

07/03/2023

INTRODUZIONE AL MACHINE LEARNING

Dato un DATA SET (training set), si utilizza un algoritmo^A che "costruisce" un altro algoritmo B a partire dal data set che attira lo scopo di effettuare la predizione / classificazione di istanze che poi verranno date in input (citando da lezione di classificazione), ovvero lo scopo di risolvere un dato TASK di machine learning.

IL MACHINE LEARNING UTILIZZA UN APPROCCIO INATTIVITIVO, OVVERO SI BASA SU ESAMPLES / CS1 PARTICOLARI PER DERIVARE UN CRITERIO GENERALE PER EFFETTUARE LA CLASSIFICAZIONE

CLASSIFICAZIONE → BINARIA (yes/no)

→ MULTI-CLASSE (il valore da predire appartiene a un insieme di cardinalità > 2)

10/03/2023

→ REGRESSIONE: consiste nella predizione di un valore reale.

→ CLASSIFICAZIONE: consiste nella predizione di un valore discreto appartenente a un insieme piccolo (il caso limite è la CLASSIFICAZIONE BINARIA).

APPRENDIMENTO SUPERVISO: è l'apprendimento che avviene sulla base di esempi (\rightarrow training set).

APPRENDIMENTO NON SUPERVISO: è l'apprendimento su come raggruppare gli elementi (ma non porta a fare una classificazione secca).

REINFORCEMENT LEARNING: è l'apprendimento delle decisioni da prendere (insieme di passi da seguire) sulla base di determinati fattori dipendenti dal tempo (vedi i sistemi autonomi che giocano a scacchi).

I predittori possono essere DETERMINISTICI o PROBABILISTICI.

Dico: "Quest'elemento appartiene alla classe X₀"

Dico: "Quest'elemento appartiene alla classe X₀ con probabilità p₀, alla classe X₁ con probabilità p₁, ecc..."

Ci piacciono di più i predittori probabilistici anche perché danno più informazioni (e non sempre la classe predetta deve corrispondere a quella con la probabilità maggiore; dipende dal contesto). → VEDERE IL CASO IN CUI DARE UN FALSO NEGATIVO È PIÙ GRAVE DI DARE UN FALSO POSITIVO.

GENERALIZZAZIONE: vogliamo che il nostro algoritmo predica bene nuovi dati sulla base dei dati che già abbiamo. Se invece l'algoritmo si limita a predire bene i dati che già abbiamo, è un algoritmo troppo specializzato e rischia di ciuccare i nuovi dati (fenomeno dell'OVERTFITTING).

MATRICE DI CONFUSIONE: matrice $n \times n$ (dove n è il numero di classi) dove gli elementi sulla diagonale indicano quante volte sono stati classificati correttamente gli elementi per ogni classe,

mentre gli elementi fuori dalla diagonale indicano quante volte sono stati classificati ma \neq la classe (e.g. riga i col. j indica "ho predetto \hat{c}_j quando in realtà era classe c_i ").

INDICI CHE INDICANO QUANTO È BUONO UN CLASSIFICATORE:

- ACCURACY = $\frac{\# \text{ elementi classificati correttamente}}{\# \text{ elementi totali}}$
- PRECISION per la classe i = $\frac{\# \text{ elementi della classe } i \text{ classificati correttamente}}{\# \text{ elementi classificati come appartenenti alla classe } i}$
- RECALL per la classe i = $\frac{\# \text{ elementi della classe } i \text{ classificati correttamente}}{\# \text{ elementi della classe } i}$
- F-SCORE per la classe i = $\frac{\text{precision per la classe } i \times \text{recall per la classe } i}{\text{precision per la classe } i + \text{recall per la classe } i}$ = MEDIA ARITMETICA DI PRECISIONE E RECALL

Noi considereremo i training set come composti da n elementi (i.e. istanze) caratterizzati da d feature.

n = DIMENSIONE DEL TRAINING SET

d = DIMENSIONALITÀ DEL TRAINING SET → elemento come un punto in uno spazio \mathbb{R}^d (dimes.)

Possiamo vedere ciascun

→ M

Perciò, possiamo rappresentare il training set come una matrice $n \times d$. Se abbiamo a che fare con apprendimento supervisionato, oltre alle d feature di partenza abbiamo anche una feature target (che sarebbe quella oggetto di predizione); dunque, oltre alla matrice $n \times d$, si sfrutta anche un vettore $1 \times n$ per rappresentare il training set in modo esauritivo.

TARGET VECTOR

Approcci per costruire predittori con apprendimento supervisionato: → PREDITTORI DETERMINISTICI

- 1) Definire un algoritmo A che accetta in input l'elemento x da predire e tutto il training set e, in base a questi, genera in output il valore y dell'attributo target di x . L'algoritmo A fa uso della funzione $h(x, X, t)$, dove X = FEATURE MATRIX, t = TARGET VECTOR.

Tipicamente quest'approccio cerca nel training set T l'elemento più vicino a x e predice per il target di x esattamente il valore del target di quell'elemento più vicino. Una variante più generale, detta K-NEAREST-NEIGHBORS, cerca nel training set i K elementi più vicini a x e predice per il target di x il valore del target che compare più volte nei K elementi più vicini.

- 2) Definire un algoritmo A che accetta in input il training set T per generare un nuovo algoritmo Ar che ha innescoicamente le caratteristiche del training set (FASE DI LEARNING); a questo punto, si butta il training set e si usa Ar che accetta in input solo l'elemento x da predire per

generare in output il valore y dell'attributo target di x (FASE DI PREDIZIONE).

Un esempio per quest'approccio è la regressione lineare.

3) Definire un algoritmo A che accetta in input il training set T per generare s nuovi algoritmi $A_1^{(s)}, \dots, A_s^{(s)}$; a questo punto, $A_1^{(s)}, \dots, A_s^{(s)}$ accettano in input solo l'elemento x da predire e generano in output s valori target $y^{(1)}, \dots, y^{(s)}$. Tali valori target vanno combinati fra loro (ad esempio con la media aritmetica o andando a maggioranza) per ottenere così il valore y finale dell'attributo target di x .

In realtà, tipicamente, a ciascuno degli s predittori $A_1^{(s)}, \dots, A_s^{(s)}$ viene assegnato un peso $w^{(i)}$ ($i = 1, \dots, s$) che indica quanto ci si fida di quel predittore.

→ QUESTO MODELLO È DETTO ENSEMBLE.

21/03/2023

Supervised learning framework:

→ MODELLO DI GENERAZIONE DEL TRAINING SET: sia X l'insieme di tutti gli elementi finiti possibili che possiamo avere nel nostro dominio (i.e. l'insieme di tutte le possibili combinazioni di valori per le 5 feature degli elementi). Gli elementi che apparterranno al training set vengono presi campionati dall'insieme X utilizzando una distribuzione di probabilità p_t : $\forall x \in X, p_t(x)$ è la probabilità che x sia il prossimo elemento estratto da X per comporre il training set; p_t spesso può essere vista come una distribuzione di probabilità uniforme, il che vuol dire che ciascun elemento di X può appartenere al training set con la stessa probabilità.

→ MODELLO DI GENERAZIONE DEI TARGET DEGLI ELEMENTI APPARTENENTI AL TRAINING SET: le label (i.e. i valori della feature target) degli elementi del training set vengono generate a partire da una distribuzione di probabilità p_s : $\forall t \in Y, p_s(t|x)$ è la probabilità di osservare la label t GIVEN THAT l'elemento in questione è x (dove Y è l'insieme di tutte le label possibili).

Inizialmente considereremo un caso particolare di questo, in cui si utilizza una funzione deterministica f l'autocile la distribuzione di probabilità p_s per determinare la label di ciascun elemento. Diciamo che:

• Se usiamo la funzione $f()$, abbiamo necessariamente che due elementi uguali (i.e. con gli stessi valori per tutte le 5 feature) avranno la stessa label.

• Se usiamo una distribuzione di probabilità, possiamo avere elementi uguali associati a label

differenti (il che ha senso per esempio nel caso in cui vogliamo prevedere il sesso di una persona a partire da peso e altezza).

→ PER OPA BASTA NOCI DUNQUE SULL'UTILIZZO DELLA FUNZIONE DETERMINISTICA.

FUNZIONE LOSS: è una funzione $L()$ che misura l'errore della predizione del target di un elemento x appartenente al training set (di cui appunto conosciamo la label reale). È una funzione sensata da applicare solo nel caso della classificazione (e non della regressione). Nel caso particolare della classificazione binaria, $L()$ restituisce 0 se la predizione si è rivelata corretta, 1 altrimenti. Se $f(x)$ è la funzione deterministica che il classificatore ^{framework} _{assegnava} per _e _{assegnava} la label t a un qualsiasi elemento x , possiamo esprimere la funzione loss così: $L(h(x), t)$, con $L: \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}$, dove t è la label corretta per l'elemento x .

La funzione loss così espressa è anche detta **FUNZIONE RISCHIO (PREDICTION RISK)**

$$R(g, t) = L(h(x), t)$$

→ Se il modello ^{framework} si basa su una distribuzione di probabilità p_2 anziché su una funzione deterministica ^{f(x)}, la funzione rischio non è più uguale alla funzione loss, bensì è uguale alla media probabilistica (su p_2) della funzione loss su tutti i valori $t \in \mathcal{Y}$:

► **NEL CASO DELLA REGRESSIONE:** $R(\hat{y}, x) = E_{p_2}[L(\hat{y}, t)] = \int_{\mathcal{Y}} L(\hat{y}, t) \cdot p_2(t|x) dt$

► **NEL CASO DELLA CLASSIFICAZIONE:** $R(\hat{y}, x) = E_{p_2}[L(\hat{y}, t)] = \sum_{t \in \mathcal{Y}} L(\hat{y}, t) \cdot p_2(t|x)$

Nel supervised learning framework, la predizione ottimale è quella che minimizza il rischio di prediz.

$$y^*(x) = \underset{y}{\operatorname{argmin}} R(y, x) = \underset{y}{\operatorname{argmin}} E_{p_2}[L(y, t)]$$

↳ $y^*(x)$ è chiamato **STIMATORE DI BAYES**, che però non può essere implementato nella pratica perché richiede la conoscenza di $f(x)$ o di p_2 (cosa impossibile nella realtà).

RISCHIO (O PREDICTOR RISK): è l'errore di un preditore $h()$; a differenza della funzione rischio, non è relativo a un singolo elemento x , bensì a \mathcal{X} perde in considerazione tutto l'insieme \mathcal{X} (mediando i valori dati da tutte le funzioni rischio ~~sui~~ ^{sull'} insieme \mathcal{X}).

► **NEL CASO DELLA ^{FUNZIONE f(x)} DETERMINISTICA:** $R(h) = E_{p_2}[L(h(x), f(x))] = \int_{\mathcal{X}} L(h(x), f(x)) \cdot p_2(x) dx$

► **NEL CASO DELLA DISTRIBUZ. PROBABILITÀ p_2 :** $R(h) = E_{p_1, p_2}[L(h(x), t)] = \int_{\mathcal{X}} \int_{\mathcal{Y}} L(h(x), t) \cdot p_1(x) \cdot p_2(t|x) dx dt$ (ALMENO PER LA REGRESSIONE)

In pratica il predictor risk punta a rispondere alla seguente domanda: presso un elemento x a caso

a) è presa una label t a caso tra quelle possibili per x , quanto ci aspettiamo che il nostro predittore $h()$ stia gli?

Purtroppo, anche il calcolo del rischio è un qualcosa di ideale poiché richiede la conoscenza di p_s e di $p_a/h(t)$, che noi non abbiamo per ipotesi: abbiamo a disposizione solo il training set T . Di conseguenza, dobbiamo introdurre il concetto di RISCHIO EMPIRICO, che media i valori dati dalla funzione loss non su tutti gli elementi di X e su tutte le label assegnabili a ciascun elemento di X , bensì su tutti gli elementi del training set T (dato, ovviamente, ogni $x \in T$ ha una unica label t già assegnata):

$$l) \text{ RISCHIO EMPIRICO} = \bar{R}_T(h) = \frac{1}{|T|} \sum_{(x,t) \in T} L(h(x), t)$$

Ci piacerebbe trovare il predittore che 'stagna meno', ovvero il predittore che minimizza il rischio empirico:

$$h^* = \underset{h \in \mathcal{H}}{\operatorname{argmin}} \bar{R}_T(h) \quad \text{dove } \mathcal{H} \text{ è l'insieme dei predittori presi in considerazione.}$$

Dobbiamo che si tratta di un problema di ottimizzazione il cui argomento è una funzione e non una variabile/insieme di variabili come nei problemi di Ricerca Operativa.

* \mathcal{H} è detto INSIEME DI IPOTESI o BIAS INDUTTIVO.

N.B.: Qui stiamo lavorando solo ai dati del training set (e non ai dati nuovi), per cui c'è il pericolo per cui $\underset{h \in \mathcal{H}}{\operatorname{argmin}} \bar{R}_T(h)$ ci dà un'informazione fallace su qual è effettivamente il predittore migliore.

24/03/2023

Scelta dell'insieme di ipotesi:

Un'idea naïf che possiamo avere per costruire l'insieme di ipotesi è prendere quante più funzioni possibili: più scelta abbiamo, potenzialmente migliore sarà la funzione h^* che possiamo scegliere. In realtà, quest'approccio può portare a conseguenze infelice.

Consideriamo ad esempio un problema di classificazione binaria con la funzione loss che può assumere solo i valori 0 (in caso di predizione corretta) e 1 (in caso di predizione mal accertata), e supponiamo che $\forall x \in X$, $p(t=1|x) = \frac{1}{2}$, il che vuol dire che la metà degli elementi possibili viene etichettata con la label $y=1$, per cui l'altra metà degli elementi possibili viene etichettata con la label $y=0$. Consideriamo inoltre la seguente funzione di classificaz.:

$$h_0(x) = \begin{cases} 1 & \text{se } x = x_i \in X \wedge t_i = 1 \\ 0 & \text{altrimenti} \end{cases}$$

In pratica, $h_1(x)$ predice $y=1$ esclusivamente per gli elementi x_i del training set che hanno la label $t_i=1$, mentre predice $y=0$ per TUTTI gli altri elementi (compresi quelli non appartenenti al training set). Per $h_2(x)$ il rischio empirico è proprio pari a 0 ma per questo esempio si tratta di un'informazione fuorviante: se ci pensiamo, su una popolazione nuova (che non compare nel training set), il predittore predice solo label pari a 0, mentre la popolazione avrà circa la metà degli elementi etichettati come 0 e circa la metà degli elementi etichettati come 1; si tratta di una predizione pessima, perfettamente alla pari di un classificatore dummy o casuale.

→ ED ECCO QUI IL FENOMENO DELL'OVERFITTING.

D'altra parte, non possiamo neanche scegliere un insieme di ipotesi \mathcal{H} troppo ristretto, perché altrimenti rischieremmo di non avere alcuna funzione $h(\cdot)$ con un comportamento soddisfacente.

→ FENOMENO DELL'UNDERFITTING.

→ Più è elevata la dimensione del training set, più deve essere grande l'insieme \mathcal{H} per sperimentare il fenomeno dell'overfitting.

Bias vs Varianza:

Il rischio associato a $h^*(\cdot)$ può essere decomposto in due parti: $R(h^*) = E_B + E_V$

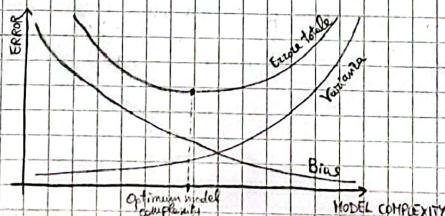
- $E_B = \text{BIAS} = \text{rischio minimo ottenibile da una qualunque funzione } h \in \mathcal{H}$. È un valore che dipende esclusivamente dall'insieme \mathcal{H} e non dal training set.

- $E_V = \text{VARIANZA} = \text{differenza tra il rischio minimo ottenibile dalla funzione } h^* \in \mathcal{H} \text{ una volta fissato il training set e } E_B$.

→ Più il bias è elevato, più i predittori in \mathcal{H} tendono a comportarsi male.

→ Più la varianza è elevata, più i predittori in \mathcal{H} tendono a cambiare comportamento al variare del training set.

TRADEOFF TRA BIAS E VARIANZA: avere bias basso porta ad avere varianza tendenzialmente elevata e viceversa.



FUNZIONI LOSS & TRAINING

Minimizzazione della loss function:

Abbiamo detto che un obiettivo che possiamo porci è minimizzare il rischio empirico associato a un training set T , che è dato da $\bar{R}_T(\theta) = \frac{1}{|T|} \sum_{(x,t) \in T} L(h_\theta(x), t)$.

Se siamo in grado di esprimere la nostra funzione $h(\cdot)$ come un insieme di parametri θ , allora minimizzare $\bar{R}_T(\theta)$ equivale a minimizzare $L(\theta; T) = \sum_{i=1}^n L(\theta; x_i, y_i) = \sum_{i=1}^n L_i(\theta)$.

Per minimizzare $L(\theta; T)$, dovremmo calcolarne il MINIMO GLOBALE. Per farlo, basterebbe porre a zero le derivate delle funzione rispetto a tutti i parametri in θ :

$$\nabla_\theta L(\theta; T) = 0 \Rightarrow \frac{\partial}{\partial \theta_j} L(\theta; T) = 0 \quad \forall i \quad \rightarrow \text{DA QUI SI RISOLVE IL SISTEMA DI EQUAZIONI CORRESPONDENTI!}$$

PROBLEMI:

- Il sistema di equazioni può avere molteplici soluzioni, le quali possono anche essere relative a punti di massimo o flessi/punti di sella - o anche punti di minimo LOCALE.
- Il sistema di equazioni potrebbe essere difficile (o addirittura impossibile) da risolvere analiticamente.

DISCESA DEL GRADIENTE:

Una metodologia che permette di trovare almeno un minimo locale in modo analiticamente semplice è la discesa del gradiente, che nella pratica consiste nel far variare i parametri in θ in modo tale che, all'interno della funzione costo (nel nostro caso $L(\cdot)$), scendiamo via via verso un minimo locale.

↓ ANCHE $\bar{R}_T(\cdot)$

→ Si ha una fase preliminare in cui si parte da un punto iniziale dello spazio dei parametri:

$$\underline{\theta}^{(0)} = (\theta_0^{(0)}, \theta_1^{(0)}, \dots, \theta_d^{(0)}) \quad \text{che avrà un rischio empirico corrispondente } \bar{R}_T(\underline{\theta}^{(0)}).$$

→ Iterativamente (all'iterazione i), il valore corrente $\underline{\theta}^{(i)}$ viene modificato nella direzione della discesa più rapida di $\bar{R}_T(\underline{\theta})$, che è una direzione corrispondente a un valore η negativo del gradiente calcolato in $\underline{\theta}^{(i)}$. In particolare, allo step i , $\theta_j^{(i+1)}$ viene aggiornato in questo modo:

$$\theta_j^{(i)} := \theta_j^{(i-1)} - \eta \frac{\partial}{\partial \theta_j} \bar{R}_T(\underline{\theta}) \Big|_{\underline{\theta}^{(i)}} = \theta_j^{(i-1)} - \frac{\eta}{|T|} \sum_{(x,t) \in T} \frac{\partial}{\partial \theta_j} L(h_\theta(x), t) \Big|_{\underline{\theta}^{(i-1)}}$$

È un iperparametro che indica quanto è 'grande' il passo che stiamo facendo. Se è molto piccolo, ci vuole molto iterazioni per raggiungere il minimo locale; se è molto grande, si rischia di superare il minimo locale e di oscillare intorno a esso per varie iterazioni.

$$\downarrow \text{IN FORMA MATEMATICA: } \underline{\theta}^{(i)} := \underline{\theta}^{(i-1)} - \eta \nabla_{\theta} \bar{R}_T(\underline{\theta}) \Big|_{\underline{\theta}^{(i-1)}} = \underline{\theta}^{(i-1)} - \frac{\eta}{|T|} \sum_{(x,t) \in T} \nabla_{\theta} L(h_\theta(x), t) \Big|_{\underline{\theta}^{(i-1)}}$$

ricordiamo però che noi vogliamo trovare UN minimo GLOBALE, non LOCALE!

A tal proposito, una tipologia di funzioni che può piacerci molto è data dalle funzioni CONVESSE, dove una funzione $f(x)$ è convessa $\Leftrightarrow \forall x_1, x_2 \quad \forall \lambda \in (0,1) \quad f(\lambda x_1 + (1-\lambda)x_2) \leq \lambda f(x_1) + (1-\lambda)f(x_2)$. Di fatto, per una funzione $f(x)$ convessa, ogni minimo locale è anche un minimo globale. Per giunta, per una funzione $f(x)$ STRETTAMENTE CONVESSA, il punto di minimo è unico.

\hookrightarrow LE FUNZIONI QUADRATICHE SONO UN ESEMPIO VALIDO.

PROPRIETÀ DELLE FUNZIONI CONVESSE:

\rightarrow La somma di funzioni (strettamente) convesse è (strettamente) convessa.

\rightarrow Il prodotto tra una funzione (strettamente) convessa e una costante è (strettamente) convesso.

E dal momento che $\bar{R}_T(h) = \frac{1}{|T|} \sum_{(x,t) \in T} L(h(x), t) \propto \sum_{(x,t) \in T} L(\theta; x, t)$:

• Se $L(\theta; x, t)$ è (strettamente) convessa $\Rightarrow \bar{R}_T(h)$ è strettamente convessa.

• Se $L(\theta; x, t)$ è convessa \Rightarrow ogni minimo locale di $\bar{R}_T(h)$ è anche un minimo globale.

• Se $L(\theta; x, t)$ è strettamente convessa $\Rightarrow \exists!$ minimo in $\bar{R}_T(h)$.

11/04/2023

Funzioni loss per la regressione:

QUADRATIC LOSS:

$L(y, t) = (y - t)^2$ \rightarrow per una specifica predizione, misura il quadrato della distanza tra il punto t (il valore corretto per l'elemento x) e il punto y (il valore) predetto dal nostro predittore.

Il rischio empirico siene dunque: $\bar{R}_T(h) = \sum_{(x,t) \in T} (h(x) - t)^2$ dove $h(x) = y$

Nel caso particolare della regressione lineare, $h(x)$ sarà una funzione lineare del tipo $h(x) = w^T x + b$, dove w è il vettore dei coefficienti definiti per una combinazione lineare delle feature (espresse nel vettore x), e b è un termine noto (\rightarrow tutti i componenti di w e b insieme compongono i parametri della funzione $h()$). Di conseguenza, la funzione da minimizzare risulterà essere questa:

$$L(\theta; T) = L(w, b; T) = \sum_{(x,t) \in T} (w^T x + b - t)^2$$

Traffarsi di una funzione quadratica, che è strettamente convessa, per cui ha un unico punto di minimo locale che è anche un minimo globale. Questo è l'unico caso che vediamo in cui porre a zero il gradiente di $L(\theta; T)$ è una tecnica che funziona per il calcolo del minimo globale; di fatto, si ottengono $d+1$ equazioni lineari a $d+1$ incognite del tipo:

$$\frac{\partial}{\partial w_i} L(w, b; T) = \sum_{(x,t) \in T} (w^T x + b - t) w_i = 0 \quad ; \quad \frac{\partial}{\partial b} L(w, b; T) = \sum_{(x,t) \in T} (w^T x + b - t) = 0$$

Svantaggio della quadratic loss: dà molto peso agli outliers.

Absolute Loss:

$$L(y, t) = |y - t| \quad \rightarrow \text{per una specifica predizione, misura il modulo della distanza tra il punto } t \text{ corretto e il punto } y \text{ predetto.}$$

Svantaggio dell'absolute loss: è difficile da trattare analiticamente.

Huber Loss:

$$L(y, t) = \begin{cases} \frac{1}{2}(t-y)^2 & \text{se } |t-y| \leq \delta \\ \delta(|t-y|) - \frac{\delta}{2} & \text{se } |t-y| > \delta \end{cases}$$

è una via di mezzo tra la quadratic loss e l'absolute loss:
al centro è esattamente una parabola, mentre agli estremi ha
l'andamento di una retta; δ è un iperparametro.

Funzioni loss per la classificazione:

Assunzioni:

- Noi consideriamo i casi di classificazione binaria, con le due classi identificate dai valori target -1 e 1.
- La predizione restituisce un valore y reale: se è positivo, ricadiamo nel caso $\text{LABEL}=1$ e viceversa.

0/1 Loss:

$$L(y, t) = \begin{cases} 1 & \text{se } \text{sgn}(y) \neq t \\ 0 & \text{se } \text{sgn}(y) = t \end{cases} \quad \begin{array}{l} \text{(predizione errata)} \\ \text{(predizione corretta)} \end{array}$$

dove $\text{sgn}(x) = 1$ se $x > 0$
 $\text{sgn}(x) = -1$ se $x \leq 0$

Problemi della 0/1 loss:

→ Non è convessa.

→ La derivata, dove esiste, è sempre nulla: la discesa del gradiente non può essere applicata.

→ Già nel caso semplice in cui $h(\cdot)$ è una funzione lineare: $\bar{R}_c(h) = \frac{1}{|T|} \sum_{(x, y) \in T} \mathbb{I}[(w^T x + b) \frac{y}{|y|} < 0]$
l'individuazione dei valori di w, b che minimizzano $\bar{R}_c(h)$ è un problema NP-hard.

Perceptron Loss:

$$L(y, t) = \max(0, -yt)$$

→ Se la predizione è corretta, allora $L(y, t) = 0$; altrimenti, $L(y, t)$ cresce linearmente col modulo di yt (ovvero, più il predittore si sente sicuro quando sbaglia, più deve pagare).

Non è una funzione SURROGATA della 0/1 loss nel senso che non la approssima dall'alto: ci sono dei punti in cui la 0/1 loss è maggiore della perceptron loss.

Hinge Loss:

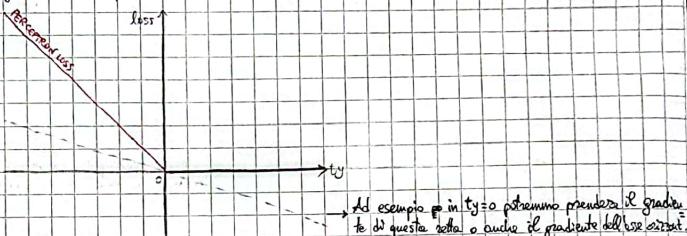
$$L(y, t) = \max(0, 1 - yt)$$

→ Già come la perceptron loss ma è spostata di 1 più a destra. Se la predizione è errata, $L(y, t)$ cresce linearmente col modulo di $|yt|$ (ma stanchi con valore iniziale 1 e non 0); se la pred. è corretta e $|yt| \geq 1$, allora $L(y, t) = 0$; altrimenti, $L(y, t) = |yt|$.

La hinge loss è una funzione sottogata della 0/1 loss.

La perceptron loss e la hinge loss presentano un punto di non derivabilità ma sono convesse.

Per affrontare il problema della non derivabilità, si introduce il concetto di SUBGRADIENTE: in un punto di non derivabilità si considera il gradiente di un'altra funzione che passa per quel punto e sta sotto alla nostra funzione di partenza. Ad esempio:



SQUARE LOSS:

$$L(y, t) = (1 - yt)^2 \rightarrow \text{è molto (troppo) elevata tutte le volte in cui il predittore è sicuro, anche quando la predizione è giusta; viene da sé che per la classificazione è una funzione loss inaccettabile.}$$

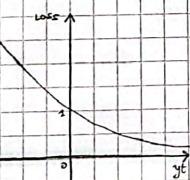
Non è una funzione sottogata della 0/1 loss.

13/04/2023

Finora abbiamo visto funzioni loss per la classificazione deterministica. Concentriamoci ora sulla classificazione probabilistica, in cui il predittore definisce una probabilità per ogni label da associare a un determinato elemento.

LOG LOSS:

$$L(y, t) = \frac{1}{\log 2} \log(1 + e^{-yt})$$



- È una versione smooth ('avvolgutata') della hinge loss.
- È continua, convessa, sottogata e ha gradiente continuo.
- Posizioni ampiamente sbagliate potrebbero essere penalizzate eccessivamente.

Poiché ora abbiamo a che fare con predittori probabilistici, dobbiamo trovare un modo per stabilire quanto la distribuzione di probabilità generata dal predittore sul valore del target si avvicini al valore del target stesso. Ad esempio, supponiamo di avere 5 classi distinte $\{1, 2, 3, 4, 5\}$; allora, il predittore restituirà in output un vettore di tipo $[p_1, p_2, p_3, p_4, p_5]$. Supponendo che il target

dell'elemento su cui si è fatta la predizione sia h_i , possiamo esprimere il target stesso col vettore $[0, 0, 0, 1, 0]$. A questo punto ci sarebbe un meccanismo per confrontare i due vettori (i.e. le due distribuzioni). Viene in aiuto la CROSS ENTROPY, che misura ^{di una} la differenza tra due distribuzioni.

$$\text{CROSS ENTROPY} = -E_p[\log q(x)] = -\int p(x) \log q(x) dx \quad (\text{o sommatoria nel discreto})$$

Se p, q sono la stessa distribuzione, la cross entropy è proprio uguale all'entropia di p :

$$H(p) = -E_p[\log p(x)] = -\int p(x) \log p(x) dx \quad (\text{o sommatoria nel discreto})$$

Ricordiamo che la cross entropy è correlata alla DIVERGENZA DI KULLBACK-LEIBLER $KL(p||q)$:

$$KL(p||q) = -\int p(x) \log \frac{q(x)}{p(x)} dx = -\int p(x) \log q(x) dx + \int p(x) \log p(x) dx = -E_p[\log q(x)] - H(p)$$

(o sommatoria nel discreto)

La cross entropy e la divergenza KL danno un'idea della bontà media della codifica che attribuisce a ciascuna parola una rappresentazione (anzi, una distribuzione) più o meno simile a quella originaria.

RICAPITOLANDO:

→ L'entropia $H(p) = -E_p[\log p]$ denota il numero medio di bit trasmessi per simbolo quando la distribuzione dei simboli $p(x)$ è nota.

→ La divergenza KL $KL(p||q)$ denota il numero aggiuntivo di bit trasmessi per simbolo (rispetto al minimo possibile, che è dato quando $p(x)$ è nota), ed è dunque dato da $H(p)$ quando viene utilizzata la distribuzione $q(x)$ al posto di $p(x)$.

→ La cross entropy $-E_p[\log q]$ denota il numero medio totale di bit trasmessi per simbolo quando viene utilizzata la distribuzione $q(x)$ al posto di $p(x)$.

MORALE DELLA FAVOLA: minimizzare la log loss corrisponde a minimizzare la cross entropy.

EXponential LOSS:

$$L(y, t) = e^{-yt}$$

→ ha una forma simile a quella della log loss, con la differenza che cresce molto più velocemente quando l'errore della predizione diventa più grande.

Così come la log loss, è continua, convessa, smaccata e con gradiente continuo.

Affondamento sulla discesa del gradiente:

BATCH GRADIENT DESCENT:

È la tecnica che abbiamo già visto:

$$\Theta^{(k+1)} = \Theta^{(k)} - \frac{\eta}{|T|} \sum_{(x, t) \in T} \nabla_{\Theta} L(h_{\theta}(x), t) \Big|_{\Theta^{(k)}}$$

$$\Theta_i^{(k+1)} = \Theta_i^{(k)} - \frac{\eta}{|T|} \sum_{(x, t) \in T} \frac{\partial}{\partial \Theta_i} L(h_{\theta}(x), t) \Big|_{\Theta^{(k)}}$$

VEDI LA SEZIONE
INTRO SULLI
ELEMENTI DEL
TRAINING SET

Si chiama BATCH GRADIENT DESCENT perché, a ogni iterazione, provvede la scansione dell'intero training set.

TRA L'ALTRÒ LA SCANSIONE AVVIENE PER OGNI CAMPIONE DI T .

Consideriamo l'esempio della regressione lineare con funzione loss quadratica:

$$h(x) = \sum_{j=1}^d \theta_j x_j + \theta_0 \quad \Rightarrow \quad L(h(x), t) = (h(x) - t)^2 = \left(\sum_{j=1}^d \theta_j x_j + \theta_0 - t \right)^2$$

corrisponde al punto x_i del training set

corrisponde a $y_i = \theta_0 + \theta_1 x_i + \dots + \theta_d x_d$

Allora il gradiente è dato da:

$$\begin{cases} \frac{\partial}{\partial \theta_i} L(h_\theta(x), t) = \left(\sum_{j=1}^d \theta_j x_j + \theta_0 - t \right) x_i \cdot 2 & \forall i = 1, \dots, d \\ \frac{\partial}{\partial \theta_0} L(h_\theta(x), t) = \left(\sum_{j=1}^d \theta_j x_j + \theta_0 - t \right) \cdot 2 \end{cases}$$

Possiamo trascurare le
fattore multipli di 2,
perché, come vedremo,
verrà ugolato in η .

In definitiva:

$$\begin{cases} \theta_i^{(k+1)} = \theta_i^{(k)} - \frac{\eta}{|T|} \sum_{(x, t) \in T} \left(\sum_{j=1}^d \theta_j^{(k)} x_j + \theta_0^{(k)} - t \right) x_i & \forall i = 1, \dots, d \\ \theta_0^{(k+1)} = \theta_0^{(k)} - \frac{\eta}{|T|} \sum_{(x, t) \in T} \left(\sum_{j=1}^d \theta_j^{(k)} x_j + \theta_0^{(k)} - t \right) \end{cases}$$

STOCHASTIC GRADIENT DESCENT:

A ogni iterazione noi calcoliamo il gradiente del rischio empirico (i.e. il gradiente della media su tutti i punti del training set della funzione loss), bensì calcoliamo il gradiente della funzione loss su un unico punto del training set; magari ~~non~~ punto x_j differente a ogni iterazione della discesa del gradiente. In tal modo, l'avvicinamento verso il punto di minimo avviene in modo più lento e irregolare.

$$\theta_j^{(k+1)} = \theta_j^{(k)} - \eta \nabla_{\theta_j} L(h_\theta(x_j), t_j) |_{\theta^{(k)}}$$

$$\theta_0^{(k+1)} = \theta_0^{(k)} - \eta \frac{\partial}{\partial \theta_0} L(h_\theta(x_j), t_j) |_{\theta^{(k)}}$$

Nel caso della regressione lineare con funzione loss quadratica abbiamo:

$$\begin{cases} \theta_i^{(k+1)} = \theta_i^{(k)} - \eta \left(\sum_{j=1}^d \theta_j^{(k)} x_{j,i} + \theta_0^{(k)} - t \right) x_i & \forall i = 1, \dots, d \\ \theta_0^{(k+1)} = \theta_0^{(k)} - \eta \left(\sum_{j=1}^d \theta_j^{(k)} x_{j,i} + \theta_0^{(k)} - t \right) \end{cases}$$

MINI-BATCH GRADIENT DESCENT:

A ogni iterazione calcoliamo il gradiente della funzione loss mediata su un insieme B_m di m punti del training set:

$$\theta^{(k+1)} = \theta^{(k)} - \frac{\eta}{m} \sum_{(x, t) \in B_m} \nabla_{\theta} L(h_\theta(x), t) |_{\theta^{(k)}}$$

$$\theta_i^{(k+1)} = \theta_i^{(k)} - \frac{\eta}{m} \sum_{(x, t) \in B_m} \frac{\partial}{\partial \theta_i} L(h_\theta(x), t) |_{\theta^{(k)}}$$

Nel caso della regressione lineare con funzione loss quadratica abbiamo:

$$\theta_i^{(k+1)} = \theta_i^{(k)} - \frac{\eta}{m} \sum_{(x, t) \in B_m} \left(\sum_{j=1}^d \theta_j^{(k)} x_{j,i} + \theta_0^{(k)} - t \right) x_i \quad \forall i = 1, \dots, d$$

$$\theta_0^{(k+1)} = \theta_0^{(k)} - \frac{\eta}{m} \sum_{(x, t) \in B_m} \left(\sum_{j=1}^d \theta_j^{(k)} x_{j,i} + \theta_0^{(k)} - t \right)$$

MOMENTUM GRADIENT DESCENT:

A ogni iterazione si sposta non solo in base all'attuale valore del gradiente, bensì anche in base a tutto lo storico del valore del gradiente delle iterazioni precedenti. Abbiamo dunque che:

$$\tilde{\theta}^{(k+1)} = \tilde{\theta}^{(k)} + \nabla^{(k+1)}$$

dove:

$$\text{SPOSTAMENTO DA FARE} = \nabla^{(k+1)} = \gamma \nabla^{(k)} - \eta \sum_{(x,t) \in B_2} \nabla_{\theta} L(h_\theta(x), t) \Big|_{\theta^{(k)}}$$

PROIEZIONE DI TUTTI GLI

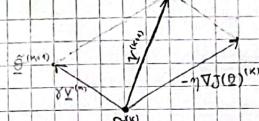
ESAMP. TUTT'UNO

GRADIENTE PER L'ITERAZIONE ATTUALE

(PER BREVITÀ POSSIAMO INDICARLO così: $-\eta \nabla J(\theta)^{(k)}$)

SPOSTAMENTO DA FARE NELL'ITERAZIONE PRECEDENTE

Graficamente parlando, la situazione è questa:



$$\text{Sviluppando } \nabla^{(k+1)} \text{ viene che: } \nabla^{(k+1)} = \nabla^{(0)} - \eta \sum_{i=0}^k \sum_{(x,t) \in B_2} \nabla_{\theta} L(h_\theta(x), t)^{(i)} \Rightarrow$$

$$\Rightarrow \tilde{\theta}^{(k+1)} = \tilde{\theta}^{(k)} + \nabla^{(k+1)} = \tilde{\theta}^{(k)} + \nabla^{(0)} - \eta \sum_{i=0}^k \sum_{(x,t) \in B_2} \nabla_{\theta} L(h_\theta(x), t)^{(i)}$$

Nel caso della regressione lineare con funzione loss quadratica abbiamo:

$$\tilde{\theta}_i^{(k+1)} = \tilde{\theta}_i^{(k)} + v_i^{(k+1)}$$

con:

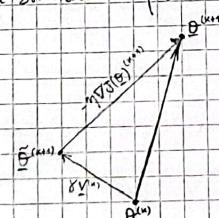
$$v_i^{(k+1)} = \begin{cases} \gamma v_i^{(k)} - \eta \sum_{(x,t) \in B_2} (\sum_{j=1}^d \tilde{\theta}_j^{(k)} x_j + \tilde{\theta}_0^{(k)} - t) x_i & \forall i = 1, \dots, d \\ \gamma v_i^{(k)} - \eta \sum_{(x,t) \in B_2} (\sum_{j=1}^d \tilde{\theta}_j^{(k)} x_j + \tilde{\theta}_0^{(k)} - t) & \text{per } i = 0 \end{cases}$$

NESTEROV GRADIENT DESCENT:

È un perfezionamento del momentum gradient descent: la differenza sta nel fatto che, a ogni iterazione, anziché calcolare il gradiente nel punto $\tilde{\theta}^{(k)}$, lo calcola in $\tilde{\theta}^{(k+1)}$ (vedere figura precedente).

Quest'approccio è valido poiché ci si aspetta che $\tilde{\theta}^{(k+1)}$ sia più vicino a $\tilde{\theta}^{(k+2)}$ rispetto a $\tilde{\theta}^{(k)}$.

Graficamente parlando, la situazione è questa:



AGGIORNAMENTO DINAMICO DEL LEARNING RATE η :

In realtà, avrebbe senso ridurre il valore di η man mano che ci avviciniamo al punto di minimo.

• STEP DECAY: ridurre η ogni T epochhe,

$$\cdot \text{EXPONENTIAL DECAY: } \eta^{(k)} = \eta^{(0)} e^{-\alpha k}$$

$$\cdot \text{1/K DECAY: } \eta^{(k)} = \frac{\eta^{(0)}}{1+\alpha k}$$

• AGGIORNAMENTO DI η SULLA BASE DEL MONITORAGGIO DEL PROCESSO DI LEARNING. → Lo aggiorniamo a partire dalla pagina successiva.

14/04/2023

ADAGRAD:

È la prima variante della discesa del gradiente in cui η varia dinamicamente.

$$\Theta_j^{(k+1)} = \Theta_j^{(k)} + \Delta_{j,k}$$

dove $\Delta_{j,k} = -\eta_j^{(k)} g_{j,k}$ è la variazione che viene fatta su $\Theta_j^{(k)}$. In particolare, $g_{j,k} = \frac{\partial}{\partial \Theta_j} L(\theta^{(k)}, x)$ è la derivata parziale rispetto a Θ_j della funzione loss calcolata in $\Theta^{(k)}$; per quanto invece riguarda $\eta_j^{(k)}$:

$$\eta_j^{(k)} = \frac{\eta}{\sqrt{G_{j,k} + \epsilon}}$$

dove:

- η è il learning rate iniziale per tutti i parametri Θ_j .

- $G_{j,k} = \sum_{i=0}^k g_{j,i}^2$ è la somma dei quadrati delle derivate rispetto a Θ_j della funzione loss calcolate su tutti i $\Theta^{(i)}$ precedenti.

- ϵ è un valore arbitrariamente piccolo che serve solo ad assicurarsi che il denominatore di $\eta_j^{(k)}$ sia mai nullo.

NB: man mano che si va avanti con le iterazioni, il denominatore di $\eta_j^{(k)}$ diventa sempre più grande $\Rightarrow \eta_j^{(k)}$ diventa sempre più piccolo. Ma se il valore dei gradienti in media è sufficientemente elevato, a un certo punto $\eta_j^{(k)}$ inizia a tendere a zero e non si è praticamente più in grado di spostarsi all'interno della funzione. Ciò rappresenta la debolezza principale di Adagrad.

RMS PROP:

È come Adagrad, solo che utilizza un termine $\tilde{G}_{j,k}$ al posto di $G_{j,k}$, dove:

$$\tilde{G}_{j,k} = \gamma \tilde{G}_{j,k-1} + (1-\gamma) g_{j,k}^2 = (1-\gamma) \sum_{i=0}^k \gamma^{k-i} g_{j,i}^2 \quad \text{con } 0 < \gamma < 1 \quad [\gamma = \text{FATORE DI DECAY}]$$

L'idea è che, tra i vari quadrati delle derivate rispetto a Θ_j della funzione loss calcolate precedentemente, ciascun termine viene moltiplicato per un fattore minore di 1, così da ridurne il peso all'interno della somma. L'effetto finale è che il denominatore di $\eta_j^{(k)}$ cresce più lentamente $\Rightarrow \eta_j^{(k)}$ decresce più lentamente.

ADADELTA:

È come RMS prop, solo che definisce lo spostamento $\Delta_{j,k}$ nel seguente modo:

$$\Delta_{j,k} = -\frac{\sqrt{\tilde{G}_{j,k-1} + \epsilon}}{\sqrt{\tilde{G}_{j,k} + \epsilon}} g_{j,k} \quad \rightarrow \text{In RMS prop il numeratore era semplicemente } \eta \cdot$$

dove $\tilde{G}_{j,k} = \gamma \tilde{G}_{j,k-1} + (1-\gamma) \Delta_{j,k}^2 = (1-\gamma) \sum_{i=0}^k \gamma^{k-i} \Delta_i^2$; è la somma smorzata dei quadrati

degli spostamenti, definita in modo del tutto analogo a $\tilde{G}_{j,k}$.

Il vantaggio di quest'approccio è che richiede la definizione di un parametro di tuning (η) in meno rispetto a RMS prop.

ADAM:

È come RMS prop, solo che definisce lo spostamento $\Delta_{j,k}$ nel seguente modo:

$$\Delta_{j,k} = -\frac{\eta}{\sqrt{\hat{G}_{j,k}} + \epsilon} \hat{H}_{j,k} \quad \text{dove:}$$

$$\cdot \hat{G}_{j,k} = \frac{\tilde{G}_{j,k}}{1-\gamma^2} \quad \text{con } \tilde{G}_{j,k} = \gamma \tilde{G}_{j,k-1} + (1-\gamma) g_{j,k}^2$$

$$\cdot \hat{H}_{j,k} = \frac{\hat{H}_{j,k}}{1-\beta^2} \quad \text{con } \tilde{H}_{j,k} = \beta \tilde{H}_{j,k-1} + (1-\beta) g_{j,k}$$

γ, β sono parametri di tuning compresi tra 0 e 1.

REGRESSIONE LINEARE

Presto che come funzione di predizione venga utilizzata una combinazione lineare dei valori delle feature dell'elemento x , offre una retta in uno spazio a $d+1$ dimensioni (la $d+1$ -esima è data da t):

$$y(x, w) = w_0 + w_1 x_1 + w_2 x_2 + \dots + w_d x_d$$

Si tratta di una funzione lineare sia rispetto ai parametri w , sia rispetto alle feature x .

Scrittura più compatta: $y(x, w) = w^T \bar{x}$ dove $\bar{x} = (1, x_1, \dots, x_d)$

Base function:

È possibile definire m funzioni $\phi_1(\cdot), \dots, \phi_m(\cdot) : \mathbb{R}^d \rightarrow \mathbb{R}$ che prendono in input un elemento x e generano in output un valore reale. Queste sono dette BASE FUNCTION e possono essere fatte in qualsiasi modo: possono essere il quadrato di una certa componente, il prodotto tra più componenti, oppure il vettore $[\phi_1(x), \dots, \phi_m(x)]$ può essere il risultato di una feature selection, e così via. Possiamo dunque scrivere:

$$y(x, w) = \sum_{j=1}^m w_j \phi_j(x) \quad \begin{array}{l} \rightarrow \text{È UN'ESPRESSIONE PIÙ GENERALE RISpetto A QUELLA RICORDATA INIZIALMENTE.} \\ \text{E COMUNQUE LINEARE RISPETTO A } w. \end{array}$$

Possiamo vedere anche in questo modo: ciascun vettore $x \in \mathbb{R}^d$ è mappato su un nuovo vettore in \mathbb{R}^m ,

che si chiama $\phi(x) = (\phi_1(x), \dots, \phi_m(x))$. In pratica il nostro problema è mappato da uno spazio d -dimensionale a uno spazio m -dimensionale.

Perché fare questo? Magari con una certa trasformazione si potrebbero fare delle predizioni migliori.

Ad esempio, se $m > d$, sto per definire funzioni con più parametri e, quindi, più complesse. Un altro modo per rendere più complesse le funzioni è avere un $\phi(x) = \text{polinomio di grado } q$.

NB: non stiamo facendo altro che cambiare il modo in cui l'elemento x viene rappresentato.

Rimane un problema aperto: come facciamo a selezionare le funzioni $\phi_1(\cdot), \dots, \phi_m(\cdot)$?

APPROCCIO 1: definire tali funzioni strettamente e in modo indipendente dal training set.

APPROCCIO 2: rappresentare tali funzioni mediante un insieme di parametri e trovare i valori per tali parametri che minimizzino la funzione costo (la loss function). Così il predittore lavora in 2 fasi:

i) REPRESENTATION LEARNING = apprendimento su come rappresentare al meglio i dati: in base al training set, si stabiliscono i parametri migliori per le funzioni $\phi_1(\cdot), \dots, \phi_m(\cdot)$.

ii) Apprendimento su come effettuare al meglio la predizione: in base al training set, si stabiliscono i parametri migliori per $y(\underline{x}, \underline{w})$.

In questo secondo approccio possiamo anche considerare tutti i parametri insieme per trovare il punto di minimo della funzione costo. Tuttavia, se ci pensiamo, si tratta di un approccio utilizzato per le RETI NEURALI, che sono dei meccanismi di predizione composti da N strati: i primi $N-1$ strati si occupano tutti di effettuare il representation learning e solo l'ultimo si occupa realmente di effettuare la predizione.

→ Dal punto di vista geometrico, con le basse funzioni, il mio training set è $\bar{\mathbf{X}}$:

$$\bar{\mathbf{X}} = \begin{bmatrix} -\bar{x}_1- \\ -\bar{x}_2- \\ \vdots \\ -\bar{x}_n- \end{bmatrix} = \begin{bmatrix} 1 & x_{11} & \cdots & x_{1d} \\ 1 & x_{21} & \cdots & x_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & \cdots & x_{nd} \end{bmatrix} \quad \leftarrow m \text{ elementi con } d \text{ feature}$$

Viene trasformato in:

$$\bar{\Phi} = \begin{bmatrix} \phi_1(x_1) & \phi_2(x_1) & \cdots & \phi_m(x_1) \\ \phi_1(x_2) & \phi_2(x_2) & \cdots & \phi_m(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_1(x_n) & \phi_2(x_n) & \cdots & \phi_m(x_n) \end{bmatrix} \quad \begin{array}{l} \text{UNA COLONNA È RELATIVA} \\ \text{A UNA FUNZIONE } \phi_i(\cdot) \end{array}$$

UNA RIGA È RELATIVA A UN'IMMAGINE (RAPPRESENTAZIONE) DI UN ELEMENTO.

Regressione lineare con loss quadratica:

$$L(y_i, t_i) = (y(x_i, \underline{w}) - t_i)^2 \Rightarrow \bar{R}_r(\underline{y}) = \frac{1}{n} \sum_{i=1}^n (y(x_i, \underline{w}) - t_i)^2 = \frac{1}{2} \sum_{i=1}^n \left(\sum_{j=0}^m w_j \phi_j(x_i) - t_i \right)^2$$

Allora il nostro obiettivo è minimizzare il costo empirico $\bar{R}_r(\underline{y})$; ma questo significa minimizzare la seguente funzione (che è la stessa cosa, è solo più comoda dal punto di vista delle costanti t_i):

$$E(\underline{w}) = \frac{1}{2} \sum_{i=1}^n (y(x_i, \underline{w}) - t_i)^2 = \frac{1}{2} \sum_{i=1}^n \left(\sum_{j=0}^m w_j \phi_j(x_i) - t_i \right)^2$$

Si tratta di una funzione convessa le cui derivate rispetto alle componenti di \underline{w} sono lineari $\xrightarrow{\text{strettamente}}$. Ponendo le derivate a zero, si ottengono molti equazioni a molti incognite e, risolvendo tale sistema:

sistema di equazioni, si ottiene proprio il punto di minimo globale.

→ così ci riscriviamo la discesa del gradiente :

Le varie derivate sono fatte così:

$$\frac{\partial E(\underline{w})}{\partial w_i} = \sum_{j=1}^n \left(\sum_{j=0}^m w_j \phi_j(x_i) - t_i \right) \phi_i^{(j)}(x_i)$$

fasi:
Additivamente la soluzione di questo sistema può essere calcolata con una forma chiusa data dalla seguente espressione: $\underline{w}^* = (\Phi^T \Phi)^{-1} \Phi^T \underline{t}$ dove \underline{t} è il vettore dei target nel training set.

18/04/2023

Limitare la complessità del modello:

Sappiamo che il nostro modello non può essere troppo complesso per evitare l'overfitting: con riferimento alle funzioni ϕ di predizione polinomiali, più il grado del polinomio è alto, più si rischia di far specificizzare troppo il predittore sul training set; d'altra parte, è anche vero che, a parità di grado del polinomio, la funzione è tanto più complessa quanto più sono elevati i coefficienti. Perciò, piuttosto che limitare il grado del polinomio, si può introdurre un nuovo termine all'interno della funzione costo (REGOLARIZZAZIONE) che spinga affinché i valori dei coefficienti (i.e. dei parametri) siano limitati.

In tal modo, la funzione costo diventa: $E(\underline{w}) = E_0(\underline{w}) + \lambda E_w(\underline{w})$, dove:

$E_0(\underline{w})$ è la vecchia $E(\underline{w})$ e dipende sia dal training set sia dai parametri in \underline{w} .

$E_w(\underline{w})$ è il nuovo termine che rappresenta il costo derivante dal valore assoluto dei parametri, ed è indipendente dal training set.

. λ è il COEFFICIENTE DI REGOLARIZZAZIONE e stabilisce il peso (l'importanza) che hanno $E_0(\underline{w})$, $E_w(\underline{w})$.
L'PIÙ È GRANDE, PIÙ TENDO ALL'UNDERFITTING (E VICEVERSA). D'ALTRA PERTA, SE È SUFF. PICCOLO, TENDO AD AVERE MAGGIORA VARIANZA; SE È SUFF. GRANDE TENDO AD AVERE MAGGIORE BIAS.

La forma più semplice e diffusa di $E_w(\underline{w})$ è data dalla norma quadratica di \underline{w} :

$$E_w(\underline{w}) = \frac{1}{2} \underline{w}^T \underline{w} = \frac{1}{2} \sum_{i=1}^m w_i^2$$

La funzione costo $E(\underline{w})$ che ne deriva è detta RIDGE REGRESSION:

$$E(\underline{w}) = \frac{1}{2} \sum_{i=1}^n (w_i^T \phi(x_i) - t_i)^2 + \frac{\lambda}{2} \underline{w}^T \underline{w} = \frac{1}{2} (\Phi \underline{w} - \underline{y})^T (\Phi \underline{w} - \underline{y}) + \frac{\lambda}{2} \underline{w}^T \underline{w}$$

La sua soluzione è data da: $\underline{w}^* = (\lambda I + \Phi^T \Phi)^{-1} \Phi^T \underline{t}$

Una forma più generale di $E_w(\underline{w})$ è: $E_w(\underline{w}) = \frac{1}{2} \sum_{i=1}^n |w_i|^q$

→ Il caso $q=1$ in letteratura viene chiamato LASSO e favorisce i MODELLI SPARSI, cioè sono modelli in cui parecchi parametri vengono posti pari a zero: se ci pensiamo, è un modo un po' "orribile" di fare feature selection.

Training set vs validation set vs testing set:

- TRAINING SET = insieme di dati utilizzato per fare l'addestramento del predittore.
- VALIDATION SET = insieme di dati utilizzato come supporto per assegnare i valori agli iperparametri.
- TESTING SET = insieme di dati utilizzato per valutare le performance del predittore.

Nel processo di validazione di un modello, prima si addestra il predittore cercando di minimizzare il rischio empirico. Per evitare l'overfitting, si sfrutta il validation set (che comprende dati "nuovi" rispetto a quelli appartenenti al training set) per individuare il valore degli iperparametri che permette di trovare la funzione di predizione che appunto predice al meglio anche i dati nuovi. Infine, si valutano le prestazioni del predittore sul testing set, che comprende dati "nuovi" sia rispetto al training set, sia rispetto al validation set.

Una suddivisione ragionevole del data set iniziale (\equiv tutti i dati che si hanno a disposizione) è:
20-30% per il testing set; 20-30% per il validation set; il resto per il training set.

Una variante di questo meccanismo prevede l'uso del K-FOLD CROSS VALIDATION. Prendiamo ad esempio i solo training set e validation set: l'unica di questi due insiemi viene suddiviso in K parti uguali e si fanno K iterazioni differenti (nella prima il validation set è dato dalla prima porzione e il training set da tutto il resto, e così via). Ciascuna delle K iterazioni dà luogo a un risultato (\rightarrow valore degli iperparametri), e i K risultati vengono mediati fra loro. In tal modo, viene meno la dipendenza del fatto che il data set iniziale viene suddiviso esattamente in un certo modo.

↳ NEL CASO ESTREMO IN CUI $K = \text{CAPACITÀ DEL DATASET}$, L'APPARICO VENIRE LETTO LEAVE-ONE-OUT: A OGNI ITERAZ. UN SOLO ELEMENTO È VALIDATION SET.

20/04/2023

Uso di predittori probabilistici:

Se vogliamo utilizzare un predittore che mi genera in output una distribuzione di probabilità su tutte le classi anziché un valore target, nella soluzioin del predittore migliore, al posto di un set di ipotesi H composto da tante funzioni $h_i()$ esprimibili con dei parametri θ_i , avremo un insieme di distribuzioni di probabilità (magari dello stesso tipo - ad esempio gaussiane) che differiscono tra loro per determinati parametri (e.g. media, varianza).

Ora, supponiamo di avere un insieme di dati X generati tramite una certa distribuzione di probabilità (nota) cui dei parametri (ignoti). Per effettuare una stima dei parametri, si considera la probabili-

ta che venga generato X GIVEN THAT la distribuzione di probabilità abbia dei parametri Θ da un certo set: più questa probabilità (che per ora indichiamo con $p(X|\Theta)$) è alta, più i parametri selezionati sono in linea col data set, migliore è l'approssimazione dei dati stessi.

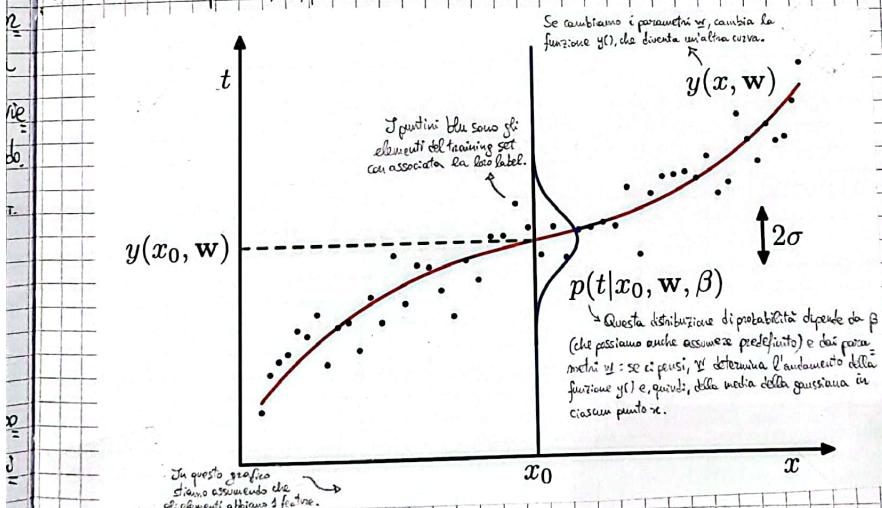
Badiamo che in questo scenario X è fisso e stiamo variando Θ ; se sommiamo tutte le possibili $p(X|\Theta)$ al variare di Θ , non otteniamo necessariamente 1. Di conseguenza, in realtà, non stiamo parlando propriamente di una probabilità, bensì di una VEROSSIMIGUANZA (LIKELIHOOD): $L(X|\Theta)$.

I parametri che approssimano meglio quello della distribuzione che ha generato X sono quelli che massimizzano la likelihood \Rightarrow METODO DELLA MASSIMA VEROSSIMIGUANZA VISTO A CPS.

Trasponiamo ora il discorso sulla regressione. Supponiamo di avere un predittore che prende in considerazione la curva $y(x, w)$, dove x rappresenta un punto elementare (e.g. un punto del training set) e w rappresenta un insieme di parametri che determina proprio com'è fatta la curva. Supponiamo inoltre che il predittore, dato un elemento x_0 , generi una distribuzione di probabilità gaussiana con media pari a $y(x_0, w)$ e varianza $\sigma^2 = \beta^{-1}$ per il valore t del target da associare a x_0 .

Tale distribuzione di probabilità può essere definita in questo modo:

$$p(t|x_0, w, \beta) = \mathcal{N}(t | y(x_0, w), \beta^{-1})$$



Dato un punto x_0 e la rispettiva distribuzione gaussiana $p(t|x_0, w, \beta)$ definita dal nostro predittore, è possibile introdurre la probabilità che il corrispondente punto (un punto blu) venga estratto a caso dalla distribuzione $p(t|x_0, w, \beta)$.

Faccendo così per un certo insieme X di punti e assumendo che tutti gli elementi del dato set X siano mesi in modo indipendente l'uno dall'altro, è possibile definire la probabilità di estrarre il dato set X a partire dalla distribuzione (una, dall'insieme di distribuzioni) $p(t|X, w, \beta)$ (che sono tutte distribuzioni gaussiane con la stessa varianza β^{-1} e cui media dipendente da x) moltiplicando tutte le singole probabilità che seguono estratto il singolo punto. Nel nostro esempio, la verosimiglianza è data proprio da questo prodotto e possiamo indicarla con $L(t|X, w, \beta)$.

Il nostro scopo è sempre ottenere il predittore migliore possibile e lo facciamo massimizzando la likelihood (sempre col metodo della massima verosimiglianza). Tipicamente si fa ciò giocando con w (i.e. calcolando $\arg_{w \in \mathbb{R}^m} L(t|X, w, \beta)$), ma nulla vieta di concentrarsi invece su β .

$$\text{Formalizzando: } L(t|X, w, \beta) = p(t|X, w, \beta) = \prod_{i=1}^n N(t_i | y(x_i; w), \beta^{-1})$$

Per calcolare il punto di massimo, è molto più comodo lavorare coi logaritmi:

$$\text{Log-verosimiglianza} = l(t|X, w, \beta) = \log p(t|X, w, \beta) = \sum_{i=1}^n \log N(t_i | y(x_i; w), \beta^{-1})$$

Facendo un po' di calcoli viene fuori che:

$$l(t|X, w, \beta) = -\frac{\beta}{2} \sum_{i=1}^n \underbrace{(t_i - y(x_i; w))^2}_{\text{TEORIA COSTANTE DI RICHIESTA}} + \frac{m}{2} \log \beta + \text{cost}$$

Se vogliamo massimizzare rispetto a w , nel calcolare la derivata ritroviamo solo questo termine, che è praticamente uguale alla $\tilde{E}(d)$ che abbiamo visto per i predittori deterministici.

21/04/2023

Così come per $E_d(w)$ nel caso di predittori deterministici, anche la likelihood, se presa da sola e massimizzata, può portare all'overfitting.

Una prima soluzione consiste nell'introdurre una funzione di penalità $P(w)$ da sommare alla log-verosimiglianza nella stima dei parametri. Perciò, la funzione da massimizzare diventa:

$$C(w|X) = l(w|X) - P(w)$$

Spesso abbiamo $P(w) = \frac{\gamma}{2} \|w\|^2$, dove γ è un parametro di tuning.

Esiste anche una soluzione più elegante che prevede l'uso della STATISTICA BAYESIANA (o APPROCCIO BAYESIANO).

Statistica bayesiana:

Introduce una nuova nozione di probabilità oltre a quella frequentista (secondo cui probabilità = percentuale delle volte che occurs un evento E su $N \rightarrow \infty$ occorrenze diverse): si tratta di una nozione

SOGGETTIVISTA, in cui definisco la probabilità dell'evento E in base a quanto io ritengo che quell'evento E possa accadere; inoltre, tale definizione di probabilità potrebbe cambiare in base ai dati che osservo.

ESEMPIO: semifinali di andata di Champions League Inter-Milan. Posso stabilire che l'Inter vincerà con probabilità 0,34 (34%) in modo soggettivista, perché non è proprio pensabile far disputare la stessa partita N volte per vedere quante volte vince l'Inter. Non solo: se il passare del tempo si infottrano certi giocatori del Milan, la mia probabilità potrebbe cambiare (\rightarrow cambio della probabilità in base ai dati osservati).

Attenzione: La statistica bayesiana, pur di assegnare un valore di probabilità secca agli eventi, definisce una distribuzione di probabilità da cui può essere estratto un valore di probabilità che un determinato evento accada: in altre parole, la probabilità di un evento è essa stessa una variabile aleatoria.

Riprendendo la logica dell'esempio precedente, prima definire a priori una certa distribuzione di probabilità per un evento E . Dopo di che, a seguito di alcune osservazioni/accadenze, posso ridefinire tale distribuzione di probabilità, che non dipenderà esclusivamente delle osservazioni fatte, ma anche dalle considerazioni fatte a priori, che tireranno la nuova distribuzione di probabilità verso di sé.
 Il massimo della densità di probabilità a posteriori è detto MAXIMUM A POSTERIORI (MAP).

AD ESEMPIO PUÒ ESSERE IL PUNTO CENTRALE DI UNA GAUSSIANA O DI UNA BETA.

NB: è desiderabile avere la a posteriori la stessa densità di probabilità che si aveva a priori (dovrebbero dunque variare solo i parametri).

NB: più il numero di dati osservati aumenta, più la distribuzione a posteriori ha varianza bassa e si discosta da quella a priori. Tuttavia, se ci pensiamo, finché i dati osservati sono relativamente pochi, il contributo della distribuzione a priori porta a fornire un risultato che è meno dipendente dai dati e, quindi, gioca lo stesso ruolo del termine $E_w(\theta)$ nel caso di predizione deterministica (i.e. regressione).

Formalizzando: dati i parametri θ della nostra distribuzione, dobbiamo essere in grado di ottenere la distribuzione a posteriori $p(\theta|X)$ a partire dalla distribuzione a priori $p(\theta)$ dopo aver osservato l'insieme di dati X . A tal proposito basta ricorrere alla formula di Bayes:

$$P(\theta|X) = \frac{P(X|\theta) P(\theta)}{P(X)}$$

$\rightarrow P(X|\theta)$ non è altro che la verosimiglianza.
 $\rightarrow P(X)$ è la somma di tutte le $P(X|\theta)$ al variare di θ .

Poiché $p(X)$ è indipendente da Θ , non è di relativa importanza, per cui possiamo scrivere:

$$p(\Theta|X) \propto p(X|\Theta) p(\Theta)$$

Nel caso in cui gli elementi in X possano essere classificati in modo binario, $p(\text{elen.}|\Theta)$ ha una distribuzione bernoulliana $\Rightarrow P(X|\Theta)$ è il prodotto di n bernoulliane, per cui è una binaria. Stando anche a osservazioni precedenti, ci piacerebbe che $p(\Theta|X)$ e $p(\Theta)$ abbiano la stessa distribuzione; con $p(X|\Theta) \sim$ Biunariale, questo è vero per $p(\Theta|X)$, e $p(\Theta) \sim$ Beta. Infatti, la binariale e la Beta sono due distribuzioni CONIUGATE.

Stima Maximum A Posteriori (MAP):

Assumendo che $X = \text{TRAINING SET } T$, abbiamo che:

$$\begin{aligned} \Theta_{\text{MAP}} &= \underset{\Theta}{\text{argmax}} \quad p(\Theta|T) = \underset{\Theta}{\text{argmax}} \quad p(T|\Theta) p(\Theta) = \underset{\Theta}{\text{argmax}} \quad L(\Theta|T) p(\Theta) = \\ &= \underset{\Theta}{\text{argmax}} \quad L(l(\Theta|T) + \ln p(\Theta)) = \underset{\Theta}{\text{argmax}} \quad \left(\sum_{t \in T} \ln p(t|x_i, \Theta) + \ln p(\Theta) \right) \end{aligned}$$

Tornando alla regressione lineare...

Ricorderemo l'espressione $p(\Theta|X) \propto p(X|\Theta) p(\Theta)$. Nel nostro esempio precedente in cui abbiamo introdotto il metodo della massima verosimiglianza, il nostro predittore genera una distribuzione di probabilità siffatta: $p(t|x, w, \beta) = \mathcal{D}(t|y(x, w), \beta^{-1})$.

Scrivendo l'analogia tra w e Θ , possiamo osservare che qui $p(X|\Theta)$ è di tipo gaussiano.

Ora: qual è quella distribuzione di probabilità che, se moltiplicata per una gaussiana, dà luogo alla medesima distribuzione di probabilità (i.e. è coniugata con la gaussiana)? Sembra la gaussiana! Infatti, in questo caso specifico si utilizza una distribuzione gaussiana per indicare la nostra probabilità a priori.

Poiché stiamo utilizzando la regressione lineare, i nostri parametri w per rappresentare la nostra funzione $y(x, w)$ (i.e. la nostra retta) saranno solo 2: w_0, w_1 . Infatti abbiamo $y = w_0 + w_1 x$.

La distribuzione più neutra possibile per rappresentare una distribuzione gaussiana (a priori) a partire dai parametri w_0, w_1 è la seguente:

$$p(w|\alpha) = \mathcal{N}(w|0, \alpha^{-1} I) = \left(\frac{\alpha}{2\pi} \right)^{\frac{m+1}{2}} e^{-\frac{\alpha}{2} w^T w}$$

una distribuzione gaussiana con media $\Theta = (0, 0)$ e matrice delle covarianze $= \begin{bmatrix} \alpha & 0 \\ 0 & \alpha \end{bmatrix}$.

In definitiva:

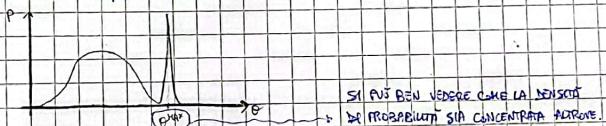
$$\Theta_{\text{MAP}} = \underset{w}{\text{argmax}} \quad p(w|X, t; \alpha, \beta) = \underset{w}{\text{argmax}} \quad p(t|X, w; \beta) p(w|\alpha) =$$

$$\begin{aligned}
 &= \underset{\beta}{\operatorname{argmax}} \prod_{i=1}^m \left(\frac{\sqrt{\beta}}{\sqrt{n}} e^{-\beta(t_i - y(x_i, w))^2} \right) \left(\frac{1}{n} \right)^{\frac{m+1}{2}} e^{-\frac{\alpha}{2} w^T w} = \\
 &= \underset{\beta}{\operatorname{argmax}} \left[-\frac{\beta}{2} \sum_{i=1}^m (t_i - y(x_i, w))^2 + \frac{m}{2} \log \beta - \frac{\alpha}{2} w^T w + \frac{m+1}{2} \log \frac{1}{n} + \text{cost} \right] = \\
 &= \underset{w}{\operatorname{argmin}} \left[\frac{\beta}{2} \sum_{i=1}^m (t_i - y(x_i, w))^2 + \frac{\alpha}{2} w^T w \right] = \underset{w}{\operatorname{argmin}} \left[\frac{1}{2} \sum_{i=1}^m (t_i - y(x_i, w))^2 + \frac{\alpha}{2\beta} \|w\|^2 \right] \cdot \beta
 \end{aligned}$$

1801 7) RICERCA LA REGRESSIONE?

27/04/2023

Negli approcci che abbiamo intradotto per effettuare la migliore predizione probabilistica possibile, sceglievamo un'unica funzione di predizione, che era quella data dai parametri θ che massimizzavano la Likelihood / la probabilità a posteriori (prendevamo quindi la moda della densità di θ). Tuttavia, questo può risultare limitante: da una parte, può essere desiderabile considerare tutta la distribuzione di θ per far entrare in gioco più funzioni di predizione (ciascuna con un suo peso, che magari dipende proprio dalla distribuzione di θ stessa); dall'altra parte, prendere la moda della densità di θ può essere sfavorevole per la scelta del miglior predittore, in particolare molti in distribuzioni di θ come queste:



Per questo motivo, rimanendo nell'ambito della statistica Bayesiana, noi possiamo evitare di considerare ~~una~~ $p(\theta|X)$ per una specifica istanza di θ e per poi "buttarci" il resto della distribuzione, ~~ma~~ possiamo prendere la distribuzione di probabilità generata in output dal nostro predittore mediato su tutta la densità di θ . In parole povere, stiamo prendendo svariate funzioni di predizione con associate le loro densità probabili $p(t|x, \theta)$ e stiamo combinando queste densità ^{probabilisticamente} fra loro in modo ~~a~~ \propto pesato (dove il peso è dato proprio da $p(\theta|X)$). Insomma, stiamo calcolando:

$$\int p(t|x, \theta) \cdot p(\theta|X) d\theta \quad \rightarrow \text{NE: nella statistica Bayesiana, tranne in MAP, ricorrono sempre questi integrali sui parametri e che possono essere integrali a tantissime dimensioni.}$$

Per se avessimo densità di base una stessa (i.e. parametri basata sulla media di $p(\theta|X)$) (TERZO DEL MOMENTO), saremmo riusciti a un integrale di questo genere: $\theta^2 = E_{p(\theta|X)}[\theta^2] = \int \theta^2 p(\theta|X) d\theta$.

L'approccio appena intradotto è detto INTERAMENTE BAYESIANO (FULLY BAYESIAN).

Approccio interamente Bayesiano nella regressione lineare:

La densità di probabilità sul valore y da prevedere per l'elemento x , che viene fuori dalla combinazione di tante densità di probabilità date in output dagli svariati predittori al vertice di w , viene espressa in questo modo:

$$p(y|x, t, \Phi, \alpha, \beta) = \int p(y|z, w, \beta) p(z|t, \Phi, \alpha, \beta) dz$$

VALORE DEL TARGET
TRAINING SET
 β È L'INFERENZA DELLE ERREUR; β^* È LA VARIANZA DELLA GAUSSIANA

ESTRATTA NELLA DISTRIBUZIONE DI PROBABILITÀ DEL TARGET.

Come osservavamo prima, si tratta di un integrale forzagnoso da calcolare. Tuttavia, esiste un caso semplice, che è proprio quello in cui sia $p(t|w, \Phi, \beta)$ (la likelihood), sia $p(w|t)$ (la probabilità a priori) sono gaussiane. Infatti, in tal caso:

$\rightarrow p(y|x, w, \beta)$ (la distribuzione di probabilità del target generata da un singolo predittore) è gaussiana:

$$p(y|x, w, \beta) = \mathcal{N}(y | w^T \phi(x), \beta^*)$$

$\rightarrow p(w|t, \Phi, \alpha, \beta)$ (la probabilità a posteriori di w) è gaussiana:

$$p(w|t, \Phi, \alpha, \beta) = \mathcal{N}(w | \beta S_n \Phi^T t, S_n) \quad \text{dove } S_n = (\alpha I + \beta \Phi^T \Phi)^{-1}$$

\rightarrow anche $p(y|x, t, \Phi, \alpha, \beta)$ è gaussiana: $p(y|x, t, \Phi, \alpha, \beta) = \mathcal{N}(y | m(x), \sigma^2(x))$

$$\text{dove: } m(x) = \beta \phi(x)^T S_n \Phi^T t \quad \sigma^2(x) = \frac{1}{\beta} + \phi(x)^T S_n \phi(x)$$

DA NOTARE CHE L'INCERTITÀ È VERA L'UNICO ELEMENTO CHE CORRISPONDE AL TRAINING SET. IN PARTICOLARE, PIÙ PRECISAMENTE, UN PUNTO CONTIENE TUTTI GLI ELEMENTI DEL TRAINING SET, PIÙ L'INCERTITÀ AUMENTA.

Notiamo che nel fully bayesian la predizione $p(y|x, t, \Phi, \alpha, \beta)$ dipende esclusivamente dal training set (e da α, β), NON dai parametri w : stiamo dunque parlando di un approccio NON PARAMETRICO. \rightarrow UN ALTRO ESEMPIO DI PREDIZIONE PARAMETRICA È IL K-NEAREST NEIGHBORS.

VANTAGGIO DEGLI APPROCCI NON PARAMETRICI: non richiedono l'utilizzo di un modello, che desira da delle scelte arbitrarie (degli iperparametri, del set di ipotesi, e così via).

Svantaggio degli approcci non parametrici: non effettuano alcun tipo di apprendimento preliminare per cui, quando devono effettuare una predizione, scandiscono / utilizzano l'intero training set (che talvolta può risultare svantaggioso dal punto di vista delle performance).

28/04/2023

REGRESSIONE NON PARAMETRICA

Ricordiammo l'output della regressione per l'approccio interamente bayesiano: $p(y|x, t, \Phi, \alpha, \beta)$.

Nel caso in cui sia gaussiano, il suo valore più probabile è dato dalla sua media $m(x)$ (che qui individuiamo con $y(x)$):

$$y(x) = \beta \phi(x)^T S_n \Phi^T t = \sum_{i=1}^n \beta \phi(x)^T S_n \phi(x_i) t_i$$

Questo fatto qui rappresenta il peso che viene fissato confrontando l'elemento a cui si riferisce il target t_i (i.e. x_i) con l'elemento di cui vogliamo effettuare la predizione (i.e. x). In pratica il peso determina quanto t_i contribuisce a definire il valore target da predire, e sarà tanto più alto quanto più x, x_i sono simili tra loro.

Possiamo indicare tale fatto con $K(x, x_i)$:

$$y(\mathbf{z}) = \sum_{i=1}^n K(\mathbf{z}, \mathbf{x}_i) t_i$$

La funzione per $K(\mathbf{z}, \mathbf{x}_i)$ è anche detta KERNEL EQUIVALENTE. Il Kernel equivalente è un caso particolare di FUNZIONE KERNEL, che è un tipo di funzione che a ogni coppia di punti associa un valore.

$$\text{COVARIANZA TRA } y(\mathbf{x}) \text{ E } y(\mathbf{z}): \text{ cov}(y(\mathbf{x}), y(\mathbf{z})) = \text{cov}(\phi(\mathbf{x})^T \mathbf{w}, \phi(\mathbf{z})^T \mathbf{w}) = \phi(\mathbf{x})^T S_{11} \phi(\mathbf{z}) = \frac{1}{\beta} K(\mathbf{x}, \mathbf{z})$$

Kernel regression:

Il Kernel equivalente è una funzione che deriva dai dati del training set in modo automatico. Nei metodi di Kernel regression, in realtà, possiamo anche definirci noi una funzione Kernel $K_h(\mathbf{z})$, in cui \mathbf{z} può essere vista come la distanza (la "differenza") tra due punti e h può essere visto come la BANDWIDTH, ovvero come un parametro che determina l'ampiezza di $K_h(\mathbf{z})$ (più la funzione è ampia, più considero anche i punti lontani). Chiaramente ha senso che $K_h(\mathbf{z})$ sia massima in 0 e tendenzialmente decresca man mano che ci si allontana da 0.

$$\text{Una } K_h \text{ molto comune è il KERNEL GAUSSIANO (o RBF): } g(\mathbf{z}) = e^{-\frac{\|\mathbf{z}\|^2}{2h^2}}$$

N.B.: PIÙ H DECRESCE, PIÙ RISCHIO L'OVERTANDING.
AL CRESCERE DI H, LA SAMMA CON PUNTITA ELABORATA DIVENTA PIÙ AMPIA.

NADARAYA-WATSON:

È un predittore nell'ambito della Kernel regression che mira a calcolare il valore medio del target t dato uno specifico elemento x per effettuare la sua predizione. Indichiamo il suo output con $f(x)$:

$$f(x) = E[t|x] = \int p(t|x) t dt = \int \frac{p(x,t)}{p(x)} t dt = \frac{\int p(x,t) t dt}{\int p(x,t) dt}$$

La distribuzione congiunta $p(x,t)$ rappresenta la probabilità di estrarre l'elemento x E di avere il target t per l'elemento estratto. Possiamo dunque approssimarla così:

$$p(x,t) \approx \frac{1}{n} \sum_{i=1}^n K_h(x - x_i) K_h(t - t_i)$$

N.B.: Nel quanto riguarda l'elemento x_i , più è vicino ai punti del training set, più è probabile estrarre (probabilità dei punti vicini al centro) l'elemento x (quello che avevo già appena dato dal training set (centro)). Farò la stessa cosa col target t .

$$\Rightarrow f(x) \approx \frac{\int \frac{1}{n} \sum_{i=1}^n K_h(x - x_i) K_h(t - t_i) t dt}{\int \frac{1}{n} \sum_{i=1}^n K_h(x - x_i) K_h(t - t_i) dt} = \frac{\sum_{i=1}^n K_h(x - x_i) \int K_h(t - t_i) t dt}{\sum_{i=1}^n K_h(x - x_i) \int K_h(t - t_i) dt}$$

Se assumiamo che $K_h(x)$ sia una distribuzione di probabilità con media 0, abbiamo che:

$$\cdot \int K_h(t - t_i) dt = 1$$

$$\cdot \int K_h(t - t_i) t dt = t_i$$

$$\Rightarrow f(x) \approx \frac{\sum_{i=1}^n K_h(x - x_i) t_i}{\sum_{i=1}^n K_h(x - x_i)}$$

Possiamo vedere anche come una combinazione lineare di tutti i valori target pesati mediante le funzioni Kernel.

$$\text{Infatti, ponendo } W_i(x) := \frac{K_h(x - x_i)}{\sum_{j=1}^n K_h(x - x_j)}, \text{ possiamo scrivere:}$$

$$f(x) \approx \sum_{i=1}^n W_i(x) t_i$$

N.B.: qui la regressione non è lineare: non è un po' più complicata.

LOCALLY WEIGHTED REGRESSION (LOESS):

È un predittore nell'ambito della Kernel regression che in realtà utilizza un approccio parametrico per trovare i parametri w migliori per la funzione $\hat{f}(x)$ che verrà usata per effettuare le predizioni. Qui, a differenza degli altri approcci parametrici, viene utilizzata una funzione Kernel con l'idea che la retta a seguito delle predizioni non sia fissa, bensì dipenda dallo specifico elemento da prevedere: di fatto la retta viene scelta in base alla sua distanza (w) funzione loss) dagli elementi del training set più vicini a x , senza avere troppi curvi degli elementi più lontani da x . Di conseguenza, strettamente, invece di minimizzare il rischio empirico, che è dato dalla media aritmetica delle funzione loss su tutti gli elementi del training set, si punta a minimizzare una funzione $L(x)$, che è data dalla media pesata della funzione loss sulla base dei pesi $K_i(x)$, dove $K_i(x) = K_i(x - x_i)$.

Sappiamo che LOESS prende in considerazione la quadratic loss, abbiamo che:

$$L(x) = \sum_{i=1}^n K_i(x)(w^\top \bar{x}_i - t_i)^2 = \sum_{i=1}^n K_i(x - x_i)(w^\top \bar{x}_i - t_i)^2$$

La minimizzazione di $L(x)$: $\hat{w}(x) = \underset{w}{\operatorname{argmin}} \sum_{i=1}^n K_i(x)(w^\top \bar{x}_i - t_i)^2$

ha una soluzione chiusa: $\hat{w}(x) = (\bar{X}^\top \Psi(x) \bar{X})^{-1} \bar{X}^\top \Psi(x) t$

dove $\Psi(x)$ è una matrice diagonale $n \times n$ con $\Psi(x)_{ii} = K_i(x)$.

Poi la predizione viene effettuata come nel caso della regressione lineare: $y = \hat{w}(x)^\top \bar{x}$

LOCAL LOGISTIC REGRESSION:

È un predittore nell'ambito della Kernel regression analogo a LOESS, con l'unica differenza che $L(x)$ è definita come una media pesata della funzione cross entropy anziché della quadratic loss (in effetti è un predittore applicabile per la classificazione):

$$L(x) = \sum_{i=1}^n K_i(x - x_i) (t_i \log p_i - (1-t_i) \log (1-p_i)) \quad \text{dove } p_i = \text{valore predetto.}$$

Processi gaussiani:

PERCHÉ CI PIACCIONO LE DISTRIBUZIONI GAUSSIANE?

Sia $x = (x_1, \dots, x_n)^\top$ un vettore elettronico gaussiano con $p(x) = \mathcal{N}(\mu, \Sigma)$, e sia $x = (x_A, x_B)$ una relativa partizione: $x_A = (x_1, \dots, x_n)^\top$; $x_B = (x_{n+1}, \dots, x_m)^\top$.

Allora: $p_A(x_A) = \mathcal{N}(\mu_A, \Sigma_A)$, $p_B(x_B) = \mathcal{N}(\mu_B, \Sigma_B)$ e:

$$\mu = (\mu_A, \mu_B)^\top, \quad \Sigma = \begin{bmatrix} \Sigma_A & \Sigma_{AB} \\ \Sigma_{BA} & \Sigma_B \end{bmatrix} \quad \rightarrow \text{ogni matrice è nulla di così particolare.}$$

Analogalemente: $p(x_A | x_B) = \mathcal{N}(\mu_{AB}, \Sigma_{AB})$, $p(x_B | x_A) = \mathcal{N}(\mu_{BA}, \Sigma_{BA})$

i. Elettronici
P.
I. SOTTIVETRI
DI SE SONTA
ZERO NELL'
GAUSSIANI.
NON SONO I FAMIGLIARES DELLA
DISTRIBUZIONE.
ANCHE LE DENSITÀ CONSIDERATE
NELL'ARCO SONO GAUSSIANE.

$$\text{dove: } \begin{aligned} \mu_{AB} &= \mu_A + \sum_B \sum_B^{-1} (x_B - \mu_B) \\ \Sigma_{AB} &= \Sigma_A - \sum_B \sum_B^{-1} \Sigma_B \end{aligned} \quad \begin{aligned} \mu_{BA} &= \mu_B + \sum_A \sum_A^{-1} (x_A - \mu_A) \\ \Sigma_{BA} &= \Sigma_B - \sum_A \sum_A^{-1} \Sigma_A \end{aligned}$$

Ecco, un processo gaussiano non è altro che un'estensione dei vettori gaussiani a infinite componenti (i.e. a infinite r.a. gaussiane). Noi nella nostra trattazione vedremo i processi gaussiani come "seguenti" da cui estrapolare dei qualsiasi vettori gaussiani.

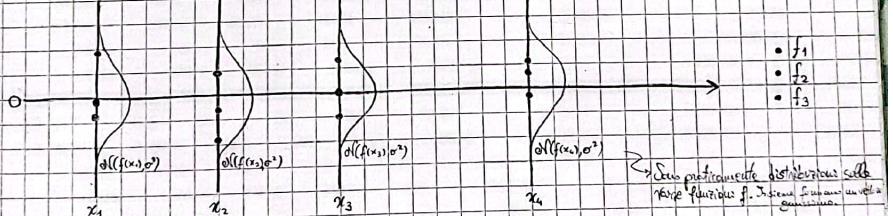
DISTRIBUZIONI DI PROBABILITÀ SULLE FUNZIONI CON DOMINI FINITI!

Sia $X = (x_1, \dots, x_m)$ un vettore di m punti in \mathbb{R}^d e sia \mathcal{F} un insieme di funzioni $f: \mathbb{R}^d \rightarrow \mathbb{R}$ che hanno X come dominio. Queste funzioni possono essere espresse come vettori di m valori reali: $y = (y_1, \dots, y_m)$, dove $y_i = f(x_i)$. Vale anche il viceversa: ciascun vettore di m valori reali rappresenta univocamente una funzione f definita nel dominio X (\leadsto c'è una corrispondenza 1-a-1 tra funzioni f e vettori y).

Se definiamo una distribuzione di probabilità $p(y)$ sui vettori in \mathbb{R}^m , siamo automaticamente definito una distribuzione di probabilità $p(f)$ sulle funzioni appartenenti ad \mathcal{F} .

Se inoltre assumiamo che $p(y)$ (o equivalentemente $p(f)$) sia una distribuzione gaussiana multivaria, con media θ e matrice di covarianza diagonale ($\sigma^2 I$), siamo praticamente supponendo che ciascun valore $y_i = f(x_i)$ abbia una distribuzione normale con media θ e varianza σ^2 (dove tutti i valori sono indipendenti) e abbiamo:

$$p(f|X; \sigma^2) = \prod_{i=1}^m \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(f(x_i) - \theta)^2}{2\sigma^2}}$$



→ SPOILER: $p(f|X; \sigma^2)$ risulta essere una distribuzione di probabilità a priori rispetto alla conoscenza dei tangentie degli elem. del training set.

* NULA MI VIETAVA DI AVERE UNA MATRICE DI COVARIANZA QUALUNQUE (Σ).

02/05/2023

Se prendiamo $p(f|X; \sigma^2)$ come una distribuzione di probabilità a priori, allora possiamo dire che:

DISTRIBUZIONE DI PROBABILITÀ A POSTERIORI = $p(f|X, t, \beta, \sigma^2) \propto p(f|X; \sigma^2) p(X, t | f, \beta)$,

dove $p(X, t | f, \beta)$ è la VEROSIMIGLIANZA, e possiamo vederela proprio come quella verosimiglianza vis.

mentre trattiamo la regressione lineare (\rightarrow l'insieme dei punti del data set delle le distribuzioni gaussiane $d^0(t_i | x_i, w, \beta)$ che lo potevano generare); in tal caso abbiamo che:

$$p(X, t | f, \beta) \propto \prod_{i=1}^n d^0(t_i | f(x_i), \beta) \quad \text{NB: } p \text{ in maiuscola}$$

$$\Rightarrow p(f | X, t, \beta, \sigma^2) \propto \prod_{i=1}^n d^0(t_i | f(x_i), \beta) \cdot p(f | \sigma^2) \quad \text{A qui trarriremo}$$

N.B.: questa probabilità a posteriori, se ci pensiamo, alla fine indica la probabilità che la funzione f dia in output un determinato valore target per l'elemento x_i : GIVEN THAT il suo valore target effettivo è t_i : non ha molto senso come cosa... dobbiamo introdurre qualcosa di altro.

DISTRIBUZIONI DI PROBABILITÀ (GAUSSIANE) SULLE FUNZIONI CON EDOMINI INFINITI:

Supponiamo ora di avere un dominio X di infiniti punti. Ad questo punto caratterizzare una funzione f in modo esauriente non è fattibile (almeno tramite un rettore di valori). Ma, fortunatamente, per i nostri scopi è sufficiente estrarre di volta in volta un sottoinsieme X finito di punti a partire da X e definire il valore di f esclusivamente per i punti appartenenti a X .

Ecco, ora siamo in grado di definire un **PROCESSO GAUSSIANO** come un processo stocastico (i.e. una collezione di variabili aleatorie indipendenti) tale che, per ogni sottoinsieme finito di punti $X = \{x_1, \dots, x_n\}$ e sottratto da X , i valori $f(x_1), \dots, f(x_n)$ ^{forniscono} ~~hanno~~ una distribuzione congiuntamente gaussiana.

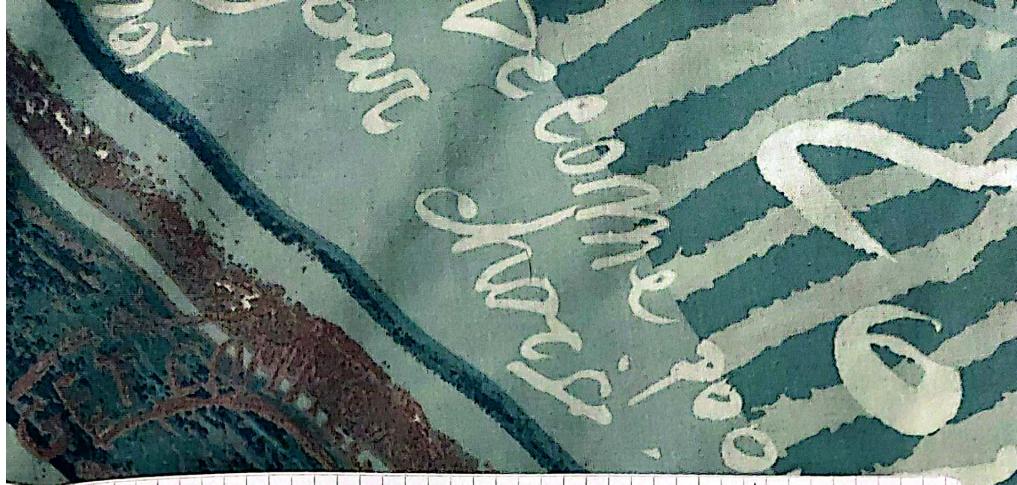
Stavolta, per definire la nostra distribuzione di probabilità (gaussiana) a priori per i valori $y_1, \dots, y_n = f(x_1), \dots, f(x_n)$ a partire da un qualunque sottoinsieme finito di punti $X = \{x_1, \dots, x_n\}$, seguiamo le seguenti due regole:

- La media deve essere 0: $\mu(X) = 0$.
- La matrice di covarianza non è più diagonale, il che implica che stiamo introducendo una correlazione tra gli elementi in X ; in particolare, la correlazione è espressa in termini di funzione Kernel $K(x)$, zero in termini di "vicinanza" fra i punti. Una matrice di covarianza così definita è detta **MA-**

TRICE DI GRAM:

$$\Sigma(X) = G = \begin{bmatrix} K(x_1, x_1) & K(x_1, x_2) & \dots & K(x_1, x_n) \\ K(x_2, x_1) & K(x_2, x_2) & \dots & K(x_2, x_n) \\ \vdots & \vdots & \ddots & \vdots \\ K(x_n, x_1) & K(x_n, x_2) & \dots & K(x_n, x_n) \end{bmatrix}$$

L'effetto di una matrice di covarianza siffatta è che, se $f(x_1) = y_1$, allora $f(x') = y'$ con y' molto vicino a y_1 . $\forall x'$ molto vicino a $x_1 \Rightarrow$ se abbiamo X molto grande, è anche possibile rappresentare f come una funzione continua.



NB: più la funzione Kernel è ampia, più sono i punti che dipendono da x_i , più la funzione varia in maniera "dolce" (lentamente).

A questo punto siamo in grado di fare la seguente cosa. Supponiamo di avere quattro punti nel training set x_1, x_2, x_3, x_4 , e supponiamo di conoscere la distribuzione congiunta (gaussiana) $p(f(x_1), f(x_2), f(x_3), f(x_4))$. Se vogliamo effettuare una predizione su un nuovo elemento x , anzitutto sappiamo calcolare $p(f(x_1), f(x_2), f(x_3), f(x_4), f(x))$; da qui, come sappiamo, siamo in grado di ricavare anche $p(f(x)|f(x_1), f(x_2), f(x_3), f(x_4))$. Per avere la predizione su x , ovvero la distribuzione di probabilità dei target associabili a x (i.e. la distribuzione di probabilità delle funzioni f), ci basta avere la distribuzione a posteriori di f rispetto ai target del x_1, x_2, x_3, x_4 (i.e. $p(f(x_1), \dots, f(x_4) | X, t)$). \rightarrow Questo permette di avere solo le funzioni che passano per t_1, t_2, t_3, t_4 all'interno della distribuzione.

04/05/2023

Anzitutto nel formale, consideriamo il training set X e l'elemento x su cui fare la predizione.

Supponendo di stare nell'ambito dei processi gaussiani, la distribuzione congiunta $(t, f(x))$ è una distribuzione gaussiana multivariata con media $\mu(X, x)$ e matrice di covariogramma $\Sigma(X, x)$, dove:

$$\mu(X, x) = (\mu(X), \mu(x))^T \quad \Sigma(X, x) = \begin{bmatrix} \Sigma(X) & \Sigma(x, X) \\ \Sigma(x, X)^T & \Sigma(x, x) \end{bmatrix} \quad \text{con:}$$

$$\Sigma(x, X) = (K(x, x_1), K(x, x_2), \dots, K(x, x_n))^T, \quad \Sigma(x, x) = K(x, x)$$

Di conseguenza, la distribuzione data come output dalla del nostro predittore è una distribuzione gaussiana fatta in questo modo: $m_p(y|X, f) = \mu(x) + \Sigma(x, X) \Sigma(X)^{-1} (t - \mu(x))$

$$\sigma^2 = \Sigma_p(x, x) = K(x, x) - \Sigma(x, X) \Sigma(X)^{-1} \Sigma(x, X)^T$$

La vicinanza del predittore che abbiamo appena visto è la NO NOISE, che prevede che qualunque funzione f all'interno della distribuzione output del predittore passi esattamente per i punti (x_i, t_i) del training set (per cui nei punti del training set l'incertezza è zero).

GAUSSIAN NOISE:

È la variante che assume che il target t_i assegnato al punto del training set x_i non è un punto esatto ma un punto APPROSSIMATIVAMENTE corretto: il nostro predittore, per i punti x_i del training set, predice $t_i + \epsilon$, dove ϵ è un errore gaussiano (\rightarrow in tal modo il predittore è più robusto).

Dal punto di vista delle formule, è tutto essenzialmente uguale a prima con la differenza che la



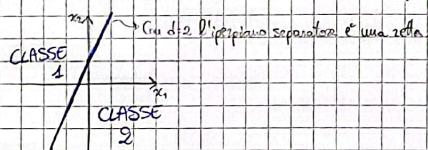
matrice di covarianza Σ della distribuzione (gaussiana) a priori per i valori $y_1, \dots, y_n = f(x_1), \dots, f(x_n)$
 è così definita:

$$\hat{\Sigma}(X) = \begin{bmatrix} K(x_1, x_1) + \sigma_f^2 & K(x_1, x_2) & \cdots & K(x_1, x_n) \\ K(x_2, x_1) & K(x_2, x_2) + \sigma_f^2 & \cdots & K(x_2, x_n) \\ \vdots & \vdots & \ddots & \vdots \\ K(x_n, x_1) & K(x_n, x_2) & \cdots & K(x_n, x_n) + \sigma_f^2 \end{bmatrix}$$

3) A

CLASSIFICAZIONE LINEARE

Se consideriamo un insieme di d dimensioni (dove $d = \#$ feature) in cui vengono rappresentati gli elementi, lo scopo della classificazione lineare è trovare uno spazio di dimensione $1 \leq d-1$ che rappresentano delle separazioni tra le diverse classi (un iper piano per la classificazione binaria, due iper piani per la classificazione ternaria, e così via). Nel caso di classificazione binaria con due feature abbiamo:



Funz
Qui

→ Due classi si dicono LINEARMENTE SEPARABILI se esiste un iper piano che separa lo spazio degli elementi in due sottospazi S_1, S_2 tali che in S_1 si trovano tutti e solo gli elementi della classe 1 e in S_2 si trovano tutti e solo gli elementi della classe 2.

Due
Siam
g w

→ TUTTAVIA, IL PIÙ DELLE VOLTE NON SI PUÒ ESSERE PROPRIO IN GRADO DI STABILIRE SE DUE CLASSI SONO LINEARMENTE SEPARABILI.

→ Un training set si dice LINEARMENTE SEPARABILE se esiste un iper piano che separa il training set in due sottoinsiemi I_1, I_2 tali che in I_1 si trovano tutti e solo gli elementi del training set della classe 1 e in I_2 si trovano tutti e solo gli elementi del training set della classe 2.

Applicazione alla classificazione:

1) FUNZIONE DISCRIMINANTE: è l'applicazione non probabilistico e consiste nel trovare una funzione $f: X \rightarrow \{1, \dots, K\}$ che mappa ciascun input x su una qualche classe C_i . È possibile fare ciò trovando gli iperpiani che suddividono lo spazio di punti in K classi (sottospazi) differenti.

Funz
Qui
L'el
L'E

2) APPROCCIO DISCRIMINATIVO: è composto da due fasi. Nella FASE DELL'INFERENZA si determina la probabilità condizionata $p(C_j | x)$ (\rightarrow si determina la distribuzione di probabilità sui valori target per l'elemento x). Nella FASE DECISIONALE, in base alla distribuzione di probabilità ottenuta, si stabilisce quale dovrà essere la classe da predire per l'elemento x .

3) APPROCCIO GENERATIVO: Utilizza la formula di Bayes per determinare $p(C_j | x)$:

$$p(C_j | x) = \frac{p(x | C_j) p(C_j)}{p(x)} \rightarrow \text{C'è riferimento a } p(x | C_j) = p(x) \text{ perché, in ultima istanza, ci interessa solo confrontare } p(C_j | x) \text{ e } p(x | C_j) \text{ tra loro. Ad esempio col posteriorio, è più si semplifica.}$$

- $p(C_j)$ = probabilità che, estraiendo un qualunque elemento, questo appartenga alla classe C_j . (\rightarrow probabilità a priori rispetto alla conoscenza dell'elemento).

- $p(C_j | x)$ = probabilità che, estraiendo esattamente l'elemento x , questo appartenga alla classe C_j . (\rightarrow probabilità a posteriori rispetto alla conoscenza dell'elemento).

- $p(x | C_j)$ = distribuzione di probabilità sui valori delle feature per la classe C_j .

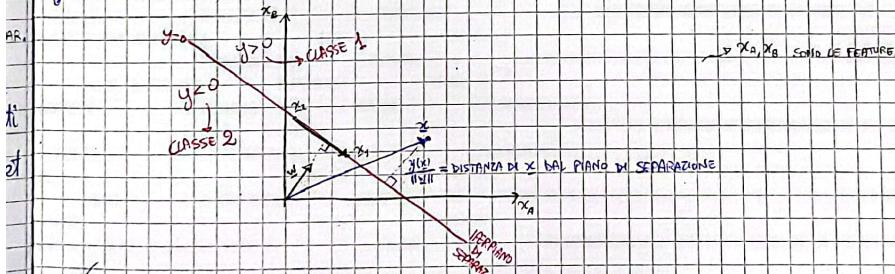
In pratica, un classificatore che usa l'approccio generativo (che è fatto CLASSIFICATORE BAYESIANO) calcola $p(C_j)$ (tipicamente col rapporto $\frac{|C_j|}{|I_{all}|}$) e $p(x | C_j)$ (dando una caratterizzazione α degli elementi di ciascuna classe) per effettuare poi le predizioni.

Funzioni discriminanti lineari nella classificazione binaria:

Qui viene calcolato l'iperpiano di separazione risolvendo l'equazione $y(x) = w^T \cdot x + w_0 = 0$ dove i parametri w descrivono proprio l'iperpiano.

Siano x_1, x_2 due punti sull'iperpiano $\Rightarrow y(x_1) = y(x_2) = 0 \Rightarrow y(x_1) - y(x_2) = 0 \Rightarrow$

$$w^T \cdot x_1 + w_0 - w^T \cdot x_2 - w_0 = w^T \cdot (x_1 - x_2) = 0 \Rightarrow x_1 - x_2 \perp w$$



Questa funzione y determina a quale classe potrebbe appartenere ciascun elemento x (in base al segno dell'output). Più il modulo dell'output è elevato, più il classificatore è 'sicuro' della sua predizione.

Funzioni discriminanti lineari nella classificazione multiclasse:

Qui si definiscono K funzioni lineari (una per ogni classe): $y_i(x) = w_i^T \cdot x + w_{i0} \quad 1 \leq i \leq K$

L'elemento x viene assegnato alla classe $C_k \Leftrightarrow y_k(x) > y_j(x) \quad \forall j \neq k$ SOLO SEMPRE LA REGIONE LIGA A UNA CLASSE

È POSSIBILE DEFINIRE LE REGIONI DI SPARZO RELATIVE ALLE K CLASSE. IN QUESTO CASO DUE LINEE, TRA L'ALTRO, LE REGIONI STESO SONO CORTESESE

05/05/2023

Funzioni discriminanti generalizzate:

Come nel caso della regressione lineare, è possibile introdurre le funzioni base per generalizzare l'espressione di $y()$:

$$y(x) = w_0 + \sum_{i=1}^n w_i \phi_i(x)$$

Un esempio è dato dalle funzioni discriminanti generalizzate:

$$y(x) = w_0 + \sum_{i=1}^n w_i x_i + \sum_{i=1}^n \sum_{j=1}^n w_{ij} x_i x_j$$

Uso della regressione per le funzioni discriminanti lineari:

È possibile vedere il discorso accennato in "Funzioni discriminanti lineari nella classificazione multiclasso" come una scomposizione del problema di classificazione in K problemi di regressione più semplici.

Si considera una matrice T con m righe (una per ogni elemento del training set) e K colonne (una per ogni classe) che rappresenta tutti gli elementi del training set con la notazione $\mathbb{1}_K$: se il primo elemento del training set appartiene alla seconda classe, allora la prima riga di T ha un 1 in seconda posizione e uno 0 in tutte le altre. A questo punto, per ogni colonna di T (i.e. per ogni classe) si definisce una diversa funzione $y_i()$, che ad esempio può essere lineare: $y_i = w_i^T \cdot x + w_{0i}$.

È possibile così definire un vettore $y = [y_1, \dots, y_K]$ che, dato un particolare elemento x , determina gli output di tutte le funzioni $y_i()$. Da y si può ottenere un altro vettore z che ha la notazione $\mathbb{1}_K$: la sua unica componente pari a 1 è quella che corrisponde al valore massimo all'interno di y . $\rightarrow z$ sta praticamente esprimendo la classe c da predire per l'elem. x , dove: $c = \arg \max_i y_i(x)$.

Un modo compatto per esprimere y è:

$$y(x) = W \cdot x = \begin{bmatrix} w_{01} & w_{11} & \dots & w_{n1} \\ w_{02} & w_{12} & \dots & w_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ w_{0K} & w_{1K} & \dots & w_{nK} \end{bmatrix} \cdot \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

* NB: non per forza z sia
solo: come abbiamo già detto,
la scelta del valore da partire
è una scelta successiva.

Questo meccanismo funziona perché la regressione è uno strumento che fornisce la stima del target dato l'input x ($E[z|x]$); $y_i(x)$ può essere visto come la media condizionata $E[z_i|x]$.

Ma ciascuno z_i possiamo vedercelo come una r.v. binomiale. Allora:

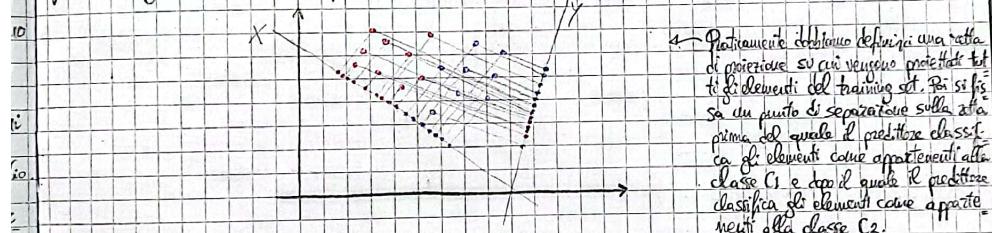
$$y_i(x) \approx E[z_i|x] = P(Z_i=1|x) \cdot 1 + P(Z_i=0|x) \cdot 0 = P(Z_i=1|x) = P(C_i|x)$$

→ Seppur $y_i(x)$ è una stima di $P(C_i|x)$, non è essa stessa una probabilità, perché può assumere anche valori al di fuori dell'intervallo $[0, 1]$.

Ciascuna riga della matrice W (i.e. ciascun vettore w_i per uno dei K sottoproblemi di regressione) può essere ottenuta minimizzando la somma dei quadrati delle differenze tra y_{ij} e t_j , $\forall j=1, \dots, D$, dove le y_{ij} sono i valori predetti dalla nostra i -esima funzione di regressione su tutti gli elementi del training set, mentre le t_j sono i target vero degli elementi del training set. Tra l'altro, la W ha una soluzione in forma chiusa: $W = (\bar{X}^T \bar{X})^{-1} \bar{X}^T T$, dove \bar{X} è la matrice delle feature degli elementi del training set che ha una prima colonna aggiuntiva composta da tutti 1.

Fisher Linear discriminant:

Detto anche Linear Discriminant Analysis (LDA), è un approccio che prevede la RIDUZIONE DELLA DIMENSIONALITÀ, ovvero porta a lavorare su d' dimensioni con $d' < d$, dove d è il numero delle feature degli elementi. Più precisamente, l'idea è la seguente (supponendo che $d=2$):



Nel modo insieme si vede bene come la retta della retta X sia decisamente migliore e meno distorsiva rispetto alla retta Y ; anzi, è ottimale perché minimizza il numero di elementi che si "accavallano".

Se ci fai caso, la retta ottimale sarebbe l'iperspazio separatore.

Diamo ora un'idea più formale di "separare il più possibile i punti del training set sulla retta di proiezione": Significa semplicemente fare in modo che le proiezioni dei baricentri degli insiemi di punti relativi alle varie classi siano le più distanti possibili (dove il baricentro di una classe è il punto "medio" di tutti gli elementi del data set appartenenti a quella classe). Se $m_1 = \#$ elementi di classe C_1 , $m_2 = \#$ elementi di classe C_2 , i baricentri m_1, m_2 possono essere definiti così:

$$\underline{m}_1 = \frac{1}{n_1} \sum_{x \in C_1} x \quad \underline{m}_2 = \frac{1}{n_2} \sum_{x \in C_2} x$$

Massimizzare la distanza tra le proiezioni di \underline{m}_1 e \underline{m}_2 vuol dire massimizzare la seguente quantità:

$$m_2 - m_1 = W^T (m_2 - m_1)$$

Attenzione: se vogliamo trovare il W che massimizza $W^T (m_2 - m_1)$ senza specificare altro,

stiamo inizialmente facendo in modo che il modulo (e quindi la norma) di w cresca indefinitamente.
Ma noi vogliamo semplicemente trovare la direzione giusta per w (i.e. per la retta di separazione),
e non abbiamo vincolo sul modulo! Di conseguenza, è necessario introdurre un vincolo che terra
fissa la lunghezza di w , come ad esempio $\|w\| = 1$. Il problema di ottimizzazione dunque è:

$$\max_{w \in \mathbb{R}^n} w^T(m_2 - m_1) \quad \text{where } \|w\| = 1$$

Si dimostra che un problema di ottimizzazione con vincoli può essere convertito in un altro proble
ma di ottimizzazione senza vincoli perfettamente equivalente. Vediamo prima un esempio stupid.

EQUIVALENTE $\max_x f(x) \quad \text{where } x \leq 3 \Leftrightarrow x - 3 \leq 0$
 $\max_w f(w) + \lambda(w - 3)$ $\quad \rightarrow \lambda$ è detto MULTIPLICATORE LAGRANGIANO.

→ Di conseguenza, il nostro problema di ottimizzazione diventa:

$$\max_{w \in \mathbb{R}^n} w^T(m_2 - m_1) + \lambda(1 - w^T w)$$

→ Massimizzare questa funzione significa porre a 0 il suo gradiente rispetto a w e anche
che la sua derivata rispetto a λ .

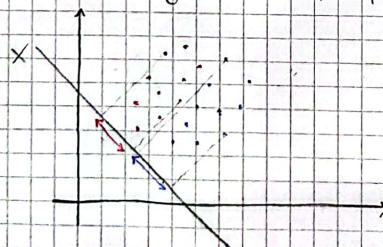
$$\frac{\partial}{\partial w} (w^T(m_2 - m_1) + \lambda(1 - w^T w)) = m_2 - m_1 + 2\lambda w = 0 \Rightarrow w = \frac{m_2 - m_1}{2\lambda}$$

$$\frac{\partial}{\partial \lambda} (w^T(m_2 - m_1) + \lambda(1 - w^T w)) = 1 - w^T w = 0 \Rightarrow \lambda = \frac{(m_2 - m_1)^T(m_2 - m_1)}{2} = \frac{\|m_2 - m_1\|^2}{2}$$

In definitiva: $w = \frac{m_2 - m_1}{\|m_2 - m_1\|^2}$ → se ci pensi, corrisponde proprio alla direzione del segmento che va da m_1 a m_2 .

C'è un grande MA: l'approccio così descritto funziona solo se gli elementi di ciascuna classe
sono raggruppati in modo omogeneo (i.e. formano un cerchio e non, ad esempio, una forma
allungata). In caso contrario, il baricentro non è rappresentativo dell'insieme di punti.

Consideriamo dunque il caso in cui gli elementi sono disposti in una forma allungata:



→ Se prendiamo una retta di proiezione come X, le proiezioni degli elementi di S sulla
classe Sono i raggruppate tra loro (cioè hanno
una distanza minima) → in casi come
questo, c'è minore probabilità che elementi
di classi diverse si accavallino sulla stessa
di proiezione.

Questo è un altro fattore da prendere in
considerazione quando selezioniamo la
retta di proiezione.

Di conseguenza, a questo punto, ci piacerebbe massimizzare una funzione che:

- 1) Cresca all'aumentare della distanza tra i barricchii di classi diverse.
- 2) Decresca all'aumentare della dispersione delle proiezioni degli elementi di una stessa classe.

Il punto (2) (i.e. la dispersione) lo esprimiamo come la somma dei quadrati delle distanze fra la proiezione di ciascun elemento della classe e la proiezione del baricentro della stessa classe. Tale somma è detta VARIANZA WITHIN-CLASS S_i^2 :

$$S_i^2 = \sum_{x \in C_i} (x^T w - m_i)^2$$

La VARIANZA WITHIN-CLASS TOTALE è definita come $S_1^2 + S_2^2$ (nel caso di classif. binaria).

Per definire la funzione da massimizzare risulta essere: $J(w) = \frac{(m_2 - m_1)^2}{S_1^2 + S_2^2}$

Poiché la massimizzazione deve avvenire rispetto a w , facciamo in modo che $J(w)$ venga espressa in termini di w .

→ MATRICE DI COVARIANZA WITHIN-CLASS: $S_i = \sum_{x \in C_i} (x - m_i)(x - m_i)^T \quad i=1,2$

→ MATRICE DI COVARIANZA WITHIN-CLASS TOTALE: $S_{\text{tot}} = S_1 + S_2$

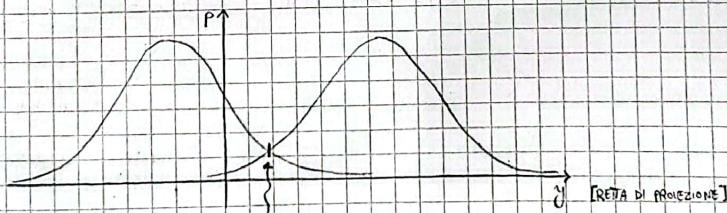
→ MATRICE DI COVARIANZA BETWEEN-CLASS: $S_B = (m_2 - m_1)(m_2 - m_1)^T$

$$\Rightarrow J(w) = \frac{w^T S_B w}{w^T S_{\text{tot}} w}$$

Nel risolvere l'equazione $\frac{\partial}{\partial w} \frac{w^T S_B w}{w^T S_{\text{tot}} w} = 0$, otteniamo: $w = S_{\text{tot}}^{-1} (m_2 - m_1)$

Ora resta da rispondere a un'altra domanda: come definiamo il punto di separazione tra le due classi sulla retta di proiezione?

Possiamo definire le distribuzioni di probabilità degli elementi appartenenti a ciascuna classe sulla retta di proiezione $\rightarrow p(y|C_1), p(y|C_2)$



Come punto di separazione possiamo prendere questo in cui $p(y|C_1), p(y|C_2)$ sono uguali (il punto di intersezione fra le due distribuzioni di probabilità).

09/05/2023

Perceptron:

È il terzo tipo di funzione discriminante che vediamo e prevede che ciascun elemento si venga classificato sulla base del segno di $w^T \cdot x$:

$$y(x) = \text{sgn}(w^T \cdot x)$$

Se $y(x) = 1$, il classificatore predice che $x \in C_1$, mentre se $y(x) = -1$, il classificatore predice che $x \in C_2$.

→ È EVIDENTE CHE IL PERCEPTRON SIA STATO PENSATO SOLO PER LA CLASSIFICAZ. BINARIA.

La funzione loss che definiamo qui generalmente si chiama cross-entropy loss che, dato un elemento x_i , vale 0 se x_i è classificato correttamente e vale $-y_i \ln t_i + (1-y_i) \ln(1-t_i)$ se x_i è classificato male.

Se M è l'insieme degli elementi classificati male, allora la funzione costo complessiva è:

$$E_p(w) = -\sum_{x_i \in M} w^T \cdot x_i t_i$$

Il minimo di $E_p(w)$ può essere trovato mediante la discesa del gradiente.

BATCH GRADIENT DESCENT:

$$w^{(t+1)} = w^{(t)} - \eta \frac{\partial E_p(w)}{\partial w}$$

$$\Rightarrow w^{(t+1)} = w^{(t)} + \eta \sum_{x_i \in M} x_i t_i$$

$$\text{dove } \frac{\partial E_p(w)}{\partial w} = -\sum_{x_i \in M} x_i t_i$$

dove $M = \text{insieme dei punti classificati male quando il parame. è } w^{(t)}$

Infatti, poiché cambia il vettore dei coefficienti w a ogni iterazione, cambia anche il piano di separazione.

STOCHASTIC GRADIENT DESCENT:

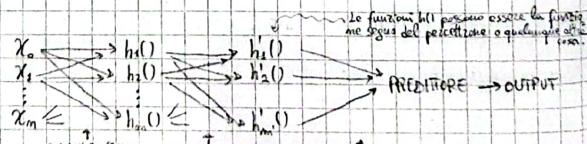
$$w^{(t+1)} = w^{(t)} + \eta x_i t_i$$

PROBLEMA: La funzione costo così definita NON tiene conto degli elementi classificati bene: è vero che a ogni iterazione tende a diminuire il contributo degli elementi mal classificati ma, modificando w solo in funzione di quelli, rischia di introdurre nuovi contributi di elementi che prima erano classificati bene. In tal modo, la funzione costo (e quindi la "bontà" del classificatore) rischia di oscillare all'infinito, senza convergere e senza far mai terminare l'algoritmo della discesa del gradiente in un numero di passi finito.

→ L'unico caso in cui si ha la garanzia che la discesa del gradiente termini in tempo finito senza oscillare è quello in cui il training set è linearmente separabile.

MULTILAYER PERCEPTRONS:

Il perceptron è alla base delle reti neurali.



11/05/2023
MODelli GENERATIVI

Modello di linguaggio:

È una distribuzione di probabilità di termini su un vocabolario di termini.

PERCHÉ INTRODURLO? → Ci è molto utile quando ad esempio vogliamo classificare dei documenti come SFAM / NON SFAM e, appunto, vogliamo utilizzare un approccio generativo per effettuare la classificazione;
RIA. In tal caso, come sappiamo, ci si calcola $p(x|C_i)$, ovvero la probabilità che esca un documento in tal caso della classe C_i , senza, fuori proprio il documento x . Un'esemplificazione molto forte ma molto semplice da affinare consiste nel considerare tutti i termini appartenenti a un documento indipendenti tra loro, per cui il problema si riduce nel calcolare la probabilità che esca il singolo termine (così ci si calcola la probabilità del documento come il prodotto delle probabilità di tutti i termini che lo compongono) → DOCUMENTO VISTO COME INSIEME ("BAG") DI TERMINI (E NON COME SEQUENZA).

Formalizzando, sia $T = \{t_1, \dots, t_n\}$ l'insieme di termini che accorrono in una collezione C di documenti, e sia N la somma delle lunghezze di tutti i documenti in C . Inoltre, $\forall i = 1, \dots, n$, sia m_i la molteplicità del termine t_i all'interno di C . Un modello di linguaggio può essere definito come una distribuzione associata al vettore di probabilità $\hat{\phi} = (\hat{\phi}_1, \dots, \hat{\phi}_n)^T$, dove $\hat{\phi}_i = p(t_i|C)$. Dunque, la probabilità di esporre il termine t_i dal dizionario può essere stimata mediante il metodo della massima verosimiglianza applicato al caso discreto:

$$\hat{\phi}_i = p(t_i|C) = \frac{m_i}{\sum_{k=1}^n m_k} = \frac{m_i}{N}$$

S. PROBLEMA: in tal modo stiamo assumendo che i termini che non compaiono in C abbiano una probabilità nulla di essere estratti \Rightarrow stiamo considerando l'osservazione di un nuovo documento contenente uno o più di questi termini come un evento impossibile (PARADOSSO DI BLACK SWAN).
SOLUZIONE: assegnare in qualche modo una probabilità molto piccola, ma non nulla, all'osservazione dei termini che non compaiono mai in C \rightarrow questa soluzione è chiamata SMOOTHING.

SMOOTING DI LAPLACE:

Si definisce un parametro $\alpha^{(2)}$ tale per cui il numeratore di $\hat{\phi}_i$ sia $m_i + \alpha$. Di conseguenza, il denominatore deve essere normalizzato in modo appropriato in modo tale che $\sum_i \hat{\phi}_i = 1$. In tal modo, $\sum_i \hat{\phi}_i = 0$.

SMOOTHING DI DIRICHLET:

L'idea è partire da una distribuzione di probabilità a priori (e.g. uniforme) per l'occorrenza dei termini t_j . Dopo che si effettua una distribuzione di probabilità a posteriori a seguito dell'osservazione dei documenti in C , quest'ultima distribuzione non potrà mai priorizzare elementi a probabilità nulla per effetto della distribuzione a priori.

Alla solita, ci piacerebbe che la probabilità a priori e la probabilità a posteriori abbiano la medesima distribuzione (che caratterizza la likelihood passiamo a prendere quella di DIRICHLET (vedi COMPLEMENTI sul retro del quaderno), con parametro $\alpha = (\alpha, \alpha, \dots, \alpha)$). In tal modo, alla fine otteniamo: $p(t_j | C_k) = \frac{m_{j,k} + \alpha}{m_k + \alpha N}$, dove $N = \dim(\text{vocabolario})$.

Poiché vogliamo l'esemplificazione per cui la probabilità che compare un termine t_j in un documento è C_k sia indipendente dalla probabilità che compare t_j nel m -esimo documento?

Semplicissimo: tutti i possibili documenti di dimensione m (t_1, t_2, \dots, t_m) sono di una quantità enorme (esponenziale) ma, tipicamente, è data se hanno una quantità di documenti molto inferiore (anche di diversi ordini di grandezza); in pratica siamo d'accordo che in generale non abbiamo abbastanza dati a disposizione per stimare una probabilità del tipo:

$$p(d | C_k) = p(t_1, t_2, \dots, t_m | C_k) = p(t_1 | C_k) p(t_2 | t_1, C_k) \cdot \dots \cdot p(t_m | t_1, t_2, \dots, t_{m-1}, C_k)$$

Ecco che allora preferiamo scrivere: $p(d | C_k) = \prod_{i=1}^m p(t_i | C_k)$

NB: l'indipendenza tra l'occorrenza dei termini è data solo SAPENDO la classe C_k a cui appartiene il documento d ma NON in assoluto.

ESEMPPIO CRETINO: stiamo assumendo che, sapendo che d è un documento SPAM, se osserviamo la parola 'esso', non cambia la probabilità di osservare anche la parola 'viagra'.

Tuttavia, se non sappiamo di che tipologia è il documento d e occorre la parola 'esso', potrebbe azzardarsi la probabilità che d è SPAM e, quindi, si altra anche la probabilità di osservare il termine 'viagra'.

ORA IN GENERALE SI ASSUME CHE IL VALORE DELLE SEMPRE PIÙ INFLUENTI DATA LA CLASSE C_k .

↳ I classificatori che sfruttano queste considerazioni sono detti **NAIVE BAYES**.

Ora, questi classificatori, oltre al calcolo di $p(d | C_k)$, prevedono anche il calcolo di $P(C_k)$. Ma stando a facile, basta usare la maximum likelihood: $P(C_k) = \frac{|C_k|}{|C_1| + |C_2|}$

In definitiva, questi classificatori lasceranno la loro predizione sul calcolo di

$$\frac{\prod_i p(t_i | C_1) p(C_1)}{\prod_i p(t_i | C_2) p(C_2)} = \frac{p(d | C_1) p(C_1)}{p(d | C_2) p(C_2)} = \frac{p(C_1 | d)}{p(C_2 | d)}$$

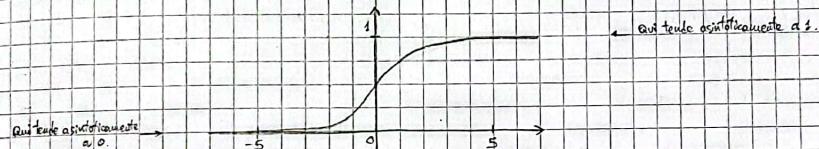
Se ci facciamo caso, abbiamo che:

$$p(C_1 | x) = \frac{p(x | C_1) p(C_1)}{p(x | C_1) p(C_1) + p(x | C_2) p(C_2)} = \frac{1}{1 + \frac{p(x | C_2) p(C_2)}{p(x | C_1) p(C_1)}}$$

Definiamo $\alpha := \log \left[\frac{p(x | C_1) p(C_1)}{p(x | C_2) p(C_2)} \right] = \log \left[\frac{p(C_1 | x)}{p(C_2 | x)} \right] \rightarrow \alpha \text{ è detto LOG ODDS}$

$$\Rightarrow p(C_1 | x) = \frac{1}{1 + e^{-\alpha}} = \sigma(\alpha) \quad ; \quad p(C_2 | x) = 1 - \frac{1}{1 + e^{-\alpha}} = \frac{e^{-\alpha}}{1 + e^{-\alpha}}$$

$\rightarrow \sigma(x)$ è la cosiddetta FUNZIONE LOGISTICA (o SIGMOIDE).



PROPRIETÀ DELLA SIGMOIDE:

$$1) \sigma(-x) = 1 - \sigma(x)$$

$$2) \frac{d\sigma(x)}{dx} = \sigma(x)(1 - \sigma(x))$$

Tutto il discorso è facilmente estensibile al caso con $K > 2$ classi:

$$p(C_k | x) = \frac{p(x | C_k) p(C_k)}{\sum_j p(x | C_j) p(C_j)}$$

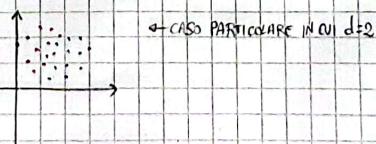
$$\forall K \text{ definiamo } \alpha_k(x) := \log(p(x | C_k) p(C_k)) = \log p(x | C_k) + \log p(C_k)$$

$$\Rightarrow p(C_k | x) = \frac{e^{\alpha_k}}{\sum_i e^{\alpha_i}} =: s(\alpha_k)$$

$\rightarrow s(x)$ è la cosiddetta FUNZIONE SOFTMAX (o ESPONENZIALE NORMALIZZATA).

Gaussian discriminant analysis (GDA):

Ottiamo un ultimo da parte il discorso sui termini e sui documenti e reconsideriamo la classificazione dei punti in \mathbb{R}^d :



A partire dai punti del dataset appartenenti a ciascuna classe, possiamo definire le distribuzioni di probabilità $p(x|C_k)$, che indica come sono distribuiti gli elementi della classe C_k . Nella GAUSSIAN DISCRIMINANT ANALYSIS si assume che $p(x|C_k)$ sia gaussiana e che abbia sempre la stessa matrice delle covarianze Σ (di dimensione $d \times d$) per tutti i C_k :

$$P(x|C_k) = \frac{1}{(2\pi)^{d/2} |\Sigma|^{1/2}} e^{-\frac{1}{2}(x-\mu_k)^T \Sigma^{-1} (x-\mu_k)}$$

→ LA MEDIA μ_k È PARI AL BARICENTRO DELL'INSIEME DI PUNTI RELATIVI ALLA CLASSE C_k .

→ LA MATRICE DI COVARIANZA Σ DIFERisce INVECE DAL DATASET PRESENTE PER INTERO.

CASO DI CLASSIFICAZIONE BINARIA:

Come abbiamo visto, $p(C_1|x) = \sigma(a(x))$ dove:

$$\begin{aligned} a(x) &= \log \frac{p(x|C_1)p(C_1)}{p(x|C_2)p(C_2)} = \log p(x|C_1) - \log p(x|C_2) + \log \frac{p(C_1)}{p(C_2)} = \\ &= -\frac{1}{2}(x-\mu_1)^T \Sigma^{-1} (x-\mu_1) + \frac{1}{2}(x-\mu_2)^T \Sigma^{-1} (x-\mu_2) + \log \frac{p(C_1)}{p(C_2)} = \\ &= \frac{1}{2}(\mu_2^T \Sigma^{-1} \mu_2 - x^T \Sigma^{-1} \mu_2 - \mu_1^T \Sigma^{-1} x) - \frac{1}{2}(\mu_1^T \Sigma^{-1} \mu_1 - x^T \Sigma^{-1} \mu_1 - \mu_1^T \Sigma^{-1} x) + \log \frac{p(C_1)}{p(C_2)} = \\ &= \frac{1}{2}(\mu_2^T \Sigma^{-1} \mu_2 - \mu_1^T \Sigma^{-1} \mu_1) + (\mu_1^T \Sigma^{-1} - \mu_2^T \Sigma^{-1}) x + \log \frac{p(C_1)}{p(C_2)} \end{aligned}$$

Se vogliamo scrivere $a(x)$ nella forma $w^T x + w_0$, abbiamo che:

$$w = \Sigma^{-1}(\mu_2 - \mu_1) ; \quad w_0 = \frac{1}{2}(\mu_2^T \Sigma^{-1} \mu_2 - \mu_1^T \Sigma^{-1} \mu_1) + \log \frac{p(C_1)}{p(C_2)}$$

↳ Qui l'unico iperpiano di separazione è dato dalla condizione $p(C_1|x) = p(C_2|x)$ \Leftrightarrow

$$\sigma(a(x)) = \sigma(-a(x)) \xrightarrow{\text{Ogni Membra}} a(x) = -a(x) \Leftrightarrow a(x) = 0 \Leftrightarrow w^T x + w_0 = 0$$

12/05/2023

CASO DI CLASSIFICAZIONE MULTICLASSE:

Come abbiamo visto, $p(C_k|x) = \sigma(a_{k,x}(x))$ dove:

$$\begin{aligned} a_{k,x}(x) &= \log(p(x|C_k)p(C_k)) = \log p(x|C_k) + \log p(C_k) = \\ &= -\frac{1}{2}(x-\mu_k)^T \Sigma^{-1} (x-\mu_k) - \frac{1}{2}\log(2\pi) - \frac{1}{2}\log|\Sigma| + \log p(C_k) = \\ &= \frac{1}{2}(\mu_k^T \Sigma^{-1} x + x^T \Sigma^{-1} \mu_k - \mu_k^T \Sigma^{-1} \mu_k) - \frac{1}{2}\log(2\pi) - \frac{1}{2}\log|\Sigma| + \log p(C_k) \end{aligned}$$

↳ Anche questo possiamo scriverlo come $w_k^T x + w_{0,k}$

↳ Qui gli iperpiani di separazione sono dati dal caso in cui \exists due classi C_j, C_k con la stessa probabilità a posteriori e con probabilità maggiore rispetto a qualunque altra classe:

$$P(C_i|x) = p(C_i|x) \wedge P(C_i|x) < p(C_k|x) \quad \forall i \neq j, k \iff$$

$$e^{a_i(x)} = e^{a_j(x)} \wedge e^{a_i(x)} < e^{a_k(x)} \quad \forall i \neq j, k \iff$$

$$a_x(x) = a_j(x) \wedge a_i(x) < a_k(x) \quad \forall i \neq j, k$$

Vediamo ora il caso in cui le distribuzioni gaussiane $p(x|C_k)$ hanno matrice di covarianza diversa.

CASO DI CLASSIFICAZIONE FINITRINA:

$$P(C_i|x) = \phi(a_i(x)) \quad \text{dove} \quad a_i(x) = \log \frac{p(x|C_1)p(C_1)}{p(x|C_2)p(C_2)} =$$

$$= -\frac{1}{2}(x - \mu_i)^T \sum_i^{-1} (x - \mu_i) + \frac{1}{2}(x - \mu_2)^T \sum_2^{-1} (x - \mu_2) + \frac{1}{2} \log \frac{|C_1|}{|C_2|} + \log p(C_1)$$

→ Questa espressione viene posta pari a zero per trovare l'iperpiano di separazione; si ottiene così un'equazione (eventualmente) quadratica, per cui l'iperpiano di separazione è al più quadrattico.

GDA è maximum Likelihood:

Le distribuzioni condizionali $p(x|C_k)$ possono essere derivate dal training set mediante il metodo della massima verosimiglianza.

Per semplicità, assumiamo $K=2$ e che entrambe le classi condividano la medesima matrice di covarianza Σ . Allora, sarà necessario stimare $\mu_1, \mu_2, \Sigma, p(C_1)=\pi$.

A questo punto supponiamo anche che il training set T includa n elementi (x_i, t_i) con:

$$t_i = \begin{cases} 0 & \text{se } x_i \in C_2 \\ 1 & \text{se } x_i \in C_1 \end{cases}$$

$$\cdot \text{Se } x \in C_1 \Rightarrow p(x|C_1) = p(x|C_2)p(C_1) = \mathcal{N}(x|\mu_1, \Sigma)$$

$$\cdot \text{Se } x \in C_2 \Rightarrow p(x|C_2) = p(x|C_1)p(C_2) = (1-\pi)\mathcal{N}(x|\mu_2, \Sigma)$$

Allora la Likelihood del training set T è data da:

$$L(\pi, \mu_1, \mu_2, \Sigma | T) = \prod_{i=1}^n (\pi \mathcal{N}(x_i|\mu_1, \Sigma))^{t_i} ((1-\pi) \mathcal{N}(x_i|\mu_2, \Sigma))^{1-t_i} \Rightarrow$$

$$\text{LOG LIKELIHOOD} = l(\pi, \mu_1, \mu_2, \Sigma | T) = \sum_{i=1}^n (t_i \log \pi + t_i \log \mathcal{N}(x_i|\mu_1, \Sigma)) +$$

$$+ \sum_{i=1}^n ((1-t_i) \log (1-\pi) + (1-t_i) \log \mathcal{N}(x_i|\mu_2, \Sigma))$$

DERIVATA RISPETTO A π :

$$\frac{\partial l}{\partial \pi} = \frac{2}{n} \sum_{i=1}^n (t_i \log \pi + (1-t_i) \log (1-\pi)) = \sum_{i=1}^n \left(\frac{t_i}{\pi} - \frac{1-t_i}{1-\pi} \right) = \frac{m_1}{\pi} - \frac{m_2}{1-\pi} \quad \text{dove } m_1 = |C_1|$$

$$\frac{\partial l}{\partial \pi} = 0 \iff \pi = \frac{m_1}{m_1+m_2} = \frac{m_1}{n}$$

DERIVATA RISPETTO A μ_1, μ_2 :

La massimizzazione della log-Likelihood rispetto a μ_1 (i.e. $\frac{\partial L}{\partial \mu_1} = 0$) la si ha per:

$$\mu_1 = \frac{1}{m_1} \sum_{x_i \in C_1} x_i \quad \rightarrow \text{Questo è il baricentro di } C_1.$$

Analogamente, la massimizzazione della log-Likelihood rispetto a μ_2 (i.e. $\frac{\partial L}{\partial \mu_2} = 0$) la si ha per:

$$\mu_2 = \frac{1}{m_2} \sum_{x_i \in C_2} x_i$$

DERIVATA RISPETTO A Σ :

La massimizzazione della log-Likelihood rispetto a Σ (i.e. $\frac{\partial L}{\partial \Sigma} = 0$) la si ha per:

$$\Sigma = \frac{m_1}{m} S_1 + \frac{m_2}{m} S_2 \quad \text{dove:} \quad S_1 = \frac{1}{m_1} \sum_{x_i \in C_1} (x_i - \mu_1)(x_i - \mu_1)^T$$

$$S_2 = \frac{1}{m_2} \sum_{x_i \in C_2} (x_i - \mu_2)(x_i - \mu_2)^T$$

Proposizione:

La proprietà per cui $p(C_k | x)$ è un modello lineare generalizzato con una funzione di attivazione data da una sigmoida (nel caso binario) o da una softmax (nel caso multiclass) vale altrettanto in generale e non solo quando modelliamo $p(x|C_k)$ come una gaussiana o una bernoulliana. In particolare, è sufficiente che $p(x|C_k)$ abbia la seguente forma:

$$p(x|C_k) = \frac{1}{S} g(\Theta_k) f\left(\frac{x}{S}\right) e^{-\mu_k(x)} \quad \rightarrow \text{FAMIGLIA DI ESPONENZIALI}$$

dove $\mu(x)$ è una funzione che mappa x su un array m -dimensionale (e m è proprio la dimensione dell'array Θ_k di parametri).

La maggior parte delle distribuzioni che conosciamo appartiene a questa famiglia.

MODELLO DISCRIMINATIVO

Analizzeremo due:

1) REGRESSIONE LOGISTICA \rightarrow fa uso della sigmoida $\sigma(w^T x + w_0)$

2) REGRESSIONE SOFTMAX \rightarrow fa uso della softmax $S(w_j^T x + w_{j0}) \quad \forall j=1, \dots, K$

VEDERMO DURANTE
MODelli LINEARI
GENERALIZZATI

Modelli lineari generalizzati (GLM):

In generale sono funzioni $y()$ siffatte: $y(x) = f(w^T x + w_0)$, dove f (che è detta RE-SPONSE FUNCTION) è in generale una funzione non lineare.

\rightarrow Il luogo dei punti tali che $y(x) = c$ è detto ISOSUPERFICIE. Si può vedere immediatamente

che: $\mathbf{w}^T \mathbf{x} + w_0 = f(y) = C \rightarrow$ le isosuperficie di un gcl sono tutte iper-piani.

Svantaggi degli approcci discriminativi:

- Permettono di derivare meno informazioni rispetto agli approcci discriminativi: ad esempio, non deriviamo $p(x|C_i)$ e, di conseguenza, non consentono di generare nuovi dati (nuovi elementi associati a una classe C_i). ↳ CREAZIONE CHE AFFONDO RICHIEDE LA CONOSCENZA DI $p(x|C_i)$

Nantagesi degli approcci discriminativi:

- Sono approcci più semplici e richiedono la derivazione di un numero di parametri molto minore: Supponiamo ad esempio di utilizzare la maximum likelihood per la stima dei parametri (ma ovviamente possono ci vuole di ricorrere a MAP o a un approccio interamente bayesiano); nel caso degli approcci discriminativi, si ha semplicemente $\mathbf{w}^T \mathbf{x} + w_0$, dove si vede bene che i parametri sono $d+1$; nel caso degli approcci generativi, supponendo di avere due sole classi e di avere $\Sigma_1 = \Sigma_2 \equiv \Sigma$, i parametri da stimare sono $d+d+d\binom{d+1}{2}$ (rispettivamente dati da μ_1, μ_2 e Σ , che è una matrice $d \times d$ simmetrica).

- Forniscono delle predizioni migliori, soprattutto nel caso in cui le assunzioni che si fanno su $p(x|C_i)$ negli approcci generativi sono vere/erate (vedi la distribuz. di $p(x|C_i)$).

Regressione logistica:

Assume per ipotesi che i target degli elementi associati a una stessa classe abbiano una distribuzione bernoulliana $\Rightarrow p(C_i|x) = \sigma(\mathbf{w}^T \mathbf{x}) = \frac{1}{1+e^{-\mathbf{w}^T \mathbf{x}}} \equiv y(x)$

Segue la stessa logica dell'utilizzo della regressione per definire un classificatore, solo che qui le $y(x)$ sono delle vere e proprie probabilità e non delle stime di probabilità.

MAXIMUM LIKELIHOOD PER LA STIMA DEI PARAMETRI:

Poiché i target degli elementi associati a una stessa classe hanno una distribuzione bernoulliana, abbiamo che: $p(t_i|x_i, \mathbf{w}) = p_i^{t_i} (1-p_i)^{1-t_i}$ dove $p_i = p(C_i|x_i) = \sigma(\mathbf{w}^T \mathbf{x}_i) \Rightarrow$

$$\text{LIKELIHOOD} = L(\mathbf{w}|X, t) = \prod_{i=1}^n p(t_i|x_i, \mathbf{w}) = \prod_{i=1}^n p_i^{t_i} (1-p_i)^{1-t_i} \Rightarrow$$

$$\text{LOG-LIKELIHOOD} = l(\mathbf{w}|X, t) = \sum_{i=1}^n (t_i \log p_i + (1-t_i) \log(1-p_i)) \quad \leftarrow \text{QUEST'ESPRESSIONE È DETTA CROSS-ENTROPY}$$

$$\Rightarrow \frac{\partial l(\mathbf{w}|X, t)}{\partial \mathbf{w}} = \sum_{i=1}^n (t_i - p_i) \mathbf{x}_i = \sum_{i=1}^n (t_i - \sigma(\mathbf{w}^T \mathbf{x}_i)) \mathbf{x}_i \quad \rightarrow \text{Poiché } \sigma \text{ è una funzione NON lineare, in genere non riusciamo a trovare il punto di massimo}$$

Ricorriamo dunque all'ascensione del gradiente (il dual della discesa). \Leftarrow annullando semplicemente queste quantità.

$$\mathbf{w}^{(1)} = \mathbf{w}^{(0)} + \alpha \frac{\partial l(\mathbf{w}|X, t)}{\partial \mathbf{w}} \Big|_{\mathbf{w}^{(0)}} = \mathbf{w}^{(0)} + \alpha \sum_{i=1}^n (t_i - \sigma((\mathbf{w}^{(0)})^T \mathbf{x}_i)) \mathbf{x}_i = \mathbf{w}^{(0)} + \alpha \sum_{i=1}^n (t_i - y(\mathbf{x}_i)) \mathbf{x}_i$$

NB: al solito, è possibile applicare la funzione base agli elementi x_i ed è possibile introdurre la regolarizzazione per prevenire l'overfitting (ad esempio aggiungendo un termine per la ridge regression o la lasso regressione all'interno dell'espressione della likelihood).

A
2) E
3) F

Regressione Softmax:

Estende l'approccio della regressione logistica al caso $K > 2$ e definisce una matrice W dei coefficienti del modello di dimensione $(d+1) \times K$, dove $W = (w_1, \dots, w_K)$ e w_j è il vettore dei coefficienti $(d+1)$ -dimensionale per la classe C_j .

REG
1) S
t.
2)
3)

MATRIX LIKELIHOOD PER LA STIMA DEI PARAMETRI:

Se T è la matrice $m \times K$ che rappresenta i target t_{ij} degli elementi del training set in formato \bar{x}_i , allora abbiamo:

$$\text{LIKELIHOOD} = p(T|X, W) = \prod_{i=1}^n \prod_{j=1}^K p(c_j|x_i^{t_{ij}}) = \prod_{i=1}^n \prod_{j=1}^K \left(\frac{e^{w_j^T \bar{x}_i}}{\sum_{k=1}^K e^{w_k^T \bar{x}_i}} \right)^{t_{ij}} \Rightarrow$$

QUESTO SOLO FATTORE È EGUAL A $p(c_j|x_i^{t_{ij}})$, MENTRE TUTTI GLI ALTRI VALGONO 0 (POICÉ HANNO $t_{ij}=0$, I.E. $x_i \notin C_j$).

$$\Rightarrow \text{LOG-LIKELIHOOD} = \ell(W) = \sum_{i=1}^n \sum_{j=1}^K t_{ij} \log \left(\frac{e^{w_j^T \bar{x}_i}}{\sum_{k=1}^K e^{w_k^T \bar{x}_i}} \right) \Rightarrow$$

$$\Rightarrow \frac{\partial \ell(W)}{\partial w_j} = \sum_{i=1}^n (t_{ij} - y_{ij}) \bar{x}_i = \sum_{i=1}^n (t_{ij} - S(w_j^T \bar{x}_i)) \bar{x}_i$$

16/05/2023
Altri tipi di regressione:

Sia y la variabile aleatoria che descrive la distribuzione dei possibili valori target (\rightarrow dovrà poi essere l'output del modello predittore) e sia \bar{x} un vettore di variabili aleatorie che descrive le distribuzioni delle feature degli elementi. Abbiamo già analizzato questi tre casi:

- $p(y|\bar{x})$ GAUSSIANA \rightarrow REGRESSIONE LINEARE.
- $p(y|\bar{x})$ BERNOLLIANA \rightarrow REGRESSIONE LOGISTICA.
- $p(y|\bar{x})$ CATEGORICA \rightarrow REGRESSIONE SOFTMAX.

Il discorso è facilmente estendibile a ulteriori casi; devono solo valere le seguenti 3 ipotesi:

- 1) $p(y|\bar{x}) = \frac{1}{S} g(\Theta(\bar{x})) f\left(\frac{y}{S}\right) e^{-\frac{1}{S} \Theta(\bar{x})^T U(y)}$
- 2) $\forall \bar{x}$ deve essere predetto $E[U(y)|\bar{x}]$.
- 3) $\Theta(\bar{x}) = W^T \cdot \bar{x}$

REGRESSIONE POISSONIANA:

- 1) Sia $y \in \mathbb{N}$ e sia $p(y|\bar{x}) = \frac{\lambda(\bar{x})^y}{y!} e^{-\lambda(\bar{x})}$ una distribuzione di Poisson con parametro $\lambda(\bar{x})$.

Allora definiamo $\Theta(x) = \log \lambda(x)$ e $U(y) = y$.

$$2) E[U(y)|x] = E[y|x] = \lambda(x) = e^{\Theta(x)}$$

$$3) \text{Funzione di precisione: } C_{\theta(x)} = e^{W^T x}$$

REGRESSIONE ESPONENZIALE:

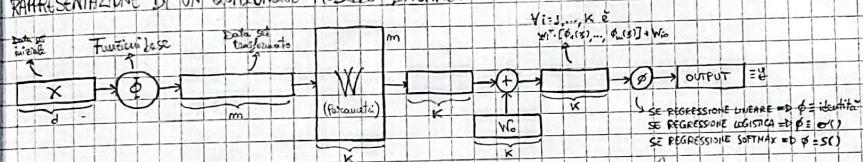
1) Sia $y \in [0, \infty)$ e sia $p(y|x) = \lambda(x) e^{-\lambda(x)y}$ una distribuzione esponenziale con parma
to $\lambda(x)$. Allora definiamo $\Theta(x) = -\lambda(x)$ e $U(y) = y$.

$$2) E[U(y)|x] = E[y|x] = \frac{1}{\lambda(x)} = -\frac{1}{\Theta(x)}$$

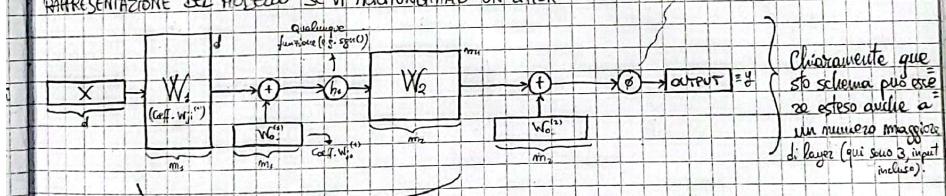
$$3) \text{Funzione di precisione: } -\frac{1}{\Theta(x)} = -\frac{1}{W^T x}$$

PERCEPTRONI MULTILAYER

RAFFRESENTAZIONE DI UN QUALSIASI MODELLO LINEARE:



RAFFRESENTAZIONE DEL MODELLO SE VI AGGIUNGHIAMO UN LAYER:



Questa porzione dello schema sostituisce l'applicazione delle funzioni base. La differenza tra i due casi è che nel caso precedente le funzioni base sono prefissate, mentre qui si utilizza un apposito parametrico (tanto è vero che in più si hanno i parametri W_1, v_0).

Lo schema appena introdotto non è altro che una RETE NEURALE.

Primo layer:

Per un vettore di input d -dimensionale $x = (x_1, \dots, x_d)$, il primo layer di una rete neurale deriva m_1 valori (detti ATTIVAZIONI) $a_1^{(1)}, \dots, a_{m_1}^{(1)}$ mediante un'opportuna combinazione lineare di x_1, \dots, x_d :

$$a_j^{(1)} = \sum_{i=1}^d w_{ji}^{(1)} x_i + b_j^{(1)} = w_j^{(1)} \cdot x$$

Ciascuna attivazione $a_j^{(1)}$ viene trasformata mediante una FUNZIONE DI ATTIVAZIONE $h_1()$ (in generale non lineare): così, il primo layer fornisce in output un vettore $z^{(1)} = (z_1^{(1)}, \dots, z_{m_1}^{(1)})^T$:

$$z_j^{(t)} = h_t(a_j^{(t)}) = h_t(w_j^{(t)} \cdot z)$$

ESEMPI DI h_t :

1) SIGMOIDE: $h_t(x) = \sigma(x) = \frac{1}{1+e^{-x}}$

2) TANGENTE HIPERBOLICA: $h_t(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{1}{1+e^{-x}} - \frac{1}{1+e^{-x}} = \sigma(x) - \sigma(-x)$

3) RELU: $h_t(x) = \max\{0, x\}$

18/05/2023

3 layer networks:

Se guardiamo tutti gli strati insieme in una rete a 3 livelli (layer di input, layer intermedio, layer di output), supponendo che l'ultimo strato utilizzi la funzione Softmax $\phi() \equiv \sigma$ per effettuare la predizione, abbiamo che l'output y_k relativo alla classe C_k è pari a:

$$y_k = \sigma \left(\sum_{j=1}^{m_1} w_{kj}^{(2)} \cdot h_2 \left(\sum_{i=1}^{n_1} w_{ji}^{(1)} \cdot x_i + b_j^{(1)} \right) + b_k^{(2)} \right) \quad \text{NOTARE CHE IN QUEST'ESPRESSIONE GLI STRATI SONO UNO DENTRO L'ALTRO.}$$

Di fatto, qui l'apprendimento di TUTTI i parametri avviene in modo contestuale: in altre parole, il REPRESENTATION LEARNING (che è il meccanismo di apprendimento dei parametri per la trasformazione degli elementi — i.e. dei parametri dei primi #LAYER - 1 = 2 strati) è causale alla precisione reale e propria.

L'ultimamente sta prendendo piede il TRANSFER LEARNING, che sfrutta l'appuccio duale: separa l'apprendimento della trasformazione degli elementi dall'apprendimento dei parametri delle funzioni di predizione. In tal modo, per ogni task predittivo, si considerano i parametri dei più bassi #LAYER - 1 = 2 strati e si effettua il learning dei parametri relativi all'ultimo layer. Chiaramente è un approccio molto più veloce ma potrebbe penalizzarne le performance dal punto di vista delle performance predittive (\sim della qualità) della rete neurale.

Le reti neurali sono delle APPROSSIMATORI UNIVERSALI perché sono sufficientemente potenti da per approssimare ~~qualsiasi~~ tipo di funzione: di fatto, basta regolare il numero di layer e la cardinalità di ciascun layer.

→ NB: ecco a meno esagerare con la complessità della rete neurale rispetto alla dimensione del training set onde evitare il rischio di overfitting.

→ A TAL PROPOSITO SI INTRODUCE ANCHE IL DROPOUT, CHE È UN MECCANISMO TALE PER CUI CIASCUN LAYER EMESSA DI PEGGIO.

Minimizzazione di una funzione costo nelle reti neurali:

Nel caso delle reti neurali, la funzione costo che si ottiene è tipicamente difficile da minimizzare analiticamente, per cui si ricorre alla discesa del gradiente. Ricordiamo che ogni step della discesa del gradiente si articola in due fasi:

1) ad partire dal valore di parametri w si determinano le predizioni.

2) ad partire dalle predizioni si calcola il gradiente della funzione costo per il passo successivo.

Lo step (1) è molto semplice: la predizione viene calcolata a partire dall'input x e mano mano si fa avanti con i layer. → FORWARD PROPAGATION

Per quanto riguarda lo step (2), in realtà, il calcolo del gradiente rispetto a un qualunque $w_j^{(k)}$ non lo sappiamo fare: sappiamo solo calcolare il gradiente rispetto alle $a_i^{(k)}$, che è il valore associato al modo i dell'ultimo layer subito prima dell'applicazione della funzione $\phi()$ ($\rightarrow D = \text{numero di layer}$).

CASO DELLA REGRESSIONE LINEARE:

Sappiamo che, con la massima verosimiglianza: $E(w) = \frac{1}{2} \sum_{i=1}^n (y(x_i, w) - t_i)^2$
• $\phi() \equiv \text{IDENTITÀ} \Rightarrow y = a^{(0)} \Rightarrow \frac{\partial E(w)}{\partial a^{(0)}} = \sum_{i=1}^n (y(x_i, w) - t_i)$

CASO DELLA CLASSIFICAZIONE BINARIA:

Sappiamo che, con la massima verosimiglianza: $E(w) = \sum_{i=1}^n (t_i \ln(y(x_i, w)) + (1-t_i) \ln(1-y(x_i, w)))$
• $\phi() \equiv \sigma() \Rightarrow y = \sigma(a^{(0)}) \Rightarrow \frac{\partial E(w)}{\partial a^{(0)}} = \sum_{i=1}^n (y_i - t_i)$

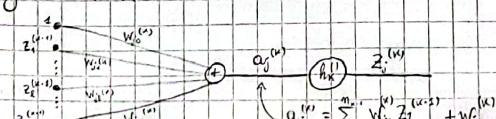
CASO DELLA CLASSIFICAZIONE MULTICLASSE:

Sappiamo che, con la massima verosimiglianza: $E(w) = -\sum_{i=1}^n \sum_{k=1}^K t_{ik} \log y_{ik}$
• $\phi() \equiv S() \Rightarrow y_i = S(a_i^{(0)}) \Rightarrow \frac{\partial E(w)}{\partial a_i^{(0)}} = \sum_{j=1}^n (y_{ij} - t_{ij})$

Alla fine della lezione, conoscendo il gradiente rispetto ad $a^{(0)}$, è possibile ottenere il gradiente rispetto ai vari parametri ripercorrendo la rete neurale all'indietro. → BACKPROPAGATION

APPROFONDIMENTO SULLA BACKPROPAGATION:

Consideriamo un singolo stato K della rete neurale:



→ La funzione costo $E(w)$ possiamo vedere come la somma di tutti gli errori relativi ai singoli punti del dataset: $E(w) = \sum_{i=1}^n E_i(w)$

→ Definiamo $\delta_j^{(k)} := \frac{\partial E_i}{\partial a_j^{(k)}}$

$$\rightarrow \frac{\partial a_j^{(k)}}{\partial w_{j,k}^{(k+1)}} = \frac{\partial}{\partial w_{j,k}^{(k+1)}} \sum_{l=1}^m w_{j,l}^{(k)} z_l^{(k+1)} = z_l^{(k+1)}$$

$$\Rightarrow \frac{\partial E_i}{\partial w_{j,k}^{(k+1)}} = \frac{\partial E_i}{\partial a_j^{(k)}} \cdot \frac{\partial a_j^{(k)}}{\partial w_{j,k}^{(k+1)}} = \delta_j^{(k)} z_l^{(k+1)} \rightarrow z_l^{(k+1)} \text{ è un valore既知 (che è possibile apprendere)} \Rightarrow$$

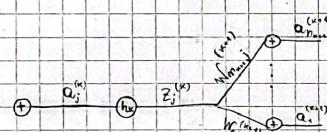
→ resta solo da ricavare $\delta_j^{(k)}$, oppure $\frac{\partial E_i}{\partial a_j^{(k)}}$ (\rightarrow per conoscere la derivata dell'errore rispetto a tutti i $w_{j,k}$,

è sufficiente sapere la derivata dell'errore rispetto ad $a_j^{(k)}$).

COSA BUONA È GIUSTA: noi la derivata rispetto ad $a_j^{(k)}$ (dove k è l'ultimo strato) la conosciamo già: prima abbiamo visto come $z_j^{(k)} = y_j$ e $\frac{\partial E_i}{\partial a_j^{(k)}} = y_j - t_j = \delta_j^{(k)}$.

Per chiudere il cerchio, riuscire solo da capire come si calcola $\frac{\partial E_i}{\partial a_j^{(k+1)}}$ a partire da $\frac{\partial E_i}{\partial a_j^{(k)}}$.

19/05/2023



È chiaro che qualunque variazione di $a_j^{(k)}$ comporta delle variazioni su tutte e sole le variabili $a_n^{(k+1)}$.

→ L'effetto di tale variazione su E_i è una funzione degli effetti su tutte le variabili $a_n^{(k+1)}$:

$$\delta_j^{(k)} = \frac{\partial E_i}{\partial a_j^{(k)}} = \sum_{n=1}^{m_{k+1}} \frac{\partial E_i}{\partial a_n^{(k+1)}} \frac{\partial a_n^{(k+1)}}{\partial a_j^{(k)}} = \sum_{n=1}^{m_{k+1}} \delta_n^{(k+1)} \frac{\partial a_n^{(k+1)}}{\partial a_j^{(k)}}$$

Già sappiamo che: $a_n^{(k+1)} = \sum_{l=1}^m w_{l,n}^{(k+1)} z_l^{(k)}$ e $z_l^{(k)} = h_k(a_j^{(k)})$

$$\Rightarrow \frac{\partial a_n^{(k+1)}}{\partial a_j^{(k)}} = \frac{\partial a_n^{(k+1)}}{\partial z_l^{(k)}} \frac{\partial z_l^{(k)}}{\partial a_j^{(k)}} = w_{j,n}^{(k+1)} h'_k(a_j^{(k)})$$

$$\Rightarrow \delta_j^{(k)} = \sum_{n=1}^{m_{k+1}} \delta_n^{(k+1)} \frac{\partial a_n^{(k+1)}}{\partial a_j^{(k)}} = h'_k(a_j^{(k)}) \sum_{n=1}^{m_{k+1}} \delta_n^{(k+1)} w_{j,n}^{(k+1)}$$

→ Ora sappiamo calcolare $\frac{\partial E_i}{\partial a_j^{(k)}}$ a partire dai valori $\frac{\partial E_i}{\partial a_n^{(k+1)}}$ $\forall n$ \Rightarrow abbiamo tutti gli elementi per calcolare $\frac{\partial E_i}{\partial w_{j,k}^{(k+1)}}$ $\forall i, j, k$ → al questo punto, $\frac{\partial E_i}{\partial w_{j,k}^{(k+1)}} = \sum_{i=1}^m \frac{\partial E_i}{\partial w_{j,i}^{(k+1)}} \quad \forall j, k$

NB: se $h'_k(a_j^{(k)})$ tende ad assumere valori molto minori di 1, si ha il fenomeno dell'**EVANISHING GRADIENT**, anche con variabili al layer $k+1$ ($\delta_{j+1}^{(k+1)}$) costanti, si rischia di avere variazioni al layer k trascurabili.

zatali, il che porta ad avere i parametri $w_k^{(i)}$ relativi al layer K pressoché costanti per tutta l'esecuzione della classificazione \rightarrow si vede bene come tale fenomeno interessi soprattutto i primi strati delle reti neurali.

La conseguenza: la rete neurale non funziona, deve.

È per questo motivo che è opportuno scegliere delle funzioni non lineari $h_k(x)$ la cui derivata non tende ad assumere valori troppo piccoli. La RELU è un buon esempio perché $\frac{\partial \text{RELU}}{\partial x} = 0 \quad \forall x < 0$, $\frac{\partial \text{RELU}}{\partial x} = 1 \quad \forall x \geq 0$.

CLASSIFICAZIONE NON PARAMETRICA

Uso degli istogrammi:

L'intero dominio degli elementi viene partitionato in m BIN (intervalli) d -dimensionali. Ciascun elemento del training set cade all'interno di uno dei bin. La probabilità che un qualunque elemento appartenga al bin contenente già l'elemento x può essere stimata come $\frac{n(x)}{N}$, dove $N = \# \text{ ELEM. TRAINING SET}$, $n(x) = \# \text{ ELEMENTI CONTENUTI NEL BIN DI } x$. Il rapporto tra tale probabilità e l'ampiezza $\Delta(x)$ del bin contenente x può essere visto come la densità di probabilità per il bin di x :

$$p_h(x) = \frac{n(x)}{N\Delta(x)} \quad \leftarrow \Delta(x) \text{ spesso è costante: } \Delta(x) = \Delta \quad \forall x \in X$$

Questa è una possibilità per definire/stimare la $p(x|C_i)$ in modo non parametrico. Diciamo però che ci sono almeno due problematiche:

• Se i bin sono troppi (cosa che tende a succedere per d grande), la stra-grande maggioranza dei bin rimarrà vuota (i.e. senza elementi del training set). Se bisogna valutare un nuovo elemento che casca in uno di questi bin vuoti, non abbiamo praticamente informazioni su come sono distribuiti gli elementi dello stesso tipo di x . Per questo motivo, un approccio del genere (così come tutti gli approssimativi parametrici) non va d'accordo con dati di elevati.

• In questo approccio, i bin sono definiti a priori, per cui c'è la possibilità che un nuovo elemento x da valutare caschi molto male a un bin tra un bin e l'altro (per cui fare fede esclusivamente al bin di appartenenza, non è il massimo).

Per queste ragioni, anche se definire i bin a priori, possiamo pensare che convenga prendere in considerazione l'intorno del nuovo elemento x da valutare (\rightarrow intorno preso ad hoc).

La probabilità P_x che un qualsiasi elemento caschi all'interno della regione $R(x)$ già contenente l'elemento x è pari a:

$$P_x = \int_{R(x)} p(z) dz$$

Dati m elementi del training set x_1, x_2, \dots, x_m , la probabilità che K di loro si trovino in $R(x)$ è data dalla distribuzione binomiale:

$$p(K) = \binom{m}{K} P_x^K (1-P_x)^{m-K} = \frac{m!}{K!(m-K)!} P_x^K (1-P_x)^{m-K}$$

$$\rightarrow E[K] = m P_x ; \quad \sigma^2_K = m P_x (1-P_x) ; \quad K = \frac{m}{n} \Rightarrow$$

$$\Rightarrow E[\bar{x}] = \frac{1}{m} E[K] = P_x ; \quad \sigma^2_{\bar{x}} = \frac{1}{m^2} \sigma^2_K = \frac{P_x(1-P_x)}{m}$$

→ Perce fuori che $\frac{K}{m} = P_x \Rightarrow \bar{x} = \frac{K}{m}$ è un buon stimatore per la probabilità P_x .

Non solo: supponiamo che il volume di $R(x)$ sia sufficientemente piccolo. Allora possiamo assumere che la densità $p(x)$ sia quasi costante all'interno di $R(x)$ e:

$$P_x = \int_{R(x)} p(z) dz \approx p(x) V \quad \text{dove } V = \text{volume di } R(x)$$

Nel momento in cui $P_x \approx \frac{K}{m} \Rightarrow p(x) \approx \frac{K}{mV}$ ($\frac{K}{mV}$ è uno stimatore per $p(x)$).

In questa relazione, m (= # ELEM. TRAINING SET) è fisso; a questo punto vi sono due possibilità:

1) Fissare V e derivare il numero K di elementi che cadono in $R(x)$ dai dati (KERNEL DENSITY ESTIMATION).

2) Fissare K e derivare V dai dati (K-NEAREST NEIGHBORS).

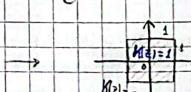
È possibile dimostrare che in entrambi i casi, sotto opportune condizioni, lo stimatore $\frac{K}{mV}$ tende alla vera densità $p(x)$ per $n \rightarrow +\infty$.

Kernel density estimation - Parzen window:

In questo approccio la regione $R(x)$ scelta è un ipercubo di lato h a d dimensioni ($\Rightarrow V = h^d$) centra-to sull'elemento x .

Qui ci facciamo aiutare da una funzione Kernel $K(z)$ detta PARZEN WINDOW che serve a contare il numero di elementi contenuti nell'ipercubo unitario centrato nell'origine:

$$K(z) = \begin{cases} 1 & |z_i| \leq \frac{1}{2}, \quad \forall i = 1, \dots, d \\ 0 & \text{otherwise} \end{cases}$$



Per i nostri scopi, serve una traslazione (dall'origine al punto x) e un ridimensionamento (lato h). Co di questa funzione \Rightarrow utilizziamo $K\left(\frac{x-x_i}{h}\right)$, che vale 1 se e solo se x casca nell'ipercubo di lato h centrato in x .

Tuttavia, il numero di elementi presenti nel nostro ipercubo è pari a:

$$K = \sum_{i=1}^m K\left(\frac{x-x_i}{h}\right)$$

La stima della densità di probabilità degli elementi è data da:

$$p_n(x) = \frac{K}{mV} = \frac{1}{mV} \sum_{i=1}^m K\left(\frac{x-x_i}{h}\right) = \frac{1}{mV} \sum_{i=1}^m K\left(\frac{x-x_i}{h}\right)$$

← TANTO PIÙ h È ELEVATO, TANTO LA STIMA DELLA DENSITÀ HA UN ANDAMENTO "BRUSCO"!

PUNTI DI DEBOLEZZA:

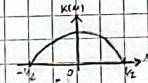
- 1) La stima delle densità risulta essere comunque una funzione discontinua (è a gradini).
- 2) Tutti gli elementi che cadono nella regione $R(x)$ hanno lo stesso peso ai fini della stima di $p(x)$: non si tiene conto dell'effettiva distanza che c'è tra x e ciascun punto x_i del training set.

SOLUZIONE:

Sostituire la Parzen Window con una funzione più "smooth", come ad esempio:

$$\rightarrow \text{FUNZIONE KERNEL GAUSSIANA: } K(u) = \frac{1}{\sqrt{\pi} \sigma^2} e^{-\frac{u^2}{2\sigma^2}}$$

$$\rightarrow \text{FUNZIONE KERNEL DI EPANECHNIKOV: } K(u) = 3\left(\frac{1}{2} - u^2\right), |u| \leq \frac{1}{2}$$



CHE AVVIENE LA CLASSIFICAZIONE?

Il solito basterebbe trovare la classe C_i tale per cui $p(C_i|x)$ è massima:

$$\begin{aligned} \underset{i}{\operatorname{argmax}} p(C_i|x) &= \underset{i}{\operatorname{argmax}} p(x|C_i)p(C_i) = \underset{i}{\operatorname{argmax}} \frac{1}{m_i V_i} \sum_{j=1}^{m_i} K\left(\frac{x-x_j}{h}\right) p(C_i) \\ &= \underset{i}{\operatorname{argmax}} \frac{1}{m_i V_i} \sum_{j=1}^{m_i} K\left(\frac{x-x_j}{h}\right) \end{aligned}$$

$$p(C_i) = \frac{m_i}{m}$$

Un elemento x viene assegnato alla classe con più elementi vicini a lui (eventualmente pesati in base alla distanza).

K-nearest Neighbors ($x_k(n)$):

In quest'altro approccio la regione $R(x)$ scelta è la più piccola sfera d-dimensionale centrata nel x elemento x la valutare che contiene esattamente K elementi del training set.

La stima della densità di probabilità degli elementi è data da:

$$p(x) = \frac{K}{mV} = \frac{K}{m \pi r_k^d(x)} \quad \text{dove:}$$

$\cdot C_i$ = VOLUME DI UNA SFERA D-DIMENSIONALE CON RAGGIO UNITARIO.

$\cdot r_k(x) =$ RAGGIO DELLA PIÙ PICCOLA SFERA CENTRATA IN x E CONTENENTE K ELEMENTI = DISTANZA TRA x E IL K-ESIMO ELEMENTO DEL TRAINING SET PIÙ VICINO A x .

CHE AVVIENE LA CLASSIFICAZIONE?

→ Probabilità che un elemento x della classe C_i caschi in $R(x)$: $p(x|C_i) = \frac{K_i}{m_i V_i}$

→ Probabilità che un elemento x caschi in $R(x)$: $p(x) = \frac{K}{mV}$

→ Probabilità di appartenere alla classe C_i : $p(C_i) = \frac{m_i}{m}$

$$\Rightarrow p(C_i|x) = \frac{p(x|C_i)p(C_i)}{p(x)} = \frac{K_i}{K} = \frac{\# \text{ ELEMENTI DELLA CLASSE } C_i \text{ CHE CADONO IN } R(x)}{\# \text{ TOTALI DI ELEMENTI CHE CADONO IN } R(x)}$$

ELEM. CLASSE C_i CHE CADONO IN $R(x)$

\hat{x}

23/05/2023

SUPPORT VECTOR MACHINES

È un modello che nasce come modello per la classificazione binaria ma può essere esteso anche per la classificazione multiclasse e la regressione. Inizialmente consideriamo il caso semplificato della classificazione binaria con training set linearmente separabile.

IDEA: se il training set è linearmente separabile, allora esistono INFINTI iperpiani di separazione che separano perfettamente gli elementi appartenenti alle due classi: di fatto, anche una minima traslazione / rotazione di un iper piano da l'altro iper piano che separano perfettamente gli elementi del training set. L'obiettivo di SVM (Support Vector Machines) è quello di selezionare l'iper piano di separazione migliore tra questi infiniti iperpiani.

Consideriamo un classificatore binario che, a ogni elemento x_i , associa il target $y_i = -1$ se x_i viene assegnata la classe C_0 , il target $y_i = 1$ se a x_i viene assegnata la classe C_1 . Il classificatore è definito:

$$h(x) = \text{sgn}(\mathbf{w}^T \cdot \mathbf{x} + w_0)$$

→ Vediamo che l'output è semplicemente la classe assegnata all'elemento: trattasi di un classificatore deterministico.

Margine funzionale:

Per un singolo punto del training set \mathcal{T} , il MARGINE FUNZIONALE $\bar{\gamma}_i$ è dato dalla seguente funzione:

$$\bar{\gamma}_i = t_i (\mathbf{w}^T \cdot \mathbf{x}_i + w_0) \rightarrow \begin{array}{l} \text{è positivo se la predizione è corretta ed è negativo altrimenti.} \\ \text{misura la distanza (con segno) tra il punto } x_i \text{ e l'iperpiano di seta.} \\ \text{Razionale: maggiore è il modulo della distanza, più il classificatore è} \\ \text{confidente nella predizione.} \end{array} \rightarrow \text{il punto di distanza più} \rightarrow \text{è proprio} \rightarrow \text{ogni}$$

Per un intero training set $\mathcal{T} = \{(x_1, t_1), \dots, (x_n, t_n)\}$ il margine funzionale $\bar{\gamma}$ è pari a:

$$\bar{\gamma} = \min_i \bar{\gamma}_i \rightarrow \begin{array}{l} \text{SE IL TRAINING SET È LINEARMENTE SEPARABILE} \\ \text{ALLORA} \bar{\gamma} \text{ È CERTAMENTE POSITIVO.} \end{array}$$

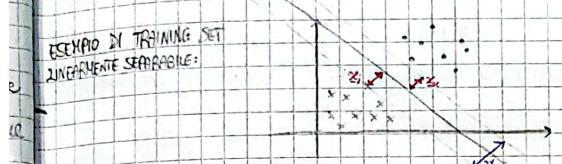
Nel momento in cui $\bar{\gamma}$ è sicuramente positivo, nel caso di training set linearmente separabile l'idea sarebbe trovare l'iperpiano che massimizza il valore di $\bar{\gamma}$.

Margine geometrico:

Per un singolo punto del training set (x_i, t_i) , il MARGINE GEOMETRICO γ_i è dato dal prodotto tra t_i e la distanza GEOMETRICA tra x_i e l'iperpiano di separazione: di fatto, $\mathbf{w}^T \cdot \mathbf{x}_i + w_0$ dà luogo alla distanza geometrica moltiplicata per la norma di w , per cui trattasi di un concetto di distanza

'relativa', ovvero dipendente non solo dall'iperpiano in sé ma anche dai coefficienti usati per descriverlo. In termini formali: $\gamma_i = t_i \left(\frac{w^T}{\|w\|} x_i + \frac{w_0}{\|w\|} \right) = \frac{\bar{\gamma}_i}{\|w\|}$

Ora, per un training set T il margine geometrico è pari a: $\gamma = \min_i \gamma_i$



L'idea è considerare i due iperpiani (parallelî) tra i quali va a cadere un punto del training set. Questi iperpiani sono da estremi della FASCIA DI SEPARAZIONE, sono detti IPERPIANI DI MASSIMO MARGINE e attraversano (e meno) un punto appartenente a una determinata classe (nell'esempio x_i per un iperpiano e x_k per l'altro).

Per massimizzare il margine geometrico di T , basterà prendere come iperpiano di separazione quello equivalente dagli iperpiani di massimo margine (oltre da x_i e x_k).

x_i e x_k sono detti punti di supporto (o VETTORI DI SUPPORTO).

Formalizzando, vogliamo risolvere il seguente problema di ottimizzazione:

$$\max_w \gamma \quad \text{where } t_i \left(\frac{w^T}{\|w\|} x_i + \frac{w_0}{\|w\|} \right) \geq \gamma \quad \forall i=1, \dots, n \quad \Leftrightarrow$$

$$\max_w \gamma \quad \text{where } t_i (w^T x_i + w_0) \geq \gamma \|w\| \quad \forall i=1, \dots, n$$

Così è possibile notare, se scaliamo i parametri w per una costante c , tutti i margini geometrici γ_i rimangono invariati. Possiamo sfruttare questo grado di libertà per introdurre il seguente vincolo:

$$\gamma = \min_i t_i (w^T x_i + w_0) = 1 \quad (\rightarrow \text{i.e. } \|w\| = \gamma \Rightarrow \text{AMPIZZA FASCA} = 2)$$

Ottieniamo dunque:

$$\max_w \gamma \quad \text{where } t_i (w^T x_i + w_0) \geq 1 \quad \forall i=1, \dots, n \quad \text{dove } \gamma = \|w\|^{-1}$$

• Un elemento x_i è detto ATTIVO se per lui vale $t_i (w^T x_i + w_0) = 1$.

• Un elemento x_i è detto INATTIVO se per lui vale $t_i (w^T x_i + w_0) > 1$.

In definitiva, il nostro problema di ottimizzazione è equivalente a:

$$\max_w \frac{1}{2} \|w\|^2 \quad \text{where } t_i (w^T x_i + w_0) \geq 1 \quad \forall i=1, \dots, n$$

QUESTA È UNA FUNZIONE QUADRATICA CONVESSA

QUESTI SONO TUTTI VINCOLI LINEARI; CIASCUNO DI LORO CONSISTE IN UN IPERPIANO CHE TAGLIA FUORI UNA REGIONE NON AMMISIBILE. È POSSIBILE MOSTRARE CHE L'INTERSEZIONE DI TUTTI I VINCOLI DÀ LUGO A UNA REGIONE CONTIGUA E CONVESSA.

\Rightarrow Questo problema è detto PROBLEMA DI OTTIMIZZAZIONE QUADRATICO CONVESSO ed è molto容易 da trattare.

25/05/2023

N.B.
ai
APP.
→
→
→

Allo stesso problema di ottimizzazione quadratica convessa P permette di definire un problema duale D che ha la stessa identica soluzione ottima (è solo rappresentata in modo diverso). A partire da P , quello che si fa dunque è ottenere e risolvere il problema D , e qui si ripete la soluzione ottima trovata nella forma "originale" (i.e. quella che si avrebbe avuto risolvendo direttamente P). La trasformazione del problema P nel suo duale avviene mediante il concetto di LAGRANGIANO.

A

LAGRANGIANO:

Consideriamo il seguente problema di ottimizzazione generico:

$$\min_{\mathbf{x} \in \Omega} f(\mathbf{x}) \quad \text{where } g_i(\mathbf{x}) \leq 0 \quad \forall i=1, \dots, K$$

Per Cesare
dove $f(\cdot)$, $g_i(\cdot)$ sono funzioni convesse. Il Ω è un insieme convesso.

Il LAGRANGIANO è definito così: $L(\mathbf{x}, \lambda) = f(\mathbf{x}) + \sum_{i=1}^K \lambda_i g_i(\mathbf{x})$ con $\lambda_i \geq 0 \quad \forall i=1, \dots, K$

$$\Rightarrow \max_{\mathbf{x}} L(\mathbf{x}, \lambda) = f(\mathbf{x}) + \max_{\mathbf{x}} \sum_{i=1}^K \lambda_i g_i(\mathbf{x})$$

- SE \mathbf{x}^* È UNA SOLUZIONE AMMISCIBILE $\Rightarrow g_i(\mathbf{x}) \leq 0 \quad \forall i=1, \dots, K \Rightarrow$ il massimo per $L(\mathbf{x}, \lambda)$ è dato da $\lambda_i = 0 \quad \forall i=1, \dots, K \Rightarrow \max_{\mathbf{x}: \lambda_i=0} L(\mathbf{x}, \lambda) = f(\mathbf{x})$
- SE \mathbf{x}^* È UNA SOLUZIONE NON AMMISCIBILE $\Rightarrow \exists i: g_i(\mathbf{x}) > 0 \Rightarrow$ il massimo per $L(\mathbf{x}, \lambda)$ è NON limitato.

Quindi, se la soluzione \mathbf{x}^* ottima ammmissibile esiste, abbiamo che:

$$\min_{\mathbf{x} \in \Omega} f(\mathbf{x}) = \min_{\mathbf{x} \in \Omega} \max_{\lambda \geq 0} L(\mathbf{x}, \lambda)$$

Inoltre, se il problema di ottimizzazione è convesso (come nel nostro caso), vale la proprietà della DUALITÀ FORTE: $\min_{\mathbf{x} \in \Omega} f(\mathbf{x}) = \min_{\mathbf{x} \in \Omega} \max_{\lambda \geq 0} L(\mathbf{x}, \lambda) = \max_{\lambda \geq 0} \min_{\mathbf{x} \in \Omega} L(\mathbf{x}, \lambda)$

CONDIZIONI DI KARUSH-KHONN-TUCKER (KKT):

Esistono 4 condizioni necessarie e sufficienti per l'esistenza della soluzione ottima $(\mathbf{x}^*, \lambda^*)$ (e tali condizioni possono essere sfruttate per semplificare la definizione del problema duale):

1) $\frac{\partial L(\mathbf{x}, \lambda)}{\partial \mathbf{x}} \Big|_{\mathbf{x}^*, \lambda^*} = 0$ NE???

2) $\frac{\partial L(\mathbf{x}, \lambda)}{\partial \lambda_i} \Big|_{\mathbf{x}^*, \lambda^*} = g_i(\mathbf{x}^*) \geq 0 \quad \forall i=1, \dots, K$

SE VALE L'UGUAGLIANZA, APPLICA IL VINCULO ATTIVO E VICEVERSA.

3) $\lambda_i^* \geq 0 \quad \forall i=1, \dots, K$

4) $\lambda_i^* g_i(\mathbf{x}^*) = 0 \quad \forall i=1, \dots, K$

N.B.
ai

APP.

→

→

A

Per Cesare

M

1.a

1.b

1.c

1.d

1.e

1.f

1.g

1.h

1.i

1.j

1.k

1.l

1.m

1.n

1.o

1.p

1.q

1.r

1.s

1.t

1.u

1.v

1.w

1.x

1.y

1.z

NB: la condizione (4) impone lo λ_i^* pari a 0 per tutti i componenti $i \in S_1 \dots K$ corrispondenti ai vincoli NON ATTIVI ($g_i(x^*) > 0$). Tale condizione è della COMPLEMENTARY SLACKNESS.

APPLICAZIONE DEL LAGRANGIANO AL PROBLEMA (PISO):

$$\rightarrow f(x) = \frac{1}{2} \|w\|^2 \quad \rightarrow g_i(x) = t_i(w^T x_i + w_b) - 1 \geq 0$$

\rightarrow I₂ possiamo vedere come l'intersezione tra K semi-piani lineari (i.e. una regione convessa).

$$\Rightarrow L(w, \lambda) = \frac{1}{2} \|w\|^2 - \sum_{i=1}^n \lambda_i (t_i(w^T x_i + w_b) - 1) = \frac{1}{2} \|w\|^2 - \sum_{i=1}^n \lambda_i t_i w^T x_i - w_b \sum_{i=1}^n \lambda_i t_i + \sum_{i=1}^n \lambda_i$$

A questo punto, il problema duale per definizione è dato da:

$$\max_w \min_{\lambda} L(w, \lambda) \quad \text{where } \lambda_i \geq 0 \quad \forall i = 1, \dots, K$$

Per poter scrivere il nostro problema duale in modo preciso, dobbiamo rifarci alle condizioni KKT. Concentriamoci ora sulla (1):

$$\begin{aligned} \frac{\partial L(w, \lambda)}{\partial w} &= w^* - \sum_{i=1}^n \lambda_i t_i x_i = 0 \\ \frac{\partial L(w, \lambda)}{\partial w_b} &= \sum_{i=1}^n \lambda_i t_i = 0 \end{aligned}$$

Mettendo insieme tutte le condizioni KKT otteniamo:

$$1.a) \quad w^* = \sum_{i=1}^n \lambda_i t_i x_i$$

$$1.b) \quad \sum_{i=1}^n \lambda_i t_i = 0$$

$$2) \quad t_i (w^* \cdot x_i + w_b^*) - 1 \geq 0 \quad \forall i = 1, \dots, n$$

$$3) \quad \lambda_i \geq 0 \quad \forall i = 1, \dots, n$$

$$4) \quad \lambda_i (t_i (w^* \cdot x_i + w_b^*) - 1) = 0 \quad \forall i = 1, \dots, n$$

È POSSIBILE MOSTRARE CHE LE CONDIZIONI (2), (4) QUI SONO SOVRAPPONDANTI (SEMPRE VERIFICATE).

Al punto siamo in grado di esprimere il Lagrangiano solo in funzione di λ :

$$\begin{aligned} L(\lambda) &= \frac{1}{2} \|w^*\|^2 - \sum_{i=1}^n \lambda_i t_i (\sum_{j=1}^n \lambda_j t_j x_i \cdot x_j) - w_b^* \sum_{i=1}^n \lambda_i t_i + \sum_{i=1}^n \lambda_i = \\ &= \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \lambda_i t_i \lambda_j t_j x_i \cdot x_j - \sum_{i=1}^n \sum_{j=1}^n \lambda_i t_i \lambda_j t_j x_i \cdot x_j - 0 + \sum_{i=1}^n \lambda_i = \\ &= \sum_{i=1}^n \lambda_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \lambda_i t_i \lambda_j t_j x_i \cdot x_j \end{aligned}$$

In definitiva, il problema di ottimizzazione duale può essere espresso nel seguente modo:

$$\max_{\lambda} L(\lambda) = \max_{\lambda} \left(\sum_{i=1}^n \lambda_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \lambda_i t_i \lambda_j t_j x_i \cdot x_j \right) \quad \text{where} \quad \begin{cases} \lambda_i \geq 0 & \forall i = 1, \dots, n \\ \sum_{i=1}^n \lambda_i t_i = 0 \end{cases}$$

NB: ricordiamo che $\lambda_i^{(b)} \neq 0$ solo per i punti di supporto, che sono due o poco più $\Rightarrow L(\lambda)$ è in realtà molto + semplice di come appare.

→ È sempre possibile applicare la funzione base ai vari elementi $x_i \Rightarrow$ in generale, definendo la funzione Kernel $K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$, la formulazione del problema divenne:

$$\max_{\lambda} L(\lambda) = \max_{\lambda} \left(\sum_{i=1}^m \lambda_i - \frac{1}{2} \sum_{i=1}^m \sum_{j=1}^m \lambda_i \lambda_j t_i t_j K(x_i, x_j) \right) \quad \text{where} \quad \begin{cases} \lambda_i \geq 0 & \forall i = 1, \dots, m \\ \sum_{i=1}^m \lambda_i t_i = 0 \end{cases}$$

SVANTAGGIO DEL PASSAGGIO AL PROBLEMA DUALE:

Il numero delle variabili cresce dal numero di feature al numero di elementi del training set.

VANTAGGIO DEL PASSAGGIO AL PROBLEMA DUALE:

Il numero effettivo delle variabili da prendere in considerazione, perché rilevanti per la classificazione è molto minore di n (nel senso anche minore del numero di feature).

DERIVAZIONE DEI PARAMETRI w_i^* A PARTIRE DALLA SOLUZIONE DEL DULE:

$$\begin{aligned} w_i^* &= \sum_{j=1}^m \lambda_j^* t_j \phi(x_j) & \forall i = 1, \dots, m \\ w_0^* &\text{ può essere ottenuto a partire dalla condizione } t_k (\phi(x_k)^T w^* + w_0^*) - 1 = 0. \end{aligned}$$

di questo punto, nel momento in cui bisogna effettuare una predizione su un nuovo elemento, il classificatore calcola:

$$y(x) = \sum_{i=1}^m w_i^* \phi(x_i) + w_0^* = \sum_{j=1}^m \lambda_j^* t_j K(x_j, x) + w_0^*$$

In più ci sono solo λ_j^* a multiplied visto così può sempre una predizione. Ovviamente anche qui è esprimibile in termini di λ_j .

I λ_j^* sono comunque chiamati ai w_i^*

⇒ L'approccio non è completamente non parametrico → è una sorta di via di mezzo.

NB: di nuovo, la sommatoria di $y(x)$ è solo sui vettori di supporto (i.e. su pochi elementi) ⇒ in pratica, ogni predizione avviene confrontando l'elemento da valutare semplicemente ai punti più importanti, ovvero quelli che si trovano 'ai bordi' di ciascuna classe (i punti di supporto, oppure il punto).

NB2: in tutto ciò, noi troviamo un iperpiano di separazione che è lineare nello spazio dei punti $\phi(x)$ trasformato (non necessariamente nello spazio dei punti x originale).

Caso in cui il training set non è linearmente separabile:

C'sono due soluzioni, che possono anche essere combinate:

1) La prima l'abbiamo già accennata: è possibile cambiare la rappresentazione degli elementi mediante l'uso di funzioni base. In tal modo, si ottiene un nuovo insieme di elementi all'interno del quale si spera di avere gli elementi appartenenti a classi diverse un po' più separati tra loro.

La seconda soluzione consiste nel rilassare i vincoli nel nostro problema di ottimizzazione in modo tale da accettare anche i punti predetti in maniera scorretta. Da qui nasce l'esigenza di impostare qualche modifica alla funzione costo: in particolare, si introduce un costo per ogni elemento predetto male. \rightarrow così i vincoli passano da essere hard a essere soft.

Formalizziamo questo approccio: il problema di ottimizzazione viene ora formulato così:

$$\min_{w, b, \xi} \frac{1}{2} w^T w + C \sum_{i=1}^n \xi_i \quad \text{where } \begin{cases} t_i(w^T \phi(x_i) + b) \geq 1 - \xi_i & \forall i = 1, \dots, n \\ \xi_i \geq 0 & \forall i = 1, \dots, n \end{cases}$$

$\xi = (\xi_1, \dots, \xi_n)$

\downarrow È IL SOLO VERRIMENTO.

I vari ξ_i rappresentano "di quanto non viene soddisfatto il vincolo da ciascun elemento nel problema originale" o, sono detti slack. Nel momento in cui $t_i(w^T \phi(x_i) + b)$ diviene proprio negativo, cadranno nel caso in cui x_i viene classificato male.

Rifacendo tutti i calcoli, otteniamo il seguente problema finale:

$$\max_{\lambda} L(\lambda) = \min_w \left(\sum_{i=1}^n \lambda_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \lambda_i \lambda_j t_i t_j K(x_i, x_j) \right) \quad \text{where } \begin{cases} 0 \leq \lambda_i \leq C & \forall i = 1, \dots, n \\ \sum_{i=1}^n \lambda_i t_i = 0 \end{cases}$$

UNICA VERA DIFFERENZA
 \downarrow RICP. PRIMA

Problema campionabile:

Col funzionamento che abbiamo esposto, il tempo di training per SVM è pari a $O(m^3)$, dove $m = \#$ ELEMENTI DEL TRAINING SET. È un costo pratico per i dataset più grandi. Per rendere più veloce SVM, si possono intraprendere questi approcci:

- Calcolare con algoritmi più veloci e più imprecisi una soluzione non ottimale per il problema.
- Ricorrere alla discesa del gradiente, dato che è possibile controllare con gli iperparametri il numero di iterazioni da effettuare. Affinché questa strada sia praticabile, è necessario avere la possibilità di definire una funzione loss compatibile con l'approccio SVM (e fortunatamente questa funzione loss esiste).

26/05/2023

SVM & discesa del gradiente:

Ricordiamo la definizione del problema di ottimizzazione primale nel caso di training set non linearmente separabile. Possiamo notare che il valore ottimale di ξ_i è dato da:

$$\xi_i = \begin{cases} 0 & \text{se } t_i(w^T \phi(x_i) + b) \geq 1 \\ 1 & \text{altrimenti} \end{cases}$$

\Rightarrow Si può essere espressa come una Hinge Loss:

$$L_h(w, w_0, x_i; t_i) = \max(0, 1 - t_i(w^T \phi(x_i) + w_0))$$

Possiamo dunque definire la nostra funzione costo da minimizzare in questo modo:

$$C(w) = \frac{1}{n} w^T w + C \sum_{i=1}^n L_h(w, w_0, x_i; t_i) \quad \text{OC} \sum_{i=1}^n L_h(w, w_0, x_i; t_i) + \frac{\lambda}{2} \|w\|^2 \quad \leftarrow \begin{array}{l} \text{RICORDA LA RIDUCE} \\ \text{REGULARIZATION.} \end{array}$$

Ai fini della ricerca del gradiente, ci serve calcolare le derivate della funzione costo:

$$\frac{\partial C(w)}{\partial w_i} = w_i - \sum_{x_i \in L} t_i \phi_i(x_i) \quad \text{dove } x_i \in L \Rightarrow t_i(w^T \phi(x_i) + w_0) < 1$$

BATCH GRADIENT DESCENT:

$$w_i^{(t+1)} = w_i^{(t)} - \eta w_i^{(t)} + \eta \sum_{x_i \in L} t_i \phi_i(x_i)$$

STOCHASTIC GRADIENT DESCENT:

$$\cdot \text{ Se } x_i \text{ è tale che } t_i(w^T \phi(x_i) + w_0) < 1 \Rightarrow w_i^{(t+1)} = (1-\eta)w_i^{(t)} + \eta \phi_i(x_i) t_i$$

$$\cdot \text{ In caso contrario, } w_i^{(t+1)} = w_i^{(t)}$$

Utilizzo della funzione Kernel in SVM:

Abbiamo visto come in SVM le predizioni vengono effettuate così:

$$y(\bar{x}) = \sum_{j=1}^n \lambda_j t_j K(x_j, \bar{x}) + w_0 \quad \text{dove } K(x_j, \bar{x}) = \phi(x_j)^T \phi(\bar{x})$$

\Rightarrow viene usata una funzione Kernel che è data dal prodotto scalare tra due vettori che rappresentano rispettivamente un elemento x_j del training set e l'elemento \bar{x} da valutare.

Di base il prodotto scalare ha due vettori e tanto più alto quanto più i vettori si assomigliano tra loro \Rightarrow ai fini della predizione, gli elementi del training set che sembrano di più a quello da predire daranno un contributo maggiore.

Comunque sia, noi siamo interessati a $\phi(x_j), \phi(\bar{x})$ solo per calcolare $K(x_j, \bar{x})$. Tuttavia, possono anche essere casi in cui $K(x_j, \bar{x})$ può essere ottenuto senza passare per $\phi(x_j), \phi(\bar{x})$.

\hookrightarrow In linea di principio, vogliamo utilizzare direttamente una funzione $K: X \times X \rightarrow \mathbb{R}$ per il calcolo dei coefficienti (*i pesi*) di t_j nella predizione, dove $K()$ deve rispettare il vincolo di essere una funzione Kernel: da un punto di vista discorsivo, ciò è verificato nel caso in cui esiste una funzione $\phi: X \rightarrow \mathcal{H}$ (dove \mathcal{H} è spazio delle feature) tale che $K(x_1, x_2) = \phi(x_1)^T \phi(x_2)$ (i.e. se esiste uno spazio degli elementi dove $K()$ misura la somiglianza degli elementi x_1, x_2 dati in input).

Ora come facciamo a verificare la condizione in pratica?

1) Una condizione necessaria e sufficiente per una funzione $K: \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}$ per essere una funzione Kernel è: \forall insieme (x_1, \dots, x_n) la matrice di Gram K t.c. $K_{ij} = K(x_i, x_j)$ è SEMI-DEFINTA POSITIVA, ovvero ha sempre gli autovalori non negativi, ovvero $\forall K \geq 0 \quad \forall \text{ vettore } v$

\hookrightarrow È CHIARO COME SIA FARMAGNOSO RECUPERARE QUESTA SPALDA IN PRATICA.

2) Effettuare la verifica in modo algebrico. Vediamolo con un esempio ($K(x_1, x_2) = (x_1 \cdot x_2)^2$, dove $x_1, x_2 \in \mathbb{R}^2$):

$$\begin{aligned} (x_1 \cdot x_2)^2 &= (x_{11} x_{21} + x_{12} x_{22})^2 = x_{11}^2 x_{21}^2 + x_{11}^2 x_{22}^2 + 2 x_{11} x_{12} x_{21} x_{22} = \\ &= (x_{11}^2, x_{21}^2, x_{11} x_{12}, x_{11} x_{22}) \cdot (x_{21}^2, x_{22}^2, x_{12} x_{21}, x_{12} x_{22}) = \phi(x_1) \cdot \phi(x_2) \end{aligned}$$

dove $\phi(x) = (x_1^2, x_2^2, x_1 x_2, x_1 x_2)^\top$ ✓

\hookrightarrow ANCHE QUESTA TECNICA RISULTA FARMAGNOSA, SOFTRUTTO NEL MOMENTO IN CUI LA FUNZIONE ϕ è.

3) Costruire una propria funzione Kernel sfruttando il fatto che le funzioni Kernel sono chiuse rispetto a qualsiasi operatore e portando dal fatto che $x_1 \cdot x_2$ è la più semplice funzione Kernel definibile. Di fatto, date le funzioni Kernel $K_1(x_1, x_2), K_2(x_1, x_2)$, allora $K(x_1, x_2)$ è a sua volta una funzione Kernel in tutti questi casi (e non solo):

- $K(x_1, x_2) = c$
- $K(x_1, x_2) = K_1(x_1, x_2) + K_2(x_1, x_2)$
- $K(x_1, x_2) = K_1(x_1, x_2) K_2(x_1, x_2)$
- $K(x_1, x_2) = c K_1(x_1, x_2) \quad \forall c > 0$
- $K(x_1, x_2) = x_1^\top A x_2$ con $A = \text{MATRICE DEFINITA POSITIVA}$

Le funzioni Kernel più rilevanti sono:

\rightarrow KERNEL POLINOMIALE: $K(x_1, x_2) = (x_1 \cdot x_2 + 1)^d$

\rightarrow KERNEL SIGMOIDALE: $K(x_1, x_2) = \tanh(c_1 x_1 \cdot x_2 + c_2)$

\rightarrow KERNEL GAUSSIANO: $K(x_1, x_2) = e^{-\frac{\|x_1 - x_2\|^2}{2\sigma^2}}$, $\sigma \in \mathbb{R}$

NB: l'utilizzo delle funzioni Kernel lo si ha solo nell'approssimazione che prevede il passaggio al problema duale (e non nella discesa del gradiente).

NB: le funzioni Kernel, essendo funzioni che misurano la somiglianza tra due oggetti, possono essere applicate a qualunque cosa (stringhe, alberi, ...) e non solo a vettori di reali.

\Rightarrow QUESTO È UN INTERESSE CHE VA CONTRA LA NECESSITÀ DI SCAMBIARE DISCESA DEL GRADIENTE X MIGLI PRESTAZIONI.

ALBERI DECISIONALI

Sono classificatori che operano ricorsivamente nel seguente modo:

→ Dato lo spazio degli elementi (o una regione), viene suddiviso in due semisparti lungo una specifica dimensione; l'iperpiano di suddivisione viene scelto 'a piacere', in base alla convenienza. → L'IDEA PER CONVENIENZA SI TRATTARE 'SEPARARE IL PIÙ POSSIBILE ELEMENTI DI CLASSE DIVERSE'

→ Lo spazio degli elementi può tenere un'analogia con un albero: a ciascuna suddivisione è associata a un nuovo livello dell'albero (co i rispettivi nodi children) ma i nodi figli dell'albero sono relativi a quelle regioni dello spazio che non vengono ulteriormente suddivise e si ottengono dunque assegnando una classe C_i (oppure un vettore di probabilità $p(C_i)$).

attenzione a non suddividere troppo lo spazio dei punti (i.e. a non rendere l'albero di decisione troppo profondo), altrimenti si rischia di incorrere all'overfitting.

→ Oltre a questo, c'è anche il discorso su come deve essere selezionata ciascuna partizione. Con tal proposito, dobbiamo tener presente che in ciascuna regione ('rettangolo') ci sono elementi appartenenti a diverse classi con le rispettive percentuali (che eventualmente corrispondono alle probabilità in ciascuna classe). Più questi elementi sonoeterogenei, più si dice che l'insieme di riferimento è IMPURO. Nei atti di purificazione bisogna di una funzione che misuri l'impurezza di un insieme, dove più l'impurezza è bassa (\Rightarrow una classe prevale sulle altre nell'insieme), meglio è.

→ La misura di impurezza deve essere una funzione $\phi: p \rightarrow \mathbb{R}$ che accetta in input una densità di probabilità discreta sulle classi C_i , dove $p_i = \phi(C_i)$. Tale funzione deve soddisfare le seguenti proprietà:

- $\phi(p) \geq 0$ • $\phi(p)$ è differenziabile ovunque
- $\phi(p)$ è minima se $\exists i: p_i = 1$
- $\phi(p)$ è massima se $p_i = \frac{1}{K} \forall i$
- $\phi(p) = \phi(p')$ se p' è una permutazione di p

Ora, supponiamo che $\phi(p_S)$ sia una misura d'impurezza dell'insieme S di elementi, e sappiamo di poter suddividere S in r sottoregioni S_1, \dots, S_r . Ciascun S_i avrà un numero di elementi (per cui avrai associato un valore p_i indicante la percentuale di elementi di S

che cadono in S_j) è un indice di impurità $\phi(p_{S_j})$. È chiaro che, con la suddivisione, l'insieme S è passato da avere un indice di impurità (medio) $\phi(p_S)$ a un indice di impurità (medio) pari a:

$$\sum_{i,j} p_i \phi(p_{S_j}) \quad \rightarrow \text{MEDIA PESATA SUI } \phi(p_{S_j})$$

→ Più questo valore è basso rispetto a $\phi(S)$, più vuol dire che S è stato partitionato come si deve. Introduciamo dunque la seguente quantità (che è bene massimizzare):

$$\Delta_\phi(S, f) := \phi(p_S) - \sum_{i,j} p_i \phi(p_{S_j})$$

dove f è la funzione che descrive come viene effettuata la partitione di S ($x \in S; \Leftrightarrow f(x) = i$).

Entropia e Information Gain:

Un esempio di $\phi(p_S)$ è proprio l'**ENTROPIA** H_S , che definiamo così:

$$H_S = - \sum_{i=1}^K \frac{|S_i|}{|S|} \log_2 \frac{|S_i|}{|S|}$$

dove $|S_i| = \# \text{elementi in regione } S_i \text{ della classe } C_i$; $|S| = \# \text{elementi totali in regione } S$.

Com'è giusto che sia, H_S è minimo (i.e. nullo) se tutti gli elementi appartengono alla stessa classe, ed è massimo (i.e. pari a $\log_2 K$) se tutte le classi sono equidistribuite in S .

La bontà della suddivisione f qui è data dall'**INFORMATION GAIN** (IG):

$$IG(S, f) = H_S - \sum_{j=1}^n \frac{|S_j|}{|S|} H_{S_j} \quad \text{dove } \frac{|S_j|}{|S|} \equiv p_j \equiv \% \text{ elementi che cadono in sottoregione } S_j$$

30/05/2023

Altri indici di impurità:

→ **GINI INDEX**: indice che rappresenta quanto una distribuzione si allontana dall'uniforme:

$$G_S = 1 - \sum_{i=1}^K \left(\frac{|S_i|}{|S|} \right)^2$$

→ **DKM**: $DKM_S = \sqrt{\left(\frac{|S_1|}{|S|} \right) \left(\frac{|S_2|}{|S|} \right)}$ (per la classificazione binaria).

Ma quando dobbiamo farci nel partitionare le nostre regioni?

Qui l'approccio è empirico: ci si basa su dei criteri predefiniti.

• **CITERI GLBESI** → altezza massima dell'albero, numero massimo di foglie.

• **CITERI LOCALI** → una specifica regione non viene ulteriormente suddivisa se comprende meno di tot elementi del training set oppure se il relativo valore di entropia / qualunque indice di impurità è inferiore a una certa soglia.

VENGONO ANCHE UTILIZZATE DELLE TECNICHE DI PRUNING (CHE SONO ENTRAPEZUE) PER RIDURRE LA DIM. E DELL'ALBERO E REGOLARIZZARE

METODI ENSEMBLE

IDEA: costruire e utilizzare molti modelli dello stesso tipo, in modo tale che l'output del modello risulti essere una combinazione di tali predittori. Questi ultimi (specialmente nel caso in cui siano dei classificatori) sono detti **CATEGORIZATORI ELEMENTARI**: infatti, la filosofia che c'è dietro è quella di disporre di diversi classificatori che da soli fanno schifo ma che, se considerati tutti insieme, danno un risultato complessivo soddisfacente.

→ ESEMPIO: ALBERI DECISIONALI A 2 LIVELLI
(CHE EFFETTUANO 3 SCHEDE)

Esistono due approcci per costruire i predittori:

1) **BAGGING**: i predittori vengono definiti in modo independente l'uno dall'altro.

2) **BOOSTING**: i predittori vengono definiti in modo dipendente l'uno dall'altro. In particolare, il secondo predittore deve tendere a predire bene gli elementi predetti male dal primo, il terzo predittore deve tendere a predire bene gli elementi predetti complessivamente male dai primi due, e così via.

Bagging:

L'idea del Bagging è quella di diminuire la varianza del classificatore (ed è quello che spontaneamente succede quando si combinano i risultati di più predittori con le relative varianze di partenza). Si pensa ad esempio al singolo albero decisionale: cambiando anche solo un paio di elementi del training set, una suddivisione può cambiare, e questo avrà effetto su tutto il subalbero che sta sotto al nodo associato a quella suddivisione. È evidente che questo è indice di alta varianza: a un piccolo cambiamento del training set corrisponde una varianza causata nel comportamento del predittore (ricordi il discorso iniziale su bias e varianza?).

Ma come facciamo a creare N classificatori dello stesso tipo, independenti e diversi fra loro (dove "diversi" significa che fanno addestramento su training set differenti)?

↳ **BOOTSTRAP**: si fissa la dimensione m del training set da costruire a partire dal data set D iniziale. Ogni T_i è costituito effettuando m estrazioni con rimpiazzamento da D . Per m sufficientemente elevato, c'è una buona probabilità che tutti i training set T_i abbiano molti elementi differenti rispetto agli altri training set.

Poiché i predittori vengono costruiti in modo independente, i loro risultati vengono combinati tutti con lo stesso peso.

Vediamo ora perché il bagging funziona bene.

BAGGING CLASSIFICATION:

Supponiamo di avere B classificatori indipendenti che, per ciascun elemento x , hanno una probabilità di classificarlo male pari a E . Sia $B_0 \leq B$ il numero di classificatori che effettivamente sbagliano per l'elemento $x \Rightarrow B_0 \sim \text{Binomial}(B, E)$.

La classificazione data dall'intero ensemble è corretta se più della metà dei classificatori sbaglia.

$$\text{La probabilità che ciò avvenga è: } P(B_0 > \frac{B}{2}) = \sum_{k=\frac{B}{2}+1}^B \binom{B}{k} E^k (1-E)^{B-k}$$

che tende a 0 all'aumentare di B .

BAGGING REGRESSION:

Supponiamo che, per ciascun elemento x , il target vero di x è dato dalla funzione $h(x)$, mentre l'output di ciascuno degli m predittori è dato dalla funzione $y_i(x)$. Supponiamo inoltre di misurare gli errori con una quadratic loss.

$$\rightarrow \text{ERRORE ATTESO DEL MODELLO SUGLI ELEMENTI } x: E_x[(y_i(x) - h(x))^2] = E_x[\epsilon_i(x)^2]$$

$$\rightarrow \text{MEDIA DELL'ERRORE ATTESO SU TUTTI I MODELLI: } E_{\text{av}} = \frac{1}{m} \sum_{i=1}^m E_x[\epsilon_i(x)^2]$$

$$\rightarrow \text{ERRORE ATTESO DEL COMBINATO (DEL PREDITTORE GUARDALE): } E_c = E_x \left[\left(\frac{1}{m} \sum_{i=1}^m \epsilon_i(x) \right)^2 \right]$$

Se tutti i predittori sono scorrelati fra loro $\Rightarrow E_c = \frac{1}{m} E_{\text{av}}$, il che è buono ma è difficile avere tali similitudini.

RANDOM FOREST:

È uno dei modelli, non a rete neurale che hanno riscosso maggior successo. In particolare, è un metamodulo molto ensemble che usa l'appicchio bagging e che prende come predittori elementari degli alberi decisionali. L'unica sua particolarità è che non segue il meccanismo del bootstrap per costruire i classificatori i quali, invece, derivano tutti dal training set completo. Ciò che distingue i predittori elementari è una componente randomica nella costruzione degli alberi decisionali: infatti, a ogni passo, viene estratta casualmente un sottoinsieme di M feature e si stabilisce la feature migliore rispetto a cui fare la suddivisione solo ha quelle estratte.

01/06/2023

Boosting:

Il boosting è una procedura per combinare l'output di tanti classificatori deboli (i.e. che si com-

portano appena meglio dei classificatori naivi (i) generati in modo dipendente.

Per la classificazione l'output complessivo viene calcolato così:

$$y(x) = \text{Sgn} \left(\sum_{j=1}^m \alpha_j y_j(x) \right)$$

FUNZIONE DA ASSEGNARE ALL'OUTPUT DEL j-ESIMO CLASSIFICATORE

IN CASO DI REGRESSIONE SEMPRE
NON VIENE APPLICATA LA FUNZIONE SGNO

PER
OTTEN
IRE IN U

Mod

Tn

bas

cose

aff

del

DP

SO

z

fo

10

ADABOST:

È un modello che segue l'approccio Boosting. La sua idea è fissare un peso diverso a ciascun elemento del training set per effettuare l'addestramento del k-esimo classificatore. In particolare, se hanno dei pesi $w_i^{(1)}$ per il primo classificatore; in base alle sue predizioni (e quindi ai suoi errori), vengono scelti i pesi $w_i^{(2)}$ per il secondo classificatore; se il primo sbaglia la predizione sull'i-esimo elemento del training set $\Rightarrow w_i^{(1)} > w_i^{(2)}$, mentre se arrecca la predizione $\Rightarrow w_i^{(1)} < w_i^{(2)}$; e così via per tutti gli m classificatori del modello.

Vediamo il funzionamento in modo semplice:

Sia j l'indice del classificatore delle che siamo correntemente definendo e sia

$$\pi^{(j)} = \frac{\sum_{x_i \in E^{(j)}} w_i^{(j)}}{\sum_i w_i^{(j)}}$$

il rapporto tra la somma dei pesi degli elementi mal classificati dal j-esimo predittore e la somma totale dei pesi degli elementi per il j-esimo predittore, dove $E^{(j)} = \text{INSIEME DEGLI ELEMENTI MAL CLASSIFICATI DAL j-ESIMO PREDITTORE}$.

L_{Se} $\pi^{(j)} > \frac{1}{2} \Rightarrow$ si considera in realtà il predittore inverso, che restituisce il risultato opposto per tutti gli elementi.

Solo a questo punto ci si calcola il peso da assegnare al j-esimo classificatore in base agli elementi classificati correttamente rispetto a quelli classificati male:

$$\alpha_j = \frac{1}{2} \log \frac{1 - \pi^{(j)}}{\pi^{(j)}} > 0$$

Dopo che, per ciascun elemento x_i si aggiorna il peso corrispondente in questo modo:

$$w_i^{(j+1)} = w_i^{(j)} e^{\alpha_j y_i(x_i)}$$

$$\text{ONVERO: } w_i^{(j+1)} = \begin{cases} w_i^{(j)} e^{\alpha_j} & \text{se } x_i \in E^{(j)} \\ w_i^{(j)} e^{-\alpha_j} & \text{altrimenti} \end{cases}$$

Infine, si normalizzano tutti i $w_i^{(j+1)}$ ottenuti dividendoli per $\sum_{i=1}^m w_i^{(j+1)}$, in modo tale da ottenere una distribuzione di probabilità.

PERCHÉ ADABOOST FUNZIONA?

adaboost minimizza l'esponezionale loss $L(y(x), t) = e^{-t y(x)}$ (dove $y(x) = \text{sgn}(f(x))$), e lo fa in un particolare schema di classificatori: i MODELLI ADDITIVI.

Modelli additivi:

In generale sono semplicemente dei modelli definiti come una composizione additiva di predittori

$$\text{base: } y(x) = \sum_{j=1}^m \alpha_j \bar{y}_j(x)$$

dove $\bar{y}_j(x) := h(x; w_j)$ è l'output fornito dal solo j -esimo predittore per l'elemento x .

↳ PER LA CLASSIFICAZIONE BINARIA $y(x) \in \{-1, 1\}$

al setto, il nostro obiettivo è minimizzare il valore della funzione loss su tutti gli elementi

$$\text{del training set: } \min_{\alpha, W} L(t, y(x)) = \min_{\alpha, W} \sum_{i=1}^n L(t_i, \sum_{k=1}^m \alpha_k h(x_i; w_k))$$

dove $\alpha = \{\alpha_1, \dots, \alpha_m\}$, $W = \cup_{j=1}^m W_j$ (\rightarrow abbiamo n elementi di T e un predittore)

Il problema di minimizzazione così definito, però, è troppo dispendioso.

↓
SOLUZIONE: FORWARD STAGEWISE ADDITIVE MODELING - è un approccio greedy che cerca a minimizzare la funzione loss in più fasi diverse; in ciascuna fase vengono trattati insieme tutti e soli i parametri associati a uno specifico classificatore. In particolare, si segue questo algoritmo:

• Set $y_0(x) = 0$

• For $K = 1, \dots, m$:

• Compute $(\hat{\alpha}_K, \hat{w}_K) = \underset{\alpha, w}{\text{argmin}} \sum_{i=1}^n L(t_i, y_{K-1}(x_i) + \alpha_K h(x_i; w_K))$

• Set $y_K(x) = y_{K-1}(x) + \hat{\alpha}_K h(x; \hat{w}_K)$

COMPOSIZIONE ADDITIVA
DEL RISULTATO DEI
TRATTAMENTI PREDITTIVI

CORRELAZIONE TRA ADABOOST E I MODELLI ADDITIVI:

→ La minimizzazione della funzione loss esponenziale si rivelava essere del tutto equivalente alla minimizzazione di $\sum_{x_i \in \mathcal{E}^{(j)}} w_i^{(j)}$, che è esattamente quel che avviene in adaboost.

→ È possibile mostrare che, imponendo a zero la derivata della funzione loss rispetto ad α ,

$$\text{otteniamo: } \hat{\alpha}_j = \frac{1}{2} \log \frac{1 - \pi^{(j)}}{\pi^{(j)}} \quad \text{dove } \pi^{(j)} = \frac{\sum_{x_i \in \mathcal{E}^{(j)}} w_i^{(j)}}{\sum_{i=1}^n w_i^{(j)}}$$

↳ anche questo corrisponde a ciò che viene fatto in adaboost.

GRADIENT BOOSTING:

È un altro modello che segue l'approccio boosting. È applicabile sia alla regressione che alla clas-

Sia comunque mai mai inizialmente lo vedremo dal punto di vista della regressione.

Supponiamo di avere un training set con (x_i, t_i) , $i=1, \dots, n$, e supponiamo di voler trovare un modello che minimizza la loss quadratico.

IDEA: prendendo il predittore $y^{(1)}(x)$, per i vari elementi del training set abbiamo i RESIDUI (gli errori) $t_i - y^{(1)}(x_i) \equiv t_i - y^{(1)}(x)$. A questo punto si costruisce un nuovo training set i cui elementi sono di tipo $(x_i, t_i - y^{(1)})$ e si costruisce un secondo predittore $y^{(2)}(x)$ che cerca di predire i valori $t_i - y^{(1)}$. Poi si passa a un terzo predittore da addestrare mediante un ulteriore training set con $(x_i, t_i - y^{(2)})$, e così via. Il modello finale, per ogni elemento del training set, dovrà effettuare la somma degli output di tutti i predittori. Se ci pensi, tale somma finisce sempre più al valore esatto t_i man mano che il numero di predittori aumenta.

Il modello si chiama GRADIENT Boosting perché è legato al concetto di discesa del grad.

Supponiamo di avere come funzione loss $L(t, y) = \frac{1}{2}(t - y)^2$, e supponiamo di voler minimizzare il rischio empirico $R = \sum_{i=1}^n L(t_i, y_i)$ utilizzando direttamente gli output dei vari parametri da aggiornare (anziché i coefficienti w_i). Per ogni y_i (i.e. per ogni elemento del training set) consideriamo la derivata $\frac{\partial R}{\partial y_i} = y_i - t_i$: notiamo che è l'opposto del residuo, per cui praticamente stiamo andando a costruire un nuovo training set definito in questo modo: $(x_i, t_i - y_i) = (x_i, -\frac{\partial R}{\partial y_i})$ $i=1, \dots, n$.

Una definizione di questo tipo ha effettivamente senso perché:

Se $\frac{\partial R}{\partial y_i}$ è molto piccolo \Rightarrow vuol dire che il rischio empirico varia poco al variare del l'output del predittore \Rightarrow varcare l'output del predittore non porta a grandi benefici dal punto di vista delle qualità della predizione; infatti, in casi come questo, il predittore successivo viene costruito in modo tale da predire un valore nullo / molto basso per l'elemento x_i .

Se $\frac{\partial R}{\partial y_i}$ è positivo e non trascurabile \Rightarrow vuol dire che il rischio empirico aumenta all'aumentare di y_i \Rightarrow per migliorare la qualità della predizione, bisogna sottrarre qualcosa a y_i , cosa che infatti succede definendo il predittore successivo che viene addestrato per predire il valore $-\frac{\partial R}{\partial y_i} < 0$ per l'elemento x_i .

Chiaro, vale il discorso doppio per il caso $\frac{\partial R}{\partial y_i}$ negativo e non trascurabile.

06/06/2023

APPRENDIMENTO NON SUPERVISONATO

Riduzione della dimensionalità:

Spesso i dati ^{solitamente} presentano il problema del **CURSE OF DIMENSIONALITY**, in cui il numero di feature degli elementi è eccessivo rispetto al numero di elementi del data set. In tal caso, gli elementi risultano essere sparsi e, quindi, non hanno significato (\rightarrow non danno sufficiente informazione).

\hookrightarrow SOLUZIONE: rappresentare gli elementi del data set con meno feature.

Vediamo i possibili approcci:

• **FEATURE SELECTION**: a partire dal data set iniziale, si individuano le feature che danno più informazioni (e sono più scartate - ha poco) e si mantengono solo quelle, scartando tutte le altre.

• **FEATURE EXTRACTION**: a partire dalle feature iniziali, si ricavano delle nuove feature create che probabilmente abbiano una dimensionalità minore. Vi sono due modi in base a:

- **LINEAR PROJECTION**: i punti del data set vengono proiettati su un particolare iper piano, il che equivale a definire le nuove feature come delle combinazioni lineari delle feature di partenza. Vedi la **Fisher Discriminant Analysis** (esempi: Principal Component Analysis, probabilistic PCA, factor analysis).

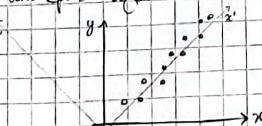
- **NON-LINEAR PROJECTION**: i punti del data set vengono proiettati su una struttura più complessa dell'iper piano (manifold learning) oppure vengono compressi mediante una rete neurale (autoencoder).

\hookrightarrow PER VERIFICARE CHE LA COMPRESSEIONE MANTENGA LA QUASI-TOTALE DELL'INFORMAZIONE, SI DEFINISCE UNA RETE NEURALE CHE MAPPI A RICORDARE NERI GLI ELEMENTI: SE RISULTANO quasi uguali A TUTTO IL PIANO \Rightarrow OK.

Principal Component Analysis (PCA):

Supponiamo di avere uno spazio degli elementi con un training set fatto in questo modo:

Non indichiamo con d il numero di feature originali e con p il numero di feature a seguito della proiezione.



Se nel ruotiamo il piano cartesiano nel modo tale da mantenere gli assi x_1, y_1 , otteniamo due nuove feature (rispetto x', y') di cui x' estremamente rappresentativa e y' quasi senza informazione. Quello che si fa è quindi rimuovere y' e rappresentare gli elementi esclusivamente con la feature x' .

Il senso di PCA è trovare le principali t.c. la distanza tra i punti originali e quelli proiettati sia minima.

PCA PER $d^1 = 0$:

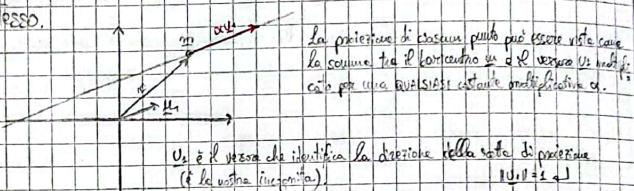
IDEA: proiettare tutti gli elementi del dataset su un unico punto \underline{x}_0 ; si vuole fare in modo che \underline{x}_0 approssimi al meglio il dataset, ovvero che la somma delle distanze tra \underline{x}_0 e ciascun punto \underline{x}_i sia la minima possibile:

$$\text{min}_{\underline{x}_0} J(\underline{x}_0) = \sum_{i=1}^n \|\underline{x}_0 - \underline{x}_i\|^2$$

E immediato mostrare che \underline{x}_0 è il baricentro del dataset: $\underline{x}_0 = \underline{m} = \frac{1}{n} \sum_{i=1}^n \underline{x}_i$

PCA PER $d^1 \neq 0$:

Qui tutti gli elementi del training set vengono proiettati su una retta che certamente passerà per il baricentro \underline{m} del dataset stesso.



Sia $\hat{x}_i = \alpha_i \underline{u}_1 + \underline{m}$ la proiezione dell'elemento \underline{x}_i sulla retta: dato il dataset $\underline{x}_1, \dots, \underline{x}_n$, ci piacerebbe trovare la retta tale da minimizzare la somma dei quadrati delle distanze tra gli \underline{x}_i e le rispettive proiezioni \hat{x}_i . Questa somma è così definita:

$$\begin{aligned} J(\alpha_1, \dots, \alpha_n, \underline{u}_1) &= \sum_{i=1}^n \|\hat{x}_i - \underline{x}_i\|^2 = \sum_{i=1}^n \|(\underline{m} + \alpha_i \underline{u}_1) - \underline{x}_i\|^2 = \sum_{i=1}^n \|\alpha_i \underline{u}_1 - (\underline{x}_i - \underline{m})\|^2 = \\ &= \sum_{i=1}^n \alpha_i^2 \|\underline{u}_1\|^2 + \sum_{i=1}^n \|\underline{x}_i - \underline{m}\|^2 - 2 \sum_{i=1}^n \alpha_i \underline{u}_1^\top (\underline{x}_i - \underline{m}) = \\ &= \sum_{i=1}^n \alpha_i^2 + \sum_{i=1}^n \|\underline{x}_i - \underline{m}\|^2 - 2 \sum_{i=1}^n \alpha_i \underline{u}_1^\top (\underline{x}_i - \underline{m}) \end{aligned}$$

$$\Rightarrow \frac{\partial}{\partial \alpha_i} J(\alpha_1, \dots, \alpha_n, \underline{u}_1) = 2 \alpha_i - 2 \underline{u}_1^\top (\underline{x}_i - \underline{m}) \rightarrow \text{SI ANNULLA PER } \alpha_i = \underline{u}_1^\top (\underline{x}_i - \underline{m})$$

Comunque sia questo mostra soltanto come le proiezioni debbano siano quelli ortogonali.

Per ottenere la direzione migliore per \underline{u}_1 :

Consideriamo la MATRICE DI COVARIANZA del dataset: $S = \frac{1}{n} \sum_{i=1}^n (\underline{x}_i - \underline{m})(\underline{x}_i - \underline{m})^\top$

• Sostituendo $\underline{u}_1^\top (\underline{x}_i - \underline{m})$ a α_i all'interno dell'espressione di $J()$:

$$\begin{aligned} J(\underline{u}_1) &= \sum_{i=1}^n [\underline{u}_1^\top (\underline{x}_i - \underline{m})]^2 + \sum_{i=1}^n \|\underline{x}_i - \underline{m}\|^2 - 2 \sum_{i=1}^n [\underline{u}_1^\top (\underline{x}_i - \underline{m})]^2 = - \sum_{i=1}^n [\underline{u}_1^\top (\underline{x}_i - \underline{m})]^2 + \sum_{i=1}^n \|\underline{x}_i - \underline{m}\|^2 = \\ &= - \sum_{i=1}^n \underline{u}_1^\top (\underline{x}_i - \underline{m})(\underline{x}_i - \underline{m})^\top \underline{u}_1 + \sum_{i=1}^n \|\underline{x}_i - \underline{m}\|^2 = - \underline{m} \underline{u}_1^\top S \underline{u}_1 + \sum_{i=1}^n \|\underline{x}_i - \underline{m}\|^2 \\ \Rightarrow \underset{\underline{u}_1}{\text{argmin}} J(\underline{u}_1) &= \underset{\underline{u}_1}{\text{argmin}} - \underline{m} \underline{u}_1^\top S \underline{u}_1 + \sum_{i=1}^n \|\underline{x}_i - \underline{m}\|^2 = \underset{\underline{u}_1}{\text{argmax}} \underline{u}_1^\top S \underline{u}_1 \end{aligned}$$

• Visto che \underline{u}_1 è un verso e ci interessa solo la sua direzione, aggiungiamo il vincolo $\|\underline{u}_1\|=1$.

\Rightarrow APPLICANDO IL LAGRANGIANO: $\underset{\underline{u}_1, \lambda_1}{\text{orizzontale}} \underline{u}_1^\top S \underline{u}_1 - \lambda_1 (\underline{u}_1^\top \underline{u}_1 - 1)$

$$\Rightarrow \frac{\partial}{\partial \mu_i} (\mu_i^T S \mu_i - \lambda_i (\mu_i^T \mu_i - 1)) = 2S\mu_i - 2\lambda_i \mu_i = 0 \Leftrightarrow S\mu_i = \lambda_i \mu_i \rightarrow \text{QUESTA E L'EQUAZIONE PER I VETTORI AUTONOMI DI } S.$$

\Rightarrow la funzione μ è effettivamente massimizzata quando μ_i è uguale all'autovettore di S associato all'autovettore maggiore di tutti.

PER $d > 1$:

Si procede allo stesso modo rispetto al caso $d=1$ solo che, quando si ha l'equazione degli autovettori e autovettori, si prendono d 'autovettori e, in particolare, i d 'autovettori associati agli autovettori maggiori. Così, l'iperpiano di proiezione verrà rappresentato da questi d 'autovettori.
 → NB: ogni autovettore rappresenta un elemento nel spazio originale.

COME SCRIUERE μ ?

Onde essenzialmente dal tableau degli autovettori: di fatto, il loro modello ci dice quanto inizialmente manteniamo prendendo l'autovettore associato a ciascuno di loro per costituire l'iperpiano di proiezione.

08/06/2023

Clustering addizionale:

DEA: suddivide il dato set in più partitioni in modo tale che ci raggruppino nella medesima partizione gli elementi: "simili" fra loro, che magari condividono una qualche caratteristica la totale in comune.
 → VIENE CHIARAMENTE DEFINITO UN CONCETTO DI DISTANZA (e.g. distanza euclidea).

→ Il numero di cluster (K) viene fissato a priori.

→ Ciascun cluster viene rappresentato da un prototipo, che è un punto fisso $m_j \in \mathbb{R}^d$.

→ È possibile indicare l'appartenenza del punto x_i al cluster j con dei flag binari: $\tau_{ij} = 1$

significa che $x_i \in$ cluster j ; $\tau_{ij} = 0$ significa che $x_i \notin$ cluster j .

L'algoritmo K-means per il clustering:

DEA: individuare i K prototipi m_j e assegnare ciascun elemento x_i a un determinato cluster in modo tale da minimizzare una certa funzione costo dipendente dalla distanza di ogni x_i dal prototipo relativo al cluster di appartenenza:

$$\min_{m_1, m_2, \dots, m_K} J(R, M) = \sum_{i=1}^n \sum_{j=1}^K \tau_{ij} \|x_i - m_j\|^2$$

L'algoritmo è iterativo (i.e. suddiviso in fasi), in cui ciascuna fase è suddivisa in due sotto-fasi:

1) Una volta fissati i K prototipi, ciascun x_i viene associato al prototipo m_j più vicino.

2) Si spostano i prototipi in modo tale da minimizzare la distanza tra gli x_i e i rispettivi m_j ; in pratica, i prototipi diventano il borsacchio di ciascun cluster.

→ Nella prima fase, i prototipi possono essere scelti casualmente per la sottofase (1).

Nelle fasi successive, si mantengono i prototipi fissati nella sottofase (2) della fase precedente e quello che si fa è ricongiungere gli elementi x_i a cluster differenti, se il precedente spostamento dei prototipi lo ha reso incosistente.

→ Con quest'astratta, anche partendo da K prototipi del tutto casuali, si riesce ad arrivare all'attuale ripartizione dei cluster ben rappresentati e, quindi, alla minimizzazione della funzione costo $J(R, M)$.

SELEZIONE DEL NUMERO DI CLUSTER K :

Esistono più approcci per stabilire il numero K di cluster:

1) Seguire una logica analoga al PCA, in cui si disegna l'andamento della funzione costo $J(R, M)$ al variare di K (circa un andamento decrescente) e si sceglie il punto in cui $J(R, M)$ è diminuita "sufficientemente" (ovviamente non possiamo far crescere troppo K , altrimenti andiamo incontro a un pesante overfitting).

2) Minimizzare delle nuove funzioni costo che dipendono sia da J che da K , e portare a minimizzazione quelle:

$$\bullet \text{ AKAICKE INFORMATION CRITERION (AIC): } AIC = 2K - 2 \ln J$$

$$\bullet \text{ BAYESIAN INFORMATION CRITERION (BIC): } BIC = K \ln n - 2 \ln J$$

3) Ricorrere ai metodi non-parametrici bayesiani, che provvedono la selezione del valore di K casualmente alla minimizzazione di J (o comunque all'algoritmo di assegnazione degli elementi ai cluster).

Clustering gerarchico:

È un tipo di clustering che si riferisce su una struttura ad albero binario detta DENDROGRAMMA.

→ Ciascun nodo dell'albero rappresenta un possibile cluster (dove le foglie determinano i singoli elementi del dataset).

• Ciascun taglio all'interno dell'albero rappresenta un possibile clustering.

→ L'altezza del minimo antenato comune tra due fratici è un'indicazione di quanto i due elementi sono simili fra loro.

APPROCCI PER COSTRUIRE IL DENSOGRAMMA:

• TOP-DOWN: si parte dall'intero dato set e lo si partitiona via via.

• BOTTOM-UP: si partite dai singoli elementi (le foglie) e mano a mano si aggregano. A ogni iterazione i cluster da accoppiare sono i più vicini secondo una metrica di distanza / somiglianza, che può essere:

→ Distanza tra i nodi più vicini dei due cluster: $d_{\text{eu}}(C_1, C_2) = \min_{x_i \in C_1, x_j \in C_2} d(x_i, x_j)$

→ Distanza tra i nodi più lontani dei due cluster: $d_{\text{cu}}(C_1, C_2) = \max_{x_i \in C_1, x_j \in C_2} d(x_i, x_j)$

→ Distanza media: $d_{\text{aa}}(C_1, C_2) = \frac{1}{|C_1||C_2|} \sum_{x_i \in C_1} \sum_{x_j \in C_2} d(x_i, x_j)$

Mistura di distribuzioni:

È una distribuzione di probabilità p data da una somma pesata di distribuzioni di probabilità

$$\text{di tipo } q \text{ dello stesso tipo: } p(x | \pi, \theta) = \sum_{j=1}^k \pi_j q(x | \theta_j) \quad \text{dove } \pi = (\pi_1, \dots, \pi_k) \\ \theta = (\theta_1, \dots, \theta_k)$$

Per affinare p sia effettivamente una distribuzione di probabilità, è necessario avere un π tale che

$$0 \leq \pi_j \leq 1 \quad \forall j = 1, \dots, K \quad \text{e} \quad \sum_{j=1}^k \pi_j = 1$$

→ Grazie alla mistura di distribuzioni è possibile definire delle distribuzioni di probabilità p più complesse di quella di base che già conosciamo.

A COSA C'ENTRA LA MISTURA DI DISTRIBUZIONI COL CLUSTERING?

Supponiamo di avere un dato set composto da m punti, e supponiamo di sapere che tali punti sono stati generati a partire da una mistura p di distribuzioni q . Ci chiediamo: quali sono i valori di π, θ che giustificano meglio l'estrazione del dato set che abbiamo? Con questa domanda stiamo introducendo un problema di massima incognigenza.

In realtà, qui ci manca un'informazione: quale distribuzione q ha generato ciascun punto: questa è una VARIABILE LATENTE del dato set. Quel che dovrà fare il nostro modello di mistura (detto MODELLO A VARIABILI LATENTI) è restituire la probabilità che un elemento sia generato da ciascuna distribuzione q .

Poiché ogni distribuzione q risulterà essere associata a un cluster, questo modello è famoso per la sua interpretazione più ricca: non solo si spiega semplicemente i punti in cluster, bensì indica la probabilità per i punti di appartenere a ciascun cluster.

In definitiva, abbiamo tre dati set:

- 1) DATA SET OBSERVATO (Ω): comprende le informazioni direttamente osservabili (i.e. quali sono i punti estratti).
- 2) DATA SET IMMAGINARIO: comprende le informazioni latenti (i.e. da quale distribuzione q è stato generato ciascun punto).
- 3) DATA SET COMPLETO = data set osservabile + data set immaginario.

09/06/2023

Stima dei parametri della mistura.

Come ricordiamo, i parametri π_i, θ_i di una mistura possono essere stimati con la massima verosimiglianza:

$$\text{Logaritmo } L(\theta, \pi | X) = \prod_{i=1}^n p(x_i | \theta_i) = \prod_{i=1}^n \prod_{j=1}^{m-1} \pi_j q(x_i | \theta_j) = \log \sum_{i=1}^n \log \left(\sum_{j=1}^m \pi_j q(x_i | \theta_j) \right)$$

$$\text{con i seguenti vincoli: } 0 \leq \pi_i \leq 1 \quad \forall i = 1, \dots, K \quad \wedge \quad \sum_{i=1}^K \pi_i = 1$$

$$\Rightarrow \text{LAGRANGIANO: } L(\theta, \pi | X) = \log \sum_{i=1}^n \log \left(\sum_{j=1}^m \pi_j q(x_i | \theta_j) \right) + \lambda \left(1 - \sum_{i=1}^K \pi_i \right)$$

Derivata parziale di π_i da parziale a 0:

$$\begin{aligned} \lambda = \frac{\partial L(\theta, \pi | X)}{\partial \pi_i} &= \frac{\partial}{\partial \pi_i} \left[\sum_{i=1}^n \log \left(\sum_{j=1}^m \pi_j q(x_i | \theta_j) \right) \right] = \sum_{i=1}^n \frac{\partial}{\partial \pi_i} \left[\log \left(\sum_{j=1}^m \pi_j q(x_i | \theta_j) \right) \right] = \\ &= \sum_{j=1}^m \frac{q(x_i | \theta_j)}{\sum_{k=1}^m \pi_k q(x_i | \theta_k)} = \sum_{j=1}^m \frac{\chi_j(x_i)}{\pi_i} = \frac{1}{\pi_i} \sum_{j=1}^m \chi_j(x_i) \\ \chi_j(x) &= \frac{\pi_j q(x | \theta_j)}{\sum_{k=1}^m \pi_k q(x | \theta_k)} \end{aligned}$$

Ora bisogna impostare a 0 anche la derivata rispetto a λ :

$$\frac{\partial L(\theta, \pi | X)}{\partial \lambda} = \frac{\partial}{\partial \lambda} \left(\log \sum_{i=1}^n \log \left(\sum_{j=1}^m \pi_j q(x_i | \theta_j) \right) + \lambda \left(1 - \sum_{i=1}^K \pi_i \right) \right) = 0 \iff \sum_{i=1}^K \pi_i = 1$$

Rimettendo insieme i punti:

$$\lambda = \frac{1}{\pi_i} \sum_{j=1}^m \chi_j(x_i) \iff \pi_i = \frac{1}{\lambda} \sum_{j=1}^m \chi_j(x_i) \iff \sum_{j=1}^m \pi_j = \frac{1}{\lambda} \sum_{j=1}^m \sum_{i=1}^n \chi_j(x_i) = 1$$

$$\iff \lambda = \sum_{i=1}^n \sum_{j=1}^m \chi_j(x_i) = \sum_{i=1}^n 1 = n \iff \pi_j = \frac{1}{n} \sum_{i=1}^n \chi_j(x_i)$$

$$\text{Derivata rispetto a } \theta_i \text{ da parte di } \phi: \frac{\partial \ell(\pi, \theta | X)}{\partial \theta_i} = \frac{\partial}{\partial \theta_i} \left[\sum_{j=1}^n \log \left(\sum_{k=1}^K \pi_k q(x_j | \theta_k) \right) \right] =$$

$$= \sum_{j=1}^n \frac{\pi_k q(x_j | \theta_k)}{\sum_{k=1}^K \pi_k q(x_j | \theta_k)} \frac{\partial \log q(x_j | \theta_k)}{\partial \theta_i} = \sum_{j=1}^n b_j(x_j) \frac{\partial \log q(x_j | \theta_i)}{\partial \theta_i} = 0$$

Si sviluppa in modo diverso a seconda di come è fatta la distribuzione q .

Colla fine della fiera, qui si determinano i π_i in funzione di $Y_i(x)$.

In definitiva, non trovare i valori di π, θ ottimali, si segue un approccio iterativo:

→ Si fissa una stima per π, θ .

→ Si deriva una stima per $Y_i(x)$.

→ Poi si deriva una nuova stima per π, θ .

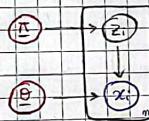
→ Di conseguenza si dà una nuova stima per $Y_i(x)$... e così via.

Misura come processi generativi:

Le misure possono essere schematizzate con una RETE BAYESIANA, che è una rappresentazione grafica che descrive le relazioni tra variabili aleatorie e misurabili:

→ E' V.A. NELLA PIAZZA, Y_h CHE NOTA DUE ALTERNATIVE E' STIMA E' SOTTO PER GENERARE IL QUOTIDIANO DATI SET.

→ X_i V.A. DI AFFERMENTO PER LA GENERAZIONE DEL QUOTIDIANO DATI SET (SISTEMA A 11 CLOUD).



- Z_i DIPENDE DA π ,
- X_i DIPENDE DA Z_i, θ
- X_i, Z_i SONO V.A.; Y_h E' SOTTO STIMA
- SI TRATTANO IN ISTANZA DI Z_i E' X_i .
- X_i E' OSSERVATA, Z_i E' LATENTE

SE $Z_i = k \Rightarrow X_i$ È CAMBIAMENTO DELLA DISTRIBUZIONE $q(x | \theta_k)$.

Formalizzando:

$$p(x | z=h, \theta, \pi) = p(x | z \neq h, \theta) = q(x | \theta_h)$$

$$\cdot \text{ MARGINALIZZAZIONE RISPETTO A } Z: p(z | \theta, \pi) = \sum_{h=1}^K p(z, z=h | \theta, \pi) = \sum_{h=1}^K p(z=h | \theta, \pi) p(z=z=h | \theta, \pi) =$$

$$= \sum_{h=1}^K p(x | z=h, \theta) p(z=h | \pi) = \sum_{h=1}^K q(x | \theta_h) p(z=h | \pi) = \sum_{h=1}^K \pi_h q(x | \theta_h) \Rightarrow \pi_h = p(z=h | \pi)$$

$$\Rightarrow \text{e allora: } Y_h(z) = \frac{\pi_h q(x | \theta_h)}{\sum_{j=1}^K \pi_j q(x | \theta_j)} = \frac{p(z=h) p(x | z \neq h)}{\sum_{j=1}^K p(z=j) p(x | z=j)} = p(z=h | x)$$

► $\pi_h = p(z=h)$ è una probabilità a priori su quale può essere stata la distribuzione q a generare un dato punto. → È DETTA SWIFTING COEFFICIENT.

► $Y_h(z) = p(z=h | x)$ è una probabilità a posteriori su quale può essere stata la distribuzione q a generare un dato specifico punto x . Il nostro obiettivo è proprio calcolare questo genio uno. → È DETTA FESTNESS.

Expectation-Maximization con la q gaussiana:

L'EXPECTATION-MAXIMIZATION è un algoritmo iterativo che ha lo scopo di stimare i parametri della distribuzione q e la responsability (proprio quello che ci serve).

Assumendo che avrei delle q gaussiane, partiamo dall'ipotesi esemplificativa in cui i valori $z_i = h$ sono tutti noti ($i = 1, \dots, n$); in tal caso ci basterà stimare i parametri delle q massimizzando la seguente log-likelihood:

$$l(\sum_{i=1}^n \mu, \Sigma | X, Z) = \log p(X, Z | \sum, \mu, \Sigma) = \log \prod_{i=1}^n \prod_{h=1}^k \pi_h^{z_{ih}} q(x_i | \mu_h, \Sigma_h)^{\pi_{ih}}$$

$$= \log \prod_{i=1}^n \prod_{h=1}^k \pi_h^{z_{ih}} N(x_i | \mu_h, \Sigma_h)^{\pi_{ih}} = \sum_{i=1}^n \sum_{h=1}^k \pi_{ih} (\log \pi_h + \log N(x_i | \mu_h, \Sigma_h))$$

Poi che in realtà i valori di π_{ih} non sono noti, dovranno essere sostituiti con le probabilità a posteriori delle z_i : ($p(z_i = h | x_i)$), ovvero con lo responsibility $\gamma_h(x_i)$. In tal modo, stiamo stimando il valore atteso risp. distrib. condizionata $p(Z|X)$ della log-likelihood:

$$E_{\text{prior}}[l(\sum, \mu, \Sigma | X, Z)] = \sum_{i=1}^n \sum_{h=1}^k \gamma_h(x_i) (\log \pi_h + \log N(x_i | \mu_h, \Sigma_h))$$

→ MASSIMIZZANDO QUESTO È POSSIBILE RICAVARE I PARAMETRI (H-STEP o MAXIMIZATION-STEP):

$$\pi_h = \frac{1}{m_x} \sum_{i=1}^n \gamma_h(x_i)$$

$$\sum_{i=1}^n \sum_{h=1}^k \gamma_h(x_i) (x_i - \mu_h)^T (x_i - \mu_h)$$

$$\mu_h = \frac{1}{m_x} \sum_{i=1}^n \gamma_h(x_i) x_i$$

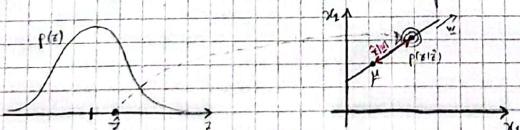
dove $m_x = \# \text{ ELEMENTI GENERATI DALLA K-ESIMA}$

al questo punto, i valori calcolati per i parametri portano a dei nuovi valori per $\gamma_h(x_i)$ e a un differente valore atteso $E_{\text{prior}}[l(\sum, \mu, \Sigma | X, Z)]$ (E-STEP o EXPECTATION-STEP).

In definitiva, l'algoritmo non è altro che un'iterazione tra E-STEP e M-STEP.

PCA probabilistico:

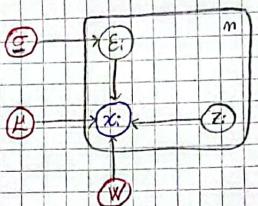
Qui l'appuccio generativo funziona così: viene generato un punto \hat{z} all'interno dello spazio di dimensione d seguendo una distribuzione di probabilità $p(z)$. Dopo che viene applicata una trasformazione in modo tale che \hat{z} cada su un iper piano di dimensione d all'interno di uno spazio a $d > d$ dimensioni; all'elemento generato viene ora aggiunto un 'rumore gaussiano', in modo tale da ottenere una distribuzione di probabilità $p(x | \hat{z})$.



- ← LA DISTRIBUZIONE $p(z)$ È UN PUNTO \hat{z} SONO LATENTI.
- ← $p(z)$ PUÒ ASSUNGERE ESSERE UNA $N(0, I)$.

→ IL RUMORE (CHE INDICHIAMO CON ϵ) PUÒ ESSERE CONSIDERATO COME AVENIRE UNA MATRICE DI COVARIANZA UGUALE PER TUTTI I PUNTI E PARTE A: $\sigma^2 I$.

Vediamo la rete bayesiana:



z_i è il punto che viene estratto nello spazio di dimensione $d_z \rightarrow$ subisce una trasformazione da un'aria (W) e una traslazione (μ) per essere rappresentato nello spazio di dimensione d_x . Infine, vi viene sommato un rumore gaussiano di deviazione standard σ .

Formalizzando:

- 1) ESTRAZIONE DELLA VARIABILE LATENTE $z \in \mathbb{R}^{d_z}$ DALLA DISTRIBUZIONE $p(z) = \frac{1}{(2\pi)^{d_z/2}} e^{-\frac{\|z\|^2}{2}}$
- 2) PROIEZIONE LINEARE DEL PUNTO SU \mathbb{R}^{d_x} : $y = Wz + \mu$
- 3) ESTRAZIONE DEL RUMORE $\epsilon \in \mathbb{R}^{d_x}$ DALLA DISTRIBUZIONE $p(\epsilon) = \frac{1}{(2\pi)^{d_x/2}} e^{-\frac{\|\epsilon\|^2}{2}}$

4) AGGIUNTA DEL RUMORE: $x = y + \epsilon$

Questo risulta in: $p(x|z) = \mathcal{N}(Wz + \mu, \sigma^2 I)$
Ma noi vogliamo apprendera $p(z|x)$

Si può dimostrare che, in un modo fatto gaussiano, è essa stessa una distribuzione gaussiana $\mathcal{N}(\mu_{zx}, \Sigma_{zx})$ con:

$$\mu_{zx} = W^T (W W^T + \sigma^2 I)^{-1} (x - \mu)$$

$$\Sigma_{zx} = \sigma^2 (I + W^T W)^{-1}$$

→ ANALISI FATTORIALE: ha complicando la matrice delle covarianze dell'errore in $\begin{bmatrix} \sigma_1^2 & \cdot \\ \cdot & \sigma_2^2 \end{bmatrix}$.