Fondamenti di Machine Learning

Laurea Triennale in Ingegneria delle Comunicazioni

6: Data preprocessing, model selection, model evaluation

Lecturer: S. Scardapane

SAPIENZA
UNIVERSITÀ DI ROMA

Data preprocessing

Feature normalization

Consideriamo dei clienti di una banca, descritti da due feature:

- **Sesso**, codificato come $[0, 1]$ (maschio) o $[1, 0]$ (femmina).
- **Reddito** [$/mese], che varia da 0 a 10.000.

Ora consideriamo un cliente $\mathbf{x} \in \mathbb{R}^3$, e due altri clienti molto simili a $\mathbf{x}$ (ricordando che $\|\mathbf{x}\| = \sqrt{\sum_i x_i^2}$):

- $\mathbf{x}'$ è identico a $\mathbf{x}$, ma di sesso opposto. Abbiamo che $\|\mathbf{x}' - \mathbf{x}\| = \sqrt{2}$.
- $\mathbf{x}''$ è identico a $\mathbf{x}$, ma il suo reddito mensile è superiore di 100. Abbiamo che $\|\mathbf{x}'' - \mathbf{x}\| = 100$.

Qualsiasi algoritmo basato sulla distanza Euclidea tra i modelli (ad esempio, k-NN) darà 10 volte più importanza al reddito rispetto al sesso, *indipendentemente dalla sua effettiva importanza per il task di classificazione*.

In pratica, quasi qualsiasi algoritmo di machine learning non funzionerà correttamente se le feature sono misurate in unità con scale molto diverse.

Il processo di trasformazione dei dati (preprocessing) per fare sì che tutte le feature si trovino nello stesso range si chiama **normalizzazione delle feature** (**feature normalization**).

Nella **normalizzazione min-max**, ridimensioniamo ogni feature nell'intervallo $[0, 1]$:

$$x_i = \frac{x_i - x_{i,\min}}{x_{i,\max} - x_{i,\min}} .$$

dove $x_{i,\min}$ ($x_{i,\max}$) sono rispettivamente il valore minimo e massimo per la i-esima feature calcolati sull'intero dataset. Possiamo successivamente mappare su un intervallo generico $[\alpha, \beta]$ applicando una seconda trasformazione:

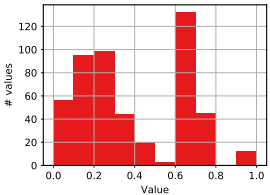
$$x_i = (\beta - \alpha)x_i + \alpha .$$

Un'altra tecnica molto usata è la **normalizzazione standard**:

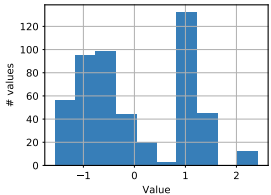
$$x_i = \frac{x_i - \mu_i}{\sigma_i^2}\,.$$

dove μ_i è la media empirica e σ_i^2 la varianza empirica, sempre calcolate sull'intero dataset.

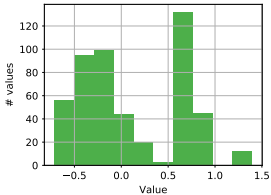
La maggior parte delle tecniche di normalizzazione può essere applicata escludendo i dati 'estremi' dal calcolo delle statistiche (ad esempio, utilizzando solo i dati tra il 15° e l'85° percentile) per fornire maggiore robustezza ai valori anomali. A volte, questo viene chiamato **normalizzazione robusta**.



(a) Normalizzazione min-max



(b) Normalizzazione standard



(c) Normalizzazione robusta

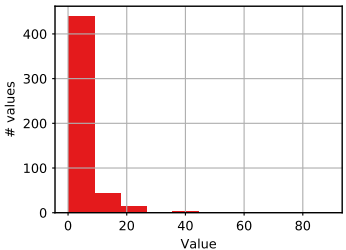
Figure 1: Un confronto tra diverse tecniche di normalizzazione delle feature sulla terza feature del dataset Boston. La distribuzione dei dati non viene mai influenzata. La normalizzazione robusta viene calcolata sull'intervallo interquartile (cioè, tra il 25° e il 75° percentile).

Molte caratteristiche nel mondo reale presentano distribuzioni fortemente asimmetriche con una coda molto lunga. Un esempio classico è il reddito di una persona.

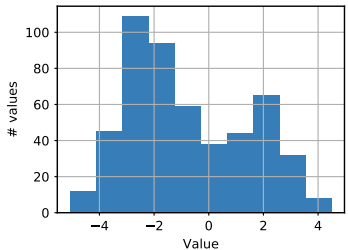
Una normalizzazione comune in questo caso è la **normalizzazione logaritmica**:

$$x_i = \log(x_i)\,.$$

La normalizzazione logaritmica è importante quando ci interessano solo i cambiamenti relativi in una quantità. Ad esempio, l'aumento relativo di $a+1$ rispetto a a è sempre $\log(1)$ utilizzando la normalizzazione logaritmica, indipendentemente dal valore di a.



(a) Distribuzione originale



(b) Dopo la normalizzazione logaritmica

Figure 2: Normalizzazione logaritmica applicata alla prima feature del dataset Boston (tasso di criminalità).

Data preprocessing

Imputazione dei dati mancanti

Un altro problema comune sono i **dati mancanti** (**missing feature**), in cui elementi della matrice di input $\mathbf{X}$ sono mancanti. La maggior parte dei metodi di apprendimento non è in grado di gestire i dati mancanti senza una forma di pre-processing.

Una strategia di base è rimuovere qualsiasi riga con una o più caratteristiche mancanti. Questo approccio può essere sub-ottimale per due motivi:

1. Se molte feature hanno valori mancanti, possiamo ottenere un dataset estremamente ridotto.
2. Se la mancanza di dati è correlata con qualche fattore sottostante (ad esempio, gli utenti maschi sono più riluttanti a condividere alcune informazioni), allora il dataset risultante sarà sbilanciato.

Un approccio efficace nella pratica è sostituire i dati mancanti con la media (o mediana) calcolata rispetto alla colonna corrispondente.

Per i problemi di classificazione, possiamo anche fare **imputazione condizionale dei dati**, dove la media (o mediana) è calcolata utilizzando solo i dati della classe corrispondente.

Se abbiamo conoscenze preliminari sul *come* mancano i dati, possiamo fare un'imputazione condizionale rispetto ad altre caratteristiche di input, ad esempio, calcolare la media rispetto alla nazionalità dell'utente.

- Se una variabile categorica ha valori mancanti, un approccio alternativo è aggiungere una nuova categoria corrispondente al caso mancante. Ad esempio, considerando una caratteristica $x = \{\text{maschio}, \text{femmina}\}$:

$$\text{male} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \ \text{female} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \ \text{missing value} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}.$$

- Lo stesso può essere fatto con una variabile continua; in quel caso, usiamo un valore predefinito per la feature quando manca e aggiungiamo una feature che è 1 se la feature manca, 0 altrimenti.

Supponiamo che solo la k-esima feature abbia valori mancanti. Possiamo modellare l'imputazione dei dati mancanti come un problema di apprendimento supervisionato:

$$f(\mathbf{x}_{-k}, y) = x_k \,,$$

dove $\mathbf{x}_{-k}$ denota l'insieme di tutte le feature escludendo la k-esima. L'imputazione della media e della mediana sono un caso speciale, dove l'output di f è costante (o costante a tratti per l'imputazione condizionale).

È raro ottenere buoni risultati con modelli eccessivamente complicati per f. La regressione lineare (o regressione logistica) sono scelte comuni.

Data preprocessing

Outlier detection

Un terzo problema comune è dato dagli **outlier**, cioè punti che non sono 'consistenti' rispetto al resto del dataset. Gli outlier possono essere dovuti a una vasta gamma di motivi:

- Rumore nel processo di raccolta dei dati.
- Vere anomalie, ad esempio, frodi in un dataset di carte di credito.

Se siamo interessati a rilevare questi punti anomali (ad esempio, per il rilevamento di frodi), il problema viene chiamato **rilevamento di anomalie** (**anomaly detection**) o **novelty detection**. Se desideriamo solo rimuovere gli outlier per migliorare l'accuratezza, abbiamo il **rilevamento degli outlier** (**outlier detection**).

Per problemi a bassa dimensionalità, una soluzione comune per il rilevamento di outlier è adattare una distribuzione parametrica nota (tipicamente una Gaussiana multivariata) e rimuovere i punti la cui distanza di Mahalanobis è maggiore di una soglia data.

La covarianza della Gaussiana è generalmente calcolata utilizzando tecniche che sono robuste agli outlier (ad esempio, stima della covarianza minima).

Le tecniche automatiche per il rilevamento degli outlier possono seriamente ostacolare il dataset se usate in modo improprio. Se gli outlier sono dovuti solo a una singola caratteristica, è più comune ispezionare visivamente il dataset utilizzando box-plot.

Consideriamo una Gaussiana 1D descritta da media μ e variance σ^2. Lo z-score:

$$z = \frac{x - \mu}{\sigma} \tag{1}$$

si interpreta come la distanza di x dal centro della Gaussiana in termini di deviazioni standard. Nel caso di una Gaussiana multivariata descritta da media $\mu \in \mathbb{R}^d$ e matrice di covarianza $\Sigma \in \mathbb{R}^{d \times d}$ questa si generalizza con la **distanza di Mahalanobis**:

$$d(\mathbf{x}) = \sqrt{(\mathbf{x} - \mu)\,\Sigma^{-1}\,(\mathbf{x} - \mu)^\top} \tag{2}$$

che rappresenta la distanza Euclidea in uno spazio ruotato per allinearsi agli assi principali della Gaussiana.

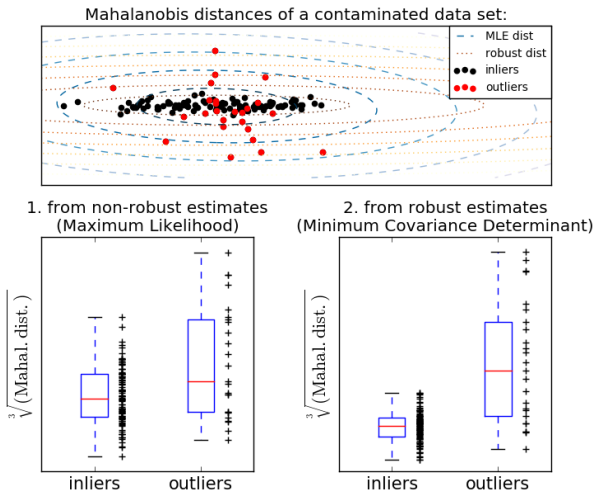


Figure 3: Esempio di rilevamento di outlier utilizzando involucri ellittici. (`http://scikit-learn.org/stable/auto_examples/covariance/plot_mahalanobis_distances.html`).

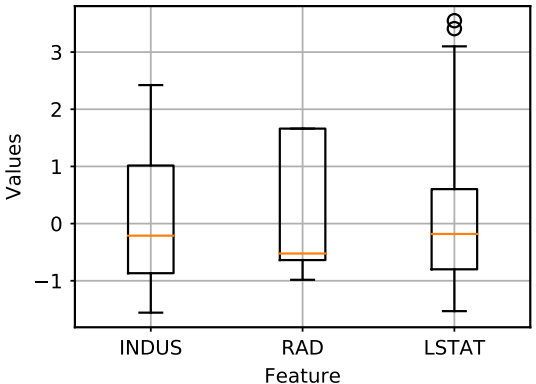


Figure 4: Un esempio di box-plot su tre feature del dataset Boston. La linea rossa è la mediana; il box copre l'intervallo interquartile (tra il 25° percentile e il 75° percentile). I punti al di fuori dei whiskers' sono considerati outlier.

Ci sono innumerevoli altre operazioni che possono essere applicate ai dati prima dell'apprendimento. Solo per citare alcuni esempi:

- **Generalizzazione**: sostituire un valore categorico con uno più generico (ad esempio, regione invece di codice postale).
- **Aggregazione**: sostituire una variabile continua raggruppando il suo valore in un insieme di possibili intervalli (questo può anche essere fatto sull'output per trasformare un problema di regressione in uno di classificazione).
- Costruire **caratteristiche polinomiali** dalle feature originali.

Selezione del modello

K-fold cross-validation

Fino ad ora, abbiamo utilizzato la procedura di holdout per stimare il possibile overfitting. Un singolo holdout può essere fuorviante, in particolare per dataset di medie dimensioni (alta varianza).

Un modo più comune per valutare i modelli è eseguire una **k-fold cross-validation**:

1. Dividiamo il dataset in k parti di uguale dimensione $\mathcal{S}_1, \ldots, \mathcal{S}_k$.
2. Alleniamo su $\mathcal{S}_2, \ldots, \mathcal{S}_k$, e testiamo su $\mathcal{S}_1$. Otteniamo un errore E_1.
3. Alleniamo su $\mathcal{S}_1, \mathcal{S}_3, \ldots, \mathcal{S}_k$, e testiamo su $\mathcal{S}_2$. Ottieniamo un errore E_2.
4. ...
5. Alleniamo su $\mathcal{S}_1, \ldots, \mathcal{S}_{k-1}$, e testiamo su $\mathcal{S}_k$. Otteniamo un errore E_k.
6. Calcoliamo l'accuratezza finale come $E = \frac{1}{k} \sum_{i=1}^{k} E_i$.

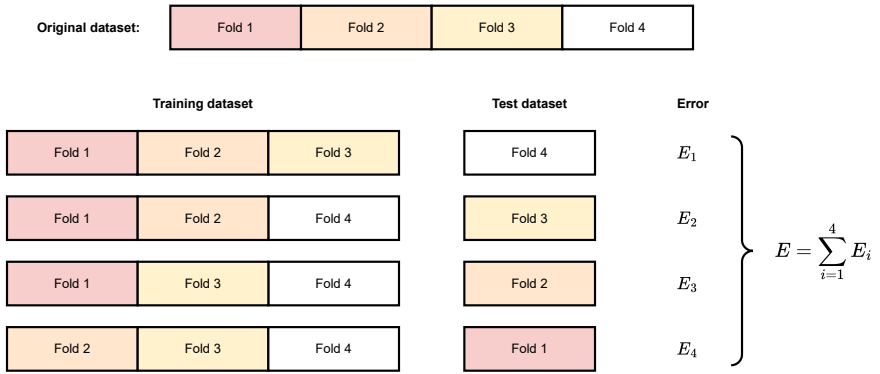


Figure 5: Esempio di 4-fold cross-validation.

In pratica il dataset è composto dalle matrice **X** e **y**, ed ogni fold corrisponde a prendere un sottoinsieme delle loro righe.

Al termine della *k*-fold cross-validation abbiamo allenato *k* modelli diversi: in questo caso è comune allenare un ultimo modello sull'intero dataset da usare in produzione, usando *E* come stima della sua accuracy.

Una *k*-fold cross-validation dove *k* è uguale alla dimensione del dataset viene detta **leave-one-out** (LOO) cross-validation. Maggiore *k*, migliore la stima dell'errore ma maggiore il costo computazionale.

Selezione del modello

Model selection

La maggior parte dei metodi di apprendimento richiede all'utente di selezionare uno o più iper-parametri, es.:

- Il parametro k nel k-NN.
- Il learning rate nella discesa al gradiente.

Supponiamo di provare diverse possibilità per ciascun iper-parametro. Come scegliamo quale sia il migliore? Sappiamo già che l'errore di allenamento è un cattivo surrogato per la capacità di generalizzazione di un modello di apprendimento. Di fatto, questi due problemi sono quasi equivalenti:

- Valutazione del modello: valutare l'errore atteso di un modello.
- Selezione del modello: valutare l'errore atteso di due modelli per selezionare il migliore.

Possiamo risolvere il problema di selezione del modello in modo equivalente al problema di valutazione del modello, ad esempio, usando un set di holdout:

1. Conserviamo una parte del dataset di allenamento (chiamato **dataset di validazione**).
2. Alleniamo diversi modelli utilizzando la parte rimanente, ognuno con una diversa scelta di iper-parametri.
3. Selezioniamo l'iper-parametro con l'errore più basso sul dataset di validazione e ri-alleniamo il modello utilizzando l'intero dataset.

Si può equivalentemente utilizzare holdout o k-fold cross-validation, sia per il testing che per la validazione.

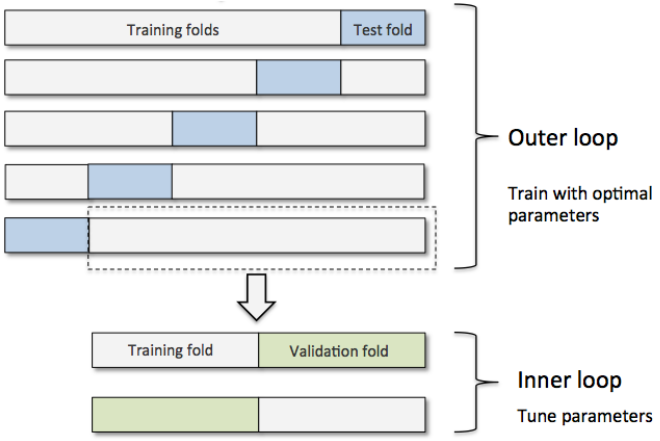


Figure 6: https://sebastianraschka.com/faq/docs/evaluate-a-model.html.

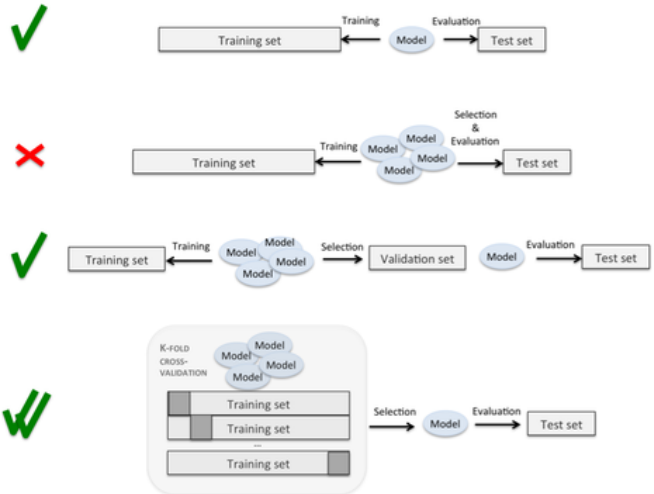


Figure 7:
https://sebastianraschka.com/faq/docs/evaluate-a-model.html.

Per i modelli con uno/due iper-parametri, il modo più comune per ottimizzarli è una **ricerca a griglia**, dove specificiamo un elenco di possibili configurazioni e cerchiamo in modo esaustivo quella migliore.

Ad esempio, possiamo ottimizzare k-NN cercando k in 5, 10, . . ., 50. Alcuni parametri sono comunemente ottimizzati in *spazio logaritmico*, poiché assumiamo che piccoli cambiamenti non siano significativi.

Possiamo anche ripetere diverse procedure di ricerca a griglia a diversi livelli di risoluzione, concentrandoci attorno all'ottimo della ricerca a griglia più grossolana.

Un'altra scelta popolare è la **ricerca casuale**:

1. Specifichiamo una distribuzione di campionamento per ogni parametro, ad esempio una distribuzione uniforme in $[-10, 10]$ per $\log(C)$.
2. Ad ogni iterazione, campioniamo casualmente una combinazione di parametri e ottimizziamo un modello fino a un certo budget di tempo computazionale.
3. Concludiamo la ricerca dopo un dato numero di iterazioni.

Sia la ricerca a griglia che la ricerca casuale possano essere parallelizzate in modo banale addestrando e valutando diversi modelli contemporaneamente in un'architettura parallela.

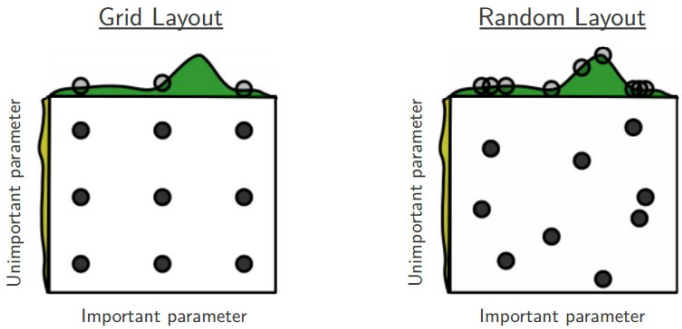


Figure 8: Bergstra, J. e Bengio, Y., 2012. **Random search for hyper-parameter optimization**. *Journal of Machine Learning Research*, 13(Feb), pp. 281-305.

Ottimizzare i parametri può essere visto come un problema di ottimizzazione; indicando con Θ l'insieme degli iper-parametri, vogliamo ottimizzare:

$$\Theta^* = \arg\min_{\Theta} \text{Err}(\Theta)\,,$$

dove $\text{Err}(\Theta)$ può essere, ad esempio, un errore validato a 5-fold sul dataset di allenamento.

I problemi sono che (i) generalmente non possiamo calcolare alcuna derivata di $\text{Err}(\cdot)$; (ii) anche una singola valutazione della funzione obiettivo può essere costosa. **L'ottimizzazione Bayesiana** può essere utilizzata come tecnica di black-box a questo scopo.

Selezione del modello

Metriche per la classificazione

Una volta che un modello è stato allenato, dobbiamo valutarne le prestazioni. Riportare l'MSE o l'accuratezza della classificazione sul set di test è una buona misura, ma potrebbe non fornire tutte le informazioni necessarie.

Come esempio concreto, consideriamo un problema di classificazione binaria, dove il 99% dei dati appartiene alla classe 0. In questo caso, un classificatore che prevede sempre 0 come output avrebbe un'accuratezza del 99%! Questo è un esempio di dataset **sbilanciato**.

Per la classificazione binaria, gli output del classificatore possono essere raggruppati nella cosiddetta **matrice di confusione**:

	Classe reale: 0	**Classe reale:** 1
Predizione: 0	TN: True negative	FN: False negative (Errore di tipo II)
Predizione: 1	FP: False positive (Errore di tipo I)	TP: True positive

Questo è facilmente esteso al caso multiclasse.

Dai valori della matrice di confusione, possiamo definire diverse nuove metriche di accuratezza:

$$\text{Precisione} = \frac{\text{TP}}{\text{FP} + \text{TP}} \tag{3}$$

$$\text{Recall (o true positive rate, TPR)} = \frac{\text{TP}}{\text{FN} + \text{TP}} \tag{4}$$

$$\text{False positive rate}(\mathit{FPR}) = \frac{\text{FP}}{\text{FP} + \text{TN}} \tag{5}$$

L'F1 score è definito come la media armonica di precisione e recall, ed è particolarmente utile in scenari sbilanciati:

$$F_1 = 2 \cdot \frac{1}{\frac{1}{\text{Precisione}} + \frac{1}{\text{Recall}}} = 2 \cdot \frac{\text{Precisione} \cdot \text{Recall}}{\text{Precisione} + \text{Recall}}. \tag{6}$$

Intutivamente, per avere un buon F1-score, un modello deve ottimizzare allo stesso tempo sia la precision che il recall.

Nella pratica, possiamo ottenere informazioni più dettagliate sul processo di classificazione ad un livello di granularità ancora più basso, ricordando che gli output di un classificatore sono quasi sempre numeri reali, che binarizziamo per ottenere le classi finali.

Fino ad ora, abbiamo considerato il caso semplice dove arrotondiamo questi output all'intero più vicino (quando $y \in \{0, 1\}$), o prendiamo il segno (quando $y \in \{-1, 1\}$). Variando effettivamente la soglia per la binarizzazione, possiamo ottenere diversi valori di recall e FPR:

- Per una soglia molto bassa, otteniamo un recall perfetto (tutti i modelli sono classificati come positivi), ma un FPR molto alto.
- Al contrario, per una soglia molto alta, otteniamo un cattivo recall con un FPR estremamente piccolo.

Tracciando il recall contro il FPR per diverse scelte della soglia, otteniamo la cosiddetta **receiver operating curve** (ROC). L'area sotto la curva ROC (AUC) è un'altra misura molto comune di performance per un classificatore binario.

Possiamo confrontare due classificatori tracciando le loro curve ROC per ottenere un set più ampio di intuizioni sul loro comportamento. Un'altra curva correlata traccia la precisione contro il recall (curva PR).

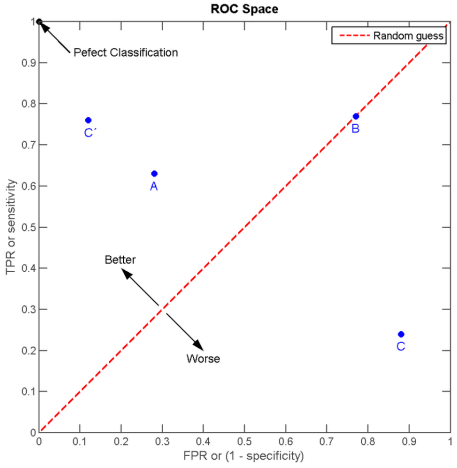


Figure 9: Se consideriamo solo una singola soglia, ogni classificatore è un punto nello spazio ROC [Wikipedia, en: Receiver Operating Characteristic].

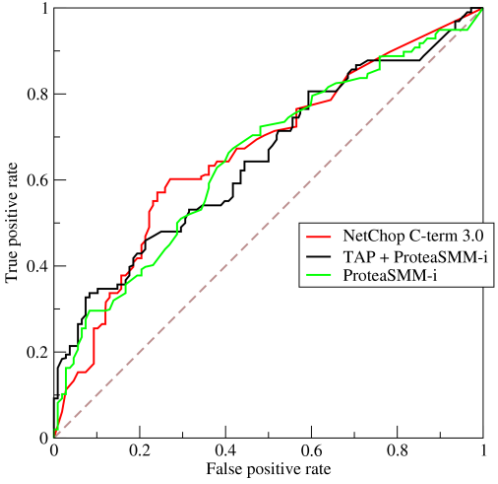


Figure 10: In generale, possiamo tracciare un'intera curva nello spazio ROC [Wikipedia, en: Receiver Operating Characteristic].

Selezione del modello

Teoria delle decisioni

Selezionare una soglia per la classificazione sulla base di un trade-off tra precision e recall è un esempio di **decision theory**.

Il modello ci aiuta ad assegnare probabilità a vari eventi (es., le classi a cui assegnare x) ma l'effettiva decisione su quale classe scegliere può dipendere da un insieme di fattori più ampio. In questa sezione vediamo un esempio più completo basato sull'assegnazione di *costi* a ciascun tipo di errore.

Supponiamo di assegnare un *costo* a ciascun elemento della matrice di confusione. Ad esempio, un falso positivo può essere valutato 10 volte peggio di un falso negativo:

	Classe 1	Classe 2
Predizione: 1	0	10
Predizione: 2	1	0

Idealmente, vorremmo predire la classe 1 solo se *realmente* confidenti di quella predizione.

Supponiamo che ci siano K classi. Indichiamo con C_{ij} il costo di assegnare un elemento di classe i alla classe j (come nella tabella sopra). $f_i(x)$ è la probabilità assegnata alla classe i dal nostro modello.

La scelta ottimale in questo caso non è altro che la classe che minimizza il costo medio:

$$c = \arg\min_i \sum_{j=1}^{K} C_{ij} f_j(x) \tag{7}$$

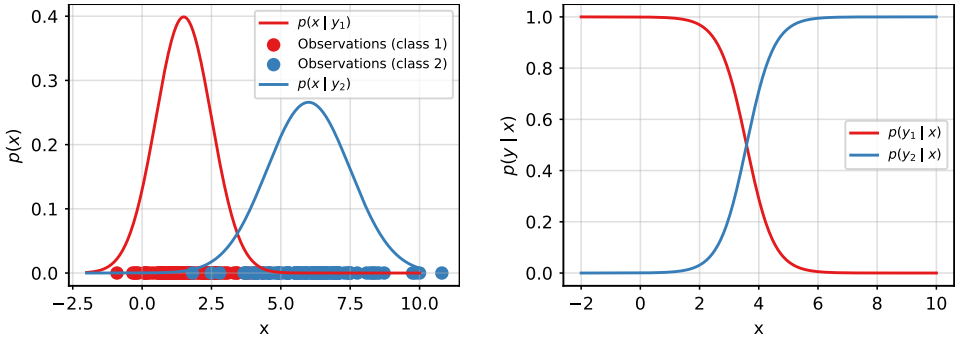


Figure 11: Sinistra: dataset sintetico 1D. I dati per ciascuna classe sono generati da due Gaussiane (in rosso e blu). Destra: le probabilità assegnate da un modello di regressione logistica alle due classi.

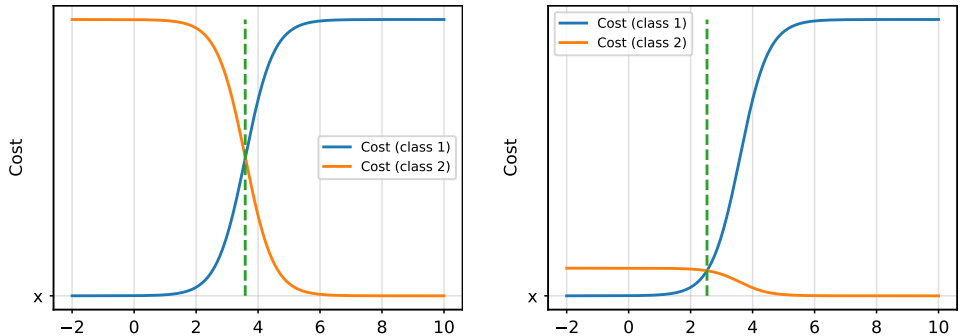


Figure 12: Sinistra: le predizioni errate hanno costi uguali. I costi delle classi sono uguali alle probabilità assegnate dal modello. Destra: i falsi positivi 'costano' 10 volte di più. In entrambi i casi scegliamo la classe con costo minore: la differenza sostanziale è nella regione 'indecisa'.