Introduzione al Machine Learning

4 febbraio 2026

Indice

Capitolo 1

Supervised

1.1 Introduzione

1.2 Classificatori

1.2.1 K-nn

1.2.2 Perceptron

1.2.3 SVM

1.2.4 Logistic regression

1.3 Regressioni

1.3.1 Linear regression

1.3.2 Ridge regression

1.3.3 Poisson regression

1.3.4 Generalized Linear Model

1.4 Modelli bayesiani

1.5 guida pratica all'ottimizzazione del machine learning

1.5.1 gradiend descend

1.5.2 adagrad

1.5.3 Loss

- koulback-Leibler divergens

Capitolo 2

Un-supervised

Se nel supervised learning il nostro obbiettivo è quello di trovare una funzione che leghi i miei dati di input con quelli si output, predicendo di fatto il risultato, l'obbiettivo del unsupervised learning è di scoprire quello che potrebbe essere un risultato. Parlando di dati e label, l'obbiettivo per il primo è di riuscire a predirre la label, per il secondo è di scoprire quali sono le possibili label dei miei dati, senza avere un idea di quali sono realmente.

2.1 Manifold Learning

Per tutti gli algoritmi successivi, l'idea è quella di trovare una struttura tra i dati, che ci permetta di rappresentarli con meno feature. In questa maniera possiamo renderli più comprensibili all'umano o più facili da manipolare per un altro algoritmo.

2.1.1 Intrinsinc dimension estimation

2.1.2 Principal Component Analysis

Immaginniamo di avere un dataset $\{x_i\}_{i=1}^n \in \mathbb{R}^d$ dove ogni x_i viene generato da un vettore u con la relazione $x_i = \alpha_i u$. Allora ogni nostro punto del dataset può essere riassunto con il solo parametro α_i. In questo caso per quanto possa essere alta la dimesione dello spazio $\mathbb{R}^b$, i nostri dati vivono in uno spazio 1-dimensionale. La stessa cosa avverrebbe anche se aggiungessimo del piccolo rumore ai dati che porterebbe a vedere una nuvola più che una linea retta nello spazio.

Il primo importante passo da fare è sottrarre ad ogni dato la media del dataset, questo ci serve perché se i nostri dati sono tutti concentrati lontano dall'origine, potremmo riassumerli tutti con il loro punto medio. Come se fossimo sulla chiama di una montagna e guardassimo delle persone nella piazza del paese. Potremmo dire che con buona approssimazione sono tutte vicino alla fontana, se invece ci troviamo in piazza vediamo come un gruppo è accanto al lavatoio a fare il bucato mentre l'altro è attorno alla bottega del falegname.
La seconda operazione da fare è dividere ogni feature per la sua varianza. Questo ci serve per evitare che feature molto grandi influenzino erroneamente le nostre analisi. Se ci sono due file per il rancio nella legione, e queste linee si presentassero più lunghe del normale, il modo più immediato che abbiamo per identificare una persona è vedere quanto questa sia distante dalla pentola, ma questo non ci permetterebbe di più di osservare in quale fila era prima. Se invece guardiamo bene le file, faremmo bene a dire da che parte della staccionata si trova ogni persona.

Sia $\mu = \frac{1}{n} \sum_i x_i$ la media dei nostri dati e $\sigma =$

Ci aspettiamo che se potessimo vedere i dati nella loro interezza, possiamo notare come questi possano essere disegnati come un iperpiano in quello spazio alto dimensionale.
Per identificare quale sia questo iperpiano, vogliamo vedere se ci siano due feature correlate. Così se varia una allora varia anche la seconda e quindi possiamo eliminare una delle due perché non aggiunge informazione.

PCA diventa utile se viene applicato su dataset normalizzati.

scree test: serve per valutare quali autovalori siano i più significativi da tenere.

Se plotto i miei autovalori e vedo in un punto un salto del loro valore, prendo tutti gli autovalori precedenti al salto che spiegheranno la maggior parte della varianza dei dati.

Percentuale di Explaned Variance $\frac{\sum_{j=0}^{d} \lambda_j}{\sum_{i=0}^{D} \lambda_i}$

2.1.3 K-PCA

2.1.4 t-SNE

2.1.5 UMAP

2.1.6 Distanze

Le distanze ci aiutano a valutare diversi algoritmi. Si vede come al variare della distanza usata, anche l'algoritmo che la utilizza varia il risultato

- Distanza di Minowski
- Coeficiente di correlazione di Pearson
- Distanza di Mahalanobis: invariante per trasformazioni lineari delle coordinate
- Similarità del coseno
- Jaccard Distance
- Distanza di hamming
- Distanza geodesica

2.1.7 Multidinensional Scaling

Vogliamo proiettare i dati in uno spazio a dimensionalità più bassa, mantendendo le distanze. Fondamentale è lo stress che misura quale sia la differenza tra le distanze prima e dopo la proiezione dei dati. Ovviamente l'obbiettivo dell'algoritmo è minimizzare lo stress riducendo la dimensionalità dei dati.

2.1.8 ISOMAP

ISOMAP è un algoritmo che costruisce un grafo dei miei dati tramite knn. Quindi se trovo due punti molto vicini questi saranno connessi.
Per valuta la distanza tra due punti, vedo quanti archi devo percorrere al'interno del grafo per passare da un punto all'altro. Questa è chiamata distanza geodesica. Quindi il grafo che ho creato è la mia rappresentazione a dimensionalità ridotta dei miei dati e la distanza geodesica viene mantenuta

2.2 Densinity estimation

2.2.1 Histograms

2.2.2 Kernel density estimation

L'idea è quella di stimare la distribuzione di probabilità del nostro dataset. L'approccio che vediamo è quello di porre intorno a ogni punto una gausiana e poi sommarle tra di loro

$$P(X) = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{h\sqrt{2\pi}} e^{-\left(\frac{x-x_i}{h}\right)^2}$$

$h = n^{-\frac{1}{4}+d}$

2.2.3 K-nn density estimator

Vogliamo valutare la nostra funzione di densità a partire dai vicini di un punto dato. Quindi deciso un K, valuto r_k come la palla che comprende il k-esimo vicino al punto. Si intende che la palla è della dimensione dei dati.
Stimiamo quindi la nostra densità come:

$$P(X) = \frac{K}{nr_x^d}$$

Il problema di questo metodo è che soffre moltissimo della course of dimensionality che ci porta, ad alte dimensionalità ad avere palle di dimensioni molto simili tra loro.

2.3 Clustering

L'obbiettivo è di ragruppare punti assieme secondo un criterio

2.3.1 Classical clustering

K-means clustering

A priori, decidiamo che secondo noi nel nostro dataset ci sono k cluster. Quindi creiamo k punti casuali nel nostro spazio che vengono chiamati centroidi. Adesso assegno ogni punti al centroide più vicino e creo una prima ipotesi di cluster. Quindi ricalcolo i centroidi come la media di ogni cluster.

problemi: sensibile all'inizializzazione -¿ k-means l'utente deve fornire k -¿ scree test. guardo il grafico e vedo dove sta il gomito sensibile agli outliars -¿ k-medioids funziona solo per cluster sferici -¿ Kernel k-means

fuzzy k-means

Invece di assegnare direttamente un dato a un certo cluster, salvo per ogni dato la probabilità che appartenenza a ogni cluster. In questa maniera possiamo avere un risultato pi+ interpretabile e vedere quanto possiamo fidarci del risultato. la loss è data da $\lessdot(\mathbb{U}) = \sum_{l=1}^{k}\sum_{i=1}^{n} u_{il}$

- 1. Il Vincolo di Appartenenza Per ogni punto dati i, la somma dei suoi gradi di appartenenza a tutti i k cluster deve essere pari a 1:

  $$\sum_{l=1}^{k} u_{il} = 1$$

  .

- 2. La Funzione Obiettivo (Loss) L'obiettivo dell'algoritmo è minimizzare una funzione di costo che pesa le distanze euclidee tra i punti x_i e i centri dei cluster c_l in base al grado di appartenenza:

  $$O(\mathbb{U}) = \sum_{l=1}^{k}\sum_{i=1}^{n} u_{il}^m \|x_i - c_l\|^2$$

  .

  In questa formula: * **n** è il numero totale di punti. * **m** è il **parametro di "fuzzificazione"** (tipicamente $m = 2$). Questo valore controlla quanto deve essere "sfumata" l'assegnazione: se m tende a infinito, l'algoritmo tende a comportarsi come il k-means rigido.

- 3. Regole di Aggiornamento (L'Algoritmo) L'algoritmo procede iterativamente partendo da un'inizializzazione casuale della matrice di appartenenza $\mathbb{U}$ e aggiornando alternativamente i centri e i gradi di appartenenza:

* **Aggiornamento dei Centri (c_l):** Ogni centro viene calcolato come una media di tutti i punti del dataset, pesata per il loro grado di appartenenza elevato a m:

$$c_l = \frac{\sum_{i=1}^{n} u_{il}^m x_i}{\sum_{i=1}^{n} u_{il}^m}$$

.

* **Aggiornamento delle Appartenenze (u_{ij}):** Il nuovo grado di appartenenza di un punto i al cluster j viene ricalcolato in base alla distanza relativa rispetto a tutti gli altri centri:

$$u_{ij} = \frac{1}{\sum_{l=1}^{k} \left(\frac{\|x_i - c_j\|}{\|x_i - c_l\|} \right)^{2/(m-1)}}$$

.

Il processo si ripete finché la variazione della matrice $\mathbb{U}$ tra due iterazioni non scende al di sotto di una soglia prefissata (convergenza).

- Vantaggi e Limitazioni * **Flessibilità:** È particolarmente utile per dati che si trovano "a metà strada" tra due centri, poiché permette di esprimere l'incertezza dell'assegnazione. * **Problemi ereditati dal k-means:** Rimane sensibile alla scelta iniziale dei parametri, al numero di cluster k scelto a priori e alla presenza di outlier, che possono spostare i centroidi in modo significativo.

k-medioids

uso i punti del dataset come centroidi

k-means ++

Scelgo random un punto del dataset. Tutti gli altri punti del dataset potrebbero diventare dei nuovi centroidi con probabilità proporzionale a $P \propto (\min ||X - X_i||_2)^2$ dove X è il punto che stiamo considerando e X_i sono tutti i centroidi già selezionati.

Kernel k-means

Hierarchical Clustering

Il **clustering gerarchico** è una tecnica di apprendimento non supervisionato che organizza i dati in una struttura a livelli, producendo rappresentazioni in cui i cluster a ogni livello sono creati fondendo (o dividendo) i gruppi del livello precedente. A differenza del k-means, non richiede di specificare a priori il numero di cluster e il risultato viene solitamente visualizzato tramite un **dendrogramma**, un albero binario dove l'altezza dei nodi è proporzionale alla dissimilarità tra i gruppi fusi.

Esistono due approcci principali: * **Agglomerativo (Bottom-up):** Si parte con ogni punto come un singolo cluster e, a ogni passaggio, si fondono i due cluster più vicini fino a ottenerne uno solo che comprende tutto il dataset. È il metodo più utilizzato per via della sua minore complessità computazionale rispetto al divisivo. * **Divisivo (Top-down):** Si parte da un unico cluster globale e lo si divide ricorsivamente in sotto-gruppi sempre più piccoli.

- Criteri di Collegamento (Linkage) Il funzionamento dell'algoritmo dipende da come viene definita la distanza tra due gruppi di osservazioni (G e H). Le formule principali sono:

  1. **Single Linkage (Nearest Neighbor):** La distanza tra due cluster come la minima distanza tra i due punti più vicini dei due cluster:

$$d_{SL}(G, H) = \min_{i \in G, i' \in H} d_{ii'}$$

  Questo metodo può gestire forme non ellittiche ma è sensibile al rumore e tende a creare cluster allungati (fenomeno del *chaining*).

2. **Complete Linkage (Furthest Neighbor):** La distanza è definita la massima distanza tra i due punti più vicini dei due cluster:

$$d_{CL}(G,H) = \max_{i\in G, i'\in H} d_{ii'}$$

Tende a produrre cluster più compatti e bilanciati, ma può forzare strutture che non esistono nei dati.

3. **Group Average Linkage:** Utilizza la distanza media tra tutti i punti dei due cluster: valuto la distanza come la distanza tra i centroidi dei vari clusters. In questo caso un problema che occorre è che, se in tutti gli altri casi la distanza tra cluster può solo aumentare, dato che il nuovo centroiede prima non esisteva, è possibile che questo compaia più vicino di prima. E questo ad Alex non piace

$$d_{GA}(G,H) = \frac{1}{N_G N_H}\sum_{i\in G}\sum_{i'\in H} d_{ii'}$$

Rappresenta un compromesso tra i due estremi precedenti.

4. **Distanza di Ward:** Considera l'aumento della varianza totale all'interno dei cluster (somma dei quadrati) derivante dalla loro fusione: La distanza tra due cluster è valutata come la distanza tra i due centroidi moltiplicata per il fattore di Ward. La potenzialità è che noi possiamo calcolare la distanza tra i cluster in maniera ricorsiva. Inoltre i coefficienti permettono alle distanze di essere crescenti e quindi di rispettare l'assunto di base del clustering gerarchico.

$$D(C_i, C_j) = \frac{2n_i n_j}{n_i + n_j} D(r_i - r_j)^2$$

$$D(C_{i\cup j}, C_k) = \frac{n_i n_k}{n_i + n_j + n_k} D(C_i, C_k) + \frac{n_j n_k}{n_i + n_j + n_k} D(C_j, C_k) - \frac{n_k}{n_i + n_j + n_k} D(C_i, C_j)$$

Dove n_i n_j sono i numeri di punti dei due cluster e r_i e r_j sono i centroidi. È spesso preferito perché meno suscettibile agli outlier e tende a creare cluster di forma sferica.

- Validazione Per valutare quanto la struttura gerarchica del dendrogramma rappresenti fedelmente le distanze originali tra i dati, si utilizza spesso il **coefficiente di correlazione cofenetica**, che misura la correlazione tra le distanze originali e quelle di fusione riportate nell'albero.

2.3.2 Modern clustering

Gaussian Mixtures

Partiamo dall'idea del Fuzzy, vogliamo quindi creare una matrice che mi lega i dati e quanto è probabile che appartengano a un certo cluster. Nel nostro caso i clusters sono rappresentati da gausiane di media e varianza ignote. Per trovare un fit adatto delle nostre gausiane Partiamo da una stima di quanti punti appartengono ad ogni cluster.

$$N_k =$$

DBSCAN

Per prima cosa definiamo due parametri del nostro clustering: ϵ, che indica il raggio che andremo ad utilizzare, e $MinPts$ che indica il minimo dei punti che accettiamo nella nostra frontiera.

Definiamo:

- Core_point: un punto che nel suo vicinato, entro ϵ di distanza, ha più di $MinPts$ punti
- Border_point: un punto che nel suo vicinato ha almeno un Core point.

Algoritmo:

-

Density Peaks

Capitolo 3

Reinforcement Learning

Ricordate che KNN è una felce. Adesso la usiamo per pulirci il culo ma se non ci fosse stata lei non esisterebbe nulla di quello che conosciamo.