

APPRENDIMENTO AUTOMATICO

Alessandro Zappatore

Università degli studi di Torino
Anno accademico 2025/2026, 1° semestre

1 Introduzione

1.1 Definizione di Machine Learning (ML)

Thomas Mitchell; 1996; Machine Learning

A computer program is said to learn from experience with respect to some class of tasks and performance measure , if its performance at tasks in , as measured by , improves with experience.

È la combinazione di tre oggetti l'**esperienza E** sotto forma di esempi del problema risolto al computer, un **task T** da risolvere (classificazione degli esempi in un certo numero di categorie, assegnazione di un numero associato agli esempi, riconoscimento di un'immagine), sotto una **misura di performance P** un valore per misurare quanto sta andando bene.

Il programma del computer fa apprendimento automatico basandosi sugli esempi se le sue performance migliorano man mano che vede gli esempi.

Esempio del melone Assumendo di avere dei dati sulla maturazione dei meloni vogliamo sapere se il frutto è maturo o no.

ID	Color	Root	Sound	Ripe
1	green	curly	muffled	true
2	dark	curly	muffled	true
3	green	straight	crisp	false
4	dark	slightly curly	dull	false

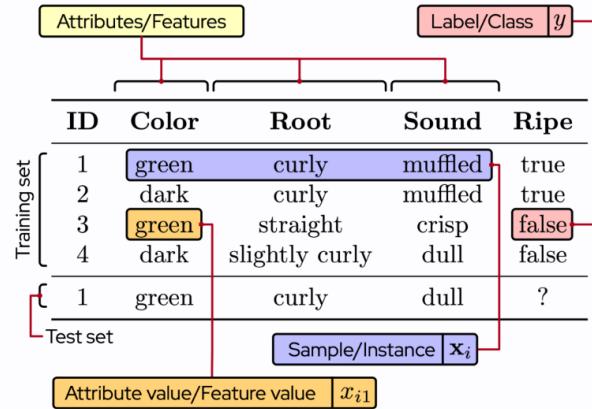
Ora troviamo un melone con le seguenti caratteristiche e vogliamo sapere se è maturo:

Color	Root	Sound
green	curly	dull

In questo caso abbiamo un **task**, sapere se è maturo, delle **performance**, quanto è accurata la nostra predizione e dell'**esperienza** i meloni comprati in precedenza.

Questo tipo di lavoro è chiamato **apprendimento supervisionato**, dove un essere umano andrà a segnare qual è la risposta giusta. Nell'**apprendimento non supervisionato** la colonna delle risponde non è presente, ma si andrà a raggruppare risposte simili.

1.2 Terminologia

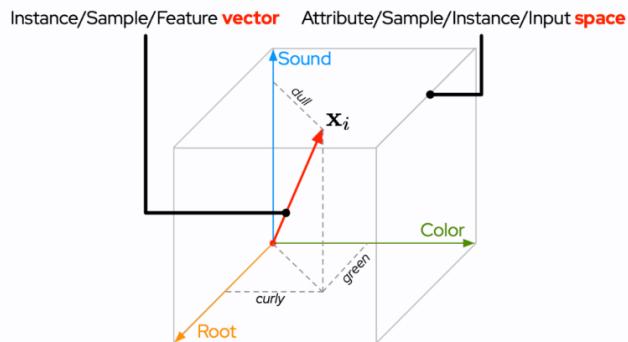


Le colonne della tabella sono chiamate **attributi** o **features** dell'esempio, l'elemento che ci dice come risolvere il task è chiamato **etichetta** o **classe** dell'esempio, tipicamente la classe viene denotata con una y_i minuscola.

Una riga della tabella, senza considerare la colonna della classe, è chiamato **sample** o **istanza** dell'esempio, in genere denotata con una X_i per dire che è l'iesimo esempio. [Tutte le volte che parleremo di vettori saranno rappresentati da lettere maiuscole, invece i valori scalari saranno rappresentati in corsivo]

Il singolo valore di una colonna è chiamato **attribute value** o **feature value** ed è rappresentato da un valore in corsivo x_{i1} in questo caso la prima feature dell'iesimo esempio.

L'insieme degli esempi che usiamo per indurre la regola che poi utilizzeremo per classificare è chiamato **set di training** e spesso al fianco si trova un altro insieme di esempio, chiamato **test set**, che serve per riuscire a capire quanto bene stiamo performingo.



Quanto gli esempi vengono immaginati in uno spazio euclideo di qualche tipo, in questo caso a tre dimensioni, spesso ci si riferisce all'esempio come **instance vector** oppure **sample vector**, **feature vector**, l'intero insieme di tutti i possibili esempi è chiamato anche **attribute space**, **sample space**, **instance space**, **input space**.

1.3 Notazione

Symbol	Meaning
\mathcal{X}	instance space, in many cases $\mathcal{X} = \mathbb{R}^d$
\mathcal{Y}	label space, e.g., $\mathcal{Y} = \{0, 1\}$ for binary classification
$\mathbf{x}_i \in \mathcal{X}$	i -th instance/sample, $\mathbf{x}_i = [x_{i1}, x_{i2}, \dots, x_{id}]^\top$
d	dimensionality of the instance space
$y_i \in \mathcal{Y}$	label of the i -th instance, e.g., $y_i = 1$ if the watermelon is ripe, $y_i = 0$ otherwise
D	Dataset, $D = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_n, y_n)\}$
n	number of instances in the dataset
h	model/hypothesis, a function $h : \mathcal{X} \mapsto \mathcal{Y}$ that maps instances to labels
\mathcal{L}	Learning algorithm $\mathcal{L} : \mathcal{P}(\mathcal{X} \times \mathcal{Y}) \rightarrow \mathcal{H}$, where \mathcal{P} is the power set and \mathcal{H} is the hypothesis space
\mathbb{I}	Indicator function, $\mathbb{I}(P)$ (sometime denoted as $\mathbb{I}[P]$) is 1 if P is true, 0 otherwise

1.4 Altra terminologia

Il **training** è un processo utilizzato da algoritmo di machine learning per costruire un modello dai dati. I dati utilizzati durante questo processo sono chiamati **training data**, dove ogni esempio è chiamato **training example** e l'insieme di tutti gli esempi è chiamato **training set**.

Obiettivo del Machine Learning

L'obiettivo dell'apprendimento automatico è trovare un **ipotesi** che è capace di approssimare la **ground-truth** su dati che non ha mai visto.

1.5 Tasks

I più importanti tasks che si possono fare con l'apprendimento automatico sono:

- **Predittivi:** predire un numero o una caratteristica, solitamente sono supervisionati
 - **Classificazione**, abbiamo un certo numero di etichette negli esempi, se le etichette sono solo due si parla di classificazione **binaria** invece se sono di più si parla di classificazione **multiclasse**.
 - **Regressione**, è un caso di predizione dove l'insieme delle etichette corrisponde con un intervallo continuo, cerchiamo di predire dei numeri.
- **Descrittivi:** descrivere qualcosa su come è strutturato il data set, solitamente non supervisionati
 - **Clustering**
 - **Regole di associazione**

Learning supervisionato l'etichetta è fornita dall'utente

Learning non supervisionato l'etichetta non esiste

1.6 Assunzione i.i.d

Nella maggioranza dei casi, in particolare in tutti gli algoritmi che vedremo nel corso, si assume che i dati siano **Indipendenti e Identicamente Distribuiti**.

Nel momento in cui ho estratto un istanza con la sua etichetta il prossimo lo estraggo dalla stessa distribuzione di probabilità, perché se cambia la distribuzione non ho speranza di apprendere nulla.

Indipendenti perché fissata la distribuzione, se io pescò due esempi, l'aver estratto un esempio non ha implicazione sull'estrazione di un altro esempio. [Ex. estraggo una pallina dall'urna, poi la rimetto dentro e riestraggo. La probabilità del primo e del secondo caso sarà la medesima]

- **Indipendenza:** $P((X_i, y_i) | (X_1, y_1), \dots, (X_{i-1}, y_{i-1})) = P((X_i, y_i)) \quad \forall i$
- **Identicamente distribuiti:** Tutte le istanze provengono dalla stessa distribuzione sottostante. Ciò significa anche che i dati di training e i dati di test devono provenire dalla stessa distribuzione.

Esempio

Immaginando di gestire una fabbrica di cupcake. Ogni cupcake è fatto:

- usando la **stessa ricetta** (identicamente distribuita)
- e **cotti in uno stampo separato** senza influenzare gli altri (indipendenti)

Quindi, se assaggi 10 cupcake a caso e hanno tutti un sapore dolce, puoi presumere che anche i cupcake futuri saranno dolci, perché il tuo campione è i.i.d.

Esempio - What if

Cosa succede se:

- All'improvviso usi **due ricette diverse** → non distribuite in modo identico.
- Cuoci i cupcake nello stesso stampo e l'impasto si mescola → non indipendente.

1.7 Spazio delle ipotesi

1.7.1 Deduzione Vs Induzione

Deduzione vuol dire che parto da degli assiomi, ho delle regole per combinare gli assiomi e ottengo una deduzione. La deduzione è un processo **corretto**, se assumo che gli assiomi siano corretti allora qualsiasi cosa io deduca da questi assiomi deve essere vera per forza.

L'Induzione è il processo inverso, si parte da oggetti specifici, gli esempi, e cerco di dedurre una regola generale.

Il processo di Apprendimento Automatico si basa interamente sul principio dell'induzione.

Esempio

Esempio di deduzione:

- Tutti gli umani sono mortali
- Socrate è un uomo
- Quindi, Socrate è mortale

Esempio di induzione:

- Dopo aver osservato che ogni essere umano conosciuto prima o poi muore, potremmo dedurre che tutti gli esseri umani sono mortali.

1.7.2 Symbolic concept learning

Con i metodi induttivi vogliamo provare a costruire delle regole logiche, e in questo caso ci troviamo nel campo del **symbolic concept learning** e una sua forma più estesa è chiamata **Inductive Logic Programming**, dei linguaggi che ci permettono di lavorare con la logica per costruire queste regole in modo induttivo.

La forma base del concept learning è chiamato **boolean concept learning** dove l'obiettivo è costruire una funzione $h : \chi \mapsto \{0, 1\}$.

Esempio

Supponendo di voler predire se un melone è maturo e la formula è:

$$\text{ripe} \iff (\text{color} = ?) \wedge (\text{root} = ?) \wedge (\text{sound} = ?)$$

Al posto dei punti interrogativi possiamo mettere un valore oppure un **asterisco *** per dire che **non interessa il valore specifico**.

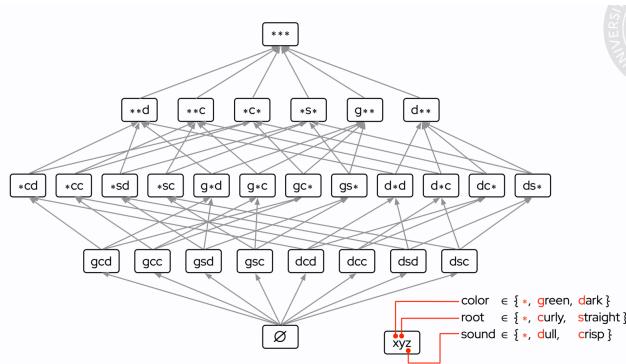
In questo caso il melone sarà maturo se il suono è dull il resto non mi interessa.

$$\text{ripe} \iff (\text{color} = *) \wedge (\text{root} = *) \wedge (\text{sound} = \text{dull})$$

1.7.3 Spazio delle ipotesi

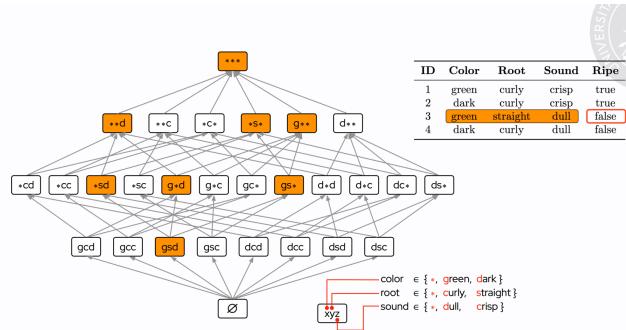
Lo **Spazio delle ipotesi** è l'insieme di tutte le ipotesi (tutti i possibili modelli) che posso apprendere utilizzando uno specifico algoritmo di apprendimento. [Nell'esempio sopra citato è l'insieme di tutte le formule booleane che posso ottenere].

Possiamo pensare all'apprendimento automatico come un **processo di ricerca all'interno dello spazio delle ipotesi**, e devo trovare l'ipotesi migliore in base ai dati che ho.

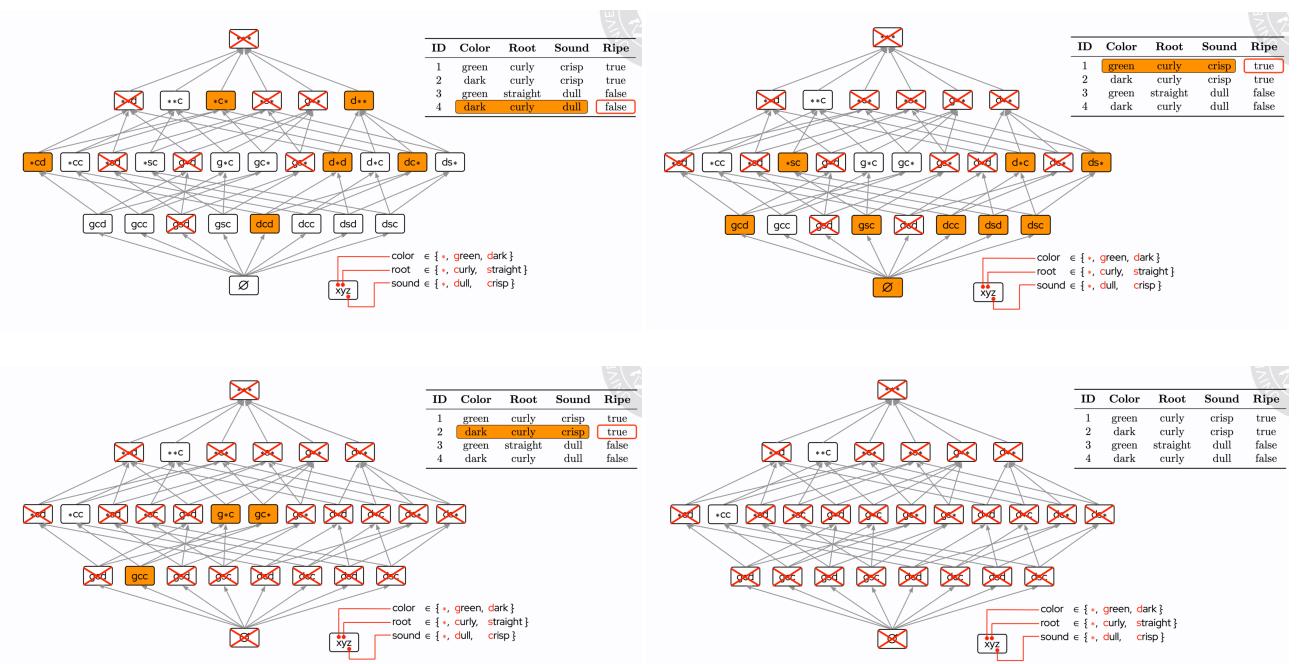


1.7.4 Spazio delle versioni

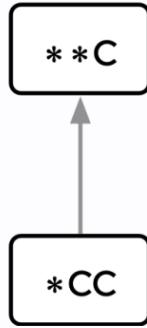
Lo **spazio delle versioni** è un **sottoinsieme dello spazio delle ipotesi** che è consistente con tutti gli **esempi di training**. Sostanzialmente prendo solo gli esempi che danno una risposta giusta [nel caso dei meloni se trovo un esempio con suono *dull* ma non maturo lo andrò a scartare].



L'ipotesi in arancione non è consistente con l'esempio, perché questo avrebbe detto a qualsiasi cosa mature. **d dice che dovrebbe dare maturo e anche questo non va bene. Tutte le caselle evidenziate in arancione non sono consistenti con l'esempio in questione, quindi decido di eliminarle.



Vado a controllare tutti gli esempi e alla fine andrò a vedere che lo spazio delle versioni finale sarà:



Nell'esempio lo spazio delle versioni consiste in due ipotesi:

- $\text{ripe} \iff (\text{color} = *) \wedge (\text{root} = *) \wedge (\text{sound} = \text{crisp})$
- $\text{ripe} \iff (\text{color} = *) \wedge (\text{root} = \text{curly}) \wedge (\text{sound} = \text{crisp})$

Dato quello che sappiamo del dataset le due sono equivalenti e non c'è una giusta.

1.7.5 Bias induttivo

Nel caso dovesse arrivare un nuovo esempio del tipo $\text{Color} = \text{green}$, $\text{Root} = \text{straight}$, $\text{Sound} = \text{crisp}$, avremmo una regola che dice maturo e una che dice non maturo.

Senza altre informazioni non possiamo decidere quale delle due ipotesi è corretta, però un algoritmo non può restituire un insieme di ipotesi.

Tutti gli algoritmi di apprendimento hanno un **bias induttivo**, non basato sui dati.

Si può scegliere di preferire delle regole più generali o all'opposto potrebbe preferire una regola più specifica.

Ogni algoritmo di apprendimento DEVE avere un bias induttivo, altrimenti non potrebbe concludere l'apprendimento. Potrei anche scegliere una risposta a caso ma non sarebbe una soluzione furba da prendere.

Occam's Razor è un principio che a parità di ipotesi consistenti devo scegliere quella più semplice come bias induttivo.

Semplice può avere molti significati diversi, in genere la più utilizzata è quella che fa meno assunzioni sui dati.

No Free Lunch Theorem in modo informale dice che nessun algoritmo di apprendimento universalmente migliore di altri quando le sue performance sono misurate su tutti i possibili mondi.

1.7.6 No Free Lunch Theorem

Denotiamo con:

- $P(h | X, \mathcal{L}_a)$: **probabilità** che l'algoritmo \mathcal{L}_a restituisca l'ipotesi h dati i dati X
 - h è l'ipotesi, quello che stiamo imparando
 - X è il training set
 - \mathcal{L}_a è l'algoritmo di apprendimento
- f è la **ground truth** il modello vero sottostante che vogliamo apprendere
- l'errore **out-of-sample** dell'ipotesi h , denotato con E_{ote} , è la media dell'errore su tutti gli elementi che non sono presenti nel training set, che sono commessi dai modelli appresi tramite \mathcal{L}_a sul training set X .

$$E_{ote}(\mathcal{L}_a | X, f) = \sum_h \sum_{x \in \chi - X} P(X) \mathbb{I}(h(X) \neq f(X)) P(h | X, \mathcal{L}_a)$$

Faccio la somma su tutte le possibili ipotesi, dopo di che sommo su tutti gli esempi che non sono nel training set ottenuto da \mathcal{X} (spazio di tutti i possibili esempi) togliendo gli elementi del training set. $P(X)$ è la probabilità di estrarre uno specifico esempio per la funzione indicatrice $\mathbb{I}(h(X) \neq f(X))$ (restituisce 1 se l'espressione è vera e il modello sta sbagliando a predire l'etichetta di X e 0 altrimenti) tutto per la probabilità di h su tutti i modelli.

Dimostrazione

Vogliamo dimostrare che l'errore non dipende da \mathcal{L}_a , quindi un algoritmo vale l'altro in media.

$$\begin{aligned} \sum_f E_{ote}(\mathcal{L}_a | X, f) &= \sum_f \sum_h \sum_{\mathbf{x} \in \mathcal{X} - X} P(\mathbf{x}) \mathbb{I}(h(\mathbf{x}) \neq f(\mathbf{x})) P(h | X, \mathcal{L}_a) \\ &= \sum_{\mathbf{x} \in \mathcal{X} - X} P(\mathbf{x}) \sum_h P(h | X, \mathcal{L}_a) \sum_f \mathbb{I}(h(\mathbf{x}) \neq f(\mathbf{x})) \\ &= \sum_{\mathbf{x} \in \mathcal{X} - X} P(\mathbf{x}) \sum_h P(h | X, \mathcal{L}_a) \frac{1}{2} 2^{|\mathcal{X}|} \\ &= \frac{1}{2} 2^{|\mathcal{X}|} \sum_{\mathbf{x} \in \mathcal{X} - X} P(\mathbf{x}) \sum_h P(h | X, \mathcal{L}_a) \\ &= \frac{1}{2} 2^{|\mathcal{X}|} \sum_{\mathbf{x} \in \mathcal{X} - X} P(\mathbf{x}) \cdot 1 \end{aligned}$$

Nella diapositiva precedente, il termine $\frac{1}{2} 2^{|\mathcal{X}|}$ è il risultato della somma degli errori di tutte le possibili ipotesi binarie. Dato uno spazio campionario χ , esistono $2^{|\chi|}$ modi per etichettare gli esempi e, quindi, $2^{|\chi|}$ possibili ipotesi distinte. Se fissiamo un esempio X e un'ipotesi h , metà di queste ipotesi sarà in disaccordo con h su X , mentre l'altra metà sarà in accordo.

Esempio

Consideriamo $\chi = \{X_1, X_2, X_3\}$, ci concentriamo su un esempio randomico (facciamo X_2), e assumiamo $h(X_2) = 0$, le funzioni in disaccordo con h su X_2 sono:

Function	$f(\mathbf{x}_1)$	$f(\mathbf{x}_2)$	$f(\mathbf{x}_3)$
f_1	0	0	0
f_2	0	0	1
f_3	0	1	0
f_4	0	1	1
f_5	1	0	0
f_6	1	0	1
f_7	1	1	0
f_8	1	1	1

In generale, per qualsiasi X ed etichetta di h , metà delle $2^{|\chi|}$ funzioni sono in disaccordo con h .

Quindi abbiamo dimostrato che:

$$\sum_f E_{ote}(\mathcal{L}_a | X, f) = \frac{1}{2} 2^{|\chi|} \sum_{X \in \chi - X} P(X)$$

Osservazione Il teorema assume che si possa mediare su tutte le possibili f ma nel mondo non tutte le ipotesi sono ugualmente importanti. Questo è un risultato interessante, perché dimostra che **l'errore medio fuori campione** di un algoritmo di apprendimento è **indipendente dall'algoritmo stesso**. Ciò significa che **nessun algoritmo di apprendimento è migliore di un altro** quando si calcola la media su tutti i possibili problemi.

2 Selezione e valutazione di modelli

2.1 Accuracy e Error rate

Il modo più semplice per misurare la bontà di un modello è quella di contare il numero di predizioni sbagliate che esso compie.

$$E = \frac{a}{m}$$

Dove a solo il numero di predizioni sbagliate ed m le predizioni totali E sarà il suo **error rate**.

L'accuracy è calcolata:

$$\text{accuracy} = 1 - E = \frac{m - a}{m}$$

L'errore quando viene valutato sul training set è chiamato **empirical error** o **training error**. Quando calcolo l'errore su un data set indipendente si chiamerà **generalization error** o **test error**.

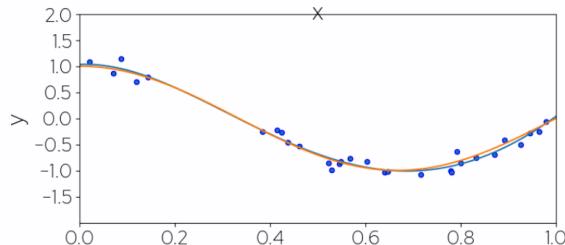
Nota

L'obiettivo principale di un algoritmo di apprendimento è quello di **minimizzare l' errore di generalizzazione**.

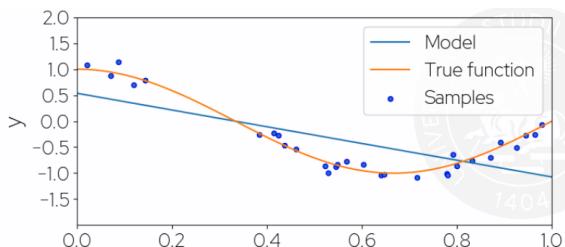
Durante l'addestramento non possiamo calcolare direttamente il **generalization error**, perché non abbiamo accesso ai nuovi esempi, quindi lo approssimiamo minimizzando il **training error**.

2.2 Overfitting

Quando un **modello si adatta troppo bene ai dati di training**, è probabile che alcune peculiarità degli esempi di training vengano assunte come proprietà generali dei dati. Questo porta a un modello che funziona bene sul set di training ma male sui nuovi campioni. Questo fenomeno è chiamato **overfitting**.



Il **fenomeno opposto**, in cui il modello è troppo semplice per catturare i modelli sottostanti nei dati, è chiamato **underfitting**.



L'**underfitting** è solitamente facilmente risolvibile utilizzando un modello più complesso. L'**overfitting** è più difficile da risolvere, poiché richiede un attento equilibrio tra complessità del modello e capacità di generalizzazione. Ciò nonostante, la maggior parte degli algoritmi di apprendimento dispone di meccanismi per prevenire l'overfitting, come **regularization techniques** o **early stopping**.

In genere si preferisce fare un leggero overfitting per avere una capacità espressiva sufficiente.

2.3 Metodi di valutazione

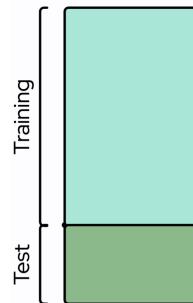
Possiamo valutare le prestazioni di un modello misurandone l'**errore di generalizzazione** su nuovi campioni. Tuttavia, non abbiamo accesso a nuovi campioni durante l'addestramento.

Utilizziamo invece diverse tecniche per stimare l'errore di generalizzazione sulla base dei dati di addestramento.

2.3.1 Validazione Hold-out

Divido l'insieme dei dati in due parti:

- **Training set:** dati che darò in pasto all'algoritmo per l'apprendimento.
- **Test set:** dati che utilizzerò per valutare.

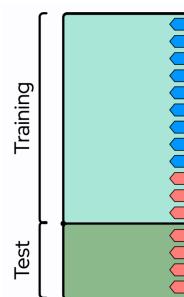


```
1 from sklearn.model_selection import train_test_split
2 ...
3
4 #Assume X,y contains the dataset
5
6 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
    random_state=42)
```

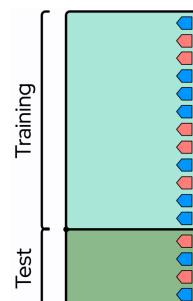
Come effettuare lo split:

- **Simple split**

I primi n dati gli prendo come training e i restanti come test. Ma se ho dei dati non casuali rischio di avere un test set non veritiero.



- **Stratified split** Si mescolano i dati prima di estrarli poi procedo a prendere i primi n come training e il restante come test.

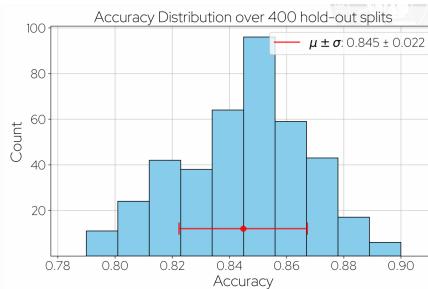


```

1   from model_selection import train_test_split
2   ...
3
4   #Assume X,y contains the dataset
5
6   X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
7       =0.3, shuffle=True, #(default) sufficient for large datasets
8       stratify=y, #useful for small and/or imbalanced datasets
9       random_state=42)

```

Se ripetiamo l'esperimento con il metodo di valutazione hold-out molte volte otterremo tanti risultati diversi, perché training e test set cambiano.



Per ottenere una stima dell'errore di generalizzazione più affidabile possiamo ripetere la validazione hold-out molteplici volte e facciamo la media dei risultati.

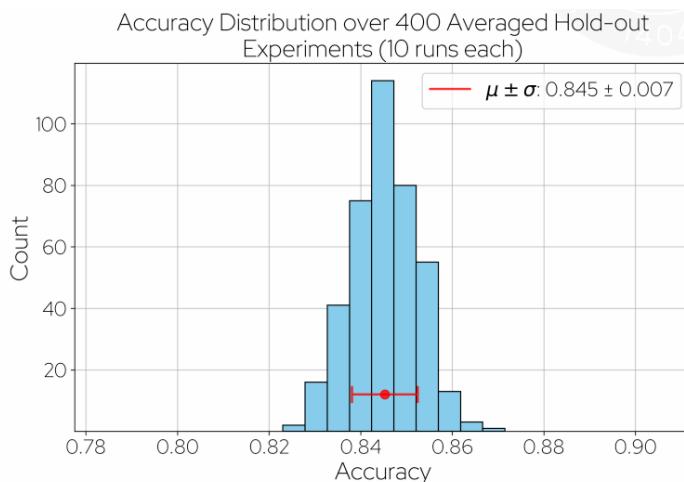
La media di n r.v. **indipendenti**, ognuno con media μ e varianza σ^2 è una nuova r.v. con:

Media

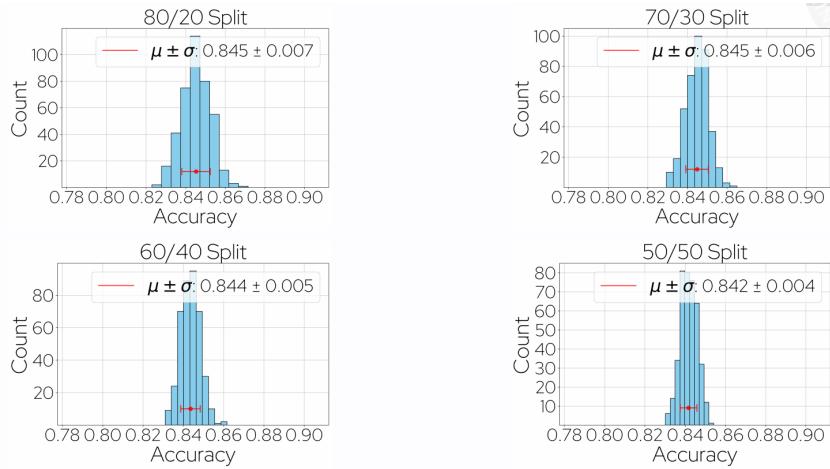
$$\frac{1}{n} \sum_{i=1}^n E_i = \frac{1}{n} \sum_{i=1}^n \mu = \mu$$

Varianza

$$\frac{1}{n^2} \sum_{i=1}^n \sigma^2 = \frac{\sigma^2}{n}$$

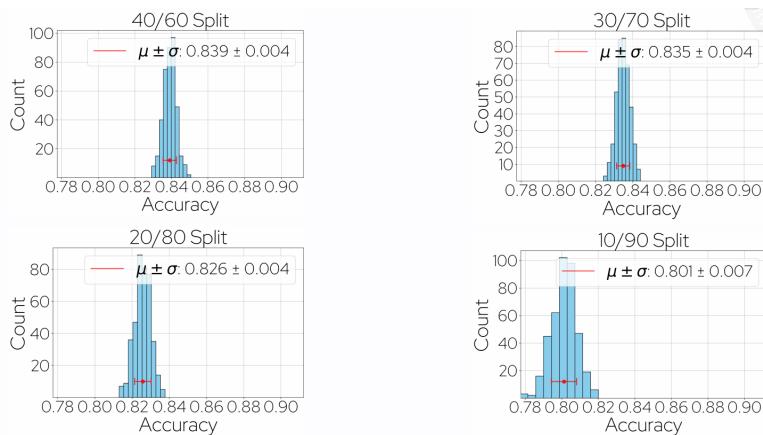


Modifica del rapporto di divisione Una cosa che influisce sull'ampiezza della campana è come stiamo separando il training set dal test set.



Nel momento in cui mettiamo più esempi nel training set otterremo un classificatore che dovrebbe essere più bravo a predire.

All'aumentare della percentuale di esempi nel test set avremo un accuratezza minore ma la varianza diminuisce.

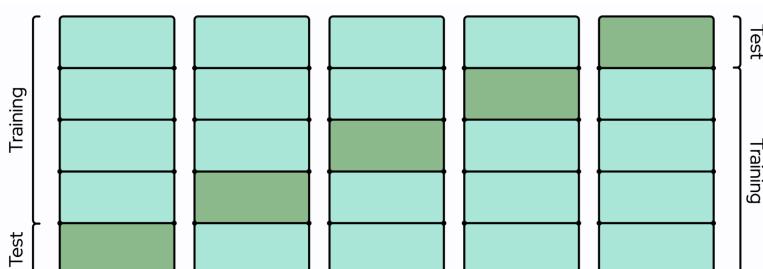


Continuando ad aumentare la percentuale di esempi per il test set la media andrà sempre peggiorando ma la varianza non migliorerà all'infinito anzi ricomincerà ad aumentare.

Se il dataset è grande abbastanza, la **validazione hold-out** è un **semplice ed efficace** metodo per stimare l'errore di generalizzazione.

Ma, se il **dataset è piccolo**, per tenere da parte una quantità sostanziale di dati per il testing potrebbe portare ad una stima non precisa dell'errore di generalizzazione. In questi casi si può usare la **cross-validation**.

2.3.2 Cross validation



Dividiamo il dataset in k parti, ripetiamo il processo di hold-out k volte. La prima volta utilizziamo i primi $k - 1$ **fold** come training set e l'ultimo come test set, la volta successiva utilizziamo il penultimo come test set e tutti gli altri come training set e così via. Alla fine facciamo una **media** e otteniamo una **stima dell'errore di generalizzazione**.

Questa tecnica ha il vantaggio che l'algoritmo di apprendimento ha avuto l'opportunità di vedere, almeno una volta, tutti gli esempi.

Metodo più flessibile

```

1   from sklearn.model_selection import KFold
2   from sklearn.linear_model import LogisticRegression
3   from sklearn.metrics import accuracy_score
4   import numpy as np
5
6   # Assume X,y contains the dataset
7
8   clf = LogisticRegression()
9
10 kf = KFold(n_splits=5, shuffle=True, random_state=42)
11 accuracies = []
12 for train_index, test_index in kf.split(X):
13     X_train, X_test = X[train_index], X[test_index]
14     y_train, y_test = y[train_index], y[test_index]
15
16     clf.fit(X_train, y_train)
17     score = accuracy_score(y_test, clf.predict(X_test))
18     accuracies.append(score)
19
20 print(f"x-val accuracy: {np.array(accuracies).mean()}")

```

Per sapere in modo veloce l'errore di generalizzazione.

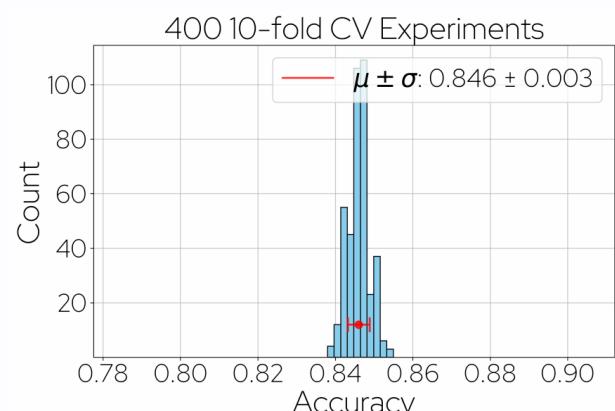
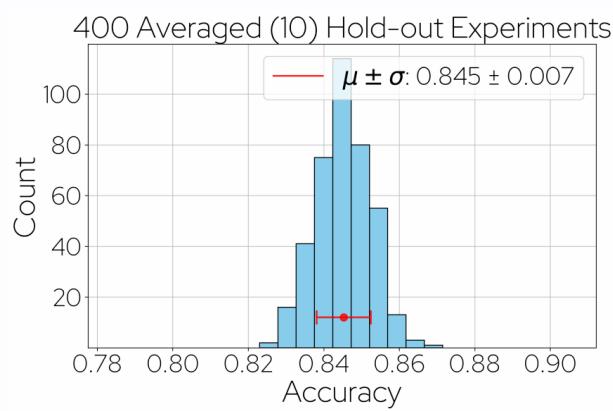
```

1   from sklearn.model_selection import cross_val_score
2   from sklearn.linear_model import LogisticRegression
3
4   # Assume X,y contains the dataset
5
6   clf = LogisticRegression()
7   scores = cross_val_score(clf, X, y, cv=5)
8   print(f"x-val accuracy: {scores.mean()}")

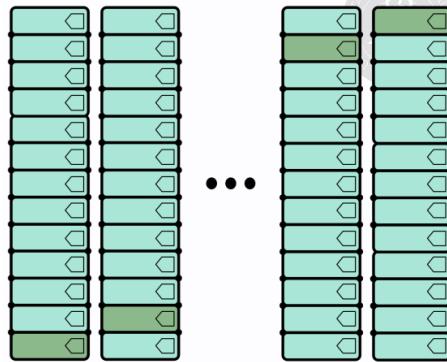
```

Differenza tra hold-out e cross-validation: Nella **cross validation**, ogni campione viene utilizzato per il test **una sola volta**, mentre se ripetiamo **hold-out**, alcuni campioni possono essere utilizzati **più volte per il test** e altri potrebbero **non essere mai utilizzati per il test**. Ciò si traduce in una stima più affidabile dell'errore di generalizzazione (**varianza minore**).

Se il **dataset è sufficientemente grande**, la differenza è trascurabile



2.3.3 Leave-One-Out



Leave-One-Out è un caso estremo di cross-validation dove **ogni campione viene utilizzato per il test una sola volta**, mentre tutti gli altri campioni vengono utilizzati per l'addestramento. Di conseguenza, l'**errore di generalizzazione** viene stimato con una **varianza molto bassa**. Tuttavia, il **costo computazionale** è molto elevato, poiché è necessario addestrare il modello m volte, dove m è il numero di campioni nel set di dati.

2.3.4 Bootstrapping

Dato un set di dati D con campioni m , il nostro obiettivo è stimare l'**errore di generalizzazione** di un modello addestrato sull'**intero set di dati**. Tuttavia, la validazione hold-out e la cross-validation richiedono di addestrare il modello su un sottoinsieme di dati, il che può portare a una stima distorta dell'errore di generalizzazione. Per superare questo problema, possiamo utilizzare il **bootstrapping**. L'idea è quella di **creare più campioni bootstrap** dal set di dati originale D , ciascuno contenente m campioni, campionando con restituzione.

Definizione

Un **campione bootstrap** è un campione ottenuto campionando con sostituzione dal set di dati originale un numero di esempi pari alla dimensione del set di dati originale.

Nota

I **campioni bootstrap non sono disgiunti**. Infatti, ogni campione è ottenuto campionando con sostituzione dal dataset originale, quindi **alcuni campioni potrebbero essere ripetuti** nello stesso campione bootstrap e **altri potrebbero non comparire** in esso.

In realtà è facile calcolare la probabilità che un campione non venga selezionato in un campione bootstrap:

$$P(\text{sample not selected}) = \left(1 - \frac{1}{m}\right)^m$$

Per valori grandi di m , questa probabilità è approssimativamente $e^{-1} \approx 0.3679$, il che significa che in media, circa il **37% dei campioni non viene selezionato** in un campione bootstrap.

Sia D' un campione bootstrap ottenuto da D . Possiamo **addestrare un modello** su D' e **valutarne le prestazioni** sui campioni in $D - D'$ che non sono stati selezionati nel campione bootstrap. In questo modo, possiamo ottenere una stima dell'errore di generalizzazione del modello addestrato su un campione che ha la stessa dimensione del dataset originale.

Nota

Il bootstrapping è **particolarmente utile** quando il **set di dati è piccolo** o quando non esiste un modo efficace per suddividerlo in set di training e test. Quando il set di dati è più grande, il bootstrapping di solito non è necessario e vengono utilizzate la validazione hold-out o la validazione incrociata.

2.4 Tuning dei parametri

Uno dei motivi per cui si utilizzano i metodi di valutazione sopra descritti è la **selezione del modello migliore** tra un insieme di modelli candidati. Questo può significare selezionare il **miglior algoritmo di apprendimento** o selezionare i **migliori iperparametri** per un dato algoritmo di apprendimento.

Definizione

I **parametri** sono gli "oggetti" che l'algoritmo di apprendimento manipola per adattare il modello ai dati, come i pesi in una rete neurale o i coefficienti in un modello di regressione lineare.

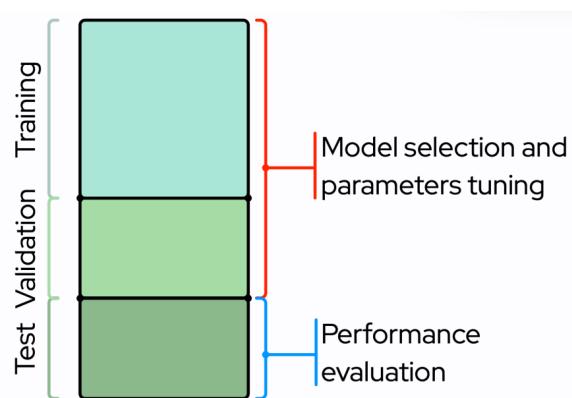
Gli **iperparametri** sono parametri dell'algoritmo di apprendimento stesso, come l'altezza di un albero decisionale o il parametro di regolarizzazione in un modello di regressione lineare.

La stima dell'errore di generalizzazione utilizzando le tecniche descritte può essere utilizzata per due scopi:

- **Selezione del modello o ottimizzazione dei parametri:** ovvero, selezionare il modello migliore tra un insieme di modelli candidati o selezionare i migliori iperparametri per un dato algoritmo di apprendimento.
- **Stima delle prestazioni:** ovvero stimare le prestazioni del modello finale sulla base di dati non visibili.

Per capire quale è l'algoritmo migliore facciamo un esperimento, prendiamo il dataset, facciamo l'addestramento prima con un set di iperparametri poi con un altro set, valutiamo l'errore di generalizzazione in entrambi i casi e scegliamo quello con l'errore più basso.

Così facendo stiamo valutando basandoci sul test set per fare una scelta quindi bisogna fare **attenzione a non sovraadattare il modello** al set di test. Ciò significa che non dovremmo utilizzare il **set di test per ottimizzare gli iperparametri** del modello, poiché ciò porterebbe a una stima distorta dell'errore di generalizzazione del modello finale.



Per non utilizzare il test set per decidere quali iperparametri utilizzare si divide il dataset in 3 parti:

- **Training set**
- **Validation set:** una parte di esempi messa da parte per fare la stima degli **iperparametri**
- **Test set:** utilizzato alla fine di tutto quando il modello è stato già scelto per sapere quanto vale l'**errore di generalizzazione** del modello

Esempio

Supponiamo di voler selezionare i migliori iperparametri per un algoritmo di apprendimento automatico. Si dispone di un set di 1000 iperparametri tra cui scegliere e si desidera misurare l'errore utilizzando il set di test. Poiché abbiamo visto che questo potrebbe rappresentare un problema, teniamo da parte anche un set aggiuntivo di esempi, il set di validazione, per misurare l'errore sul classificatore scelto.

Esempio

Poiché sia i set di validazione che quelli di test sono **campioni finiti**, la misurazione dell'errore su di essi introdurrà un po' di rumore nella stima dell'errore di generalizzazione, ovvero:

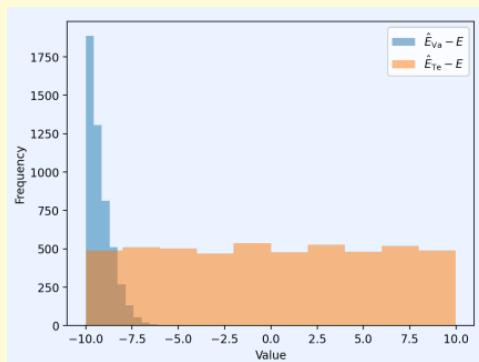
$$\hat{E}_{Te} = E + \epsilon_{Te}, \quad \hat{E}_{Val} = E + \epsilon_{Val}$$

Si scelgono quindi gli iperparametri che riducono al minimo l'errore sul set di convalida.

Domanda: alla fine del processo, i due errori \hat{E}_{Te} e \hat{E}_{Val} saranno ancora stimatori senza bias dell'errore di generalizzazione? In altri termini, $\mathbb{E}[\hat{E}_{Te}] = \mathbb{E}[\hat{E}_{Val}] = E$?

La media dello stimatore basata sul Validation set avrà un bias, invece quella basata sul Test set sarà senza bias.

```
1 # 1000 hyper-parameters to set, each yielding an error in (0, 100)
2 means = np.random.uniform(0, 100, 1000)
3
4 valid_error_dist = []
5 test_error_dist = []
6
7 N = 5000
8 for i in range(N):
9     samples = np.array(
10         [
11             [
12                 m,                                     #E
13                 m + np.random.uniform(-10, 10)        #E_val
14                 m + np.random.uniform(-10, 10)        #E_test
15             ]
16             for m in means
17         ]
18     )
19
20 #selects the best hp according to R_val
21 amin = np.argmin(samples[:, 1])
22
23 valid_error_dist.append(samples[amin, 1] - samples[amin, 0])
24 test_error_dist.append(samples[amin, 2] - samples[amin, 0])
```



2.5 Misurazione delle performance

Diversi indicatori di performance riflettono le diverse esigenze dei compiti e producono risultati di valutazione diversi. La scelta dell'indicatore di performance corretto è fondamentale per una corretta valutazione del modello.

2.5.1 Mean Square Error (MSE)

L'errore quadratico medio è una misura di prestazione comune per le attività di regressione. In generale, se si conosce la **distribuzione dei dati** \mathcal{D} , l'errore quadratico medio di una funzione è definito come:

$$E(f; \mathcal{D}) = \mathbb{E}_{(X,y) \sim \mathcal{D}}[(f(X) - y)^2] = \int_{(X,y) \sim \mathcal{D}} (f(X) - y)^2 p(X, y) dX dy$$

Nella maggior parte dei casi, non conosciamo la distribuzione dei dati \mathcal{D} , ma disponiamo di un set di dati $D = \{(X_i, y_i)\}_{i=1}^m$ di m campioni estratti da \mathcal{D} . In questo caso, possiamo stimare l'errore quadratico medio come:

$$E(f; D) = \frac{1}{m} \sum_{i=1}^m (f(X_i) - y_i)^2$$

2.5.2 Error rate

Analogamente all'errore quadratico medio, possiamo definire il **tasso di errore** e l'**accuratezza** per le attività di classificazione. La forma generale, supponendo di conoscere la distribuzione dei dati, è:

$$E(f; \mathcal{D}) = \mathbb{E}_{(X,y) \sim \mathcal{D}}[\mathbb{I}(f(X) \neq y)] = \int_{(X,y) \sim \mathcal{D}} \mathbb{I}(f(X) \neq y) p(X, y) dX dy$$

Dove \mathbb{I} è la **funzione indicatrice** che restituisce 1 se la condizione è vera e 0 altrimenti.

Se disponiamo di un set di dati $D = \{(X_i, y_i)\}_{i=1}^m$ di m campioni estratti da \mathcal{D} , possiamo stimare il tasso di errore come:

$$E(f; D) = \frac{1}{m} \sum_{i=1}^m \mathbb{I}(f(X_i) \neq y_i)$$

2.5.3 Accuracy

L'accuratezza è il **complementare dell'error rate**:

$$acc(f; \mathcal{D}) = 1 - E(f; \mathcal{D}) = 1 - \mathbb{E}_{(X,y) \sim \mathcal{D}}[\mathbb{I}(f(X) \neq y)]$$

o, considerando un dataset $D = \{(X_i, y_i)\}_{i=1}^m$ di m campioni estratti da \mathcal{D} :

$$acc(f; D) = 1 - E(f; D) = 1 - \frac{1}{m} \sum_{i=1}^m \mathbb{I}(f(X_i) \neq y_i)$$

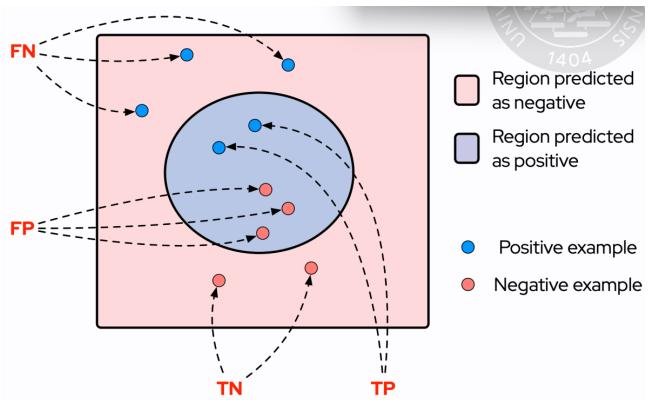
2.5.4 Matrice di confusione

È possibile definire molte altre misure di performance, a seconda del compito e dei requisiti specifici. Per le attività di classificazione, molte di queste misure possono essere derivate dalla **matrice di confusione**.

	Predicted Positive	Predicted Negative
Actual Positive	TP	FN
Actual Negative	FP	TN

con:

- **TP:** True Positive (numero degli esempi che sono stati predetti positivi ed erano veramente positivi)
- **FP:** False Positive (predetti positivi ma in realtà negativi)
- **TN:** True Negative (predetti negativi e veramente negativi)
- **FN:** False Negative (predetti negativi ma in realtà positivi)



2.5.5 Precision, Recall

A volte, gli **errori** non sono **ugualmente importanti**. Ad esempio, in un'attività di diagnosi di cancro, un falso negativo (mancanza di una malattia) può essere più critico di un falso positivo (diagnosi errata di una malattia).

In questo esempio, potremmo voler dare **priorità al richiamo** (recall) (la capacità di identificare tutti i casi di cancro) rispetto alla **precisione** (la capacità di evitare false diagnosi di cancro).

In un'attività di rilevamento dello spam, d'altra parte, potremmo voler dare **priorità alla precisione** (per evitare di etichettare erroneamente email legittime come spam) rispetto al **richiamo** (recall) (per identificare tutti i messaggi di spam).

Più formalmente, la **precisione** è definita come:

$$precision = \frac{TP}{TP + FP} = \frac{TP}{\#predicted\ positives}$$

e il **richiamo** (recall) è definito come:

$$recall = \frac{TP}{TP + FN} = \frac{TP}{\#actual\ positives}$$

Precision e **recall** vengono spesso utilizzati insieme, poiché forniscono informazioni complementari sulle prestazioni del modello. **Di solito sono contraddittorie**, il che significa che l'aumento di una spesso porta a una diminuzione dell'altra.

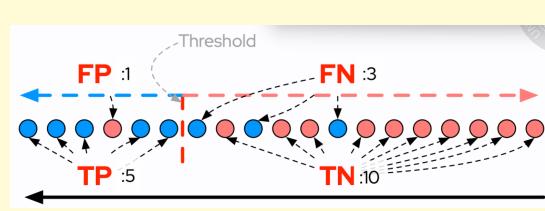
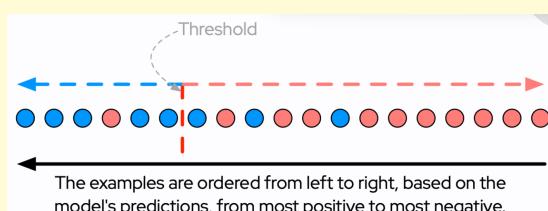
Ad esempio, se iniziamo a classificare **più campioni come positivi**, aumenteremo il numero di veri positivi, ma anche il numero di falsi positivi, **diminuendo** così la **precisione**, ma **aumentando il recall**.

D'altra parte, se iniziamo a classificare **meno campioni come positivi**, diminuiremo il numero di falsi positivi, **aumentando** così la **precisione**, ma anche il numero di falsi negativi, **diminuendo** così il **recall**.

2.5.6 P-R Plots

Supponiamo di avere un classificatore che fornisca un **punteggio di confidenza** per ciascun campione, indicando quanto è sicuro che il campione appartenga alla classe positiva. È quindi possibile variare una **soglia** su questo punteggio per ottenere diversi valori di precision e recall.

Esempio

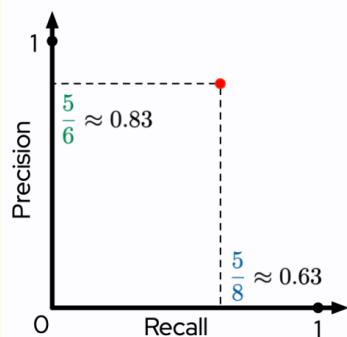


Esempio

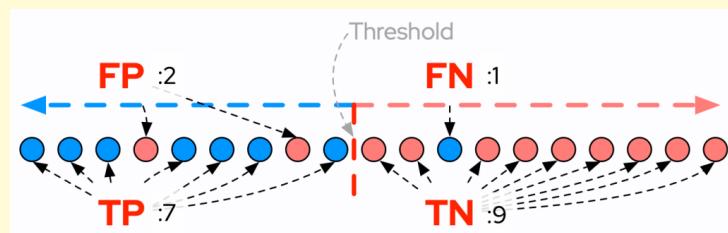
In questo esempio ho $FP = 1$, $FN = 3$, $TP = 5$, $TN = 10$

$$precision = \frac{TN}{TP + FP} = \frac{5}{6}$$

$$recall = \frac{TP}{TP + FN} = \frac{5}{8}$$

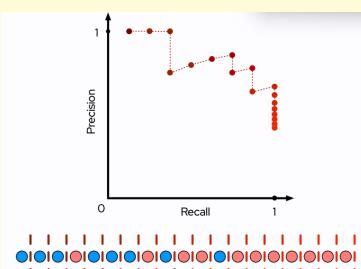
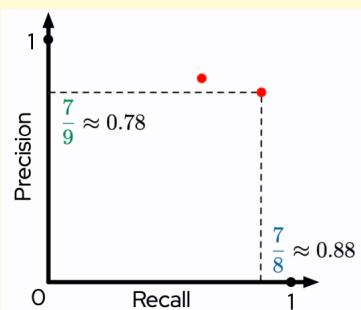


Cambiando la posizione della soglia otterò:



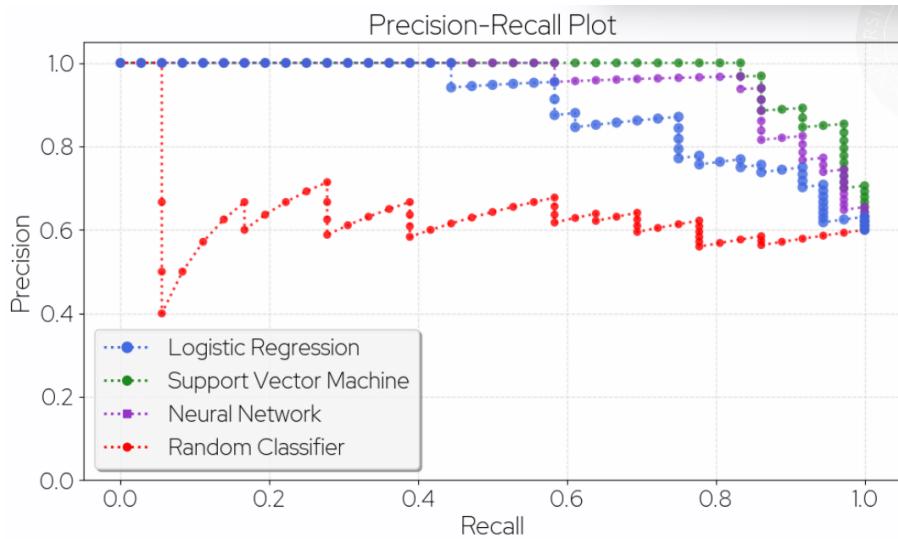
$$precision = \frac{TN}{TP + FP} = \frac{7}{9}$$

$$recall = \frac{TP}{TP + FN} = \frac{7}{8}$$



Dovendo confrontare più classificatori che restituiscono una soglia di confidenza ottengo un grafico

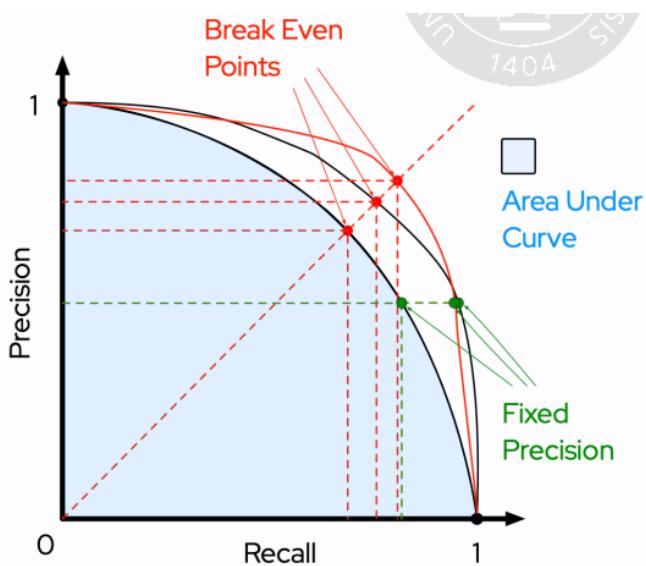
del tipo:



In particolare si può vedere che le **Support Vector Machine** (in verde) dominano tutti gli altri classificatori, se dovessi scegliere un classificatore sceglieri questo perché a parità di threshold vince su tutti gli altri.

Come selezioniamo gli studenti migliori date alcune curve precisione-richiamo?

- **Fissare un dato punto di precision** o recall e selezionare il modello che raggiunge il miglior richiamo o precisione in quel punto.
- Selezionare la curva con l'**area sotto la curva (AUC) più alta** (n.b. difficile da calcolare).
- Selezionare il **miglior punto di break-even**, ovvero il punto in cui precisione e recall sono uguali.



F1-score Una misura importante che combina precisione e recall è il **punteggio F1**, definito come la **media armonica** di precisione e recall:

$$F1 = \frac{2}{\frac{1}{precision} + \frac{1}{recall}} = \frac{2 \times precision \times recall}{precision + recall} = \frac{2 \times TP}{TOT + FP - FN}$$

o in forma più generale (F_β -score):

$$F_\beta = \frac{1 + \beta^2}{\frac{1}{precision} + \frac{\beta^2}{recall}} = \frac{(1 + \beta^2) \times precision \times recall}{\beta^2 \times precision + recall}$$

Media armonica ponderata

La media armonica ponderata di due valori a e b è definita come:

$$\frac{(w_a + w_b)}{\frac{w_a}{a} + \frac{w_b}{b}}$$

Nota

La F-misura definita da F_β misura quanto sono bravi a recuperare gli esempi rispetto ad un utente che pensa che la recall sia β volte più importante della precision.

Supponiamo di voler **calcolare la media** dei contributi di diverse matrici di confusione (ad esempio, perché ci troviamo in un problema multi-classe, o vogliamo combinare i risultati di diversi round di convalida incrociata). Indichiamo con P e R le statistiche di **precision** e **recall**.

Questo può essere fatto in due modi:

1. **Macro-averaging:** calcolo la Precision e la Recall per ogni matrice di confusione, denotate come $(P_1, R_1), (P_2, R_2), \dots, (P_n, R_n)$. Prendendo la loro media abbiamo la precision, recall e F_1 macro-averaged:

$$macro\text{-}P = \frac{1}{n} \sum_{i=1}^n P_i$$

$$macro\text{-}R = \frac{1}{n} \sum_{i=1}^n R_i$$

$$macro\text{-}F_1 = \frac{2 \times macro\text{-}P \times macro\text{-}R}{macro\text{-}P + macro\text{-}R}$$

2. **Micro-averaging:** calcolare le medie elemento per elemento nelle matrici di confusione, per ottenere $\overline{TP}, \overline{FP}, \overline{TN}, \overline{FN}$, e quindi calcola la micro precision, recall e F_1 come:

$$micro\text{-}P = \frac{\overline{TP}}{\overline{TP} + \overline{FP}}$$

$$micro\text{-}R = \frac{\overline{TP}}{\overline{TP} + \overline{FN}}$$

$$micro\text{-}F_1 = \frac{2 \times micro\text{-}P \times micro\text{-}R}{micro\text{-}P + micro\text{-}R}$$

- **Macro-averaging** tratta tutte le matrici di confusione allo stesso modo, indipendentemente dalle loro dimensioni. Questo significa che matrici di confusione di piccole dimensioni possono avere un impatto significativo sul risultato finale.
- **Micro-averaging** attribuisce maggiore peso alle matrici di confusione più grandi, poiché contribuiscono maggiormente al conteggio complessivo di TP, FP, TN e FN. Ciò significa che il risultato finale è più rappresentativo della performance complessiva di tutte le matrici di confusione.

Ciò è importante, ad esempio, nei problemi di classificazione multiclasse in cui le classi possono essere sbilanciate.

Esempio Macro- F_1

Assumendo di avere tree matrici di confusione:

$$CM_1 = \begin{bmatrix} 10 & 2 \\ 1 & 7 \end{bmatrix} \quad CM_2 = \begin{bmatrix} 8 & 3 \\ 2 & 6 \end{bmatrix} \quad CM_3 = \begin{bmatrix} 9 & 1 \\ 1 & 8 \end{bmatrix}$$

CM	precision	recall	F1-score
CM ₁	0.9091	0.8333	0.8695
CM ₂	0.8000	0.7273	0.7619
CM ₃	0.9000	0.9000	0.9000
Avg	0.8697	0.8202	0.8438

$$\text{macro-}F_1 = \frac{2 \times \text{macro-}P \times \text{macro-}R}{\text{macro-}P + \text{macro-}R} = \frac{2 \times 0.8697 \times 0.8202}{0.8697 + 0.8202} \approx 0.8442$$

Esempio Micro- F_1

Con le stesse matrici di confusione di prima:

$$CM_1 = \begin{bmatrix} 10 & 2 \\ 1 & 7 \end{bmatrix} \quad CM_2 = \begin{bmatrix} 8 & 3 \\ 2 & 6 \end{bmatrix} \quad CM_3 = \begin{bmatrix} 9 & 1 \\ 1 & 8 \end{bmatrix} \quad \Rightarrow \quad \overline{CM} = \begin{bmatrix} 9 & 2 \\ 1.3333 & 7 \end{bmatrix}$$

La precision e recall micro-averaged sono:

$$\text{micro-}P = \frac{9}{9 + 1.3333} = 0.8710 \quad \text{micro-}R = \frac{9}{9 + 2} = 0.8182$$

La F_1 micro-averaged è:

$$\text{micro-}F_1 = \frac{2 \times \text{micro-}P \times \text{micro-}R}{\text{micro-}P + \text{micro-}R} = \frac{2 \times 0.8710 \times 0.8182}{0.8710 + 0.8182} = 0.8438$$

2.5.7 ROC e AUC

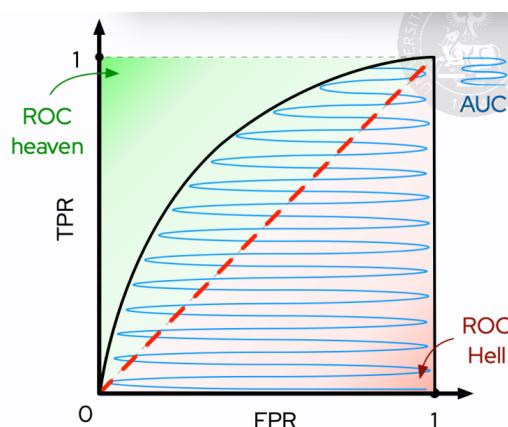
Se, invece di utilizzare precision e recall, tracciamo il **false positive rate (FPR)** rispetto al **true positive rate (TPR)**, otteniamo un grafico **ROC** (Receiver Operating Characteristic curve).

$$FPR = \frac{FP}{FP + TN} = \frac{FP}{\# \text{actual negative}}$$

$$TPR = \frac{TP}{TP + FN} = \frac{TP}{\# \text{actual positives}}$$

Come nei grafici P-R, diciamo che un learner *A* è **migliore di un learner *B*** se la curva ROC di *A* racchiude interamente la curva ROC di *B*. Tuttavia, quando esistono intersezioni, nessun learner è generalmente migliore dell'altro.

Un modo per confrontare le curve ROC intersecate è calcolare le **aree sotto le curve ROC**.



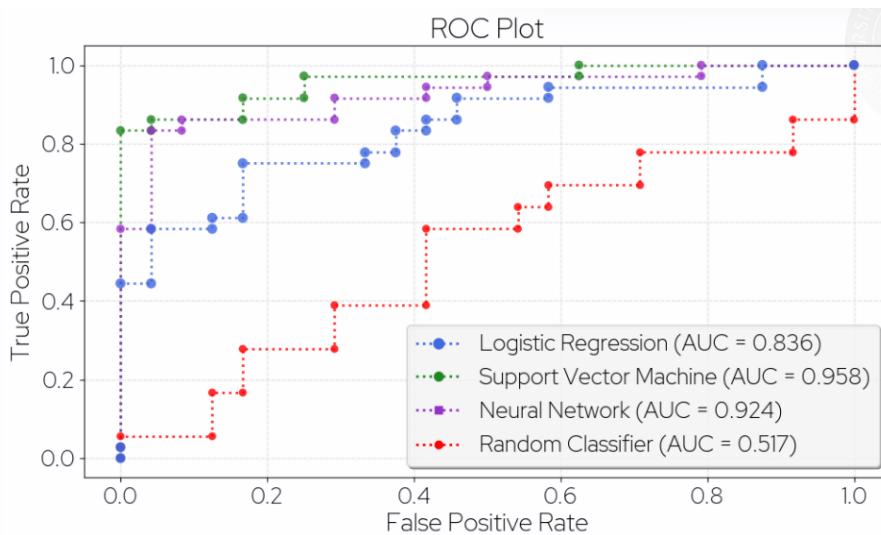
In alto a sinistra (**ROC heaven**) troviamo classificatori che **non sbagliano mai**, in basso a destra (**ROC Hell**) troviamo classificatori che **sbagliano sempre** e al centro sono presenti i peggiori classificatori perché rispondono sempre a caso. Noi **vogliamo stare il più lontano possibile dalla diagonale**.

Area Under the Curve (AUC) Nel caso delle curve ROC, l'area sotto la curva (AUC) è facile da calcolare (anche per le curve teoriche, poiché è sempre non decrescente).

Si può anche dimostrare che è correlata all'errore di ranking del classificatore.

$$1 - AUC = \ell_{rank} \triangleq \frac{1}{m^+ m^-} \sum_{X^+ \in D^+} \sum_{X^- \in D^-} \mathbb{I}[f(X^+) < f(X^-)] + \frac{1}{2} \mathbb{I}[f(X^+) = f(X^-)]$$

Somma su tutti gli esempi positivi, somma su tutti gli esempi negativi calcoliamo un punto di errore se il ranking dell'esempio positivo è minore del ranking dell'esempio negativo, abbiamo anche un altro termine perché possiamo avere dei casi in cui c'è un pareggio in questo caso assegnano mezzo punto di errore. Facciamo la somma di tutto e per normalizzarlo dividiamo tutto per il numero di esempi positivi moltiplicata per il numero di esempi negativi.



```

1 # ...
2 # Assume X_train, X_test, y_train, y_test are defined
3
4 clf = LogisticRegression()
5 clf.fit(X_train, y_train)
6
7 y_scores = clf.predict_proba(X_test)[:, 1]
8
9 # Compute ROC curve and AUC
10 fpr, tpr, _ = roc_curve(y_test, y_scores)
11 roc_auc = auc(fpr, tpr)
12
13 # You can easily plot the ROC curve using matplotlib
14 plt.plot(fpr, tpr)
15
16 # Alternatively, you can directly use roc_auc_score
17 roc_auc2 = roc_auc_score(y_test, y_scores)
18
19 # For Precision-Recall, we don't usually compute
20 # AUC, but we can compute the average precision score
21 avg_precision = average_precision_score(y_test, y_scores)

```

2.6 Cost-Sensitive Error rates

A volte si potrebbe voler adattare l'accuratezza (o l'errore) in modo che siano **sensibili** ai diversi costi di errore per FP e FN . Si consideri $cost_{xy}$ il costo di una classificazione errata x come y .

Per un compito di classificazione binaria potremmo avere la matrice $cost$:

	Predicted as 0	Predicted as 1
Actual 0	0	$cost_{01}$
Actual 1	$cost_{10}$	0

Quindi, supponendo che la classe sia positiva, si può definire il **tasso di errore sensibile ai costi** come:

$$E(f; \mathcal{D}; cost) = \frac{1}{m} \left(\sum_{(X,y) \in \mathcal{D}^+} \mathbb{I}[f(X) \neq y] \times cost_{01} + \sum_{(X,y) \in \mathcal{D}^-} \mathbb{I}[f(X) \neq y] \times cost_{10} \right)$$

Allo stesso modo, possiamo definire il tasso di errore e l'accuratezza basati sulla distribuzione.

2.7 Comparison Test

Confrontare i risultati di diversi learner **non** è semplice per diversi motivi:

- Vogliamo valutare le **prestazioni di generalizzazione**, ma abbiamo accesso solo a un **set di test finito**.
- Le prestazioni del modello possono variare tra **diversi set di test**, anche se hanno le stesse dimensioni.
- La **casualità** intrinseca di molti algoritmi di apprendimento significa che possono produrre modelli diversi se addestrati più volte sugli stessi dati.

L'**Hypothesis testing** consente un rigoroso confronto statistico tra diversi modelli.

2.7.1 Test binomiale

Dato un modello f con un **tasso di errore reale sconosciuto** ϵ , vogliamo verificare se il modello ha prestazioni significativamente migliori rispetto a una determinata soglia di tasso di errore $\epsilon_0 \in [0, 1]$ e ipotizziamo di osservare un **tasso di errore empirico** $\hat{\epsilon}$ su un set di test di dimensioni m tali da consentire un modello f .

Formalizziamo questo come un test di ipotesi:

- **Null hypothesis** (il modello non è migliore di ϵ_0):

$$H_0 : \epsilon \geq \epsilon_0$$

- **Alternative hypothesis** (il modello è migliore di ϵ_0):

$$H_1 : \epsilon < \epsilon_0$$

La difficoltà di tutto questo sta nel fatto che ϵ è sconosciuto, quindi non possiamo calcolare direttamente la probabilità di osservare un tasso di errore empirico $\hat{\epsilon}$ dato un tasso di errore reale ϵ . Poter assumere l'**ipotesi nulla** ci permette di fare dei ragionamenti probabilistici. L'**ipotesi alternativa** è quella che vogliamo dimostrare.

Sia E la variabile casuale che conta il numero di errori fatti dal modello sul test set. Sotto l'**ipotesi nulla** H_0 , E segue una distribuzione binomiale con parametri m e ϵ_0 : $E \sim \text{Bin}(m, \epsilon_0)$

Nota

Di seguito, indicheremo con $\hat{\epsilon}$ il tasso di errore osservato e con $\hat{e} = \hat{\epsilon} \cdot m$ il numero di errori.

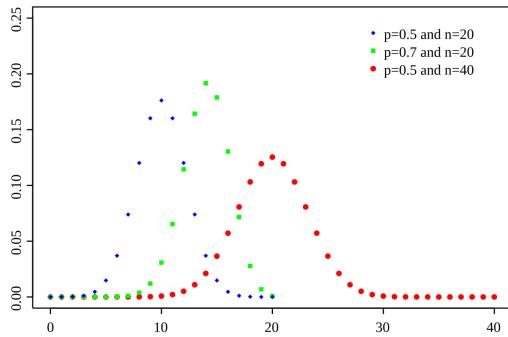


Figura 1: Distribuzione binomiale

Decision Rule

Rifiutare l'ipotesi nulla H_0 a livello di significatività α (o con confidenza $1 - \alpha$) se:

$$P(E \leq \hat{e} | \epsilon = \epsilon_0) < \alpha$$

In altre parole, se è **improbabile** (sotto l'ipotesi nulla H_0) osservare un numero di errori pari o inferiore a \hat{e} , allora concludiamo che il modello **funziona meglio** di ϵ_0 .

In caso contrario, **non rifiutiamo** H_0 .

P-value

Il **p-value** di un test statistico è la probabilità di **osservare** una statistica del test **altrettanto estrema o più estrema di quella effettivamente osservata**, supponendo che l'ipotesi nulla sia vera.

Il **p-value**, nel nostro caso, è definito come:

$$p\text{-value} = P(E \leq \hat{e} | \epsilon = \epsilon_0) = \sum_{k=0}^{\hat{e}} \binom{m}{k} \epsilon_0^k (1 - \epsilon_0)^{m-k}$$

La probabilità di osservare esattamente k successi su m esperimenti.

Se il p-value è **minore** di α , il risultato è considerato statisticamente significativo e si rifiuta l'ipotesi nulla H_0 .

Esempio

Assumiamo di avere un modello che mostra un tasso di errore del 20%: $\hat{\epsilon} = 0.2$ su un set di test di dimensione $m = 100$, cioè compie $\hat{\epsilon} \times 100 = 20$ errori sul set di test. Vogliamo verificare l'ipotesi che il vero tasso di errore è significativamente minore di 0.3.

L'ipotesi nulla è:

$$H_0 : \epsilon \geq 0.3$$

L'ipotesi alternativa è:

$$H_1 : \epsilon < 0.3$$

Sotto l'ipotesi nulla, la probabilità di osservare \hat{e} errori è data da:

$$P(\hat{e}; \epsilon_0) = \binom{m}{\hat{e}} \epsilon_0^{\hat{e}} (1 - \epsilon_0)^{m-\hat{e}}$$

Il p-value del test è data da:

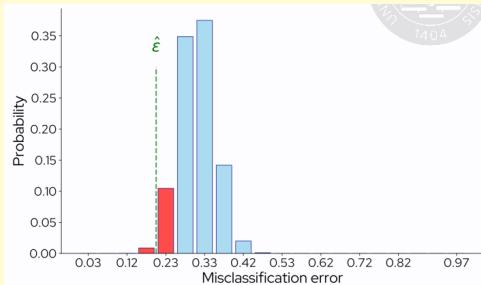
$$P(e \leq \hat{e} | \epsilon = 0.3) = \sum_{k=0}^{20} \binom{100}{k} (0.3)^k (1 - 0.3)^{100-k} = 0.0165$$

Esempio

Se impostiamo il livello di significatività a $\alpha = 0.05$, possiamo rifiutare l'ipotesi nulla H_0 fino a quando $0.0165 < 0.05$. Questo significa che abbiamo abbastanza prove per concludere che il modello performa significativamente meglio di un tasso di errore del 30%.

Fissato un livello di confidenza α , in rosso nella figura è mostrato la barra corrispondente alla coda sinistra della distribuzione binomiale tale che la probabilità cumulativa è minore o uguale a α .

Poiché $\hat{\epsilon}$ cade nella regione, possiamo rifiutare l'ipotesi nulla.



Nel caso in cui volessimo ripetere il test molteplici volte, vorremmo calcolare il tasso di errore massimo $\hat{\epsilon}$ che possiamo osservare per rifiutare ancora l'ipotesi nulla.

Questo valore è chiamato **valore critico** e, per il test binomiale, è definito come il più grande valore di $\hat{\epsilon}$ tale che il p-value è minore o uguale a α :

$$\bar{\epsilon} = \max_{\epsilon} \epsilon \quad \text{subject to} \quad \sum_{i=0}^{\lfloor \epsilon \times m \rfloor} \binom{m}{i} \epsilon^i (1 - \epsilon)^{m-i} \leq \alpha$$

2.7.2 Test di Student

Spesso otteniamo più tassi di errore nei test da cross-validation o da hold-out validation ripetuti. Sia $\hat{\epsilon}_1, \hat{\epsilon}_2, \dots, \hat{\epsilon}_n$ i tassi di errore osservati in k ripetizioni indipendenti del test. Possiamo facilmente calcolare la **media** e la **deviazione standard** dei tassi di errore osservati:

$$\mu = \frac{1}{k} \sum_{i=1}^k \hat{\epsilon}_i$$

$$\sigma^2 = \frac{1}{k-1} \sum_{i=1}^k (\hat{\epsilon}_i - \mu)^2$$

Nota

Per essere rigorosi, l'hold-out ripetuto e (soprattutto) la cross-validation non sono indipendenti, poiché i campioni sono estratti dallo stesso set di dati. Tuttavia, se il set di dati è sufficientemente ampio, la dipendenza è trascurabile e la conclusione rimane valida.

Possiamo considerare questi valori come k campioni indipendenti e identicamente distribuiti (i.i.d.) da una distribuzione normale con media μ e varianza σ^2 e quindi la variabile:

$$\tau = \sqrt{k} \frac{\mu - \epsilon_0}{\sigma}$$

è distribuita come una distribuzione di Student con $k - 1$ gradi di libertà. La distribuzione t di Student è una famiglia di distribuzioni parametrizzata dal valore k che è chiamato **gradi di libertà** (degrees of freedom, df) della distribuzione. Un parametro da fissare per dire quale distribuzione voglio usare.

Il **test di Student** è un test statistico utilizzato per determinare se le medie di due gruppi sono significativamente diverse tra loro. Nel nostro caso, possiamo utilizzarlo per verificare se il tasso di errore medio μ è significativamente diverso da una determinata soglia di tasso di errore ϵ_0 .

L'ipotesi nulla è:

$$H_0 : \mu = \epsilon_0$$

L'ipotesi alternativa è:

$$H_1 : \mu \neq \epsilon_0$$

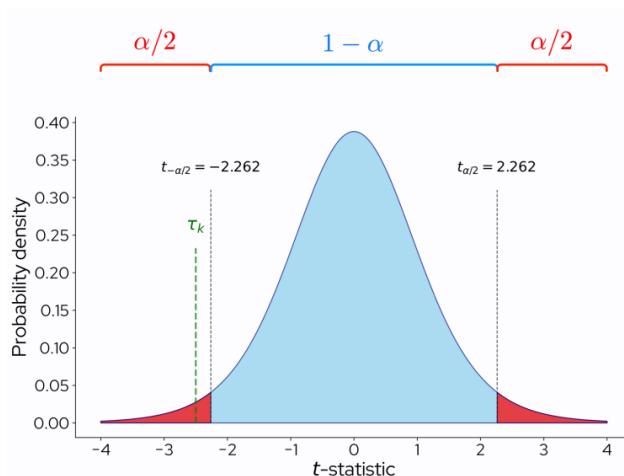
La statistica del test è data da:

$$\tau = \sqrt{k} \frac{\mu - \epsilon_0}{\sigma}$$

Possiamo **rifiutare l'ipotesi nulla** a un **livello di significatività α** se τ è **al di fuori** dell'intervallo $[-t_{\alpha/2}, t_{\alpha/2}]$, dove t_z è il valore critico della distribuzione di Student con $k - 1$ gradi di libertà a livello di significatività α . t_z lo ottengo da delle tabelle precompilate e controllo che il mio τ sia fuori dall'intervallo.

Nelle formule, **rifiuta l'ipotesi nulla** se:

$$\tau \notin [-t_{\alpha/2}, t_{\alpha/2}]$$



Valori critici comunemente utilizzati

df (k-1)	$\alpha = 0.10$	$\alpha = 0.05$	$\alpha = 0.025$	$\alpha = 0.01$
1	3.078	6.314	12.706	31.821
2	1.886	2.920	4.303	6.965
3	1.638	2.353	3.182	4.541
4	1.533	2.132	2.776	3.747
5	1.476	2.015	2.571	3.365
6	1.440	1.943	2.447	3.143
7	1.415	1.895	2.365	2.998
8	1.397	1.860	2.306	2.896
9	1.383	1.833	2.262	2.821
10	1.372	1.812	2.228	2.764
11	1.363	1.796	2.201	2.718
12	1.356	1.782	2.179	2.681
13	1.350	1.771	2.160	2.650
14	1.345	1.761	2.145	2.624
15	1.341	1.753	2.131	2.602
16	1.337	1.746	2.120	2.583
17	1.333	1.740	2.110	2.567
18	1.330	1.734	2.101	2.552
19	1.328	1.729	2.093	2.539
20	1.325	1.725	2.086	2.528
25	1.316	1.708	2.060	2.485
30	1.310	1.697	2.042	2.457
∞	1.282	1.645	1.960	2.326

Il libro riporta i valori critici usand k come numero di campioni (qui sono riportati i gradi di libertà), e α è già diviso per 2. Quindi, la voce per $k = 5$ e $\alpha = 0.05$ sul libro corrisponde alla voce per $df = 4$ e $\alpha = 0.025$ nella tabella.

```

1 from scipy.stats import t
2 import numpy as np
3
4 # Assume we have k error rates in a list
5 error_rates = [0.2, 0.18, 0.22, 0.19, 0.21]
6 k = len(error_rates)
7 mu = np.mean(error_rates)
8 sigma = np.std(error_rates, ddof=1) # ddof=1 for sample standard deviation
9
10 alpha = 0.05
11
12 # ppf == percent point function, i.e., the inverse of the cumulative
13 # distribution function (CDF)
13 t_crit_0 = t.ppf(alpha/2, k-1) # negative crit for left tail
14 t_crit_1 = t.ppf(1 - alpha/2, k-1) # positive critical value for right tail
15
16 tau_k = np.sqrt(k) * (mu - 0.3) / sigma
17
18 if(tau_k < t_crit_0 or tau_k > t_crit_1):
19     print("Reject null hypothesis H0")
20 else:
21     print("Do not reject null hypothesis H0")
22
23 # Alternatively, you can use the ttest_1samp function from scipy.stats
24 from scipy.stats import ttest_1samp
25 t_stat, p_value = ttest_1samp(error_rates, 0.3)
26 if p_value < alpha:
27     print("Reject null hypothesis H0")
28 else:
29     print("Do not reject null hypothesis H0")

```

2.7.3 Cross-Validated Student's t-test

Assumiamo di avere due algoritmi di apprendimento A e B , e denotiamo con $\epsilon_1^A, \dots, \epsilon_k^A$ e $\epsilon_1^B, \dots, \epsilon_k^B$ i tassi di errore ottenuti da **k-fold cross validation**.

Se vogliamo confrontare i due algoritmi, possiamo usare il **k-fold cross-validation Student's t-test**.

L'idea è che, se le performance dei due algoritmi sono le stesse, allora i tassi di errore nei test dovrebbero essere simili nelle stesse suddivisioni training/test: $\epsilon_i^A \approx \epsilon_i^B$ per tutte le $i = 1, \dots, k$.

Il test richiede di calcolare $\Delta_i = \epsilon_i^A - \epsilon_i^B$ per $i = 1, \dots, k$, e quindi calcolare la media μ e la deviazione standard σ della differenza Δ_i .

Note

Nota 1: si presume che i tassi di errore siano indipendenti, il che non è vero nella pratica, ma l'ipotesi è ragionevole se il set di dati è sufficientemente ampio.

Nota 2: la tecnica può essere utilizzata anche con validation hold-out accoppiate (vale a dire, le stesse suddivisioni di formazione/test vengono utilizzate per entrambi gli algoritmi).

Ipotesi nulla: i due algoritmi hanno le stesse prestazioni, cioè, $\mu = 0$.

Ipotesi alternativa: i due algoritmi hanno prestazioni diverse, cioè, $\mu \neq 0$.

La statistica del test è data da:

$$\tau = \frac{\sqrt{k} \cdot \mu}{\sigma}$$

Regola di decisione: rifiutare l'ipotesi nulla H_0 a un livello di significatività α se:

$$\tau \notin [-t_{\alpha/2}, t_{\alpha/2}]$$

dove $t_{\frac{\alpha}{2}}$ è il valore critico della distribuzione di Student con $k-1$ gradi di libertà a livello di significatività $\frac{\alpha}{2}$.

2.7.4 5x2 Cross-Validated t-test

Come notato, l'ipotesi di indipendenza è violata nel caso di CV k-fold. Per risolvere questo problema, è possibile utilizzare metodi più robusti come il **5x2 cross-validated t-test**.

La procedura è la seguente:

1. Eseguire 2-fold cross-validation. Siano $\Delta_{1,1}$ e $\Delta_{1,2}$ le differenze di errore tra i due algoritmi nei due fold.
2. Ripetere questo processo 4 altre volte, ogni volta con una diversa suddivisione dei dati, ottenendo così un totale di 5 paia di differenze in totale. Denotiamo le differenze di errore come $\Delta_{i,1}$ e $\Delta_{i,2}$ per $i = 1, \dots, 5$.
3. Per ognuna delle 5 ripetizioni, calcoliamo la varianza delle due differenze come:

$$\sigma_i^2 = (\Delta_{i,1} - \bar{\Delta}_i)^2 + (\Delta_{i,2} - \bar{\Delta}_i)^2$$

$$\bar{\Delta}_i = \frac{\Delta_{i,1} + \Delta_{i,2}}{2}$$

4. Il test statistico è calcolato **usando la differenza tra i due fold della prima ripetizione e la media delle varianze calcolate nelle 5 ripetizioni**:

$$\tau = \frac{\Delta_{1,1}}{\sqrt{\frac{1}{5} \sum_{i=1}^5 \sigma_i^2}}$$

5. Sotto l'ipotesi nulla, questa statistica è distribuita secondo una **distribuzione di Student con 5 gradi di libertà**. La regola di decisione è la stessa di prima, ma usando il valore critico $t_{\alpha/2}$ con 5 gradi di libertà.

Nota

Il libro di testo suggerisce di usare la media $0.5(\Delta_{i,1} + \Delta_{i,2})$ al posto di $\Delta_{i,1}$ nella formula per τ andando in contrasto con *original paper by Dietterich*.

```

1 from scipy.stats import t
2
3 # ...
4
5 # Assume errors_A and errors_B are lists of error rates for learners A and B
6 # obtained from k-fold cross-validation
7
8 k = len(errors_A)
9 deltas = np.array(errors_A) - np.array(errors_B)
10 mu = np.mean(deltas)
11 sigma = np.std(deltas, ddof=1) # Sample standard deviation
12
13 tau = np.sqrt(k) * mu / sigma
14 t_crit = t.ppf(1 - alpha/2, k-1) # critical value for two-tailed test
15
16 # Decision rule
17 if abs(tau) > t_crit:
18     print("Reject null hypothesis H0: learners A and B have different
19           performance")
20 else:
21     print("Do not reject null hypothesis H0: learners A and B have similar
22           performance")
```

2.7.5 McNemar's Test

Assumiamo di avere due classificatori, A e B , e vogliamo confrontare le loro prestazioni usando il metodo hold-out. Addestriamo entrambi i classificatori sullo stesso set di addestramento e li valutiamo sullo stesso set di test, così facendo possiamo ottenere la seguente tabella di contingenza:

	Classifier A Correct	Classifier A Incorrect
Classifier B Correct	e_{11}	e_{10}
Classifier B Incorrect	e_{01}	e_{00}

L'idea è che, sotto l'**ipotesi nulla** che i due classificatori abbiano le stesse prestazioni, ci aspettiamo che il **numero di errori fatti da entrambi i classificatori sia simile**, cioè:

$$e_{01} \approx e_{10}$$

Il McNemar's test considera la variabile:

$$\tau_{\chi^2} = \frac{(|e_{10} - e_{01}| - 1)^2}{e_{10} + e_{01}}$$

che segue una distribuzione χ^2 con 1 grado di libertà sotto l'ipotesi nulla.

Regola di decisione: rifiutare l'ipotesi nulla H_0 a un livello di significatività α se:

$$\tau_{\chi^2} > \chi_{\alpha,1}^2$$

dove $\chi_{\alpha,1}^2$ è il valore critico della distribuzione χ^2 con 1 grado di libertà a livello di significatività α .

Tipicamente i valori critici più comuni sono:

$$\begin{array}{ccc} \alpha = 0.10 & \alpha = 0.05 & \alpha = 0.01 \\ \hline 2.706 & 3.841 & 6.635 \end{array}$$

2.7.6 Friedman Test

Entrambi i test di Student e McNemar sono progettati per confrontare due algoritmi di apprendimento su un singolo dataset. Tuttavia, in pratica, spesso vogliamo **confrontare più di due algoritmi su più dataset**.

In questo caso, possiamo confrontare ciascuna coppia di algoritmi utilizzando il test di Student o il test di McNemar, ma questo può essere computazionalmente costoso e può comportare un aumento del rischio di errori di tipo I (falsi positivi).

Il **test di Friedman** è un test di ranking che consente di confrontare più algoritmi su più set di dati utilizzando un singolo test. Se l'ipotesi nulla viene rifiutata, è necessario utilizzare un **test post-hoc** (come il test di Nemenyi) per identificare le coppie specifiche di algoritmi che presentano prestazioni significativamente diverse.

Assumendo di comparare gli algoritmi A , B , e C sui dataset D_1 , D_2 , D_3 e D_4 . Iniziamo usando la **validazione hold-out** o la **cross-validation** per **ottenere i tassi di errore** per ogni algoritmo su ogni dataset, poi **ordiniamo gli algoritmi** per ogni dataset in base al loro tasso di errore (il più basso ottiene il rango 1, il secondo più basso ottiene il rango 2, e così via), per ogni set di dati in cui agli algoritmi con lo stesso tasso di errore viene assegnato il rango medio.

Esempio

Supponiamo di avere i seguenti tassi di errore per i tre algoritmi sui quattro set di dati:

Error rates:

Dataset	Algorithm A	Algorithm B	Algorithm C
D_1	0.2	0.3	0.4
D_2	0.1	0.2	0.2
D_3	0.05	0.1	0.2
D_4	0.3	0.4	0.5

Ranks:

Dataset	Algorithm A	Algorithm B	Algorithm C
D_1	1	2	3
D_2	1	2.5	2.5
D_3	1	2	3
D_4	1	2	3
Averages	1	2.125	2.875

Sia k il **numero di algoritmi**, N il numero di dataset, e r_i il rango medio dell'algoritmo i su tutti i dataset. Per calcolare la statistica del test di Friedman, partiamo a calcolare la formula:

$$\tau_{\chi^2} = \frac{12N}{k(k+1)} \left(\sum_{i=1}^k r_i^2 - \frac{k(k+1)^2}{4} \right)$$

Il test statistico segue una distribuzione χ^2 con $k - 1$ gradi di libertà sotto l'ipotesi nulla che tutti gli algoritmi abbiano le stesse prestazioni.

Tuttavia, questa statistica **può essere utilizzata solo quando è di k grandi dimensioni**. Normalmente, la statistica viene corretta utilizzando la **statistica corretta di Iman e Davenport**:

$$\tau_F = \frac{(N-1) \cdot \tau_{\chi^2}}{N(k-1) - \tau_{\chi^2}}$$

Questa statistica corretta segue una distribuzione F con $k - 1$ e $(k - 1)(N - 1)$ gradi di libertà sotto l'ipotesi nulla.

Critical values for the F-test at $\alpha = 0.05$

N	$k = 2$	$k = 3$	$k = 4$	$k = 5$	$k = 6$	$k = 7$	$k = 8$	$k = 9$	$k = 10$
4	10.128	5.143	3.863	3.259	2.901	2.661	2.488	2.355	2.250
5	7.709	4.459	3.490	3.007	2.711	2.508	2.359	2.244	2.153
8	5.591	3.739	3.072	2.714	2.485	2.324	2.203	2.109	2.032
10	5.117	3.555	2.960	2.634	2.422	2.272	2.159	2.070	1.998
15	4.600	3.340	2.827	2.537	2.346	2.209	2.104	2.022	1.955
20	4.381	3.245	2.766	2.492	2.310	2.179	2.079	2.000	1.935

Critical values for the F-test at $\alpha = 0.1$

N	$k = 2$	$k = 3$	$k = 4$	$k = 5$	$k = 6$	$k = 7$	$k = 8$	$k = 9$	$k = 10$
4	5.538	3.463	2.813	2.480	2.273	2.130	2.023	1.940	1.874
5	4.545	3.113	2.606	2.333	2.158	2.035	1.943	1.870	1.811
8	3.589	2.726	2.365	2.157	2.019	1.919	1.843	1.782	1.733
10	3.360	2.624	2.299	2.108	1.980	1.886	1.814	1.757	1.710
15	3.102	2.503	2.219	2.048	1.931	1.845	1.779	1.726	1.682
20	2.990	2.448	2.182	2.020	1.909	1.826	1.762	1.711	1.668

2.7.7 Nemenyi Post-hoc Test

Se l'ipotesi nulla del test di Friedman viene rifiutata, possiamo procedere con un test post-hoc per identificare quali coppie di algoritmi hanno prestazioni significativamente diverse. Il **test di Nemenyi** è una scelta comune a questo scopo.

Il test di Nemenyi confronta le prestazioni di due classificatori osservando la differenza nei loro ranghi medi. Le prestazioni di due classificatori sono considerate significativamente diverse se i ranghi medi corrispondenti differiscono almeno della **differenza critica (CD)**, che si calcola come:

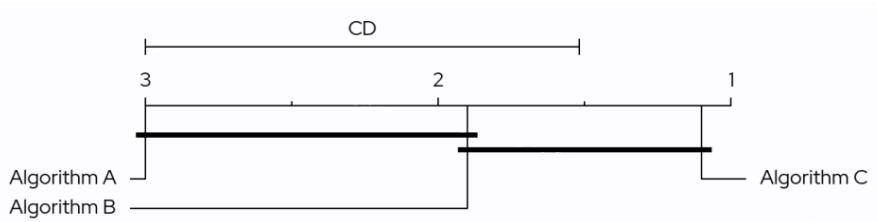
$$CD = q_\alpha \sqrt{\frac{k(k+1)}{6N}}$$

dove k è il numero di algoritmi, N il numero di dataset, e q_α è il valore critico della distribuzione di Studentizzata al livello di significatività α .

Si ritiene che due algoritmi abbiano prestazioni significativamente diverse se la differenza assoluta dei loro ranghi medi è maggiore del CD:

$$|r_i - r_j| > CD$$

2.7.8 Critical difference diagram



2.8 Bias Variance decomposition

La **decomposizione Bias-Varianza** è un modo per comprendere le fonti di errore in un modello di apprendimento automatico.

Per diversi set di apprendimento, il risultato è spesso diverso, sebbene i campioni siano estratti dalla stessa distribuzione. La decomposizione Bias-Varianza ci aiuta a capire perché questo accade scomponendo l'errore in tre componenti:

1. **Bias:** l'errore dovuto a ipotesi errate nel modello di apprendimento, che può portare a dell'uniderrfitting.
2. **Variance:** l'errore dovuto alla sensibilità del modello a piccole fluttuazioni nel set di addestramento, che può portare a un overfitting.
3. **Irreducible error:** l'errore che non può essere ridotto da nessun modello, spesso dovuto al rumore nei dati.

Consideriamo un problema di **regressione** e sia X un **campione di prova**, y_D è l'etichetta associata al dataset D^1 , y l'etichetta della ground truth per X , e $f(X; D)$ la precisione del modello f addestrato su D .

Expected prediction: la previsione prevista per il modello su X è data da:

$$\bar{f} = \mathbb{E}_D[f(X; D)]$$

Varianza:

$$var = \mathbb{E}_D[(f(X; D) - \bar{f})^2]$$

Rumore:

$$\epsilon^2 = \mathbb{E}_D[(y_D - y)^2]$$

Bias: cioè, la differenza tra la previsione prevista e l'etichetta reale:

$$bias = (\bar{f} - y)^2$$

Dimostrazione

Per facilitare la discussione, supponiamo che l'aspettativa di rumore sia zero, ovvero, $\mathbb{E}_D[y_D - y] = 0$.

L'MSE previsto del modello può essere scritto come:

$$\begin{aligned} E(f; D) &= \mathbb{E}_D[(f(\mathbf{x}; D) - y_D)^2] \\ &= \mathbb{E}_D[(\underbrace{f(\mathbf{x}; D)}_a - \bar{f} + \underbrace{\bar{f} - y_D}_b)^2] = \mathbb{E}_D[(a + b)^2] \\ &\quad \text{since } y_D \perp f_D \\ &= \mathbb{E}_D[(f(\mathbf{x}; D) - \bar{f})^2] + \mathbb{E}_D[(\bar{f} - y_D)^2] + \cancel{2\mathbb{E}_D[(f(\mathbf{x}; D) - \bar{f})(\bar{f} - y_D)]} \\ &= \mathbb{E}_D[(f(\mathbf{x}; D) - \bar{f})^2] + \mathbb{E}_D[(\bar{f} - y_D)^2] \\ &= var + \mathbb{E}_D[(\bar{f} - y + y - y_D)^2] \\ &\quad \text{since } \mathbb{E}_D[y_D - y] = 0 \Rightarrow \mathbb{E}_D[y_D] = y \\ &= var + \cancel{\mathbb{E}_D[(\bar{f} - y)^2] + \mathbb{E}_D[(y - y_D)^2]} + \cancel{2\mathbb{E}_D[(\bar{f} - y)(y - y_D)]} \\ &= var + bias^2 + \varepsilon^2 \end{aligned}$$

Bias misura la differenza tra la previsione attesa dell'algoritmo di apprendimento e l'etichetta di ground-truth, che riflette la **capacità di adattamento dell'algoritmo di apprendimento**.

Varianza misura quanto le prestazioni di apprendimento cambiano a causa delle variazioni nel set di addestramento, riflettendo l'**impatto delle fluttuazioni dei dati sul risultato dell'apprendimento**.

Noise rappresenta il limite inferiore dell'errore di generalizzazione atteso che può essere raggiunto da **qualsiasi algoritmo di apprendimento** per il compito dato.

Un altro modo di vedere la cosa è che la decomposizione bias-varianza ci dice che la **performance della generalizzazione** è determinata congiuntamente da:

- la capacità dell'algoritmo di apprendimento
- sufficienza dei dati
- e l'errore intrinseco del problema di apprendimento

Per ottenere eccellenti prestazioni di generalizzazione, un modello necessita di un **piccolo bias** (per adattarsi adeguatamente ai dati) e di una **piccola varianza** (per ridurre al minimo l'impatto delle fluttuazioni dei dati).

2.8.1 Bias-Variance Dilemma

Il dilemma bias-varianza si riferisce al compromesso tra bias e varianza in un algoritmo di apprendimento: in generale gli algoritmi con un basso bias tendono ad avere un'alta varianza, e viceversa.

