p(x, y) = p(x)p(y) \qquad \texttt{Indipendenza} \tag{4}$$

Nel caso più semplice, assumiamo che i nostri dati siano generati come segue:

$$y = g(\mathbf{x}) + \varepsilon\,, \tag{5}$$

dove $g(\cdot)$ è deterministica (ma sconosciuta), $\varepsilon \sim \mathcal{N}(0, \sigma^2)$, e quindi:

$$p(y \mid \mathbf{x}) = \mathcal{N}(y; g(\mathbf{x}), \sigma^2)\,. \tag{6}$$

Cerchiamo di approssimare questo dataset con una qualche $f(\mathbf{x})$ (senza assunzioni), minimizzando la squared loss $(y - f(\mathbf{x}))^2$.

Siamo interessati a capire come si distribuisce tale errore quadratico medio:

$$\mathbb{E}[(y - f(\mathbf{x}))^2] = \mathbb{E}[y^2] + \mathbb{E}[f(\mathbf{x})^2] - 2\mathbb{E}[yf(\mathbf{x})] \tag{7}$$

dove (informalmente) il valore atteso è calcolato rispetto ad ogni possibile scelta di dataset di training. Ricordando $\text{Var}(x) = \mathbb{E}[x^2] - \mathbb{E}[x]^2$ possiamo riscrivere il secondo termine come:

$$\mathbb{E}[f(\mathbf{x})^2] = \text{Var}(f(\mathbf{x})) + \mathbb{E}[f(\mathbf{x})]^2 \tag{8}$$

Il primo termine può essere semplificato date le nostre assunzioni sui dati:

$$\mathbb{E}[y^2] = \mathbb{E}[(g(\mathbf{x}) + \varepsilon)^2] = \mathbb{E}[g(\mathbf{x})^2] + \mathbb{E}[\varepsilon^2] + 2\mathbb{E}[\varepsilon g(\mathbf{x})] \tag{9}$$

Da cui:

$$\mathbb{E}[g(\mathbf{x})^2] = g(\mathbf{x})^2 \qquad g \texttt{ non dipende dal dataset} \tag{10}$$

$$\mathbb{E}[\varepsilon^2] = \sigma^2 \qquad \texttt{Definizione di una Gaussiana} \tag{11}$$

$$2\mathbb{E}[\varepsilon g(\mathbf{x})] = 2\mathbb{E}[g(\mathbf{x})]\mathbb{E}[\varepsilon] = 0 \qquad \texttt{Indipendenza} \tag{12}$$

Rimane l'ultimo termine:

$$2\mathbb{E}[yf(\mathbf{x})] = 2\mathbb{E}[(g(\mathbf{x}) + \varepsilon)f(\mathbf{x})] = 2\mathbb{E}[g(\mathbf{x})f(\mathbf{x})] + \underbrace{2\mathbb{E}[\varepsilon f(\mathbf{x})]}_{=0} \tag{13}$$

Ed infine, sempre per l'indipendenza delle due variabili:

$$2\mathbb{E}[g(\mathbf{x})f(\mathbf{x})] = 2g(\mathbf{x})\mathbb{E}[f(\mathbf{x})] \tag{14}$$

Rimettendo tutto insieme:

$$\mathbb{E}[(y - f(\mathbf{x}))^2] = \mathbb{E}[f(\mathbf{x})]^2 + g(\mathbf{x})^2 - 2g(\mathbf{x})\mathbb{E}[f(\mathbf{x})] + \text{Var}(f(\mathbf{x})) + \sigma^2 \tag{15}$$

Definendo:

$$\text{Bias}(f(\mathbf{x})) = \mathbb{E}[f(\mathbf{x})] - g(\mathbf{x}) \tag{16}$$

Otteniamo:

$$\mathbb{E}[(y - f(\mathbf{x}))^2] = \text{Bias}(f(\mathbf{x}))^2 + \text{Var}(f(\mathbf{x})) + \sigma^2 \tag{17}$$

Vediamo che l'errore quadratico medio si decompone su tre assi:

1. Il **bias** descrive l'errore medio compiuto da f, mediato su tutti i possibili dataset. Ad esempio, se $g(x)$ è lineare, la soluzione data dal least-squares ha bias 0.
2. La **varianza** descrive quanto l'errore varia per un *singolo* dataset.
3. Il **rumore** descrive la componente della loss che non può essere ridotta in quanto dipende dalla aleatorietà dei dati stessi.

Ritornando al k-NN, per k piccoli abbiamo un bias basso ma una varianza molto alta, mentre per k larghi abbiamo un bias molto grande ma una varianza molto piccola. Il giusto trade-off si ottiene per una scelta di k che bilancia questi due fattori.

Topic aggiuntivi

Curse of dimensionality

Finora abbiamo considerato problemi relativamente semplici, con una dimensionalità d bassa. Non tutti gli algoritmi, però, scalano ugualmente bene all'aumentare della dimensionalità.

Ad esempio, consideriamo questa semplice variante del k-NN: dividiamo lo spazio Euclideo in celle di dimensioni fissa; per ogni cella prendiamo tutti gli elementi del training set che vi cadono dentro; assegniamo un voto di maggioranza *all'intera cella* sulla base di questi punti.

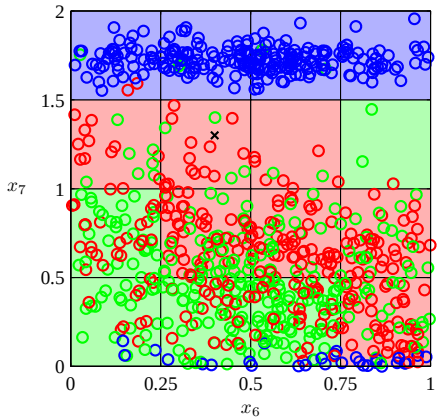

Figure 7: Semplificazione del *k*-NN in cui scegliamo una predizione per ogni cella in cui viene ripartito lo spazio Euclideo (riprodotto da Bishop e Bishop, 2023).

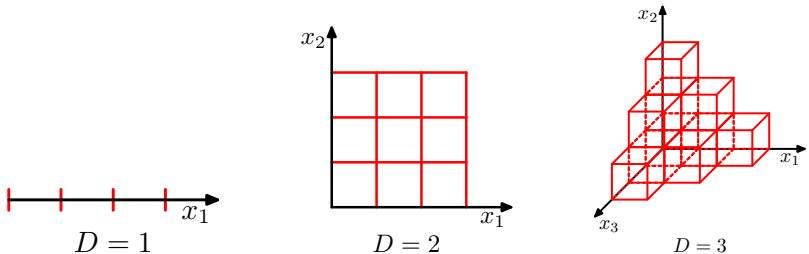

Figure 8: Il numero di elementi necessari a riempire lo spazio mantenendo la stessa densità cresce esponenzialmente con la dimensione dello spazio (riprodotto da Bishop e Bishop, 2023).

Si usa il termine *curse of dimensionality* per riferirsi a tutte quelle proprietà che dipendono in maniera esponenziale dalla dimensionalità dello spazio (es., i dati necessari a campionare lo spazio nell'esempio precedente).

Come altro esempio: in uno spazio ad n dimensioni, il volume di una sfera di raggio r si può scrivere come $V(r) = k_n r^n$, dove k_n è una costante che dipende da n. La frazione di volume contenuta tra $r - \epsilon$ ed r è quindi:

$$\frac{V(r) - V(r - \epsilon)}{V(r)} = r - (r - \epsilon)^n \tag{18}$$

Per grandi n, la maggior parte del volume di una sfera è concentrato sulla superficie!

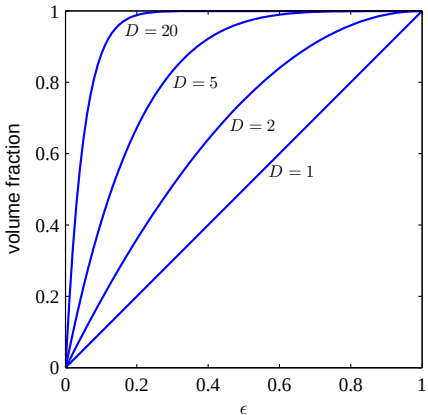

Figure 9: Grafico della frazione $\frac{V(r)-V(r-\epsilon)}{V(r)}$ per vari n, supponendo $r = 1$ (riprodotto da Bishop e Bishop, 2023).