

MACHINE LEARNING

Feature Selection e Regularizzazione -

$x_j, j = 1, \dots, d \rightarrow$ variabili indipendenti/features

$\mathbf{x} = [x_0, \dots, x_d] \rightarrow$ feature vector ($x_0=1$ per comodità)

$y \rightarrow$ variabile target

$(\mathbf{x}^{(i)}, y^{(i)}), i = 1, \dots, n \rightarrow$ training samples

$\theta \rightarrow$ weight/parameter vector

Relazione tra overfitting e scelta del modello

Finora abbiamo visto solo funzioni ipotesi (o «modelli») lineari

Possiamo usare dei modelli non lineari? Forse potrebbero essere «più potenti», ovvero avere una capacità descrittiva maggiore...

Modello predittivo lineare:

$$h_{\theta}(x_1, x_2) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

Esempio di modello predittivo non-lineare (polinomio di grado 2):

$$h_{\theta}(x_1, x_2) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \theta_4 x_1^2 + \theta_5 x_2^2$$

$$h(x) = \frac{1}{\sqrt{2\pi\theta_2}} \exp - \frac{(x - \theta_1)^2}{2\theta_2^2}$$

Anche quello sopra è un esempio di modello non lineare e, ovviamente, ne potremmo pensare molti altri...

Attenzione ad un'ambiguità terminologica che è comune in ML: con «modello» spesso si indica sia la *classe* dei modelli (e.g., funzioni lineari, funzioni quadratiche, ecc.), sia la *specifica* funzione ipotesi «istanziata» con uno specifico vettore di parametri

In realtà, piuttosto che usare modelli polinomiali (o altro), per una regressione non lineare conviene usare, e.g., Support Vector Regression oppure Neural Network o Decision Tree o altri metodi che vedremo in seguito

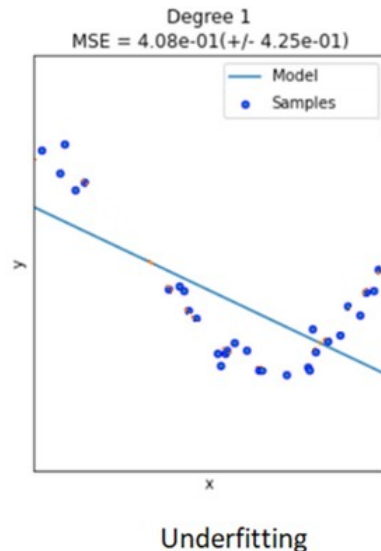
Underfitting

In un task di regressione l'utilizzo di un modello non lineare può essere utile quando so (o «sospetto»...) che la relazione tra la variabile target e le feature sia di natura, appunto, non lineare

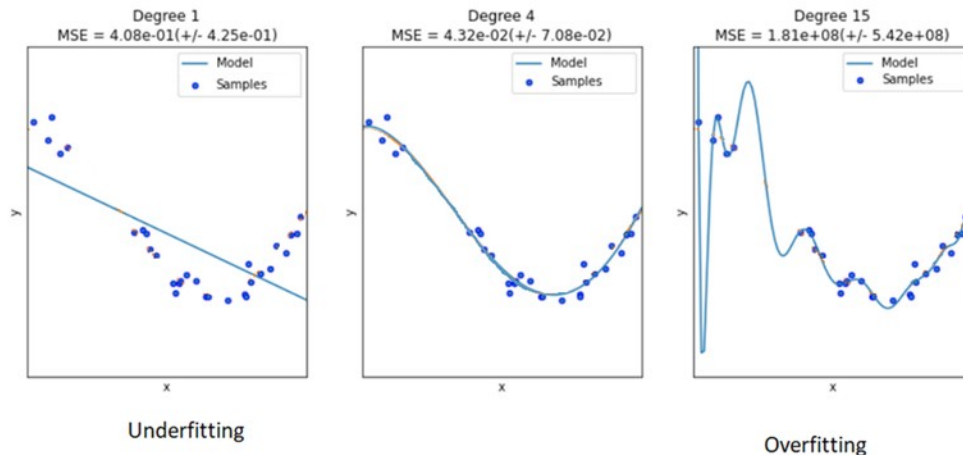
Nell'esempio della figura accanto un modello lineare non avrebbe una sufficiente capacità descrittiva dei dati perché la variabile y dipende dalla x in maniera non lineare

Quindi un modello statistico lineare, in questo caso, farebbe un sacco di errori. In questo caso si parla di «underfitting», ovvero il modello (la classe di modelli, per essere più precisi) che ho scelto non ha una capacità descrittiva sufficiente per adattarsi alla distribuzione dei dati

Come abbiamo accennato in una lezione precedente, però, esiste anche il rischio opposto, ovvero avere un modello «troppo potente» che genera overfitting.

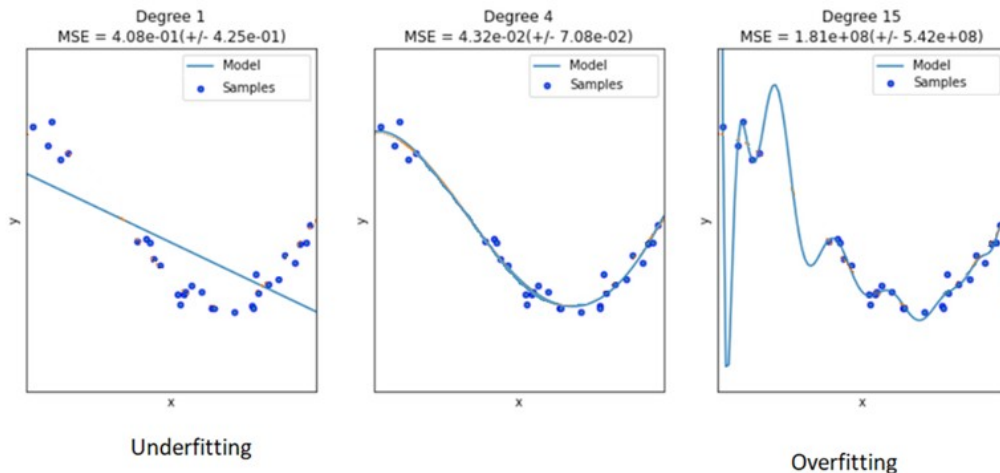


Underfitting e Overfitting: Esempio



- Addestriamo il modello lineare (polinomio di grado 1, quindi, linear regression) a sinistra e notiamo che non si adatta bene ai dati
- Addestriamo il modello polinomiale (di grado 4) al centro e notiamo che si adatta meglio ai dati
- Addestriamo il modello polinomiale di grado 15 a destra e notiamo che si adatta molto meglio
- In principio, potremmo addestrare un modello polinomiale di grado $n-1$ (dati n samples) e ottenere errore 0 sul training set, ma quasi sicuramente ciò porterà ad overfitting

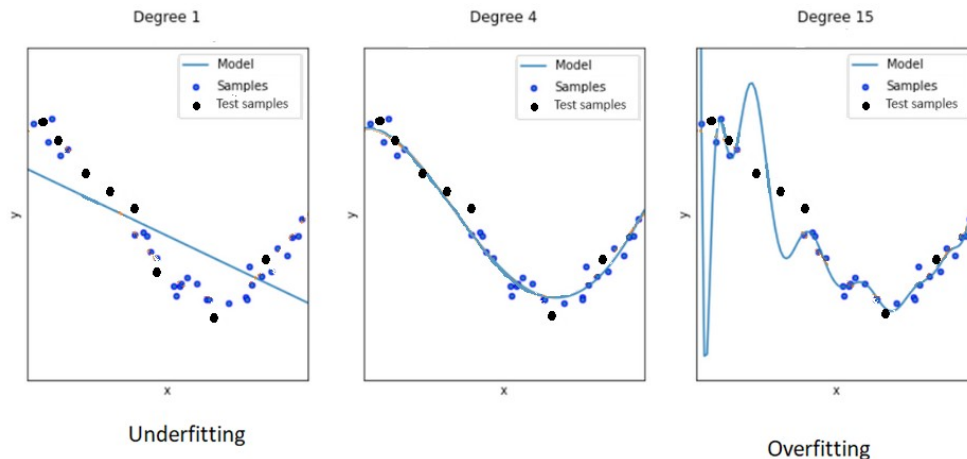
Overfitting e Rasoio di Occam



Questo è un esempio del principio del Rasoio di Occam:

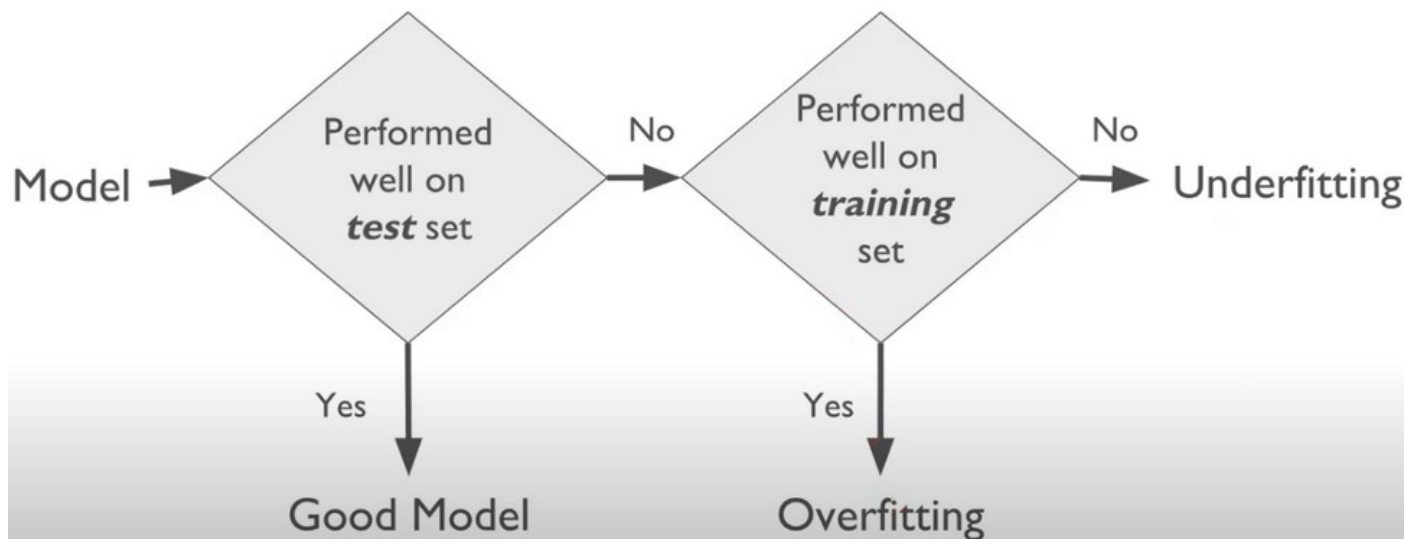
- Il modello di grado 1 è troppo semplice per spiegare le osservazioni che ho raccolto
- Il modello di grado 15 è molto complesso, spiega quasi perfettamente le mie osservazioni, ma è poco probabile che rappresenti la realtà della dipendenza statistica tra y ed x ($P(y|x)$)
- Il modello di grado 4 è un buon compromesso
- Che il modello di grado 4 sia un buon compromesso vuol dire che (probabilmente) *generalizzerà* meglio, ad inference time, quando dovrà fare delle predizioni usando dei dati non contenuti nel training set

Overfitting e Rasoio di Occam



Ora nelle figure ho aggiunto dei punti (neri) che rappresentano i dati a inference time
L'intuito (e Occam...) ci dice che è più probabile che i dati a inference time seguano il profilo del modello 4, così come, infatti, li ho disegnati in quest'esempio
In tal caso il modello 4 sarà quello che farà meno errori (= generalizzerà meglio), sia rispetto al modello 1 (che ha fatto underfitting), ma anche rispetto al modello 15 (overfitting)

Underfitting e Overfitting



Come faccio a scegliere il modello (ovvero, *la classe di modelli*) giusto?

Purtroppo non esiste una regola generale, e questo è un pò il tallone di Achille del ML

Bisogna fare delle prove, addestrare modelli con potenza descrittiva diversa e scegliere il migliore

Dove “il migliore” vuol dire quello che generalizza meglio

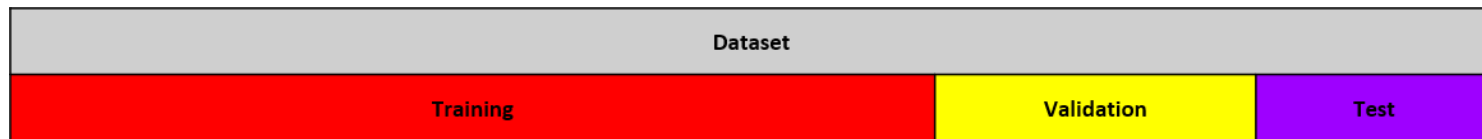
Per testare le capacità di generalizzazione dei vari modelli potrei, ad esempio, pensare di usare il testing set e valutarli usando una qualche misura di performance e infine scegliere il migliore (e.g., v. il diagramma della slide precedente)

Non posso però usare il testing set per scegliere il modello perchè il testing set mi serve per fare la valutazione finale, quando tutte le scelte implementative (modello e valore degli iperparametri compresi) sono state fissate

Se usassi il testing set per fare delle scelte in fase di sviluppo, come, ad ese., scegliere il modello migliore, *queste scelte potrebbero dipendere dagli esempi specifici del testing set*. E non avrei poi modo di valutare il modello scelto, perchè mi mancherebbe un dataset di dati “freschi”, non coinvolti nelle scelte

Scelta del modello

Userò quindi un dataset diverso, detto di *validation*, che ottengo partizionando il training set in training e validation (v. figura sotto)

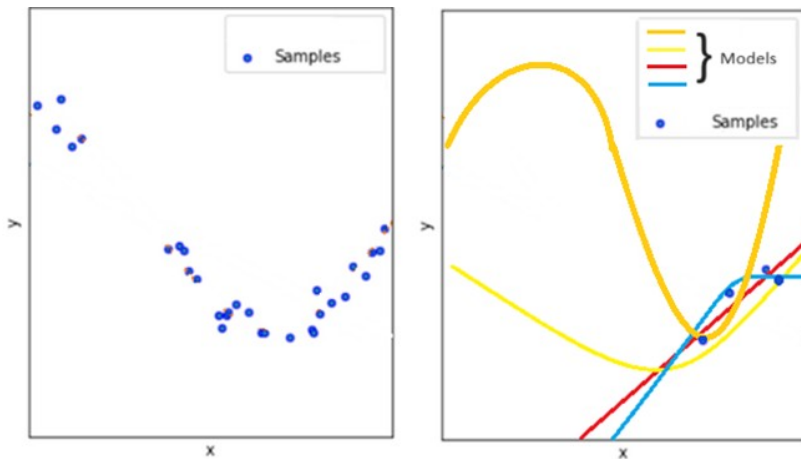


Uso il validation set per stimare la capacità di generalizzazione di vari modelli addestrati insieme ad una qualche metrica di performance
In tal modo posso scegliere il modello migliore (rispetto alla misura di performance sul validation), che, sperabilmente, sarà nè troppo complesso (overfitting), nè troppo semplice (underfitting)

Relazione tra overfitting e numero di esempi di training

L'overfitting, oltre ad essere collegato con l'utilizzo di un modello "troppo potente", è spesso collegato anche e soprattutto con la scarsità di dati di training

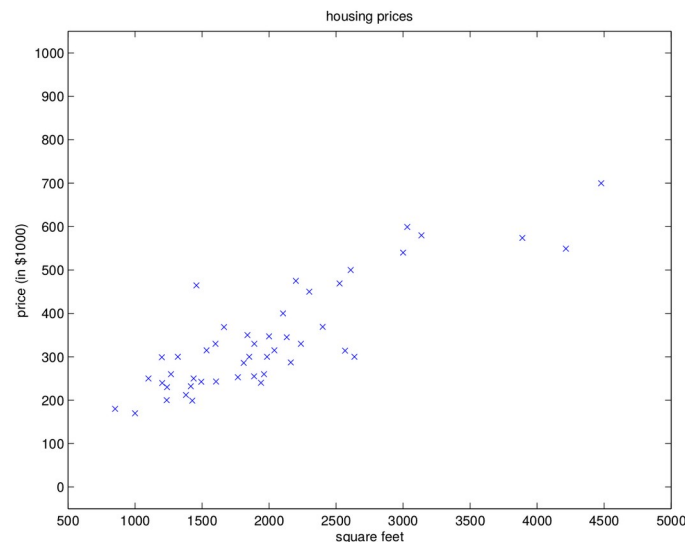
La figura a destra mostra che, se ho pochi dati di training, ovvero ho troppe poche "osservazioni", allora ci sono molte "ipotesi plausibili", ed è difficile capire quale modello si adatta meglio ai dati, dato che ognuno, a suo modo, potrebbe fare overfitting, cioè riuscire a spiegare i pochi dati di training senza però riuscire a generalizzare a inference time



Relazione tra overfitting e numero di feature

In questo esempio abbiamo una sola feature e un certo numero (n) di esempi, che supponiamo siano sufficienti a rappresentare la relazione di dipendenza statistica con la variabile target

Living area (feet ²)	Price (1000\$)
2104	400
1600	330
2400	369
1416	232
3000	540
⋮	⋮



Ma di quanti esempi ho bisogno se ho 2 feature?
E se ho d feature?

Vediamo il caso delle feature categoriche (che è più intuitivo)

- Se ho una sola feature (e.g., di natura categorica), ho bisogno di un certo numero di esempi che mi rappresentino tutti i suoi possibili valori
- Ad esempio, vorrei osservare almeno una volta ogni valore distinto di quella feature
- Per cui, se la feature può assumere m possibili valori, allora ho bisogno di almeno $n \geq m$ esempi (un solo esempio per ogni possibile valore è un po' poco...)
- Nel caso a sinistra, $m = 4$, per cui dovrei avere almeno $n \geq 4$ esempi di training

Home Country
USA
ITALY
ITALY
GERMANY
FRANCE

Vediamo il caso delle feature categoriche (che è più intuitivo)

- Se ho due feature, però, avrei bisogno di osservare almeno una volta tutte le possibili *combinazioni* dei loro valori
- Per cui, ho bisogno di almeno $n \geq m*m$ samples
- Se ho d feature avrei bisogno di m^d samples...
- Ciò vale anche per le feature numeriche...

Home Country	Job Salary
USA	LOW
ITALY	VERY LOW
ITALY	MODERATE
GERMANY	HIGH
FRANCE	HIGH

Curse of dimensionality

- Nel caso di una sola feature numerica, intuitivamente, gli esempi, per essere effettivi, dovrebbero «coprire» in maniera abbastanza uniforme tutto il suo intervallo di valori possibili
- Per cui, se suppongo che n esempi siano sufficienti con una sola feature, ne ho bisogno di n^2 per 2 feature
- E di n^d per d feature...
- Questo effetto è chiamato «curse of dimensionality» (la «maledizione della dimensionalità»)
- Per fortuna, empiricamente, il numero di esempi necessari di solito non cresce in maniera esponenziale col numero delle feature e, tipicamente, si assume una relazione lineare tra le due entità (ovvero il numero di esempi dovrebbe essere proporzionale al numero delle feature, i.e., $O(n*d)$)
- Nel deep learning (che vedremo alla fine del corso) questa relazione, a volte, riesce ad essere anche sub-lineare
- Rimane il fatto che, più feature ho, più esempi di training devo raccogliere. E, se ne ho troppo pochi, rischio l'overfitting...

Relazione tra overfitting e numero di feature

Da una parte, quindi, avere più feature aumenta la capacità descrittiva del mio modello (e.g., rappresentare un appartamento con tante feature lo descrive meglio)

Dall'altra, aumentare le feature aumenta il rischio di overfitting, a meno di non avere a disposizione un training set grandissimo

Perciò, devo trovare un compromesso tra aumentare l'espressività del modello (+ feature) e alleviare il rischio di overfitting (- feature)

Se le feature di un dato modello sono viste come parte integrante del modello stesso, e, quindi, come parte della sua "complessità", tutto ciò si riconduce al caso precedente: bisogna trovare un compromesso nella complessità totale di un modello per evitare sia l'underfitting che l'overfitting

Vedremo ora alcune tecniche di *feature selection* e di *regolarizzazione* che possono essere applicate sia alla regressione che alla classificazione e che ci aiutano nel cercare (automaticamente) questo compromesso

Feature selection

1. Filter Methods:

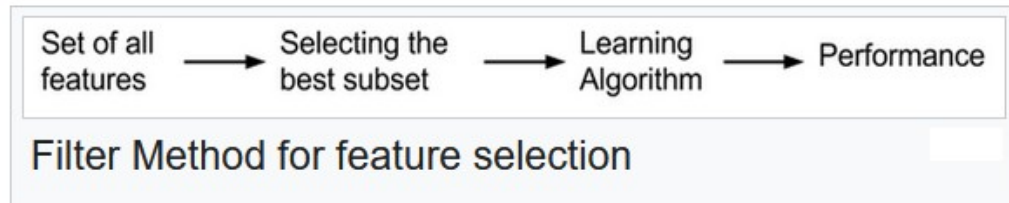
Indipendenti dal modello predittivo

2. Wrapper Methods:

Dipendenti dal modello predittivo

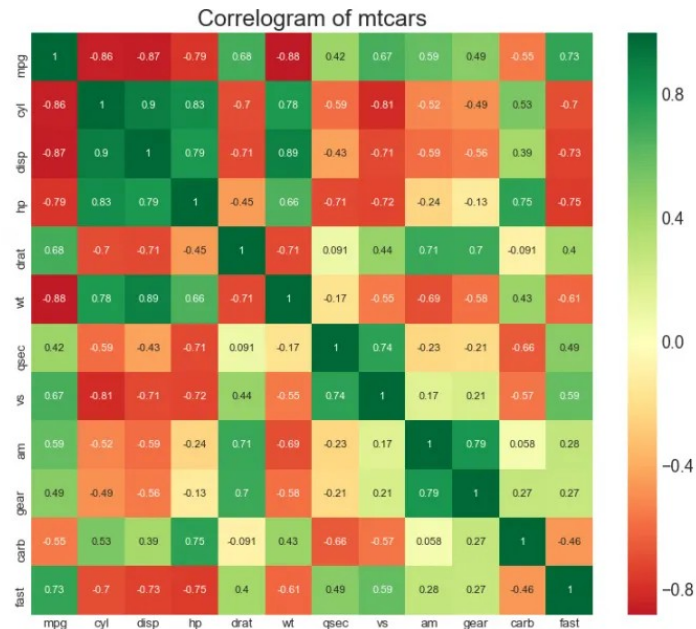
Filter Methods

La selezione avviene prima
del training

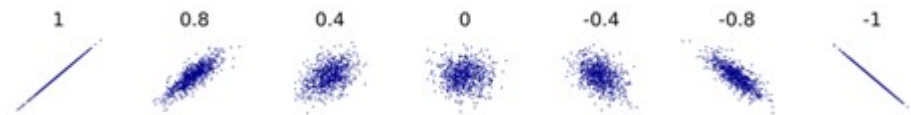


«Filtrare» le feature in base a proprietà statistiche come, ad esempio:

- Alta correlazione con la variabile target (feature da tenere)
- Alta correlazione con un'altra feature già selezionata (è una feature ridondante, da scartare)



La correlazione può essere sia positiva che negativa, per cui si prende il valore assoluto del coefficiente di correlazione



$$\rho_{X,Y} = \frac{\sum_{i=1}^n (x_i - \mu_x)(y_i - \mu_y)}{\sqrt{\sum_{i=1}^n (x_i - \mu_x)^2} \sqrt{\sum_{i=1}^n (y_i - \mu_y)^2}}$$

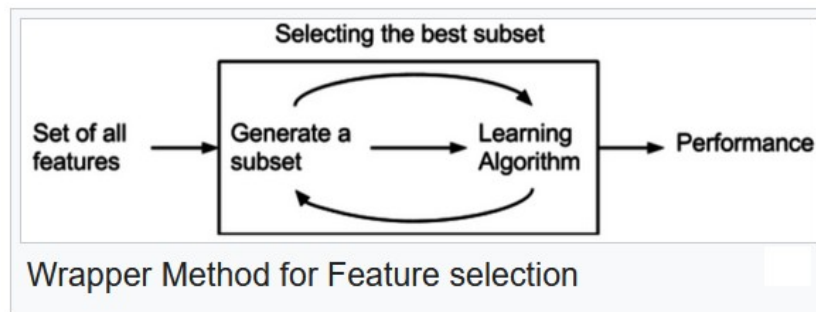
I Filter method sono veloci ma hanno dei problemi:

- Non è facile stabilire dei criteri universali per la selezione.
Ad esempio: quanto deve essere alto il valore (assoluto) di correlazione per accettare/scartare una feature?
- Le feature selezionate sono indipendenti dal modello/classe di modelli di predizione (regressione/classificazione) che userò dopo, per cui potrebbero non essere la scelta migliore per quel modello specifico
- Detto in altri termini, una specifica classe di modelli potrebbe sfruttare meglio alcune specifiche feature rispetto ad altre classi

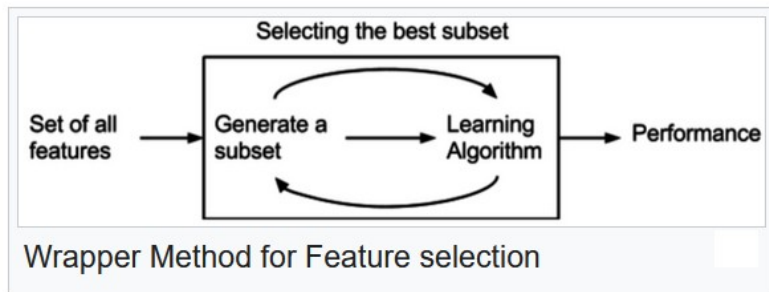
Wrapper Methods

La selezione viene fatta in maniera iterativa, ogni volta *riaddestrando il modello con l'insieme selezionato*:

1. Seleziono un sottoinsieme di feature
2. Addestro il modello (un'istanza di modello...) con quel sottoinsieme
3. Lo valuto sul validation set
4. Torno al passo 1...
5. Alla fine scelgo il modello con le prestazioni migliori e, *di conseguenza, il sottoinsieme di feature ad esso associato*



Wrapper Methods: ricerca esaustiva



Dato un insieme iniziale di feature F di cardinalità d :

1. Per ogni possibile sottoinsieme A di F :
 - a. Utilizzo le feature in A per addestrare un modello
 - b. Calcolo l'errore (e.g., usando MSE) *sul validation set*
 - c. Memorizzo quest'errore in una lista
2. Scelgo il modello corrispondente all'errore minimo

Problema: complessità $O(2^d)$

In questo algoritmo di ricerca *greedy*:

1. Inizio con sottoinsiemi di una sola feature e le provo tutte, una alla volta, addestrando un modello su ognuna di esse
2. Scelgo la feature che corrisponde al modello con la prestazione migliore (sul validation set). Supponiamo sia la feature x_i , corrispondente al modello M_1
3. A questo punto, provo con sottoinsiemi di feature di cardinalità 2
4. Ovvero provo a combinare x_i con tutte le altre $d-1$ feature, e, per ogni coppia di feature, addestro un modello
5. Come prima, scelgo la coppia (x_i, x_j) corrispondente al modello con prestazioni migliori (M_2)
6. Ora provo con sottoinsiemi di cardinalità 3, cercando “la compagna” di (x_i, x_j) , ovvero scegliendo una terza feature tra le $d-2$ rimaste. Sia x_k tale feature, corrispondente al modello M_3
7. Proseguo finchè non è rimasta nessuna feature da aggiungere

Alla fine di questo ciclo, mi ritroverò con d modelli corrispondenti a d possibili sottoinsiemi di feature:

- $x_{i'}$ associata al modello M_1
- $(x_{i'}, x_j)$ associate al modello M_2
- $(x_{i'}, x_{j'}, x_k)$ associate al modello M_3
- ...

L'ultimo passo è scegliere il modello tra M_1, \dots, M_d che ha ottenuto la prestazione migliore. Supponiamo che scelga M_p . Il sottoinsieme di feature associato ad M_p è il sottoinsieme "migliore" *rispetto alla classe di modelli che ho usato*

Il principio è simile a quello della Forward Selection, con la differenza che questa volta parto dall'insieme di tutte le d feature e vado a ritroso, eliminandone una alla volta

Regolarizzazione

Un altro modo per prevenire l'overfitting è la **regolarizzazione**

Ricordiamoci anzitutto che, nella linear regression, e in genere con modelli predittivi lineari, lo spazio dei parametri e quello delle feature sono in corrispondenza uno ad uno (tranne che per θ_0)

$$h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

$$\begin{aligned} x &= [x_0 = 1, x_1, \dots, x_{d-1}] \\ \theta &= [\theta_0, \theta_1, \dots, \theta_{d-1}] \end{aligned}$$

La regolarizzazione consiste nell'aggiungere una «penalità» alla loss function, ovvero un termine che favorisca valori dei parametri 'schiacciati' verso lo zero

In questo modo, le feature meno importanti saranno associate a pesi più piccoli, quindi avranno poca o nulla influenza all'interno della funzione ipotesi

Anche la regolarizzazione è una tecnica applicabile sia agli algoritmi di regressione che a quelli di classificazione

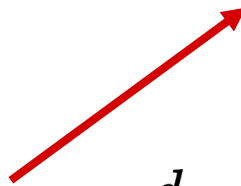
Ridge Regression

$$J(\theta) = \frac{1}{2} \sum_{i=1}^n [y^{(i)} - (\theta_0 + \sum_{j=1}^d \theta_j x_j^{(i)})]^2 + \frac{1}{2} \lambda \sum_{j=1}^d \theta_j^2$$

Nella **Ridge** Regression, la penalità è la norma (al quadrato) del vettore dei parametri
L'algoritmo di ottimizzazione (e.g., il GD) dovrà quindi trovare un «compromesso» tra minimizzare l'errore e minimizzare (la somma de-) i pesi

Finirà quindi per «sacrificare» i pesi corrispondenti alle feature meno importanti, portandoli vicini allo 0

Questo perché un coefficiente piccolo per una feature poco importante ha un impatto minore sull'errore di predizione


$$\|\theta\|_2^2 = \sum_{j=1}^d \theta_j^2$$

norma L_2 (al quadrato)

$$J(\theta) = \frac{1}{2} \sum_{i=1}^n [y^{(i)} - (\theta_0 + \sum_{j=1}^d \theta_j x_j^{(i)})]^2 + \frac{1}{2} \lambda \sum_{j=1}^d |\theta_j|$$

Nella Least Absolute Shrinkage and Selection Operator (**LASSO**) Regression, il meccanismo è simile, ma la penalità si basa su L_1

Può sorprendere il fatto che stia usando un termine non differenziabile nella loss function, ma, in realtà, il GD in alcuni casi (come questo) può essere adattato per funzioni obiettivo che siano «differenziabili»

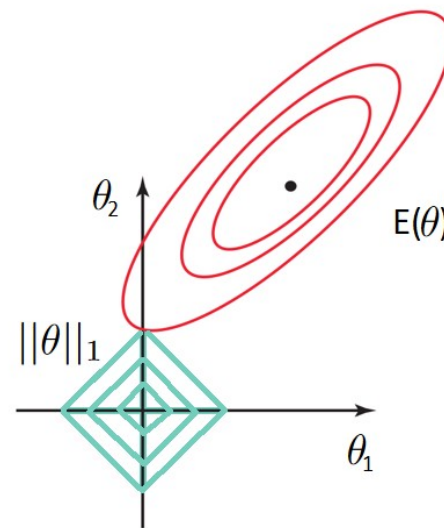
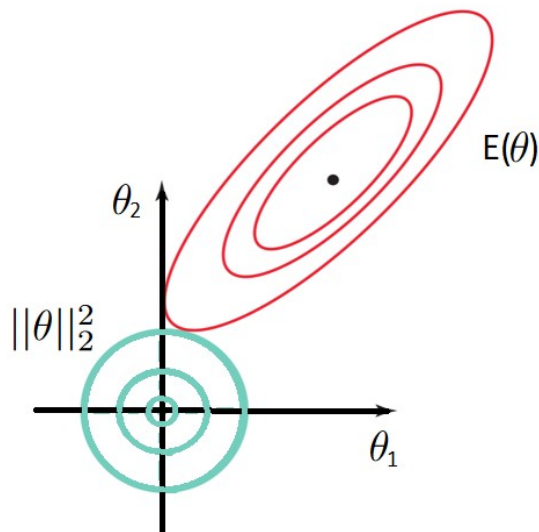

$$\|\theta\|_1 = \sum_{j=1}^d |\theta_j|$$

norma L_1

Ridge Regression e LASSO Regression

Ridge Regression: i pesi non vengono portati esattamente a zero

LASSO Regression: i pesi possono essere portati esattamente a zero



Funzione obiettivo: $J(\theta) = E(\theta) + ||\theta||_2^2$

2

Funzione obiettivo: $J(\theta) = E(\theta) + ||\theta||_1$

1

- **A tu per tu col Machine Learning. L'incredibile viaggio di un developer nel favoloso mondo della Data Science**, Alessandro Cucci, The Dot Company, 2017 [cap.5]
- **Pattern Classification, second edition**, R. O. Duda, P. E. Hart, D. G. Stork, Wiley-Interscience, 2000
- **An introduction to statistical learning**, Gareth M. James, Daniela Witten, Trevor Hastie, Robert Tibshirani, New York: Springer, 2013 [cap. 6]