

# Quale intelligenza artificiale a scuola? Spunti di riflessione di fronte alla sfida del Machine Learning

Davide Passaro, Liceo B. Russell di Roma

## 1. Introduzione

Negli ultimi venti anni l'uso di sistemi di intelligenza artificiale è diventato pervasivo nella società. L'aumento delle capacità dei computer e, in particolare, dei sistemi di parallelizzazione di calcolo per mezzo delle GPU (Graphic Processing Unit) ha permesso lo sviluppo e la diffusione di quello che viene normalmente indicato con il termine di *Machine Learning* (ML), in italiano *apprendimento automatico* (Bishop, C. 2006) e in particolare del *Deep Learning* (Goodfellow et al. 2016). La creazione di diverse architetture di reti neurali profonde ha permesso di ottenere dei risultati con una accuratezza anche superiore a quella umana.

Più recentemente la creazione degli LLM (*Large Language Model*) nati a seguito dell'introduzione dell'architettura dei *Trasformers* (Vaswani, A. et al. 2017) ha permesso la creazione di *Chatbot* in grado di parlare e interagire con gli utenti.

Per quanto già prima il Machine Learning fosse pervasivo della vita delle persone (basti pensare al diffuso uso che viene fatto dell'apprendimento automatico da parte dei social network e di aziende come Google, Apple, Netflix e Amazon), il fatto che chatbot come ChatGPT o Claude o Gemini siano in grado di parlare, rispondere e svolgere compiti di elevata complessità mai vista prima ha fatto comprendere a tutta l'opinione pubblica quanto gli sviluppi dell'apprendimento automatico possano potenzialmente cambiare l'attuale società.

In questo articolo si cercherà di motivare perché la scuola che si rivolge agli studenti - futuri cittadini - non può rimanere ai margini e ignorare le questioni relative all'introduzione e all'uso di questi sistemi di Machine Learning. In particolare, si ritiene che, in base all'attuale struttura della scuola secondaria in cui l'insegnamento dell'informatica è svolto in una esigua minoranza di indirizzi, questo

compito debba essere principalmente svolto dagli insegnanti di matematica.

Si presenteranno, quindi, delle proposte di percorsi da realizzarsi nel corso del quinquennio della scuola superiore.

## **2. Il problema dell'oracolo**

Se ormai è diventata a tutti evidente la diffusa presenza dei sistemi di ML nella società, meno evidenti sono le problematiche che un uso inconsapevole e acritico di questi strumenti può comportare.

In realtà si potrebbe dire che questa consuetudine di usare degli strumenti senza sapere come questi funzionino sia pervasiva della nostra società. Quanti saprebbero aprire uno smartphone e spiegare come funziona? Quanti sarebbero in grado di spiegare il meccanismo alla base del funzionamento del motore di un'automobile?

Che differenza c'è con l'Intelligenza Artificiale?

La differenza, forse sottile ma essenziale, è che il ML è in grado di suggerire risposte per noi, di guidare le nostre conoscenze, il nostro lavoro, di influenzare le nostre scelte.

Se, come pare, in misura sempre maggiore ci rivolgeremo a sistemi di Intelligenza Artificiale allenati su enormi dataset senza capirne il funzionamento, agiremo allo stesso modo degli antichi greci che si rivolgevano all'oracolo di Delfi.

L'Intelligenza Artificiale rischia di diventare un oracolo che offre risposte a persone che non sono in grado di capire i limiti e i rischi di questi potenti strumenti e che, loro malgrado, inizieranno a prendere decisioni sulla propria vita basandosi su uno "strumento magico" a cui "crederanno" in modo del tutto simile a quello che gli antichi greci facevano con l'oracolo di Delfi.

Se è vero che una parte degli studenti avranno modo di approfondire il funzionamento dell'apprendimento automatico all'università, è anche vero che per la maggioranza degli altri, che non si iscriveranno ad una facoltà scientifica che tratta questi temi, i fondamenti del funzionamento di questi sistemi resteranno ignoti. Si ritiene invece che, in una società democratica, sia essenziale che i cittadini siano messi nelle condizioni di comprendere in modo critico il mondo in

cui vivono e la sua tecnologia, specie quando questa può profondamente influenzare non solo le abitudini (pensiamo all'introduzione della luce elettrica o dei mezzi di trasporto a motore) ma anche il modo di pensare e le decisioni che verranno prese.

### **3. Il primo passo**

Se si concorda sull'idea di scuola come luogo di sviluppo del sapere critico e sull'importanza che vengano offerti gli strumenti di base per comprendere il funzionamento del ML, è chiaro che un ruolo essenziale è attribuito ai docenti di matematica, unica disciplina presente in qualunque indirizzo scolastico.

Il primo passo, che, come insegnanti di questa disciplina, è necessario fare, è comprendere noi per primi i più recenti sviluppi del ML.

La letteratura per farlo è assolutamente vasta e in rete esistono innumerevoli risorse che si possono utilizzare.

Qui si propongono alcune letture che, oltre ad essere assolutamente valide, si ritiene possano essere un buon punto di partenza per chi, pur avendo una laurea scientifica, non ha incontrato questi temi nel corso della sua formazione.

Il primo libro che si suggerisce è *“Perché le macchine imparano”* (Ananthaswamy 2024): è un testo divulgativo dove, però, gli aspetti matematici non sono tralasciati e vengono spiegati abbastanza dettagliatamente oltre che con sufficiente rigore.

Il secondo testo, un classico per chiarezza e completezza, è *“An introduction to statistical learning”* (Hastie et al. 2023)

### **4. Il secondo passo: il problema del tempo**

Pur concordando sull'idea di fondo della necessità che gli studenti non usino gli strumenti dell'intelligenza artificiale come un oracolo, una delle principali obiezioni che si potrebbero fare a questa proposta di introdurre le idee di base che hanno portato allo sviluppo di questi algoritmi, è che nella scuola attuale la risorsa del tempo è molto scarsa. Manca il tempo per riuscire ad affrontare con un adeguato

spazio gli argomenti che si ritiene essere fondamentali in coerenza con le indicazioni nazionali.

Inoltre, se si insegna nel liceo scientifico, è fortemente sentita l'esigenza di portare gli studenti a poter rispondere al più ampio spettro possibile di domande poste durante la prova scritta dell'esame di maturità.

Chi scrive vive in prima persona queste difficoltà.

Il secondo passo che è necessario fare in questi casi è:

- sfruttare le possibilità concesse dall'autonomia scolastica e utilizzare le ore aggiuntive della sperimentazione/curvatura per realizzare questo percorso;
- sfruttare il supporto, laddove presente come disciplina, del docente di informatica con cui potrebbe essere condiviso parte di questo progetto;
- decidere di dedicare meno tempo a parti di programma che si ritengono meno essenziali.

Soprattutto l'ultima scelta non è semplice e certamente comporta delle criticità; è, però, essenziale per intraprendere questo percorso in attesa di eventuali modifiche delle indicazioni nazionali.

Per rendere questo percorso realizzabile, comunque, si propone che sia suddiviso nel corso del quinquennio in modo che copra parte degli argomenti che normalmente vengono svolti: informatica, statistica, probabilità.

### **5. Il terzo passo: superare lo scoglio dell'argomento nuovo**

Per intraprendere il percorso che vedremo è però necessario un ultimo passo: superare la paura di dover affrontare un argomento nuovo su cui ci si sente poco preparati perché non affrontato all'università.

È comune, infatti, specie nei docenti più scrupolosi e attenti, cercare di evitare di introdurre argomenti su cui ci si sente meno preparati e su cui non è presente già una consolidata tradizione didattica nonché

un ampio materiale fornito (per esempio dalle case editrici) a supporto.

Se da una parte questo è condivisibile, dall'altra è necessario osservare che potrebbe essere formativo esplicitare agli studenti che questo argomento trattato è per il docente nuovo, che anche lui, come professionista, studia, si aggiorna e, visto che l'argomento è inedito (in realtà nuovo non solo per lui ma proprio per tutti) è normale che spesso non si sappiano dare risposte.

L'atteggiamento del docente che si mette in gioco può non solo essere un incentivo per la classe ma mostra come le discipline scientifiche siano intrinsecamente aperte ed in espansione. Il docente e i suoi studenti, messi di fronte ad un nuovo problema, simulano esattamente il lavoro di un ricercatore che affronta la scoperta di un nuovo campo di studio.

## **6. Proposte per il quinquennio di scuola superiore**

In questo paragrafo cercheremo di dettagliare la proposta suddividendola in primo biennio, secondo biennio e ultimo anno.

### *- Primo biennio*

Nel primo biennio la proposta è innanzitutto quella di declinare la voce "informatica" dell'insegnamento della matematica, scegliendo di proporre un percorso di apprendimento incentrato sull'insegnamento di un *linguaggio di programmazione*. Fra i possibili linguaggi si propone di introdurre Python, perché è un linguaggio ritenuto più semplice (Passaro 2016) e dotato di librerie apposite per lo sviluppo di algoritmi di intelligenza artificiale. Per la relativa facilità di apprendimento, l'uso di Python permette di concentrarsi sulla parte algoritmica e computazionale mettendo in secondo piano gli aspetti più strettamente relativi alle difficoltà della programmazione.

L'altra scelta necessaria è quella di affrontare la parte *delle indicazioni nazionali relativa alla statica e alla probabilità* anche avvalendosi dell'uso della programmazione.

Il percorso potrebbe (o meglio, dovrebbe) partire dalla proposizione di un problema reale.

In rete, infatti, sono disponibili una ampia serie di dataset; un sito che ne raccoglie diversi è l'UCI Machine Learning Repository (disponibile al seguente link: <https://archive.ics.uci.edu/>).

Ogni dataset presenta un problema specifico da risolvere e per meglio comprendere i dati gli studenti saranno invitati ad eseguire una analisi esplorativa del dataset per arrivare a formulare delle ipotesi di soluzione dello stesso.

Questa attività, che, partendo da un problema reale, porta gli studenti a formulare delle ipotesi, ha un grande valore didattico.

All'interno di questo percorso si potrebbero introdurre alcuni dei concetti chiave dell'apprendimento automatico:

- l'idea di *apprendimento supervisionato* (ovvero quell'apprendimento in cui si parte da un insieme di dati di cui è nota la risposta) o *non supervisionato*
- i concetti di *classificazione/regressione/clustering* dei dati
- l'idea di *suddivisione del dataset in training e test* per verificare
- le *misure di performance*, ovvero come misurare i risultati degli algoritmi di classificazione/regressione sviluppati.
- introduzione di semplici algoritmi di classificazione come, per esempio, l'algoritmo dei *k-vicini* (*k-nearest neighbors* o *k-NN*) o l'algoritmo *naive bayesiano*.

- *Secondo biennio e ultimo anno*

Nell'ultimo triennio potrebbero essere introdotti altri semplici algoritmi di classificazione, come per esempio gli *alberi decisionali*, e una volta che sono note le funzioni esponenziali si potrebbe introdurre l'idea di rete neurale.

Nel quinto, affrontato l'argomento della derivata di funzione composta, si potrebbe introdurre *l'algoritmo di ottimizzazione della discesa del gradiente e della retropropagazione*.

## 7. Semplici algoritmi di classificazione e misure di performance

In questo paragrafo vogliamo offrire degli spunti applicativi che detaglino le proposte elencate nel paragrafo precedente. Ovviamente è disponibile una ampia letteratura su questi temi e il nostro intento è mostrare semplicemente alcuni algoritmi e la loro effettiva applicabilità nelle classi.

Un semplice algoritmo (forse il più semplice) che necessita solo della conoscenza della distanza euclidea in più dimensioni è quello indicato con il nome *k-Nearest Neighbors* (“i  $k$  vicini più prossimi”), è uno dei metodi più semplici e intuitivi del machine learning. È un algoritmo supervisionato che sfrutta il dataset di allenamento di cui è nota l'attribuzione delle classi.

Per classificare un nuovo punto,  $k$ -NN:

- calcola la distanza (euclidea, ma volendo si possono usare anche altre distanze) tra quel punto e tutti i punti del dataset di addestramento.
- Seleziona i  $k$  punti più vicini (neighbors).
- Guarda a quale classe appartiene la maggioranza di quei  $k$  vicini.
- Assegna la classe più frequente al nuovo punto.

La distanza fra due punti nel piano è un argomento che viene normalmente svolto a livello di biennio e può essere facilmente generalizzato in  $n$  dimensioni.

La scrittura con il linguaggio Python di questo algoritmo è estremamente semplice e in rete, ovviamente, esistono codici già pronti da utilizzare.

Una possibile implementazione è la seguente:

```
import numpy as np
from collections import Counter
def knn_predict(X_train, y_train, X_test, k=3):
    X_train = np.array(X_train)
    y_train = np.array(y_train)
    X_test = np.array(X_test)
    predictions = []
    for x in X_test:
```

```
# Distanze euclidee da tutti i punti del training
distances = np.sqrt(np.sum((X_train - x) ** 2, axis=1))
k_indices = np.argsort(distances)[:k]
k_labels = y_train[k_indices]
most_common = Counter(k_labels).most_common(1)[0][0]
predictions.append(most_common)
return np.array(predictions)
```

In realtà esistono già delle librerie di Python che hanno l'algoritmo *k-NN* implementato ed è sufficiente utilizzare queste librerie.

Didatticamente può essere utile mostrare l'implementazione degli algoritmi più semplici ed eventualmente utilizzare le librerie con gli stessi algoritmi già implementati per fare un confronto.

Qui di seguito inseriamo un esempio di codice che utilizza la libreria *scikit-learn*, applica l'algoritmo *k-NN* su un dataset di esempio diviso in training e test (in questo è il dataset "iris") e ne restituisce le previsioni.

```
# Esempio di classificazione k-NN usando libreria scikit-learn
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
# Caricamento un dataset di esempio (Iris)
iris = load_iris()
X = iris.data
y = iris.target
#Divisione dei dati in training e test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
# Normalizzazione le variabili (importante per k-NN che usa distanza euclidea)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
# Creazione e uso del classificatore k-NN
k = 5 # numero di vicini
knn = KNeighborsClassifier(n_neighbors=k)
```

```
knn.fit(X_train, y_train)
# Predizione sui dati di test
y_pred = knn.predict(X_test)
```

Poiché il  $k$ - $NN$  utilizza la distanza euclidea e, in generale, le variabili potrebbero essere molto diverse fra loro come ordine di grandezza, è buona norma normalizzare le variabili.

Un altro algoritmo molto semplice è quello indicato con il nome *Naive Bayes*. È chiamato “naïve” (“ingenuo”) perché assume che tutte le variabili siano indipendenti tra loro. Questa è un’ipotesi molto semplificata che nella realtà spesso non è vera, ma che funziona sorprendentemente bene in molti casi pratici. Il teorema di Bayes viene normalmente introdotto anche al biennio. Nel caso di una sola variabile  $X$  si può scrivere che la probabilità di appartenere alla classe  $C$  data una osservazione  $x$  della variabile  $X$  è:

$$p(C|x) = p(x|C) \cdot p(C) / p(x).$$

Nei casi reali dove sono coinvolte più variabili si tratta di spiegare agli studenti l’approccio “naïve” ovvero che date le variabili  $X_1, X_2, \dots, X_n$ , la probabilità che una  $n$ -pla  $x_1, x_2, \dots, x_n$  appartenga ad una classe  $C$  può essere scritta nel seguente modo:

$$p(C|x_1, x_2, \dots, x_n) = p(x_1, x_2, \dots, x_n | C) \cdot p(C) / p(x_1, x_2, \dots, x_n).$$

E, seguendo l’approccio naïve la  $p(x_1, x_2, \dots, x_n | C)$  può essere stimata assumendo che siano indipendenti:

$$p(x_1, x_2, \dots, x_n | C) = p(x_1 | C) \cdot p(x_2 | C) \cdot p(x_n | C).$$

Anche questo algoritmo può essere semplicemente implementato in Python e ne esistono in rete innumerevoli versioni che si possono scaricare ed utilizzare. Ovviamente la stima delle probabilità delle singole variabili dipende dal tipo di problema in esame.

Una volta attribuita la classe è necessario stabilire delle misure di performance dell’algoritmo di classificazione.

La premessa fondamentale è quella di dividere il dataset di cui è disponibile la “risposta” in due gruppi: il *training* e il *test*.

Con il primo gruppo di dati si allena (da cui training) l'algoritmo e con il secondo gruppo lo si testa.

L'idea è quella che un buon algoritmo deve mantenere buone performance non solo sui dati con cui è stato allenato ma anche sui dati di test.

Il rischio altrimenti è cosiddetto “*overfitting*” ovvero l'adattare troppo il modello ai dati di allenamento non permettendo così al modello di essere generale e quindi funzionare anche su nuovi dati. Per vedere quanto l'algoritmo restituisce una risposta corretta, si deve ovviamente confrontare la risposta data dall'algoritmo con quella disponibile.

Nel caso di una classificazione binaria si ottiene una matrice 2x2 in cui i quattro valori della matrice corrispondono alle classi TP (vero positivo), FN (falso negativo), FP (falso positivo) e TN (Vero negativo).

	<b>Predetto: Positivo</b>	<b>Predetto: Negativo</b>
<b>Reale: Positivo</b>	TP (True Positive)	FN (False Negative)
<b>Reale: Negativo</b>	FP (False Positive)	TN (True Negative)

Tabella 2x2 detta *matrice di confusione* del classificatore binario

Ovviamente l'obiettivo di un algoritmo classificatore è quello di ottenere una matrice che sia il più diagonale possibile, ovvero in cui i valori più alti siano TP e TN (veri positivi e veri negativi)

A partire dalla matrice precedente è possibile calcolare le seguenti misure di performance:

*Accuracy (accuratezza):*

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

*Precision (precisione):*

$$\text{Precision} = \frac{TP}{(TP+FP)}$$

*Recall (sensibilità o vero positivo rate)*

$$Recall = TP/(TP+FN)$$

*F1-score (media armonica di precision e recall)*

$$F1 = 2 \times Precision \times Recall / (Precision + Recall)$$

Ovviamente un buon algoritmo di classificazione dovrebbe avere un alto valore di accuratezza non solo sui dati di allenamento ma anche su quelli di test. In base al contesto si potrà essere interessati ad ottenere elevati valori di *precisione* o di *sensibilità*. Un esempio di codice Python per calcolare l'accuratezza è il seguente:

```
def accuracy_score(y_true, y_pred):
    """ Calcola l'accuratezza del modello prendendo in input: y_true : array con
    le etichette vere y_pred : array con le etichette predette """
    y_true = np.array(y_true)
    y_pred = np.array(y_pred)
    return np.sum(y_true == y_pred) / len(y_true)
```

Si osserva che, anche in sede di biennio, è possibile parlare di matrici che vengono introdotte in genere per affrontare i sistemi lineari con Cramer.

Se pure in generale, un percorso così illustrato permette di presentare agli studenti gli aspetti essenziali del Machine Learning.

## 8. Ulteriori argomenti: il clustering e regressione

Anche se non sono stati inclusi nella nostra proposta di percorso, segnaliamo che si può introdurre non solo l'idea di classificazione ma anche di clustering e di regressione.

Un metodo di clustering serve a raggruppare dati simili tra loro senza conoscere le "etichette" ed è, quindi, un *algoritmo non supervisionato*. L'idea è quella di creare degli algoritmi in grado di raggruppare elementi omogenei in un insieme di dati, ad esempio facendo dividere *automaticamente* gli elementi del dataset in  $n$  insiemi.

Un esempio di algoritmo è il *k-means*, che utilizza nuovamente il concetto di distanza fra punti in  $n$  dimensioni, tramite procedura

iterativa. Il  $k$ -means cerca di dividere i dati in un certo numero di gruppi, indicato con  $k$ , in modo che i punti che appartengono allo stesso gruppo siano simili tra loro e che i punti appartenenti a gruppi diversi siano differenti. In particolare, si seguono i seguenti passaggi:

- Scelta di  $k$ : si sceglie il numero di cluster,  $k$ . È un valore deciso dall'utente, per esempio  $k = 5$  se vogliamo cinque gruppi.
- Inizializzazione: l'algoritmo sceglie a caso  $k$  punti del dataset, detti *centroidi*, cioè i centri provvisori dei cluster.
- Assegnazione dei punti ai centroidi: ogni punto del dataset viene assegnato al centroide temporaneo più vicino calcolando, di solito, la distanza euclidea.
- Aggiornamento dei centroidi: una volta che tutti i punti sono stati assegnati ad un gruppo, si calcola la nuova posizione di ciascun centro; il nuovo centro del cluster è la media (mean) di tutti i punti che ne fanno parte (da qui il nome *k-means*).
- Ripetizione: i passi di assegnazione e aggiornamento vengono ripetuti finché i centroidi non cambiano più (cioè, il sistema si è stabilizzato), oppure si raggiunge un numero massimo di iterazioni. A quel punto l'algoritmo si dice "convergente" e i gruppi trovati sono i risultati finali.

Un ulteriore problema di previsione utilizzato in modelli di apprendimento automatico e che si potrebbe affrontare in un biennio è quello della *regressione*. Nella *classificazione* il modello di previsione risponde con una classe o una categoria; nella *regressione* invece il modello produce una stima numerica, individuando la retta (di *regressione lineare*) o la funzione che meglio rappresenta la relazione tra le variabili. Di tutti gli argomenti di questo articolo, la regressione è quello più vicino alla pratica didattica, dato che per esempio è abbastanza comune affrontare il tema del *fit* dei dati negli esperimenti in Fisica. Poiché è una attività abbastanza diffusa non verrà discussa in questo articolo ma è comunque bene ricordare che anche queste tipologie di attività di analisi dei dati sono da considerarsi come facenti parti di un percorso di introduzione al Machine Learning.

## 8. Primo percorso al Triennio: Alberi decisionali

In questo paragrafo affrontiamo alcuni dei possibili argomenti che potrebbero essere introdotti in un triennio di scuola superiore. Per l'ampiezza di questo argomento ci limiteremo, in questo caso, a delineare delle possibili strade rimandando il lettore ad un approfondimento bibliografico. La descrizione degli aspetti teorici di ciascuno dei metodi richiederebbe uno spazio ben più ampio di quello disponibile. Si ritiene, però, che con gli adeguati riferimenti bibliografici e l'uso delle risorse disponibili in rete sia possibile per un insegnante acquisire la conoscenza di base per introdurre i fondamenti di questi argomenti nelle proprie classi.

Un primo possibile filone di algoritmi è quello degli *alberi decisionali*. L'algoritmo degli alberi decisionali è uno dei metodi più intuitivi del Machine Learning. Si ispira al modo in cui le persone prendono decisioni: di fronte a una serie di informazioni, si pongono domande in sequenza fino ad arrivare a una conclusione. Allo stesso modo, un albero decisionale costruisce una struttura ad albero in cui ogni nodo interno rappresenta una domanda su una delle variabili del dataset; ogni ramo corrisponde a una possibile risposta e ogni foglia finale indica una decisione o una classe predetta. Il processo di costruzione dell'albero inizia dal dataset intero. L'algoritmo deve scegliere quale variabile utilizzare per effettuare la prima divisione dei dati, cioè quale domanda permette di distinguere meglio le diverse classi. Per esempio, in un problema in cui si vuole prevedere se una persona acquisterà un prodotto, l'albero potrebbe cominciare chiedendo "L'età è minore di 40 anni?" oppure "Il reddito di chi acquista è alto?". Per stabilire quale divisione è la più efficace, l'algoritmo misura quanto sono "pure" le classi che si ottengono dopo ogni possibile suddivisione.

Una partizione è considerata più pura se contiene elementi appartenenti quasi tutti alla stessa classe. Esistono diverse formule per misurare questa purezza, le più usate sono l'impurità di Gini e l'entropia. Dato che il concetto di entropia viene visto in fisica al

quarto anno quando si affronta la termodinamica, potrebbe essere interessante mostrare agli studenti come un concetto di origine fisica possa essere utilizzato in un contesto così diverso. Sia che usi l'indice di Gini o l'entropia, lo scopo è scegliere la divisione che produce i gruppi più omogenei possibile riducendo il più possibile l'impurità complessiva.

Dopo aver scelto lo split migliore in base alla misura di purezza utilizzata, il dataset viene diviso in due o più sottoinsiemi, e lo stesso procedimento si ripete in modo ricorsivo su ciascun gruppo. Si continua a suddividere i dati fino a quando tutti i punti in un gruppo appartengono alla stessa classe, oppure fino a quando l'albero raggiunge una profondità massima stabilita dall'utente. Alla fine, ogni ramo dell'albero termina in una foglia che rappresenta una previsione: la classe più frequente tra i dati che arrivano in quel punto. Un esempio intuitivo può aiutare a capire meglio. Immaginiamo un piccolo dataset con le variabili "età" e "reddito", e con l'obiettivo di prevedere se una persona acquista o meno un prodotto. L'albero potrebbe iniziare chiedendo "Il reddito è basso?". Se la risposta è sì, il modello predice "Non compra"; se invece il reddito è medio o alto, la previsione è "Compra". In questo modo, l'albero traduce un insieme di dati numerici in una serie di regole logiche facilmente interpretabili.

Gli alberi decisionali sono molto apprezzati per la loro semplicità e per il fatto che le loro decisioni possono essere visualizzate e comprese facilmente anche da chi non ha conoscenze tecniche approfondite. Questi algoritmi presentano anche delle problematiche poiché tendono a creare modelli molto complessi che si adattano perfettamente ai dati del dataset di addestramento. Sono, quindi, fortemente soggetti al problema dell'"overfitting" che, come anticipato in precedenza, comporta il fatto che l'albero decisionale ottenuto non riesca a fare previsioni corrette se applicato a nuovi dati (per esempio al dataset di test). C'è l'ulteriore problema che piccole variazioni nei dati iniziali possono produrre alberi completamente diversi, rendendo il modello poco stabile. Per superare questi limiti,

nella pratica si usano spesso metodi più robusti che combinano molti alberi insieme, come la *Random Forest* o il *Gradient Boosting*, che riescono a mantenere la capacità esplicativa degli alberi ma con una maggiore affidabilità e precisione. Non è necessario entrare nel dettaglio di questi algoritmi, ma già accennare al problema è comunque utile agli studenti per comprendere la complessità dell'affrontare un problema reale.

## 10. Secondo percorso al triennio: reti neurali

Anche se la conoscenza approfondita e dettagliata della struttura di una rete neurale profonda e la matematica collegata è al di sopra delle competenze che si acquisiscono in una scuola superiore (anche solo banalmente perché la maggior parte della matematica delle superiori si limita allo studio delle funzioni in una variabile), si ritiene che sia possibile:

- introdurre l'idea di *perceptrone* con *pesi* e *funzione di attivazione*;
- introdurre l'*algoritmo della discesa del gradiente*;
- dare l'idea dell'*algoritmo di retropropagazione* come applicazione della derivazione di funzione composta.

Il *perceptrone* è il modello più semplice di una rete neurale, ed è anche il punto di partenza da cui è nato tutto. La prima rete neurale fu concepita dal neuroanatomista Warren McCulloch e dal matematico Walter Pitts, nel 1943 (McCulloch 1943). In quell'anno, i due pionieri pubblicarono un importante articolo sul funzionamento dei neuroni e diedero successivamente corpo alle loro idee, creando una semplice rete per mezzo di circuiti elettrici.

Il perceptrone può essere immaginato come un piccolo “neurone artificiale” che riceve dei numeri in ingresso, li elabora e restituisce una risposta. Ovviamente quella con il neurone è *solo una analogia*. Nella figura 1 è presentato lo schema del perceptrone, come descritto attualmente nei testi di Machine Learning.

Una serie di input  $x_0, x_1, \dots, x_n$  viene pesata tramite i valori  $w_1, w_2, \dots, w_n$  e data in pasto a una *funzione di attivazione*, ovvero una funzione

matematica che ha lo scopo di fornire l'output della rete e che introduce non linearità nel modello.

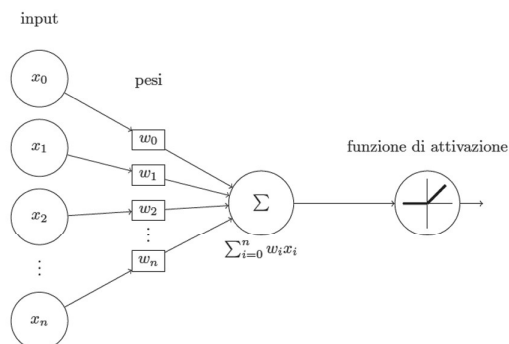


Figura 1: schema del perceptrone

Ogni ingresso (per esempio una misura, una caratteristica o un dato) viene moltiplicato per il peso corrispondente. Questi pesi indicano quanto ciascun ingresso è importante per la decisione finale.

Il perceptrone somma tutti questi valori pesati e poi applica una funzione, detta *funzione di attivazione* non lineare.

La non linearità della funzione di attivazione è essenziale perché consente alla rete neurale di apprendere e rappresentare relazioni complesse tra input e output. Tale funzione deve possedere anche altre caratteristiche, come la derivabilità, utile ad applicare, come vedremo, l'algoritmo di retropropagazione. Un esempio di funzione di attivazione è la funzione sigmoidea (o logistica)

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Un esercizio utile per gli studenti potrebbe essere quello di ragionare sulla forma di tali funzioni e su come il loro comportamento sia sfruttato. Attualmente molto diffusa è anche la funzione ReLu

(Rectified Linear Unit), non derivabile in 0 e non limitata superiormente così definita:

$$\text{ReLU}(x) = \max(x, 0)$$

Nel 1969 Minsky e Papert (Minsky e Papert 1969), tuttavia, in un loro celebre articolo dimostrarono che il perceptrone da solo può risolvere solo problemi “lineari”, cioè quelli in cui una linea (o un piano, se ci sono più dimensioni) è sufficiente a separare i dati. Quando i dati sono più complessi, serve qualcosa di più potente: una rete neurale.

Una *rete neurale* è formata da tanti perceptron collegati tra loro, organizzati in strati. Lo strato iniziale riceve i dati in ingresso, quelli centrali (chiamati “nascosti”) li elaborano, e lo strato finale produce il risultato. Ogni collegamento tra neuroni ha un peso, proprio come nel perceptrone, e la rete impara modificando questi pesi in modo da migliorare le sue previsioni.

Ma come fa la rete a capire come cambiare i pesi? Qui entra in gioco l’algoritmo di retropropagazione dell’errore (in inglese *backpropagation*). Dopo che la rete ha prodotto una risposta, confronta il risultato ottenuto con quello corretto (la “verità”). La differenza tra i due, cioè quanto la rete ha sbagliato, viene chiamata errore. La retropropagazione serve proprio a calcolare come questo errore si distribuisce tra tutti i pesi della rete. In altre parole, il modello “capisce” quali collegamenti hanno contribuito di più all’errore e li corregge di conseguenza.

Per capire di quanto correggere i pesi, si usa un altro algoritmo: la *discesa del gradiente*. In genere viene spiegato in modo intuitivo come un metodo per scendere da una collina cercando il punto più basso, che rappresenta l’errore minimo possibile. L’algoritmo calcola la direzione in cui l’errore diminuisce più velocemente e fa un piccolo passo in quella direzione. Poi ripete il processo molte volte, finché non arriva in un punto in cui l’errore è il più piccolo possibile.

Questo algoritmo, finché si rimane nel caso unidimensionale, può essere esaustivamente spiegato a studenti degli ultimi anni di liceo una volta che conoscano l'uso delle derivate e la loro interpretazione geometrica. Si possono individuare vari passi in un algoritmo di training di una rete neurale.

- *Inizializzazione dei pesi*: pesi della rete neurale vengono inizializzati casualmente. Questi pesi sono i parametri che determinano come i neuroni della rete combinano gli input per produrre output; quindi, è importante notare come la rete inizialmente non “sappia nulla”.

- *Propagazione in avanti (forward propagation)*: L'input del dataset di allenamento viene dato in pasto alla rete, a partire dal primo strato. In ogni strato, l'input viene moltiplicato per i pesi e viene applicata una funzione di attivazione per produrre l'output, che viene inviato allo strato successivo. Questo processo continua fino a calcolare l'output finale.

- *Calcolo del costo*: una volta ottenuto l'output finale, va valutato l'errore rispetto al valore atteso (detto anche valore *target*). Tale errore, o *Loss* o *costo*, si ottiene confrontando l'output con il valore corretto, che è noto perché durante una sessione di training si lavora con un dataset di cui “qualcuno”, tipicamente un utente esperto, ha fornito i risultati a priori. Un esempio semplice di funzione di costo, presentabile agli studenti, è quello dell'*errore quadratico medio*, introdotta a scuola in Statistica e in Fisica.

- *Propagazione all'indietro (backward propagation)*: avendo a disposizione l'errore, se ne calcola “l'entità” tramite la *regola della catena*; si parte dal suo valore finale e lo si “propaga all'indietro”, strato per strato, valutando il *gradiente dell'errore rispetto ai pesi*.

Si calcola il gradiente dell'errore rispetto all'output di ciascun neurone e poi rispetto ai pesi che lo hanno generato. Anche se il concetto di gradiente non fa parte delle indicazioni nazionali per i licei (mentre le funzioni a più variabili sono argomento trattato nel

quinto anno degli istituti tecnici) riteniamo che, a partire dal concetto di derivata di funzione composta, sicuramente noto almeno nella seconda parte del quinto anno di liceo, si possa illustrare e far comprendere, anche se in modo non completamente rigoroso, il significato di “derivata” applicato ad una funzione di più variabili.

- *discesa del gradiente*: Il gradiente dell'errore rispetto ai pesi fornisce una direzione di massima variazione, grazie alla quale è possibile procedere con l'algoritmo di ottimizzazione di *discesa del gradiente*. In altre parole, il gradiente dell'errore  $E$  fornisce la direzione verso cui muoversi per aggiornare i pesi, utilizzando la seguente formula:

$$w_{nuovo} = w_{vecchio} - \eta \frac{\partial E}{\partial w}$$
$$w_{nuovo} = w_{vecchio} - \eta \frac{\partial E}{\partial w}$$

dove  $w_{nuovo}$  e  $w_{vecchio}$  rappresentano rispettivamente i nuovi e i vecchi pesi,  $\eta$  il tasso di apprendimento (*learning rate*), un valore da scegliere accuratamente per incrementare la velocità di convergenza. L'algoritmo di propagazione in avanti, calcolo dell'errore, retropropagazione e aggiornamento dei pesi viene ripetuto per molte iterazioni (o *epoche*) fino a che l'errore  $E$  totale non è minimizzato, e da quel momento la rete è considerata sufficientemente addestrata.

Qui di seguito inseriamo, a titolo di esempio il codice Python che mostra come illustrare il funzionamento in una dimensione della discesa del gradiente della funzione  $y = x^2 + 5x + 6$

```
import matplotlib.pyplot as plt
import numpy as np
# Funzione di costo
def function(x):
    return x**2 + 5*x + 6
# Derivata della funzione
def derivative(x):
    return 2*x + 5
# Gradient Descent
def gradient_descent(initial_x, learning_rate, epochs):
    x = initial_x
    x_vals = []
    f_vals = []
    for _ in range(epochs):
        gradient = derivative(x)
        x = x - learning_rate * gradient
        x_vals.append(x)
        f_vals.append(function(x))
    plt.figure(figsize=(8, 6)) # dimensione in pollici - larghezza- altezza
    plt.title(f"Gradient Descent - Ultima Iterazione (Epoch {epochs})")
    plt.xlabel('x')
    plt.ylabel('f(x)')
    # Disegna la funzione
    x_range = np.linspace(-10, 5, 100)
    plt.plot(x_range, function(x_range), label='$f(x) = x^2 + 5x + 6$')
    # Disegna i punti del gradient descent
    plt.scatter(x_vals, f_vals, color='red', label='Gradient Descent Points')
    plt.legend()
    plt.grid(True)
    plt.show()
    return x
gradient_descent(5, 0.1, 12)
```

L'immagine seguente è ottenuta chiamando la funzione `gradient_descent(5,0.1,12)`.

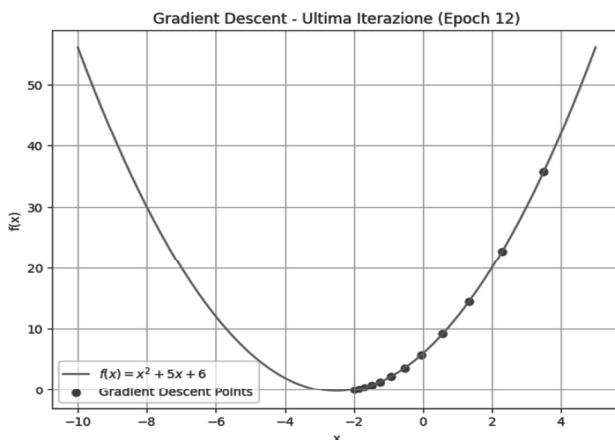


Figura 2: esempio di grafico illustrativo della discesa del gradiente in una dimensione

## 11 Riflessioni finali

Questo articolo, senza avere finalità di esaustività, si è posto come obiettivo quello di proporre spunti per realizzare nel quinquennio della scuola secondaria di secondo grado un percorso che illustri alcuni concetti base del Machine Learning. L'obiettivo è quello di mostrare agli studenti come la matematica svolga un ruolo essenziale in tutta l'Intelligenza Artificiale in modo che ogni volta che un qualsiasi sistema di apprendimento automatico verrà usato, potrà essere usato con maggiore consapevolezza, senza vederlo come un "oracolo" ma come uno strumento con limiti e potenzialità, frutto, comunque, dell'ingegno umano.

## Riferimenti bibliografici

Bishop, C., *Pattern Recognition and Machine Learning* (2006), 10.1117/1.2819119.

Goodfellow, I., Bengio, Y., Courville, A., *Deep Learning* (2016) The MIT Press.

Vaswani, A., Shazeer, N.M., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., & Polosukhin, I. (2017). *Attention is All you Need*. Neural Information Processing Systems.

Ananthaswamy, A. (2024). *Perché le macchine imparano. L'eleganza della matematica dietro all'AI*. Trad. Corrado Ghinamo. Milano: Apogeo. ISBN 978-88-503-3738-5.

James, G., Witten, D., Hastie, T., Tibshirani, R., Taylor, J. (2023). *An Introduction to Statistical Learning: with Applications in Python*. 10.1007/978-3-031-38747-

Passaro, D., *Matematica e programmazione : usare Python al Liceo in "Archimede : 1, 2016, Firenze (FI) : Le Monnier, 2016 , 2239-6314 - Casalini id: 3117219" - P. 42-48 - Permalink: <https://digital.casalini.it/10.1400/239797>*.

McCulloch, W.S., Pitts, W. *A logical calculus of the ideas immanent in nervous activity*. Bulletin of Mathematical Biophysics 5, 115–133 (1943). <https://doi.org/10.1007/BF02478259>.

Minsky, M. Papert, S. (1969). *Perceptron: an introduction to computational geometry*. The MIT Press, Cambridge, expanded edition, 19(88), 2.