Machine Learning

Emanuele Galiano
Damiano Trovato

Anno Accademico 2025/2026

Indice

1 Introduzione al Machine Learning **1**
1.1 Definizione di Machine Learning (Arthur Samuel - 1959) 1
1.2 Perchè abbiamo bisogno del Machine Learning 1
1.2.1 Esempio delle email spam e non-spam 2
1.3 Machine Learning Algorithm (Tom Mitchel - 1998) 2
1.4 Definizione di Task 2
1.5 Definizione di Esperienza 2
1.6 Definizione di Performance 3
1.7 Esempio completo 3
1.8 Task, Esempi ed Etichette 4
1.9 Estrazione delle features 4
1.10 Features 5
1.11 Esempio delle Email Spam e Non Spam 5
1.12 Tipologie di Task 6
1.12.1 Classificazione 6
1.12.2 Regressione 7
1.13 Supervised Learning e Unsupervised Learning 8
1.14 Reinforcement Learning 9
1.15 Misura di Performance (P) 9
1.15.1 Esempio 10
1.16 Experience (E) 10
1.17 Dataset (D) 11
1.18 Design Matrix 11
1.18.1 Esempio 11
1.19 Learning 12
1.19.1 In cosa consiste il training? 12
1.19.2 Esempio 13
1.20 Il paradigma Training/Testing 13
1.21 Sui dati 14
1.21.1 Testing - Training 14
1.21.2 Normalizzazione 14
1.22 Iperparametri 15
1.22.1 k-fold Cross Validation 15

1.23 Overfitting e underfitting 16
1.23.1 Bias e Variance 16

2 I primi modelli di Machine Learning 19
2.1 Neurone computazionale - modello di McCulloch-Pitt 19
2.1.1 Limitazioni del modello di McCulloch-Pitt 20
2.2 Percettrone - Rosenblatt 20
2.2.1 Processo generale di training del percettrone 22
2.2.2 Implementazione del training 22
2.2.3 Versione di Adaline 23
2.3 Modelli lineari 23
2.3.1 Problema della separabilità non lineare 23
2.3.2 Problema dello XOR 25

3 Regressione 27
3.1 Ingredienti del task 27
3.1.1 Tipi di regressione 27
3.2 Regressione lineare semplice 27
3.2.1 Funzione di loss 28
3.3 Regressione lineare multipla 28
3.4 Feature scaling 28
3.5 Algoritmo di discesa del gradiente 28
3.5.1 Versione batch per la regressione lineare 29
3.6 Regressione polinomiale e feature mapping 29
3.7 Regolarizzazione 30
3.8 Regressione multivariata 30
3.9 Valutazione 30

Capitolo 1

Introduzione al Machine Learning

1.1 Definizione di Machine Learning (Arthur Samuel - 1959)

Nel 1959, Arthur Samuel fornisce una **definizione di machine learning**: il machine learning è il campo di studi che abilita i computer ad **imparare**, **senza essere esplicitamente programmati**.

Il paradigma alla base è fortemente diverso da quello tipico: se un algoritmo classico è **programmato ad hoc** per risolvere un task, e si comporta in maniera prettamente **deterministica**, un algoritmo di machine learning può **imparare a risolvere problemi** associati a vasti set di dati (classificare elementi, riconoscerli, trovare correlazioni). Questo permette di spostare il nostro focus non più sullo sviluppo di tutti gli step necessari affinchè un algoritmo risolvi **un problema specifico**, ma sulla creazione di un modello che riesca ad apprendere come risolvere **un insieme di problemi simili**.

1.2 Perchè abbiamo bisogno del Machine Learning

L'approccio utilizzato finora per risolvere i problemi è quello di:

- Trovare una logica per risolvere il problema.
- Scrivere un programma.
- Suddividerlo in pezzi più piccoli (funzioni).
- Automatizzare l'approccio.

Questo funziona per problemi di natura fortemente univoca, che sappiamo come risolvere, ad esempio:

- Computare l'area di un poligono.
- Risolvere equazioni differenziali.

Nel caso del poligono, supponendo di voler calcolare l'area di un rombo i dati presi in input sarebbero dati dalla coppia (x_1, x_2), contenente le lunghezze della diagonale principale e secondaria. Questi dati,passati ad un algoritmo, permettono di calcolarne l'area $\frac{x1*x2}{2}$ e generarne un output

Dati → Programma che risolve un task → Output

Alcuni problemi tuttavia presentano un alto grado di **incertezza**, che li rende più difficili da affrontare. Non poter fare assunzioni sui dati in input, e non conoscere tutti i possibili task, rende impossibile l'utilizzo di algoritmi standard per compiti del tipo:

- Classificazione di email spam e non spam
- Object detection

Il machine learning rappresenta la soluzione ideale a problemi di questo tipo, proponendo una nuova pipeline:

Dati + Output Atteso → Machine Learning → Soluzioni su nuovi dati

1.2.1 Esempio delle email spam e non-spam

Vogliamo creare un algoritmo di machine learning in grado di determinare se una mail è spam o meno. Il nostro obiettivo è quindi classificare ciascuna di queste come **spam**, o **ham**[1].

- `Compra prodotto a 10$! Oferta imperdibile!` → Spam
- `Ciao Giovanni, come stai?` → Ham

1.3 Machine Learning Algorithm (Tom Mitchel - 1998)

Un algoritmo **apprende dall'esperienza** E rispetto a una certa classe di **Task** T e a una misura di **performance** P. Se la sua **performance** nel compito T, misurata tramite P, migliora con l'**esperienza** E, allora quel modello ha appreso con successo.

1.4 Definizione di Task

Rappresenta il problema che deve essere risolto. Nell'esempio di determinare se una mail è spam o meno, il task è quello di **predire** l'etichetta (Y ="spam" oppure Y ="ham"), ed è strettamente legata al modello, che rappresentiamo come funzione parametrizzata, indicata con h_θ.

1.5 Definizione di Esperienza

Rappresentano i dati, ovvero i valori assunti dalle **random variables**, nell'esempio X è il contenuto della mail ed Y l'etichetta. La coppia di valori:

$$\{(X = x_i, Y = y_i)\}_{i=1}^{N}$$

[1] Email legittima, non spam.

Rappresenta l'esperienza. Generalmente vista come una collezione di elementi chiamati **esempi**.

1.6 Definizione di Performance

Funzione P che **valuta quanto bene** il modello è in grado di **risolvere un certo task** T. Supponiamo che il nostro algoritmo abbia previsto un insieme di etichette per un dato numero di email che indichiamo con:

$$\{\hat{y}_i\}$$

Dove il simbolo 'hat' indica che il dato non è stato osservato ma **previsto**. L'insieme delle etichette corrette è invece dato da

$$\{y_i\}$$

Per valutare la qualità del nostro metodo, dovremmo confrontare i due insiemi di previsioni utilizzando una **misura di performance:**

$$P(\{y_i\}, \{\hat{y}_i\})$$

Questa funzione restituisce un valore reale appartenente al range [0,1].

- Un **valore elevato** indica che le previsioni sono accurate
- Un **valore basso** indica che le previsioni non sono accurate.

Indichiamo con il termine **misura di errore** il valore: $1 - P$. Per risolvere problemi di machine learning ci affidiamo a modelli statistici che dipendono dal task.

1.7 Esempio completo

Siano:

- $x^{(1)}$: Il testo dell'email 1: "Compra prodotto a 10$! Oferta imperdibile!"
- $x^{(2)}$: Il testo dell'email 2: "Ciao Giovanni, come stai?"
- $y^{(1)}$: L'etichetta **spam**
- $y^{(2)}$: L'etichetta **ham**
- h_θ: Il modello

Allora

$$h_\theta(x^{(1)}) = \hat{y}^{(1)}$$

e

$$h_\theta(x^{(2)}) = \hat{y}^{(2)}$$

1.8 Task, Esempi ed Etichette

Un esempio è generalmente espresso come una raccolta di valori che sono stati misurati quantitativamente da un evento osservato. Un esempio è generato da un vettore:

$$x \in \mathbb{R}^d$$

Scritto anche come:

$$x = (x_1, x_2, ..., x_d)$$

I valori del vettore x sono detti **features**, in quanto rappresentano **proprietà specifiche** degli esempi in input. Se la dimensionalità di x è 10, diremo che ha 10 features Nella maggior parte dei casi, ogni esempio x è anche abbinato a un output desiderato y. Tali output desiderati, sono anche chiamati **etichette**. Un'attività può quindi essere definita come un certo modo di elaborare un esempio di input per ottenere un output.

Torniamo al nostro esempio: determinare se un'e-mail è spam o ham. In questo caso, l'input è l'email, le features possono essere caratteristiche dell'email, come il numero di errori ortografici o la presenza di alcune parole chiave, mentre l'output atteso è l'etichetta (spam o ham).

1.9 Estrazione delle features

Per gestire le email, dobbiamo prima trasformarle in un'**entità quantificabile**. Questo di solito viene fatto **identificando alcune caratteristiche** dei dati che sono **rilevanti per il compito dato** (numero di errori ortografici o la presenza di alcune parole chiave). In pratica, stiamo cercando una funzione f che trasformi l'entità dalla sua forma originale a una forma di destinazione, che è buona per risolvere un compito specifico:

$$x \rightsquigarrow f(x) \rightsquigarrow \overline{x}$$

Dove x è il dato grezzo di input (ad esempio, il messaggio di posta elettronica completo), f è la funzione di trasformazione e $\overline{x}$[2] è l'output della trasformazione, che sarà l'input dell'algoritmo di apprendimento automatico.

La funzione f è chiamata *rappresentazione*. L'output della trasformazione x è anche chiamato rappresentazione. Poiché rappresentando i dati otteniamo un vettore di funzionalità, il processo di rappresentazione dei dati è talvolta chiamato **features extraction**. Non ci sono «rappresentazioni universali», ma solo rappresentazioni che servono a qualche compito.
Le rappresentazioni sono di 2 tipi:

- Create a mano
- Apprese

[2]Su questa notazione: da ora in poi, quando ci riferiremo all'output della trasformazione, non useremo più $\overline{x}$, ma direttamente x, dando per scontato il passaggio di rappresentazione $f(x)$.

L'estrazione delle features **mette in luce caratteristiche salienti** trascurandone altre.

1.10 Features

Generalmente, l'output di una funzione di rappresentazione è nella forma:

$$x = (x_1, x_2, \ldots, x_d), \quad x \in \mathbb{R}^d$$

Questo, è composto da un insieme di features . **Una feature è la specifica di un attributo**. Si tratta di una misura che rappresenta **aspetti dei dati** che è utile **evidenziare per risolvere il problema considerato**. Ad esempio, il colore può essere un attributo. "Il colore è blu" è una funzionalità estratta da un esempio.
Le caratteristiche possono essere di due tipi principali:

Categoriche: un numero finito di valori discreti. Questi possono essere:

- **Nominali:** a indicare che non esiste **alcun ordinamento** tra i valori, ad esempio cognomi e colori.
- **Ordinali:** a indicare che esiste un **ordinamento rilevante**, ad esempio in un attributo che assume i valori basso, medio o alto.

Continue: comunemente, **sottoinsieme di numeri reali**, dove c'è una differenza misurabile tra i valori possibili. I numeri interi sono solitamente trattati come continui nei problemi pratici.

1.11 Esempio delle Email Spam e Non Spam

Consideriamo il nostro esempio in cui vogliamo distinguere le e-mail spam da quelle non spam. L'input del processo sono i messaggi di posta elettronica, quindi dobbiamo trasformarli in vettori di features:

$$x = (x_1, x_2, ..., x_n)$$

con un processo di *features extraction*.

Naturalmente, ci aspettiamo che le funzionalità estratte siano **utili per risolvere il nostro compito** di determinare se un'e-mail è spam o ham. Possiamo notare che le e-mail di spam spesso includono errori ortografici e parole come "Acquista", "occasione" e "10$". Quindi, potremmo decidere di rappresentare ogni messaggio di posta elettronica con due numeri:

- Il conteggio degli errori ortografici.
- Il numero di volte in cui alcune parole o pattern specifici appaiono nel testo.

Una volta che i messaggi di input sono stati convertiti in **vettori di funzionalità**, possono essere visti come **vettori nello spazio** $\mathbb{R}^2$.

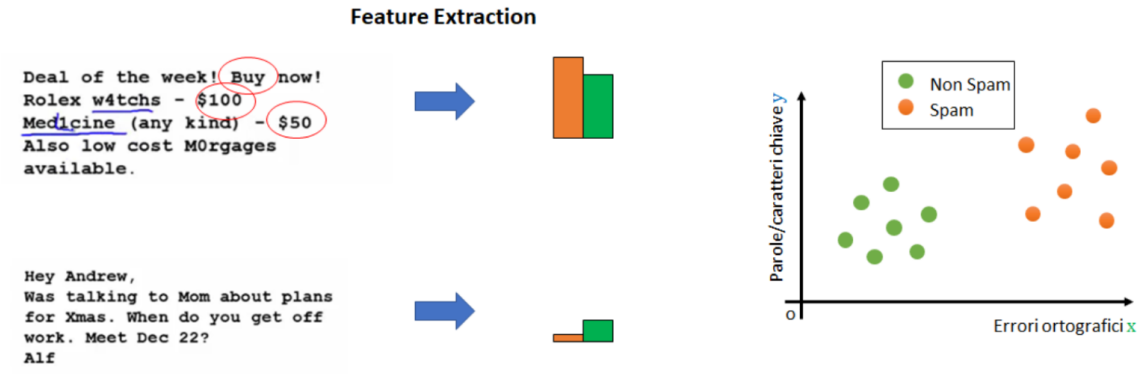

Figura 1.1: Estrazione delle feature: le e-mail vengono trasformate in vettori 2D, dove x = errori ortografici e y = pattern ripetibili, e proiettate nello spazio delle feature, , dove la separazione tra spam (arancione) e non spam (verde) risulta evidente.

1.12 Tipologie di Task

Le attività possono essere di diversi tipi. Di seguito, discuteremo due compiti principali:

- **Classificazione**
- **Regressione**

Assumeremo che ogni algoritmo di apprendimento automatico prenda come input esempi che sono già stati rappresentati con una funzione di rappresentazione adeguata.

1.12.1 Classificazione

In questo tipo di attività, alla macchina viene chiesto di specificare a quale di un insieme predefinito di categorie K appartiene l'input.
Esempi di questo compito sono:

- Classificare i post di Facebook come riguardanti la politica o qualcos'altro (classificazione politica vs non politica).
- Rilevamento delle e-mail di spam (classificazione dello spam vs legittima delle e-mail).
- Riconoscimento dell'oggetto raffigurato in un'immagine tra 1000 oggetti diversi (riconoscimento dell'oggetto).

L'algoritmo di apprendimento è solitamente fornito con un insieme di esempi:

$$\{x^{(1)}, x^{(2)}, ..., x^{(n)}\} \text{ dove: } x^{(j)} \in \mathbb{R}^N \forall j$$

e un insieme di etichette corrispondenti

$$\{y^{(1)}, y^{(2)}, ..., y^{(n)}\} \text{ dove: } y^{(j)} \in \{1, .., k\} \forall j$$

che specificano a quale delle categorie K appartiene ogni esempio.

Ad esempio, se $y^{(j)} = 3$, allora $x^{(j)}$ appartiene alla classe "3".

Nel caso della classificazione binaria (ad esempio, spam vs non spam), $y^{(j)} \in \{0, 1\}$. Per risolvere questo compito, l'algoritmo di apprendimento automatico assume la forma di una funzione:

$$h_\theta : \mathbb{R}^N \to \{1, ..., K\}$$

tale che:

$$y^{(j)} = h_\theta(x^{(j)})$$

Esempio:

- **Classification Task:** data un'e-mail, classificarla come spam o non spam.
- **Input:** esempi n-dimensionali $x = (x_1, x_2, ..., x_n)$ contenenti le caratteristiche dell'email, come il numero di errori ortografici e l'occorrenza di parole specifiche.
- **Output:** etichette $y \in \{0, 1\}$ che indicano se l'e-mail è legittima o spam.

Alcuni algoritmi di classificazione **non prevedono un output discreto**, ma un vettore di **probabilità**, contenente la probabilità relativa a ciascuna delle etichette possibili. In questo caso, puntiamo ad avere la probabilità massima nello slot del vettore relativo all'etichetta vera.

1.12.2 Regressione

In questo tipo di compito, al programma del computer viene chiesto di **prevedere un valore numerico dato un input**, tipo:

- Prevedere il prezzo delle case date alcune caratteristiche come la città, l'età, la zona, ecc.
- Prevedere il valore futuro delle azioni di una società dai valori di altre società o da altre statistiche sul mercato (previsione del mercato azionario).
- Conta il numero di auto presenti in un'immagine.

Analogamente alla classificazione, l'algoritmo viene fornito con esempi di training $x \in \mathbb{R}^N$ e con gli output desiderati $y \in \mathbb{R}$. L'algoritmo di apprendimento automatico assume la forma di una funzione $h_\theta : \mathbb{R}^N \to \mathbb{R}$ tale che $y^{(j)} = h_\theta(x^{(j)})$.
Esempio:

- **Regression task:** Predire il prezzo di una casa in base ai suoi metri quadrati.
- **Input:** Dimensione della casa x (valore scalare)
- **Output:** Prezzo y.

Sono algoritmi ottimi per trovare relazioni tra i dati ed effettuare predizioni.

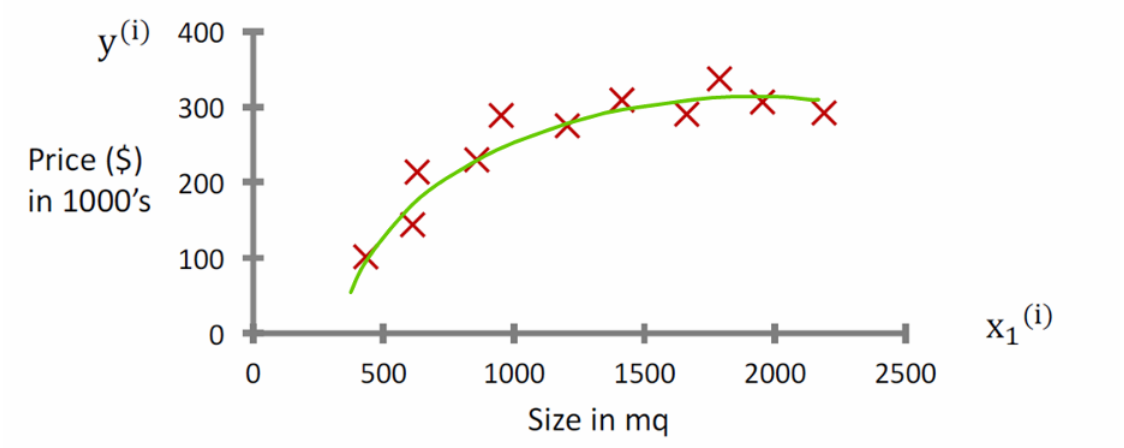

Figura 1.2: Relazione tra dimensione dell'immobile ($x_1 1^{(i)}$, in mq) e prezzo ($y^{(i)}$, in migliaia di \$): i punti rossi sono i dati osservati, la linea blu rappresenta un modello di regressione lineare che non approssima bene l'andamento non lineare.

1.13 Supervised Learning e Unsupervised Learning

Gli approcci di Machine Learning possono essere approssimativamente divisi in **supervised** e **unsupervised learning**.

Supervised Learning: L'algoritmo viene addestrato su un insieme di esempi di input e **output desiderati**. L'obiettivo è allenare un modello a mappare gli input agli output corretti. Il punto chiave qui è il conoscere gli output desiderati: i dati del nostro dataset dovranno quindi essere preventivamente **etichettati**.

Unsupervised Learning: L'algoritmo viene addestrato solo su esempi di input, senza output desiderati. L'obiettivo è trovate struttura, pattern e associazioni nei dati, spesso molto eterogenei.

$$\{x^{(1)}, x^{(2)}, ..., x^{(n)}\} \qquad \text{dove: } x^{(j)} \in \mathbb{R}^N \forall j$$

Questi tipi di compiti mirano generalmente a **modellare la struttura dei dati**. Un esempio di unsupervised learning è il **clustering**, in cui non viene fornita alcuna informazione aggiuntiva oltre agli esempi.

Gli approcci supervised sono generalmente più facili da gestire, ma richiedono la presenza di **labels**. Ottenere labels è spesso un problema costoso in termini di tempo, poiché richiede che le persone annotino manualmente i dati. Ad esempio, se dobbiamo costruire un spam-detector utilizzando un approccio supervised, è necessario che qualcuno etichetti manualmente diverse email come 'spam' o 'non-spam'.

1.14 Reinforcement Learning

Alcuni autori fanno riferimento anche a una terza classe di algoritmi di Machine Learning: il **Reinforcement Learning**.

Il Reinforcement Learning mira a **scoprire la soluzione a un problema** attraverso il metodo *trial and error*, piuttosto che tramite istruzioni esplicite su come risolvere il compito. Questo avviene permettendo all'algoritmo di **interagire con un environment** e ricevere **positive rewards** quando compie azioni che portano a un buon risultato (rispetto al problema da risolvere) e **negative rewards** quando compie azioni che portano a un risultato negativo.

L'obiettivo degli algoritmi di Reinforcement Learning è apprendere una policy π, che possa essere utilizzata per **determinare quale azione a intraprendere** quando si acquisisce un'**osservazione** del mondo o. Questo processo **ricorda il modo naturale in cui gli animali imparano a risolvere problemi**. Ad esempio, si può pensare a un topo che deve trovare l'uscita da un labirinto (immagine 1.3).

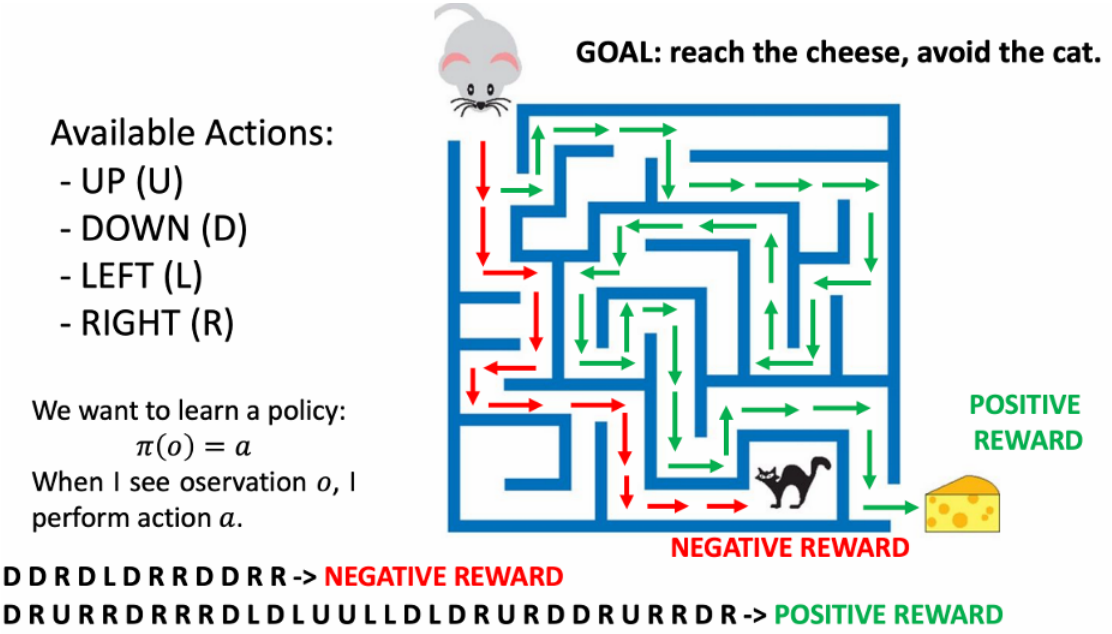

Figura 1.3: Reinforcement Learning nel labirinto: l'agente sceglie tra azioni U/D/L/R e apprende una policy $\pi(o) = a$ che massimizza la ricompensa, raggiungendo il formaggio (positiva) ed evitando il gatto (negativa).

1.15 Misura di Performance (P)

Per valutare le capacità di un algoritmo di Machine Learning nel risolvere un determinato compito, è necessaria una misura quantitativa delle sue prestazioni. Solitamente, questa **performance measure** P è **specifica per il task** T che il sistema sta eseguendo.

Per compiti come la classification, spesso si misura la performance utilizzando l'**accuracy**, ovvero la percentuale di esempi classificati correttamente dal modello. Nel caso della regression, invece, si possono usare altre metriche come il mean squared error.
Le misure di performance sono utilizzate per due motivi principali:

- Capire quando un algoritmo di Machine Learning sta migliorando in un determinato compito.
- Valutare la performance dell'algoritmo una volta finalizzato.

Una performance measure può anche essere vista in termini di error. Ad esempio, l'**accuracy** corrisponde a un error rate (la percentuale di esempi classificati in modo errato), calcolato come $1 - accuracy$.

1.15.1 Esempio

Un spam detector analizza cinque email. Le prime tre sono spam, le ultime due non lo sono. L'algoritmo classifica come spam le prime due email e come non spam le ultime tre. In questo caso, la prima e le ultime due classificazioni sono corrette, mentre la terza è errata. La accuracy si calcola come la percentuale di esempi classificati correttamente:

$$\frac{4}{5} = 0.8 \quad \text{ovvero} \quad 80\%$$

1.16 Experience (E)

Un algoritmo di Machine Learning apprende dall'**experience** per migliorare una performance measure su un determinato task.
L'**experience** è costituita da una raccolta di esempi

$$x^{(i)}$$

(noti anche come data points, poiché possono essere mappati in uno spazio multi-dimensionale tramite una funzione di rappresentazione), eventualmente accompagnati dalle relative labels

$$y^{(i)}$$

(a seconda del task considerato).
Esistono due principali tipi di algoritmi di Machine Learning:

- Supervised approaches (quando abbiamo le paired labels, ad esempio nella classification e nella **regression**).
- Unsupervised approaches (quando non abbiamo paired labels, come nel **clustering**).

L'**experience** assume forme diverse a seconda del tipo di approccio di Machine Learning utilizzato.

1.17 Dataset (D)

Le performance measures vengono generalmente calcolate rispetto a un insieme di esempi, piuttosto che su singoli esempi. Un insieme di esempi (eventualmente con labels) è chiamato **dataset**. I datasets sono generalmente omogenei, nel senso che i dati contenuti al loro interno hanno un formato simile. Ad esempio:

- Nel Fisher's Iris dataset, tutti gli esempi hanno 4 features e una label corrispondente a una delle tre classi.
- In un dataset di immagini di food, ogni immagine è associata a una class che indica il piatto specifico.

1.18 Design Matrix

Un modo comune per rappresentare un dataset è utilizzare una design matrix. Poiché ogni esempio è una collezione di n features, un dataset di m elementi può essere rappresentato tramite una matrice

$$X \in \mathbb{R}^{m \times n},\ m, n \in \mathbb{N}$$

di dimensione $m \times n$.

- Ogni riga della design matrix rappresenta un esempio.
- Ogni colonna rappresenta una delle features.

Nel caso del supervised learning, si considera spesso anche un'altra matrice

$$Y \in \mathbb{A}^{m \times k}, k \in \mathbb{N}$$

dove k è la dimensionalità degli output desiderati.
Ad esempio, nel caso della classification,

$$\mathbb{A} = \{1, \ldots, M\}$$

dove M è il numero di classi e k è spesso uguale a 1.

1.18.1 Esempio

Supponiamo di avere un dataset composto da 1000 email, alcune classificate come spam e altre come not spam. Assumiamo che ogni email sia rappresentata da due features, come discusso nei precedenti esempi.
La design matrix che rappresenta il dataset è una matrice

$$X \in \mathbb{R}^{1000 \times 2}$$

- Ogni elemento della matrice rappresenta una delle features di un esempio nel dataset.

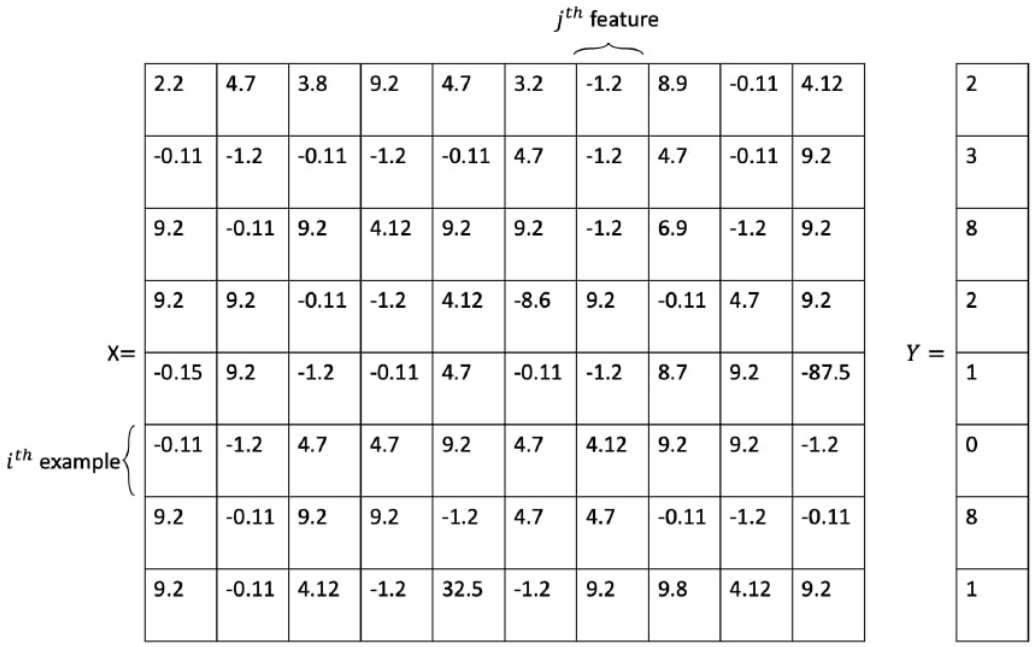

Figura 1.4: Design Matrix: rappresentazione di un dataset con m esempi e n features. Ogni riga corrisponde a un esempio, ogni colonna a una feature.

- Ad esempio, $X_{i,1}$ indica il numero di **errori** ortografici nell'i-**esima email**, mentre $X_{j,2}$ rappresenta il numero di **occorrenze** di parole chiave nell'j-**esima email**, e così via.

Le labels sono contenute in un vettore

$$Y \in \{0,1\}^{1000}$$

dove Y_i rappresenta la label associata all'i-**esimo esempio** (ad esempio, 0 = not spam, **1 = spam**).

1.19 Learning

Un algoritmo di Machine Learning **utilizza un dataset di esempi** per **migliorare la sua performance** in un determinato **task**. Il processo di miglioramento della performance dell'algoritmo è chiamato **learning** o **training**.

1.19.1 In cosa consiste il training?

- Un algoritmo di Machine Learning ha alcuni parametri chiamati parameters, che possono essere regolati per modificarne il comportamento. Questi parametri sono legati a un model (una funzione matematica) utilizzata per risolvere il task.
- Un algoritmo chiamato training procedure utilizza gli esempi forniti per trovare i valori ottimali per questi parameters.

- Alcuni parametri non possono essere regolati automaticamente dal training. Questi sono detti hyperparameters e devono essere ottimizzati al di fuori della training procedure, spesso attraverso un metodo trial and error.

1.19.2 Esempio

Consideriamo un semplice spam detector che classifica le email come spam o non-spam in base al numero di errori ortografici.
L'algoritmo può essere scritto come segue:

```
1 def classify(x):
2     if x > a:
3         return 1  # Spam
4     else:
5         return 0  # Non-spam
```

Listing 1.1: Soglia sul numero di errori ortografici, chiaramente un approccio naive

L'algoritmo dipende da un singolo parametro a. La domanda è: quale valore dovremmo assegnare ad a? La *training procedure* permette di trovare un valore ottimo presumibilmente adatto per a. Una semplice training procedure consisterebbe nel provare diversi valori per a e registrare le performance dell'algoritmo per ciascun valore di a. Alla fine, possiamo scegliere il valore di a che massimizza la performance measure P.

1.20 Il paradigma Training/Testing

Ciascun algoritmo di Machine Learning presenta al suo interno dei parametri. Una scelta opportuna dei parametri, dovrebbe garantire il funzionamento del modello. La procedura generale, per la scelta e la verifica dei parametri dell'algoritmo, è suddivisa in due parti:

1. **Training dell'algoritmo**.
   Effettuato usando i dati dell'omonimo **training set**, consiste nella scelta dei parametri del modello, vedendo ogni volta i valori della *loss function*, o funzione di costo, che offre un'indice dell'errore da parte del modello, permettendo di capire se i parametri vanno modificati, e di quanto.
2. **Testing dell'algoritmo - Inferenza.**
   Passiamo ora ai dati di testing, usati per verificare se i parametri scelti sul dataset di training, sono opportuni su dati mai visti prima. Andremo a valutare quindi il nostro modello, secondo quelle che chiamiamo **misure di valutazione**.

1.21 Sui dati

1.21.1 Testing - Training

Come accennato in precedenza, distinguiamo due insiemi di dati: i **dati di training** e i **dati di testing**. È importante che i dati usati in fase di training, e i dati usati in fase di inferenza, siano **totalmente disgiunti**.

$$\text{Training} \cap \text{Testing} = \varnothing$$

Tuttavia, entrambi i set di dati appartengono in origine all'insieme X. La suddivisione dei dati in training e testing sarà effettuata randomicamente, in modo da evitare bias di qualsiasi tipo. Dobbiamo inoltre assicurarci che il numero di dati, per ogni etichetta, sia uguale. Prendere l'80% di un dataset in fase di testing, con due classi (Cane, non-cane), significa prendere l'80% di ciascuna delle due classi. L'obiettivo è **evitare favoritismi** durante la classificazione. Parleremo più avanti di tecniche che stabiliscono come effettuare la suddivisione tra training e testing, sfruttando al meglio i propri dati, come la k-fold cross-validation.

1.21.2 Normalizzazione

I nostri dati $x \in \mathbb{R}^d$ possono avere valori appartenenti a range molto ampi, con range diversi per ciascuna di queste features. Per creare spazi in cui le features hanno più o meno la stessa dimensione, si usano delle strategie di normalizzazione, come di seguito.

Normalizzazione in $[0, 1]$**.** Per ogni j-esima feature, dobbiamo andare a stabilire il minimo e il massimo del dataset. Dato un dataset di dimensione m:

$$x_j^{\min} = \min\{x_j^{(i)} \mid i = 1, 2, \ldots, m\},$$
$$x_j^{\max} = \max\{x_j^{(i)} \mid i = 1, 2, \ldots, m\}.$$

Possiamo quindi normalizzare secondo la seguente formula:

$$\hat{x}_j = \frac{x_j - x_j^{\min}}{x_j^{\max} - x_j^{\min}},$$

e ogni dato sarà $\hat{x}_j \in [0, 1]$.

Normalizzazione in $[-1, 1]$**.**

$$\hat{x}_j = \frac{x_j - \mu_j}{\sigma_j}$$

con σ_j varianza. Il risultato è un centramento a $(0, 0)$ dei nostri dati. L'origine coinciderà con la nostra media. Ogni dato sarà $\hat{x}_j \in [-1, 1]$.

Quantità statistiche per la j-esima feature

Per completezza, riportiamo le definizioni delle quantità usate sopra:

$$\text{Media:} \quad \mu_j = \frac{1}{m}\sum_{i=1}^{m} x_j^{(i)},$$

$$\text{Varianza:} \quad \text{Var}(x_j) = \frac{1}{m}\sum_{i=1}^{m}\left(x_j^{(i)} - \mu_j\right)^2,$$

$$\text{Deviazione standard:} \quad \sigma_j = \sqrt{\frac{1}{m}\sum_{i=1}^{m}\left(x_j^{(i)} - \mu_j\right)^2} \;=\; \sqrt{\text{Var}(x_j)}.$$

Bisognerà conservare rispettivamente $x_j^{\min}, x_j^{\max}$ o μ_j, σ_j, per permettere la normalizzazione del testing set, in quanto la normalizzazione è fatta in fase di training.

1.22 Iperparametri

Sono dei parametri che stabiliscono aspetti del nostro modello, e che non sono calcolati in fase di training. In fase di training, questi vengono fissati, e **non influiscono sul training**. Nonostante ciò, gli iperparametri avranno un effetto sulla nostra loss-function. Bisognerà variare gli iperparametri usando un **validation set**, un terzo set di dati sempre ottenuto dal dataset X.

1.22.1 k-fold Cross Validation

È una strategia che permette di ottenere ottimi risultati anche con dataset più piccoli. Dividiamo in due parti il nostro dataset X, ottenendo l'insieme di training e l'insieme di testing. Dividiamo nuovamente l'insieme di **training** in k parti. Supporremo $k = 3$. Utilizzeremo un $\frac{1}{k}$ per la validazione e le $\frac{k-1}{k}$ rimanenti come training. In questo caso, 1/3 e 2/3. Effettueremo tre volte training e validazione, variando ogni volta il subset usato per la validazione e quelli usati per il testing. Da ciascuno dei processi di training, emergerà un valore della loss function: sceglierò gli iperparametri associati a valori della loss function più bassi.

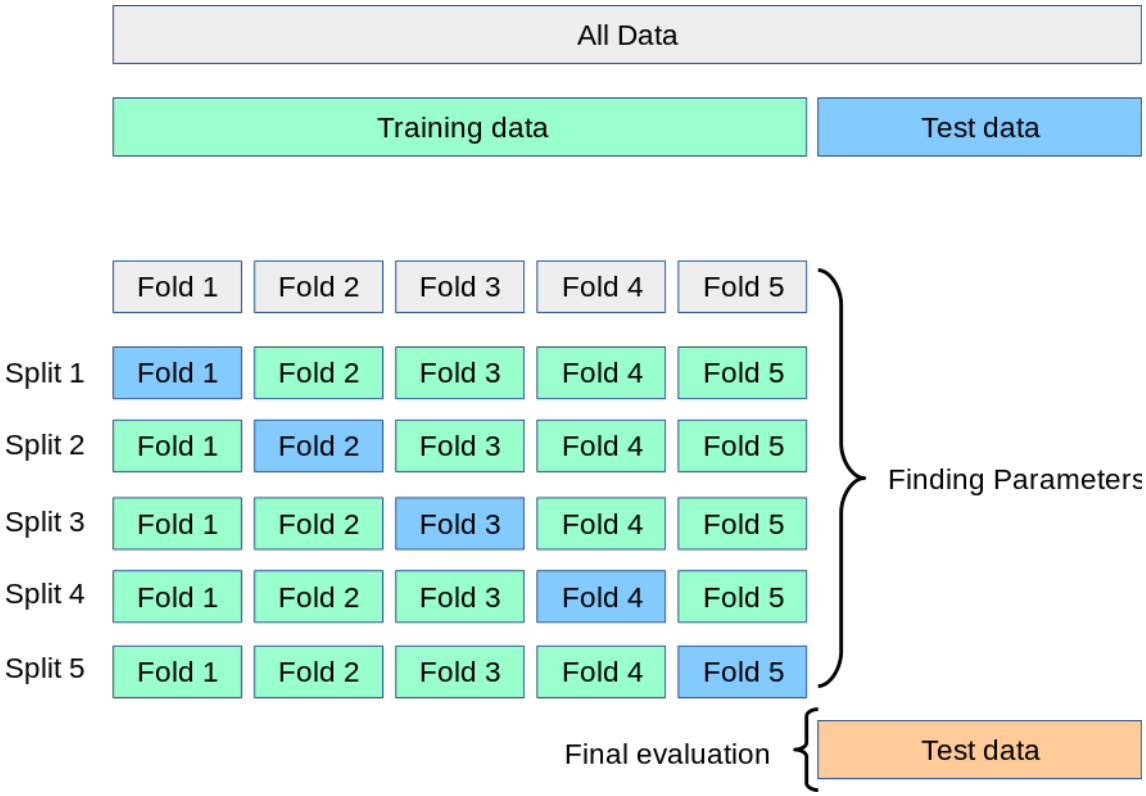

Figura 1.5: Procedura di cross-validation con $k = 5$.

1.23 Overfitting e underfitting

Sono tra i problemi più grandi che possono emergere con i propri algoritmi di machine learning, e riguardano spesso la **scelta del modello**.

Overfitting Generalmente si parla di overfitting, quando il modello è così complesso da offrire performance ottime in fase di training (impara facilmente e bene), ma fortemente **peggiori in fase di testing**: il modello è così complesso da imparare troppo bene i dati su cui ha appreso, ma da pessime performance sui dati inediti.

Underfitting Caratterizzato da **pessime performance in fase di training**, troppe poche variabili, modello troppo semplice. L'apprendimento non è tale da permettere al modello di conoscere il proprio dataset. Il modello non impara!

Bisogna quindi trovare lo *sweetspot*, il compromesso, tra i modelli più complessi, e quelli più semplici. Il numero di parametri è un fattore da tenere sempre in considerazione.

1.23.1 Bias e Variance

Osservando i risultati dei nostri modelli, sarà possibile individuare facilmente due tipologie di errori:

Bias Error. Le previsioni non sono corrette a causa di assunzioni errate nel modello. Un alto bias può portare il modello a non rilevare importanti relazioni tra le features e gli

output target, portando a underfitting (es. un modello che erca di approssimare una funzione polinomiale con una lineare). Usare modelli con alto bias causa underfitting.

Variance Error. Le previsioni del modello sono fortemente influenzate da piccole fluttuazioni nel training set. Ad esempio, se rimuoviamo un punto dati e riaddestriamo l'algoritmo, otteniamo previsioni molto diverse. Questo può essere dovuto al fatto che l'algoritmo sta cercando di modellare il rumore casuale presente nei dati di training, il che di solito porta a overfitting.

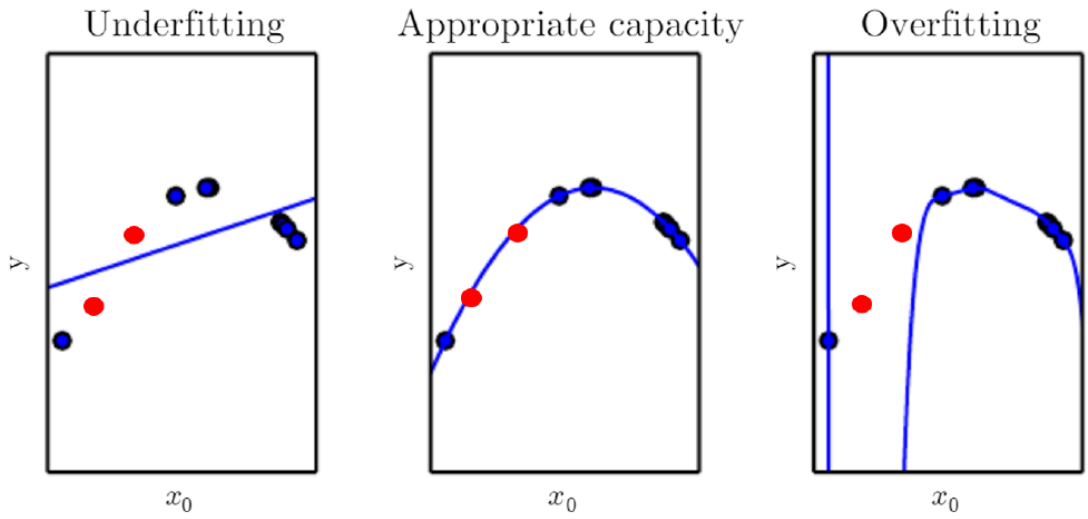

Figura 1.6: In blu i dati di training, in rosso quelli di testing. Nel caso dell'underfitting, vediamo una funzione troppo semplice (una funzione lineare) per approssimare una funzione più complessa. Nel caso dell'overfitting, il modello da output assurdi.

Capitolo 2

I primi modelli di Machine Learning

2.1 Neurone computazionale - modello di McCulloch-Pitt

L'idea iniziale era quella di replicare la struttura di un neurone biologico con un modello matematico, in modo da poter simulare il funzionamento del cervello umano.

Neurone biologico. Un neurone biologico è costituito da:

- Dendriti: ricevono segnali da altri neuroni.
- Corpo cellulare: elabora i segnali ricevuti.
- Assone: trasmette il segnale elaborato ad altri neuroni.

Il primo modello di neurone computazionale fu proposto da McCulloch e Pitts nel 1943. Si basa su tre componenti principali:

- Degli input $x_1, x_2, \ldots, x_d$ binari, che possono essere **eccitatori** e **inibitori**.
- Una threshold v, una soglia
- Un output binario.

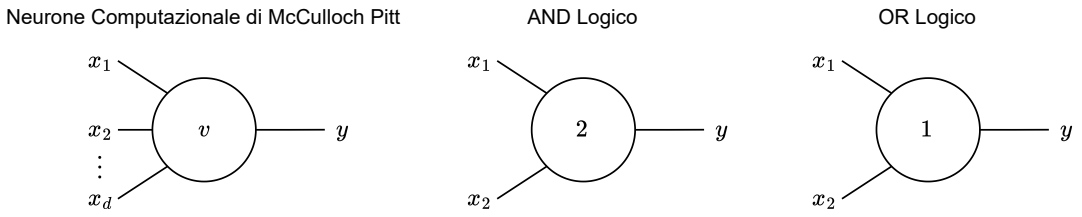

Figura 2.1: Modello di McCulloch Pitt e due computazioni possibili: AND logico e OR logico

Supponiamo che $x_1, \ldots, x_j$ siano eccitatori, $x_{j+1}, \ldots, x_d$ sono inibitori. Se $j \geq 1$ e almeno un inibitore è $= 1$, allora il neurone ritorna 0. Un inibitore è sufficiente per bloccare

l'output. Altrimenti, si calcola $z = x_1 + \cdots + x_j = x_1 + \cdots + x_d$, ossia la somma di tutti gli input[1]. Se la somma è $\geq v$, allora l'output sarà 1, altrimenti 0.

Questo modello è tale da poter computare un AND ($v = n$ e n input eccitatori), un OR ($v = 1$ e n input eccitatori, vedere figura 2.1), ma non uno XOR!

2.1.1 Limitazioni del modello di McCulloch-Pitt

- Non esiste un modo automatico di fare training.
- Gli input sono binari.
- Gli input hanno tutti lo stesso peso.
- Tutte le funzioni computabili sono linearmente separabili.

2.2 Percettrone - Rosenblatt

Il successore del modello di McCulloch-Pitt è il **percettrone**. Questo modello risolve il problema dei pesi, assegnando ad ogni ingresso un peso differente, e introduce una procedura di apprendimento automatico per stabilire i pesi in maniera opportuna.

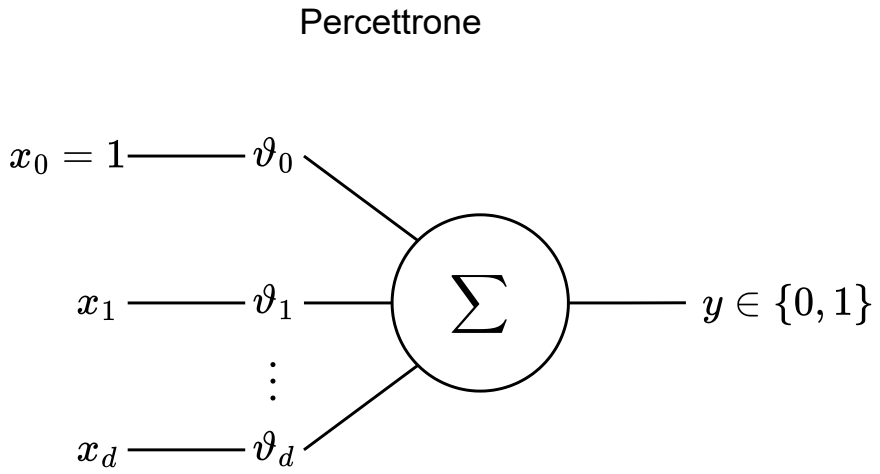

Figura 2.2: Modello di Rosenblatt, noto come Percettrone

Le componenti sono:

- Features $x_1, x_2, \ldots, x_d$ normalizzate in $[0, 1]$. Ciascuno di questi input ha associato un peso $\vartheta_1, \ldots, \vartheta_d$.
- Una threshold ϑ_0, una soglia.
- Un output binario.

[1] posso considerare anche gli input degli inibitori in quanto = 0.

- Una **procedura di learning automatico** per stabilire i parametri (il peso di ciascun input).

Il comportamento del percettrone sarà il seguente:

$$f(x_1, \ldots, x_d) = \begin{cases} 1 & \sum_{j=1} x_j v_j \geq \vartheta_0 \\ 0 & \text{altrimenti} \end{cases}$$

Per semplificare la computazione, verrà aggiunta una feature $x_0 = 1$ con peso ϑ_0, uguale alla threshold. Questo ci aiuterà a scrivere la funzione con la seguente notazione:

$$f(x_1, \ldots, x_d) = \begin{cases} 1 & \sum_{j=1} x_j v_j \geq 0 \\ 0 & \text{altrimenti} \end{cases}$$

E poi come prodotto matriciale:

$$f(x) = [\vartheta^T \overline{x} > 0]$$

dove ϑ^T è il vettore dei pesi. Supponiamo ora di avere 2 parametri, andremo a ottenere:

$$\vartheta_0 + x_1\vartheta_1 + x_2\vartheta_2 \geq 0$$

e con dei semplici passaggi, possiamo tracciare una retta nello spazio, chiamata **decision boundary**, che dividerà lo spazio in due parti, quella per cui $f(x_1, \ldots, x_d) = 1$, e quella per cui $f(x_1, \ldots, x_d) = 0$

$$x_2 = x_1 \frac{\vartheta_1}{\vartheta_2} + \frac{\vartheta_0}{\vartheta_2}$$

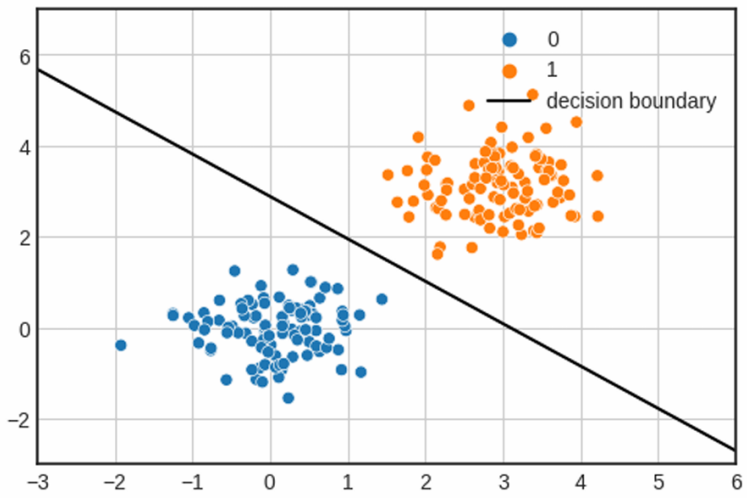

Figura 2.3: Decision boundary del percettrone in 2D. I punti arancioni sono classificati come 1, quelli blu come 0 e la retta al centro è la *decision boundary*.

Questa retta, in un task di classificazione, suddivide lo spazio in due classi. In uno spazio 2D è una retta, in uno spazio 3D un piano. Lo spazio dovrà essere **linearmente separabile**.

2.2.1 Processo generale di training del percettrone

Descriviamo la procedura di training nel seguente modo:

1. Inizializzazione casuale dei pesi $\vartheta_1, \ldots, \vartheta_d \in \mathbb{R}$.
2. Computa $\forall x^{(i)}$ il valore di $\hat{y}^{(i)}$.
3. Confronta $\hat{y}^{(i)}$ con $y^{(i)}$
   - Se $\hat{y}^{(i)} = y^{(i)}$: non fare nulla.
   - Se $\hat{y}^{(i)} \neq y^{(i)}$: vanno aggiornati i pesi. Analizziamo come modificare i pesi, in funzione dei risultati.
4. Aggiornamento dei pesi:

$\hat{y}^{(i)}$	$y^{(i)}$	Cosa fare
0	0	ok!
0	1	riduci i pesi.
1	0	aumenta i pesi.
1	1	ok!

   Questo sposterà la retta in maniera opportuna, convergendo ad un risultato opportuno per il dataset di training.

Questa descrizione generale riflette a grandi linee il processo, tuttavia è opportuno conoscere la procedura dal punto di vista matematico, che riflette poi l'implementazione effettiva.

2.2.2 Implementazione del training

L'aggiornamento dei pesi avviene nel seguente modo:

$$\underbrace{\hat{\vartheta}_j}_{\text{nuovo peso}} \leftarrow \vartheta_j + (y^{(i)} - \hat{y}^{(i)})x_j$$

Chiaramente, il percettrone non dovrà modificare i suoi pesi se $\hat{y}^{(i)} = y^{(i)}$. Questa formula lo contempla, in quanto $y^{(i)} - \hat{y}^{(i)} = 0$: il peso non verrà aggiornato. Si può aggiungere un parametro α, detto **learning rate**. Il percettrone standard non lo prevede.

$$\hat{\vartheta}_j \leftarrow \vartheta_j + \alpha(y^{(i)} - \hat{y}^{(i)})x_j$$

Scelto in maniera opportuna, potrebbe migliorare l'apprendimento e la convergenza.

2.2.3 Versione di Adaline

Una versione successiva dell'algoritmo di ricalcolo dei pesi, per stabilire con maggiore precisione di quanto incrementare o decrementare i pesi, stabilisce che

$$\hat{\vartheta}_j \leftarrow \vartheta_j + \left(y^{(i)} - \sum_i \vartheta_i \cdot x_i \right) x_j$$

Osserviamo che, per ogni esempio i-esimo, l'uscita desiderata $y^{(i)}$ può assumere solo due valori, 0 oppure 1. Allo stesso tempo, la somma pesata

$$\sum_i \vartheta_i \cdot x_i$$

rappresenta una combinazione lineare normalizzata degli ingressi, e pertanto assume valori compresi nell'intervallo $[0, 1]$. Ne consegue che la differenza tra il valore target $y^{(i)}$ e la somma pesata non potrà che appartenere all'intervallo $[-1, 1]$. In altre parole, se il neurone commette l'errore massimo possibile, questo sarà pari a 1 in valore assoluto, cioè la distanza più grande tra un'uscita binaria (0 o 1) e un valore previsto all'interno di $[0, 1]$.

2.3 Modelli lineari

Il percettrone funziona funziona correttamente solo se i dati sono *linearmente separabili*, ovvero se esiste una frontiera lineare che separa le due classi. Ad esempio, il problema AND è linearmente separabile.

2.3.1 Problema della separabilità non lineare

Questa parte è un approfondimento teorico sulla separabilità lineare. Può essere saltata senza problemi.

Definizione (separabilità lineare). Sia $X = \{x^{(1)}, \ldots, x^{(m)}\} \subset \mathbb{R}^d$ e sia $y \in \{0, 1\}^m$ un insieme di etichette. Diciamo che i due insiemi di punti $A = \{x^{(i)} : y^{(i)} = 1\}$ e $B = \{x^{(i)} : y^{(i)} = 0\}$ sono *linearmente separabili* se esistono $w \in \mathbb{R}^d$ e $b \in \mathbb{R}$ tali che

$$w^\top x > b \quad \forall x \in A, \qquad w^\top x < b \quad \forall x \in B.$$

Equivalente: esiste un iperpiano $w^\top x = b$ la cui parte positiva contiene tutti i punti di A e la parte negativa tutti i punti di B.

Quante funzioni booleane sono linearmente separabili? Per m argomenti binari esistono 2^{2^m} funzioni booleane possibili; solo una parte è realizzabile con un singolo

classificatore lineare. Dati noti:

$$\begin{aligned} m = 2 &\Rightarrow \text{14 su 16 sono separabili linearmente,} \\ m = 3 &\Rightarrow \text{104 su 256 sono separabili linearmente,} \\ m = 4 &\Rightarrow \text{1882 su 65536 sono separabili linearmente.} \end{aligned}$$

Non esiste una formula chiusa semplice che dia, in funzione di m, quante funzioni sono linearmente separabili; tuttavia si osserva che, all'aumentare di m, la *frazione* di funzioni separabili decresce rapidamente.

Dicotomie di un insieme di punti. Dato $X = \{x^{(1)}, \ldots, x^{(m)}\} \subset \mathbb{R}^d$, le possibili etichettature (o *dicotomie*) sono 2^m:

$$\mathcal{Y} = \{0, 1\}^m.$$

Solo una parte di queste dicotomie è realizzabile mediante una frontiera lineare. Con d fissata, il numero di dicotomie realizzabili cresce in modo polinomiale in m (ordine al piu m^d), mentre il totale cresce esponenzialmente (2^m); di conseguenza,

$$\Pr\{\text{dicotomia separabile linearmente}\} \longrightarrow 0 \quad \text{quando } m \gg d.$$

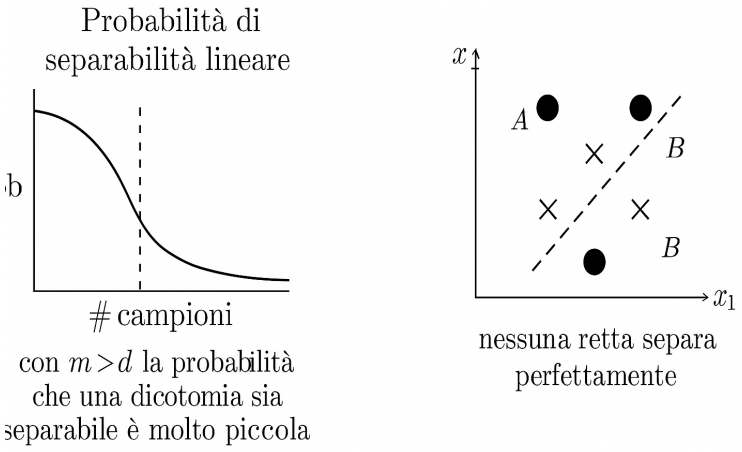

Figura 2.4: Separabilità non lineare. A sinistra: andamento qualitativo della probabilità che una dicotomia sia linearmente separabile al crescere del numero di campioni m (con dimensione d fissata); a destra: esempio in $\mathbb{R}^2$ di punti non separabili con un singolo iperpiano.

Intuitivamente: se il numero di campioni cresce molto a parità di dimensione d, è sempre meno probabile che una separazione perfetta con una sola retta/iperpiano esista.

2.3.2 Problema dello XOR

Supponiamo però di voler computare lo XOR logico. Per farlo, costruiamo la tabella:

x_1	x_2	$XOR(x_1, x_2)$
0	0	0
0	1	1
1	0	1
1	1	0

Il problema XOR non è linearmente separabile (immagine 2.5): i quattro punti $(0,0), (1,0), (0,1), (1,1)$ con etichetta di uscita $0, 1, 1, 0$ non possono essere separati da una singola frontiera lineare.
Un percettrone (che produce solo confini di decisione lineari) non riesce quindi a risolverlo.

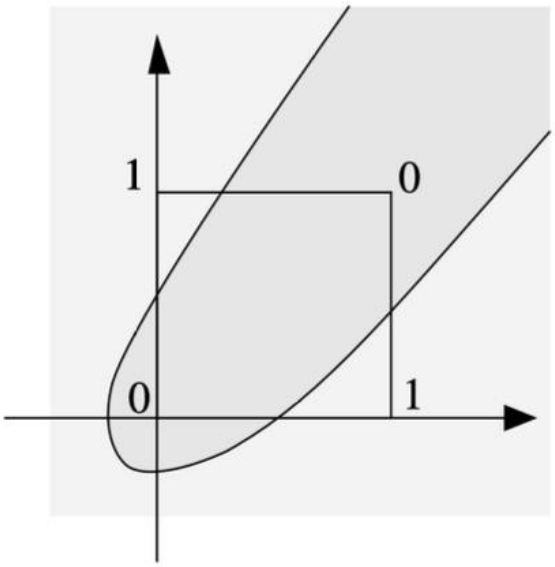

Figura 2.5: Schema del problema XOR: i vertici $(0,1)$ e $(1,0)$ appartengono alla classe 1, $(0,0)$ e $(1,1)$ alla classe 0. Nessuna frontiera lineare può separare le classi; è necessaria una frontiera non lineare (ottenibile con uno strato nascosto).

Dimostrazione della non-linearità dello XOR. Assumendo la tabella di verità dello XOR e un percettrone con pesi v_1, v_2 e soglia v_0, le condizioni di attivazione diventano

$$1 \cdot v_1 + 0 \cdot v_2 > v_0,$$
$$1 \cdot v_1 + 1 \cdot v_2 < v_0,$$
$$0 \cdot v_1 + 1 \cdot v_2 > v_0,$$
$$0 \cdot v_1 + 0 \cdot v_2 < v_0.$$

Da cui si ottiene

$$2\, v_0 < v_1 + v_2 < v_0,$$

ovvero una contraddizione: non esiste configurazione dei parametri di un singolo percettrone che risolva XOR.

Rete di percettroni. La soluzione consiste nell'usare una *rete di percettroni* (uno strato nascosto) che trasformi lo spazio in modo da rendere il problema linearmente separabile all'uscita. Un esempio minimale è:

$$f_1(x_1, x_2) = [\, x_1 - x_2 \geq 0.5 \,],$$
$$f_2(x_1, x_2) = [\, x_2 - x_1 \geq 0.5 \,],$$
$$f_3(f_1, f_2) = [\, 1 \cdot f_1 + 1 \cdot f_2 \geq 0.5 \,],$$

dove $[\cdot]$ indica una funzione soglia (vale 1 se la condizione è vera, 0 altrimenti). Le funzioni f_1 e f_2 realizzano una trasformazione non lineare degli input; nello spazio così trasformato l'uscita f_3 è ottenuta tramite una semplice soglia lineare.

Infine, sebbene reti di percettroni possano risolvere XOR, l'algoritmo di apprendimento del singolo percettrone non è direttamente applicabile a reti multistrato; storicamente ciò contribuì al primo *AI winter* e la situazione migliorò con la formalizzazione della *retropropagazione* (*backpropagation*).

Capitolo 3

Regressione

Definiamo il compito di regressione in termini di come un algoritmo elabora un esempio di input e produce un valore (o vettore) reale di output. Formalmente, vogliamo apprendere una funzione

$$h_\theta : \mathbb{R}^{d_1} \to \mathbb{R}^{d_2}, \qquad d_1 \geq 1,\ d_2 \geq 1,$$

a partire da un dataset supervisionato

$$D = \{(x^{(i)}, y^{(i)})\}_{i=1}^{m}, \qquad x^{(i)} \in \mathbb{R}^{d_1},\ y^{(i)} \in \mathbb{R}^{d_2}.$$

3.1 Ingredienti del task

- **Task**: predire valori reali a partire da input reali.
- **Modello**: ipotesi parametrica h_θ che mappa input in output.
- **Dati**: coppie etichettate $(x^{(i)}, y^{(i)})$.
- **Algoritmo di learning**: metodo per stimare θ (ottimizzazione).
- **Funzione di loss**: misura lo scarto tra predizioni e target.
- **Valutazione**: metriche su validation/test per giudicare il modello.

3.1.1 Tipi di regressione

- **Semplice (univariata)**: $h_\theta : \mathbb{R} \to \mathbb{R}$ ($d_1 = d_2 = 1$).
- **Multipla**: $h_\theta : \mathbb{R}^{d_1} \to \mathbb{R}$ con $d_1 > 1$.
- **Multivariata**: $h_\theta : \mathbb{R}^{d_1} \to \mathbb{R}^{d_2}$ con $d_2 > 1$ (più uscite).

3.2 Regressione lineare semplice

Nel caso $d_1 = d_2 = 1$ modelliamo la relazione tra un singolo ingresso $x \in \mathbb{R}$ e un'uscita reale $y \in \mathbb{R}$ con una retta:

$$h_\theta(x) = \theta_0 + \theta_1\, x,$$

dove θ_0 è l'intercetta e θ_1 la pendenza. "Imparare" significa scegliere $\theta = (\theta_0, \theta_1)$ in modo che le predizioni siano vicine ai corrispondenti target.

3.2.1 Funzione di loss

Misuriamo la qualità della retta con l'errore medio quadratico (MSE):

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2.$$

Il fattore $1/2$ è una costante di comodo che semplifica le derivate; il quadrato rende positivi tutti i contributi ed enfatizza gli errori grandi. In regressione lineare J è convessa rispetto ai parametri, quindi il minimo è unico (se c'è variabilità negli $x^{(i)}$).

3.3 Regressione lineare multipla

Estendiamo al caso con più variabili in ingresso. Dato il vettore $x = (x_1, \ldots, x_n)$, usiamo un parametro per ogni dimensione più il bias:

$$f(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n.$$

Per semplicità poniamo $x_0 = 1$ e definiamo il vettore esteso $x = (x_0, x_1, \ldots, x_n)^\top$; allora

$$f(x) = \sum_{i=0}^{n} \theta_i x_i \;=\; \boldsymbol{\theta}^\top x.$$

Questo modello è, in sostanza, un *percettrone senza soglia*: la combinazione lineare $\boldsymbol{\theta}^\top x$ è l'uscita continua del regressore.

3.4 Feature scaling

Per far funzionare bene (e in fretta) la discesa del gradiente, le feature vanno portate su scale simili. Due opzioni comuni:

$$\text{z-scoring:} \quad x_j \leftarrow \frac{x_j - \mu_j}{\sigma_j}, \qquad \text{min–max:} \quad x_j \leftarrow \frac{x_j - x_j^{\min}}{x_j^{\max} - x_j^{\min}}.$$

Le statistiche $(\mu_j, \sigma_j, x_j^{\min}, x_j^{\max})$ si calcolano solo sul training e si riusano (senza ricalcolarle) su validation/test, per evitare data leakage.

3.5 Algoritmo di discesa del gradiente

L'obiettivo è scegliere θ che minimizza $J(\theta)$. La *discesa del gradiente* è un metodo iterativo che aggiorna i parametri nella direzione di massima diminuzione di J.

3.5.1 Versione batch per la regressione lineare

Sia $X \in \mathbb{R}^{m\times(n+1)}$ la design matrix con prima colonna di 1, $y \in \mathbb{R}^m$ il vettore dei target e $\hat{y} = X\theta$ le predizioni. La loss è

$$J(\theta) = \frac{1}{2m}\|X\theta - y\|_2^2, \qquad \nabla_\theta J(\theta) = \frac{1}{m}X^\top(X\theta - y).$$

Pseudocodice.

1. **Inizializza** in modo casuale: $\theta^{(0)} = (\theta_0, \dots, \theta_n)^\top$.
2. **Calcola le derivate parziali**:

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}\left(\theta^\top \tilde{x}^{(i)} - y^{(i)}\right)\tilde{x}_j^{(i)}, \qquad j = 0, \dots, n,$$

   dove $\tilde{x}^{(i)} = (1, x_1^{(i)}, \dots, x_n^{(i)})^\top$.
3. **Aggiorna** i parametri (passo di ampiezza $\alpha > 0$):

$$\theta_j \leftarrow \theta_j - \alpha\,\frac{\partial J(\theta)}{\partial \theta_j}, \qquad j = 0, \dots, n.$$

4. **Ripeti** i passi 2–3 finché non è soddisfatto un criterio d'arresto (iterazioni massime, $\|\nabla J(\theta)\|$ sotto soglia, variazione di J piccola).

In forma compatta: $\theta \leftarrow \theta - \frac{\alpha}{m}X^\top(X\theta - y)$.

Equazioni normali (soluzione chiusa). Se $X^\top X$ è invertibile, il minimo di J è

$$\theta^\star = (X^\top X)^{-1}X^\top y.$$

Utile per problemi piccoli; per n o m grandi conviene la discesa del gradiente.

3.6 Regressione polinomiale e feature mapping

Se la relazione input–output è non lineare, usiamo un mapping $\phi : \mathbb{R}^n \to \mathbb{R}^p$ (poteri e interazioni) e poi una regressione *lineare nei parametri* nello spazio trasformato:

$$h_\theta(x) = \theta^\top \phi(x).$$

Esempio (grado 2, due feature):

$$\phi(x_1, x_2) = \big(1,\ x_1,\ x_2,\ x_1^2,\ x_2^2,\ x_1x_2\big).$$

Gradi più alti aumentano la capacità (meno underfitting) ma anche il rischio di overfitting e i costi: il numero di termini cresce rapidamente con n e con il grado.

3.7 Regolarizzazione

Per ridurre l'overfitting penalizziamo pesi grandi. In **Ridge** (L2) la loss è

$$J_\lambda(\theta) = \frac{1}{2m}\|X\theta - y\|_2^2 + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2,$$

dove tipicamente θ_0 non si penalizza. Aggiornamenti:

$$\theta_0 \leftarrow \theta_0 - \alpha\,\frac{1}{m}\sum_{i=1}^{m}(\hat{y}^{(i)} - y^{(i)}), \qquad \theta_j \leftarrow \theta_j - \alpha\left[\frac{1}{m}\sum_{i=1}^{m}(\hat{y}^{(i)} - y^{(i)})\tilde{x}_j^{(i)} + \frac{\lambda}{m}\theta_j\right] \ (j \geq 1).$$

3.8 Regressione multivariata

Se $d_2 > 1$, stimiamo più uscite in parallelo. Con $Y \in \mathbb{R}^{m\times d_2}$ (righe $y^{(i)\top}$) e $\Theta \in \mathbb{R}^{(n+1)\times d_2}$, il modello è $\hat{Y} = X\Theta$. L'MSE matriciale è $J(\Theta) = \frac{1}{2m}\|X\Theta - Y\|_F^2$. Le colonne di Θ (una per uscita) si ottimizzano in modo indipendente; la soluzione chiusa generalizza a

$$\Theta^\star = (X^\top X + \lambda\tilde{M})^{-1}X^\top Y \qquad \text{(con Ridge opzionale)}.$$

3.9 Valutazione

Metriche tipiche:

$$\mathrm{MSE} = \frac{1}{m}\sum_{i=1}^{m}\big(\hat{y}^{(i)} - y^{(i)}\big)^2, \quad \mathrm{RMSE} = \sqrt{\mathrm{MSE}}, \quad \mathrm{MAE} = \frac{1}{m}\sum_{i=1}^{m}|\hat{y}^{(i)} - y^{(i)}|.$$

R^2 (coefficiente di determinazione):

$$R^2 = 1 - \frac{\sum_i \big(y^{(i)} - \hat{y}^{(i)}\big)^2}{\sum_i \big(y^{(i)} - \bar{y}\big)^2}.$$

REC curve: ordinando gli errori assoluti $e_i = |\hat{y}^{(i)} - y^{(i)}|$ e tracciando, al variare di ε, la frazione cumulativa di esempi con errore $\leq \varepsilon$, si ottiene una curva di facilissima lettura; un'AUC maggiore indica in genere prestazioni migliori.