Appunti di Machine Learning

Maurizio Dimatteo
dimatteomaurizio@gmail.com

19 aprile 2014

Indice

Premesse

- Ho scritto questi appunti (non sono sbobinature) durante le lezioni di *Machine Learning* tenute dall'ing. Ostuni nell'a.a. 2013/2014.
- Ho fatto del mio meglio, ma non posso garantire che siano privi di imprecisioni teoriche e strafalcioni. Se ne trovi scrivimi all'indirizzo email in copertina.
- Il loro scopo principale è coprire gli argomenti che non sono trattati sul libro di riferimento del corso [3] o che sono trattati diversamente da come ci sono stati esposti a lezione. Per questo motivo gli appunti non sono esaustivi e per alcuni argomenti rimando direttamente al libro. Se hai voglia di completare le parti mancanti scrivimi tramite email.
- Questa dispensa è complementare alle lezioni in aula ed alle slide dell'ing. Ostuni. Studiarla non è condizione sufficiente (e tantomeno necessaria) per il superamento dell'esame.
- Né l'ing. Ostuni, né il prof. Di Noia hanno rivisto, corretto o autorizzato ufficialmente questi appunti.

Changelog

- 2014/04/19
  - Modificato: Funzione costo.
- 2014/04/15
  - Aggiunto: Derivazione della regola di aggiornamento dei pesi;
  - Ampliato: Regolarizzazione con norma ℓ^2;
  - Altre correzioni minori.
- 2014/03/17: Prima pubblicazione.

1 Introduzione

1.1 Ragionamento deduttivo e induttivo

Si dice *ragionamento deduttivo* il processo per cui, partendo dalla conoscenza generale del fenomeno, si arriva a conclusioni specifiche. Con riferimento ad una *blackbox* è come conoscere la funzione di trasferimento ed applicarla agli ingressi per ricavarne le uscite.

Il *ragionamento induttivo*, invece, si basa sull'osservazione di esempi specifici per ricavare una regola generale che giustifichi gli esempi visti. È equivalente ad osservare ingressi ed uscite di una *blackbox* ed individuare la migliore approssimazione della funziona di trasferimento.

Esempio

- Ragionamento deduttivo:
  - assioma generale: tutti i cani hanno quattro zampe, una coda ed abbaiano;
  - conclusione specifica: questo che ho davanti è un cane (perché ha quattro zampe, una coda ed abbaia).
- Ragionamento induttivo:
  - esempio specifico: questi animali hanno quattro zampe, una coda ed abbaiano;
  - descrizione generale: deve esistere una categoria di animali con queste caratteristiche (chiamati "cani").

1.2 Apprendimento supervisionato e non supervisionato

1.2.1 Supervisionato

Dato un *traininig set* in cui ciascuna istanza è costituita da un vettore di *feature* ($\mathbf{x}$) ed il corrispondente *output* (y), l'apprendimento supervisionato consiste nell'individuare la migliore funzione $h(x)$ (detta "ipotesi") che approssima la funzione *target* $f(x)$ (sconosciuta) tale che $y = f(x) \cong h(x)$ anche per valori esterni al *training set*.

Si possono individuare due categorie di algoritmi supervisionati:

- algoritmi di regressione;
- algoritmi di classificazione.

Algoritmi di regressione Forniscono un *output* numerico e tentano di approssimare una funzione non nota. Ad esempio potremmo istruire una macchina per emulare il comportamento di una funzione matematica $f(x)$. L'*output* $y = h(x)$ sarà numerico e rappresenterà una stima di $f(x)$.

Algoritmi di classificazione Forniscono un *output* qualitativo, specificano cioè la classe di appartenenza di ciascun campione in ingresso. Ad esempio un classificatore binario potrebbe essere addestrato con dati relativi ad altezza e peso di un certo numero di uomini e di donne. In fase di predizione, ricevuti in ingresso altezza e peso di una persona, l'uscita del classificatore sarà 1 o 0 a seconda che altezza e peso siano più consoni ad una donna o ad un uomo.

1.2.2 Non supervisionato

In questa categoria di algoritmi non abbiamo a disposizione le uscite corrispondenti a ciascun vettore di *feature*. Lo scopo della macchina è estrapolare delle informazioni significative dagli ingressi. A tal proposito si possono perseguire i seguenti due scopi principali.

Clusterizzazione Consiste nell'individuare gruppi di vettori di *input* con caratteristiche comuni. Ad esempio fornendo altezza e peso di 100 persone, tramite un algoritmo di clusterizzazione potremmo individuare una suddivisione dell'insieme dei campioni in due *cluster*: adulti e bambini, oppure uomini e donne. A priori, però, non conosciamo per ciascun vettore di *feature* la corrispondente classe di appartenenza.

Dimensionality reduction Consiste nell'individuare quali *feature* sono poco rilevanti ai fini della classificazione in modo tale da poterle trascurare e ridurre le dimensioni del problema.

1.3 Scelta della migliore ipotesi

Essa idealmente dovrebbe:

- minimizzare l'errore commesso sul *training set*;
- essere capace di generalizzare l'andamento del fenomeno, ovvero predire correttamente anche *output* non noti relativi ad ingressi mai visti durante l'addestramento.

I due obiettivi sono spesso incompatibili, per cui occorre trovare il giusto compromesso ed evitare *underfitting* ed *overfitting* (sottosezione 4.3).

1.3.1 Rasoio di Occam

Quella del Rasoio di Occam è una regola di massima che ci suggerisce come scegliere l'ipotesi più opportuna. Preso un insieme di ipotesi di grado massimo k, assumendo che rappresentino tutte adeguatamente il modello, il principio del Rasoio di Occam (che ha origini lontane dall'AI e dalla scienza) ci suggerisce di optare per quella più semplice. È ragionevole pensare che l'ipotesi più semplice sia quella di grado minore. Specifichiamo che un'ipotesi rappresenta adeguatamente un modello quando è consistente con l'insieme di esempi forniti, cioè quando si accorda con tutti i dati osservati[1]. Quindi nella scelta del modello occorre cercare un buon compromesso tra consistenza e bassa complessità dell'ipotesi.

1.4 Modelli discriminativi e generativi

Un'ulteriore classificazione dei modelli è tra *discriminativi* e *generativi*.

1.4.1 Discriminativi

Si tratta di modelli che, in seguito all'addestramento su un *training set*, quando ricevono in ingresso un vettore $\mathbf{x}$, sono in grado di valutare quale tra le possibili uscite è quella più probabile (ad esempio qual è la classe di appartenenza più probabile del campione ricevuto). Hanno lo scopo di stimare la probabilità condizionata $p(y|\mathbf{x})$, cioè la probabilità che, dato l'ingresso $\mathbf{x}$, esso generi l'uscita y (dove y potrebbe rappresentare la possibile classe del campione).

1.4.2 Generativi

Contrariamente ai precedenti, stimano la probabilità congiunta $p(\mathbf{x}, y)$. A partire da ciascun possibile *output* viene generato un modello probabilistico degli *input* ad esso associato. Ciò consente di generare, dato un possibile *output*, un *input* che con buona probabilità potrà essere associato ad esso.

[1] AIMA - Paragrafo 18.2

2 Algoritmi di regressione lineari

2.1 Regressione semplice lineare

Si chiama semplice perché prevede ingressi scalari (e non vettori di *feature*). È lineare perché, dato un *training set* di coppie (x_i, y_i), cerca la miglior retta che ne approssima l'andamento.

La generica ipotesi ha espressione:

$$h(x) = \theta_0 + \theta_1 x$$

dove:

- θ_0 prende il nome di *bias* e rappresenta l'intercetta con l'asse delle y;
- θ_1 è il coefficiente angolare della retta desiderata.

Lo scopo della fase di *training* è individuare i valori ottimi dei parametri θ_i (vedremo a breve cosa si intende per valori ottimi).

2.2 Regressione multipla lineare

È simile alla regressione semplice lineare, ma gli ingressi sono vettori di *feature*. Dunque non cerchiamo una retta, ma un iperpiano separatore, dalla generica espressione:

$$h(x) = \theta_0 + \sum_{i=1}^{n} \theta_i x_i.$$

Introducendo $x_0 = 1$ possiamo riscrivere la precedente come segue

$$h(x) = \sum_{i=0}^{n} \theta_i x_i = \theta^T \mathbf{x}$$

dove

- $\theta = (\theta_0, \theta_1, \ldots, \theta_n)^T$ rappresenta il vettore dei parametri da apprendere durante la fase di addestramento;
- n rappresenta la dimensione degli ingressi, cioè il numero di *feature*. Notiamo che per n ingressi occorre apprendere $n + 1$ parametri θ_i.

2.2.1 Funzione costo

Per individuare i valori ottimi dei θ_i costruiamo una funzione costo $J(\theta)$. Essa deve tener conto della differenza tra i valori di *output* forniti dal *training set* e gli stessi valori calcolati tramite l'ipotesi $h(x)$. Avrà quindi questa espressione:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(\mathbf{x^{(i)}}) - y^{(i)})^2 \tag{1}$$

dove:

- m è il numero di istanze del *training set*;
- $\frac{1}{2m}$ è un fattore di normalizzazione, il coefficiente 2 viene introdotto per semplificare le derivate parziali (vedi sottosezione 2.4);
- $(h_\theta(\mathbf{x^{(i)}}) - y^{(i)})$ è l'errore di predizione sul campione *i-esimo*, calcolato come differenza tra l'*output* stimato dal modello e quello letto dal *training set*.

Elevando al quadrato e sommando ciascun errore si ottiene una funzione $J(\theta)$ quadratica facilmente (e sicuramente) minimizzabile. Lo scopo della fase di apprendimento è quindi individuare per quali θ la funzione costo è minima:

$$\theta_{min} = \underset{\theta}{\operatorname{argmin}}(J(\theta)).$$

2.3 Metodo delle equazioni normali

Un primo approccio alla minimizzazione della funzione costo è quello analitico. Definiamo innanzitutto una matrice $\mathbf{X}$ di dimensioni $m \times (n+1)$ in cui ciascuna riga rappresenta un ingresso nel *training set*. Definiamo, inoltre, un vettore colonna $\mathbf{y}$ di m elementi, ciascuno dei quali rappresenta l'uscita associata all'ingresso *i-esimo* (nonché riga *i-esima* di $\mathbf{X}$):

$$\mathbf{X} = \begin{bmatrix} \mathbf{x^{(1)T}} \\ \mathbf{x^{(2)T}} \\ \vdots \\ \mathbf{x^{(i)T}} \\ \vdots \\ \mathbf{x^{(m)T}} \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(i)} \\ \vdots \\ y^{(m)} \end{bmatrix}$$

La funzione costo può essere riscritta sotto forma di prodotto tra matrici:

$$J(\theta) = \frac{1}{2}(\mathbf{X}\theta - \mathbf{y})^T(\mathbf{X}\theta - \mathbf{y}).$$

Cerchiamo il punto di minimo annullandone il gradiente:

$$\nabla J(\theta) = \mathbf{X}^T\mathbf{X}\theta - \mathbf{X}^T\mathbf{y} = 0$$

da cui si ricava che

$$\hat{\theta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}.$$

Questo metodo richiede il calcolo dell'inversa di $(\mathbf{X}^T\mathbf{X})$, di dimensioni $(n+1) \times (n+1)$. Quando il numero di *feature* è piuttosto grande, questo calcolo diventa troppo costoso: tipicamente la complessità computazionale del calcolo dell'inversa è $\mathcal{O}(n^3)$ e questo può rendere il metodo delle equazioni normali anche più lento di metodi iterativi come la discesa a gradiente[2]. Per tale motivo, quando il numero di *feature* è almeno nell'ordine di 10^5, si preferiscono metodi numerici come quello della discesa del gradiente che analizzeremo in seguito. Il metodo delle equazioni normali, tuttavia, essendo analitico non richiede la determinazione empirica di alcun parametro (contrariamente a quanto accade con il *learning rate* nella discesa a gradiente) e non è necessario effettuare alcun tipo di *feature scaling* (di cui parleremo in sezione 4.1.2).

In ultimo, consideriamo il caso in cui $\mathbf{X}^T\mathbf{X}$ non sia invertibile. Questo si verifica raramente, e se accade le cause più probabili sono due[3]:

- ci sono 2 o più *feature* linearmente dipendenti, per cui è sufficiente eliminarle tutte tranne una;
- il numero di esempi di addestramento è troppo basso rispetto al numero di *feature*. Questo problema può essere risolto eliminando alcune *feature* tramite tecniche di *feature selection* oppure adottando tecniche di regolarizzazione.

2.3.1 Dimostrazione

Deriviamo ora le espressioni mostrate nel paragrafo precedente[4]. Innanzitutto definiamo una funzione costo non normalizzata rispetto al numero di

[2] Coursera, Machine Learning (Andrew Ng), Lezione 4.6
[3] Coursera, Machine Learning (Andrew Ng), Lezione 4.7
[4] Questa parte non è stata trattata a lezione, chi si fida dei risultati può saltarla senza problemi.

campioni, che useremo durante la dimostrazione[5]:

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{m}(h_\theta(\mathbf{x^{(i)}}) - y^{(i)})^2. \tag{2}$$

Si vede facilmente che la precedente è uguale a:

$$J(\theta) = \frac{1}{2}(\mathbf{X}\theta - \mathbf{y})^T(\mathbf{X}\theta - \mathbf{y}). \tag{3}$$

Infatti è vero che:

$$\mathbf{X}\theta - \mathbf{y} = \begin{bmatrix} 1 & x_1^{(1)} & \dots & x_n^{(1)} \\ 1 & x_1^{(2)} & \dots & x_n^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(m)} & \dots & x_n^{(m)} \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix} - \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix} = \begin{bmatrix} h_\theta(\mathbf{x^{(1)}}) - y^{(1)} \\ h_\theta(\mathbf{x^{(2)}}) - y^{(2)} \\ \vdots \\ h_\theta(\mathbf{x^{(m)}}) - y^{(m)} \end{bmatrix}$$

E quindi:

$$\begin{aligned} &(\mathbf{X}\theta - \mathbf{y})^T(\mathbf{X}\theta - \mathbf{y}) \\ &= \begin{bmatrix}(h_\theta(\mathbf{x^{(1)}}) - y^{(1)}) & \dots & (h_\theta(\mathbf{x^{(m)}}) - y^{(m)})\end{bmatrix} \begin{bmatrix} h_\theta(\mathbf{x^{(1)}}) - y^{(1)} \\ \vdots \\ h_\theta(\mathbf{x^{(m)}}) - y^{(m)} \end{bmatrix} \\ &= \sum_{i=1}^{m}(h_\theta(\mathbf{x^{(i)}}) - y^{(i)})^2. \end{aligned}$$

Sostituendo la sommatoria in Equazione 2 con il prodotto tra matrici, si ottiene l'Equazione 3. A noi interessa trovare il minimo della funzione costo, cioè il punto in cui si annulla il suo gradiente:

$$\nabla J(\theta) = \begin{bmatrix} \frac{\partial J(\theta)}{\partial \theta_0} \\ \vdots \\ \frac{\partial J(\theta)}{\partial \theta_k} \\ \vdots \\ \frac{\partial J(\theta)}{\partial \theta_n} \end{bmatrix} = 0$$

[5] Minimizzare questa funzione costo o la sua variante normalizzata porta allo stesso risultato.

Similmente a quanto dimostreremo in seguito (sottosezione 2.4), si vede che la *k-esima* derivata parziale del gradiente assume la seguente espressione:

$$\begin{aligned}\frac{\partial J(\theta)}{\partial \theta_k} &= \sum_{i=1}^{m}(h_\theta(\mathbf{x^{(i)}}) - y^{(i)})x_k^{(i)} \\ &= \begin{bmatrix}(h_\theta(\mathbf{x^{(1)}}) - y^{(1)}) & \dots & (h_\theta(\mathbf{x^{(m)}}) - y^{(m)})\end{bmatrix} \begin{bmatrix} x_k^{(1)} \\ \vdots \\ x_k^{(m)} \end{bmatrix}.\end{aligned}$$

A questo punto riscriviamo per esteso l'espressione del gradiente[6]:

$$\nabla J(\theta) = \begin{bmatrix} \sum_{i=1}^{m}(h_\theta(\mathbf{x^{(i)}}) - y^{(i)}) \\ \sum_{i=1}^{m}(h_\theta(\mathbf{x^{(i)}}) - y^{(i)})x_1^{(i)} \\ \vdots \\ \sum_{i=1}^{m}(h_\theta(\mathbf{x^{(i)}}) - y^{(i)})x_n^{(i)} \end{bmatrix}$$

Questa può essere riscritta come prodotto tra matrici:

$$\begin{aligned}\nabla J(\theta) &= \begin{bmatrix} 1 & 1 & \dots & 1 \\ x_1^{(1)} & x_1^{(2)} & \dots & x_1^{(m)} \\ \vdots & \vdots & \ddots & \vdots \\ x_n^{(1)} & x_n^{(2)} & \dots & x_n^{(m)} \end{bmatrix} \begin{bmatrix} h_\theta(\mathbf{x^{(1)}}) - y^{(1)} \\ h_\theta(\mathbf{x^{(2)}}) - y^{(2)} \\ \vdots \\ h_\theta(\mathbf{x^{(m)}}) - y^{(m)} \end{bmatrix} \\ &= \mathbf{X}^T(\mathbf{X}\theta - \mathbf{y}) \\ &= \mathbf{X}^T\mathbf{X}\theta - \mathbf{X}^T\mathbf{y}\end{aligned}$$

Adesso risolviamo l'equazione di annullamento del gradiente:

$$\nabla J(\theta) = \mathbf{X}^T\mathbf{X}\theta - \mathbf{X}^T\mathbf{y} = 0$$

dalla quale si ricava che

$$\hat{\theta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}.$$

2.4 Metodo della discesa del gradiente

Come già accennato, il metodo della discesa del gradiente consente di trovare il minimo di una funzione per via numerica. Noi lo applicheremo alla funzione costo.

Data una funzione, sono note alcune proprietà del suo gradiente:

[6] Ricordiamo che $x_0^{(i)} = 1$ per $i = 1, \dots, m$.

1. esso è un vettore che punta sempre nel verso in cui la pendenza della funzione aumenta;

2. è esso stesso un indicatore della pendenza della curva.

Dalle precedenti consegue che il gradiente è tanto più grande quanto maggiore è la pendenza della curva e punta nel verso in cui aumenta la pendenza.

Esempio. È facile riscontrare questo comportamento con un paraboloide:

$$y = f(x_1, x_2) = x_1^2 + x_2^2$$

Il gradiente è

$$\nabla(f(x_1, x_2)) = (2x_1, 2x_2)$$

Calcolando il gradiente in un punto $P : (1, 2)$ otteniamo

$$\nabla(f(1, 2)) = (2, 4)$$

il quale è un vettore parallelo al piano su cui giace la curva di livello nel punto P, ha direzione perpendicolare alla tangente a tale curva in P e punta verso l'esterno del paraboloide, lì dove la pendenza aumenta.

Sfruttando queste conoscenze possiamo elaborare un metodo per aggiornare i coefficienti θ_i fino a raggiungere il minimo di $J(\theta)$. L'idea di base è quella di partire da θ_i casuali ed aggiornarli in maniera tale da spostarsi verso il minimo, cioè in verso opposto rispetto a quello del gradiente nel punto attuale (ricorda che ragioniamo su $J(\theta)$, quindi le $x^{(i)}$ sono solo coefficienti numerici, mentre le θ_i sono le variabili del problema). Per fare ciò si adotta la seguente regola di aggiornamento dei coefficienti:

$$\theta_k^{NEW} = \theta_k^{OLD} - \alpha \frac{\partial J(\theta)}{\partial \theta_k}$$

dove

- θ_k^{NEW} rappresenta il nuovo valore del *k-esimo* coefficiente, cioè quello che stiamo calcolando;
- θ_k^{OLD} è il valore dello stesso coefficiente all'iterazione precedente;
- α rappresenta un parametro detto *learning rate*, il cui valore è positivo e tipicamente minore di 1;
- $\frac{\partial J(\theta)}{\partial \theta_k}$ rappresenta la derivata parziale della funzione costo rispetto al coefficiente *k-esimo*.

Osserviamo il segno negativo prima della derivata parziale, poiché, come giustificato in precedenza, vogliamo spostarci nel verso opposto rispetto a quello del gradiente. Notiamo, inoltre, che α è un fattore moltiplicativo il cui scopo è "amplificare" l'effetto del secondo addendo, esso influisce dunque sull'ampiezza degli incrementi/decrementi dovuti alla derivata parziale. Osserviamo però che:

- un *learning rate* troppo basso può rallentare l'agoritmo e portare a lunghi tempi di convergenza;
- un *learning rate* troppo alto rischia di non farci trovare il minimo, poiché "saltiamo" da un lato all'altro della quadratica senza cadere nel centro.

In ultimo notiamo che il gradiente diminuisce man mano che ci avviciniamo al minimo, quindi all'inizio i decrementi subiti dai θ_i saranno più ampi, per poi diventare sempre più piccoli (ciò accadrebbe anche in assenza di α).

Notiamo, in ultimo, che questo algoritmo funziona correttamente in assenza di minimi locali. Se questi ci fossero si potrebbe riavviare l'algoritmo più volte, partendo da diversi θ_i casuali, e vedere se il punto finale di convergenza rimane pressoché invariato o se l'algoritmo è incappato in minimi locali. Ovviamente nel caso della funzione costo definita in precedenza questo problema non si presenta.

In ultimo esplicitiamo le derivate parziali:

$$\begin{aligned}\frac{\partial J(\theta)}{\partial \theta_k} &= \frac{\partial}{\partial \theta_k}\left(\frac{1}{2m}\sum_{i=1}^{m}(h_\theta(\mathbf{x^{(i)}}) - y^{(i)})^2\right) \\ &= \frac{1}{2m}\sum_{i=1}^{m}\frac{\partial}{\partial \theta_k}\left(h_\theta(\mathbf{x^{(i)}}) - y^{(i)}\right)^2,\end{aligned}$$

ricordandoci che $h(x) = \theta^T\mathbf{x}$ possiamo esaminare il singolo addendo della precedente sommatoria

$$\frac{\partial}{\partial \theta_k}\left(h_\theta(\mathbf{x^{(i)}}) - y^{(i)}\right)^2 = 2\left(h_\theta(\mathbf{x^{(i)}}) - y^{(i)}\right)\frac{\partial}{\partial \theta_k}h_\theta(\mathbf{x^{(i)}})$$

dove

$$\begin{aligned}\frac{\partial}{\partial \theta_k}h_\theta(\mathbf{x^{(i)}}) &= \frac{\partial}{\partial \theta_k}\theta^T x^{(i)} \\ &= \frac{\partial}{\partial \theta_k}(\theta_0, \theta_1, \ldots, \theta_k, \ldots, \theta_n)(1, x_1^{(i)}, x_2^{(i)}, \ldots, x_k^{(i)}, \ldots, x_n^{(i)})^T \\ &= (0, 0, \ldots, 1, \ldots, 0)(1, x_1^{(i)}, x_2^{(i)}, \ldots, x_k^{(i)}, \ldots, x_n^{(i)})^T \\ &= x_k^{(i)}.\end{aligned}$$

Ricomponendo i passaggi otteniamo

$$\frac{\partial}{\partial \theta_k}\left(h_\theta(\mathbf{x^{(i)}}) - y^{(i)}\right)^2 = 2\left(h_\theta(\mathbf{x^{(i)}}) - y^{(i)}\right)x_k^{(i)}$$

e tornando alla derivata parziale

$$\begin{aligned}\frac{\partial J(\theta)}{\partial \theta_k} &= \frac{1}{2m}\sum_{i=1}^{m}\frac{\partial}{\partial \theta_k}\left(h_\theta(\mathbf{x^{(i)}}) - y^{(i)}\right)^2 \\ &= \frac{1}{m}\sum_{i=1}^{m}(h_\theta(\mathbf{x^{(i)}}) - y^{(i)})x_k^{(i)}.\end{aligned}$$

A questo punto possiamo esplicitare l'equazione di aggiornamento dei coefficienti:

$$\theta_k^{NEW} = \theta_k^{OLD} - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(\mathbf{x^{(i)}}) - y^{(i)})x_k^{(i)}.$$

2.4.1 Batch gradient descent

È un algoritmo che utilizza la regola di aggiornamento dei coefficienti precedentemente descritta e può essere sintetizzato così:

- inizializza casualmente i θ_i
- finché non raggiungi la condizione di convergenza aggiorna i coefficienti:

$$\theta_k^{NEW} = \theta_k^{OLD} - \alpha\frac{1}{m}\sum_{i=1}^{m}(h(\mathbf{x^{(i)}}) - y^{(i)})x_k^{(i)} \qquad k = 0, 1, \ldots, n$$

La condizione di convergenza è tipicamente un valore massimo della funzione errore, al di sotto del quale ci accontentiamo del risultato ottenuto, o un numero massimo di iterazioni (oppure entrambe, in maniera tale che la seconda intervenga solo quando la prima non viene mai soddisfatta).

L'algoritmo si chiama *batch* perché per aggiornare ciascun singolo θ_k usa tutte le istanze nel *training set* (una sua variante, chiamata *mini batch*, ne usa solo una porzione). I limiti di questo approccio emergono quando il *training set* è molto grade e l'algoritmo tende ad essere lento (oltre a ciò, occorre caricare in memoria tutte le istanze per poter procedere nell'algoritmo, e ciò potrebbe essere impossibile su calcolatori modesti).

2.4.2 Stochastic gradient descent

A differenza dell'algoritmo *batch*, la versione stocastica[7] (anche detta "incrementale") aggiorna tutti i coefficienti dopo aver esaminato un singolo campione, solo in seguito passa al campione successivo. Di conseguenza occorre rivisitare leggermente la regola di aggiornamento dei pesi.

L'algoritmo può essere sintetizzato come segue:

- inizializza casualmente i θ_i
- ripeti fino a convergenza
  - per i = 1, . . . , m
    - per k=1, . . . , n

$$\theta_k^{NEW} = \theta_k^{OLD} - \alpha(h(\mathbf{x^{(i)}}) - y^{(i)})x_k^{(i)}$$

Notiamo che all'interno del ciclo `for` più interno ciascuna iterazione comporta l'aggiornamento di tutti i θ_i in base all'informazione proveniente da un singolo esempio. Questo implica che per ogni passo nel `for`, il punto attuale viene leggermente modificato tenendo conto di un solo esempio, quindi lo spostamento compiuto non va necessariamente nella direzione del minimo, ma cerca solo di fare un *fitting* migliore verso l'istanza attualmente analizzata. A causa di ciò la "traiettoria" con cui l'algoritmo si sposta verso il minimo non è diretta come per la versione *batch*, ma sembra più casuale. Per lo stesso motivo l'algoritmo potrebbe ritrovarsi a girare attorno al minimo senza raggiungerlo, pur restando in un intorno ragionevole. Tuttavia, poiché ciascun aggiornamento dei θ_i non implica la scansione di tutto il *training set*, l'algoritmo converge più velocemente (quando il *set* è grande).

La differenza sostanziale è questa: mentre nella versione *batch* occorre analizzare tutto il *training set* per compiere un piccolo passo, con quella *stochastic* si aggiorna leggermente il punto per ogni singola istanza che si analizza ed al termine dell'intero ciclo `for` lo spostamento verso il minimo potrebbe essere più significativo di quanto accadrebbe con una singola iterazione di algoritmo *batch*. Inoltre, nella variante stocastica la condizione di convergenza può essere verificata dopo ogni piccolo aggiornamento e non necessariamente dopo un intero ciclo.

[7] Coursera, Machine Learning (Andrew Ng), Lezione 17.2

2.4.3 Nota sull'aggiornamento dei coefficienti

Riprendiamo la formula di aggiornamento dei coefficienti:

$$\theta_k^{NEW} = \theta_k^{OLD} - \alpha \frac{\partial J(\theta)}{\partial \theta_k}$$

È importante notare che la derivata parziale dipende, ovviamente, da $h_\theta(x)$, la quale cambia al variare dei θ_i. Quando implementiamo concretamente questa regola, occorre calcolare prima tutte le derivate parziali, mantenendo invariati i coefficienti θ_i, memorizzare i nuovi coefficienti in variabili temporanee e poi aggiornare contemporaneamente tutti i coefficienti[8], ovvero:

$$\begin{aligned} temp1 &:= \theta_1 - \alpha \frac{\partial J(\theta)}{\partial \theta_1}; \\ temp2 &:= \theta_2 - \alpha \frac{\partial J(\theta)}{\partial \theta_2}; \\ &\dots \\ \theta_1 &:= temp1; \\ \theta_2 &:= temp2; \end{aligned}$$

La maniera *sbagliata* di fare ciò è la seguente:

$$\begin{aligned} \theta_1 &:= \theta_1 - \alpha \frac{\partial J(\theta)}{\partial \theta_1}; \\ \theta_2 &:= \theta_2 - \alpha \frac{\partial J(\theta)}{\partial \theta_2}; \\ &\dots \end{aligned}$$

Così facendo, infatti, abbiamo aggiornato θ_1 e per aggiornare θ_2 calcoliamo la derivata di $J(\theta)$ con *il nuovo valore di* θ_1, che è diverso da ciò che dovremmo fare. Questa variante, che abbiamo etichettato come errata, potrebbe funzionare, ma non è il *gradient descent* e potrebbe comportarsi diversamente da come ci aspettiamo.

[8] Coursera, Machine Learning (Andrew Ng), Lezione 2.5

2.5 Interpretazione probabilistica della regressione lineare

Siano $(\mathbf{x^{(i)}}, y^{(i)})$ le m istanze di un *training set*. Quando applichiamo la regressione lineare cerchiamo un'ipotesi $h(\mathbf{x}) = \theta^T \mathbf{x}$ che approssimi la relazione tra vettori di *feature* ed *output*. Nel fare ciò, ciascun valore di *output* del *training set* potrà essere espresso come somma di un valore calcolato tramite l'ipotesi ed un errore:

$$y^{(i)} = \theta^T \mathbf{x^{(i)}} + e^{(i)}. \tag{4}$$

Assumiamo che i campioni siano indipendenti, e dunque lo saranno anche gli errori $e^{(i)}$ che assumiamo essere anche identicamente distribuiti con distribuzione normale, a valor medio nullo e varianza σ^2:

$$p(e^{(i)}) = \mathcal{N}(0, \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(e^{(i)})^2}{2\sigma^2}} \qquad i = 1 \dots m. \tag{5}$$

Fissato un modello, e dunque i parametri θ, la probabilità che l'ingresso $\mathbf{x^{(i)}}$ generi l'uscita $y^{(i)}$ avrà la stessa distribuzione dell'errore $e^{(i)}$. Sostituendo $e^{(i)} = y^{(i)} - \theta^T \mathbf{x^{(i)}}$ (dall'Equazione 4) nell'Equazione 5 otteniamo:

$$p(y^{(i)} | \mathbf{x^{(i)}}; \theta) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(y^{(i)} - \theta^T \mathbf{x^{(i)}})^2}{2\sigma^2}} \qquad i = 1 \dots m.$$

A questo punto vogliamo trovare un'espressione della probabilità che, fissati i parametri θ, al *set* di ingressi $\mathbf{x^{(i)}}$ corrispondano le uscite $y^{(i)}$ per ogni $i = 1, \dots, m$. Questa probabilità viene chiamata "verosimiglianza" (*likelihood*) ed è così espressa:

$$L(\theta) = L(\theta; \mathbf{X}; \mathbf{y}) = p(\mathbf{y} | \mathbf{X}; \theta)$$

dove

- $\mathbf{X}$ rappresenta l'insieme degli *input* (possiamo vederlo come una matrice $m \times (n+1)$),
- $\mathbf{y}$ rappresenta il vettore delle uscite (di dimensioni $m \times 1$).

Avendo ipotizzato che tutti i campioni siano indipendenti, la probabilità dell'evento congiunto è la seguente:

$$L(\theta) = p(\mathbf{y} | X; \theta) = \prod_{i=1}^{m} p(y^{(i)} | \mathbf{x^{(i)}}; \theta) = \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(y^{(i)} - \theta^T \mathbf{x^{(i)}})^2}{2\sigma^2}}$$

Adottando il criterio di "massima verosimiglianza" siamo interessati al vettore θ che massimizza la funzione $L(\theta)$. Così facendo stiamo cercando i θ_i che danno vita all'ipotesi che ha maggior probabilità di restituire le uscite note quando si inseriscono i corrispondenti dati in *input*. Cerchiamo quindi

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}}\, L(\theta),$$

e poiché la funzione logaritmo è monotona crescente e non altera il punto di massimo, possiamo riformulare il problema come segue:

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}}\, L(\theta) = \underset{\theta}{\operatorname{argmax}}\, \log L(\theta) = \underset{\theta}{\operatorname{argmax}}\, l(\theta)$$

dove $\log L(\theta) = l(\theta)$. Questa formulazione consente di "trasformare" la produttoria in una sommatoria, più facile da gestire da un punto di vista computazionale:

$$\begin{aligned}
l(\theta) &= \log \prod_{i=1}^{m} p(y^{(i)}|\mathbf{x^{(i)}};\theta) \\
&= \sum_{i=1}^{m} \log \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(y^{(i)}-\theta^T\mathbf{x^{(i)}})^2}{2\sigma^2}} \\
&= \sum_{i=1}^{m} \left(\log \frac{1}{\sqrt{2\pi}\sigma} + \log e^{-\frac{(y^{(i)}-\theta^T\mathbf{x^{(i)}})^2}{2\sigma^2}} \right) \\
&= m \log \frac{1}{\sqrt{2\pi}\sigma} - \sum_{i=1}^{m} \frac{(y^{(i)} - \theta^T\mathbf{x^{(i)}})^2}{2\sigma^2} \\
&= m \log \frac{1}{\sqrt{2\pi}\sigma} - \frac{1}{2\sigma^2} \sum_{i=1}^{m} (y^{(i)} - \theta^T\mathbf{x^{(i)}})^2
\end{aligned}$$

Ritorniamo al problema di massimizzazione:

$$\begin{aligned}
\max_{\theta} l(\theta) &= \max_{\theta} \left(m \log \frac{1}{\sqrt{2\pi}\sigma} - \frac{1}{2\sigma^2} \sum_{i=1}^{m} (y^{(i)} - \theta^T\mathbf{x^{(i)}})^2 \right) \\
&= \max_{\theta} \left(-\frac{1}{2\sigma^2} \sum_{i=1}^{m} (y^{(i)} - \theta^T\mathbf{x^{(i)}})^2 \right) \\
&= \max_{\theta} \left(-\frac{1}{2} \sum_{i=1}^{m} (y^{(i)} - \theta^T\mathbf{x^{(i)}})^2 \right) \\
&= \min_{\theta} \left(\frac{1}{2} \sum_{i=1}^{m} (y^{(i)} - \theta^T\mathbf{x^{(i)}})^2 \right).
\end{aligned}$$

Confrontando l'ultima espressione con l'errore $J(\theta)$ (Equazione 1) e notiamo che il problema è identico, dato che l'unica differenza tra le due minimizzazioni è una costante di normalizzazione non influente ai fini del risultato.

$$\min_{\theta}\left(\frac{1}{2m}\sum_{i=1}^{m}(h(\mathbf{x^{(i)}})-y^{(i)})^2\right)=\min_{\theta}\left(\frac{1}{2}\sum_{i=1}^{m}(y^{(i)}-\theta^T\mathbf{x^{(i)}})^2\right) \tag{6}$$

2.6 Modelli lineari generalizzati (GLM)

A volte la linearità dei modelli esaminati può essere limitante. Ad esempio, quando i dati del *training set* non sono linearmente separabili, un modello lineare è del tutto inefficace. In tal caso, prima di passare a modelli non lineari, potrebbe essere interessante utilizzare delle *funzioni base*.

2.6.1 Funzioni base

Le funzioni base rappresentano trasformazioni non lineari che applichiamo a ciascun vettore di ingresso. Esse, tipicamente, aumentano il numero di *feature* e, tramite l'introduzione di non-linearità nei dati stessi, consentono di usare semplici modelli lineari per la classificazione.

Con riferimento alla regressione lineare, in condizioni normali l'ipotesi applicata ad un singolo ingresso è:

$$h(\mathbf{x^{(i)}})=\sum_{j=1}^{m}\theta_j x_j^{(i)}=\theta^T\mathbf{x^{(i)}}.$$

Per introdurre una non-linearità possiamo avvalerci di una funzione base $\phi(x)$ che applichiamo all'ingresso $x^{(i)}$. Il risultato $(\phi(x^{(i)}))$ sarà un vettore con più di n componenti, ottenute tramite combinazione non lineare delle n di partenza. Applichiamo quindi l'ipotesi (lineare) a tali vettori:

$$h(\mathbf{x^{(i)}})=\sum_{j=1}^{m}\theta_j\phi(\mathbf{x^{(i)}})_j=\theta^T\boldsymbol{\phi}(\mathbf{x^{(i)}}).$$

Esempio. Una possibile funzione di base potrebbe associare ad un ingresso a 3 dimensioni $< x_1, x_2, x_3 >$ un vettore a 9 dimensioni, così formato:

$$< x_1, x_2, x_3, x_1x_2, x_2x_3, x_1x_3, x_1^2, x_2^2, x_3^2 > .$$

3 Algoritmi di classificazione lineari

3.1 Regressione logistica

Il modello è nella forma:

$$h_\theta(\mathbf{x}) = \frac{1}{1 + e^{-\theta^T \mathbf{x}}}$$

Poiché la funzione logistica varia tra 0 ed 1, possiamo dare un'interpretazione probabilistica alla sua uscita. Fissata una soglia (*threshold*) fissiamo la classe di appartenenza a seconda che $h_\theta(x)$ sia maggiore o uguale alla soglia (classe 1) o minore (classe 0).

$$y = \begin{cases} 1 & \text{se } h_\theta(\mathbf{x}) \geq threshold \\ 0 & \text{se } h_\theta(\mathbf{x}) < threshold \end{cases}$$

3.1.1 Funzione costo

Similmente a quanto fatto per la regressione lineare, individuiamo un'espressione della funzione costo che tenga conto della differenza tra le uscite reali e quelle predette tramite il modello. Per trovare tale espressione calcoliamo direttamente la funzione *likelihood* ed applichiamo l'ipotesi di massima verosimiglianza (cioè massimiziamo la funzione di verosimiglianza).

Ricordiamo che $L(\theta)$ rappresenta la probabilità che, fissati i parametri θ, agli ingressi $x^{(i)}$ corrispondano le uscite $y^{(i)}$ note, ed assumiamo che i campioni di addestramento siano indipendenti ed identicamente distribuiti:

$$L(\theta) = L(\theta; X; \mathbf{y}) = p(\mathbf{y}|X;\theta) = \prod_{i=1}^{m} p(y^{(i)}|\mathbf{x^{(i)}};\theta). \tag{7}$$

Resta da capire quale sia la distribuzione di ciascun campione. Notiamo che l'uscita $y^{(i)}$ è binaria, quindi abbiamo due soli possibili eventi ($y^{(i)} = 1$ e $y^{(i)} = 0$), siamo quindi di fronte ad una distribuzione di Bernoulli $\mathcal{B}(p)$, dove p rappresenta la probabilità dell'evento $y^{(i)} = 1$ che ci apprestiamo a determinare. Abbiamo già detto che l'uscita della funzione logistica è compresa tra 0 ed 1, possiamo quindi interpretarla come la probabilità che all'ingresso $x^{(i)}$ corrisponda $y^{(i)} = 1$. Ad esempio, data una x in ingresso ad un modello "perfetto" la sua uscita dovrebbe essere 0 o 1, ma il nostro modello ci restituisce $h_\theta(x) = 0.7$; dovendo decidere se y sia 0 o 1, concludiamo che c'è il

70% di probabilità che sia 1. Riepiloghiamo dicendo che:

$$\begin{cases} p(y^{(i)} = 1|\mathbf{x^{(i)}}; \theta) = h_\theta(\mathbf{x^{(i)}}) \\ p(y^{(i)} = 0|\mathbf{x^{(i)}}; \theta) = 1 - h_\theta(\mathbf{x^{(i)}}) \end{cases}$$

che può essere scritto in maniera compatta come segue:

$$p(y^{(i)}|\mathbf{x^{(i)}}; \theta) = h_\theta(\mathbf{x^{(i)}})^{y^{(i)}} \cdot (1 - h_\theta(\mathbf{x^{(i)}}))^{1-y^{(i)}}.$$

Sostituiamo quest'ultima nell'Equazione 7:

$$L(\theta) = \prod_{i=1}^{m} h_\theta(\mathbf{x^{(i)}})^{y^{(i)}} \cdot (1 - h_\theta(\mathbf{x^{(i)}}))^{1-y^{(i)}}$$

e, come già fatto nella regressione lineare, passiamo al logaritmo:

$$\begin{aligned} l(\theta) &= \log \prod_{i=1}^{m} h_\theta(\mathbf{x^{(i)}})^{y^{(i)}} \cdot (1 - h_\theta(\mathbf{x^{(i)}}))^{1-y^{(i)}} \\ &= \sum_{i=1}^{m} \left(y^{(i)} \log h_\theta(\mathbf{x^{(i)}}) + (1 - y^{(i)}) \log (1 - h_\theta(\mathbf{x^{(i)}})) \right). \end{aligned}$$

Questa è la funzione che ci interessa massimizzare, nel dettaglio, però, minimizziamo il suo opposto normalizzato rispetto al numero di campioni di addestramento (che prende il nome di *cross entropy error function*):

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} \left(y^{(i)} \log h_\theta(\mathbf{x^{(i)}}) + (1 - y^{(i)}) \log (1 - h_\theta(\mathbf{x^{(i)}})) \right).$$

In conclusione, quindi, ci interessa il θ ottimo che minimizzi la precedente:

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}}\, J(\theta).$$

La funzione costo appena definita non può essere studiata da un punto di vista analitico, ma possiamo vedere quale contributo fornisce all'errore ciascun addendo:

$$e^{(i)} = y^{(i)} \log h_\theta(\mathbf{x^{(i)}}) + (1 - y^{(i)}) \log (1 - h_\theta(\mathbf{x^{(i)}}))$$

dove $e^{(i)}$ rappresenta il contributo all'errore apportato da un *i-esimo* campione di addestramento. Distinguamo due casi:

- il campione appartiene alla classe 1 ($y^{(i)} = 1$);
- il campione appartiene alla classe 0 ($y^{(i)} = 0$).

Nel primo caso il singolo addendo assume l'espressione:

$$e^{(i)} = -\log h_\theta(\mathbf{x^{(i)}})$$

In Figura 1 (linea rossa) è mostrato il suo andamento, concludiamo che:

- se il campione è di classe 1, ma il classificatore restituisce un valore basso, l'errore commesso è alto. Al limite accade che:

$$h_\theta(\mathbf{x^{(i)}}) = 0 \Rightarrow e^{(i)} \to \infty$$

- se il campione è di classe 1 ed il classificatore restituisce un valore alto, l'errore commesso è basso. Al limite accade che:

$$h_\theta(\mathbf{x^{(i)}}) = 1 \Rightarrow e^{(i)} \to 0$$

Nel secondo caso ($y^{(i)} = 0$), il contributo del singolo campione sarà:

$$e^{(i)} = -\log\left(1 - h_\theta(\mathbf{x^{(i)}})\right)$$

In Figura 1 (linea blu) è mostrato il suo andamento, concludiamo che:

- se il campione è di classe 0 ed il classificatore restituisce un valore basso, l'errore commesso è basso. Al limite accade che:

$$h_\theta(\mathbf{x^{(i)}}) = 0 \Rightarrow e^{(i)} \to 0$$

- se il campione è di classe 0, ma il classificatore restituisce un valore alto, l'errore commesso è alto. Al limite accade che:

$$h_\theta(\mathbf{x^{(i)}}) = 1 \Rightarrow e^{(i)} \to \infty$$

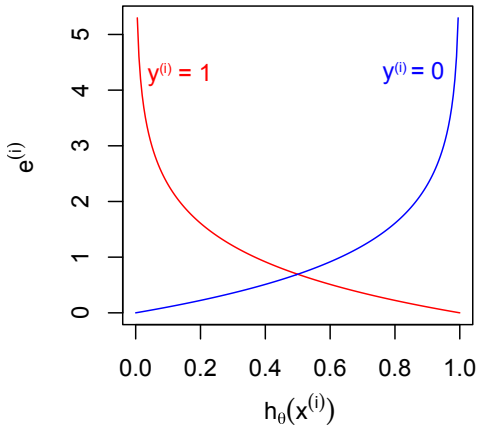


Figura 1: Contributi all'errore nella regressione logistica

3.2 Metodo della discesa del gradiente

Poiché non esiste un metodo analitico per minimizzare la funzione costo, l'unica l'alternativa è un metodo numerico, come quello basato sulla discesa del gradiente. Come già visto, questo metodo si basa sulla seguente regola di aggiornamento dei pesi:

$$\theta_k^{NEW} = \theta_k^{OLD} - \alpha \frac{\partial J(\theta)}{\partial \theta_k}.$$

Si vede facilmente che per la derivata della funzione logistica vale la seguente proprietà:

$$g'(z) = \frac{\partial}{\partial z}\left(\frac{1}{1+e^{-z}}\right) = g(z)(1-g(z)).$$

Sfruttando la precedente nel calcolo delle derivate di $J(\theta)$ si vede che la regola di aggiornamento dei pesi è identica a quella della regressione lineare, cioè:

$$\theta_k^{NEW} = \theta_k^{OLD} - \alpha \frac{1}{m} \sum_{i=1}^{m} (h(\mathbf{x^{(i)}}) - y^{(i)}) x_k^{(i)}.$$

Essa può essere applicata sia per il *batch gradient descent* che per le sue varianti stocastiche.

3.2.1 Derivazione della regola di aggiornamento dei pesi

Vediamo ora quali sono i passaggi che portano all'espressione della regola di aggiornamento dei pesi[9]. Come già detto, la derivata della funzione logistica gode della seguente proprietà:

$$g'(z) = g(z)(1 - g(z)).$$

Partendo da essa, calcoliamo la derivata parziale di $h_\theta(\mathbf{x})$, che tornerà utile in seguito:

$$\frac{\partial h_\theta(\mathbf{x})}{\partial \theta_k} = h_\theta(\mathbf{x})(1 - h_\theta(\mathbf{x}))\frac{\partial(\theta^T \mathbf{x})}{\partial \theta_k}$$

dove, come già visto in sottosezione 2.4, risulta che:

$$\frac{\partial(\theta^T \mathbf{x})}{\partial \theta_k} = x_k$$

e quindi

$$\frac{\partial h_\theta(\mathbf{x})}{\partial \theta_k} = h_\theta(\mathbf{x})(1 - h_\theta(\mathbf{x}))x_k.$$

Passiamo ora al calcolo della derivata parziale di $J(\theta)$:

$$\frac{\partial J(\theta)}{\partial \theta_k} = -\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}\frac{\partial}{\partial \theta_k}(\log h_\theta(\mathbf{x^{(i)}})) + (1 - y^{(i)})\frac{\partial}{\partial \theta_k}(\log(1 - h_\theta(\mathbf{x^{(i)}})))\right). \tag{8}$$

Esaminiamo separatamente le due derivate:

$$\frac{\partial}{\partial \theta_k}\log h_\theta(\mathbf{x^{(i)}}) = \frac{1}{h_\theta(\mathbf{x^{(i)}})}h_\theta(\mathbf{x^{(i)}})(1 - h_\theta(\mathbf{x^{(i)}}))x_k^{(i)} = (1 - h_\theta(\mathbf{x^{(i)}}))x_k^{(i)}$$

$$\frac{\partial}{\partial \theta_k}\log(1 - h_\theta(\mathbf{x^{(i)}})) = \frac{1}{1 - h_\theta(\mathbf{x^{(i)}})}(-h_\theta(\mathbf{x^{(i)}}))(1 - h_\theta(\mathbf{x^{(i)}}))x_k^{(i)} = -h_\theta(\mathbf{x^{(i)}})x_k^{(i)}$$

Sostituiamo le precedenti nell'Equazione 8, raccogliamo x_k e sviluppiamo il prodotto per ottenere la derivata finale:

[9] Approfondimento non trattato a lezione

$$\begin{aligned}\frac{\partial J(\theta)}{\partial \theta_k} &= -\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}(1-h_\theta(\mathbf{x^{(i)}}))x_k^{(i)} + (1-y^{(i)})(-h_\theta(\mathbf{x^{(i)}}))x_k^{(i)}\right) \\ &= -\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}(1-h_\theta(\mathbf{x^{(i)}})) + (1-y^{(i)})(-h_\theta(\mathbf{x^{(i)}}))\right)x_k^{(i)} \\ &= \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(\mathbf{x^{(i)}}) - y^{(i)}\right)x_k^{(i)}\end{aligned}$$

3.3 Regressione logistica: caso multiclasse

Con la regressione logistica siamo in grado di realizzare un classificatore binario. Se le classi sono più di due, si possono adottare due approcci:

1. usare una funzione logistica a più livelli (che non approfondiremo);
2. usare più funzioni logistiche in un approccio *one-vs-all*, che ci apprestiamo ad approfondire.

Supponiamo di dover distinguere k classi. Per fare ciò addestreremo k classificatori (quindi avremo k funzioni logistiche) ciascuno dei quali sarà in grado di identificare come classe 1 una classe desiderata, e tutte le altre $k-1$ saranno riconosciute come classe 0. Una volta addestrati i k classificatori, per classificare un nuovo ingresso lo diamo in *input* a tutte le funzioni logistiche, ciascuna restituirà un valore tra 0 ed 1, quello più alto determinerà la classe di appartenenza dell'ingresso (cioè ci dirà quale classificatore ha dato la risposta più alta, presumibilmente quello specializzato sulla reale classe di appartenenza dell'ingresso). Di fatto in questo approccio interpretiamo l'uscita dell'*i-esimo* classificatore come la probabilità che l'ingresso appartenga alla classe *i-esima*.

3.4 Regolarizzazione

Uno dei problemi in cui è possibile incorrere è l'*overfitting* e tramite la "regolarizzazione" possiamo evitarlo.

L'idea di base è che, fissato il grado di un modello, il suo andamento sarà tanto più oscillatorio quanto più grandi saranno i θ_i. La regolarizzazione, quindi, si propone di prevenire l'*overfitting* penalizzando i θ_i grandi, senza agire direttamente sul grado del modello.

3.4.1 Regolarizzazione con norma ℓ^2

Il concetto di base è molto semplice: sommiamo alla funzione costo un qualcosa che penalizzi i θ_i grandi, nella fattispecie il nostro "qualcosa" è il quadrato della norma in modulo 2 del vettore θ (escluso θ_0 poiché è l'intercetta e non contribuisce all'andamento della funzione).

Nel caso della regressione lineare, la funzione costo regolarizzata prende il nome di *Ridge Regression* ed è la seguente:

$$J(\theta) = \frac{1}{2m}\left(\sum_{i=1}^{m}(h(\mathbf{x^{(i)}}) - y^{(i)})^2 + \lambda\sum_{j=1}^{n}\theta_j^2\right) \tag{9}$$

dove:

- $\sum_{j=1}^{n}\theta_j^2 = ||\theta||_2^2$; sommare questo contributo è equivalente a minimizzare $J(\theta)$ originale ponendo un vincolo sul valore massimo della norma;
- λ è detto parametro di regolarizzazione e determina l'influenza del fattore di regolarizzazione: più è grande, più piccoli saranno i θ e più smussata sarà l'ipotesi.

Nel caso della regressione logistica, questo approccio porta a riformulare la funzione costo come segue:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}\log h_\theta(\mathbf{x^{(i)}}) + (1-y^{(i)})\log\left(1-h_\theta(\mathbf{x^{(i)}})\right)\right) + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2.$$

In maniera simile a quanto già fatto, possiamo calcolare le derivate delle funzioni costo regolarizzate e dedurre la regola di aggiornamento dei pesi. Si vede facilmente che, sia nel caso di regressione lineare che logistica, essa diventa:

$$\begin{cases}\theta_0^{NEW} = \theta_0^{OLD} - \alpha\dfrac{1}{m}\displaystyle\sum_{i=1}^{m}\left(h_\theta(\mathbf{x^{(i)}}) - y^{(i)}\right)x_0^{(i)} \\ \theta_k^{NEW} = \theta_k^{OLD} - \alpha\left[\dfrac{1}{m}\displaystyle\sum_{i=1}^{m}\left(h_\theta(\mathbf{x^{(i)}}) - y^{(i)}\right)x_k^{(i)} + \dfrac{\lambda}{m}\theta_k\right] \quad k = 1, 2, \ldots, n\end{cases}$$

La regola di aggiornamento di θ_0 rimane invariata, come ci aspetteremmo sapendo che esso non è coinvolto nella regolarizzazione. Per tutti gli altri coefficienti possiamo riscrivere la regola come segue:

$$\theta_k^{NEW} = \theta_k^{OLD}\left(1 - \alpha\frac{\lambda}{m}\right) - \alpha\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(\mathbf{x^{(i)}}) - y^{(i)}\right)x_k^{(i)} \quad k = 1, 2, \ldots, n$$

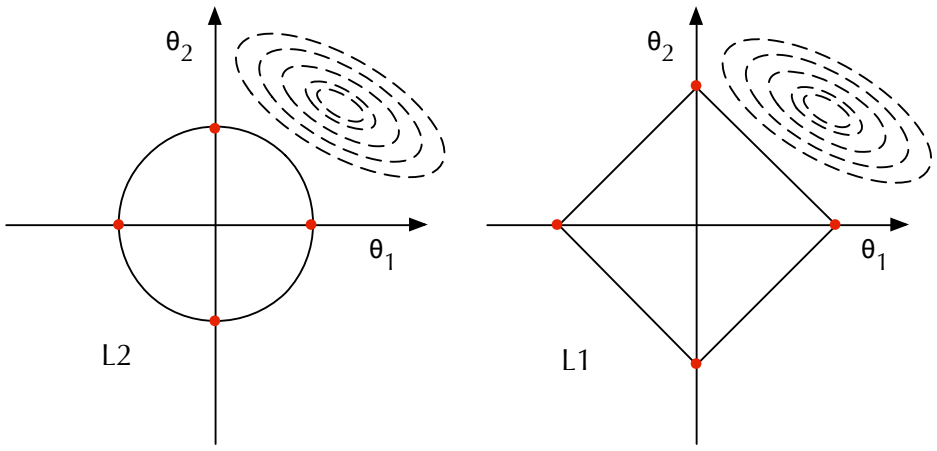


Figura 2: Regolarizzazione con norma ℓ^1 ed ℓ^2

Notiamo che la seconda parte della regola è identica alla versione senza regolarizzazione. In condizioni "tipiche" accade che $\alpha < 1$, $\lambda > 0$ ed m è piuttosto grande. Questo fa sì che il termine $\left(1 - \alpha\frac{\lambda}{m}\right)$ sia minore di 1. Ciò ci fornisce un'interessante interpretazione della nuova regola di aggiornamento. Infatti è come se l'algoritmo aggiornasse θ_k applicando la classica regola (senza regolarizzazione) a partire da una versione scalata del precedente valore di θ_k (poiché viene moltiplicato per una quantità minore di 1).

3.4.2 Regolarizzazione con norma ℓ^1

Un'ulteriore tecnica di regolarizzazione (detta *lasso*) prevede di usare la norma ℓ^1 piuttosto che il quadrato della norma ℓ^2. Le espressioni precedenti rimangono invariate, ad eccezione dell'ultima sommatoria che viene sostituita da:

$$\ell^1 = \sum_{j=1}^{n} |\theta_j|.$$

Questo tipo di sostituzione, però, implica che la nuova funzione costo $J(\theta)$ non sia più derivabile con continuità (a causa di ℓ^1).

Ma allora perché preferire ℓ^1 ad ℓ^2?

3.4.3 ℓ^2 vs ℓ^1

Le persone serie (non noi) hanno dimostrato le seguenti conclusioni.

Come già detto, l'introduzione della norma nella funzione costo equivale ad imporre un limite al suo valor massimo. Possiamo quindi riscrivere i due problemi come segue (ignoriamo i fattori di normalizzazione m al denominatore, senza che ciò cambi i risultati):

$$\text{Norma } \ell^2: \quad \min \sum_{i=1}^{m} (\dots)^2 \tag{10}$$

$$\text{s.t.} \quad ||\theta||_2^2 < t \tag{11}$$

$$\text{Norma } \ell^1: \quad \min \sum_{i=1}^{m} (\dots)^2 \tag{12}$$

$$\text{s.t.} \quad ||\theta||_1 < t \tag{13}$$

Per capire i risultati ottenuti dalla gente seria, riconduciamoci al caso $n = 2$, per cui le precedenti diventano:

$$\text{Norma } \ell^2: \quad \min \sum_{i=1}^{m} (\dots)^2 \tag{14}$$

$$\text{s.t.} \quad \theta_1^2 + \theta_2^2 < t \tag{15}$$

$$\text{Norma } \ell^1: \quad \min \sum_{i=1}^{m} (\dots)^2 \tag{16}$$

$$\text{s.t.} \quad |\theta_1| + |\theta_2| < t \tag{17}$$

Sappiamo che:

- $\sum(\dots)^2$ rappresenta un paraboloide (il cui *contour plot* è tratteggiato in Figura 2);
- $\theta_1^2 + \theta_2^2 < t$ delimita l'area interna ad una circonferenza centrata nell'origine e di raggio $\sqrt{t}$ (vedi Figura 2);
- $|\theta_1| + |\theta_2| < t$ delimita l'area interna di un rombo di diagonale $2t$ (vedi Figura 2).

Lo scopo della minimizzazione è trovare θ_1 e θ_2 che minimizzano il paraboloide e ricadano nelle aree delimitate dal dominio.

Si vede che, se questi punti ricadono nelle intersezioni tra il perimetro del dominio ed uno dei due assi, uno dei due θ_i sarà nullo (punti rossi in Figura 2). Se $\theta_i = 0$ significa che l'attributo x_i non comparirà nell'espressione del modello, ed avremo operato una *feature selection* "naturale". Il motivo per cui la norma ℓ^1 viene usata è perché si dimostra che comporta una maggiore probabilità di ricadere nelle intersezioni appena descritte, e quindi è molto utile per problemi con un gran numero di attributi (per contro occorre trovare un metodo alternativo alla discesa del gradiente che richiede la derivabilità della funzione costo).

4 Progettazione di un sistema di ML

La progettazione di un sitema di *machine learning* può essere sintetizzata nelle seguenti fasi:

1. analisi e *pre-processing* dei dati;
2. *feature* selection;
3. scelta del modello ed addestramento;
4. validazione del modello sul *validation set* ed eventualmente si ritorna al passo precedente;
5. valutazione delle prestazioni su un *test set*;
6. adozione del modello nel contesto finale.

Analizziamo di seguito le singole fasi.

4.1 Analisi e pre-processing dei dati

4.1.1 Pulizia dei dati

I dati ricevuti e provenienti dal mondo reale potrebbero essere sporchi, ovvero:

- potrebbero essere incompleti, ad esempio quando alcuni valori di un attributo sono mancanti;
- potrebbero essere inaccurati o discostarsi molto da quanto ci si aspetterebbe (ad esempio a causa di una digitazione errata da parte di un operatore umano).

Poiché vale la regola GIGO (Garbage In - Garbage Out) secondo cui un modello addestrato con dati sporchi produrrà risultati non affidabili, è necessario individuare tutte queste anomalie e ripulire i dati. Per quanto questa operazione non segua sempre un flusso standard, ma dipende dai particolari dati e dalle proprietà che possiamo osservare, si possono distinguere alcune operazioni comuni:

- rimozione degli *outlier*: per *outlier* si intendono gli esempi il cui valore d'uscita supera il terzo quartile (sezione 4.1.1);
- rimozione dei duplicati: un campione d'addestramento duplicato non aggiunge alcuna informazione utile;

- rimozione del rumore: cioè delle informazioni inutili (ad esempio l'ID utente è tipicamente un valore non significativo ai fini della classificazione).

Quantili e quartili L'α-quantile, con $\alpha \in [0, 1]$, identifica quel valore x_α tale che una quota α della popolazione delle x sia $\leq x_\alpha$. Esistono diversi quantili tipicamente utilizzati, e sono:

- quartili, dividono la distribuzione in 4 parti uguali. Ciascun quartile rappresenta il 25% della popolazione (quindi il terzo quartile è quel valore di x di cui il 75% della popolazione è più piccolo);
- decili, dividono la popolazione in 10 parti uguali (ciascun decile rappresenta il 10% della popolazione);
- percentili, dividono la popolazione in 100 parti uguali (ciascuna parte rappresenta l'1% della popolazione).

4.1.2 Pre-processing

Oltre ad aver pulito i dati è necessario pre-elaborarli prima di darli in pasto al classificatore, alcune operazioni comuni sono:

- discretizzazione ed aggregazione, ad esempio potremmo raggruppare gli ingressi in base all'attributo "età" in "adulti", "ragazzi" e "bambini";
- normalizzazione e *re-scaling* (trattate nel paragrafo successivo);
- creazione di nuovi attributi, ad esempio quando adottiamo un GLM stiamo aumentando il numero di feature. Ciò è utile a patto che consenta una separazione più netta delle classi, se ciò non accade introduciamo una complicazione superfla al problema.

Normalizzazione e re-scaling La normalizzazione consiste nel restringere l'intervallo di variabilità di un attributo preservando le distanze relative tra i suoi valori. Il *re-scaling* è un semplice cambio di scala, cioè divisione o moltiplicazione per uno stesso valore.

Queste operazioni si rendono necessarie perché ciascun attributo può variare in un intervallo più o meno ampio rispetto agli altri (ad esempio l'altezza ed il peso espressi in cm e kg hanno intervalli di variabilità molto diversi). Molti algoritmi di classificazione non riescono ad operare su dati

così fatti, ad esempio la funzione logistica satura ad 1 quando l'ingresso è troppo grande. Anche lì dove questo problema non si presenta, come nella regressione lineare, si vede che l'algoritmo di discesa a gradiente converge più velocemente quando i dati sono normalizzati[10]. Ribadiamo ulteriormente che ciascun attributo viene normalizzato separatamente dagli altri e lo scopo della normalizzazione è far sì che tutti gli attributi varino in uno stesso intervallo. Una volta normalizzati gli ingressi, occorre memorizzare i parametri usati per la normalizzazione, in maniera tale che questa possa essere riapplicata invariata agli ingressi incontrati in fase di predizione. Le tecniche principali sono:

- *min-max normalization*, si scalano i valori dell'attributo in funzione del minimo e del massimo per farli rientrare in un intervallo [a,b] desiderato (tipicamente [0,1]).

$$x_i \leftarrow \frac{x_i - \min_i(x_i)}{\max_i(x_i) - \min_i(x_i)} \cdot (b - a) + a$$

  Svantaggi:

  1. gli *outlier* influiscono molto su min e max, e quindi sulla normalizzazione;
  2. in fase di predizione potrebbe arrivare un ingresso al di fuori dell'intervallo $[\min, \max]$ per cui la normalizzazione sarebbe errata (in risultato sarebbe più piccolo di a o più grande di b).

- *z-score normalization*, normalizza i valori nell'intervallo [-1,1] e si basa su valor medio e deviazione standard, quindi non subisce l'influenza degli *outlier*.

$$x_i \leftarrow \frac{x_i - \overline{x}}{\sigma_x}$$

4.2 Feature selection

Possono capitare spesso due scenari:

- nel *dataset* sono presenti attributi superflui. Ad esempio in una base di dati che contiene ⟨ID, altezza, peso, sesso⟩, l'ID è superfluo ai fini della classificazione;

[10] Coursera, Machine Learning (Andrew Ng), Lezione 4.3

- alcuni attributi sono linearmente dipendenti e quindi mantenerne uno solo è sufficiente, dato che tutti gli altri non aggiungi informazione. Ad esempio se il *dataset* prevede ⟨sesso, peso (kg), peso (lb)⟩ è evidente che uno dei due pesi è eliminabile.

Ridurre le dimensioni dei vettori delle *feature* consente di semplificare il problema e ciò si traduce in tempi di addestramento più brevi e modelli meno complessi (di grado inferiore, riducendo dunque il rischio di *overfitting*).

Le tecniche adottate per individuare gli attributi superflui sono:

1. Filtri: grazie ai quali è possibile stabilire un *ranking* degli attributi in cui gli ultimi sono quelli che discriminano meno la classe di appartenenza dell'esempio (misure tipiche: *information gain*, *entropy*, *mutual information*);

2. *Wrappers:* si suddivide il *set* di dati di addestramento in due parti. La prima parte viene utilizzata per addestrare il modello, selezionando però solo un sottoinsieme di attributi per volta. Per ciascuno di questi sottoinsiemi si valutano le prestazioni sulla seconda parte (*validation set*) per capire quali attributi hanno influenza maggiore;

3. Algoritmi di riduzione della dimensionalità (PCA, trattata in seguito).

4.3 Scelta del modello

4.3.1 Overfitting ed underfitting

Nella scelta del modello è importante evitare due situazioni:

- *overfitting*: si verifica quando il modello si comporta molto bene sul *training set*, ma non è in grado di prevedere correttamente le classi dei campioni nel *validation set*. Significa che il modello non è in grado di generalizzare ed è quindi troppo complesso (ad esempio un polinomio di grado troppo elevato). Un modello troppo complesso oscillerà molto e sarà caratterizzato da una forte varianza.

- *underfitting*: il modello è troppo semplice e genera errori troppo grandi sia sui dati di addestramento che su quelli di validazione. Ciò si traduce in un errore medio di predizione elevato (*bias*);

4.3.2 Trade-off tra bias e varianza

Cerchiamo di capire qual è l'influenza di *bias* e varianza sull'errore mediamente commesso da un modello[11].

Formulazione del problema Ipotizziamo di avere a disposizione infiniti *dataset* D_i la cui unione costutuisce l'intera popolazione di campioni che il nostro modello potrà mai incontrare nella sua esistenza.

Ipotizziamo di fissare il tipo di modello da usare e di addestrarlo ogni volta con un diverso D_i, otterremo quindi infinite ipotesi $h^{(D_i)}(x)$.

Ciascun campione del *dataset* è nella forma $< \mathbf{x^{(i)}}, y^{(i)} >$ dove $y^{(i)}$ è somma di un valore vero (a noi sconosciuto) e di un errore gaussiano a media nulla e varianza σ_e:

$$y^{(i)} = f(\mathbf{x^{(i)}}) + e^{(i)}$$

Ciascuna delle ipotesi $h^{(D_i)}$ commetterà un errore nel predire i campioni *rispetto al loro valore vero*, e lo possiamo esprimere sotto forma di errore quadratico medio (MSE):

$$MSE\left(h^{(D_i)}(x)\right) = E_x\left[(h^{(D_i)}(x) - f(x))^2\right]$$

A noi interessa valutare l'aspettazione (valore atteso) di quest'ultima quantità rispetto ai D_i. In pratica è come se misurassimo l'errore quadratico medio commesso da ciascun singolo modello su tutti i possibili ingressi e mediassimo a loro volta questi risultati; ovviamente il calcolo di questo valor medio richiederebbe la conoscenza di tutti gli infiniti D_i, per cui ci accontentiamo del valore atteso (che chiameremo *Generalization Error* o *GER*):

$$\begin{aligned} GER &= E_D[MSE] \\ &= E_D\left[E_x\left[\left(h^{(D_i)}(x) - f(x)\right)^2\right]\right] \\ &= E_x\left[E_D\left[\left(h^{(D)}(x^{(i)}) - f(x^{(i)})\right)^2\right]\right] \end{aligned}$$

L'ultimo passaggio è possibile perché l'operatore E è lineare (si tratta di una sommatoria nel caso discreto). Essendo tutti gli addendi di E_x non negativi, per ottenere un *GER* piccolo non ci resta che minimizzare E_D. La presenza degli apici (i) nella formula più interna, ci ricorda che l'aspettazione

[11] Questa parte è stata scritta partendo dagli appunti ed integrandoli con il libro e con quanto letto qui.

E_D che desideriamo conoscere è calcolata fissato un singolo campione $x^{(i)}$ e facendo variare il modello usato.

Decomposizione Studiamo ora come è fatto il seguente termine[12]:

$$E_D\left[\left(h^{(D)}(x^{(i)}) - f(x^{(i)})\right)^2\right]. \tag{18}$$

Introduciamo la grandezza:

$$\overline{h}(x^{(i)}) = E_D\left[h^{(D)}(x^{(i)})\right]$$

che rappresenta la media statistica dei valori restituiti in uscita da ciascun modello $h^{(D_i)}$ quando tutti ricevono in ingresso uno stesso $x^{(i)}$. In altre parole fissiamo $x^{(i)}$, lo diamo in pasto a tutti i modelli addestrati e calcoliamo la media delle uscite. Questa quantità rappresenterà la miglior stima di $f(x^{(i)})$.

A questo punto sommiamo e sottraiamo $\overline{h}(x^{(i)})$ nell'Equazione 18:

$$E_D\left[\left(h^{(D)}(x^{(i)}) - \overline{h}(x^{(i)}) + \overline{h}(x^{(i)}) - f(x^{(i)})\right)^2\right]$$

sviluppiamo il quadrato raggruppando gli addendi 2 per volta e sfruttiamo la linearità di E:

$$\begin{aligned} &E_D\left[\left(h^{(D)}(x^{(i)}) - \overline{h}(x^{(i)})\right)^2\right] + E_D\left[\left(\overline{h}(x^{(i)}) - f(x^{(i)})\right)^2\right] \\ &+ E_D\left[2\left(h^{(D)}(x^{(i)}) - \overline{h}(x^{(i)})\right)\left(\overline{h}(x^{(i)}) - f(x^{(i)})\right)\right] \end{aligned}$$

Analizziamo ora i singoli addendi. Il primo rappresenta, per definizione, la varianza:

$$E_D\left[\left(h^{(D)}(x^{(i)}) - \overline{h}(x^{(i)})\right)^2\right] = Var\left(h^{(D)}(x^{(i)})\right).$$

Nel secondo termine entrambi gli addendi non dipendono più da D, fissato un $x^{(i)}$ rappresentano dei numeri, per cui:

$$E_D\left[\left(\overline{h}(x^{(i)}) - f(x^{(i)})\right)^2\right] = (\overline{h}(x^{(i)}) - f(x^{(i)}))^2$$

[12]La decomposizione del MSE in *bias* e varianza è valida in generale. Qui riporto la dimostrazione usando la notazione del contesto in analisi.

e questa quantità rappresenta per definizione il $Bias^2\left(h^{(D)}(x^{(i)})\right)$. Lo stesso discorso può essere fatto per il terzo fattore del terzo addendo:

$$\begin{aligned}
&2\left(\overline{h}(x^{(i)}) - f(x^{(i)})\right) E_D\left[h^{(D)}(x^{(i)}) - \overline{h}(x^{(i)})\right] \\
&= 2\left(\overline{h}(x^{(i)}) - f(x^{(i)})\right)\left(E_D\left[h^{(D)}(x^{(i)})\right] - E_D\left[\overline{h}(x^{(i)})\right]\right) \\
&= 2\left(\overline{h}(x^{(i)}) - f(x^{(i)})\right)\left(\overline{h}(x^{(i)}) - \overline{h}(x^{(i)})\right) \\
&= 0
\end{aligned}$$

In definitiva:

$$MSE(h^{(D)}(x^{(i)})) = Bias^2\left(h^{(D)}(x^{(i)})\right) + Var\left(h^{(D)}(x^{(i)})\right).$$

Commento dei risultati L'errore di generalizzazione di un modello è l'aspettazione del MSE appena definito, abbiamo quindi interesse a rendere l'MSE piccolo. Ciò può essere ottenuto riducendo il *bias* o la varianza:

- il *bias* tiene conto di quanto la migliore stima (cioè la media delle stime) sia in grado di avvicinarsi al valore vero. Un *bias* alto è indice di un modello troppo semplice che, a prescindere dal *dataset* D_i di addestramento, non ha una buona capacità predittiva. Questo è riconducibile, ad esempio, un polinomio di grado troppo basso;

- la varianza tiene conto di quanto ciascuna ipotesi si discosta dal risultato dell'ipotesi media. Una varianza alta è indice del fatto che, addestrando uno stesso modello con D_i diversi, si possono ottenere ipotesi molto diverse. Ciò significa che il modello è troppo complesso, tende ad andare in *overfitting* e manca di capacità di generalizzazione.

L'ideale sarebbe avere sia *bias* che varianza piccoli, ciò garantisce che il modello approssimi bene i risultati reali pur rimanendo stabile, cioè non subisce fortemente l'influenza del particolare *training set* utilizzato (a patto che questo sia ben costruito). In Figura 3 è mostrato il tipico andamento di *bias* e varianza in funzione della complessità del modello. Notiamo che le due componenti hanno un andamento opposto (al diminuire dell'una, aumenta l'altra), il miglior compromesso è rappresentato dal punto in cui la loro somma è minima.

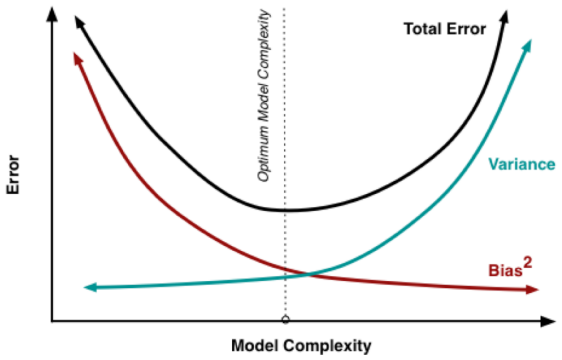


Figura 3: *Bias* e varianza in funzione della complessità del modello.

4.3.3 Cross-validation

Non sapendo quale modello scegliere, l'idea della *cross-validation* è quella di addestrarne alcuni su un sottoinsieme del *training set* a disposizione e poi valutarne le prestazioni sulla restante parte (*validation set*).

La scelta del modello finale va fatta sulla base delle misure di prestazione calcolate sul *validation set* e non sul *training set*. Questo ha senso perché il modello è stato addestrato per minimizzare l'errore quadratico sul *training set* ed in virtù di ciò ci aspettiamo che l'errore commesso su di esso sia basso. In particolare, si dimostra che l'errore commesso su *training set* è una sottostima dell'errore di generalizzazione, mentre quello sul *validation set* lo approssima in maniera migliore.

Dopo aver scelto quale modello usare, questo può essere riaddestrato utilizzando tutto il *training set* iniziale.

Questo stesso approccio può essere sfruttato per determinare metaparametri (ad esempio il coefficiente di penalizzazione del fattore di regolarizzazione) oppure per capire quali siano le *feature* meno significative.

Due tecniche comunemente usate per effettuare *cross-validation* prendono il nome di *holdout* e *k-folds*

4.3.4 Holdout cross validation

La *holdout cross validation* prevede che il *training set* su cui addestrare i modelli in prova sia costituito dai due terzi di quello iniziale, il restante terzo costituirà il *validation set*. Occorre prestare attenzione affinché i campioni scelti per l'addestramento e per la valutazione siano rappresentativi di tutte le classi presenti. Se ad esempio la classe 0 è completamente assente tra

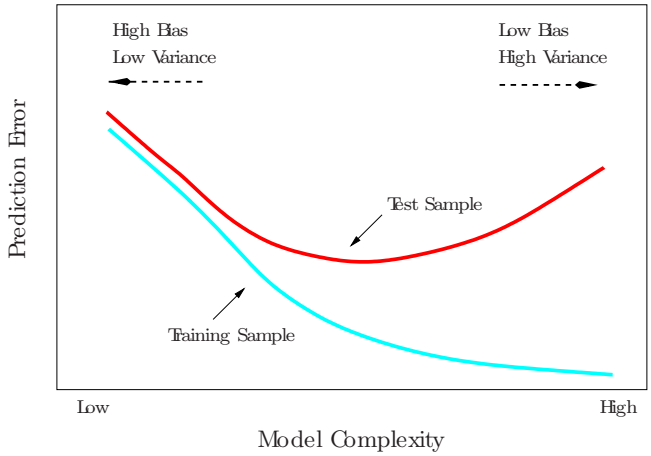


Figura 4: Errore di predizione su *trainig set* e *validation set* al variare della complessità del modello

i dati di addestramento, e presente in quelli di validazione, il modello mostrerà una pessima capacità di riconoscere tali campioni e ciò si ripercuoterà negativamente sulle misure di prestazione.

4.3.5 K-folds cross validation

Questa tecnica prevede che l'insieme dei dati venga suddiviso in k sottoinsiemi di uguali dimensioni (tipicamente $k = 10$). Durante ciascuna delle k iterazioni dell'algoritmo ciascun modello viene addestrato sull'unione di $k-1$ sottoinsiemi e validato sul $k-esimo$. Ad ogni iterazione si cambiano i $k-1$ sottoinsiemi di addestramento e di conseguenza il $k-esimo$. L'errore commesso da ciascun modello viene calcolato come media degli errori da esso commessi in seguito a ciascuno dei k addestramenti.

4.4 Valutazione del modello

La valutazione finale del modello viene fatta dopo averlo scelto, e quindi dopo averlo riaddestrato usando l'unione di *training set* e *validation set*. È evidente, quindi, che il *test set* su cui valutiamo le prestazioni del modello debba essere distinto dagli altri due.

Occorre, inoltre, evitare il *peeking*, cioè non usare i dati del *test set* come se fossero parte del *validation set*. In questa maniera il *test set* viene usato per guidare la scelta del modello (è grave quasi come usare il *test set* per l'addestramento), mentre per ottenere misure di prestazione credibili il

test set non deve avere alcuna influenza su come il modello viene scelto ed addestrato.

4.4.1 Matrice di confusione e metriche

La matrice di confusione consente di rappresentare in maniera compatta alcune grandezze di interesse per il calcolo delle misure di prestazione di un classificatore binario. Supponendo che l'uscita del classificatore sia *yes* ($y = 1$) o *no* ($y = 0$), la matrice di confusione viene costruita indicando sulle righe la classe reale dei campioni del *test set*, sulle colonne la classe predetta dal classificatore per tali campioni (Figura 5). All'interno della matrice identifichiamo 4 quantità:

- TP (*true positive*), rappresenta il numero di veri positivi, cioè il numero di campioni classificati come positivi (classe 1) e che effettivamente sono positivi;
- TN (*true negative*), rappresenta il numero di veri negativi, cioè il numero di campioni classificati come negativi (classe 0) e che effettivamente sono negativi;
- FP (*false positive*), rappresenta il numero di falsi positivi, cioè il numero di campioni classificati come positivi (classe 1), ma che in realtà sono negativi;
- FN (*false negative*), rappresenta il numero di falsi negativi, cioè il numero di campioni classificati come negativi (classe 0), ma che in realtà sono positivi.

Osserviamo che:

- TP+TN+FP+FN è il numero totale di campioni nel *test set*;
- TP+FN è il numero di istanze positive nel *test set*;
- TN+FP è il numero di istanze negative nel *test set*;
- TP+FP è il numero di predizioni positive a partire dal *test set*;
- TN+FN è il numero di predizioni negative a partire dal *test set*.

Partendo da queste 4 quantità si definiscono alcune metriche di uso comune:

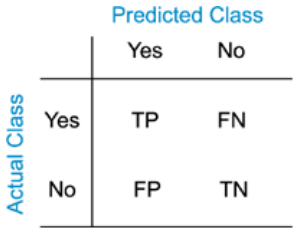


Figura 5: Matrice di confusione

- *Accuracy*, rappresenta il tasso di predizioni corrette:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

- *Error rate*, rappresenta il tassso di predizioni errate (complemento ad 1 dell'*accuracy*):

$$Error\ Rate = 1 - Accuracy = \frac{FP + FN}{TP + TN + FP + FN}$$

Le due metriche precedenti tengono conto delle prestazioni complessive del classificatore. Non ci consentono di capire quanto sia buono nel discriminare casi positivi e casi negativi. A tal proposito vengono adottate le seguenti misure:

- *Recall*, detta anche *Sensitivity* o *True Positive Rate*, rappresenta la percentuale di casi positivi correttamente calssificati:

$$Recall = \frac{TP}{TP + FN}$$

- *Specificity*, detta anche *True Negative Rate*, in maniera duale alla precedente rappresenta la percentuale di casi negativi correttamente calssificati:

$$Specificity = \frac{TN}{TN + FP}$$

- *False Positive Rate*, rappresenta il complemento ad uno della *Specificity*, cioè la percentuale di casi negativi erroneamente classificati:

$$False\ Positive\ Rate = 1 - Specificity = \frac{FP}{TN + FP}$$

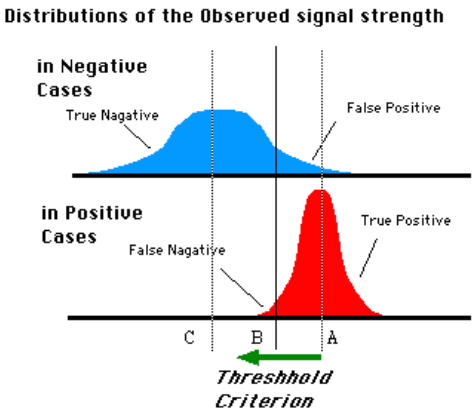


Figura 6: Distribuzione delle uscite associate ai campioni positivi e negativi nel *test set*

In ultimo può essere interessante valutare la *Precision*, cioè il tasso di predizioni vere che si rivelano corrette, ovvero il rapporto tra i veri positivi e tutti i positivi:

$$Precision = \frac{TP}{TP + FP}.$$

Tipicamente vorremmo un classificatore con *Recall* e *Precision* alte, cioè un classificatore con un alto numero di veri positivi e pochi falsi negativi e falsi positivi. Per unire queste due esigenze in un'unica metrica viene introdotta la *F-measure*, calcolata come media armonica di *Precision* e *Recall*:

$$F\text{-}measure = 2\frac{Precision \cdot Recall}{Precision + Recall}.$$

4.4.2 Curve ROC

Vediamo ora come è possibile usare le informazioni provenienti dalle metriche appena introdotte per effettuare il *tuning* di alcuni parametri del modello.

Ipotizziamo di lavorare con una regressione logistica. Come abbiamo visto in precedenza, l'uscita della funzione ipotesi è numerica ed è compresa tra 0 ed 1. La classe di appartenenza di ciascun campione viene calcolata in base al fatto che tale uscita superi o meno un valore di soglia (*threshold*). In precedenza, però, non abbiamo discusso su come debba essere scelta questa soglia.

Supponiamo che le distribuzioni delle uscite che il classificatore associa ai campioni di classe 1 e 0 del *test set* siano gaussiane, rappresentate rispettivamente in azzurro e rosso in Figura 6.

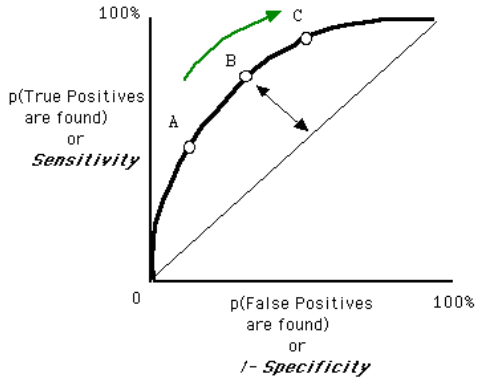


Figura 7: Curva ROC

Se B rappresenta il valore di soglia scelto, è facile vedere che:

- i campioni negativi a sinistra di B rappresentano i veri negativi;
- i campioni negativi a destra di B rappresentano i falsi positivi;
- i campioni positivi a sinistra di B rappresentano i falsi negativi;
- i campioni positivi a destra di B rappresentano i veri positivi.

Quindi la scelta della soglia influisce direttamente sulla matrice di confusione, e quindi sulle metriche analizzate.

Tramite le curve ROC siamo in grado di vedere esattamente come il valore della soglia influisce sul *True Positive Rate* e sul *False Positive Rate*. Per tracciare una curva ROC facciamo variare la soglia in un intervallo di possibili valori, per ciascun valore calcoliamo *True Positive Rate* e *False Positive Rate* e disegniamo il punto individuato da questa coppia di valori su un sistema di assi cartesiani. La curva risultante rappresenterà l'andamento di *TPR* e *FPR* al variare della soglia (Figura 7). A seconda di quali sono le esigenze del caso specifico, sceglieremo il valore di soglia più opportuno.

Ad esempio, ipotizziamo che il sistema che stiamo addestrando consenta di classificare un paziente come sano o malato. Ovviamente vorremmo che esso fosse in grado di rivelare il 100% dei malati, ovvero ottenere un *TPR* pari ad 1. Paradossalmente potremmo ottenere ciò impostando una soglia nulla in maniera tale che tutti i pazienti vengano riconosciuti come malati, a prescindere dall'effettivo stato di salute. Da un punto di vista medico questo implicherebbe ulteriori esami sui pazienti per verificare l'effettiva presenza della malattia, e quindi un costo superiore da sostenere (per il

paziente o per la Sanità Pubblica). Quindi ciò che vorremmo fare è anche minimizzare il *FPR*, ed ecco che la curva ROC ci consente di individuare il giusto compromesso tra un *TPR* alto ed *FPR* basso. Nello stesso scenario, però, potremmo ipotizzare che la malattia sia grave e quindi potrebbe essere preferibile qualche falso positivo in più pur di non abbassare il tasso di veri positivi, evitando quindi che alcuni pazienti malati non vengano riconosciuti come tali. Anche in questo caso l'analisi della curva ROC può farci capire quale sia il miglior valore della soglia.

Le curve ROC possono essere usate anche per determinare i valori ottimali di altri metaparametri del modello, o per fare un confronto tra modelli diversi. In particolare possiamo tracciare la curva ROC di uno stesso modello, addestrato sullo stesso *training set*, modificando di volta in volta il valore un metaparametro. Dopo ciascun addestramento possiamo tracciare la curva ROC e selezionare il modello la cui curva è migliore, cioè quella che sottende l'area maggiore, avvicinandosi maggiormente all'andamento ideale costante di una una retta passante per TPR=1.

5 Algoritmi di classificazione non lineari

5.1 Reti neurali artificiali

Vedi AIMA, Volume 2, paragrafo 20.5.

5.2 Support Vector Machine

Lo scopo di questi modelli è la classificazione binaria. Analizzeremo inizialmente il caso in cui i dati da classificare siano linearmente separabili, in seguito estenderemo il discorso a dati non linearmente separabili[13].

5.2.1 Dati linearmente separabili

Il modello che costruiamo è ancora una volta un iperpiano (come la regressione lineare ed il *perceptron*), ma cambierà il modo in cui lo addestreremo, ovvero cambierà lo scopo dell'addestramento. L'espressione dell'ipotesi è quindi:

$$h(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b.$$

Porre $h(\mathbf{x}) = 0$ ci consente di definire l'equazione dell'iperpiano separatore:

$$\mathbf{w}^T\mathbf{x} + b = 0.$$

A questo punto, tutti i punti che si trovano al di sopra dell'iperpiano verrano classificati come $y = +1$, quelli al di sotto del piano come $y = -1$. Fare ciò è equivalente a porre l'uscita della funzione ipotesi in ingresso alla funzione segno, ovvero:

$$y(\mathbf{x}^{(i)}) = \text{sign}(h(\mathbf{x}^{(i)})) = \text{sign}(\mathbf{w}^T\mathbf{x}^{(i)} + b).$$

Possiamo esplicitare l'espressione precedente come segue:

$$\begin{aligned} \mathbf{w}^T\mathbf{x}^{(i)} + b > 0 \quad &\rightarrow \quad y^{(i)} = +1 \\ \mathbf{w}^T\mathbf{x}^{(i)} + b < 0 \quad &\rightarrow \quad y^{(i)} = -1 \end{aligned}$$

A questo punto vediamo come determinare il vettore dei parametri $\mathbf{w}$ e b, cioè come determinare l'iperpiano separatore. Lo scopo della procedura di ottimizzazione che vogliamo costruire è massimizzare il *margine*. Il margine si ottiene sommando un contributo d^+ ed un contributo d^-. d^+ rappresenta

[13] Qualcosa su questo argomento si trova in AIMA, Volume 2, paragrafo 20.6. Per gli amanti del brivido, questa dispensa sembra decisamente più completa.

la minima distanza tra un punto di classe $+1$ e l'iperpiano separatore. d^- èil corrispettivo per la classe -1, cioè la distanza minore tra un punto di classe -1 e l'iperpiano. I punti che danno origine a tale distanza, cioè i più vicini all'iperpiano, costituiscono i due *vettori di supporto*.

Prima di formalizzare il problema di ottimizzazione, imponiamo due vincoli che ci torneranno utili in seguito: per tutti i punti appartenenti ai vettori di supporto l'ipotesi deve valere esattamente $+1$ o -1 (a seconda che il punto appartenga al vettore di supporto positivo o negativo):

$$\begin{aligned} \mathbf{w}^T\mathbf{x}^{(i)} + b = +1 \quad & \text{se } \mathbf{x}^{(i)} \in s.v.^+ \\ \mathbf{w}^T\mathbf{x}^{(i)} + b = -1 \quad & \text{se } \mathbf{x}^{(i)} \in s.v.^- \end{aligned}$$

A questo punto possiamo sintetizzare i 4 vincoli imposti nell'unica espressione:

$$y^{(i)}\left(\mathbf{w}^T\mathbf{x}^{(i)} + b\right) - 1 \geq 0$$

vediamo perché questa ha senso:

- se $\mathbf{x}^{(i)}$ è di classe $+1$, risulta $y^{(i)} = +1$ e quindi il vincolo precedente impone che $\mathbf{w}^T\mathbf{x}^{(i)} + b \geq +1$;
- se $\mathbf{x}^{(i)}$ è di classe -1, risulta $y^{(i)} = -1$ e quindi il vincolo precedente impone che $\mathbf{w}^T\mathbf{x}^{(i)} + b \leq -1$.

Torniamo al problema di massimizzazione: è noto[14] che la distanza tra un generico $\mathbf{x}^{(i)}$ ed un iperpiano è pari a[15]:

$$\frac{|(\mathbf{w}^T\mathbf{x}^{(i)} + b)|}{||\mathbf{w}||}$$

dove ricordiamo che:

- $||\mathbf{w}|| = \sqrt{\sum_{i=0}^{n} w_i^2}$

In base alle condizioni imposte, i punti dei vettori di supporto rendono unitario il numeratore della precedente, per cui il margine $(d^+ + d^-)$ è proprio $\frac{2}{||\mathbf{w}||}$.

[14] Point-Plane Distance

[15] Nelle slide del corso il numeratore è $|(\mathbf{w}^T\mathbf{x}^{(i)} + b) \cdot y^{(i)}|$. Probabilmente questo è errato e l'introduzione di $y^{(i)}$ serve ad eliminare il valore assoluto. Infatti, per come abbiamo impostato il problema, risulta che $|(\mathbf{w}^T\mathbf{x}^{(i)} + b)| = (\mathbf{w}^T\mathbf{x}^{(i)} + b) \cdot y^{(i)}$ (cioè $|x| = x \cdot \text{sign}(x)$).

Il problema desiderato è quindi il seguente: vogliamo trovare $\mathbf{w}$ e b che massimizzino $\frac{2}{||\mathbf{w}||}$ sotto i seguenti vincoli:

$$y^{(i)}\left(\mathbf{w}^T\mathbf{x}^{(i)}+b\right)-1\geq 0 \qquad i=1,\ldots,m.$$

Si vede che:

$$\underset{\mathbf{w},b}{\operatorname{argmax}}\frac{2}{||\mathbf{w}||}=\underset{\mathbf{w},b}{\operatorname{argmin}}\frac{||\mathbf{w}||^2}{2}.$$

Ciò ci consente di passare da una massimizzazione ad un problema di minimizzazione convesso e con vincoli di disuguaglianza lineari. In realtà a partire da quest'ultima formulazione (detta *primale*) si genera il problema *duale* (facendo utilizzo della funzione di Lagrange) che è di più semplice risoluzione. La funzione obiettivo diventa quindi:

$$L(\mathbf{w},b,\boldsymbol{\alpha})=\frac{1}{2}||\mathbf{w}||^2-\sum_{i=1}^{m}\alpha_i(y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)}+b)-1).$$

Tramite una serie di passaggi ed osservazioni è possibile riformulare il problema come segue[16]:

$$\begin{aligned}\max_{\boldsymbol{\alpha}}\sum_{i=1}^{m}\alpha_i-\frac{1}{2}\sum_{i,j=1}^{m}y^{(i)}y^{(j)}\alpha_i\alpha_j(\mathbf{x^{(i)}}\cdot\mathbf{x^{(j)}})\\ s.t. \qquad \alpha_i\geq 0 \quad i=1,\ldots,m\\ \sum_{i=1}^{m}\alpha_i y^{(i)}=0\end{aligned}$$

dove α_i sono i coefficienti di Lagrange introdotti dall'omonima funzione ed a partire dai loro valori ottimi è possibile ricavare $\mathbf{w}^*$ e b^*. Si vede che α_i è non nullo per tutti e soli i punti appartenenti ai vettori di supporto.

Una volta individuati il vettore di coefficienti ottimi $\boldsymbol{\alpha}^*$ è possibile passare alla soluzione del problema primario tramite due relazioni:

- la prima consente di trovare i coefficienti ottimi dell'iperpiano ($\mathbf{w}^*$)

$$w^*=\sum_{i=1}^{m}\alpha_i^* y^{(i)}x^{(i)}; \tag{19}$$

[16] C'è una differenza tra il libro e le slide, qui ho riportato la versione del libro di pag. 440 del secondo volume (paragrafo 20.6).

Inoltre credo che per $\mathbf{x^{(i)}}\cdot\mathbf{x^{(j)}}$ gli autori intendano il prodotto scalare tra due vettori di ingressi. Se così fosse, per correttezza formale, il primo dovrebbe essere trasposto.

- il secondo *set* di equazioni consente di trovare il valore ottimo dell'intercetta (b^*)

$$\alpha_i^* - (y^{(i)}(\mathbf{w}^{*T}\mathbf{x}^{(i)} + b^*) - 1) = 0 \quad i = 1, \ldots, m$$

In particolare, soluzione trovata gode di alcune proprietà:

- $\alpha_i^* = 0$ per tutti i punti che non giacciono sui margini;
- $\alpha_i^* \neq 0$ per tutti i punti che giacciono sui margini, cioè per quei punti facenti parte dei vettori di supporto.

Date le due precedenti conclusioni e l'Equazione 19, notiamo che *la soluzione ottima è una combinazione lineare dei soli vettori di supporto*. Quindi, l'equazione del piano separatore ottimo è:

$$\mathbf{w}^{*T}\mathbf{x} + b^* = \sum_{i \in \{s.v.^+ \cup s.v.^-\}} \alpha_i^* y^{(i)}(\mathbf{x}^{(i)T} \cdot \mathbf{x}) + b^*$$

5.2.2 Soft margin SVM

Esaminiamo ora come usare una SVM nel caso in cui i dati *non* siano linearmente separabili. Notiamo che nella formulazione originale del problema ciascun vincolo

$$y^{(i)}\left(\mathbf{w}^T\mathbf{x}^{(i)} + b\right) - 1 \geq 0 \qquad i = 1, \ldots, m$$

impone che il punto *i-esimo* non oltrepassi il semispazio delimitato dall'iperpiano in cui risiedono tutti i campioni della sua stessa classe. In altre parole, se un generico $\mathbf{x}^{(i)}$ è di classe y il problema di ottimizzazione ammette soluzione solo se esiste una soluzione in cui tutti i punti si trovano nel semipiano "corretto". In caso di *outlier* questa procedura non può funzionare. Ecco dunque che per contemplare la possibilità di "far commettere errori" alla procedura di ottimizzazione introduciamo per ciascun vincolo una variabile *slack* $\xi_i \geq 0$. Il valore di ogni ξ_i è proporzionale a quanto $\mathbf{x}^{(i)}$ sta "sforando" nel semipiano errato, cioè a quanto il punto anomalo si discosta dall'iperpiano e se è maggiore di 1 indica una misclassificazione. I vincoli diventano quindi:

$$y^{(i)}\left(\mathbf{w}^T\mathbf{x}^{(i)} + b\right) - 1 + \xi_i \geq 0 \qquad i = 1, \ldots, m.$$

Ovviamente è necessario penalizzare i punti misclassificati, introducendo un fattore additivo nella funzione costo:

$$\min_w \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{m} \xi_i$$

dove C rappresenta un parametro di *regolarizzazione* il cui valore può essere determinato tramite *cross validation*. Infatti una C troppo grande rischia di produrre un addestramento troppo "severo" e non trovare una soluzione, il contrario produce un addestramento troppo tollerante e dunque un modello che genera troppi errori di classificazione. Il problema duale, quello effettivamente risolto, diventa:

$$\begin{aligned} \max_{\alpha} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j (\mathbf{x^{(i)}} \cdot \mathbf{x^{(j)}}) \\ s.t. \qquad 0 \leq \alpha_i \leq C \quad i = 1, \ldots, m \\ \sum_{i=1}^{m} \alpha_i y^{(i)} = 0. \end{aligned}$$

5.2.3 Metodi kernel

Quanto appena visto non risolve tutti i problemi, infatti affinché il metodo funzioni è necessario che i punti siano distributi in regioni di spazio "quasi" linearmente separabili, cioè ammettiamo la presenza di *outlier* ma non di non-linearità. L'idea alla base dei metodi *kernel* è simile a quella delle funzioni di base (ϕ), le quali applicano una funzione a ciascuna istanza di ingresso che ne aumenta il numero di *feature* e renda linearmente separabili i campioni in uno spazio degli *input* a dimensionalità superiore.

Il problema in tutto ciò sorge quando le non-linearità sono molto forti e quando i vettori delle *feature* sono molto grandi già prima dell'applicazione delle funzioni di base. Nell'espressione da ottimizzare compare infatti il prodotto $\mathbf{x}^{(i)}\mathbf{x}^{(j)}$:

$$\max_{\alpha} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j (\mathbf{x}^{(i)}\mathbf{x}^{(j)}).$$

Dopo aver applicato le funzioni base, questo prodotto viene sostituito da $\phi(\mathbf{x}^{(i)})^T \phi(\mathbf{x}^{(j)})$ e questo calcolo può diventare oneroso nelle condizioni sopra descritte. Lo scopo di una funzione kernel $k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$[17] è quello di restituire lo stesso risultato del precedente prodotto tra vettori, senza però richiedere una trasformazione esplicita degli ingressi, cioè:

$$k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \phi(\mathbf{x}^{(i)})^T \phi(\mathbf{x}^{(j)}).$$

[17] Vedi le slide per alcuni esempi di funzioni kernel.

Per rappresentare gli ingressi sto continuando ad usare la notazione introdotta fin dall'inizio, pur essendo leggermente diversa da quella delle slide e del libro.

Applicando una funzione simile, il problema di ottimizzazione diventa:

$$\max_{\alpha} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}).$$
$$s.t. \qquad 0 \leq \alpha_i \leq C \quad i = 1, \ldots, m$$
$$\sum_{i=1}^{m} \alpha_i y^{(i)} = 0.$$

A questo punto la procedura di ottimizzazione richiede la scelta di:

- tipo di Kernel da usare;
- parametri del Kernel scelto;
- valore di C;

questi possono essere tutti determinati euristicamente tramite *cross-validation*. Il vantaggio di pochi parametri da calibrare si aggiunge ai vantaggi delle SVM, cioè:

- la funzione da minimizzare è quadratica, e non c'è il pericolo di incappare in minimi locali;
- la soluzione può essere trovata in tempo polinomiale tramite le note tecniche di ottimizzazione;
- le soluzioni sono stabili, cioè non dipendono da inizializzazioni casuali, come nel caso delle reti neurali;
- la soluzione è sparsa, in quanto dipende solo dai vettori di supporto.

5.3 K-nearest neighbors

Vedi AIMA, Volume 2, paragrafo 20.4.

6 Apprendimento statistico

In questa parte analizzeremo un esempio di classificatore basato su apprendimento statistico: il classificatore *bayesiano naive*. Esso consente di predirre la classe di appartenenza di un campione sfruttando le regole della statistica, in particolar modo il suo funzionamento è incentrato sulla regola di *Bayes*[18].

6.1 Regola di Bayes

Dati due eventi A e B, la regola di Bayes ci consente di calcolare la cosidetta "probabilità a posteriori", cioè la probabilità che dato l'evento B si verifichi anche A. Essa è:

$$p(A|B) = \frac{p(A,B)}{P(B)}.$$

Per la regola del prodotto possiamo scrivere:

$$p(A,B) = p(A|B) \cdot p(B),$$

per simmetria

$$p(A,B) = p(B,A) = p(B|A) \cdot p(A).$$

Sostituendo l'ultima nella prima equazione, otteniamo la forma più usata della regola di Bayes:

$$p(A|B) = \frac{p(B|A) \cdot p(A)}{p(B)}.$$

Noi useremo questo risultato mettendo in relazione ingressi ed uscite di un classificatore, come segue:

$$p(y|x) = \frac{p(x|y) \cdot p(y)}{p(x)}$$

dove:

- $p(y|x)$ prende il nome di "stima a posteriori", ed infatti rappresenta la probabilità che si presenti un'uscita y avendo osservato un ingresso x;
- $p(x|y)$ si chiama "verosimiglianza" ed è la probabilità che venga generato un ingresso x sapendo che la classe d'appartenenza è y (vedi esempio successivo);

[18] AIMA, Volume 2, paragrafi 20.1 e 20.2 (il capitolo 13 è propedeutico).

- $p(y)$ è la probabilità marginale derivante dalla conoscenza del fenomeno (vedi esempio);
- $p(x)$ è l'"evidenza", cioè la probabilità marginale di x.

Chiariamo i concetti con un esempio. Ipotizziamo che un produttore di caramelle commercializzi 5 tipi di confezioni così assortite[19]:

- y_1: 100% caramelle alla ciliegia;
- y_2: 75% ciliegia + 25% lime;
- y_3: 50% + 50%
- y_4: 25% + 75%
- y_5: 100% lime.

Esteticamente, però, i sacchetti sono tutti uguali e non abbiamo modo di conoscere di che tipo è il nostro sacchetto. Ciò che vogliamo fare, ed è ciò a cui ambisce un classificatore bayesiano, è stimare con quale probabilità il nostro sacchetto è di uno dei 5 tipi man mano che scartiamo caramelle prelevate da esso. Così facendo possiamo ipotizzare di che tipo è il sacchetto ed in base a ciò ipotizzare il gusto della prossima caramella estratta. Per questo motivo annoveriamo i classificatori bayesiani tra i modelli "generativi". Inizialmente, dunque, vogliamo rispondere alla domanda: se estraggo M caramelle dal sacchetto, quante probabilità ci sono che il sacchetto sia di tipo y_j (j=1,...,5)? In formule:

$$p(y_j|\mathbf{x_M}) = \frac{p(\mathbf{x_M}|y_j) \cdot p(y_j)}{p(\mathbf{x_M})}$$

dove:

- $p(y_j|\mathbf{x_M})$ rappresenta la probabilità che desideriamo conoscere;
- $p(\mathbf{x_M}|y_j)$ rappresenta la probabilità che la sequenza di caramelle $\mathbf{x_M}$ (che possiamo interpretare come un vettore) sia estratta da un sacchetto di tipo y_j. Ad esempio la probabilità di estrarre *una* caramella alla ciliegia da un sacchetto di tipo y_1 è unitaria;
- $p(y_j)$ rappresenta la probabilità marginale che un sacchetto sia di tipo y_j. Ad esempio il fornitore potrebbe dirci d'aver prodotto il 10% di sacchetti y_1, il 20% y_2 e così via, per cui $p(y_1) = 0.1$, $p(y_2) = 0.2$, ecc.;

[19] Questo esempio è tratto dal libro (paragrafo 20.1).

- $p(\mathbf{x_M})$ rappresenta la probabilità marginale di pescare una sequenza $\mathbf{x_M}$ senza sapere nulla sul tipo di sacchetto. Ad esempio potremmo prendere in considerazione tutti i sacchetti venduti e misurare la probabilità media con cui ciascuna possibile sequenza di M caramelle viene estratta da un sacchetto. Tali sacchetti saranno assortiti nella maniera più varia, ma ai fini della probabilità marginale ciò non importa.

Tramite l'applicazione della regola di Bayes possiamo calcolare la probabilità che il sacchetto sia di ciascuna categoria man mano che estraiamo nuove caramelle. Ogni volta che estraiamo una nuova caramella abbiamo una nuova informazione a disposizione (è come se aggiungessimo un nuovo elemento ad $\mathbf{x_M}$) ed in base a questa nuova informazione possiamo aggiornare tutte le probabilità a posteriori, che presumibilmente diventeranno più accurate nel predirre la classe di appartenenza. Quando avremo estratto tutte le caramelle desiderate, affermeremo che la classe a cui appartiene il nostro sacchetto è quella la cui probabilità a posteriori è maggiore.

6.2 Classificatore bayesano naive

Supponiamo ora di lavorare con campioni costituiti da n ingressi discreti ed appartenenti a k possibili classi. Lo scopo del nostro simpatico classificatore è osservare ciascun campione in ingresso e determinare qual è la classe di appartenenza più probabile[20]:

$$y_{MAP} = \underset{j=1,\dots,k}{\operatorname{argmax}} \left(p(y_j|x_1, x_2, \dots, x_n)\right).$$

Questa operazione si chiama "stima MAP" (*Maximum A Posteriori probability*) poiché ogni volta che osserviamo un campione stimiamo qual è la sua classe di appartenenza più probabile (ad esempio, ogni volta che scartiamo un'ulteriore caramella dallo stesso sacchetto cerchiamo di capire di che tipo è il sacchetto). Il fatto che sia una stima *a posteriori* significa che prediciamo la classe di appartenenza del campione *dopo* averlo osservato.

Applichiamo la regola di Bayes e ci accorgiamo di un'interessante conseguenza:

$$\begin{aligned} y_{MAP} &= \underset{j=1,\dots,k}{\operatorname{argmax}} \left(p(y_j|x_1, x_2, \dots, x_n)\right) \\ &= \underset{j}{\operatorname{argmax}} \left(\frac{p(x_1, x_2, \dots, x_n|y_j) \cdot p(y_j)}{p(x_1, x_2, \dots, x_n)}\right) \\ &= \underset{j}{\operatorname{argmax}} \left(p(x_1, x_2, \dots, x_n|y_j) \cdot p(y_j)\right). \end{aligned}$$

[20] Si noti che è proprio ciò che facevamo cercando di "indovinare" il tipo di pacchetto di caramelle in nostro possesso.

Poiché al variare di y_j, $p(x_1, x_2, \ldots, x_n)$ resta immutato, esso rappresenta un fattore di normalizzazione non influente ai fini della massimizzazione. Eliminandolo semplifichiamo il problema.

Studiamo ora la cardinalità del fattore $p(x_1, x_2, \ldots, x_n|y_j)$. Nel caso più semplice ciascun attributo è binario e lo spazio degli ingressi è costituito da 2^n valori distinti. Per ciascun valore di y_j dobbiamo quindi conoscere $2^n - 1$ probabilità[21], per un totale di $k(2^n - 1)$ calcoli di probabilità necessari. Con problemi di grandi dimensioni questo potrebbe essere un limite, ecco dunque che con il classificatore *naive* introduciamo una notevole semplificazione: ipotizzamo che, fissata una classe, tutte le variabili dipendano solo da essa e non abbiano dipendenza reciproche ("indipendenza condizionata"[22]). In base a ciò riduciamo il problema al calcolo di sole kn probabilità e quindi:

$$y_{MAP} = \underset{j}{\operatorname{argmax}} \left(p(y_j) \prod_{i=1}^{n} p(x_i|y_j) \right). \tag{20}$$

In ultimo evidenziamo che, in genere, si passa alla massimizzazione della forma logaritmica della precedente equazione poiché è numericamente più stabile:

$$y_{MAP} = \underset{j}{\operatorname{argmax}} \left(\log p(y_j) + \sum_{i=1}^{n} \log p(x_i|y_j) \right). \tag{21}$$

6.2.1 Classificatore di testi basato su naive bayes

Analizziamo ora una semplice applicazione di un modello *naive bayes* per la classificazione di testi. Il *training set* a nostra dispozione è costituito da una serie di documenti per ciascuno dei quali è specificata la classe di appartenenza. Ad esempio questi documenti potrebbero essere pagine web di siti universitari e far riferimento ad un docente, ad un corso, ad un progetto o ad una facoltà. Vogliamo addestrare il classificatore in modo tale che, quando osserva una nuova pagina, ci sappia dire qual è la classe di appartenenza più probabile.

Pre-processing La prima operazione da fare consiste nell'eliminazione di articoli, preposizioni, ecc. dai testi. Infatti queste sono parole ricorrenti in qualsiasi lingua ed in quanto tali non apportano informazione utile ai fini della classificazione.

[21] la $2^n - esima$ può essere calcoalta come il complemento ad 1 della somma delle restanti.
[22] AIMA, Volume 2, paragrafo 13.1 e successivi.

Inoltre notiamo che, ai fini della classificazione, parole come "studiare", "studiai", "studiò" sono equivalenti perché fanno riferimento tutte al verbo "studiare". Per fare in modo che il classificatore le tratti alla stessa maniera, si effettua l'operazione di *stemming*, cioè di riduzione di ciascuna parola alla sua radice. In seguito allo *stemming* ciascuna delle tre parole precedenti viene *trasformata* in "stud".

Modellazione del problema Dopo aver preprocessato i testi occorre capire cosa utilizzare come ingressi e quali sono le uscite. La determinazione delle uscite è piuttosto immediata, infatti il nostro scopo è classificare ciascun documento e quindi ciascun possibile valore d'uscita dovrà essere associato una classe di appartenenza (con riferimento all'esempio delle pagine web queste potrebbero essere $y = 1$ per *docente*, $y = 2$ per *corso*, $y = 3$ per *progetto* e così via). Per quanto riguarda gli ingressi, possiamo pensare di avere tante *feature* quante sono le parole distinte presenti in tutti i testi del *training set*. Associamo ciascuna parola ad un elemento di un vettore. L'elemento *i-esimo* di un vettore in ingresso sarà pari al numero di volte che la parola ad esso associata è ripetuta nel documento a cui il vettore fa riferimento. La versione più semplice del classificatore ignora il numero di ripetizioni, e prevede che un elemento del vettore sia 1 quando la parola è presente almeno una volta nel testo e 0 in caso contrario.

Esempio. Ipotizziamo di avere 2 documenti, ciascuno dei quali è costituito da una sola frase:

1. "Il gatto è sopra il tavolo";
2. "Il cane è sotto il tavolo".

Dopo il *preprocessing* queste diventeranno:

1. "gatto sopra tavolo";
2. "cane sotto tavolo".

In esse sono presenti 5 parole distinte, per cui un generico ingresso sarà costituito da un vettore di 5 elementi, che corrispondono alle parole:

$$\langle gatto, cane, sopra, sotto, tavolo \rangle.$$

In base a ciò i due ingressi vengono rappresentati come:

1. $x^{(1)} =< 1, 0, 1, 0, 1 >$;

2. $x^{(2)} =< 0, 1, 0, 1, 1 >$;

Notiamo che adottando un modello naive stiamo ipotizzando che la presenza di ciascuna parola sia condizionalmente indipendente dalle altre. Questa ipotesi intuitivamente sembra non vera, infatti se è presente la parola "automobile" è ragionevole credere che la presenza della parola "guidare" non sia completamente scorrelata. In realtà si vede che questa ipotesi semplificativa porta a classificatori che funzionano piuttosto bene.

Stima delle probabilità L'ultima informazione necessaria per applicare la regola di Bayes per determinare la stima MAP sono le probabilità a priori, cioè le varie $p(y_j)$, e le probabilità condizionate semplici $p(x_i|y_j)$. Quando il campione di dati a disposizione è piuttosto ampio, possiamo adottare un approccio frequentista per individuare una stima di tali probabilità. In particolare, per ciascuna classe, è sufficiente calcolare il rapporto tra il numero di documenti di tale classe ed il numero totale di documenti:

$$\hat{p}(y_j) = \frac{\text{n.ro documenti } y_j}{\text{n.ro totale documenti}}.$$

In maniera simile potremmo stimare le probabilità condizionate:

$$p(x_i|y_j) = \frac{p(x_i, y_j)}{p(y_j)}$$

ovvero:

$$\hat{p}(x_i|y_j) = \frac{\text{n.ro documenti } (x_i \cap y_j)}{\text{n.ro documenti } y_j}$$

in cui il numeratore rappresenta il numero di documenti di classe y_j in cui compare il termine x_i. Ma cosa accadrebbe se per una particolare coppia classe-ingresso non ci fosse alcun campione d'esempio tra i documenti considerati? Avremmo $p(x_i|y_j) = 0$, e quindi l'Equazione 20 si annullerebbe mentre l'Equazione 21 andrebbe a $-\infty$.

Laplace smoothing Questa tecnica modifica leggermente le formule per il calcolo delle probabilità condizionate e consente di risolvere il problema appena descritto. La formula proposta consiste nella seguente espressione:

$$\hat{p}(x_i|y_j) = \frac{\text{n.ro documenti } (x_i \cap y_j) + 1}{\text{n.ro documenti } y_j + |x|}.$$

Così facendo, gli attributi non rappresentati in alcun documento saranno equiprobabili con probabilità $1/(\text{n.ro documenti } y_j + |x|)$ dove $|x|$ è il numero totale di tutti i possibili valori delle x.

7 Algoritmi non supervisionati

A differenza degli algoritmi supervisionati, in quelli non supervisionanti per ciascun esempio non è nota l'uscita, cioè la sua classe di appartenenza, e dunque non è possibile effettuare una fase di *training*. Lo scopo di questi algoritmi può essere quello di raggruppare esempi simili (*clustering*) oppure ridurre la dimensionalità dei vettori delle *feature* (*dimensionality reduction*).

7.1 Clustering

Lo scopo del *clustering* è raggruppare in maniera "naturale" le istanze fornite. Si punta quindi a creare dei *cluster* a cui appartengano degli elementi che condividono alcune proprietà e che, dunque, sono simili tra loro. Le proprietà che caratterizzano un *cluster* sono due:

- alta similarità *intra-cluster*, cioè tra istanze appartenenti allo stesso *cluster* (che pertanto si dirà "omogeneo");
- bassa similarità *inter-cluster*, cioè tra istanze appartenenti a *cluster* distinti.

Gli algoritmi di *clustering* si distinguono in *hard* e *soft clustering*:

- si parla di *hard clustering* quando l'algoritmo determina la classe di appartenenza di un'istanza, senza dare alcuna informazione su quanto è probabile che quella sia la reale classe di appartenenza;
- si parla di *soft clustering* quando l'algoritmo indica, per ciascuna istanza, qual è la probabilità che appartenga a ciascun *cluster*.

7.1.1 K-means

Si tratta di un semplice algoritmo di *hard clustering* che si articola nelle seguenti fasi:

1. fissa il numero di *cluster* k in maniera casuale o basandoti su un euristica nota;
2. scegli casualmente k punti tra le istanze note ed assumi che siano i centroidi dei k *cluster*;
3. assegna ogni rimanente punto al *cluster* il cui centroide è più vicino (per valutare la distanza si può usare una distanza Euclidea, di Manhattan o altro ancora);

4. in base ai campioni aggiunti in ciascun *cluster* ricalcola la posizione dei centroidi[23];

5. torna al punto 3 fino al raggiungimento della condizione di convergenza.

La condizione di convergenza può essere un numero massimo di iterazioni oppure il momento in cui ciascun centroide dista dal suo valore precedente di una quantità minore ad una soglia desiderata. Al termine dell'algoritmo avremo minimizzato la distanza quadratica delle istanze dai rispettivi centroidi.

Un vantaggio di questo algoritmo è costituito dalla sua semplicità, ma presenta alcuni svantaggi, tra cui:

- *Non è adatto per dati di tipo categoriale.* Per funzionare, infatti, è necessario poter stabilire una distanza tra due generiche istanze. Affinché ciò sia possibile i vettori delle *feature* devono essere numerici, o quantomeno dobbiamo essere in grado di applicare una metrica di distanza tra coppie di istanze.

- *Necessita di specificare k in anticipo.* In questa categoria di problemi non abbiamo informazioni a priori su quali e quante classi cercare, quindi diventa difficile indovinare il k esatto fin da subito. Una possibilità è avviare più volte l'algoritmo con k diversi e confrontare i risultati. Vedremo a breve come si può realizzare ciò.

- *Può restare bloccato in minimi locali.* Ad esempio può accadere quanto mostrato in Figura 8, nella quale in quello che dovrebbe essere un unico *cluster* è possibile individuare sottogruppi di istanze molto simili che agli occhi dell'algoritmo appaiono come *cluster* distinti. La *clusterizzazione* che ne consegue è palesemente errata.

- *Molto sensibile a rumore ed outlier.* Se, casualmente, un *outlier* fosse scelto come centroide iniziale, questo potrebbe influire negativamente sul resto dell'algoritmo. Per ovviare a ciò si può rilanciare *k-means* più volte e vedere a che risultati porta.

- *Non adatto a cluster con forme non convesse.*

Abbiamo visto che uno dei punti delicati del *k-means* è la scelta di k. Per capire quale sia il valore ottimo è possibile variare k e *plottare* i risultati.

[23] Ciascuna coordinata del centroide di un *cluster* è calcolata come la media del valore che tale coordinata assume per tutti i punti del *cluster*.

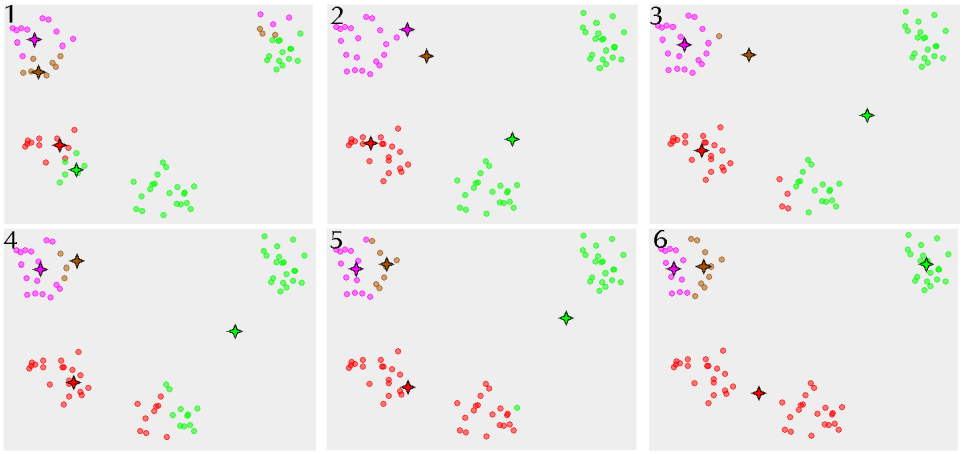


Figura 8: K-Means converge in un minimo locale

In particolare studieremo come varia la *Misura di Distorsione dei Cluster* in funzione di k, dove essa è definita come segue[24]:

$$J(k) = \frac{1}{N} \sum_{i=1}^{N} ||x^{(i)} - \mu_c^{(i)}||^2$$

in cui:

- N è il numero complessivo di istanze;
- $x^{(i)}$ è la *i-esima* istanza;
- $\mu_c^{(i)}$ sono le coordinate del centroide del *cluster* a cui appartiene $x^{(i)}$;
- $||x^{(i)} - \mu_c^{(i)}||^2$ è la distanza quadratica tra ciascuna istanza ed il suo centroide.

Ciò che ci interessa è minimizzare la somma delle distanze quadratiche, plottando J in funzione di k. Un valore troppo alto di J ci indica, ad esempio, che il *clustering* non è significativo perché due o più *cluster* potrebbero essere stati raggruppati attorno ad un unico centroide. Un valore troppo basso, però, è indice del fatto che abbiamo usato un numero troppo elevato di centroidi. Nel caso estremo avremo tanti centroidi quanti punti e questo porterà ad avere errore nullo. Tipicamente si può vedere che l'andamento

[24] A volte si usa la *Within Groups Sum of Squares* che è identica a $J(k)$ ma non normalizzata rispetto al numero di punti.

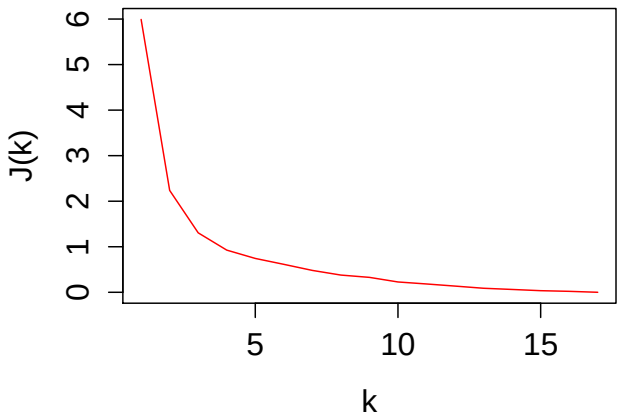


Figura 9: Andamento tipico della misura di distorsione dei cluster in funzione del numero di centroidi.

di J è decrescente (simile a quello in Figura 9) e si assume che il valore vero di k coincida con il ginocchio di tale curva (cioè il punto in cui la pendenza comincia ad essere meno ripida, determinato tramite un'opportuna famiglia di algoritmi).

7.2 Principal Component Analysis

7.2.1 In sintesi

Lo scopo di questa tecnica è passare da una rappresentazione delle istanze in n variabili ad una rappresentazione in p variabili *latenti* (con $p << n$). Nota bene che queste variabili sono *latenti*, cioè non si tratta semplicemente di eliminare alcune *feature* da quelle iniziali, ma di applicare un modello (lineare) alle *feature* per *proiettarle* in un nuovo dominio con dimensionalità inferiore. Questo nuovo dominio sarà caratterizzato da alcuni assi cartesiani, le coordinate di ciascun punto secondo il nuovo sistema di riferimento possono essere ottenute proiettando ciascun punto su ciascun asse. Le proiezioni dei punti su ciascun asse daranno luogo ad una distribuzione di valori e per ciascun asse è possibile calcolare la varianza di tale distribuzione. Poiché una varianza maggiore è indice del fatto che quella componente *discrimina* maggiormente i punti, saremo interessati a prendere in considerazione so-

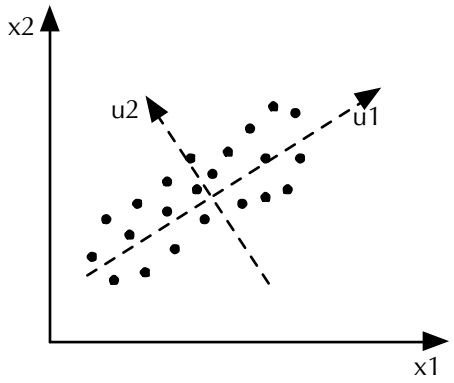


Figura 10: Esempio PCA

lo le p componenti a varianza maggiore (componenti principali). In questa maniera passiamo da una matrice degli ingressi di dimensione $m \times n$ ad una di dimensione $m \times p$. L'idea alla base delle tecniche di questo tipo è che le variabili osservabili siano in qualche maniera correlate e derivabili tutte da un insieme (più piccolo) di variabili latenti.

Capiamo meglio il concetto con un esempio. In Figura 10 sono mostrati alcuni punti, ciascuno dei quali è rappresentato da una coppia di valori $< x_1, x_2 >$. La PCA ci consente di passare da una rappresentazione in $\mathcal{R}^2$ ad una rappresentazione in $\mathcal{R}$ (scegliendo $p = 1$). Le componenti individuate dall'algoritmo (in seguito vedremo come) sono u_1 ed u_2. Proiettando i punti su u_1 ed u_2 ci rendiamo conto che la varianza delle proiezioni su u_1 è maggiore. Concludiamo che sarà questo asse la *componente principale* ricercata e sarà sufficiente a descrivere il *set* di istanze. In questo semplice esempio è facile capire che quanto fatto sia corretto, infatti tra x_1 ed x_2 sembra esserci una relazione lineare, e tra due variabili linearmente dipendenti è sufficiente conoscerne una per non perdere alcuna informazione (in questo caso la lineare dipendenza non è perfetta, quindi preserviamo gran parte dell'informazione, ma non tutta).

7.2.2 Passi necessari per la PCA

Descriviamo quali passi sono necessari per portare a termine la PCA.

La prima operazione da fare è l'*azzeramento del valor medio*, cioè a ciascuna *feature*, di ciascuna istanza, sottraiamo il valor medio della stessa

feature calcolato su tutte le istanze. Passiamo quindi da

$$x_1^{(i)}, x_2^{(i)}, \ldots, x_n^{(i)} \qquad i = 1, \ldots, m$$

a

$$x_1^{(i)} - \mu_1, x_2^{(i)} - \mu_2, \ldots, x_n^{(i)} - \mu_n \qquad i = 1, \ldots, m$$

con

$$\mu_j = \frac{1}{m} \sum_{i=1}^{m} x_j^{(i)}.$$

Calcoliamo poi la matrice di covarianza (Σ) tra tutte le possibili coppie di *feature*, il cui singolo elemento è così definito:

$$\begin{aligned} \Sigma_{ij} &= \mathrm{Cov}(x_i, x_j) \\ &= E[(x_i - \mu_i)(x_j - \mu_j)] \\ &= E[(x_i)(x_j)] \end{aligned}$$

Ricordiamo che la covarianza tiene conto di quanto due variabili aleatorie variano insieme, cioè di quanto sono simili nel loro comportamento.

Ora occorre conoscere autovalori ed autovettori della matrice delle covarianze. Per fare ciò applichiamo la *Singular Value Decomposition* (SVD), anche nota come *Decomposizione in Valori Singolari*, alla matrice Σ. Si tratta di una particolare fattorizzazione che, partendo da una matrice $A \in \mathbb{C}^{m \times n}$ ci consente di scrivere:

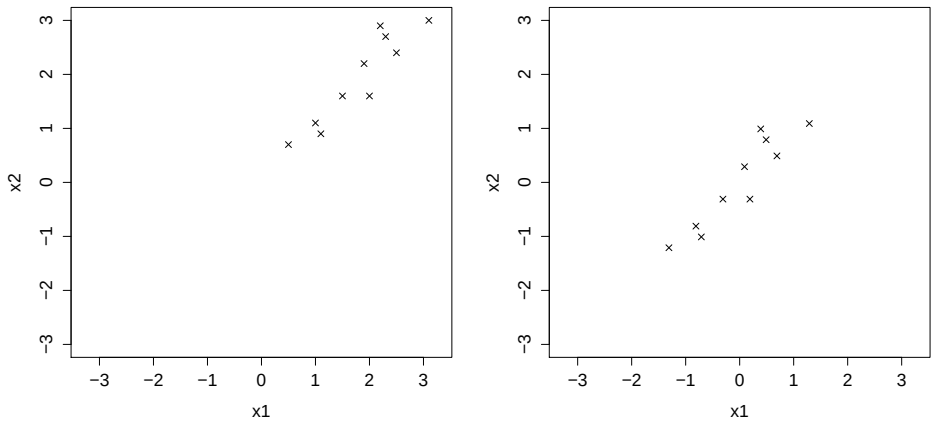
$$A = USV^*$$

dove:

- U è una matrice unitaria sinistra di dimensioni $m \times m$, cioè tale che moltiplicandola per la sua trasposta coniugata si ottiene la matrice identità ($UU^* = I$). Questa proprietà, nel caso di matrici reali, si traduce in $U^{-1} = U^T$;
- S è una matrice diagonale di dimensioni $m \times n$ (dove solo i primi n elementi sono non nulli e costituiscono gli autovalori di A);
- V^* è la trasposta coniugata di una matrice unitaria di dimensioni $n \times n$.

Ricordiamo che la trasposta coniugata di una matrice si ottiene trasponendone gli elementi e sostituendo ciascuno di essi con il suo complesso coniugato (l'ordine delle due operazioni è irrilevante).

Gli elementi sulla diagonale principale di S rappresentano gli autovalori di A, posti in ordine di valore decrescente man mano che ci si sposta verso



(a) Punti iniziali (b) Punti con media azzerata

Figura 11: Campioni a cui applicare la PCA

il basso. Le colonne della matrice U rappresentano gli autovettori sinistri di A (cioè i vettori $\mathbf{u}$ t.c. $\mathbf{u}A = \mathbf{u}\lambda$ con λ autovalore) e le colonne della matrice V (e non V^*) rappresentano gli autovettori destri di A (calcolati come $A\mathbf{v} = \lambda\mathbf{v}$). Nel caso di matrici simmetriche (come la matrice delle covarianze) questi autovalori coincidono, e quindi $U = V$.

Ciò che si fa nel caso della PCA è applicare la SVD alla matrice delle covarianze:

$$\Sigma = USV^*$$

a questo punto di tutti gli autovalori che compaiono in S ci interessa conservare solo i primi p, per questi autovalori consideriamo i corrispondenti autovettori. Ciascun autovettore sarà costituito da n elementi, avremo quindi una matrice degli autovettori di dimensione $n \times p$. Moltiplicare la matrice degli ingressi per l'autovettore *i-esimo* significa calcolare il valore della coordinata *i-esima* di ciascun ingresso rispetto al nuovo sistema di riferimento. Quindi moltiplicare la matrice degli ingressi $(m \times n)$ per l'intera matrice degli autovettori $(n \times p)$ equivale ad effettuare il cambio di coordinate richiesto per tutti gli ingressi. Il risultato sarà la nuova matrice dei campioni di dimensione ridotta $(m \times p)$.

Esempio. Chiariamo la procedura tramite un esempio numerico[25] eseguito

[25] Esempio trattato a lezione e preso da qui.

in R. I dati a disposizione sono rappresentati dalle seguenti coppie di punti (Figura 11a):

```
> show (dataset)
    x1  x2
1  2.5 2.4
2  0.5 0.7
3  2.2 2.9
4  1.9 2.2
5  3.1 3.0
6  2.3 2.7
7  2.0 1.6
8  1.0 1.1
9  1.5 1.6
10 1.1 0.9
```

Eseguiamo ora i passi della PCA:

1. *Azzeramento della media*, che effettuiamo tramite la funzione `scale`, per ottenere la nuova matrice degli ingressi con le medie azzerate (mostrati in Figura 11b):

```
> scaled.dataset <- scale(dataset, TRUE, FALSE)
> show(scaled.dataset)
          x1    x2
 [1,]  0.69  0.49
 [2,] -1.31 -1.21
 [3,]  0.39  0.99
 [4,]  0.09  0.29
 [5,]  1.29  1.09
 [6,]  0.49  0.79
 [7,]  0.19 -0.31
 [8,] -0.81 -0.81
 [9,] -0.31 -0.31
[10,] -0.71 -1.01
```

2. *Calcolo della matrice delle covarianze*:

```
>  Sigma <- cov(scaled.dataset)
>  show(Sigma)
          x1        x2
x1 0.6165556 0.6154444
x2 0.6154444 0.7165556
```

3. *Decomposizione ai valori singolari*:

```
> results <- svd(Sigma)
>  show(results)
$d
[1] 1.2840277 0.0490834

$u
           [,1]       [,2]
```

```
[1,] -0.6778734 -0.7351787
[2,] -0.7351787  0.6778734

$v
           [,1]       [,2]
[1,] -0.6778734 -0.7351787
[2,] -0.7351787  0.6778734
```

Notiamo che l'*output* è costuito da un vettore e due matrici. Il vettore (`d`) contiene i due autovalori (è la diagonale principale di S), mentre le matrici (`u` e `v`) contengo rispettivamente gli autovettori sinistri e destri, uno per colonna, e corrispondono quindi ad U e V (non V^*). Notiamo anche che, essendo Σ reale, accade che $u = v$.

4. *Proiezione dei dati sul nuovo sistema di riferimento.* Volendo ridurre il numero di *feature* da 2 ad 1, ci limitiamo a considerare l'autovalore più grande (il primo) ed il corrispondente autovettore (la prima colonna di `u` o `v`). A questo punto moltiplichiamo la matrice dei campioni per questo autovettore ed otteniamo le coordinate dei punti nel nuovo spazio:

```
> new.dataset <- scaled.dataset %*% results$u[,1]
> show(new.dataset)
             [,1]
 [1,] -0.82797019
 [2,]  1.77758033
 [3,] -0.99219749
 [4,] -0.27421042
 [5,] -1.67580142
 [6,] -0.91294910
 [7,]  0.09910944
 [8,]  1.14457216
 [9,]  0.43804614
[10,]  1.22382056
```

A questo punto la PCA è conclusa ma, poiché stiamo lavorando con punti del piano, possiamo visualizzare graficamente quali siano le due componenti individuate. Per disegnare ciascuna componente è sufficiente sovrappore al *plot* dei punti (scalati) la retta con intercetta nulla e coefficiente angolare pari al rapporto tra il primo ed il secondo elemento dell'autovettore (equivalentemente si potrebbero usare i punti iniziali e *plottare* questa retta con le coordinate traslate di una quantità pari alle medie sottratte in partenza).

```
plot(scaled.dataset, pch = 4, xlim = c(-3,3), ylim = c(-3,3))

m1 <-  results$u[,1][1] / results$u[,1][2]
comp1 <- matrix(c(seq(-3,3), m1*seq(-3,3)), 7, 2)
points(comp1, col="red", type = "l")
```

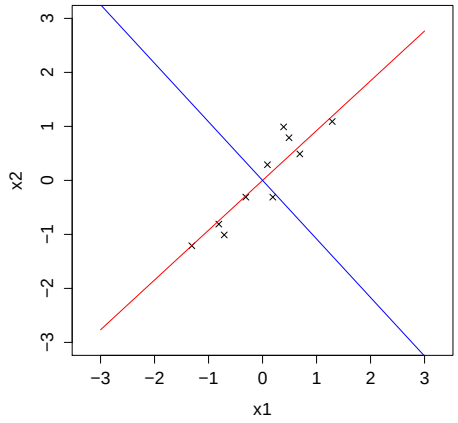


Figura 12: Componenti individuate tramite PCA

```
m2 <-  results$u[,2][1] / results$u[,2][2]
comp2 <- matrix(c(seq(-3,3), m2*seq(-3,3)), 7, 2)
points(comp2, col="blue", type = "l" )
```

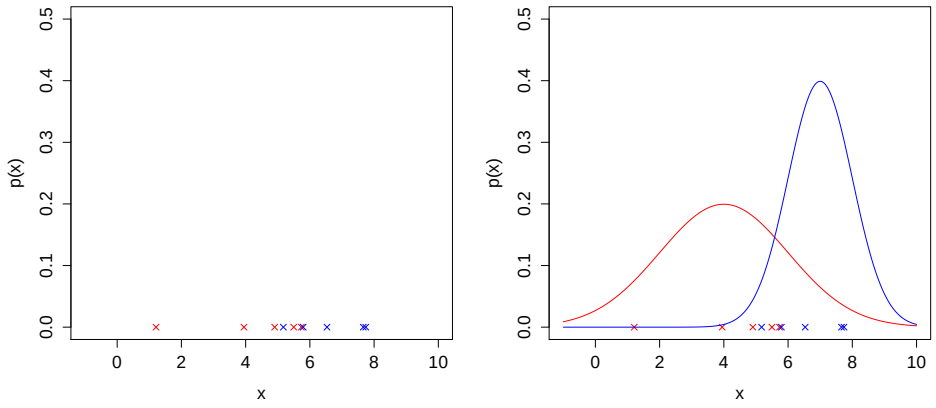
Il risultato è mostrato in Figura 12. Effettivamente notiamo che le proiezioni dei punti sulla prima componente (quella disegnata in rosso) hanno varianza maggiore. Verifichiamo se quest'ultima affermazione è vera calcolando le varianze delle proiezioni dei punti sui due assi:

```
> var(scaled.dataset %*% results$u[,1])
          [,1]
[1,] 1.284028
> var(scaled.dataset %*% results$u[,2])
           [,1]
[1,] 0.0490834
```

Questi risultati confermano quanto ci aspettavamo e notiamo che la varianza lungo ciascuna componente è pari proprio all'autovalore che determina tale componente (motivo per cui consideriamo gli autovalori più grandi).

7.3 Soft clustering tramite GMM

In questo capitolo esamineremo i *mixture model*. Si tratta di modelli probabilistici, cioè di funzioni densità di probabilità, ottenuti dalla somma pesata di densità di probabilità elementari. In particolare ci occuperemo di *Gaussian Mixture Model* (GMM) cioè di modelli probabilistici ottenuti dalla somma pesata di un certo numero di funzioni gaussiane. L'ambito in cui applicheremo le GMM è il *soft clustering*, cioè esamineremo una procedura, basata sull'utilizzo di GMM, che sia in grado di dirci qual è la probabilità che un punto appartenga a ciascun *cluster* individuato in una nuvola di punti.



(a) (b)

Figura 13: Punti da *clusterizzare* e gaussiane da cui sono stati estratti.

7.3.1 L'idea in sintesi

Per capire l'approccio adottato partiamo da un semplice esempio. Ipotizziamo di voler effettuare il *clustering* di un certo numero di punti caratterizzati da una sola coordinata. Essi sono rappresentabili su una retta, come mostrato sull'asse x in Figura 13a. Per comodità i punti di ciascun *cluster* sono rappresentati con un colore differente. Ipotizziamo che i punti appartenenti a ciascun *cluster* siano distribuiti come se fossero stati estratti da una gaussiana. Quindi, nel nostro esempio, avremo due gaussiane con parametri diversi tra di loro (Figura 13b). Se riuscissimo a determinare tali parametri, saremmo in grado di calcolare la probabilità che un nuovo punto sia stato generato dalla prima o dalla seconda gaussiana. Ciò significherebbe conoscere

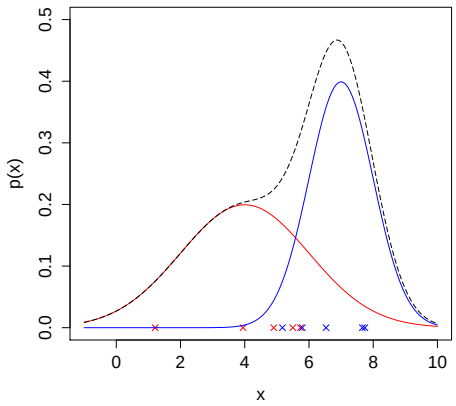


Figura 14: Somma delle due gaussiane.

la probabilità che il punto appartenga al primo *cluster* e la probabilità che appartenga al secondo *cluster*, ovvero avremmo realizzato *soft clustering*.

Per determinare i parametri delle due gaussiane operiamo direttamente sul GMM corrispondente, cioè stiamo ipotizzando che i punti da *clusterizzare* siano stati generati da un GMM. In prima analisi potremmo immaginarlo come la somma delle due gaussiane, così come mostrato dalla linea tratteggiata in Figura 14. Tuttavia sorge un problema: come ben noto l'area sottesa da ciascuna gaussiana è unitaria, quindi l'area sottesa dalla loro somma non lo sarà. Affinché il GMM sia un modello probabilistico, ovvero affinché sia una densità di probabilità, è necessario che sottenda anch'esso un'area unitaria. Per tale motivo ciascuna delle gaussiane di partenza viene moltiplicata per una quantità compresa tra 0 ed 1 prima di essere sommata alle altre. Intuitivamente questo giustifica il fatto che un GMM sia una somma pesata di distribuzioni elementari. Vedremo in seguito che significato fisico è possibile dare a questi pesi.

Il caso esaminato fin qui prevede che ciascun punto sia descritto da una sola coordinata. Nel caso di punti a 2 o più variabili l'idea di fondo rimane invariata, mentre cambiano alcuni dettagli. In particolare non potremo più usare una semplice distribuzione gaussiana univariata, ma dovremo ricorrere ad una gaussiana multivariata. Nel caso in cui i punti da *clusterizzare* si trovino sul piano ($n = 2$) la mistura di gaussiane assume una forma simile a quella mostrata in Figura 15. In essa è rappresentato un GMM formato da 3 distribuzioni gaussiane bidimensionali (o bivariate).

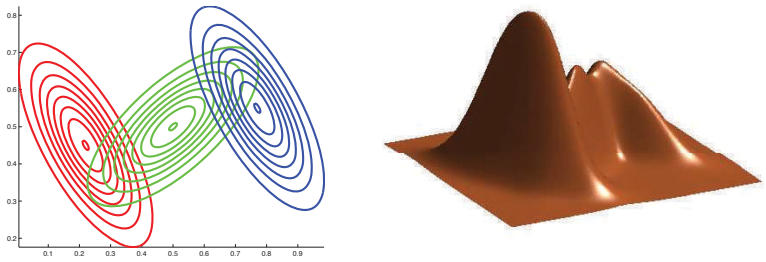


Figura 15: Esempio di GMM con 3 gaussiane bivariate

7.3.2 Dettagli matematici

Nei paragrafi che seguono formalizziamo meglio il problema ed approfondiamo alcuni dettagli.

Mixture model in formule Fin qui abbiamo definito una mistura di gaussiane come una somma pesata di densità di probabilità gaussiane. Più in generale possiamo parlare di *Mixture Model* come una somma pesata di densità di probabilità, senza specificarne la natura. Ipotizzando di voler creare una mistura con k componenti, la formulazione più generica per la densità di probabilità del modello è[26]:

$$p(\mathbf{x}) = \sum_{j=1}^{k} p(\mathbf{x} \wedge z = j).$$

Nella precedente $p(\mathbf{x} \wedge z = j)$ rappresenta la probabilità che il punto $\mathbf{x}$ sia stato generato dalla *j-esima* componente. Applicando la regola del prodotto possiamo riscriverla nella formulazione tipicamente usata:

$$p(\mathbf{x}) = \sum_{j=1}^{k} p(\mathbf{x}|z = j)p(z = j).$$

In essa z rappresenta una variabile casuale discreta (latente) che assume valori da 1 a k. $p(z = j)$ indica la probabilità che la *j-esima* componente della mistura venga "attivata" e generi il campione. A livello intuitivo possiamo pensare al processo di generazione di un punto in questa maniera:

- la componente *j-esima* viene selezionata per la generazione del punto con una probabilità $p(z = j)$;

[26] Per una formulazione più formale vedi qui.

- il punto viene generato dalla componente *j-esima* con una probabilità $p(\mathbf{x}|z = j)$ che dipende *solo* dalla natura di questa componente (cioè dalla sua specifica distribuzione di probabilità).

In definitiva, possiamo vedere la precedente formula come una risposta alla domanda:

- "Con quale probabilità la mistura può generare un punto $\mathbf{x}$?"

La risposta fornita dalla formula è:

- "Per ciascuna distribuzione, considera la probabilità che sia essa a generare quel punto e moltiplicala per la probabilità con cui essa lo genererebbe. In seguito somma tutte queste quantità e saprai con che probabilità la mistura darà vita al punto $\mathbf{x}$."

A questo punto possiamo semplificare la notazione introducendo i coefficienti $\pi_j = p(z = j)$ che ci consentono di scrivere:

$$p(\mathbf{x}) = \sum_{j=1}^{k} \pi_j p_j(\mathbf{x}).$$

$p_j(\mathbf{x})$ rappresenta la densità di probabilità della *j-esima* componente calcolata nel punto $\mathbf{x}$, mentre i vari π_j prendono il nome di *mixing coefficients* e sono proprio i pesi di cui abbiamo parlato finora. Notiamo anche che, trattandosi di valori di probabilità, la loro somma è unitaria:

$$\sum_{j=1}^{k} \pi_j = \sum_{j=1}^{k} p(z = j) = 1.$$

Poiché il nostro scopo sarà individuare i parametri migliori per le singole distribuzioni, conviene adottare una notazione più esplicita, cioè la seguente:

$$p(\mathbf{x}|\boldsymbol{\theta}) = \sum_{j=1}^{k} \pi_j p_j(\mathbf{x}|\boldsymbol{\theta_j})$$

in cui $\boldsymbol{\theta}_j$ rappresenta i parametri della componente *j-esima*, ad esempio media e varianza di una gaussiana univariata, mentre $\boldsymbol{\theta}$ è una particolare istanza di tutti i parametri $\boldsymbol{\theta}_j$.

Parametri di un GMM Fin qui abbiamo parlato genericamente di "parametri" di un modello. Scendiamo ora nel dettaglio delle GMM. Il primo passo da fare è sostituire la generica densità di probabilità $p_j(\mathbf{x}|\boldsymbol{\theta_j})$ con la *j-esima* gaussiana multivariata, la cui espressione è:

$$\mathcal{N}(\mathbf{x}|\boldsymbol{\mu_j}, \boldsymbol{\Sigma_j}) = \frac{1}{\sqrt{(2\pi)^n |\boldsymbol{\Sigma}_j|}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_j)^T \boldsymbol{\Sigma}_j{}^{-1}(\mathbf{x} - \boldsymbol{\mu}_j)\right)$$

nella quale:

- n rappresenta la dimensione di ciascun ingresso (e quindi il numero di *feature*);
- $\boldsymbol{\mu}_j$ è il vettore dei valori medi della *j-esima* gaussiana;
- $\boldsymbol{\Sigma}_j$ è la matrice delle covarianze della *j-esima* gaussiana.

I parametri che desideriamo individuare sono quindi:

$$\boldsymbol{\mu}_j \text{ e } \boldsymbol{\Sigma}_j \quad j = 1, \ldots, k.$$

Nel caso di gaussiane univariate questi si riducono ai valori medi (μ_j) ed alle varianze (σ_j^2). Se per ogni *cluster* sapessimo quali punti appartengono ad esso, potremmo calcolare le stime di media e varianza tramite le formule note:

$$\hat{\mu}_j = \frac{1}{|C_j|} \sum_{x^{(i)} \in C_j} x^{(i)}$$

$$\hat{\sigma}_j^2 = \frac{1}{|C_j|-1} \sum_{x^{(i)} \in C_j} (x^{(i)} - \hat{\mu}_j)^2$$

dove C_j rappresenta il *j-esimo cluster* e $|C_j|$ è il numero di punti di cui è composto. Chiaramente lo stesso discorso può essere esteso al caso multivariato:

$$\hat{\boldsymbol{\mu}}_{\boldsymbol{j}} = \frac{1}{|C_j|} \sum_{\mathbf{x^{(i)}} \in C_j} \mathbf{x^{(i)}}$$

$$\hat{\Sigma}_j = \frac{1}{|C_j|-1} \sum_{\mathbf{x^{(i)}} \in C_j} (\mathbf{x^{(i)}} - \hat{\boldsymbol{\mu}}_{\boldsymbol{j}})(\mathbf{x^{(i)}} - \hat{\boldsymbol{\mu}}_{\boldsymbol{j}})^T$$

Purtroppo, però, non è noto a priori quali punti siano stati generati da ciascun cluster, altrimenti il problema del *clustering* non si porrebbe neppure. Potremmo quindi pensare di assegnare ciascun punto ad un *cluster* in base alla gaussiana a cui appartiene con maggiore probabilità. Fare ciò, tuttavia, richiederebbe di conoscere i parametri delle gaussiane, che non sono affatto noti. Ci troviamo quindi in questa situazione:

- per stimare i parametri di ciascuna gaussiana dovremmo conoscere i punti che le appartengono;
- per conoscere i punti che appartengono a ciascuna gaussiana dovremmo stimarne i parametri.

La procedura che risolve questo problema si chiama *Expectation-Maximization* (EM).

Algoritmo EM per GMM L'algoritmo EM applicato ad un GMM può essere sintetizzato nei seguenti passaggi[27]:

1. fissato k inizializza casualmente i parametri delle gaussiane ed i pesi π_j;
2. passo E (*Expectation*): per ogni coppia punto-*cluster* calcola alcuni coefficienti, detti *responsibility*, definiti come segue:

   $$r_{ij} \triangleq p(z = j|\mathbf{x^{(i)}}, \boldsymbol{\theta}^{t-1})$$

   dove $\boldsymbol{\theta}^{t-1}$ rappresenta i parametri del modello all'iterazione precedente. Ciascun coefficiente tiene conto di quanto il *cluster j-esimo* sia responsabile del punto *i-esimo* ed è calcolato come il rapporto tra la probabilità che il punto sia stato generato dal *cluster* e quella che sia stato generato dall'intero GMM:

   $$r_{ij} = \frac{\pi_j p_j(\mathbf{x}^{(i)}|\boldsymbol{\theta_j}^{(t-1)})}{\sum_{j=1}^{k} \pi_j p_j(\mathbf{x}^{(i)}|\boldsymbol{\theta_j}^{(t-1)})}$$

3. passo M (*Maximization*): per ogni gaussiana ricalcola i parametri π_j, $\boldsymbol{\mu}_j$ e $\boldsymbol{\Sigma}_j$ in base alle *responsibility* appena calcolate, con lo scopo di

[27] I dettagli sulle formule sono in "Machine Learning - A Probabilistic Perspective", Sezione 11.4.2

massimizzare la *likelihood* (cioè la probabilità che quei punti siano stati generati dal GMM):

$$\pi_j = \frac{1}{m}\sum_{i=1}^{m} r_{ij}$$

$$\boldsymbol{\mu_j} = \frac{\sum_i r_{ij}\mathbf{x^{(i)}}}{\sum_i r_{ij}}$$

$$\mathbf{\Sigma_j} = \frac{\sum_i r_{ij}(\mathbf{x}^{(i)} - \boldsymbol{\mu_j})(\mathbf{x}^{(i)} - \boldsymbol{\mu_j})^T}{\sum_i r_{ij}}$$

4. ripeti i passi E ed M fino a convergenza.

7.4 Outlier detection tramite densità di probabilità

La gaussiana multivariata può essere usata anche per fare *outlier detection*. Partendo da un *dataset* noto, ed ipotizzando che i campioni siano stati estratti da una distribuzione gaussiana, è possibile calcolare le stime dei parametri, come mostrato in precedenza. Prendendo come caso d'esempio quello della gaussiana univariata, sappiamo che sommando e sottraendo alla media 3 deviazioni standard ($\mu \pm 3\sigma$) otteniamo i due estremi dell'intervallo in cui ricade il 99.7% dei campioni (Figura 16). Data questa informazione possiamo classificare come *outlier* tutte le istanze del *dataset* che fuoriescono dall'intervallo stabilito. Ovviamente il discorso può essere ripetuto per gaussiane multivariate.

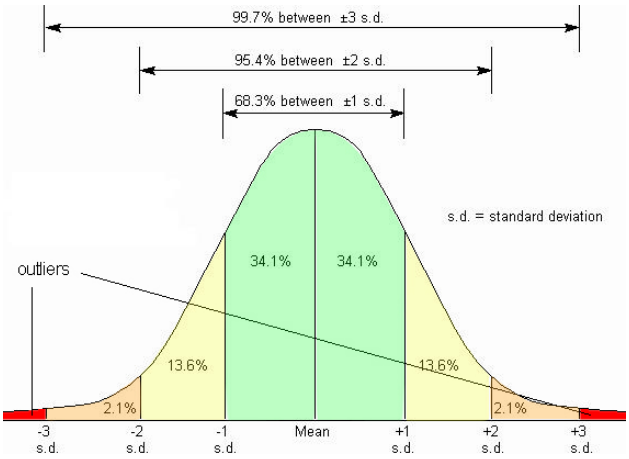


Figura 16: *Outlier detection* tramite gaussiana univariata

Riferimenti bibliografici

[1] Trevor Hastie, Robert Tibshirani e Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. New York, NY, USA: Springer New York Inc., 2001.

[2] Kevin P Murphy. *Machine learning: a probabilistic perspective*. Cambridge, MA, 2012.

[3] Stuart J. Russell et al. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Inc., 1996.