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.

É 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

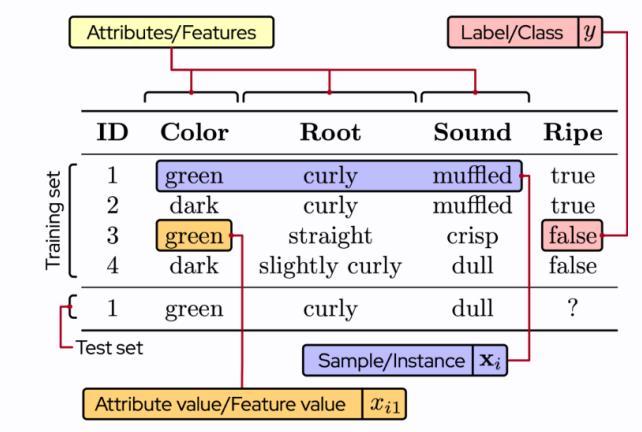
Ora troviamo un melone con le seguenti caratteristiche e vogliamo sapere se è maturo:

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

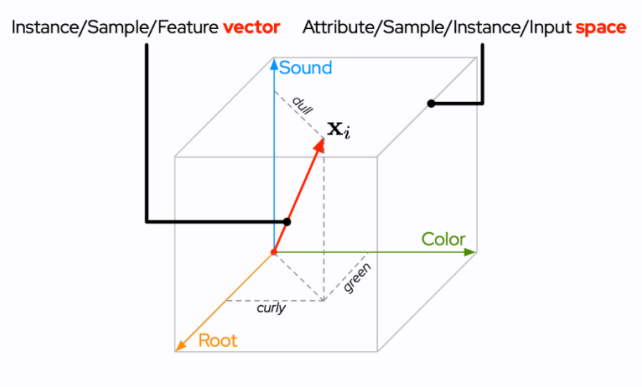


Le colonne della tabella sono chiamate **attributi** o **features** dell'esempio, l'elemento che ci dice come risolvere il task è chiamato **etichetta** o **classe** dell'esempio, tipicamente la classe viene denotata con una y_i minuscola.

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 performando.



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, $\mathbf{x}_i = [x_{i1}, x_{i2}, \ldots, x_{id}]^\top$
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, $D = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)\}$
n	number of instances in the dataset
h	model/hypothesis, a function $h : \mathcal{X} \mapsto \mathcal{Y}$ that maps instances to labels
$\mathfrak{L}$	Learning algorithm $\mathfrak{L} : \mathcal{P}(\mathcal{X} \times \mathcal{Y}) \to \mathcal{H}$, 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 pesco 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**: $P((X_i, y_i) \mid (X_1, y_1), ..., (X_{i-1}, y_{i-1})) = P((X_i, y_i)) \quad \forall i$
- **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 di cupcake. Ogni cupcake è fatto:

- usando la **stessa ricetta** (identicamente distribuita)
- e **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** $\rightarrow$ non distribuite in modo identico.
- Cuoci i cupcake nello **stesso stampo e l'impasto si mescola** $\rightarrow$ 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 funziona $h : \chi \mapsto \{0, 1\}$.

Esempio

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

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

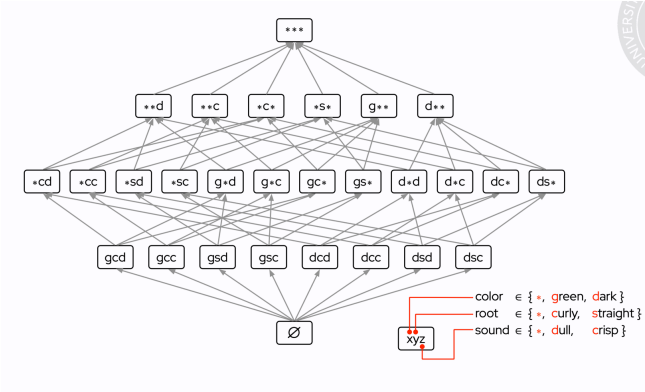
Al posto dei punti interrogativi possiamo mettere un valore oppure un **asterisco** * per dire che **non interessa il valore specifico**.
In questo caso il melone sarà maturo se il suono è dull il resto non mi interessa.

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

1.7.3 Spazio delle ipotesi

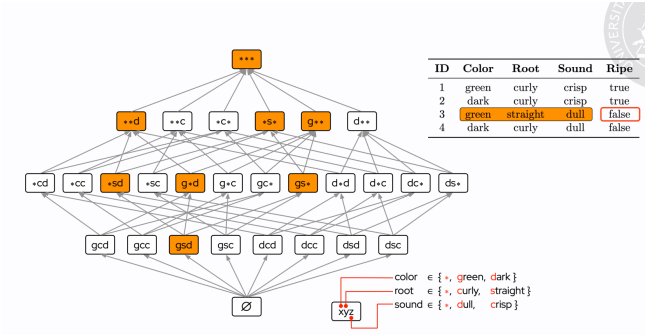
Lo **Spazio delle ipotesi** è l'insieme di tutte le ipotesi (tutti i possibili modelli) che posso apprendere utilizzando uno specifico algoritmo di apprendimento. [Nell'esempio sopra citato è l'insieme di tutte le formule booleane che posso ottenere].

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.

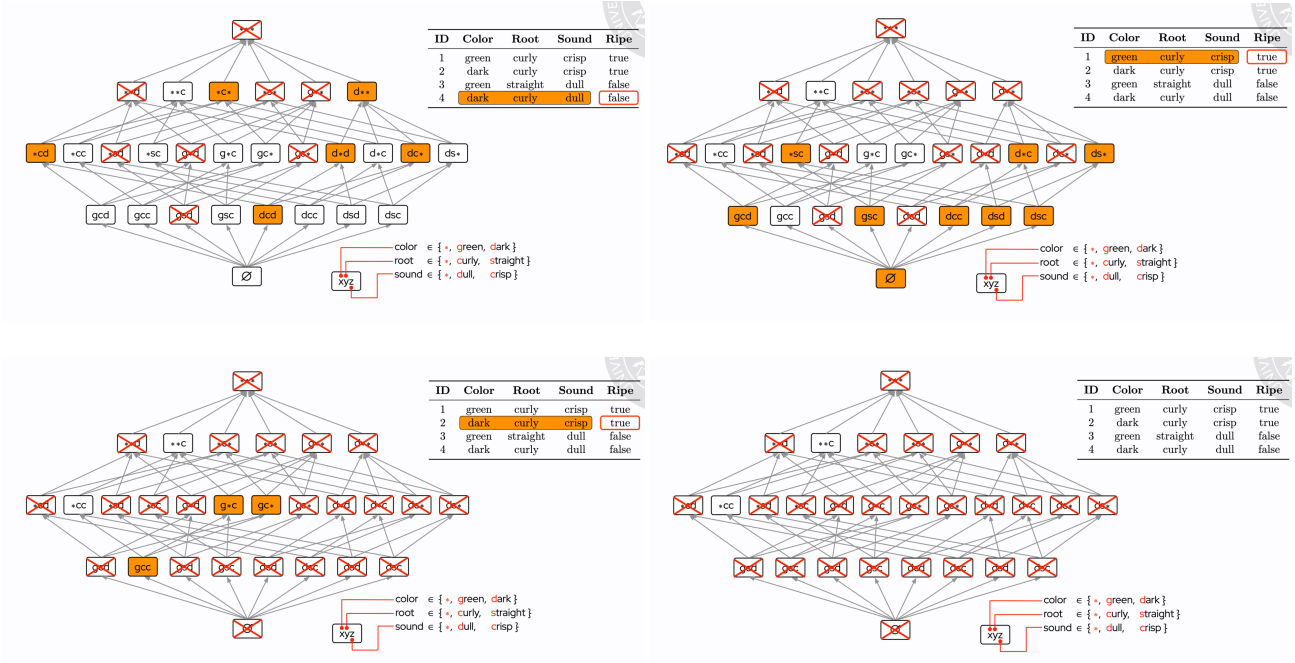


1.7.4 Spazio delle versioni

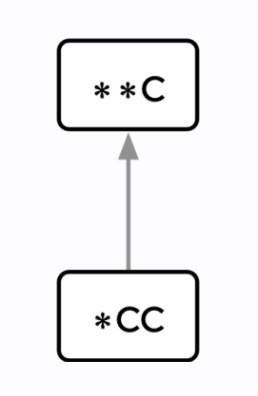
Lo **spazio delle versioni** è un **sottoinsieme dello spazio delle ipotesi** che è consistente con tutti gli **esempi di training**. Sostanzialmente prendo solo gli esempi che danno una risposta giusta [nel caso dei meloni se trovo un esempio con suono *dull* ma non maturo lo andrò a scartare].



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:

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

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 $Color = green,\ Root = straight,\ Sound = crisp$, 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:

- $P(h \mid X, \mathcal{L}_a)$: **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.

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

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.

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

Nella diapositiva precedente, il termine $\frac{1}{2}2^{|\chi|}$ 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:

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

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'**accuratezza** è calcolata:

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

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**.

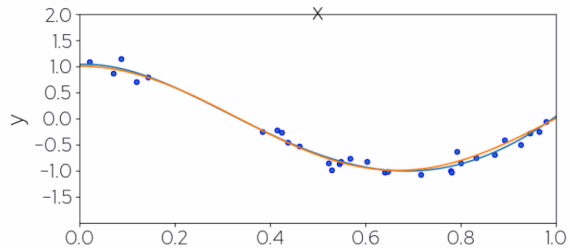
Nota

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

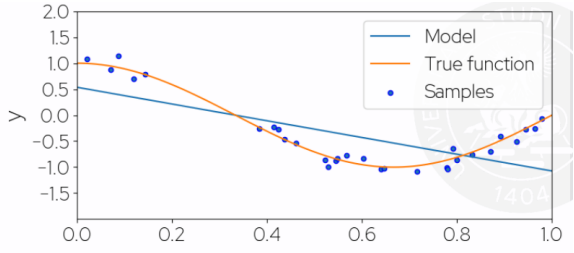
Durante l'addestramento non possiamo calcolare direttamente il **generalization error**, perché non abbiamo accesso ai nuovi esempi, quindi lo approssimiamo minimizzando il **training error**.

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 techiniques** o **early stopping**.

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

2.3 Metodi di valutazione

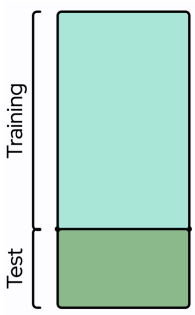
Possiamo valutare le prestazioni di un modello misurandone l'**errore di generalizzazione** su nuovi campioni. Tuttavia, non abbiamo accesso a nuovi campioni durante l'addestramento.

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:

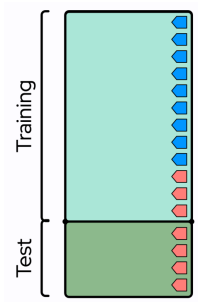
- **Training set**: dati che darò in pasto all'algoritmo per l'apprendimento.
- **Test set**: dati che utilizzerò per valutare.



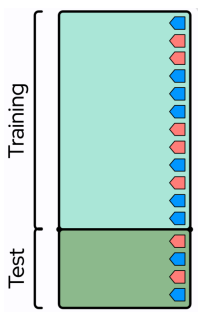

```
from sklearn.model_selection import train_test_split
...

#Assume X,y contains the dataset

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.

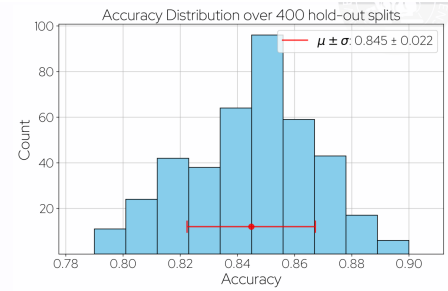


- **Stratified split** Si mescolano i dati prima di estrarli poi procedo a prendere i primi n come training e il restante come test.



```python
from model_selection import train_test_split
...

#Assume X,y contains the dataset

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
    =0.3, shuffle=True, #(default) sufficient for large datasets
     stratify=y, #usefule for small and/or imbalanced datasets
     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.

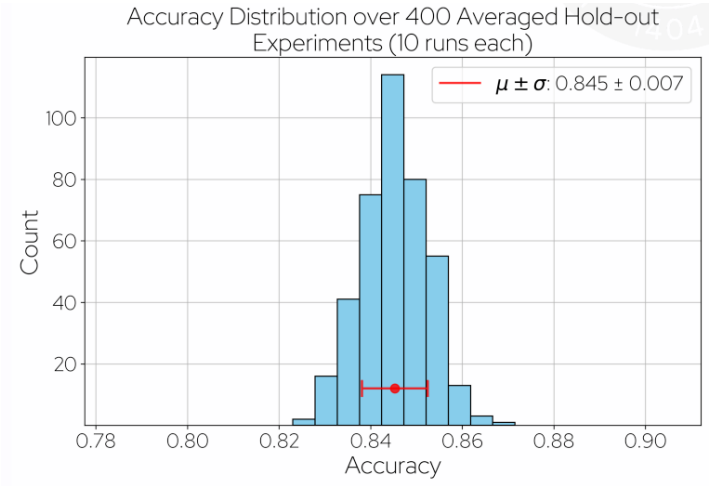
La media di n r.v. **indipendenti**, ognuno con media μ e varianza σ^2 è una nuova r.v. con:

Media

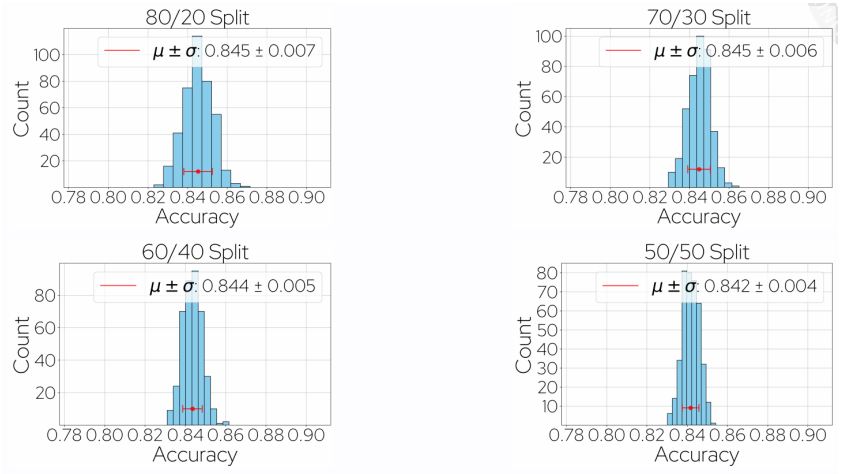
$$\frac{1}{n}\sum_{i=1}^{n} E_i = \frac{1}{n}\sum_{i=1}^{n} \mu = \mu$$

Varianza

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

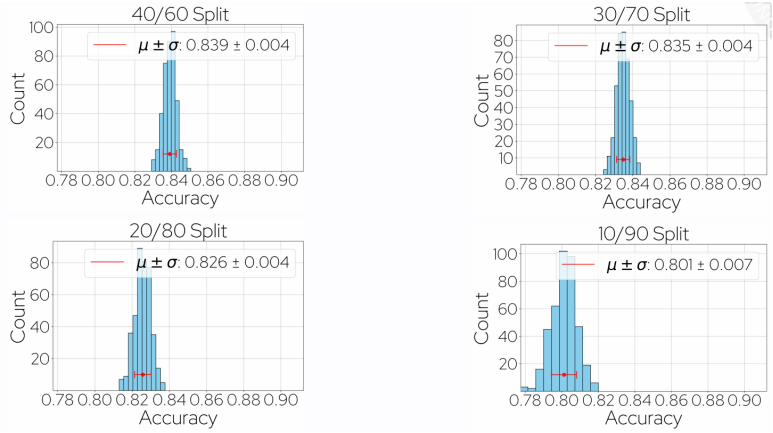


Modifica del rapporto di divisione Una cosa che influisce sull'ampiezza della campana è come stiamo separando il training set dal test set.



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.

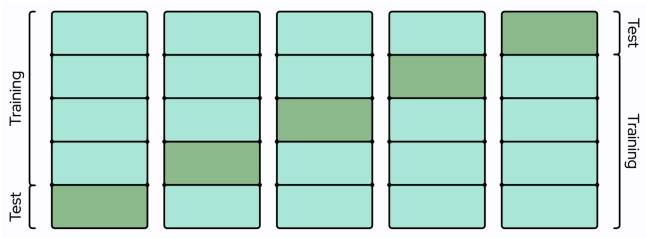


Continuando ad aumentare la percentuale di esempi per il test set la media andrà sempre peggiorando ma la varianza non migliorerà all'infinito anzi ricomincerà ad aumentare.

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

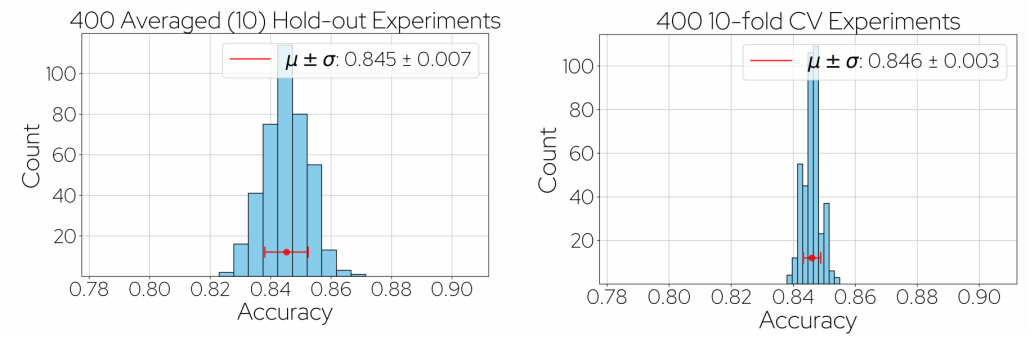
```python
from sklearn.model_selection import KFold
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
import numpy as np

# Assume X,y contains the dataset

clf = LogisticRegression()

kf = KFold(n_splits=5, shuffle=True, rando_state=42)
accuracies = []
for train_index, test_index in kf.split(X):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    clf.fit(X_train, y_train)
    score = accuracy_score(y_test, clf.predict(X_test))
    accuracies.append(score)

print(f"x-val accuracy: {np.array(accuracies).mean()}")
```

Per sapere in modo veloce l'errore di generalizzazione.

```python
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression

#Assume X,y contains the dataset

clf = LogisticRegression()
scores = cross_val_score(clf, X, y, cv=5)
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

AAUT-2025-I2 : 22:00