

# Introduzione all'apprendimento automatico

Matteo Mazzaretto

2024/2025

## Indice

<b>1</b>	<b>Prima parte</b>	<b>1</b>
1.1	Quali sono i principali paradigmi del machine learning? Se ne riporti una descrizione sintetica – chiarendo quali siano le principali differenze – con particolare enfasi per il caso del supervised learning. Si distinguano in particolare classificazione e regressione	1
1.2	Si descrivano nel modo più accurato possibile i concetti di bias e variance, il loro rapporto e come nella pratica possano essere affrontati e ridotti i problemi. A tal fine si riportino anche esempi concreti che aiutino a chiarire i diversi aspetti coinvolti . . . . .	2
1.3	Si descriva in modo dettagliato il modello di logistic regression (con regolarizzazione), le sue principali caratteristiche ed il contributo dei diversi elementi presenti nella funzione di costo. Si riporti infine una descrizione accurata delle differenze e degli elementi in comune di tale modello rispetto ad un semplice classificatore lineare, anche mediante esempi qualitativi. Infine, si descriva chiaramente la procedura di addestramento mediante l'applicazione di gradient descent . . . . .	3
1.3.1	Introduzione e motivazione . . . . .	3
1.3.2	Formulazione del modello . . . . .	3
1.3.3	Interpretazione probabilistica . . . . .	3
1.3.4	Funzione di costo con regolarizzazione . . . . .	3
1.3.5	Confronto con il classificatore lineare . . . . .	4
1.4	Si descriva dettagliatamente la procedura di crossvalidation, motivandone scopo ed utilità, e fornendo una chiara descrizione della (corretta) procedura di addestramento di un qualunque sistema di machine learning. Si descrivano inoltre i concetti di true error ed empirical error e se ne evidenzino le relazioni con la procedura di cross-validation . . . . .	4
1.5	Si descriva dettagliatamente la procedura di model selection (aiutandosi con un esempio concreto) e si fornisca una chiara giustificazione teorica/concettuale a tale procedura . . . . .	5
1.6	Cosa si intende per “one learning algorithm hypothesis” e come tale ipotesi si relaziona con le reti neurali artificiali? Si fornisca inoltre una descrizione esaustiva degli elementi/ingredienti principali che permettono la definizione di una rete neurale multistrato	6

1.7	Spiegare in dettaglio gli elementi fondamentali del perceptron e, più in generale, delle reti neurali, illustrando chiaramente la fase di feed-forward	
	Si riporti inoltre una breve descrizione di come tale modello possa essere esteso mediante la realizzazione di un'architettura a più strati, fornendo un esempio che evidenzi le differenze/vantaggi di tale architettura. Si riporti inoltre un esempio di rete neurale per la realizzazione di un semplice operatore logico OR, AND, XOR, XNOR e NAND . . . . .	7
<b>2</b>	<b>Seconda parte</b>	<b>9</b>
2.1	Spiegare in dettaglio gli elementi fondamentali del perceptron e, più in generale, delle reti neurali. Si riporti inoltre una breve descrizione di come tale modello possa essere esteso mediante la realizzazione di un'architettura multistrato, fornendo un esempio che evidenzi le differenze ed i vantaggi di tale architettura. Illustrare chiaramente le due fasi di feedforward e backpropagation . . . . .	9
2.1.1	Feedforward . . . . .	9
2.1.2	Backpropagation . . . . .	10
2.2	Support Vector Machines (SVM) . . . . .	11
2.2.1	Interpretazione geometrica . . . . .	11
2.2.2	Funzione di costo . . . . .	11
2.2.3	Confronto con altri modelli . . . . .	12
2.2.4	Kernel Trick . . . . .	12
2.3	Si descriva in modo accurato l'algoritmo k-NN, illustrando il ruolo dei principali iperparametri, i vantaggi e le debolezze del modello nei confronti di altri algoritmi affrontati nel corso, e si evidenzii il principale inductive bias di tale algoritmo . . . . .	12
2.3.1	Vantaggi e svantaggi . . . . .	13
2.3.2	Confronto con Algoritmi Visti . . . . .	13
2.3.3	Scelta k . . . . .	14
2.4	Si descrivano nel modo più accurato possibile gli alberi di decisione, i loro vantaggi e svantaggi rispetto ad altri modelli (ad es. reti neurali) e si evidenzii il principale inductive bias di tale algoritmo. Si fornisca inoltre un semplice esempio di albero di decisione, successivamente lo XOR. Infine, si illustri brevemente l'estensione di tale modello attraverso random forest . . . . .	14

# 1 Prima parte

## 1.1 Quali sono i principali paradigmi del machine learning? Se ne riporti una descrizione sintetica – chiarendo quali siano le principali differenze – con particolare enfasi per il caso del supervised learning. Si distinguano in particolare classificazione e regressione

I principali paradigmi sono:

1. Supervised learning:  
lo scopo di questo paradigma è quello di dare la risposta corretta ad ogni esempio, ovvero, dati gli  $x^i$  e gli  $y^i$ , trovare la funzione  $h \approx f : X \rightarrow Y$   
Si chiama "supervised" perché il supervisore fornisce i valori di  $h(\cdot)$  alle varie istanze  $x^i$   
Può essere utilizzato sia per i casi di classificazione (valori discreti per capire se un dato appartiene ad una classe o ad un'altra) e per i casi di regressione (trovare la vera funzione che mappa correttamente gli input e gli output)
2. Unsupervised learning:  
in questo paradigma non esistono supervisori, l'obiettivo di questo paradigma è trovare la regolarità oppure i pattern, dunque dati esempio  $x^i$  trovare le regolarità presenti in tutto il dominio
3. Reinforcement learning:  
in questo paradigma si ha un agente che può essere in uno stato "s", esegue azione "a" (tra quelle ammissibili per lo stato s) ed opera in un ambiente "e" che in risposta all'azione "a" e lo stato "s" ritorna un nuovo stato e una ricompensa (positiva, negativa o neutrale)  
L'obiettivo dell'agente è massimizzare la funzione delle ricompense

Altri paradigmi possono essere:

1. Weak-supervised learning:  
è una branca dell'apprendimento automatico in cui vengono utilizzati dati non organizzati o imprecisi per fornire indicazioni per etichettare una grande quantità di dati non supervisionati in modo che possa essere utilizzata nell'apprendimento automatico o nell'apprendimento supervisionato
2. Self-supervised learning:  
consente ai sistemi di intelligenza artificiale di apprendere da ordini di grandezza maggiori di dati, il che è importante per riconoscere e comprendere modelli di rappresentazioni del mondo più sottili e meno comuni
3. Federated learning:  
(noto anche come apprendimento collaborativo) è una tecnica di apprendimento automatico che addestra un algoritmo su più dispositivi edge decentralizzati o server che conservano campioni di dati locali, senza scambiarli
4. Deep Learning:  
la cui traduzione letterale significa apprendimento profondo, è una sottocategoria del Machine Learning e indica quella branca dell'intelligenza

artificiale che fa riferimento agli algoritmi ispirati alla struttura e alla funzione del cervello chiamate reti neurali artificiali

**1.2 Si descrivano nel modo più accurato possibile i concetti di bias e variance, il loro rapporto e come nella pratica possano essere affrontati e ridotti i problemi. A tal fine si riportino anche esempi concreti che aiutino a chiarire i diversi aspetti coinvolti**

Il bias rappresenta l'errore sistematico introdotto da un modello nel semplificare troppo il problema, portando a una rappresentazione inaccurata dei dati  
L'inductive bias rappresenta l'insieme di ipotesi che un algoritmo assume sulla funzione target per poter generalizzare dai dati di training ai dati nuovi  
Infatti, nel caso della regressione lineare, l'inductive bias principale è che l'insieme di input e output può essere rappresentato come una funzione lineare, oppure, nel caso dei "Nearest Neighbors", si assume che la maggior parte dei casi in uno spazio ravvicinato faccia parte della stessa classe  
Ci sono esattamente due tipi:

1. restriction: limita lo spazio delle ipotesi
2. preference: impone l'ordine delle preferenze

Il bias alto causa all'algoritmo la mancanza di relazioni rilevanti fra feature e target, ad esempio un modello di regressione lineare ha un bias alto se i dati reali seguono una relazione quadratica: la sua rigidità lo porterà a fare errori sistematici

La variance misura quanto il modello è sensibile alle variazioni nei dati di training

Un modello ha varianza alta quando è molto sensibile ai cambiamenti nei dati di training: piccole variazioni possono portare a grandi differenze nelle predizioni  
Questo accade quando il modello è troppo complesso e tende a sovradattarsi (overfitting)

Il loro rapporto principale porta al bias-variance tradeoff, ovvero ciò che rappresenta il rapporto tra bias e varianza: un modello con bias alto semplifica troppo il problema (underfitting), mentre un modello con varianza alta si adatta troppo ai dati di training (overfitting)

L'obiettivo è trovare un equilibrio tra i due per minimizzare l'errore totale

La cosa migliore per un algoritmo sarebbe avere bias basso, varianza bassa

Alcuni metodi pratici per bilanciare bias e varianza:

- 1) Ridurre il bias: usare modelli più complessi (es. passare da regressione lineare a polinomiale)
- 2) Ridurre la varianza: usare più dati di training

Le learning curve (errori su training e validation) aiutano a diagnosticare questi problemi:

bias elevato si manifesta con errori alti su entrambi, mentre la variance si riconosce da un errore basso sul training ma alto sulla validation

**1.3** Si descriva in modo dettagliato il modello di logistic regression (con regolarizzazione), le sue principali caratteristiche ed il contributo dei diversi elementi presenti nella funzione di costo. Si riporti infine una descrizione accurata delle differenze e degli elementi in comune di tale modello rispetto ad un semplice classificatore lineare, anche mediante esempi qualitativi. Infine, si descriva chiaramente la procedura di addestramento mediante l'applicazione di gradient descent

#### 1.3.1 Introduzione e motivazione

La **logistic regression** è un modello di classificazione che evita i problemi della regressione lineare applicata a task di classificazione:

- La regressione lineare è sensibile a *outliers* e può produrre valori al di fuori dell'intervallo  $[0, 1]$ .
- La logistic regression garantisce invece output probabilistici in  $[0, 1]$  tramite la **funzione sigmoide**  
Comunque il suo scopo è quello di trovare una funzione  $h \approx f : X \rightarrow Y$  dove  $Y$  è un insieme di classi (2) e non in  $\mathcal{R}$  come nella regressione lineare

#### 1.3.2 Formulazione del modello

L'ipotesi è data da:

$$h_{\theta}(\mathbf{x}) = g(\theta^T \mathbf{x}) = \frac{1}{1 + e^{-\theta^T \mathbf{x}}}, \quad \text{dove } g(z) = \frac{1}{1 + e^{-z}}$$

#### 1.3.3 Interpretazione probabilistica

- $h_{\theta}(\mathbf{x}) = P(y = 1 \mid \mathbf{x}; \theta)$
- Per due classi:  $P(y = 0 \mid \mathbf{x}; \theta) = 1 - h_{\theta}(\mathbf{x})$
- **Limite di decisione:**

$$y = \begin{cases} 1 & \text{se } h_{\theta}(\mathbf{x}) \geq 0.5 \\ 0 & \text{se } h_{\theta}(\mathbf{x}) < 0.5 \end{cases}$$

#### 1.3.4 Funzione di costo con regolarizzazione

La **cross-entropy loss** con regolarizzazione:

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^m \left[ y^{(i)} \log(h_{\theta}(\mathbf{x}^{(i)})) + (1 - y^{(i)}) \log(1 - h_{\theta}(\mathbf{x}^{(i)})) \right] + \frac{\lambda}{2m} \sum_{j=1}^n \theta_j^2$$

Questa funzione serve per quando ci sono tantissime feature, però serve mantenere le due più importanti e le altre renderle molto piccole, dove quest'ultime sono molto poco influenti però danno il loro contributo per trovare  $y$   
Come in tutti gli altri casi, bisogna trovare  $\min(J(\Theta))$

### 1.3.5 Confronto con il classificatore lineare

- **Similarità:**

- Entrambi usano una funzione lineare  $\theta^T \mathbf{x}$ .
- Utilizzano ottimizzazione tramite gradient descent.

- **Differenze:**

- La regressione lineare minimizza l'MSE, la logistic regression la cross-entropy.
- La logistic regression produce probabilità, mentre il classificatore lineare valori continui.
- *Inductive bias*: La logistic regression assume una relazione logistica tra features e classi.

- **Esempio:** Per dati con outlier, la logistic regression è robusta mentre la regressione lineare può produrre un iperpiano distorto.

L'addestramento con gradient descent usa la procedura iterativa aggiornando i parametri  $\theta$  come:

$$\theta_j := \theta_j - \alpha \frac{\partial J(\theta)}{\partial \theta_j} = \theta_j - \frac{\alpha}{m} \left( \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^i) x_j^i + \lambda \theta_j \right)$$

dove  $\alpha$  è il learning rate

L'algoritmo termina quando le derivate sono vicine a zero (punto di minimo), si può notare che è molto simile al procedimento di "gradient descent" della regressione lineare

### 1.4 Si descriva dettagliatamente la procedura di cross-validation, motivandone scopo ed utilità, e fornendo una chiara descrizione della (corretta) procedura di addestramento di un qualunque sistema di machine learning. Si descrivano inoltre i concetti di true error ed empirical error e se ne evidenzino le relazioni con la procedura di cross-validation

La **cross-validation** è una tecnica fondamentale per la selezione e valutazione di modelli nel machine learning, con tre obiettivi chiave:

- Stimare le prestazioni del modello su dati non visti (*generalizzazione*)
- Ottimizzare gli iperparametri evitando l'overfitting
- Utilizzare efficientemente dataset limitati

Un'implementazione particolarmente diffusa è la **k-fold cross-validation**, dove il dataset viene suddiviso in  $k$  sottoinsiemi (*fold*) di uguale dimensione, utilizzati iterativamente per training e validazione

Facendo un esempio a 3 set tipicamente si divide in:

il più grande, detto training set, da 70%, gli altri due da 15%, detti validation

set, che servono a valutare la retta (o in generale la funzione  $h \approx f: X \rightarrow Y$ ) ottenuta dallo studio sul validation set

Questo serve a mantenere un learning molto più approfondito rispetto allo studio sul 100% del dataset, dove si rischierebbe di andare incontro all'overfitting. Inoltre, c'è la procedura di "leave-one-out", che è un particolare caso di k-fold dove si divide il dataset in k set di dimensione uguale disgiunti a due a due e volta per volta dei sottoinsiemi vengono considerati validation set e training set fra di loro

Relativamente a questa procedura, si possono associare i concetti di "true error" ed "empirical error":

1. il "true error" di una particolare ipotesi  $h$  che approssima  $f$  rispetto al dataset  $D$  è la probabilità che  $h$  non classificherà correttamente un'istanza di  $D$

$$error_D(h) = Prob_{x \in D} \{f(x) \neq h(x)\}$$

2. il "empirical error" dell'ipotesi  $h$  è dato da tutti i casi in cui  $h$  sbaglierà

$$error_T(h) = \frac{\#\{(x, f(x)) \in T : f(x) \neq h(x)\}, T \subseteq D}{\text{instances of } T}$$

Questi due errori diversi servono ad identificare il bias (assunzione troppo "forte/debole" che facciamo sulla funzione target) e la variance (la dipendenza dai dati)

Inoltre, serve ad identificare l'overfitting, il quale avviene quando:

data  $h \in H$ , overfitting avviene se  $\exists h' \in H : error_T(h) < error_T(h')$  ma  $error_D(h) > error_D(h')$

### 1.5 Si descriva dettagliatamente la procedura di model selection (aiutandosi con un esempio concreto) e si fornisca una chiara giustificazione teorica/concettuale a tale procedura

La model selection è il processo grazie al quale si può scegliere il miglior modello/ipерparametri dato un problema

Uno dei tanti metodi per trovarli è la cross-validation, che fa parte della famiglia della model selection

Facendo un esempio di model selection per la selezione di immagini, ipotizzo che si voglia cercare il miglior modello per la selezione di immagini (classificazione binaria) che identifica se un'immagine rappresenta un cane oppure no

Nell'esempio selezionato, la predizione si può classificare come:

- vero positivo (TP)
- falso positivo (FP)
- vero negativo (TN)
- falso negativo (FN)

L'accuratezza del modello è data da  $\frac{TP+TN}{P+N} = \frac{\text{all correct}}{\text{all instances}}$ , invece la performance si passa su:

- precision:  $\frac{TP}{TP+FP}$ : frazione delle istanze recuperate che sono rilevanti
- recall:  $\frac{TP}{TP+FN}$ : frazione delle istanze rilevanti che sono recuperate

Il rapporto corretto, che purtroppo non può essere sempre preciso al 100%, soprattutto su animali rarissimi di cui girano poche foto su Internet, si valuta tramite gli integrali fra le curve di precision e recall

Si possono però stimare bias e variance, infatti: le learning curve rappresentano: bias elevato se manifesta con errori alti su entrambi, la variance si riconosce da un errore basso sul training ma alto sulla validation

Inoltre, spesso si computa Prec@k e Rec@k nei top k risultati

Infine, un'altra misurazione è l'average precision (AP) che è una misura che combina recall e precision per risultati ordinati, quindi restituisce ciò che ha una precisione maggiore, che si calcola sommando la precisione quando si trova un True positive identificato realmente

$$AP = \frac{\sum_{k=1}^n P(k) \cdot rel(k)}{\text{all groundtruth instances}}$$

Quest'ultima sommatoria serve a trovare la corretta sequenza in cui i primi k risultati corretti compaiono, infatti se AP=1 vuol dire che tutti i risultati antecedenti al 1° sbagliato saranno corretti

Grazie a quest'ultima, nel caso del nostro esempio ma anche per altri modelli, si può trovare chiaramente qual è il modello migliore per un determinato problema

## 1.6 Cosa si intende per “one learning algorithm hypothesis” e come tale ipotesi si relaziona con le reti neurali artificiali? Si fornisca inoltre una descrizione esaustiva degli elementi/ingredienti principali che permettono la definizione di una rete neurale multistrato

”One learning algorithm hypothesis” si riferisce all'idea che esista un unico algoritmo di apprendimento che possa essere utilizzato per affrontare una vasta gamma di compiti, indipendentemente dalla specificità dei dati o della struttura del problema

In altre parole, questa ipotesi suggerisce che un solo algoritmo di apprendimento, se opportunamente configurato e allenato, possa essere in grado di risolvere una varietà di problemi, inclusi quelli complessi come la classificazione e la previsione

Nell'ambito delle reti neurali artificiali, questa ipotesi si relaziona al fatto che, nonostante le reti neurali possano variare in complessità e architettura, un singolo algoritmo di apprendimento, come la retropropagazione (backpropagation), può essere utilizzato per addestrare una rete neurale su vari tipi di dati, resolvendo compiti complessi come la classificazione o la regressione



**1.7 Spiegare in dettaglio gli elementi fondamentali del perceptron e, più in generale, delle reti neurali, illustrando chiaramente la fase di feed-forward**  
**Si riporti inoltre una breve descrizione di come tale modello possa essere esteso mediante la realizzazione di un'architettura a più strati, fornendo un esempio che evidenzi le differenze/vantaggi di tale architettura**  
**Si riporti inoltre un esempio di rete neurale per la realizzazione di un semplice operatore logico OR, AND, XOR, XNOR e NAND**

Il perceptron è un modello semplice:

un neurone che fa una trasformazione per combinazione lineare con una "funzione di attivazione" definita come  $f(\Theta^T x)$

Questa funzione di attivazione può essere la sigmoide ( $\frac{1}{1+e^{-z}}$ ), il tanh (tangente iperbolico), la ReLU ( $\max(0, x)$ ), Leaky ReLU:  $\max(0.1x, x)$ , Maxout:  $\max(w_1^T x + b_1, w_2^T x + b_2)$ , ELU :  $x$  quando  $x \geq 0$ ,  $\alpha(e^x - 1)$  quando  $x < 0$

Ogni neurone implementa da solo la logistic regression (nel caso in cui usi la funzione sigmoide)

Esso prende in input quindi tutti i pesi  $\Theta$  e definisce, tramite la funzione scelta fra quelle descritte precedentemente, un output

In questo caso, i pesi  $\Theta$  sono definiti dalla fase di feed-forward dello strato precedente

Questi perceptron sono la base per la costruzione di una rete neurale multi-strato, infatti essa è formata da più strati, divisi in 3 macrocategorie:

1. input layer
2. hidden layer ( $\geq 1$ )
3. output layer

Il passaggio da uno strato all'altro (feed-forward) si misura come

$$a_i^{(l)} = f \left( \sum_{k=1}^{n^{(l-1)}} \Theta_{ik}^{(l)} a_k^{(l-1)} + b_i^{(l)} \right)$$

**Spiegazione dei simboli:**

- $a_i^{(l)}$ : Attivazione del neurone  $i$ -esimo nel layer  $l$ .
- $\Theta_{ik}^{(l)}$ : Peso tra il neurone  $k$  del layer  $l - 1$  e il neurone  $i$  del layer  $l$ .
- $b_i^{(l)}$ : **Bias** (spesso incluso come  $\Theta_{i0}^{(l)}$  con  $a_0^{(l-1)} = 1$ ).
- $f$ : Funzione di attivazione (ReLU, sigmoide, ecc.).
- $n^{(l-1)}$ : Numero di neuroni nel layer precedente.

Mentre i perceptron riescono a fare unicamente una combinazione lineare, il collegamento fra più perceptron permette architetture molto più complesse che

permettono di definire operazioni più difficili che un unico perceptron non riuscirebbe a fare

Questo permette la costruzione di un sistema che offre nettamente più vantaggi rispetto al perceptron singolo, ma anche un numero di collegamenti decisamente più elevato (per le reti che abbiamo visto fino ad ora ogni perceptron è collegato a tutti i perceptron dello stato successivo, portando la complessità computazionale ad essere molto elevata per sistemi formati da centinaia se non migliaia di strati)

Queste reti neurali seguono una fase di feed-forward, ovvero una propagazione in avanti dei dati dove si porta ogni dato da uno strato a quello successivo

L'architettura multistrato e la propagazione feed-forward permettono la costruzione di operatori logici OR, AND, XOR (XNOR  $\rightarrow$  NOT AND NOT) e NAND (NOT AND)

La differenza principale fra questi operatori logici è che OR e AND sono operatori logici linearmente separabili, quindi non richiedono strati nascosti, invece XOR e NAND non sono linearmente separabili e richiedono degli strati nascosti per arrivare al corretto risultato

**Gli esempi sono nelle slide/negli appunti**

## 2 Seconda parte

### 2.1 Spiegare in dettaglio gli elementi fondamentali del perceptron e, più in generale, delle reti neurali. Si riporti inoltre una breve descrizione di come tale modello possa essere esteso mediante la realizzazione di un'architettura multistrato, fornendo un esempio che evidenzi le differenze ed i vantaggi di tale architettura. Illustrare chiaramente le due fasi di feedforward e backpropagation

Il perceptron è un modello semplice:

un neurone che fa una trasformazione per combinazione lineare con una "funzione di attivazione" definita come  $f(\Theta^T x)$

Questa funzione di attivazione può essere la sigmoide ( $\frac{1}{1+e^{-z}}$ ), il tanh (tangente iperbolico), la ReLU ( $\max(0, x)$ ), Leaky ReLU:  $\max(0.1x, x)$ , Maxout:  $\max(w_1^T x + b_1, w_2^T x + b_2)$ , ELU :  $x$  quando  $x \geq 0$ ,  $\alpha(e^x - 1)$  quando  $x < 0$

Ogni neurone implementa da solo la logistic regression (nel caso in cui usi la funzione sigmoide)

Esso prende in input quindi tutti i pesi  $\Theta$  e definisce, tramite la funzione scelta fra quelle descritte precedentemente, un output

In questo caso, i pesi  $\Theta$  sono definiti dalla fase di feed-forward dello strato precedente

Questi perceptron sono la base per la costruzione di una rete neurale multistrato, infatti essa è formata da più strati, divisi in 3 macrocategorie:

1. input layer
2. hidden layer ( $\geq 1$ )
3. output layer

#### 2.1.1 Feedforward

Il passaggio da uno strato all'altro (feed-forward) si misura come

$$a_i^{(l)} = f \left( \sum_{k=1}^{n^{(l-1)}} \Theta_{ik}^{(l)} a_k^{(l-1)} + b_i^{(l)} \right)$$

**Spiegazione dei simboli:**

- $a_i^{(l)}$ : Attivazione del neurone  $i$ -esimo nel layer  $l$ .
- $\Theta_{ik}^{(l)}$ : Peso tra il neurone  $k$  del layer  $l - 1$  e il neurone  $i$  del layer  $l$ .
- $b_i^{(l)}$ : **Bias** (spesso incluso come  $\Theta_{i0}^{(l)}$  con  $a_0^{(l-1)} = 1$ ).
- $f$ : Funzione di attivazione (ReLU, sigmoide, ecc.).
- $n^{(l-1)}$ : Numero di neuroni nel layer precedente.

Mentre i perceptron riescono a fare unicamente una combinazione lineare, il collegamento fra più perceptron permette architetture molto più complesse che permettono di definire operazioni più difficili che un unico perceptron non riuscirebbe a fare

Questo permette la costruzione di un sistema che offre nettamente più vantaggi rispetto al perceptron singolo, ma anche un numero di collegamenti decisamente più elevato (per le reti che abbiamo visto fino ad ora ogni perceptron è collegato a tutti i perceptron dello stato successivo, portando la complessità computazionale ad essere molto elevata per sistemi formati da centinaia se non migliaia di strati)

Queste reti neurali seguono una fase di feed-forward, ovvero una propagazione in avanti dei dati dove si porta ogni dato da uno strato a quello successivo

L'architettura multistrato e la propagazione feed-forward permettono la costruzione di operatori logici OR, AND, XOR (XNOR  $\rightarrow$  NOT AND NOT) e NAND (NOT AND)

La differenza principale fra questi operatori logici è che OR e AND sono operatori logici linearmente separabili, quindi non richiedono strati nascosti, invece XOR e NAND non sono linearmente separabili e richiedono degli strati nascosti per arrivare al corretto risultato

**Gli esempi sono nelle slide/negli appunti**

### 2.1.2 Backpropagation

La backpropagation è un algoritmo fondamentale per l'addestramento delle reti neurali, che calcola efficientemente il gradiente della funzione di costo rispetto ai pesi della rete mediante la **regola della catena** (chain rule). Questo permette di ottimizzare i parametri attraverso metodi come la discesa del gradiente.

#### Algoritmo

L'algoritmo si compone di due fasi principali:

1. **Forward pass**: calcolo dell'output della rete dato un input.
2. **Backward pass**: propagazione all'indietro dell'errore per aggiornare i pesi.

**Formule matematiche** Sia:

- $L$ : numero di strati (layer) nella rete
- $\Theta^{(l)}$ : matrice dei pesi al layer  $l$
- $a^{(l)}$ : attivazioni al layer  $l$
- $z^{(l)} = \Theta^{(l)} a^{(l-1)}$ : input al layer  $l$  prima dell'attivazione
- $f$ : funzione di attivazione (es. sigmoide, ReLU)

1. **Errore al layer di output** ( $\delta^{(L)}$ ):

$$\delta^{(L)} = a^{(L)} - y$$

dove  $y$  è il vettore target.

2. **Errore nei layer nascosti** ( $l = L - 1, L - 2, \dots, 2$ ):

$$\delta^{(l)} = (\Theta^{(l)})^T \delta^{(l+1)} \odot f'(z^{(l)})$$

dove  $\odot$  indica il prodotto elemento per elemento (Hadamard product) e  $f'$  è la derivata della funzione di attivazione.

3. **Gradiente dei pesi**:

$$\frac{\partial J}{\partial \Theta^{(l)}} = \delta^{(l+1)} (a^{(l)})^T$$

4. **Aggiornamento dei pesi** (gradient descent):

$$\Theta^{(l)} := \Theta^{(l)} - \alpha \frac{\partial J}{\partial \Theta^{(l)}}$$

con  $\alpha$  tasso di apprendimento.

**Osservazioni chiave**

- L'inizializzazione casuale dei pesi (non nulla) è cruciale per evitare simmetria nell'apprendimento
- La backpropagation è applicabile a qualsiasi architettura feedforward

## 2.2 Support Vector Machines (SVM)

Le Support Vector Machines (SVM) sono un algoritmo di apprendimento supervisionato per classificazione e regressione. Pur essendo un classificatore lineare, può affrontare problemi non lineari attraverso il *kernel trick*.

### 2.2.1 Interpretazione geometrica

L'SVM cerca l'**iperpiano di separazione ottimale** che massimizza il **margin** tra le classi. Geometricamente:

- I **vettori di supporto** sono i punti più vicini all'iperpiano e determinano la soluzione
- Il **margin**  $M$  è la distanza tra l'iperpiano e i vettori di supporto:  $M = \frac{2}{\|\Theta\|}$
- Per dati non linearmente separabili, si introducono **variabili di slack**  $\xi^{(i)}$  per permettere errori di classificazione

### 2.2.2 Funzione di costo

La SVM utilizza una funzione di costo modificata rispetto alla logistic regression:

$$\min_{\Theta} \frac{1}{\delta} \sum_{i=1}^m \left[ \underbrace{y^{(i)} \max(0, 1 - h_{\Theta}(x^{(i)}))}_{\text{Per } y^{(i)}=1} + \underbrace{(1 - y^{(i)}) \max(0, 1 + h_{\Theta}(x^{(i)}))}_{\text{Per } y^{(i)}=0} \right] + \frac{1}{2} \|\Theta\|^2$$

dove:

- $h_{\Theta}(x) = \Theta^T x + \Theta_0$  è il classificatore lineare
- $\delta$  (o  $C = \frac{1}{\delta}$ ) regola il trade-off tra errore e margin
- Il termine  $\frac{1}{2} \|\Theta\|^2$  massimizza il margin di separazione

### 2.2.3 Confronto con altri modelli

- **vs Logistic Regression:**

- Entrambi sono classificatori lineari
- LR minimizza la cross-entropy, SVM massimizza il margine
- SVM più robusta a outliers

- **vs Reti Neurali:**

- SVM trova soluzione globale, NN può convergere a minimi locali
- SVM più efficiente con pochi dati
- NN più flessibile per problemi complessi

- **vs Decision Trees:**

- SVM fornisce iperpiani di separazione, DT divisioni assiali (paralleli agli assi)
- SVM migliore per spazi continui, DT per dati categoriali

### 2.2.4 Kernel Trick

Per problemi non lineari, SVM applica una trasformazione  $\phi : R^n \rightarrow R^m$ :

- Evita il calcolo esplicito di  $\phi(x)$  usando funzioni kernel:

$$K(x, z) = \phi(x)^T \phi(z)$$

- Kernel comuni:

- Gaussiano (RBF):  $K(x, z) = \exp\left(-\frac{\|x-z\|^2}{2\sigma^2}\right)$
- Polinomiale:  $K(x, z) = (x^T z + c)^d$

- Il modello diventa:

$$f(x) = \sum_{i=1}^m \alpha_i y^{(i)} K(x^{(i)}, x) + b$$

dove  $\alpha_i$  sono i coefficienti dei vettori di supporto

## 2.3 Si descriva in modo accurato l'algoritmo k-NN, illustrando il ruolo dei principali iperparametri, i vantaggi e le debolezze del modello nei confronti di altri algoritmi affrontati nel corso, e si evidenzi il principale inductive bias di tale algoritmo

L'algoritmo k-NN è un modello non parametrico, ovvero quel tipo di modello che richiede come parametri solamente i dati in input, a differenza di altri modelli come le reti neurali oppure la logistic regression

È un *lazy learner* poiché non costruisce un modello esplicito durante il training, ma posticipa tutta la computazione alla fase di prediction

Questo tipo di modello è usato per approssimare una funzione target  
Nella fase di learning semplicemente si memorizzano i dati, invece, nella fase di prediction

**Fase di prediction:** Per un nuovo punto  $x_{\text{test}}$ :

1. Calcola le distanze tra  $x_{\text{test}}$  e tutti gli  $x_i \in \mathcal{D}$
2. Seleziona i  $k$  vicini più prossimi ( $\mathcal{N}_k$ )
3. Assegna la classe più frequente in  $\mathcal{N}_k$  (moda)

L'**inductive bias** è il **principio di località**: punti vicini nello spazio delle feature hanno etichette simili

Formalmente:

$$\forall x_i, x_j \in \mathcal{X}, \quad d(x_i, x_j) \approx 0 \implies P(y_i = y_j) \approx 1$$

L'idea chiave è che il valore della funzione target per un sample è stimato dagli esempi di training memorizzati, calcolando la distanza fra il sample e tutti gli esempi di training, scegliendo la classe dell'esempio meno distante (oppure in base ai  $k$  elementi meno distanti, scegliendo la classe più presente)

Le metriche includono:

- Norma 2 (Euclidea):  $\sqrt{\sum_{i=1}^n (x_i - y_i)^2}$
- Norma 1 (Manhattan):  $\sum_{i=1}^n |x_i - y_i|$

Il decision boundary viene descritto tramite il diagramma di Voronoi che mostra come gli spazi di input sono suddivisi fra le classi nel piano d-dimensionale

Formalmente, il **diagramma di Voronoi** partiziona lo spazio in celle dove:

$$V(x_i) = \{x \in \mathcal{X} \mid d(x, x_i) \leq d(x, x_j), \forall j \neq i\}$$

### 2.3.1 Vantaggi e svantaggi

Il principale vantaggio è che la fase di learning ha un tempo di  $O(1)$ , ovvero richiede solo la memorizzazione dei dati, cosa non presente in diversi algoritmi visti fino ad ora e che può essere utilizzato per classificare le istanze di test fra più classi (a differenza ad esempio della logistic regression)

Il principale svantaggio è anche la complessità, infatti:

- Training:  $O(1)$  (memorizzazione)
- Prediction:  $O(mn)$  (brute-force), riducibile a  $O(n \log n)$  con strutture dati specializzate

Si può ridurre a qualcosa di simile ad  $O(n \log n)$  con una struttura ad albero per scendere più velocemente in base alla distanza

Inoltre, altri problemi sono il fatto che alcune feature hanno range più larghi, quindi devono essere considerate più importanti (feature scaling), poi alcune feature possono essere anche irrilevanti (feature selection) e bisogna cercare di ridurre la dimensione dei dati (PCA)

### 2.3.2 Confronto con Algoritmi Visti

- **vs Alberi di Decisione:**
  - k-NN cattura relazioni non lineari complesse, ma è più lento
  - Alberi più efficienti ma con divisioni assiali
- **vs SVM:**
  - SVM più efficienti con grandi dataset
  - k-NN non richiede kernel per non-linearità
- **vs Reti Neurali:**
  - NN apprendono rappresentazioni, k-NN opera nello spazio originale
  - NN più scalabili in alta dimensionalità

### 2.3.3 Scelta k

La scelta della k (parametro che indica quanti punti meno distante dall'istanza di test bisogna considerare) è molto importante, e con un dataset molto sbilanciato può portare la scelta ad essere sempre verso un'unica classe evitando le altre meno presenti

Fortunatamente, nei casi realistici i dataset difficilmente sono sbilanciati, e k viene scelta più piccola di  $\sqrt{m}$ , m = elementi dataset

## 2.4 Si descrivano nel modo più accurato possibile gli alberi di decisione, i loro vantaggi e svantaggi rispetto ad altri modelli (ad es. reti neurali) e si evidenzii il principale inductive bias di tale algoritmo. Si fornisca inoltre un semplice esempio di albero di decisione, successivamente lo XOR. Infine, si illustri brevemente l'estensione di tale modello attraverso random forest

Gli alberi di decisione sono una struttura in cui:

- i nodi interni rappresentano feature
- i rami rappresentano l'esito del test (esempio: vero o falso)
- nodi foglia rappresentano classi o valori target

Forniscono risultati interpretabili e supportano analisi di decisione

Ogni albero rappresenta la funzione d'ipotesi  $h(x)$  che vogliamo modellare

L'idea generale è che, dato un attributo per volta, si fa un test, poi, condizionatamente alla scelta, si prende un altro attributo per fare un altro test, fino a quando non si arriva alla foglia che assegna un valore all'input

Gli alberi di decisione possono essere visti come modelli generati dalle regole d'induzione, che è una differenza sostanziale rispetto ad altri algoritmi come la regressione lineare (il cui inductive bias principale è che la classificazione sia di tipo binaria) oppure la regressione lineare (che cerca di minimizzare la MSE)

Il loro principale vantaggio è che viene costruito partizionando le scelte su un



attributo in una determinata situazione, permettendo la costruzione di un albero di decisione ottimale, soprattutto con pochi attributi. Ciò permette di restringersi ad una dimensione per volta, dichiarando l'output solo quando si arriva alla fine

Il principale svantaggio è proprio il modo in cui costruire quest'albero, infatti ad oggi esistono diversi algoritmi ma, essendo un problema NP completo (non esprimibile con scelta greedy) non esiste ancora un algoritmo ottimale che permette la costruzione di un albero "perfetto". Un'idea è quella di minimizzare l'entropia per ogni sottoinsieme oppure di massimizzare l'Information Gain ( $H(S) - H(S | a)$ ), dove  $H(S)$  è l'entropia

Inoltre, un altro svantaggio è che forniscono divisioni assiali (paralleli agli assi) ed è più complicato permettere la costruzione di divisioni oblique come si fa con il margine di separazione nelle SVM

Un'estensione di questo modello è il Random forest:

è un modello di classificazione composto da più alberi di decisione

Usa bagging (genera più pezzi casuali per aumentare variabilità soluzione) e feature random per combinare alberi

Riduce la varianza dell'albero originale, però porta a ridondanza e quindi a maggiore complessità computazionale

La classificazione finale si fa in base ai voti o alla media