

Laboratorio di Intelligenza Artificiale

Elementi di Intelligenza Artificiale in Python

Luciano Capitanio

2024-11-25

Table of contents

Benvenuti	5
License	5
Introduzione	6
1 Turing vs Searle: Un'Analisi del Pensiero Computazionale e della Coscienza	8
Alan Turing e il Test di Turing	9
1.0.1 Critiche al Test di Turing	10
John Searle e l'Argomento della Stanza Cinese	11
1.1 Conclusioni	12
2 Algoritmi	13
2.1 Inferenza Logica	13
2.1.1 Proposizioni Logiche	14
2.1.2 Calcolo delle Proposizioni Logiche	15
2.1.3 Basi della Conoscenza	17
2.1.4 Sistemi basati sulla conoscenza	17
2.1.5 Semplice Sistema Esperto in ambito penale	19
2.2 Inferenza Probabilistica	23
2.2.1 Inferenza probabilistica e La teoria delle probabilità	24
2.2.2 Calcolo della probabilità incondizionata o a priori	26
2.2.3 Variabili aleatorie	27
2.2.4 Distribuzioni di probabilità	28
2.2.5 Probabilità congiunta	28
2.2.6 Indipendenza delle variabili aleatorie	28
2.2.7 Negazione	29
2.2.8 Inclusione	29
2.2.9 Marginalizzazione	29
2.2.10 probabilità condizionata	29
2.2.11 Condizionamento	32
2.3 Inferenza Bayesiana	34
2.3.1 Il Teorema di Bayes	34
2.3.2 Applicazioni Pratiche	35
2.3.3 reti di Bayes	38

2.4	Algoritmi di Ricerca	44
2.4.1	Glossario della ricerca	45
2.4.2	Problemi di ricerca	45
2.4.3	Algoritmo “generale” di ricerca	46
2.4.4	Strategie di ricerca non informate	47
2.4.5	Algoritmi di ricerca informati	57
2.5	Algoritmi Equitativi	62
2.5.1	agenti partecipanti	64
2.5.2	beni	64
2.5.3	Regole e assunzioni	65
2.5.4	Matematica elementare per algoritmi di ripartizione equa	66
2.5.5	Algoritmi di Ripartizione Equitativa	66
2.5.6	Gli algoritmi di divisione equa in letteratura	75
2.6	Algoritmi Predittivi	78
2.6.1	Definizione del Problema	81
2.6.2	Raccolta dei Dati	81
2.6.3	Pre-elaborazione dei Dati	82
2.6.4	Divisione dei Dati	83
2.6.5	Scelta dell’ algoritmo	84
2.6.6	Addestramento del modello	84
2.6.7	Valutazione del modello	85
2.6.8	Ottimizzazione degli iperparametri	86
2.6.9	Deployment	87
2.6.10	Monitoraggio e manutenzione	88
3	Machine learning	90
3.1	Apprendimento Supervisionato	91
3.1.1	Definizione e Principi di Base	91
3.1.2	Classificazione	91
3.1.3	Regressione	92
3.1.4	Algoritmi Principali dell’Apprendimento Supervisionato	93
3.1.5	Apprendimento per Rinforzo	96
3.1.6	Overfitting e Underfitting	97
3.1.7	Valutazione dei Modelli	98
3.2	Apprendimento Non Supervisionato	102
3.2.1	Clustering	102
3.2.2	Riduzione della Dimensionalità	103
3.2.3	Algoritmi Principali	104
3.2.4	Applicazioni	104
3.3	Bias	105
3.3.1	Tipologie di Bias	105
3.3.2	Cause e Impatti del Bias	107
3.3.3	Tecniche di Mitigazione del Bias	108

3.4	Reti Neurali	109
3.4.1	Regressione Lineare e Logistica	110
3.4.2	Percettrone	118
3.4.3	Deep learning	123
Appendices		145
A	Elementi di Python	145
A.1	Perché Python è utile in ambito giuridico?	145
A.2	Introduzione alla sintassi di Python	145
A.2.1	1. Variabili e Tipi di Dati	145
A.2.2	2. Condizioni: Prendere Decisioni	146
A.2.3	3. Cicli: Ripetere Azioni	146
A.2.4	4. Gestione degli Errori	146
A.2.5	5. Strutture Dati Utili	147
A.3	Applicazioni Pratiche	147
A.3.1	Creazione di Moduli Personalizzati	148
A.4	Conclusione	148

Benvenuti

Questo sito è la versione web del libro “Laboratorio di Intelligenza Artificiale in Python” che vuole essere una guida pratica pensata per professionisti legali senza esperienza di programmazione. Il libro introduce in modo semplice l’uso di Python per applicazioni nell’ambito giuridico, aiutando a comprendere come l’intelligenza artificiale può trasformare il settore legale.

License

L’uso di questo sito è **gratuito** ed i suoi contenuti sono sottoposto a una licenza [Creative Commons Attribution-NonCommercial-NoDerivs 4.0](#). Se vuoi avere una **copia fisica** o una **versione e-book** del libro puoi ordinarlo da ...; è stato pubblicato il ...

Se vuoi contribuire, è gradito ogni commento e correzione di refusi, puoi farlo su github.com/capitanio/laboratorio-ia.

[Luciano Capitanio](#)

Introduzione

L'Intelligenza Artificiale (IA) è un campo di studio che si occupa della creazione di macchine e software in grado di mostrare comportamenti intelligenti. Questa disciplina è stata definita per la prima volta da John McCarthy nel 1955 come “la scienza e l'ingegneria della costruzione di macchine intelligenti” (McCarthy et al. 1955).

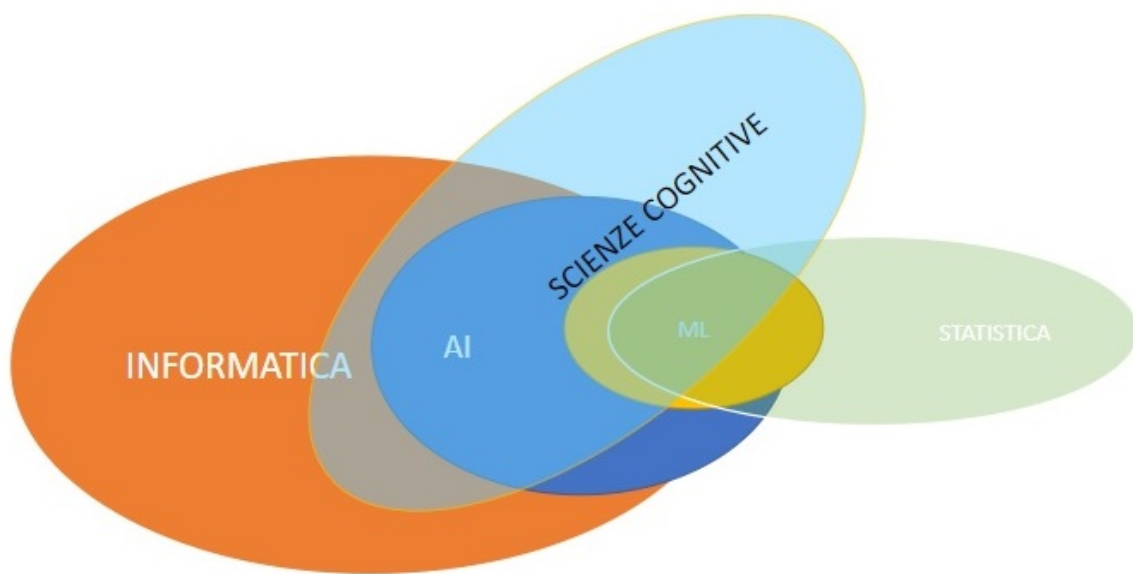


Figure 1: IA è una disciplina di confine

L'IA è sia una scienza che una tecnica, un'area di ricerca in cui convergono diverse discipline, tra cui l'informatica, la statistica e le scienze cognitive.

L'IA si divide principalmente in due categorie: IA debole e IA forte. L'IA debole si riferisce a sistemi progettati per eseguire compiti specifici, come il riconoscimento vocale o la guida autonoma. L'IA forte, invece, riguarda sistemi che potrebbero possedere una coscienza e un'intelligenza simile a quella umana (Searle 1980).

L'IA sta trasformando molti settori, tra cui l'assistenza sanitaria, l'istruzione, i trasporti e l'industria. Ad esempio, nel campo sanitario, l'IA supporta i medici nella diagnosi delle malattie e nella creazione di piani di trattamento personalizzati (Jiang et al. 2017).

Nonostante i progressi, l'IA presenta diverse sfide, tra cui questioni etiche come la privacy dei dati e l'impatto sull'occupazione, nonché problemi tecnici legati alla comprensione del linguaggio naturale e allo sviluppo di algoritmi di apprendimento automatico efficaci (Russell and Norvig 2016).

In conclusione, l'IA è un campo in rapida evoluzione con il potenziale di rivoluzionare la società. Tuttavia, è essenziale affrontare le sfide etiche e tecniche per garantire che i suoi benefici siano raggiunti in modo sicuro e sostenibile.

1 Turing vs Searle: Un'Analisi del Pensiero Computazionale e della Coscienza

💡 Da ricordare!

1. Alan Turing:

- **Contributo principale:** Propose il “Test di Turing” come metodo per determinare l'intelligenza artificiale.
- **Idea chiave:** L'intelligenza può essere valutata dalla capacità di una macchina di imitare un essere umano in una conversazione.

2. John Searle:

- **Contributo principale:** Ideò l'esperimento mentale della “Stanza Cinese”.
- **Critica chiave:** L'IA può simulare la comprensione ma non avere una vera coscienza.

3. Temi fondamentali:

- Differenza tra imitazione e comprensione reale.
- Implicazioni filosofiche dell'intelligenza artificiale.

Alan Turing e John Searle sono due figure centrali nella riflessione sull'intelligenza artificiale e la coscienza. Turing ha introdotto il famoso “Test di Turing” (Turing 1950), concentrandosi sulla capacità delle macchine di simulare il comportamento umano, mentre Searle, con il suo “Argomento della Stanza Cinese”, ha criticato l'idea che una macchina possa davvero comprendere o avere coscienza (Searle 1980).

Alan Turing e il Test di Turing

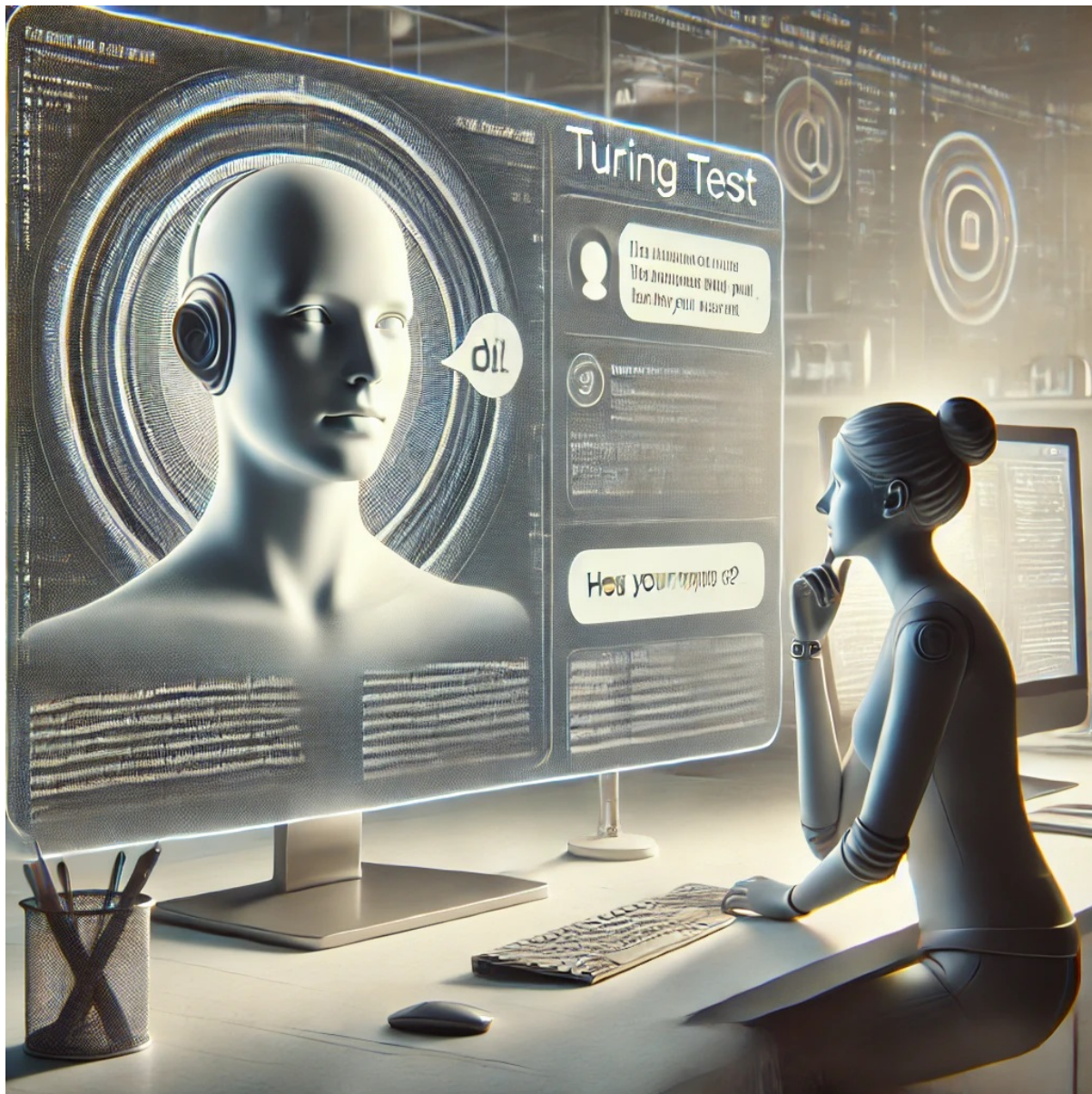


Figure 1.1: Immagine generata da DALL-E del Test di Turing

Alan Turing, pioniere dell'informatica, propose il “Test di Turing” nel 1950 come criterio per valutare se una macchina potesse essere considerata intelligente (Turing 1950). Secondo Turing, se un essere umano che comunica con una macchina attraverso uno schermo non riesce a

distinguerla da un altro essere umano, allora la macchina può essere considerata intelligente.

Punti chiave del Test di Turing: - Il test misura la capacità di imitare il comportamento umano, non la comprensione o la coscienza. - Ha ispirato lo sviluppo di chatbot e sistemi di IA conversazionale.

1.0.1 Critiche al Test di Turing

Molti filosofi e scienziati sostengono che il test non catturi la vera natura dell'intelligenza o della coscienza. John Searle è uno dei principali critici.

John Searle e l'Argomento della Stanza Cinese



Figure 1.2: Immagine generata da DALL-E del Test della Stanza Cinese

John Searle propose l'esperimento mentale della "Stanza Cinese" nel 1980 come critica al concetto di IA forte. L'idea è la seguente:

- Una persona che non conosce il cinese si trova in una stanza con un manuale di istruzioni

che spiega come rispondere ai messaggi scritti in cinese.

- Usando il manuale, la persona può rispondere correttamente ai messaggi senza comprendere realmente il cinese.
- Searle sostiene che le macchine, similmente, possono elaborare simboli senza comprendere il loro significato.

Conclusione della Stanza Cinese: - L'elaborazione simbolica non equivale alla comprensione. - Questo pone un limite all'idea che una macchina possa essere veramente intelligente.

1.1 Conclusioni

Il confronto tra Turing e Searle evidenzia due prospettive opposte sull'intelligenza artificiale:

- Turing vede l'imitazione come un segno sufficiente di intelligenza.
- Searle insiste sulla distinzione tra simulazione e comprensione reale.

Questo dibattito rimane centrale nella filosofia dell'intelligenza artificiale e continua a ispirare nuove domande e riflessioni.

2 Algoritmi

Gli algoritmi sono il cuore pulsante dell'intelligenza artificiale e svolgono un ruolo cruciale nel trasformare i dati grezzi in informazioni utili. In questo capitolo, esploreremo vari tipi di algoritmi utilizzati nell'IA. Approfondiremo l'inferenza logica, probabilistica e bayesiana, nonché gli algoritmi di ricerca, equitativi e predittivi.

2.1 Inferenza Logica

definizione Treccani: inferenza logica sinonimo di «argomentazione logica» utilizzato per designare il processo di deduzione di una formula A , detta conclusione, a partire da una o più formule, dette premesse. Secondo A. De Morgan, una inferenza è la «produzione di una proposizione come conseguenza necessaria di una o più proposizioni».

L'inferenza logica è un processo fondamentale nel campo della logica, della matematica e della filosofia, utilizzato per derivare conclusioni a partire da premesse o informazioni date. Questo processo può essere visto come un mezzo per scoprire nuove verità o per confermare la validità di affermazioni esistenti. L'inferenza logica si suddivide principalmente in due categorie: deduttiva e induttiva.

L'inferenza deduttiva è quella in cui la conclusione deriva necessariamente dalle premesse; se le premesse sono vere, la conclusione non può che essere vera. Un classico esempio di inferenza deduttiva è il sillogismo: “Tutti gli uomini sono mortali; Socrate è un uomo; quindi, Socrate è mortale.” In questo caso, la verità delle premesse garantisce la verità della conclusione.

L'inferenza induttiva, invece, opera diversamente: partendo da osservazioni specifiche o da una serie di dati, arriva a conclusioni più generali, che non sono necessariamente certe ma probabili. Ad esempio, se si osserva che il sole è sorto ogni giorno, si potrebbe inferire che il sole sorgerà anche domani. Questa forma di inferenza è molto utilizzata nella scienza, dove gli scienziati formulano ipotesi basate su dati osservati e sperimentali.

Un altro tipo di inferenza logica è l'abduzione, che implica la formazione della migliore spiegazione possibile data un insieme di osservazioni. Questo tipo di inferenza è spesso utilizzato nella diagnosi medica, nella ricerca scientifica e nelle indagini criminali, dove si cerca di spiegare i dati osservati nel modo più coerente possibile.

L'inferenza logica è strettamente legata al concetto di validità e di correttezza degli argomenti. Un'argomentazione è valida se la sua struttura logica è tale che, qualora le premesse siano vere, anche la conclusione deve essere vera. Tuttavia, un'argomentazione può essere valida senza essere corretta; per essere corretta, deve avere anche premesse vere. Ad esempio, l'argomentazione "Tutti gli unicorni sono verdi; io possiedo un unicorno; quindi, il mio unicorno è verde" è valida dal punto di vista logico, ma non è corretta perché le premesse non sono vere.

L'inferenza logica è alla base di molti sistemi di intelligenza artificiale e di calcolo automatico, dove gli algoritmi vengono progettati per inferire nuove informazioni a partire da dati iniziali. Nei sistemi esperti, per esempio, vengono utilizzate regole di inferenza per simulare il processo decisionale umano. In conclusione, l'inferenza logica è uno strumento potente e versatile che permea molte aree del pensiero umano e della tecnologia, consentendo di avanzare nella conoscenza e nella comprensione del mondo che ci circonda. L'inferenza logica è una tecnica fondamentale dell'intelligenza artificiale che utilizza le regole logiche per derivare nuove informazioni da quelle esistenti. Nella giurisprudenza, l'inferenza logica può essere utilizzata per analizzare le leggi e determinare le conseguenze logiche delle azioni legali.

I sistemi esperti - Negli anni '80, l'inferenza logica è stata fondamentale nello sviluppo dei sistemi esperti, strumenti avanzati di intelligenza artificiale progettati per risolvere problemi complessi emulando il ragionamento umano. Due noti prodotti commerciali di quel periodo sono stati MYCIN, un sistema esperto per la diagnosi di infezioni del sangue, e XCON, utilizzato per configurare sistemi di computer VAX di Digital Equipment Corporation. MYCIN e XCON sfruttavano regole di inferenza per elaborare informazioni e fornire raccomandazioni o soluzioni, dimostrando l'efficacia dell'inferenza logica in applicazioni pratiche e commerciali >
- ["Rule-based Expert Systems : The MYCIN Experiments of the Stanford Heuristic Programming Project"](#), edited by Bruce G. Buchanan, Edward H. Shortliffe (AddisonWesley, 1984) - ["RI: an Expert in the Computer Systems Domain"](#)

2.1.1 Proposizioni Logiche

Le proposizioni logiche sono dichiarazioni atomiche che possono essere valutate come vere o false. Le proposizioni possono essere combinate utilizzando operatori logici come AND, OR, NOT, IMPLIES, che permettono di costruire regole complesse rappresentate da formule logiche.

Ecco alcuni esempi di proposizioni logiche:

p: "Il sole è luminoso" (Vero) q: "La Luna è fatta di formaggio" (Falso) r: "Se piove, allora la strada sarà bagnata" (Condizionale)

2.1.2 Calcolo delle Proposizioni Logiche

Le proposizioni logiche possono essere manipolate utilizzando vari operatori logici che eseguono operazioni specifiche:

Congiunzione (AND - \wedge): L'operatore AND restituisce vero solo quando entrambe le proposizioni coinvolte sono vere. Ad esempio, se abbiamo due proposizioni p e q , $p \wedge q$ è vero solo se entrambe p e q sono vere. La cosiddetta tabella di verità riportata qui sotto consente di vedere come funziona l'operatore AND.

Table 2.1: Tavola della verità per la congiunzione

p	q	$p \wedge q$
False	False	False
False	True	False
True	False	False
True	True	True

Disgiunzione (OR - \vee): L'operatore OR restituisce vero se almeno una delle due proposizioni coinvolte è vera. Ad esempio, $p \vee q$ è vero se p è vero oppure se q è vero oppure se entrambi sono veri. La tabella di verità riportata qui sotto consente di vedere come funziona l'operatore OR.

Table 2.2: Tavola della verità per la disgiunzione

p	q	$p \vee q$
False	False	False
False	True	True
True	False	True
True	True	True

Negazione (NOT - \neg): L'operatore NOT cambia il valore di verità di una proposizione. Ad esempio, $\neg p$ è vero se p è falso e viceversa.

Table 2.3: Tavola della verità per la negazione

p	$\neg p$
False	True
True	False

Implicazione (\rightarrow): L'implicazione è un'operazione logica che collega due proposizioni e stabilisce una relazione di condizionalità. Si rappresenta con il simbolo " \rightarrow " e si legge come "se... allora". In un'implicazione del tipo " $p \rightarrow q$ ", la proposizione p è chiamata l'antecedente e la proposizione q è il conseguente. L'implicazione è falsa solo nel caso in cui l'antecedente è vero e il conseguente è falso. In tutti gli altri casi, l'implicazione è considerata vera. Poiché questa operazione è alla base di molti algoritmi di inferenza, è importante capire come funziona. La tabella di verità riportata qui sotto consente di vedere come funziona l'operatore implicazione.

Table 2.4: Tavola della verità per l'implicazione

p	q	$p \rightarrow q$
False	False	True
False	True	True
True	False	False
True	True	True

Esempio di Implicazione: supponiamo di avere le seguenti proposizioni: - p : Il sole splende. - q : Faccio una passeggiata.

L'implicazione che possiamo formulare è: "Se il sole splende, allora faccio una passeggiata", che si scrive come " $p \rightarrow q$ ". Dalla tabella della verità, possiamo vedere che in tre dei quattro casi l'implicazione " $p \rightarrow q$ " è vera. L'unico caso in cui l'implicazione è falsa è quando il sole splende (p è vero) ma non faccio una passeggiata (q è falso).

Quindi, in base alla logica dell'implicazione, se il sole splende, sto effettivamente facendo una passeggiata o potrei anche non farla (ad eccezione del caso in cui il sole splenda e io non faccio una passeggiata, in cui l'implicazione è falsa).

Implicazione Bilaterale (\leftrightarrow): L'implicazione bilaterale è un'operazione logica che stabilisce che due proposizioni sono equivalenti, cioè che entrambe le proposizioni hanno lo stesso valore di verità. Si rappresenta con il simbolo " \leftrightarrow " e si legge come "se e solo se". L'implicazione bilaterale è vera solo quando le proposizioni hanno lo stesso valore di verità, sia entrambe vere che entrambe false.

Table 2.5: Tavola della verità per l'implicazione bilaterale

p	q	$p \leftrightarrow q$
False	False	True
False	True	False
True	False	False
True	True	True

L'implicazione bilaterale, anche conosciuta come “se e solo se”, è un importante concetto logico che stabilisce che due proposizioni sono logicamente equivalenti, cioè entrambe sono vere o entrambe sono false contemporaneamente.

Esempio di Implicazione Bilaterale: supponiamo di avere le seguenti proposizioni:

- p : Oggi è venerdì.
- q : Domani è sabato.

L'implicazione bilaterale tra p e q può essere scritta come $p \leftrightarrow q$, che si legge come “Oggi è venerdì se e solo se domani è sabato”.

Dalla tabella di verità, possiamo notare che l'implicazione bilaterale “Oggi è venerdì se e solo se domani è sabato” è vera solo nei casi in cui entrambe le proposizioni sono vere (primo e ultimo caso) o entrambe sono false. Se c'è una discrepanza nelle verità delle proposizioni, l'implicazione bilaterale diventa falsa (secondo e terzo caso).

Quindi, nel nostro esempio, l'affermazione “Oggi è venerdì se e solo se domani è sabato” è vera solo quando entrambe le proposizioni sono vere o entrambe sono false, evidenziando l'equivalenza logica tra le due proposizioni nel contesto dell'implicazione bilaterale.

2.1.3 Basi della Conoscenza

La base della conoscenza in un agente a inferenza logica è costituita da proposizioni logiche, che sono affermazioni dichiarative che possono essere vere o false. Le proposizioni possono essere atomiche o composte e sono spesso rappresentate utilizzando variabili proposizionali. Queste variabili assumono valori di verità (vero o falso) e vengono combinate tramite operatori logici per formare regole logiche complesse. La base della conoscenza in un sistema logico definisce le relazioni tra le proposizioni e fornisce le fondamenta per il ragionamento e l'inferenza. Un agente a inferenza logica usa la Base della Conoscenza per giungere a conclusioni circa il mondo che la circonda; Per fare ciò ha bisogno di regole di implicazione logica (\rightarrow): Se $p \rightarrow q$, ovvero se p implica logicamente q , in ogni mondo dove p è vera allora q è vera. È diversa dall'implicazione perché non è un connettivo logico ma una relazione che dice che se p è vera allora q è vera e basta!

2.1.4 Sistemi basati sulla conoscenza

I sistemi basati sulla conoscenza sono strumenti informatici progettati per emulare il processo decisionale umano attraverso l'utilizzo di una base di conoscenza strutturata. Questi sistemi raccolgono, organizzano e utilizzano informazioni specifiche di un dominio per risolvere problemi complessi che richiedono competenza specialistica. Una componente fondamentale è la **base di conoscenza**, che contiene fatti, regole ed euristiche rappresentative del sapere umano in un determinato campo. Il **motore di inferenza** è l'altro elemento chiave: applica regole

logiche ai dati presenti nella base di conoscenza per dedurre nuove informazioni o prendere decisioni informate.

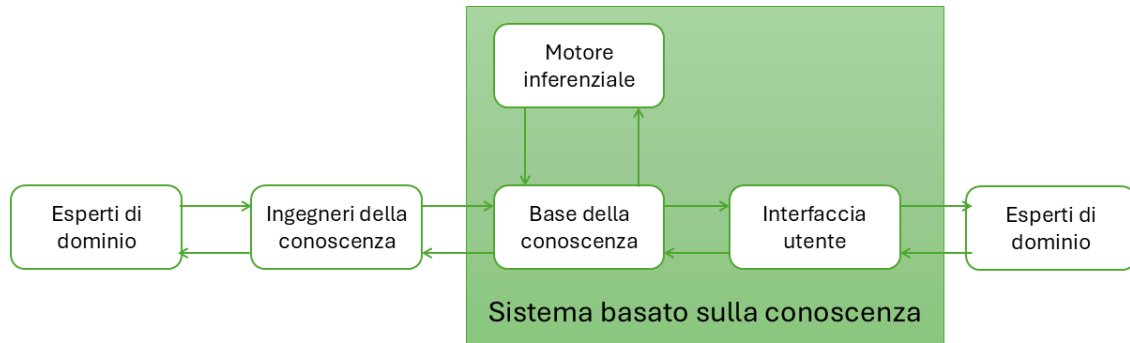


Figure 2.1: Processo di creazione e gestione di un sistema basato sulla conoscenza

Il processo di creazione di un sistema esperto basato sull'inferenza logica inizia con l'acquisizione della conoscenza, dove gli esperti del dominio collaborano per estrarre informazioni e regole rilevanti. Queste conoscenze vengono poi formalizzate nella rappresentazione della conoscenza, utilizzando strutture come regole if-then, ontologie o reti semantiche, che alimentano la base di conoscenza. Il motore di inferenza viene sviluppato per applicare queste regole logiche ai dati forniti, deducendo nuove informazioni o prendendo decisioni informate. La gestione del sistema include l'aggiornamento continuo della base di conoscenza per riflettere nuove scoperte o cambiamenti nel dominio, nonché la verifica e la validazione del sistema per garantirne l'accuratezza e l'affidabilità. Gli utenti interagiscono con il sistema attraverso un'interfaccia che facilita l'inserimento dei dati e la visualizzazione dei risultati, permettendo anche il feedback per miglioramenti futuri.

Questi sistemi trovano applicazione in vari settori, come la medicina, l'ingegneria, la finanza e l'assistenza clienti. Ad esempio, in ambito medico, un sistema basato sulla conoscenza può aiutare nella diagnosi di malattie analizzando sintomi e storie cliniche dei pazienti. L'efficacia di tali sistemi dipende dalla qualità e dall'aggiornamento costante della base di conoscenza, nonché dalla capacità del motore di inferenza di elaborare correttamente le informazioni.

Un vantaggio significativo dei sistemi basati sulla conoscenza è la possibilità di conservare e diffondere l'esperienza di esperti, rendendola accessibile a un pubblico più ampio e contribuendo alla standardizzazione delle pratiche. Tuttavia, la creazione e la manutenzione di una base di conoscenza richiedono notevoli risorse e competenze. Con l'avanzamento dell'intelligenza artificiale e dell'apprendimento automatico, questi sistemi continuano a evolversi, integrando nuove tecniche per migliorare l'efficienza, l'accuratezza e la capacità di apprendimento autonomo nelle loro applicazioni.

2.1.5 Semplice Sistema Esperto in ambito penale

In questo paragrafo, useremo la libreria **SymPy** in Python per creare un semplice sistema esperto basato sull'inferenza logica nell'ambito del diritto penale. Questo sistema aiuterà a determinare se determinati comportamenti costituiscono un reato, in base ai fatti noti e alle norme applicabili. Si noti che la libreria **SymPy** è stata sviluppata per consentire il calcolo simbolico in Python. In questo caso useremo le funzionalità di calcolo simbolico per la rappresentazione della conoscenza, usando le funzionalità di calcolo logico, e per l'inferenza logica.

2.1.5.1 Introduzione

Il diritto penale si basa su norme che definiscono quali comportamenti sono considerati reati e quali elementi devono essere presenti affinché un'azione sia punibile. Un sistema esperto in questo contesto può aiutare a:

- Valutare se un'azione specifica costituisce un reato.
- Identificare gli elementi costitutivi del reato.
- Fornire una base logica per decisioni legali.

Utilizzeremo SymPy per modellare proposizioni logiche, regole legali e per effettuare inferenze.

2.1.5.2 Installazione di SymPy

Assicurati di avere SymPy installato:

```
pip install sympy
```

Se stai utilizzando questo notebook in un ambiente in cui SymPy non è installato, esegui la seguente cella:

```
!pip install sympy
```

2.1.5.3 Concetti di Base nel Diritto Penale

Prima di iniziare, definiamo alcuni concetti chiave:

- **Fatti:** Eventi o azioni specifiche accadute.
 - **Reati:** Comportamenti definiti come illeciti dalla legge penale.
 - **Elementi Costitutivi del Reato:** Condizioni che devono essere soddisfatte perché un comportamento sia considerato un reato (ad esempio, azione, intenzione, nesso causale).
 - **Regole Legali:** Norme che stabiliscono le condizioni in cui un comportamento è punibile.
-

2.1.5.4 Modellazione con SymPy

Passo 1: Importare i Moduli Necessari

Importiamo i moduli necessari da SymPy per lavorare con la logica proposizionale.

```
from sympy import symbols
from sympy.logic.boolalg import And, Or, Not, Implies, Equivalent
from sympy.logic.inference import satisfiable
```

Passo 2: Definire le Proposizioni Logiche

Definiamo le variabili che rappresentano i fatti e gli elementi costitutivi del reato.

```
# Fatti
Azione, Intenzione, NessoCausale = symbols('Azione Intenzione NessoCausale')

# Reato
Omicidio = symbols('Omicidio')
```

Passo 3: Definire le Regole che discendono dal Codice Penale

Ad esempio, secondo il codice penale, l'omicidio richiede:

- **Azione:** Causare la morte di una persona.
- **Intenzione:** Volontà di causare la morte (dolo).
- **Nesso Causale:** La morte è conseguenza dell'azione.

Definiamo la regola:

```
# Regola: Se c'è Azione, Intenzione e Nesso Causale, allora si configura l'Omicidio
regola_omicidio = Implies(And(Azione, Intenzione, NessoCausale), Omicidio)
```

Passo 4: Definire i Fatti Noti

Supponiamo di avere i seguenti fatti:

- Una persona ha compiuto un'azione che ha causato la morte di un'altra.
- Aveva l'intenzione di causare la morte.
- Esiste un nesso causale tra l'azione e la morte.

```
# Fatti noti
fatto1 = Azione # L'azione di causare la morte
fatto2 = Intenzione # Intenzione di causare la morte
fatto3 = NessoCausale # La morte è conseguenza dell'azione
```

Passo 5: Creare la Base di Conoscenza

Combiniamo fatti e regole:

```
# Base di conoscenza
base_conoscenza = And(fatto1, fatto2, fatto3, regola_omicidio)
```

Passo 6: Inferenza Logica

Verifichiamo se, sulla base dei fatti e delle regole, possiamo concludere che si tratta di omicidio.

```
# Verifichiamo se Omicidio è deducibile
ipotesi = And(base_conoscenza, Not(Omicidio))
risultato = satisfiable(ipotesi)

if not risultato:
    print("Si configura il reato di omicidio.")
else:
    print("Non possiamo concludere che si tratti di omicidio.")
```

Si configura il reato di omicidio.

Output atteso:

Si configura il reato di omicidio.

2.1.5.5 Espansione del Sistema

Caso con Mancanza di Intenzione

Supponiamo che l'intenzione non sia presente (ad esempio, si tratta di omicidio colposo).

```
# Fatti noti senza Intenzione
fatto1 = Azione
fatto2 = Not(Intenzione) # Mancanza di intenzione
fatto3 = NessoCausale

# Base di conoscenza aggiornata
base_conoscenza = And(fatto1, fatto2, fatto3, regola_omicidio)
```

Inferenza per Omicidio

```
# Inferenza
ipotesi = And(base_conoscenza, Not(Omicidio))
risultato = satisfiable(ipotesi)

if not risultato:
    print("Si configura il reato di omicidio.")
else:
    print("Non possiamo concludere che si tratti di omicidio.")
```

Non possiamo concludere che si tratti di omicidio.

Output atteso:

Non possiamo concludere che si tratti di omicidio.

2.1.5.5.1 Aggiunta di Altre Regole

Aggiungiamo la regola per l'omicidio colposo:

```
# Definizione del reato di Omicidio Colposo
OmicidioColposo = symbols('OmicidioColposo')

# Regola per Omicidio Colposo: Azione e Nesso Causale senza Intenzione
regola_omicidio_colposo = Implies(And(Azione, Not(Intenzione), NessoCausale), OmicidioColposo)

# Aggiorniamo la base di conoscenza
base_conoscenza = And(fatto1, fatto2, fatto3, regola_omicidio, regola_omicidio_colposo)
```

Inferenza per Omicidio Colposo

```
# Verifichiamo se si configura l'Omicidio Colposo
ipotesi = And(base_conoscenza, Not(OmicidioColposo))
risultato = satisfiable(ipotesi)

if not risultato:
    print("Si configura il reato di omicidio colposo.")
else:
    print("Non possiamo concludere che si tratti di omicidio colposo.")
```

Si configura il reato di omicidio colposo.

Output atteso:

Si configura il reato di omicidio colposo.

2.1.5.6 Conclusione

Abbiamo visto come utilizzare SymPy per modellare un semplice sistema esperto nel campo del diritto penale. Questo esempio illustra come le regole legali e i fatti possono essere formalizzati utilizzando la logica proposizionale, permettendo al sistema di effettuare inferenze logiche.

Ricorda che questo è un modello semplificato e che il diritto penale è complesso e richiede una comprensione approfondita per essere modellato accuratamente. Questo sistema può essere un punto di partenza per sviluppi più avanzati e per esplorare l'intersezione tra intelligenza artificiale e diritto.

2.2 Inferenza Probabilistica

L'inferenza probabilistica utilizza la teoria delle probabilità per fare previsioni o inferenze basate su dati incompleti o incerti. Questo tipo di inferenza è particolarmente utile in tutti i contesti dove le informazioni possono essere incomplete o incerte.

L'agente artificiale in situazioni di incertezza affronta la sfida di prendere decisioni razionali quando non si dispone di informazioni complete o quando le informazioni disponibili sono soggette a variabilità. In tali contesti, l'agente deve essere in grado di gestire e interpretare l'incertezza in modo efficace per agire in modo intelligente e adattivo.

L'incertezza può derivare da diversi fattori, come la natura incompleta delle informazioni, la presenza di rumore nei dati, l'aleatorietà degli eventi o la complessità dei problemi da affrontare.

Gli agenti artificiali, dotati di capacità di ragionamento probabilistico e di inferenza, sono in grado di valutare le conseguenze di diverse azioni in base alle probabilità associate agli eventi futuri e agli esiti attesi.

L'inferenza probabilistica è una tecnica utilizzata nell'ambito dell'Intelligenza Artificiale per prendere decisioni o formulare previsioni basate su informazioni incerte o parziali. In pratica, ciò significa che invece di avere risposte binarie (vero o falso), lavoriamo con probabilità, cioè con il grado di certezza o incertezza riguardo a una determinata affermazione o evento.

Esempio in un contesto legale

Immagina un caso giuridico in cui una persona è accusata di un crimine. Nel sistema giuridico, la giuria deve prendere una decisione sulla colpevolezza o innocenza dell'imputato. Tuttavia, spesso non abbiamo prove definitive o testimonianze che garantiscono una certezza assoluta. In questo contesto, l'inferenza probabilistica può essere applicata. Invece di dire semplicemente "colpevole" o "innocente," i giurati possono assegnare una probabilità alla colpevolezza dell'imputato. Ad esempio, potrebbero dire che ci sono il 70% di probabilità che l'imputato sia colpevole e il 30% di probabilità che sia innocente. Infine, una soglia sulla probabilità di colpevolezza potrebbe dare origine alla sentenza.

Esempio in un generico contesto di diagnosi

Nel caso di una diagnosi (in qualunque settore) è necessario prendere una decisione con conoscenza incerta. Si è in una situazione di incertezza in quanto la lista di situazioni e cause da descrivere non può essere esaustiva (praticamente infinita per la mancanza di conoscenza universale). Non si può usare la logica del primo ordine per gestire la diagnosi perché: - è impossibile elencare l'insieme praticamente infinito di antecedenti e conseguenti per evitare eccezioni - è impossibile avere una conoscenza metodologica completa - è impossibile avere una conoscenza applicativa completa - L'agente non potrà mai agire con una piena consapevolezza di verità e correttezza, avrà solo un grado di credenza sulla bontà delle azioni da intraprendere e dei risultati.

2.2.1 Inferenza probabilistica e La teoria delle probabilità

L'inferenza probabilistica è un processo di ragionamento che utilizza il calcolo delle probabilità per prendere decisioni o formulare previsioni in situazioni in cui le informazioni sono incomplete o incerte. L'inferenza probabilistica è fondamentale in vari campi, come la statistica, l'apprendimento automatico, la medicina, la finanza e molti altri.

Il calcolo delle probabilità è una branca della matematica che si occupa di misurare e analizzare la probabilità di eventi casuali. La probabilità è una misura numerica che descrive la possibilità che un evento specifico accada.

“Il concetto di probabilità è il più importante della scienza moderna, soprattutto perché nessuno ha la più pallida idea del suo significato.” (Bertrand Russel)

La teoria della probabilità assume la stessa assunzione ontologica della logica: - i fatti del mondo sono: veri o falsi (con una certa probabilità) - Ogni possibile situazione in cui si trova il nostro agente è un mondo μ ; Esempio: nel caso del gioco del Lotto, per la singola estrazione ci possono essere 90 mondi, uno per ogni numero che può essere estratto. - Ogni mondo μ è un insieme di fatti: - fatti veri (V) - fatti falsi (F) - fatti incerti (I)

Tale teoria può essere formulata in diversi modi a seconda del tipo di assunzioni iniziali che si utilizzano. In questo testo si utilizza la teoria della probabilità basata sui cosiddetti assiomi di Kolmogorov e per questo detta **Teoria Assiomatica della Probabilità**.

Gli assiomi di Kolmogorov costituiscono la base matematica della teoria delle probabilità, formulata dal matematico russo Andrey Kolmogorov nel suo lavoro “Grundbegriffe der Wahrscheinlichkeitsrechnung” nel 1933.

Ecco una descrizione dei tre assiomi di Kolmogorov:

- **Primo Assioma (Non-negatività):** La probabilità di un evento è sempre un numero reale non negativo: $P(A) \geq 0$ per ogni evento A. Per rappresentare la probabilità di un certo mondo si usa il simbolo $P(\mu)$, $0 \leq P(\mu) \leq 1$
 - $P(\mu) = 0$ significa che il mondo μ non ha nessuna possibilità di verificarsi. Ad esempio la probabilità che al lotto venga estratto il numero 0 (zero)
 - $P(\mu) = 1$ significa che il mondo μ è certo. Ad esempio la probabilità che il risultato di una estrazione sia minore o uguale a 90 è 1 Più è «grande» $P(\mu)$ è più è verosimile che si verifichi il mondo μ .
- **Secondo Assioma (Normalizzazione) :** La somma delle probabilità di tutti gli eventi possibili nello spazio campione è uguale a 1: $P(S) = 1$, dove S rappresenta lo spazio campione. Ad esempio, la somma delle probabilità di estrazione di tutti i numeri del lotto è pari a 1
- **Terzo Assioma (Additività) :** Se A_1, A_2, A_3, \dots sono eventi mutuamente esclusivi (cioè non possono accadere simultaneamente), allora la probabilità dell'unione di questi eventi è uguale alla somma delle loro probabilità individuali: $P(A_1 \cup A_2 \cup A_3 \cup \dots) = P(A_1) + P(A_2) + P(A_3) + \dots$ Ad esempio, la probabilità di estrarre un numero pari al lotto è pari alla somma delle probabilità di estrarre i numeri pari da 2 a 90.

Gli assiomi di Kolmogorov forniscono un fondamento rigoroso per definire le probabilità e garantiscono che le probabilità siano consistenti e soddisfino le proprietà chiave della teoria delle probabilità. Questi assiomi sono essenziali per lo studio formale della probabilità e vengono utilizzati per sviluppare e applicare concetti probabilistici in varie discipline, inclusi statistica, teoria dei giochi, intelligenza artificiale e molti altri campi scientifici.

2.2.2 Calcolo della probabilità incondizionata o a priori

calcolo della probabilità di estrazione di un numero x al lotto Usando i tre assiomi di Kolmogorov :

si può calcolare la probabilità di estrazione di un numero x al lotto. - Dal primo assioma si ha che $P(x) \geq 0$. - Dal secondo assioma si ha che la somma di tutti i $P(x)$, con x che va da 1 a 90, è pari a 1. - Dal terzo si evince che essendo le probabilità di estrazione di un numero x uguale a quella di estrarre un numero y, con x diverso da y, si ha che la probabilità di estrarre un numero x è pari a $1/90$.

calcolo della probabilità del risultato x nel lancio di un dado Nel lancio di un dado a 6 facce:

la probabilità $P(n)$ di ottenere il numero n è $P(n) = 1/6$ perché all'esito del lancio tutte le facce del dado hanno uguale probabilità.

calcolo della probabilità del risultato nel lancio di due dadi : Nell'esito del lancio di due dadi, dobbiamo considerare che i mondi possibili ed equiprobabili sono $6 \times 6 = 36$ e quindi la probabilità di uno di questi mondi è $1/36$.

calcolo della probabilità del risultato x come somma dei valori nel lancio di due dadi : Nell'esito del lancio di due dadi, se vogliamo calcolare la probabilità che esca un certo valore x come somma dei valori dei due dadi dobbiamo considerare che i valori possibili di x $[2, 12]$ non sono equiprobabili. Infatti, per esempio, la probabilità di ottenere 2 è $1/36$, mentre la probabilità di ottenere 7 è $6/36$. Per calcolare la probabilità del valore x è sufficiente contare quanti sono i mondi in cui il valore x si ottiene come somma dei valori dei due dadi e poi dividere per il numero totale di mondi possibili. I possibili risultati del lancio di due dadi sono 36 :

Table 2.6: Lancio di due dadi

+	1	2	3	4	5	6
1	2	3	4	5	6	7
2	3	4	5	6	7	8
3	4	5	6	7	8	9
4	5	6	7	8	9	10
5	6	7	8	9	10	11
6	7	8	9	10	11	12

La probabilità di ottenere 2 è data dal numero di esiti favorevoli al risultato 2, in questo caso è solo uno, diviso il numero totale di esiti possibili, in questo caso 36. La possibilità di ottenere 3 è data dal numero di esiti favorevoli al risultato 3, in questo caso sono 2, diviso il numero totale di esiti possibili, in questo caso 36. ...

$P(2)=1/36$, $P(3)=2/36$, $P(4)=3/36$, $P(5)=4/36$, $P(6)=5/36$, $P(7)=6/36$, $P(8)=5/36$,
 $P(9)=4/36$, $P(10)=3/36$, $P(11)=2/36$, $P(12)=1/36$

2.2.3 Variabili aleatorie

Una variabile aleatoria nel calcolo delle probabilità è una variabile che può assumere uno dei possibili valori in un certo dominio: - La variabile lancio nel lancio di un dado può assumere uno dei valori nel dominio $\{1,2,3,4,5,6\}$ - La variabile sentenza nel processo penale può assumere uno dei valori nel dominio $\{\text{«Non luogo a procedere»}, \text{«Proscioglimento»}, \text{«Condanna»}\}$ - La variabile diagnosi in campo medico può assumere uno dei valori nel dominio $\{\text{«Malattia»}, \text{«Non malattia»}\}$.

Nell'inferenza probabilistica si è interessati alla probabilità che una certa variabile aleatoria assuma un certo valore. Ad esempio, in un determinato processo penale si potrebbe avere:

- $P(\text{sentenza} = \text{«Non luogo a procedere»}) = 0,1$
- $P(\text{sentenza} = \text{«Proscioglimento»}) = 0,1$
- $P(\text{sentenza} = \text{«Condanna»}) = 0,8$

Per codificare la variabile aleatoria “sentenza” in Python, si può utilizzare ad esempio la struttura dati dizionario che mappa i possibili esiti (“Non luogo a procedere”, “Proscioglimento”, “Condanna”) ai rispettivi valori numerici di probabilità. Ecco un esempio di come si potrebbe codificare la variabile aleatoria “sentenza” in Python utilizzando un dizionario:

```
# Definizione della variabile aleatoria sentenza con i suoi possibili valori
sentenza = {
    "Non luogo a procedere": 0.128, # probabilità del 12.8% di non luogo a procedere
    "Proscioglimento": 0.548,      # probabilità del 54.8% di proscioglimento
    "Condanna": 0.324              # probabilità del 32.4% di condanna
}
somma = 0
for esito, probabilita in sentenza.items():
    print(f"La probabilità di '{esito}' è: {probabilita}")
    somma = somma + probabilita
print(" la somma delle probabilità è pari a ", somma)
```

La probabilità di 'Non luogo a procedere' è: 0.128

La probabilità di 'Proscioglimento' è: 0.548

La probabilità di 'Condanna' è: 0.324

la somma delle probabilità è pari a 1.0

2.2.4 Distribuzioni di probabilità

le distribuzioni di probabilità sono funzioni che descrivono la probabilità di ogni possibile valore di una variabile aleatoria. Ad esempio, la distribuzione di probabilità della variabile aleatoria “sentenza” nel processo penale può essere rappresentata come segue: $P(\text{sentenza}) = \{0.1, 0.1, 0.8\}$. Nel seguito vedremo alcune distribuzioni di probabilità notevoli.

2.2.5 Probabilità congiunta

La probabilità congiunta è la probabilità che due eventi si verifichino contemporaneamente. Ad esempio, la probabilità che un processo penale porti a una condanna e che il condannato sia colpevole è data dalla probabilità congiunta di questi due eventi. Oppure, in ambito medico, la probabilità che un paziente abbia una certa patologia e che il test diagnostico sia positivo è data dalla probabilità congiunta di questi due eventi. Oppure, in ambito meteorologico, la probabilità che sia nuvoloso e che piova è la probabilità congiunta di questi due eventi:

probabilità che sia nuvoloso:

	nuvoloso	\neg nuvoloso
P(n)	0,7	0,3

probabilità che piova:

	piove	\neg piove
P(p)	0,2	0,8

probabilità che sia nuvoloso e piova:

P(p,n)	nuvoloso	\neg nuvoloso
piove	0,55	0,05
\neg piove	0,15	0,25

2.2.6 Indipendenza delle variabili aleatorie

L'indipendenza di due eventi indica che il verificarsi di uno non influenza il verificarsi dell'altro. Ad esempio: Lancio di due dadi. Il lancio del primo non influenza il secondo; Il contrario, la dipendenza, indica che il verificarsi di uno influenza il verificarsi dell'altro. Nel caso in cui due variabili aleatorie siano indipendenti si ha la seguente proprietà:

$$P(a \cap b) = P(a) \cdot P(b)$$

2.2.7 Negazione


La negazione di un evento è l'evento che si verifica quando l'evento originale non si verifica. Ad esempio, la negazione dell'evento "piove" è "non piove".

Se la probabilità che un evento è p , la probabilità che l'evento non si verifichi è $1 - p$.

$P(A) = p$, allora $P(\neg A) = 1 - p$.

2.2.8 Inclusione

$$P(a \cup b) = P(a) + P(b) - P(a \cap b).$$

La probabilità che si verifichi l'evento a o l'evento b è uguale alla somma delle probabilità dei due eventi meno la probabilità congiunta. Si noti che se gli eventi sono incompatibili la probabilità congiunta è nulla! 

2.2.9 Marginalizzazione

La marginalizzazione è una tecnica utilizzata per calcolare la probabilità di un evento dato un insieme di eventi. Ad esempio, la probabilità che un processo penale porti a una condanna dato che il condannato è colpevole è data dalla marginalizzazione della probabilità congiunta di questi due eventi.

$$P(a) = P(a, b) + P(a, \neg b).$$

La probabilità che si verifichi b è disgiunta dalla probabilità che si verifichi $\neg b$. Quindi, quando si verifica a si ha b oppure $\neg b$ ma non entrambi quindi se sommo le probabilità $P(a, b) + P(a, \neg b)$ ottengo $P(a)$.

2.2.10 probabilità condizionata

La probabilità condizionata è la probabilità che un evento si verifichi dato che un altro evento si è verificato. Ad esempio, la probabilità che un processo penale porti a una condanna dato che il sottoposto a giudizio è colpevole è data dalla probabilità condizionata di questi due eventi.

E' possibile fare inferenze a proposito della probabilità di una proposizione ignota A , data la prova B , calcolando $P(A/B)$ (probabilità di A dato che tutto ciò che sappiamo è B) (inferenza probabilistica)

Un'interrogazione ad un sistema di ragionamento probabilistico chiederà di calcolare il valore di una particolare probabilità condizionata.

Fin qui abbiamo visto casi in cui il singolo evento non era condizionato da altro evento:

- Prima estrazione del lotto;
- Lancio di uno o due dadi

Cosa succede alla probabilità quando l'avverarsi di una proposizione è condizionata all'avverarsi di un'altra proposizione?

$P(a|b)$ = probabilità dell'evento a dato che noi sappiamo che l'evento b si è verificato. Oppure, “ la probabilità di a dato b” Possiamo chiederci:

- Qual'è la probabilità che vinca la “ Roma” se ha vinto la “ Lazio”?, $P(\text{Roma}/\text{Lazio})$.
- Qual'è la probabilità che arrivi il “38” se è arrivato il “52”?, $P(\text{“38”}/\text{“52”})$.

La formula per calcolare la probabilità condizionata di a dato b è la seguente:

$$P(a/b) = (P(a \wedge b))/(P(b));$$

“siamo interessati agli eventi dove a e b sono vere, ma solo nei mondi dove b è vera!”

$$P(a \wedge b) = P(b)P(a/b)$$

$$P(a \wedge b) = P(b)P(a/b)$$

2.2.10.1 calcolo della probabilità condizionata: lancio di due dadi

Qual è la probabilità che si ottenga una somma pari 9 lanciando due dadi se il primo dado è 6, $P(9/6)$? La risposta si ottiene direttamente dalla formula della probabilità condizionata e da quella della probabilità congiunta: $p(9/6) = P(9 \text{ e } 6) / P(6) = 1/36 / 1/6 = 1/6$

La proposizione a = «somma=9» si verifica con i seguenti lanci:

$$a = \{(6,3),(5,4),(4,5),(3,6)\} \rightarrow P(a) = 4/36$$

La proposizione b = «primo dado=6» si verifica con i seguenti lanci:

$$b = \{(6,1),(6,2),(6,3),(6,4),(6,5),(6,6)\} \rightarrow P(b) = 6/36$$

$$a \text{ “ ” } b = \{(6,3)\} \quad P(a \text{ “ ” } b) = 1/36$$

$$P(a/b) = (P(a \text{ “ ” } b) / P(b)) = (1/36) / (6/36) = 1/6$$

Table 2.10: Lancio di due dadi

+	1	2	3	4	5	6
1	2	3	4	5	6	7
2	3	4	5	6	7	8
3	4	5	6	7	8	9
4	5	6	7	8	9	10

+	1	2	3	4	5	6
5	6	7	8	9	10	11
6	7	8	9	10	11	12

Ovvero, per risolvere l'esercizio dobbiamo osservare l'ultima riga della tabella relativa al lancio del primo dado con risultato 6. i casi favorevoli sono solo 1, mentre i casi possibili sono 6. Quindi la probabilità è $1/6$.

2.2.10.2 calcolo della probabilità condizionata: caso penale

Supponiamo di avere un dataset di 1.000 casi penali. Per ogni caso, raccogliamo le seguenti informazioni:

- **Alibi** del sospettato (Sì/No)
- **Testimone oculare** presente (Sì/No)
- **Condanna** del sospettato (Sì/No)

Alibi	Testimone Oculare	Condanna	Numero di Casi
Sì	Sì	Sì	50
Sì	Sì	No	20
Sì	No	Sì	30
Sì	No	No	100
No	Sì	Sì	200
No	Sì	No	50
No	No	Sì	150
No	No	No	400
totale	-	-	1.000

Questo dataset può essere utilizzato per calcolare varie probabilità condizionate.

Esercizio 2.1: Probabilità di condanna dato che il sospettato non ha un alibi e c'è un testimone oculare.

- **Formula:** $P(\text{Condanna} = \text{Sì} \mid \text{Alibi} = \text{No}, \text{Testimone} = \text{Sì})$
- **Calcolo:**
 - Numero di casi con Alibi = No, Testimone = Sì, Condanna = Sì: 200
 - Numero totale di casi con Alibi = No, Testimone = Sì: $200 + 50 = 250$
 - $P = \frac{200}{250} = 0,8(80\%)$

Esercizio 2.2: Probabilità che ci sia un testimone oculare dato che il sospettato è stato condannato.

- **Formula:** $P(\text{Testimone} = \text{Sì} \mid \text{Condanna} = \text{Sì})$
- **Calcolo:**
 - Numero di casi con Testimone = Sì, Condanna = Sì: $50 + 200 = 250$
 - Numero totale di casi con Condanna = Sì: $50 + 30 + 200 + 150 = 430$
 - $P = \frac{250}{430} \approx 0,581(58,1\%)$

Esercizio 2.3: Probabilità che un sospettato non abbia un alibi dato che non è stato condannato.

- **Formula:** $P(\text{Alibi} = \text{No} \mid \text{Condanna} = \text{No})$
- **Calcolo:**
 - Numero di casi con Alibi = No, Condanna = No: $50 + 400 = 450$
 - Numero totale di casi con Condanna = No: $20 + 100 + 50 + 400 = 570$
 - $P = \frac{450}{570} \approx 0,789(78,9\%)$

Questo dataset consente di esplorare come diverse variabili influenzino le probabilità di determinati esiti nel contesto del diritto penale. Può essere utilizzato per analisi statistiche, studi accademici o simulazioni di casi giudiziari.

2.2.11 Condizionamento

$$P(a) = P(a/b)P(b) + P(a/\neg b)P(\neg b).$$

Il condizionamento discende immediatamente dalla marginalizzazione. La probabilità che si verifichi a è data dalla marginalizzazione della probabilità congiunta di questi due eventi. La probabilità che si verifichi b è disgiunta dalla probabilità che si verifichi $\neg b$. Quindi, quando si verifica a si ha b oppure $\neg b$ ma non entrambi quindi se sommo le probabilità $P(a, b) + P(a, \neg b)$ ottengo $P(a)$.

Vediamo due esempi di applicazioni della formula di condizionamento in Python, uno nel campo medico e uno nel campo giuridico penale:

Campo medico Supponiamo di avere le seguenti informazioni:

La probabilità che una persona sviluppi un certo effetto collaterale a seguito di un farmaco è $P(\text{Effettocollaterale}) = 0.2$.

Si sa che se una persona sviluppa l'effetto collaterale, la probabilità che abbia assunto il farmaco è $P(\text{Farmaco} \mid \text{Effettocollaterale}) = 0.9$. D'altra parte, se una persona non mostra l'effetto collaterale, la probabilità che abbia comunque assunto il farmaco è $P(\text{Farmaco} \mid \neg \text{Effettocollaterale}) = 0.1$. In questo caso, il condizionamento riguarda la

probabilità di assunzione del farmaco date le informazioni sull'effetto collaterale senza coinvolgere il teorema di Bayes.

Quindi, la probabilità di assunzione del farmaco è

$$P(\text{Farmaco}) = P(\text{Farmaco}|\text{Effetto collaterale})P(\text{Effetto collaterale}) + \\ P(\text{Farmaco}|\neg\text{Effetto collaterale})P(\neg\text{Effetto collaterale}).$$

la codifica in Python è la seguente:

```
#definizioni delle probaibilità
P_effetto_collaterale = 0.2
P_farmaco_effetto_collaterale = 0.9
P_farmaco_non_effetto_collaterale = 0.1
P_farmaco = P_farmaco_effetto_collaterale * P_effetto_collaterale + \
            P_farmaco_non_effetto_collaterale * (1 - P_effetto_collaterale)
print(f"La probabilità che una pesona abbia assunto il farmaco è : {P_farmaco}")
```

Output atteso :

```
La probabilità che una pesona abbia assunto il farmaco è : 0.26
```

Campo Giuridico Penale :

Immaginiamo di avere le seguenti informazioni in un contesto giuridico penale: $P(\text{Condanna con prove schiaccianti}) = 0.95$

$$P(\text{Condanna senza prove schiaccianti}) = 0.2$$

$$P(\text{Prove schiaccianti}) = 0.3$$

Utilizziamo la formula di condizionamento per calcolare la probabilità di condanna:

$$P(\text{Condanna}) = P(\text{Condanna con prove schiaccianti})P(\text{Prove schiaccianti}) + \\ P(\text{Condanna senza prove schiaccianti})(1 - P(\text{Prove schiaccianti}))$$

La codifica in Python è la seguente:

```
prob_condanna_prove_schiaccianti = 0.95
prob_condanna_no_prove_schiaccianti = 0.2
prob_prove_schiaccianti = 0.3
prob_condanna = prob_condanna_prove_schiaccianti * prob_prove_schiaccianti + \
                prob_condanna_no_prove_schiaccianti * (1 - prob_prove_schiaccianti)
print(f"La probabilità che un imputato sia colpevole dato che ci sono prove schiaccianti è: ")
```

Output atteso :

```
La probabilità che un imputato sia colpevole  
dato che ci sono prove schiaccianti è: 0.42499999999999993
```

Questi due esempi illustrano come la formula di condizionamento possa essere applicata in contesti medici e giuridici penali per calcolare probabilità condizionate basate su informazioni specifiche relative agli eventi considerati.

2.3 Inferenza Bayesiana

“Mr. Bayes ... design ... was to find out a method by which we might judge concerning the probability that an event has to happen, in given circumstances, upon supposition that we know nothing concerning it but that, under the same circumstances, it has happened a certain number of times, and failed a certain other number of times.” - (Richard Price, presentando lo scritto dell'amico Thomas Bayes alla Royal Society of London)

Il Teorema di Bayes, formalizzato dal reverendo Thomas Bayes nel XVIII secolo, è uno strumento fondamentale nell'ambito della statistica e dell'intelligenza artificiale che permette di aggiornare le nostre credenze riguardo ad un'ipotesi sulla base di nuove evidenze. Le tecniche di inferenza basate su questo teorema sono ampiamente utilizzate in diversi campi, dall'analisi dei dati alla diagnostica medica, dalla finanza alla progettazione di algoritmi di machine learning.

2.3.1 Il Teorema di Bayes

Il Teorema di Bayes fornisce un modo per calcolare la probabilità condizionata di un'ipotesi data l'evidenza osservata. Formalmente, il teorema può essere espresso come:

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$$

Dove: - $P(A|B)$ è la probabilità dell'ipotesi A dato l'evidenza B. - $P(B|A)$ è la probabilità dell'evidenza B dato l'ipotesi A. - $P(A)$ è la probabilità a priori dell'ipotesi A. - $P(B)$ è la probabilità dell'evidenza B.

2.3.2 Applicazioni Pratiche

Diagnostica Medica

Nel campo della diagnostica medica, il Teorema di Bayes è utilizzato per valutare la probabilità che un paziente abbia una certa malattia sulla base dei sintomi presentati e dei risultati dei test di laboratorio. Ad esempio, se la probabilità di un test positivo dato che il paziente ha la malattia e la probabilità che il paziente abbia effettivamente la malattia sono note, il teorema di Bayes può essere impiegato per calcolare la probabilità che il paziente abbia la malattia date le informazioni disponibili.

Finanza

Nel settore finanziario, il Teorema di Bayes viene adoperato per valutare il rischio e formulare previsioni basate su dati storici e informazioni di mercato. Ad esempio, il teorema può essere utilizzato per stimare la probabilità di un evento futuro, come un aumento dei tassi di interesse, sulla base di indicatori economici attuali.

Machine Learning

Nei modelli di machine learning, come le reti bayesiane, il Teorema di Bayes svolge un ruolo chiave nell'aggiornare le probabilità delle variabili all'interno del modello in risposta ai nuovi dati. Questo processo di apprendimento bayesiano consente ai modelli di essere più flessibili ed adattabili all'evoluzione dei dati nel tempo.

diritto penale

Dalle statistiche (false perché inventate da me :) di un certo tribunale o dal Ministero della Giustizia abbiamo che - 80% degli imputati condannati hanno precedenti penali $P(\text{precedenti}/\text{condanna}) = 0,8$; - 10% degli imputati sono condannati $P(\text{condanna}) = 0,1$ - 20% degli imputati hanno precedenti penali $P(\text{precedenti}) = 0,2$

Applicando il teorema di Bayes abbiamo che la probabilità che un imputato con precedenti sia condannato è

$$P(\text{condanna}/\text{precedenti}) = P(\text{condanna}) \cdot \frac{P(\text{precedenti}/\text{condanna})}{P(\text{precedenti})} = 0,1 \cdot \frac{0,8}{0,2} = 0,4$$

Si osservi che la probabilità di essere condannati era del 10%. Applicando il teorema di Bayes abbiamo scoperto che la probabilità di essere condannato è del 40% se sappiamo che la persona sottoposta a giudizio ha dei precedenti penali. Ovvero, la probabilità iniziale di essere condannati senza sapere se sono presenti o meno precedenti penali viene moltiplicata per 4 (il cosiddetto fattore di Bayes).

Le tecniche di inferenza basate sul Teorema di Bayes forniscono un approccio potente per il ragionamento probabilistico e l'aggiornamento delle credenze in base alle evidenze disponibili.

Utilizzate in una vasta gamma di settori, queste tecniche consentono di prendere decisioni informate e di sfruttare al meglio le informazioni a disposizione. La comprensione e l'applicazione corretta del Teorema di Bayes sono cruciali per ottenere risultati accurati e significativi nelle analisi statistiche e nel machine learning.

2.3.2.1 Applicazione del Teorema di Bayes: caso penale

Applicazione del teorema di Bayes al data set del caso pratico Section [2.2.10.2](#) il Teorema di Bayes per calcolare la probabilità che un sospettato venga **condannato dato che non ha un alibi**, cioè $P(\text{Condanna} = \text{Sì} \mid \text{Alibi} = \text{No})$. Questo ci permette di comprendere meglio l'impatto dell'assenza di un alibi sulla probabilità di condanna.

Obiettivo: Calcolare $P(\text{Condanna} = \text{Sì} \mid \text{Alibi} = \text{No})$.

Teorema di Bayes:

$$P(\text{Condanna} = \text{Sì} \mid \text{Alibi} = \text{No}) = \frac{P(\text{Alibi} = \text{No} \mid \text{Condanna} = \text{Sì}) \times P(\text{Condanna} = \text{Sì})}{P(\text{Alibi} = \text{No})}$$

Passo 1: Calcolare $P(\text{Alibi} = \text{No} \mid \text{Condanna} = \text{Sì})$

- **Numero totale di casi con Condanna = Sì:**
 $50 + 30 + 200 + 150 = 430$ casi.
- **Numero di casi con Alibi = No e Condanna = Sì:**
 $200 + 150 = 350$ casi.
- **Calcolo:**

$$P(\text{Alibi} = \text{No} \mid \text{Condanna} = \text{Sì}) = \frac{350}{430} \approx 0,8139 \text{ (81,39\%)}$$

Passo 2: Calcolare $P(\text{Condanna} = \text{Sì})$

- **Numero totale di casi con Condanna = Sì:** 430 (come sopra).
- **Numero totale di casi:** 1.000 .
- **Calcolo:**

$$P(\text{Condanna} = \text{Sì}) = \frac{430}{1.000} = 0,43 \text{ (43\%)}$$

Passo 3: Calcolare $P(\text{Alibi} = \text{No})$

- **Numero totale di casi con Alibi = No:**
200 + 50 + 150 + 400 = 800 casi.
- **Numero totale di casi:** 1.000.
- **Calcolo:**

$$P(\text{Alibi} = \text{No}) = \frac{800}{1.000} = 0,8 \text{ (80\%)}$$

Passo 4: Applicare il Teorema di Bayes

$$P(\text{Condanna} = \text{Sì} \mid \text{Alibi} = \text{No}) = \frac{0,8139 \times 0,43}{0,8} = \frac{0,349977}{0,8} = 0,4375 \text{ (43,75\%)}$$

Risultato:

La probabilità che un sospettato venga **condannato dato che non ha un alibi** è circa **43,75%**.

Verifica Diretta dai Dati del Dataset

Per confermare il risultato, possiamo calcolare direttamente $P(\text{Condanna} = \text{Sì} \mid \text{Alibi} = \text{No})$:

- **Numero di casi con Alibi = No:** 800 (come calcolato sopra).
- **Numero di casi con Alibi = No e Condanna = Sì:** 200 + 150 = 350.
- **Calcolo diretto:**

$$P(\text{Condanna} = \text{Sì} \mid \text{Alibi} = \text{No}) = \frac{350}{800} = 0,4375 \text{ (43,75\%)}$$

Il risultato conferma il calcolo effettuato tramite il Teorema di Bayes. Questo risultato indica che, secondo i dati del dataset:

- **Se un sospettato non ha un alibi**, ha una probabilità del **43,75%** di essere condannato.
- L'assenza di un alibi non aumenta significativamente la probabilità di condanna rispetto alla probabilità generale di condanna nel dataset, che è del **43%**.
- **Importanza dell'Alibi:** In questo dataset, l'alibi non sembra essere un fattore importante nel determinare l'esito di un caso. Avere un alibi può ridurre la probabilità di condanna.

- **Utilizzo del Teorema di Bayes:** Questo esempio illustra come il Teorema di Bayes possa essere utilizzato per aggiornare le probabilità in base a informazioni nuove o specifiche, nel contesto del diritto penale.

2.3.3 reti di Bayes

Una **rete bayesiana** (BN, Bayesian network) è un modello grafico probabilistico che rappresenta un insieme di variabili stocastiche con le loro dipendenze condizionali attraverso l'uso di un **grafo aciclico diretto (DAG)**. Per esempio una rete Bayesiana potrebbe rappresentare la relazione probabilistica esistente tra i sintomi e le malattie. Dati i sintomi, la rete può essere usata per calcolare la probabilità della presenza di diverse malattie. Le reti Bayesiane sono Grafi diretti. Ogni nodo rappresenta una variabile aleatoria e ogni freccia da X a Y indica che X è un genitore di Y. Ovvero, indica che la distribuzione probabilistica di Y dipende da X. Ogni nodo ha la distribuzione probabilistica $P(X \mid \text{Genitori}(X))$. Vediamo un esempio sui mezzi di trasporto (:

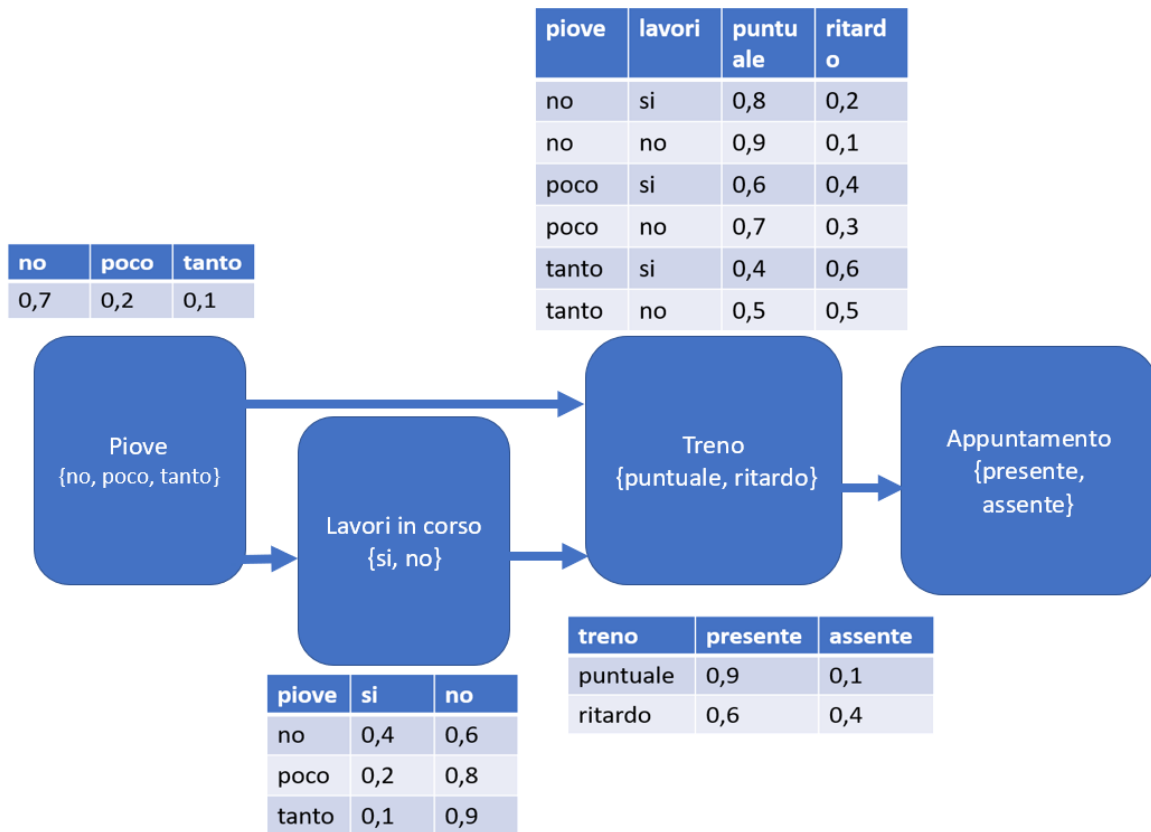


Figure 2.2: Rete di Bayes puntualità treno

Una rete bayesiana (di credenza) richiede che ogni nodo del grafo sia condizionatamente indipendente da qualsiasi sottoinsieme di nodi che non siano discendenti dei predecessori diretti del nodo stesso. Ci si affida ad un esperto di dominio per la definizione della topologia della rete di credenze (quali nodi e quali relazioni condizionali di dipendenza), poi si calcolano le influenze dirette e le conseguenti probabilità. Ciò equivale a definire la conoscenza del mondo in cui può avvenire un evento. Ovvero, la rete rappresenta le assunzioni che si possono fare su quel dominio. Le probabilità condizionate tra i nodi riassumono un insieme potenzialmente infinito di circostanze a noi ignote e che potrebbero influenzare l'evento. La topologia della rete è la base di conoscenza generale ed astratta dell'ambiente in cui si possono verificare gli eventi e rappresenta la struttura generale del processo causale nel dominio, piuttosto che fornire dettagli su un particolare elemento. Nelle reti bayesiane gli archi che connettono i nodi esprimono le relazioni causali dirette (causa \rightarrow effetto). Una volta definita la topologia bisogna specificare la tabella delle probabilità condizionate associata ad ogni nodo. Ogni riga della tabella esprime la probabilità del valore di ogni nodo per un caso condizionante (combinazione di valori dei nodi genitori produttoria delle prob. condiz.) Un nodo con nessun genitore è rappresentato dalla probabilità a priori.

2.3.3.1 Esempio di rete bayesiana: indagine criminale

Scriviamo il codice Python necessario per creare un modello di Rete Bayesiana per analizzare la probabilità di colpevolezza di un sospetto in un'indagine criminale. Il modello considera tre elementi di prova: la presenza di un'arma (Arma), un movente (Motive) e un alibi (Alibi), e come questi influenzano la probabilità di colpevolezza (Colpevolezza).

Il codice non richiederà input diretti dall'utente. Invece, definisce la struttura della Rete Bayesiana e imposta le tabelle di probabilità per ciascun fattore basate su valori predefiniti che dovrebbero essere estrapolati da statistiche sulle indagini criminali.

Descrizione del codice

L'output di questo codice è: - un modello di Rete Bayesiana verificato; - la stampa del modello; - la stampa delle Distribuzioni di Probabilità Condizionata (CPD) per ogni variabile nella rete; - il grafo della rete Bayesiana.

Inizialmente, definiamo la struttura della Rete Bayesiana, mostrando come i fattori di prova (Arma, Motive, Alibi) influenzano la colpevolezza (Colpevolezza). Quindi, definiamo le tabelle di probabilità per ciascun fattore. Ad esempio, la probabilità che un'arma sia presente sia presente sul luogo del delitto la poniamo pari al 70% (0.7) e la sua assenza al 30% (0.3). Più complessa è la definizione della tabella di probabilità per la colpevolezza, che considera tutte le possibili combinazioni dei fattori di prova. Tutte queste tabelle sono aggiunte al modello di Rete Bayesiana. Infine, verifichiamo se il modello è definito correttamente e stampiamo tutte le distribuzioni di probabilità.

La logica chiave in questo codice è come esso rappresenta le relazioni tra diversi elementi di prova e la colpevolezza. Ad esempio, la presenza di un'arma, un movente e la mancanza di un alibi aumenterebbero la probabilità di colpevolezza, mentre la loro assenza la diminuirebbe. Questo è riflesso nella tabella di probabilità per 'Colpevolezza', che considera tutte le possibili combinazioni di prove:

Arma	Motive	Alibi	P(Non Colpevole)	P(Colpevole)
0	0	0	0.1	0.9
0	0	1	0.2	0.8
0	1	0	0.3	0.7
0	1	1	0.4	0.6
1	0	0	0.5	0.5
1	0	1	0.6	0.4
1	1	0	0.7	0.3
1	1	1	0.8	0.2

In questa tabella:

0 rappresenta l'assenza (di arma, movente o alibi) 1 rappresenta la presenza Le ultime due colonne mostrano le probabilità di non colpevolezza e colpevolezza per ogni combinazione di evidenze

Questa Rete Bayesiana può essere utilizzata per calcolare la probabilità di colpevolezza dato uno scenario di prove, aiutando gli investigatori a quantificare e ragionare sull'incertezza nei casi criminali.

```
from pgmpy.models import BayesianNetwork
from pgmpy.factors.discrete import TabularCPD

# Definizione del modello
model = BayesianNetwork([('Arma', 'Colpevolezza'),
                        ('Motive', 'Colpevolezza'),
                        ('Alibi', 'Colpevolezza')])

# Definizione delle probabilità condizionate
cpd_arma = TabularCPD(variable='Arma', variable_card=2,
                      values=[[0.7], [0.3]])
cpd_Movente = TabularCPD(variable='Motive', variable_card=2,
                          values=[[0.6], [0.4]])
cpd_alibi = TabularCPD(variable='Alibi', variable_card=2,
                       values=[[0.5], [0.5]])
cpd_colpevolezza = TabularCPD(variable='Colpevolezza', variable_card=2,
```



```

        values=[[0.9, 0.7, 0.6, 0.4, 0.6, 0.4, 0.3, 0.1],
                [0.1, 0.3, 0.4, 0.6, 0.4, 0.6, 0.7, 0.9]],
        evidence=['Arma', 'Movente', 'Alibi'],
        evidence_card=[2, 2, 2])

# cpd_colpevolezza = TabularCPD(variable='Colpevolezza', variable_card=2,
#                               values=[[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8],
#                                       [0.9, 0.8, 0.7, 0.6, 0.5, 0.4, 0.3, 0.2]],
#                               evidence=['Arma', 'Movente', 'Alibi'],
#                               evidence_card=[2, 2, 2])

# Aggiunta delle probabilità condizionate al modello
model.add_cpds(cpd_arma, cpd_movente, cpd_alibi, cpd_colpevolezza)

# Verifica del modello
print("Il modello è corretto: ", model.check_model())

# Stampa del modello
for cpd in model.get_cpds():
    print("CPD di {variable}:".format(variable=cpd.variable))
    print(cpd)

```

Il modello è corretto: True

CPD di Arma:

```

+-----+-----+
| Arma(0) | 0.7 |
+-----+-----+
| Arma(1) | 0.3 |
+-----+-----+

```

CPD di Movente:

```

+-----+-----+
| Movente(0) | 0.6 |
+-----+-----+
| Movente(1) | 0.4 |
+-----+-----+

```

CPD di Alibi:

```

+-----+-----+
| Alibi(0) | 0.5 |
+-----+-----+
| Alibi(1) | 0.5 |
+-----+-----+

```

CPD di Colpevolezza:

```

+-----+-----+-----+-----+-----+

```

Arma	Arma(0)	...	Arma(1)	Arma(1)	
+-----+	+-----+	+-----+	+-----+	+-----+	+
Movente	Movente(0)	...	Movente(1)	Movente(1)	
+-----+	+-----+	+-----+	+-----+	+-----+	+
Alibi	Alibi(0)	...	Alibi(0)	Alibi(1)	
+-----+	+-----+	+-----+	+-----+	+-----+	+
Colpevolezza(0)	0.9	...	0.3	0.1	
+-----+	+-----+	+-----+	+-----+	+-----+	+
Colpevolezza(1)	0.1	...	0.7	0.9	
+-----+	+-----+	+-----+	+-----+	+-----+	+

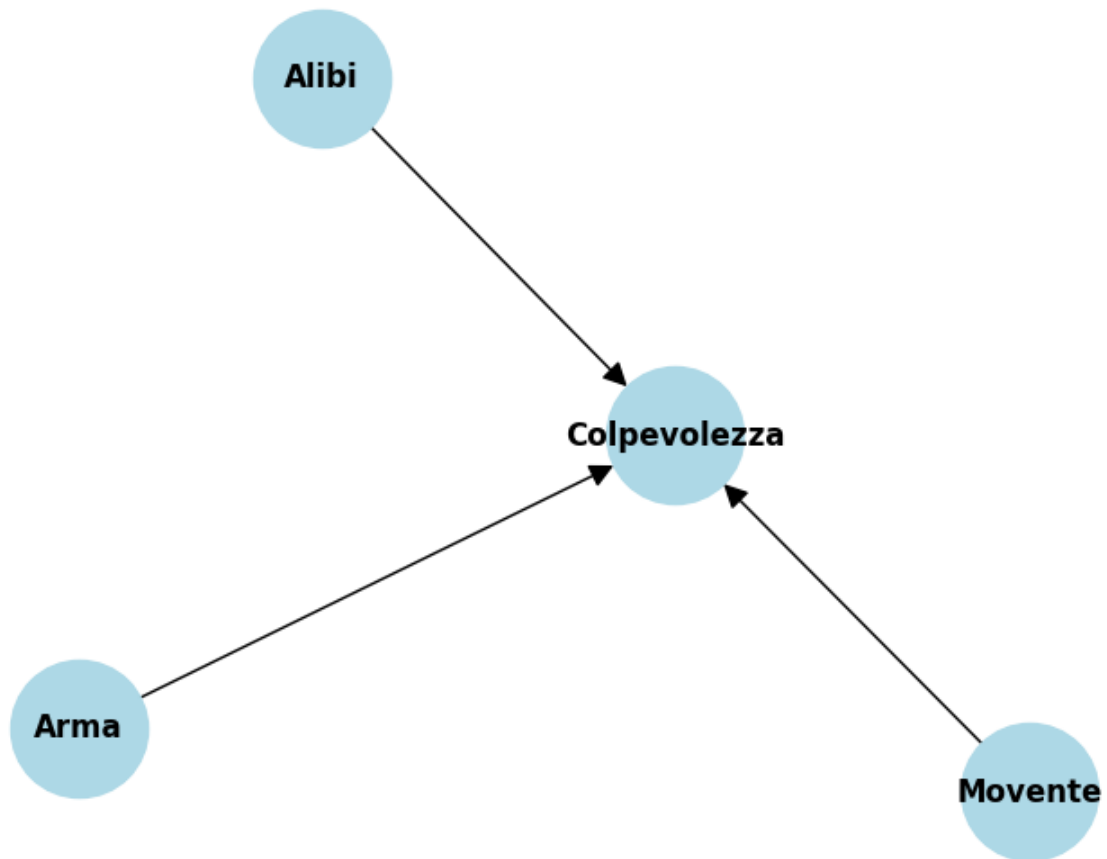
```
import networkx as nx
import matplotlib.pyplot as plt

# Assumendo che 'model' sia il tuo BayesianNetwork già definito
G = nx.DiGraph()
G.add_edges_from(model.edges())

pos = nx.spring_layout(G)
nx.draw(G, pos, with_labels=True, node_color='lightblue',
        node_size=3000, arrowsize=20, font_size=12, font_weight='bold')

plt.title("Rete di Bayes per l'Analisi della Colpevolezza")
plt.axis('off')
plt.show()
```

Rete di Bayes per l'Analisi della Colpevolezza



Infine, una volta costruita la rete di Bayes possiamo interrogarla per avere una stima della probabilità di un determinato evento. Qual è la probabilità che un indagato senza alibi, senza movente e in assenza di arma del delitto sia colpevole?

```
from pgmpy.inference import VariableElimination

# Creiamo un oggetto per l'inferenza
inference = VariableElimination(model)

# Definiamo l'evidenza per la situazione descritta
evidence = {
    'Alibi': 0, # 0 rappresenta l'assenza di alibi
    'Movente': 0, # 0 rappresenta l'assenza di motivo
    'Arma': 0 # 0 rappresenta che l'arma non è stata trovata
}
```

```
# Calcoliamo la probabilità di colpevolezza dato l'evidenza
result = inference.query(['Colpevolezza'], evidence=evidence)

# Stampiamo il risultato
print("Probabilità di colpevolezza:")
print(result.values)
```

Probabilità di colpevolezza:
[0.9 0.1]

```
from pgmpy.inference import VariableElimination

# Creiamo un oggetto per l'inferenza
inference = VariableElimination(model)

# Definiamo l'evidenza per la situazione descritta
evidence = {
    'Alibi': 1, # 0 rappresenta l'assenza di alibi
    'Movente': 0, # 0 rappresenta l'assenza di motivo
    'Arma': 0 # 0 rappresenta che l'arma non è stata trovata
}

# Calcoliamo la probabilità di colpevolezza dato l'evidenza
result = inference.query(['Colpevolezza'], evidence=evidence)

# Stampiamo il risultato
print("Probabilità di colpevolezza:")
print(result.values)
```

Probabilità di colpevolezza:
[0.7 0.3]

2.4 Algoritmi di Ricerca

Gli algoritmi di ricerca sono utilizzati per esplorare spazi di soluzione vasti e complessi. Nel contesto legale, possono essere utilizzati per trovare precedenti giuridici o per esplorare possibili risultati di un caso. In generale, i problemi di ricerca coinvolgono un agente a cui viene assegnato uno stato iniziale e uno stato obiettivo e restituisce una soluzione su come passare dal primo al secondo.

Gli algoritmi di ricerca sono usati in molte applicazioni della intelligenza artificiale, tra cui:

- **Problemi di pianificazione:** trovare una sequenza di azioni per raggiungere un obiettivo.
- **Risoluzione di puzzle e giochi:** come il cubo di Rubik, gli scacchi o il gioco del 15.
- **Navigazione e percorsi:** trovare il percorso migliore in mappe o reti stradali.
- **Ottimizzazione di processi:** trovare la configurazione ottimale in problemi complessi.
- **Scheduling:** organizzare attività o risorse in modo efficiente.
- **Riconoscimento di pattern:** identificare strutture o sequenze in dati complessi.
- **Diagnosi medica:** identificare possibili malattie basandosi su sintomi.
- **Elaborazione del linguaggio naturale:** analisi sintattica e semantica.
- **Visione artificiale:** riconoscimento di oggetti e scene in immagini.
- **Robotica:** pianificazione del movimento e navigazione autonoma.

Questi algoritmi sono versatili e possono essere adattati a molti altri domini, rendendo la ricerca un'area fondamentale dell'intelligenza artificiale.

2.4.1 Glossario della ricerca

- **agente:** Una entità che percepisce e agisce nel suo ambiente.
- **stato:** Una configurazione di un agente nel suo ambiente.
- **stato iniziale:** Lo stato iniziale di un agente.
- **stato finale:** Lo stato obiettivo di un agente.
- **azioni:** Le azioni che un agente può eseguire da un determinato stato
- **modello di transizione di stato:** Un modello che descrive come un agente può cambiare lo stato in seguito a un'azione.
- **spazio degli stati:** L'insieme di tutti gli stati raggiungibili da uno stato iniziale.
- **costo** di un cammino o percorso: La somma dei costi delle azioni lungo un percorso.
- **soluzione:** Un percorso che porta da uno stato iniziale a uno stato finale. Se ha un costo minimo, è una soluzione ottima.
- **algoritmo di ricerca:** Un algoritmo che cerca di trovare una soluzione.

2.4.2 Problemi di ricerca

I problemi che si possono affrontare con gli algoritmi di ricerca sono generalmente caratterizzati da una struttura specifica:

1. **Stato iniziale:** Questo rappresenta la condizione o la configurazione di partenza del problema.
2. **Azioni o operatori:** Questi rappresentano le possibili mosse o le transizioni che possono essere effettuate a partire da uno stato.
3. **Test obiettivo (goal test):** Questo è un criterio che determina se uno stato specifico risolve il problema.

4. **Funzione costo:** Questa associa un costo a ogni operatore o azione. Il costo può rappresentare, ad esempio, il tempo, lo sforzo o le risorse necessarie per eseguire un'azione.

A seconda della natura del problema, lo stato iniziale può essere un singolo stato o un insieme di stati. Inoltre, i problemi possono essere classificati in base alla conoscenza che l'agente ha sullo stato in cui si trova e sulle azioni.

Una volta definito il problema in questi termini, l'algoritmo di ricerca può essere utilizzato per esplorare lo spazio degli stati e trovare una soluzione, che è una sequenza di azioni che porta dallo stato iniziale a uno stato che soddisfa il test obiettivo¹.

esempio di problema di ricerca

Un esempio classico di problema che segue questa struttura: il problema del commesso viaggiatore ([Travelling Salesman Problem](#), TSP).

Dato un insieme di città, e note le distanze tra ciascuna coppia di esse, trovare il tragitto di minima percorrenza che un commesso viaggiatore deve seguire per visitare tutte le città una ed una sola volta e ritornare alla città di partenza

1. **Stato iniziale:** Il commesso viaggiatore si trova in una città specifica (ad esempio, Roma) e deve visitare tutte le altre città una sola volta e tornare alla città di partenza.
2. **Azioni o operatori:** Il commesso viaggiatore può scegliere di viaggiare da una città all'altra. Ogni possibile percorso da una città all'altra rappresenta un'azione.
3. **Test obiettivo (goal test):** Il test obiettivo verifica se tutte le città sono state visitate una sola volta e se il commesso viaggiatore è tornato alla città di partenza.
4. **Funzione costo:** Il costo di un percorso può essere la distanza totale percorsa o il tempo totale impiegato per il viaggio.

L'obiettivo del problema del commesso viaggiatore è trovare il percorso più breve (o il percorso che minimizza il tempo di viaggio) che visita tutte le città una sola volta e ritorna alla città di partenza. Gli algoritmi di ricerca possono essere utilizzati per esplorare lo spazio degli stati (cioè, tutti i possibili percorsi) e trovare la soluzione ottimale.

2.4.3 Algoritmo “generale” di ricerca

In ogni istante l'agente si troverà davanti un insieme di stati possibili da esplorare. Questo insieme di stati è noto come la «frontiera» (Come nel far west :). Abbiamo bisogno di una struttura dati in grado di contenere gli stati della frontiera che l'agente può esplorare. Vedremo almeno due implementazioni. Lo pseudocodice dell'algoritmo “generale” di ricerca è il seguente:

```

1 Se la Frontiera è vuota, Finito!. Si tratta di un problema insolubile!.
2 Rimuovi un nodo dalla frontiera e consideralo come candidato.
3   Se il nodo contiene lo stato finale, Restituisci la soluzione. Finito!
4   Altrimenti
5       Cerca tutti i nodi raggiungibili dal nodo corrente e aggiungili alla frontiera.
6       Aggiungi il nodo corrente all'insieme dei nodi visitati.
7 Torna al passo 1.

```

L'algoritmo di ricerca generale è un approccio generale per risolvere problemi di ricerca. Nella descrizione di questo algoritmo è stato omesso un passo fondamentale: come si fa a scegliere il nodo da rimuovere dalla frontiera? La scelta del nodo da rimuovere dalla frontiera è un passo cruciale nell'algoritmo di ricerca. Questa scelta è basata su una strategia di ricerca, che determina l'ordine in cui i nodi vengono esplorati. L'implementazione della frontiera è quindi legata alla strategia di ricerca. Le strutture dati usate per la frontiera sono le seguenti:

- **Stack:** L'ultimo nodo inserito è il primo estratto (LIFO = Last In First Out) → Algoritmo Depth First search (**DFS**)
- **Queue:** Il primo nodo inserito è il primo estratto (FIFO = First In First Out) → Algoritmo Breadth First Search (**BFS**)
- **Priority Queue:** Il nodo con il valore di priorità più alto è il primo estratto → Algoritmo Best First Search (**BFS**)
- **Set:** L'elemento con il valore di costo più basso è il primo estratto → Algoritmo A* (**A***)

```

def ricerca_generale(problema, strategia):
    frontiera = Strategia(problema)
    while not frontiera.vuota():
        nodo = frontiera.rimuovi_nodo()
        if problema.test_obiettivo(nodo.stato):
            return nodo
        frontiera.aggiungi_nodi(nodo.genera_successori())
    return None

```

2.4.4 Strategie di ricerca non informate

Nelle strategie di ricerca non informate l'agente non vede e non sente se non il proprio stato.

Le strategie di ricerca non informate sono un tipo di algoritmo di ricerca che non utilizza alcuna conoscenza specifica o informazione aggiuntiva sul problema da risolvere. Questi algoritmi utilizzano solo la struttura del problema e la definizione di stato e azione per esplorare lo spazio degli stati. Esempi di strategie di ricerca non informate:

1. **Ricerca in profondità (Depth-First Search, DFS):** Questa strategia esplora lo spazio degli stati andando in profondità prima di esplorare i nodi adiacenti. È una strategia ricorsiva che inizia dallo stato iniziale e procede fino a quando non raggiunge uno stato finale o non può più espandere ulteriormente.
2. **Ricerca in ampiezza (Breadth-First Search, BFS):** Questa strategia esplora lo spazio degli stati espandendo prima i nodi adiacenti allo stato iniziale, quindi i nodi adiacenti ai nodi adiacenti, e così via. È una strategia che esplora lo spazio degli stati in modo uniforme, garantendo che vengano esplorati prima i nodi più vicini allo stato iniziale.
3. ...

2.4.4.1 Ricerca in profondità (Depth-First Search, DFS)

La ricerca in profondità (Depth-First Search, DFS) è una strategia di ricerca che esplora lo spazio degli stati andando in profondità prima di esplorare i nodi adiacenti. È una strategia ricorsiva che inizia dallo stato iniziale e procede fino a quando non raggiunge uno stato finale o non può più espandere ulteriormente. Questo algoritmo si basa sull'adozione di una struttura dati a coda per implementare la frontiera.

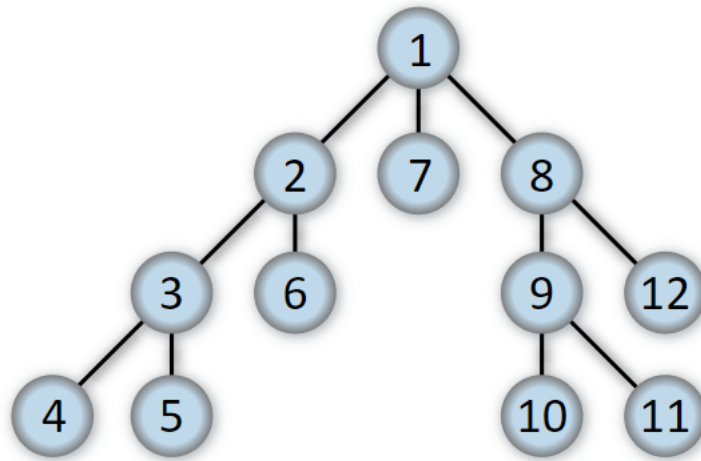


Figure 2.3: Depth-First Search, DFS

2.4.4.2 Ricerca in ampiezza (Breadth-First Search, BFS)

La ricerca in ampiezza (Breadth-First Search, BFS) è una strategia di ricerca che esplora lo spazio degli stati espandendo prima i nodi adiacenti allo stato iniziale, quindi i nodi adiacenti ai nodi adiacenti, e così via. È una strategia che esplora lo spazio degli stati in modo uniforme, garantendo che vengano esplorati prima i nodi più vicini allo stato iniziale. Questo algoritmo si basa sull'adozione di una struttura dati a pila per implementare la frontiera.

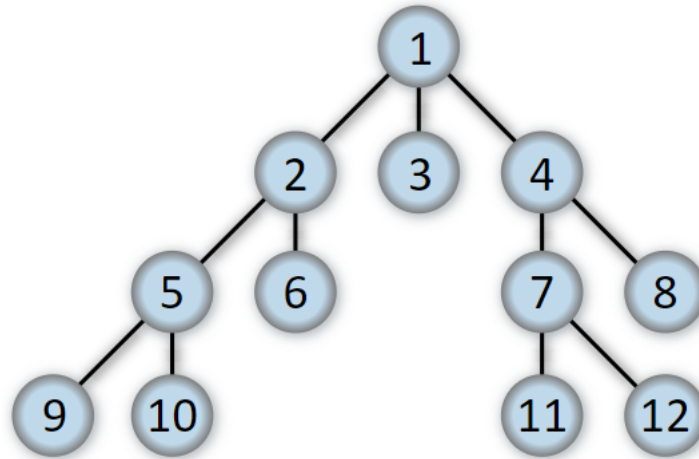


Figure 2.4: Breadth-First Search, BFS

2.4.4.3 DFS e BFS in Python

Usando l'algoritmo generico di ricerca che abbiamo visto fin qui si possono risolvere diversi problemi. Ad esempio, con l'agente AI fin qui sviluppata possiamo risolvere il problema di trovare il percorso in un labirinto. Per fare questo dobbiamo solo codificare lo spazio degli stati di questo problema e applicare la nostra AI a qualche caso reale.

struttura dati per i nodi

La struttura dati per memorizzare il generico nodo del grafo dei possibili stati è

```
class Nodo():
    def __init__(self, stato, genitore, azione):
        self.stato = stato
        self.genitore = genitore
        self.azione = azione
```

Struttura dati per la frontiera

L'altra struttura dati di cui abbiamo bisogno è una struttura dati per la Frontiera. Come abbiamo visto ci sono due tipo di struttura dati che possiamo usare per la frontiera.

- Struttura dati “pila” per la frontiera Depth First Search (DFS). La pila è una struttura dati che adotta una logica Last In First Out (l'ultimo a entrare è il primo ad uscire come accade per una pila, appunto, di piatti :)
- Struttura dati “coda” per la frontiera Breadth First Search (BFS). La coda è una struttura dati che adotta una logica First In First Out (l'elemento che entra per primo è l'elemento che esce per primo come accade nella coda ad uno sportello :)

```

class FrontieraPila():
    def __init__(self):
        self.frontiera = []

    def aggiungiStato(self, nodo):
        self.frontiera.append(nodo)

    def contieneStato(self, stato):
        return any(nodo.stato == stato for nodo in self.frontiera)

    def eVuota(self):
        return len(self.frontiera) == 0

    def rimuoviStato(self):
        if self.eVuota():
            raise Exception("Frontiera vuota")
        else:
            nodo = self.frontiera[-1]
            self.frontiera = self.frontiera[:-1]
            return nodo

class FrontieraCoda(FrontieraPila):
    def rimuoviStato(self):
        if self.eVuota():
            raise Exception("Frontiera vuota")
        else:
            nodo = self.frontiera[0]
            self.frontiera = self.frontiera[1:]
            return nodo

```

Possiamo valutare il funzionamento delle due strutture dati introdotte con una semplice simulazione. Memorizziamo 3 stati A,B e C nelle due strutture dati e osserviamo quale stato è estratto dal metodo rimuoviStato:

```

pila = FrontieraPila()
pila.aggiungiStato("A") # pila = ["A"]
pila.aggiungiStato("B") # pila = ["A", "B"]
pila.aggiungiStato("C") # pila = ["A", "B", "C"]
pila.rimuoviStato()     # pila = ["A", "B"]

```

'C'

```

coda = FrontieraCoda()
coda.aggiungiStato("A") # coda = ["A"]
coda.aggiungiStato("B") # coda = ["A", "B"]
coda.aggiungiStato("C") # coda = ["A", "B", "C"]
coda.rimuoviStato() # coda = ["B", "C"]

```

'A'

Per descrivere un labirinto useremo un semplice formato testuale come il seguente:

```

#####B#
##### #
A      #
#### ##
      ##
#####

```

Dove il simbolo # rappresenta una parete, il simbolo A un punto di partenza, il simbolo B un punto di arrivo e lo spazio una cella libera. Il labirinto sarà memorizzato in un file testuale (.txt) e sarà “passato” al risolutore di labirinti. La classe Python che si definirà qui di seguito si occupa del caricamento del labirinto da file di tipo testuale, della sua rappresentazione grafica e della sua risoluzione.

```

class Labirinto():

    def __init__(self, nomeFile):

        # legge il file del labirinto e imposta altezza e larghezza del labirinto
        with open(nomeFile) as f:
            contenuti = f.read()

        # Verifica che il file contenga almeno uno stato iniziale (= un ingresso) e uno finale
        if contenuti.count("A") != 1:
            raise Exception("Un labirinto deve avere esattamente un punto di partenza")
        if contenuti.count("B") != 1:
            raise Exception("Un labirinto deve avere esattamente un obiettivo")

        # Calcola l'altezza e la larghezza del labirinto
        contenuti = contenuti.splitlines()
        self.altezza = len(contenuti)

```

```

self.larghezza = max(len(line) for line in contenuti)

# tiene traccia dei muri del labirinto
self.muri = []
for i in range(self.altezza):
    riga = []
    for j in range(self.larghezza):
        try:
            if contenuti[i][j] == "A":
                self.start = (i, j)
                riga.append(False)
            elif contenuti[i][j] == "B":
                self.goal = (i, j)
                riga.append(False)
            elif contenuti[i][j] == " ":
                riga.append(False)
            else:
                riga.append(True)
        except IndexError:
            riga.append(False)
    self.muri.append(riga)

self.soluzione = None

def nodiVicini(self, stato):
    riga, col = stato
    candidati = [
        ("su", (riga - 1, col)),
        ("giu", (riga + 1, col)),
        ("sin", (riga, col - 1)),
        ("des", (riga, col + 1))
    ]

    risultato = []
    for azione, (r, c) in candidati:
        if 0 <= r < self.altezza and 0 <= c < self.larghezza and not self.muri[r][c]:
            risultato.append((azione, (r, c)))
    return risultato

```

```

def risolvi(self,frontiera):
    """Trova una soluzione al labirinto, se ne esiste una!"""

    # contiene il conteggio degli stati esplorati
    self.numeroStatiEsplorati = 0

    # Inizializziamo la frontiera con lo stato iniziale
    start = Nodo(stato=self.start, genitore=None, azione=None)
    frontiera.aggiungiStato(start)

    # Inizializzazione di un set di stati esplorati al momento vuoto
    self.statiEsplorati = set()

    # Continua a eseguire in ciclo finché non si trova una soluzione
    # o il problema non è risolvibile
    while True:

        # Se non c'è nulla nella frontiera vuol dire che il problema
        # non è risolvibile. Ovvero, non c'è un cammino tra start e goal!
        if frontiera.eVuota():
            raise Exception("Non esiste una soluzione")

        # Sceglie un nodo dalla frontiera
        nodo = frontiera.rimuoviStato()
        self.numeroStatiEsplorati += 1

        # Se il nodo estratto è il nodo goal allora abbiamo trovato il nodo di arrivo
        # e risolto il problema
        if nodo.stato == self.goal:
            azioni = []
            celle = []
            # ripercorre il cammino al contrario dal goal verso lo stato start
            # per creare il cammino che porta dallo start al goal. Ovvero, la soluzione.
            while nodo.genitore is not None:
                azioni.append(nodo.azione)
                celle.append(nodo.stato)
                nodo = nodo.genitore
            # Siccome ha costruito il cammino soluzione in direzione inversa, ovvero
            # da goal a start per avere il cammino orientato in modo corretto deve
            # invertire sia la lista degli stati che quella delle azioni
            azioni.reverse()
            celle.reverse()

```

```

        # Quindi memorizza la lista degli stati e la lista delle azioni da
        # intraprendere nell'attributo soluzione della classe Labirinto
        self.soluzione = (azioni, celle)
        return

    # Altrimenti marchiamo il nodo estratto come esplorato
    self.statiEsplorati.add(nodo.stato)

    # E aggiungiamo i nodi vicini alla frontiera
    for azione, stato in self.nodiVicini(nodo.stato):
        if not frontiera.contieneStato(stato) and stato not in self.statiEsplorati:
            child = Nodo(stato=stato, genitore=nodo, azione=azione)
            frontiera.aggiungiStato(child)

def stampaLabirinto(self, mostraSoluzione=True, mostraStatiEsplorati=False):
    soluzione = self.soluzione[1] if self.soluzione is not None else None
    print()
    for i, riga in enumerate(self.muri):
        for j, col in enumerate(riga):
            if col:
                print(" ", end="")
            elif (i, j) == self.start:
                print("A", end="")
            elif (i, j) == self.goal:
                print("B", end="")
            elif soluzione is not None and mostraSoluzione and \
                 (i, j) in soluzione:
                print("*", end="")
            # Stati esplorati
            elif soluzione is not None and mostraStatiEsplorati and \
                 (i, j) in self.statiEsplorati:
                print("o", end="")
            else:
                print(" ", end="")
        print()
    print()

```

Possiamo caricare un labirinto e vedere la sua stampa a video:

```
l = Labirinto("labirinto2.txt")
print("Labirinto:")
l.stampaLabirinto(False,False)
```

Labirinto:

B

A

Adesso, usando il metodo risolvi della classe labirinto appena definita possiamo risolvere il labirinto. La prima soluzione la cerchiamo con l'algoritmo DFS passando al risolutore una frontiera a pila

```
import time
from datetime import timedelta

print("Sto cercando una soluzione con una frontiera pila (DFS)...")
tempoIniziale = time.time_ns()
l.risolvi(FrontieraPila())
print("Numero di stati esplorati : ", l.numeroStatiEsplorati)
tempoImpiegato = time.time_ns() - tempoIniziale
msg = "L'esecuzione del codice ha richiesto : %s microsecondi (Wall clock time)" \
      % timedelta(microseconds=round(tempoImpiegato/1000))
print(msg)
print("Soluzione trovata : ")
l.stampaLabirinto(mostraStatiEsplorati=True,mostraSoluzione=True)
```

Sto cercando una soluzione con una frontiera pila (DFS)...

Numero di stati esplorati : 194

L'esecuzione del codice ha richiesto : 0:00:00.000809 microsecondi (Wall clock time)

Soluzione trovata :

```

      oooooooooooooooooo
ooo          ooo o
o  oooooooooooooooooo o o o
o          o o o o
oooooooooooooooooooooooo o o o
          o o o o
      *****ooooooooo o o o
      *      *      o o o
****      B oooooooooo o o
* o          o o o
* oooooooooooooooooo o o o
*          o o o o o
****ooooooooooooo oooo o o o
*      o      o o o
*      oooooooooooooo ooo
A*****
```

Quindi, cerchiamo con l'algoritmo BFS passando al risolutore una frontiera a coda

```
print("Sto cercando una soluzione con una frontiera a coda (BFS)...")
tempoIniziale = time.time_ns()
l.risolvi(FrontieraCoda())
print("Numero di stati esplorati : ", l.numeroStatiEsplorati)
tempoImpiegato = time.time_ns() - tempoIniziale
msg = "L'esecuzione del codice ha richiesto : %s microsecondi (Wall clock time)" \
      % timedelta(microseconds=round(tempoImpiegato/1000))
print(msg)
print("Soluzione trovata : ")
l.stampaLabirinto(mostraStatiEsplorati=True,mostraSoluzione=True)
```

Sto cercando una soluzione con una frontiera a coda (BFS)...

Numero di stati esplorati : 77

L'esecuzione del codice ha richiesto : 0:00:00 microsecondi (Wall clock time)

Soluzione trovata :


```

ooo *****o
o o * o *
o **** ooo B
o * o
* oooooooooo
*
****ooooooooo
*      o      o
*      ooooooooooooo
A*****

```

Osserviamo dai risultati ottenuti che per questo labirinto l'algoritmo di ricerca più veloce è il BFS perché ha trovato la soluzione visitando 77 nodi mentre l'algoritmo DFS ha visitato 194 nodi per arrivare alla stessa soluzione.

2.4.5 Algoritmi di ricerca informati

Gli algoritmi di ricerca informati sono una classe di algoritmi di ricerca che utilizzano una funzione euristica per guidare la ricerca verso la soluzione in modo più efficiente rispetto agli algoritmi di ricerca non informati come BFS e DFS. Questi algoritmi sfruttano informazioni aggiuntive sul problema, come la distanza stimata dalla soluzione, per esplorare in modo intelligente lo spazio di ricerca.

Uno degli algoritmi di ricerca informati più noti è l'algoritmo A* (pronunciato "A star"). Esso combina in modo bilanciato le informazioni sulla distanza già percorsa e una stima della distanza rimanente dalla soluzione, utilizzando una funzione euristica. L'algoritmo A* è completo, ovvero garantisce di trovare una soluzione se esiste, ed è anche ottimale, cioè trova il percorso più breve verso la soluzione se la funzione euristica è ammissibile.

Altri algoritmi di ricerca informati includono la ricerca di best-first, che espande sempre il nodo più promettente in base alla funzione euristica, e la ricerca greedy, che si basa esclusivamente sulla stima euristica senza considerare il costo del percorso già fatto. Questi algoritmi possono essere più veloci dell'A* in alcuni casi, ma non garantiscono necessariamente di trovare la soluzione ottimale.

Gli algoritmi di ricerca informati trovano applicazione in numerosi campi, come l'intelligenza artificiale, la robotica, la pianificazione di percorsi e la risoluzione di problemi di ottimizzazione. La scelta dell'algoritmo più appropriato dipende dalle caratteristiche del problema,

come la complessità dello spazio di ricerca, la disponibilità di informazioni euristiche accurate e i requisiti di ottimalità della soluzione.

2.4.5.1 Funzioni euristiche

Le funzioni euristiche svolgono un ruolo cruciale negli algoritmi di ricerca informati, fornendo una stima della distanza o del costo rimanente per raggiungere la soluzione. Queste funzioni sono progettate per guidare la ricerca in modo intelligente, evitando di esplorare percorsi poco promettenti e concentrandosi sulle regioni dello spazio di ricerca più vicine alla soluzione.

Una buona funzione euristica dovrebbe essere ammissibile, ovvero non sovrastimare mai il costo effettivo per raggiungere la soluzione. Ciò garantisce che l'algoritmo di ricerca, come A^* , trovi una soluzione ottimale se esiste. Inoltre, una funzione euristica accurata e informativa può accelerare notevolmente la ricerca, riducendo il numero di nodi esplorati prima di trovare la soluzione.

Nella risoluzione di labirinti, una funzione euristica comune è la distanza di Manhattan o la distanza euclidea tra la posizione corrente e l'uscita del labirinto. Queste funzioni forniscono una stima della distanza minima rimanente, ignorando gli ostacoli presenti nel labirinto. Tuttavia, funzioni euristiche più sofisticate possono tenere conto di ulteriori informazioni, come la disposizione degli ostacoli o la topologia del labirinto, per ottenere stime più accurate.

La progettazione di funzioni euristiche efficaci è spesso una sfida cruciale nell'applicazione degli algoritmi di ricerca informati a problemi complessi del mondo reale.

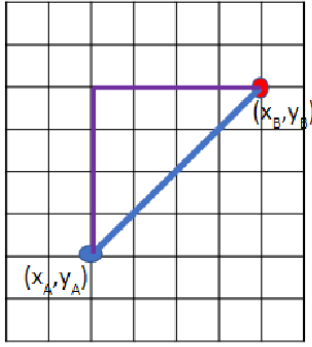


Figure 2.5: Distanze euclidee e di Manhattan

Distanza euclidea La distanza euclidea, anche nota come distanza in linea retta, è una misura della distanza tra due punti in uno spazio euclideo, come il piano cartesiano o lo spazio tridimensionale. Essa rappresenta la lunghezza del segmento di retta che congiunge i due punti.

La formula per calcolare la distanza euclidea tra due punti $A = (x_A, y_A)$ e $B = (x_B, y_B)$ in un piano cartesiano bidimensionale è:

$$d_{euclidean} = \sqrt{(x_B - x_A)^2 + (y_B - y_A)^2}$$

La distanza euclidea è ampiamente utilizzata come funzione euristica negli algoritmi di ricerca informati, come l'algoritmo A*, per stimare la distanza rimanente dalla soluzione. Essa fornisce una stima ammissibile (non sovrastima) della distanza effettiva, soddisfacendo così i requisiti per garantire l'ottimalità dell'algoritmo di ricerca.

Tuttavia, la distanza euclidea può essere una stima poco accurata in alcuni contesti, come nei labirinti o in presenza di ostacoli, poiché non tiene conto degli impedimenti lungo il percorso. In questi casi, possono essere utilizzate funzioni euristiche più sofisticate per ottenere stime più precise.

Distanza di Manhattan La distanza di Manhattan, anche nota come distanza city-block o distanza tassista, è una metrica utilizzata per calcolare la distanza tra due punti in uno

spazio a coordinate cartesiane. Essa prende il nome dalla griglia di strade di Manhattan, dove i percorsi possibili sono limitati a spostamenti orizzontali e verticali.

La formula per calcolare la distanza di Manhattan tra due punti $A = (x_A, y_A)$ e $B = (x_B, y_B)$ in un piano cartesiano bidimensionale è:

$$d_{Manhattan} = (x_B - x_A) + (y_B - Y_A)$$

Essenzialmente, la distanza di Manhattan è la somma delle differenze assolute delle coordinate x e y dei due punti.

La distanza di Manhattan è spesso utilizzata come funzione euristica negli algoritmi di ricerca informati, come l'algoritmo A*, per risolvere problemi di ricerca su griglie o labirinti. Essa fornisce una stima ammissibile della distanza effettiva, garantendo così l'ottimalità dell'algoritmo di ricerca.

Rispetto alla distanza euclidea, la distanza di Manhattan può essere una stima più accurata in contesti come i labirinti, poiché tiene conto delle restrizioni di movimento lungo le direzioni orizzontali e verticali. Tuttavia, può sottostimare la distanza effettiva in situazioni in cui sono possibili percorsi diagonali.

La scelta tra la distanza euclidea e la distanza di Manhattan come funzione euristica dipende dalle caratteristiche specifiche del problema di ricerca e dalle proprietà dello spazio di ricerca.

2.4.5.2 Algoritmo di ricerca informato Greedy Best-First Search

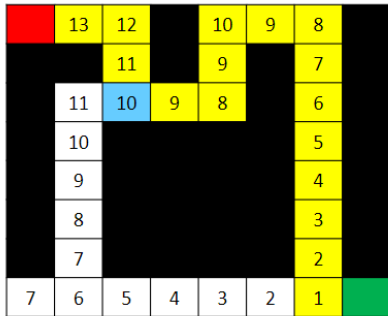


Figure 2.6: ricerca informata

Quando l'algoritmo si trova nella cella colorata di azzurro deve scegliere se proseguire nella cella a distanza 9 o in quella a distanza 11 dal goal. Quale scegliere? GBF sceglie la cella con distanza 9 dall'obiettivo. È una buona scelta? Direi di no! Il cammino scelto è il più lungo. Forse, si può fare di meglio? Se esaminiamo le celle a e b notiamo l'euristica che ci ha portato in b con GBS è minore dell'euristica in a. Ma, che cosa succede se oltre a considerare la distanza dall'obiettivo aggiungo il cammino fatto all'euristica h? Ovvero:

$$H = \text{distanza da percorrere} + \text{distanza percorsa}$$

Entra l'algoritmo informato A*

2.4.5.3 Algoritmo di ricerca informato A*



Figure 2.7: Ricerca informata A*

L'algoritmo A* è un algoritmo di ricerca informato che utilizza una funzione di valutazione per guidare la ricerca verso la soluzione ottimale. La funzione di valutazione, nota come funzione di costo $h(n)$, è composta da due componenti: 1. **Costo del cammino:** $g(n)$, che rappresenta il costo del cammino per raggiungere il nodo n dalla radice. 2. **Stima del costo rimanente:** $f(n)$, che rappresenta una stima del costo rimanente per raggiungere la soluzione ottimale partendo dal nodo n . La funzione di valutazione $h(n)$ è definita come:

$$h(n) = g(n) + f(n)$$

L'algoritmo A* utilizza una coda di priorità per mantenere i nodi da esplorare in base al valore della funzione di valutazione $f(n)$. I nodi con il valore più basso di $h(n)$ vengono estratti dalla coda e esplorati prima. Nel caso in figura, con questa nuova euristica vediamo che a è una scelta migliore di b perché ha una euristica minore.

2.5 Algoritmi Equitativi

L'avvento dell'Intelligenza Artificiale e il progresso nella capacità computazionale delle moderne macchine hanno rivoluzionato molteplici aspetti della nostra vita quotidiana. Tra le numerose applicazioni dell'IA, gli algoritmi predittivi e gli algoritmi di ripartizione equitativa si distinguono per il loro potenziale straordinario e per le sfide etiche che presentano.

Gli algoritmi predittivi, capaci di effettuare previsioni estremamente accurate in svariati scenari, sono diventati strumenti essenziali in settori cruciali come quello giudiziario, creditizio,

assicurativo e sanitario. Questi algoritmi analizzano enormi quantità di dati per prendere decisioni che possono avere un impatto significativo sulla vita delle persone. Tuttavia, il loro crescente impiego ha sollevato una preoccupazione fondamentale: l'equità.

L'equità negli algoritmi si riferisce alla loro capacità di trattare in modo imparziale tutti i gruppi di persone, indipendentemente da attributi sensibili come etnia, genere, età o stato socioeconomico. Senza adeguate misure di equità, gli algoritmi rischiano di perpetuare o addirittura amplificare disuguaglianze sociali ed economiche esistenti, poiché i dati utilizzati per il loro addestramento possono contenere pregiudizi storici e sistemici.

Parallelamente, gli algoritmi di ripartizione equitativa giocano un ruolo cruciale nella distribuzione giusta e bilanciata di risorse o beni tra più parti. Questi algoritmi trovano applicazione in scenari diversi, dalla divisione dei beni durante un divorzio alla distribuzione di fondi di emergenza in situazioni di crisi.

Per affrontare le sfide legate all'equità, sono stati sviluppati vari approcci e metodologie. Tra questi, la pre-elaborazione dei dati mira a correggere i bias presenti nei dati prima dell'addestramento degli algoritmi. Durante lo sviluppo, si possono includere vincoli di equità nei processi di ottimizzazione per evitare che l'algoritmo favorisca ingiustamente un gruppo rispetto a un altro. Il post-processamento dei risultati permette di aggiustare le previsioni per eliminare disparità tra gruppi diversi.

La trasparenza e la spiegabilità degli algoritmi sono essenziali per affrontare le questioni etiche correlate. Spesso, gli algoritmi più avanzati sono percepiti come "scatole nere", rendendo difficile comprendere i processi decisionali e attribuire responsabilità in caso di errori. Questo solleva importanti questioni di responsabilità e trasparenza.

L'implementazione di algoritmi equitativi può avere impatti significativi in vari settori. Nel sistema giudiziario, strumenti equi possono promuovere la fiducia nel sistema legale. Nei processi di assunzione, possono garantire valutazioni basate sulle competenze piuttosto che su caratteristiche personali. Nel campo sanitario, possono migliorare l'accesso e la qualità delle cure per tutte le popolazioni.

In conclusione, l'equità nell'IA è un aspetto cruciale che richiede attenzione e considerazione. Mentre gli algoritmi predittivi e gli algoritmi di ripartizione equitativa rivestono un ruolo fondamentale nella nostra vita quotidiana, è essenziale garantire che siano sviluppati e utilizzati in modo equo e trasparente. Solo così possiamo trarre il massimo beneficio dalla potenza dell'IA senza incorrere in potenziali disuguaglianze e pregiudizi. In questo paragrafo, esploreremo gli algoritmi di ripartizione equitativa, che sono essenziali per garantire che le risorse o i beni siano distribuiti in modo equo tra le parti interessate. Questi algoritmi trovano applicazione in scenari diversi, dalla divisione dei beni durante un divorzio alla distribuzione di fondi di emergenza in situazioni di crisi.

2.5.1 agenti partecipanti

In generale, si ha a che fare con un insieme N di agenti o partecipanti o giocatori. Questi agenti hanno la necessità di mettersi d'accordo sulla divisione di un certo numero M di beni, risorse, oggetti, ecc.

2.5.2 beni

Gli algoritmi di suddivisione equa possono essere applicati a diversi tipi di beni, ciascuno con caratteristiche specifiche che influenzano il modo in cui la suddivisione deve essere effettuata. Questi beni possono essere classificati in diverse categorie:

Beni Divisibili I beni divisibili sono quelli che possono essere suddivisi in parti più piccole senza perdere il loro valore intrinseco. Esempi includono:

- **Cibo:** come una torta o una pizza, che possono essere tagliati in fette.
- **Terreni:** una proprietà terriera può essere suddivisa in appezzamenti più piccoli.
- **Denaro:** che può essere facilmente diviso in unità più piccole.

Beni Indivisibili I beni indivisibili non possono essere suddivisi senza perdere il loro valore o funzionalità. Esempi includono:

- **Oggetti fisici unici:** come una macchina, un'opera d'arte o un elettrodomestico.
- **Ruoli o incarichi:** come una posizione lavorativa o un incarico specifico in un progetto.

Beni Combinati Alcuni beni possono essere considerati una combinazione di elementi divisibili e indivisibili. Ad esempio:

- **Pacchetti di beni:** come un set di mobili dove ogni pezzo è indivisibile, ma il set complessivo può essere suddiviso.
- **Progetti con compiti specifici:** dove i singoli compiti possono essere indivisibili, ma l'intero progetto può essere suddiviso tra diversi partecipanti.

Beni con Valore Soggettivo Alcuni beni hanno un valore che varia a seconda delle preferenze individuali dei partecipanti. Esempi includono:

- **Oggetti con valore sentimentale:** come regali o ricordi di famiglia.
- **Elementi artistici o culturali:** come quadri o libri, il cui valore può dipendere dal gusto personale.

Beni Temporanei Questi sono beni che possono essere utilizzati per un certo periodo di tempo e poi riassegnati. Esempi includono:

- **Uso di risorse comuni:** come una sala conferenze, un campo sportivo o un'attrezzatura condivisa.

- **Servizi o turni di lavoro:** dove il tempo di servizio o il turno può essere diviso tra più persone.

Beni Digitali I beni digitali possono essere suddivisi e duplicati senza perdere valore. Esempi includono:

- **Software:** che può essere concesso in licenza a più utenti.
- **Contenuti digitali:** come e-book, musica o video, che possono essere condivisi tra più persone. Per affrontare questi problemi, gli algoritmi di suddivisione equa utilizzano criteri diversi, come la proporzionalità, l'efficienza, l'equità, l'invidia-zero (nessun partecipante dovrebbe invidiare la parte degli altri) e altre nozioni di giustizia.

Esempi di criteri di equità

1. **Proporzionalità:** Ogni partecipante riceve una quota proporzionale alle proprie pretese o contributi.
2. **Invidia-zero:** Nessun partecipante deve preferire la parte assegnata a un altro partecipante alla propria parte.
3. **Efficienza Pareto:** Non è possibile riassegnare le risorse in modo che qualcuno sia in una situazione migliore senza che qualcun altro sia in una situazione peggiore.
4. **Equità equitativa:** Ogni partecipante percepisce di aver ricevuto una parte equa in base a criteri specifici.

Gli algoritmi di suddivisione equa cercano di trovare soluzioni che bilancino questi criteri, a seconda delle specifiche esigenze del contesto in cui vengono applicati.

2.5.3 Regole e assunzioni

Regole Affinché la divisione di un bene S sia equa: - i giocatori devono essere partecipanti volontari e accettare le regole del gioco come vincolanti.

- I giocatori devono agire razionalmente secondo il loro sistema di credenze.
- Le regole della matematica si applicano quando si assegnano valori agli oggetti in S .
- Solo i giocatori sono coinvolti nel gioco, non ci sono agenti esterni come avvocati o altri intermediari.

Se i giocatori seguono le regole, il gioco terminerà dopo un numero finito di mosse dei giocatori e risulterà in una divisione di S .

Assunzioni

Gli algoritmi si basano sulle seguenti assunzioni quanto segue:

- Tutti i giocatori giocano in modo corretto.

- Non hanno informazioni precedenti sui gusti o le avversioni degli altri giocatori.
- Non assegnano valori in modo da manipolare il gioco.
- Tutti i giocatori hanno uguali diritti nella condivisione dell'insieme S. In altre parole, se ci sono tre giocatori, ogni giocatore ha diritto ad almeno $1/3$ di S.

Se queste assunzioni non sono soddisfatte, la divisione potrebbe non essere completamente equa.

2.5.4 Matematica elementare per algoritmi di ripartizione equa

- **Percentuale:** Una percentuale è una frazione con un denominatore di 100. Ad esempio, il 50% è la metà, il 25% è un quarto e il 100% è l'intero.
- **Percentuale di un numero:** Per trovare la percentuale di un numero, moltiplica il numero per la percentuale come decimale. Ad esempio, il 25% di 80 è $0,25 \times 80 = 20$.
- **Percentuale di un numero n1 rispetto a un numero n2:** Per calcolare la percentuale di un numero n1 rispetto a un numero n2, dividi n1 per n2 e moltiplica per 100. Ad esempio, se 20 è una parte di 80, si ha che 20 è il $(20/80) \times 100$ di 80 o il 25% di 80.

esempio: 1. Alice e Bob hanno un sacchetto di 100 monete. Alice ha 60 monete, mentre Bob ha 40 monete. Qual è la percentuale di monete di Alice rispetto a Bob? Soluzione: - La percentuale di monete di Alice rispetto a Bob: $(60/40) \times 100 = 150\%$. - Quindi, Alice ha il 150% delle monete di Bob.

2. Alice, Bob, Claudia e Daniele hanno una torta. Alice ha 2 fette, Bob ha 3 fette, Claudia ha 4 fette e Daniele ha 1 fetta. Qual è la percentuale di fette di Alice rispetto a Bob? Soluzione:

- La percentuale di fette di Alice rispetto a Bob: $(2/3) \times 100 = 66,67\%$. Qual è la percentuale di torta che ha Alice? Soluzione:
- La percentuale di torta che ha Alice: $(2/10) \times 100 = 20\%$.

2.5.5 Algoritmi di Ripartizione Equitativa

Un problema tipico di ripartizione equa può essere formulato come segue: Alice, Bruno, Carla e Davide hanno una torta. La torta è divisa in 4 parti non necessariamente uguali e non con la stessa farcitura e/o copertura. Ognuno dei 4 partecipanti ha una sua preferenza per ognuna delle quattro parti. Le preferenze sono riassunte nella seguente tabella:

Partecipante	porzione 1	porzione 2	porzione 3	porzione 4
Alice	10%	50%	30%	10%
Bruno	30%	30%	10%	30%

Partecipante	porzione 1	porzione 2	porzione 3	porzione 4
Carla	40%	20%	20%	20%
Davide	25%	25%	25%	25%

La domanda che ci si pone è: Quale porzione di torta considererebbe equa ogni giocatore? È importante ricordare che un algoritmo di ripartizione equa si basa sulle assunzioni del paragrafo 3.5.3. Pertanto, ogni giocatore ha espresso la propria preferenza senza conoscere le preferenze degli altri partecipanti. La soluzione, in questo caso, è illustrata nella tabella seguente, dove è evidenziata la porzione assegnata a ciascun giocatore, rispettando le preferenze manifestate.

Partecipante	porzione 1	porzione 2	porzione 3	porzione 4
Alice	10%	50%	30%	10%
Bruno	30%	30%	10%	30%
Carla	40%	20%	20%	20%
Davide	25%	25%	25%	25%

L'implementazione dell'algoritmo usato per ottenere la soluzione è riportata nella prossima sezione

```
def divisione_equa(preferenze):
    # 1. **Inizializzazione** : Converti il dizionario delle preferenze in una lista
    # di tuple per una gestione più semplice
    partecipanti = list(preferenze.keys())
    valori_preferenze = list(preferenze.values())

    # Numero di partecipanti e porzioni
    num_partecipanti = len(partecipanti)
    num_porzioni = len(valori_preferenze[0])

    # Inizializza la lista di allocazione
    allocazione = [-1] * num_partecipanti
    porzioni_usate = [False] * num_porzioni

    # 2. **Funzione `trova_preferenza_massima`** : Funzione per trovare il partecipante
    # con la preferenza più alta per una data porzione
    def trova_preferenza_massima(porzione):
        preferenza_massima = -1
        indice_partecipante = -1
        for i in range(num_partecipanti):
            if valori_preferenze[i][porzione] > preferenza_massima and allocazione[i] == -1:
```

```

        preferenza_massima = valori_preferenze[i][porzione]
        indice_partecipante = i
    return indice_partecipante

# 3. **Assegnazione delle Porzioni** : Assegna le porzioni ai partecipanti
for porzione in range(num_porzioni):
    indice_partecipante = trova_preferenza_massima(porzione)
    allocazione[indice_partecipante] = porzione
    porzioni_usate[porzione] = True

# 4. **Creazione del Risultato** : Crea un dizionario di risultato per mappare
# i partecipanti alle loro porzioni allocate
risultato = {partecipanti[i]: f"porzione {allocazione[i] + 1}" \
              for i in range(num_partecipanti)}

return risultato

# 5. **Esecuzione del Codice** : Esegui il codice con le preferenze fornite

# Preferenze dei partecipanti
preferenze = {
    'Alice': [10, 50, 30, 10], # Preferenze per le porzioni 1, 2, 3, e 4
    'Bruno': [30, 30, 10, 30],
    'Carla': [40, 20, 20, 20],
    'Davide': [25, 25, 25, 25]
}

# Trova la divisione equa delle porzioni
allocazione = divisione_equa(preferenze)

# Stampa il risultato
print("Una divisione equa delle porzioni è la seguente:")
for partecipante, porzione in allocazione.items():
    print(f"{partecipante} riceve {porzione}")

```

Una divisione equa delle porzioni è la seguente:

Alice riceve porzione 2
 Bruno riceve porzione 4
 Carla riceve porzione 1
 Davide riceve porzione 3

Il semplice codice Python proposto implementa un algoritmo di divisione equa nel seguente

modo:

1. Inizializzazione:

- Convertiamo il dizionario delle preferenze in una lista di tuple per una gestione più semplice.
- Otteniamo i nomi dei partecipanti e le loro preferenze.
- Inizializziamo le liste `allocazione` e `porzioni_usate` per tenere traccia delle porzioni assegnate e delle porzioni già utilizzate.

2. Funzione `trova_preferenza_massima`:

- Questa funzione trova il partecipante con la preferenza più alta per una data porzione che non ha ancora ricevuto una porzione.
- Scorre tutti i partecipanti e confronta le loro preferenze per la porzione corrente, restituendo l'indice del partecipante con la preferenza massima.

3. Assegnazione delle Porzioni:

- Per ogni porzione, troviamo il partecipante con la preferenza più alta utilizzando la funzione `trova_preferenza_massima`.
- Assegniamo la porzione a quel partecipante e segniamo la porzione come utilizzata.

4. Creazione del Risultato:

- Creiamo un dizionario `risultato` che mappa i partecipanti alle loro porzioni assegnate.
- Restituiamo il dizionario `risultato`.

5. Esecuzione del Codice:

- Definiamo le preferenze dei partecipanti.
- Chiamiamo la funzione `divisione_equa` per ottenere la divisione delle porzioni.
- Stampiamo il risultato.

Evidentemente, l'algoritmo proposto non è l'unica soluzione possibile e può dare origine a situazioni di iniquità o di conflitti. Ad esempio se le preferenze dei partecipanti sono:

Partecipante	porzione 1	porzione 2	porzione 3	porzione 4
Alice	10%	50%	30%	10%
Bruno	30%	30%	10%	30%
Carla	20%	40%	20%	20%
Davide	25%	25%	25%	25%

Si noti il conflitto tra Alice e Carla per la porzione 2. La soluzione a cui arriva l'algoritmo visto è:

```

preferenze = {
    'Alice': [10, 50, 30, 10], # Preferenze per le porzioni 1, 2, 3, e 4
    'Bruno': [30, 30, 10, 30],
    'Carla': [20, 40, 20, 20],
    'Davide': [25, 25, 25, 25]
}

# Trova la divisione equa delle porzioni
allocazione = divisione_equa(preferenze)

# Stampa il risultato
print("Una divisione equa delle porzioni è la seguente:")
for partecipante, porzione in allocazione.items():
    print(f"{partecipante} riceve {porzione}")

```

Una divisione equa delle porzioni è la seguente:

Alice riceve porzione 2
 Bruno riceve porzione 1
 Carla riceve porzione 4
 Davide riceve porzione 3

In questo caso, Carla riceve una porzione per la quale ha espresso un interesse minore e potrebbe invidiare Alice, che ha ottenuto proprio la porzione che lei avrebbe preferito. Tuttavia, non ci si può fare niente. La soluzione proposta è la migliore possibile date le preferenze espresse e i beni indivisibili disponibili.

Valore soggettivo di un bene

Prima di trattare l'argomento dell'invidia, c'è un altro aspetto interessante da approfondire: il valore soggettivo del bene. Ad esempio, nella suddivisione esaminata, se la torta costa 18 €, quale sarebbe il valore economico di ogni porzione di torta per ciascun partecipante?

Temendo conto che le preferenze sono le seguenti:

Partecipante	porzione 1	porzione 2	porzione 3	porzione 4
Alice	10%	50%	30%	10%
Bruno	30%	30%	10%	30%
Carla	40%	20%	20%	20%
Davide	25%	25%	25%	25%

Alice ha il 50% di preferenze per la porzione 2 quindi per lei la porzione 2 vale il 50% di 18€ = 9€ e così via. La tabella dei valori delle singole porzioni per ogni partecipante è la seguente:

Partecipante	porzione 1	porzione 2	porzione 3	porzione 4
Alice	1,80€	9,00€	5,40€	1,80€
Bruno	5,40€	5,40€	1,80€	5,40€
Carla	7,20€	3,60€	3,60€	3,60€
Davide	4,50€	4,50€	4,50€	4,50€

Vediamo un altro esempio di calcolo del valore soggettivo di un bene. Alice vede una torta che costa 18€ di cui una metà al cioccolato e l'altra al pistacchio. Alice ama il cioccolato e non ama il pistacchio. Alice valuta la porzione al cioccolato al 90% e la porzione al pistacchio al 10%. Quindi Alice valuta la metà al cioccolato al 90% di $18€ = 16,20€$ e la metà al pistacchio al 10% di $18€ = 1,80€$.

In generale, il valore soggettivo di un bene può essere influenzato da una varietà di fattori, che riflettono le percezioni individuali e le circostanze specifiche. Ecco alcuni degli aspetti principali che possono influenzare il valore soggettivo:

- *Scarsità*: Un bene raro o difficile da reperire tende ad avere un valore soggettivo più elevato¹. Domanda: La popolarità o la desiderabilità di un bene possono aumentarne il valore percepito¹.
- *Preferenze personali*: Gli interessi, i gusti e le preferenze individuali giocano un ruolo significativo nel determinare il valore di un bene per una persona¹.
- *Significato culturale*: Il valore di un bene può essere influenzato dal suo significato o dalla sua importanza in una determinata cultura.
- *Circostanze situazionali*: Eventi specifici o situazioni particolari possono alterare il valore di un bene. Ad esempio, l'acqua potrebbe avere un valore molto più alto in un deserto rispetto a una città².
- *Affinità personale*: Il legame emotivo o la storia personale con un bene possono aumentarne il valore per un individuo².
- *Incertezza e mancanza di conoscenza*: A volte le persone possono valutare l'importanza di un bene in modo non conforme alla sua reale importanza a causa dell'incertezza o della mancanza di informazioni.

Questi fattori dimostrano che il valore di un bene non è fisso o intrinseco, ma è piuttosto determinato dalle percezioni e dalle circostanze individuali. La teoria del valore soggettivo sostiene che il valore di un bene dipende dall'ambiente e dalle persone che lo percepiscono, piuttosto che dai costi di produzione o dal lavoro necessario per crearlo¹.

Mitigazione dell'invidia

Come anticipato, è necessario approfondire il problema della divisione equa senza invidia. Immaginiamo due amici, Alice e Bruno, che devono dividersi una serie di oggetti di valore in modo che nessuno dei due si senta invidioso dell'altro. Ad esempio, supponiamo che Alice e Bruno debbano dividersi i seguenti beni con le rispettive valutazioni:

Oggetto	Alice	Bruno
orologio	4	2
libro	2	3
penna	2	3
quadro	2	2

Nel seguito si propone una implementazione dell'algoritmo envy free in Python:

```
def allocazione_senza_invidia(beni, valutazioni): # 1. **Definizione della Funzione**
    """
    Algoritmo di Envy-Free per la divisione di beni indivisibili tra due persone.

    beni: lista di beni da dividere
    valutazioni: dizionario con le valutazioni dei beni per ciascun partecipante
    """
    # 2. **Inizializzazione delle Assegnazioni**:
    assegnazione = {'Alice': [], 'Bob': []}
    valori_totali = {'Alice': 0, 'Bob': 0}

    # 3. **Ordinamento dei Beni**: Ordina gli oggetti in base alla somma delle valutazioni
    beni_ordinati = sorted(beni, key=lambda x: valutazioni['Alice'][x] + \
        valutazioni['Bob'][x], reverse=True)

    # 4. **Assegnazione dei Beni**: Assegna gli oggetti in modo da bilanciare i valori totali
    for bene in beni_ordinati:
        if valori_totali['Alice'] <= valori_totali['Bob']:
            assegnazione['Alice'].append(bene)
            valori_totali['Alice'] += valutazioni['Alice'][bene]
        else:
            assegnazione['Bob'].append(bene)
            valori_totali['Bob'] += valutazioni['Bob'][bene]

    # 5. **Verifica e Correzione delle Invidie**: Verifica e corregge eventuali invidie
    for bene in beni_ordinati:
        if valutazioni['Alice'][bene] > valutazioni['Bob'][bene] and \
            bene in assegnazione['Bob']:
            if valori_totali['Alice'] < valori_totali['Bob']:
                assegnazione['Bob'].remove(bene)
                assegnazione['Alice'].append(bene)
                valori_totali['Alice'] += valutazioni['Alice'][bene]
                valori_totali['Bob'] -= valutazioni['Bob'][bene]
```



```

elif valutazioni['Bob'][bene] > valutazioni['Alice'][bene] and \
    bene in assegnazione['Alice']:
    if valori_totali['Bob'] < valori_totali['Alice']:
        assegnazione['Alice'].remove(bene)
        assegnazione['Bob'].append(bene)
        valori_totali['Bob'] += valutazioni['Bob'][bene]
        valori_totali['Alice'] -= valutazioni['Alice'][bene]

return assegnazione # 6. **Ritorno delle Assegnazioni**:
```

Il funzionamento del codice è il seguente:

1. Definizione della Funzione:

```
def allocazione_senza_invidia(beni, valutazioni):
```

- Definisce una funzione chiamata `allocazione_senza_invidia` che prende due parametri: `beni` (lista di beni da dividere) e `valutazioni` (dizionario con le valutazioni dei beni per ciascun partecipante).

2. Inizializzazione delle Assegnazioni:

```
assegnazione = {'Alice': [], 'Bob': []}
valori_totali = {'Alice': 0, 'Bob': 0}
```

- Inizializza due dizionari: `assegnazione` per tenere traccia dei beni assegnati a ciascun partecipante e `valori_totali` per tenere traccia del valore totale dei beni assegnati a ciascun partecipante.

3. Ordinamento dei Beni:

```
beni_ordinati = sorted(beni, key=lambda x: valutazioni['Alice'][x] + \
    valutazioni['Bob'][x], reverse=True)
```

- Ordina i beni in base alla somma delle valutazioni di Alice e Bob, in ordine decrescente.

4. Assegnazione dei Beni:

```
for bene in beni_ordinati:
    if valori_totali['Alice'] <= valori_totali['Bob']:
        assegnazione['Alice'].append(bene)
        valori_totali['Alice'] += valutazioni['Alice'][bene]
    else:
        assegnazione['Bob'].append(bene)
        valori_totali['Bob'] += valutazioni['Bob'][bene]
```

- Assegna i beni in modo da bilanciare i valori totali tra Alice e Bob. Se il valore totale di Alice è minore o uguale a quello di Bob, il bene viene assegnato ad Alice, altrimenti a Bob.

5. Verifica e Correzione delle Invidie:

```
for bene in beni_ordinati:
    if valutazioni['Alice'][bene] > valutazioni['Bob'][bene] and \
        bene in assegnazione['Bob']:
        if valori_totali['Alice'] < valori_totali['Bob']:
            assegnazione['Bob'].remove(bene)
            assegnazione['Alice'].append(bene)
            valori_totali['Alice'] += valutazioni['Alice'][bene]
            valori_totali['Bob'] -= valutazioni['Bob'][bene]
        elif valutazioni['Bob'][bene] > valutazioni['Alice'][bene] and \
            bene in assegnazione['Alice']:
            if valori_totali['Bob'] < valori_totali['Alice']:
                assegnazione['Alice'].remove(bene)
                assegnazione['Bob'].append(bene)
                valori_totali['Bob'] += valutazioni['Bob'][bene]
                valori_totali['Alice'] -= valutazioni['Alice'][bene]
```

- Verifica se ci sono invidie e corregge le assegnazioni di conseguenza. Se Alice valuta un bene più di Bob e il bene è assegnato a Bob, viene riassegnato ad Alice se il valore totale di Alice è inferiore a quello di Bob, e viceversa.

6. Ritorno delle Assegnazioni:

```
return assegnazione
```

- Restituisce il dizionario delle assegnazioni finali.

Caso d'Uso Reale

Divisione di una serie di oggetti di valore tra Alice e Bruno, in modo che nessuno dei due si senta invidioso dell'altro. Ad esempio, supponiamo che Alice e Bruno debbano dividersi i seguenti beni con le rispettive valutazioni personali:

```
beni = ['orologio', 'libro', 'penna']
valutazioni = {
    'Alice': {'orologio': 10, 'libro': 5, 'penna': 1},
    'Bob': {'orologio': 8, 'libro': 7, 'penna': 2}
}
```

Utilizzando la funzione `allocazione_senza_invidia`, possiamo ottenere una divisione equa:

```
assegnazione = allocazione_senza_invidia(beni, valutazioni)
print(assegnazione)
```

```
{'Alice': ['orologio'], 'Bob': ['libro', 'penna']}
```

In questo caso, Alice riceve l'orologio, mentre Bob riceve il libro e la penna, garantendo che nessuno dei due si senta invidioso dell'altro.

2.5.6 Gli algoritmi di divisione equa in letteratura

Gli algoritmi di ripartizione equa sono un insieme di metodi utilizzati per distribuire risorse limitate tra più utenti o gruppi in modo equo e imparziale. Questi algoritmi mirano a garantire che ogni utente o gruppo riceva una quota equa delle risorse, tenendo conto di fattori come le esigenze individuali, le priorità e le limitazioni. Esistono diversi tipi di algoritmi di ripartizione equa, tra questi si citano:

1. Metodi per la Divisione Equa tra Due Partecipanti - Adjusted Winner (AW): Algoritmo per la divisione equa di oggetti tra due partecipanti. - Citazione: Brams, S. J., & Taylor, A. D. (1996). *Fair Division: From Cake-Cutting to Dispute Resolution*. Cambridge University Press.

- **Divide and Choose:** Metodo classico per la divisione equa tra due partecipanti.
 - Citazione: Steinhaus, H. (1948). *The problem of fair division*. *Econometrica*, 16(1), 101-104.
- **Last Diminisher:** Estensione del metodo Divide and Choose per più partecipanti.
 - Citazione: Steinhaus, H. (1948). *The problem of fair division*. *Econometrica*, 16(1), 101-104.

Metodi per la Divisione Equa tra Tre O Più Partecipanti - Algoritmo di Selfridge-Conway: Metodo per la divisione equa di una torta tra tre partecipanti. - Citazione: Robertson, J., & Webb, W. (1998). *Cake-cutting algorithms: Be fair if you can*. AK Peters/CRC Press.

- **Algoritmo di Dubins-Spanier:** Metodo per la divisione equa di una risorsa continua tra n partecipanti.
 - Citazione: Dubins, L. E., & Spanier, E. H. (1961). *How to cut a cake fairly*. *The American Mathematical Monthly*, 68(1), 1-17.

- **Algoritmo di Stromquist:** Metodo per la divisione envy-free tra tre partecipanti usando un numero finito di tagli.
 - Citazione: Stromquist, W. (1980). *How to cut a cake fairly*. The American Mathematical Monthly, 87(8), 640-644.
- **Algoritmo di Austin:** Metodo per la divisione envy-free tra quattro o più partecipanti.
 - Citazione: Austin, A. K. (1982). *Sharing a cake*. The Mathematical Gazette, 66(437), 212-215.
- **Algoritmo di Brams-Taylor-Zwicker:** Metodo per la divisione equa tra più di tre partecipanti.
 - Citazione: Brams, S. J., Taylor, A. D., & Zwicker, W. S. (1997). *A moving-knife solution to the four-person envy-free cake-division problem*. Proceedings of the American Mathematical Society, 125(2), 547-554.
- **Algoritmo di Brams-Taylor per n partecipanti:** Metodo per la divisione envy-free tra n partecipanti.
 - Citazione: Brams, S. J., & Taylor, A. D. (1995). *An envy-free cake division protocol*. The American Mathematical Monthly, 102(1), 9-18.

Metodi di Allocazione Proporzionale - Proportional Allocation: Metodi per allocare risorse in modo proporzionale alle richieste o ai diritti delle parti. - Citazione: Young, H. P. (1994). *Equity: In Theory and Practice*. Princeton University Press.

- **Maximin Share Allocation:** Concetto di equità per l'allocazione di beni indivisibili.
 - Citazione: Budish, E. (2011). *The combinatorial assignment problem: Approximate competitive equilibrium from equal incomes*. Journal of Political Economy, 119(6), 1061-1103.
- **Competitive Equilibrium from Equal Incomes (CEEI):** Meccanismo per l'allocazione equa di risorse divisibili.
 - Citazione: Varian, H. R. (1974). *Equity, envy, and efficiency*. Journal of Economic Theory, 9(1), 63-91.

Metodi di Assegnazione Casuale - Random Priority (RP): Meccanismo di assegnazione casuale utilizzato in vari contesti, come l'assegnazione di dormitori universitari. - Citazione: Abdulkadiroğlu, A., & Sönmez, T. (1998). *Random serial dictatorship and the core from random endowments in house allocation problems*. Econometrica, 66(3), 689-701.

- **Probabilistic Serial (PS):** Meccanismo di assegnazione con garanzie di efficienza ordinale.

- Citazione: Bogomolnaia, A., & Moulin, H. (2001). *A new solution to the random assignment problem*. Journal of Economic Theory, 100(2), 295-328.

Metodi Vari - Undercut Procedure: Metodo per la divisione equa di beni indivisibili. -

Citazione: Brams, S. J., & Taylor, A. D. (1999). *The Win-Win Solution: Guaranteeing Fair Shares to Everybody*. W. W. Norton & Company.

- **Picking Sequence Protocols:** Metodi di divisione basati su sequenze di scelte.

- Citazione: Bouveret, S., & Lang, J. (2011). *A general elicitation-free protocol for allocating indivisible goods*. In *Twenty-Second International Joint Conference on Artificial Intelligence*.

- **Algoritmo di Robertson-Webb:** Framework per misurare la complessità degli algoritmi di cake-cutting.

- Citazione: Robertson, J., & Webb, W. (1998). *Cake-cutting algorithms: Be fair if you can*. AK Peters/CRC Press.

Per chi vuole provare ad applicare gli algoritmi equitativi in casi reali su una piattaforma online si segnala l'interessante sito: [Spliddit Algorithms](#): Suite di algoritmi equitativi implementati sulla piattaforma web Spliddit (Goldman, J., & Procaccia, A. D. (2014). *Spliddit: Unleashing fair division algorithms*. ACM SIGecom Exchanges, 13(2), 41-46).

```
def adjusted_winner(items, alice_values, bob_values, max_iterations=1000):
    n = len(items)
    alice_items = []
    bob_items = []

    # Fase 1: Assegnazione iniziale
    for i in range(n):
        if alice_values[i] >= bob_values[i]:
            alice_items.append(items[i])
        else:
            bob_items.append(items[i])

    # Fase 2: Calcolo dei punteggi
    alice_score = sum(alice_values[items.index(item)] for item in alice_items)
    bob_score = sum(bob_values[items.index(item)] for item in bob_items)

    # Fase 3: Aggiustamento
    iterations = 0
    while abs(alice_score - bob_score) > 0.001 and iterations < max_iterations:
        if alice_score > bob_score:
```

```

        item_to_transfer = max(alice_items, key=lambda x: bob_values[items.index(x)] / \
                                alice_values[items.index(x)])
        alice_items.remove(item_to_transfer)
        bob_items.append(item_to_transfer)
        alice_score -= alice_values[items.index(item_to_transfer)]
        bob_score += bob_values[items.index(item_to_transfer)]
    else:
        item_to_transfer = max(bob_items, key=lambda x: alice_values[items.index(x)] / \
                                bob_values[items.index(x)])
        bob_items.remove(item_to_transfer)
        alice_items.append(item_to_transfer)
        bob_score -= bob_values[items.index(item_to_transfer)]
        alice_score += alice_values[items.index(item_to_transfer)]
    iterations += 1

    return alice_items, bob_items, alice_score, bob_score, iterations

# Esempio di utilizzo
items = ['A', 'B', 'C', 'D']
alice_values = [30, 25, 35, 10]
bob_values = [20, 30, 25, 25]

alice_final, bob_final, alice_final_score, bob_final_score, iterations_final = \
    adjusted_winner(items, alice_values, bob_values)
print("Iterazioni necessarie:", iterations_final)
print("Alice riceve:", alice_final)
print("Bob riceve:", bob_final)

```

```

Iterazioni necessarie: 1000
Alice riceve: ['A', 'C']
Bob riceve: ['B', 'D']

```

2.6 Algoritmi Predittivi

Gli algoritmi di predizione sono usati per creare modelli basati sull'apprendimento dei dati misurati o prodotti in un certo dominio applicativo al fine di fare previsioni su eventi futuri. Questi algoritmi possono essere classificati in diverse categorie in base al tipo di apprendimento, al tipo di output, e alle tecniche utilizzate:

Classificazione Basata sul Tipo di Apprendimento: 1. **Apprendimento Supervisionato:** Gli algoritmi di apprendimento supervisionato richiedono un set di dati etichettato per l'addestramento. Utilizzano queste etichette per apprendere una funzione che mappa gli input agli output desiderati. Esempi includono la regressione lineare, gli alberi decisionali e le reti neurali¹. 2. **Apprendimento Non Supervisionato:** Questi algoritmi scoprono pattern nascosti o strutture nei dati non etichettati. Tecniche comuni sono la clusterizzazione e la riduzione della dimensionalità¹. 3. **Apprendimento Semi-supervisionato e Rinforzato:** Combinano elementi dei primi due tipi, utilizzando un piccolo set di dati etichettati insieme a una grande quantità di dati non etichettati, o apprendendo attraverso il rinforzo da un ambiente¹.

Classificazione Basata sul Tipo di Output: 1. **Classificazione:** Quando l'output è una categoria, come "spam" o "non spam" in un filtro di posta elettronica, si parla di classificazione. Gli algoritmi di classificazione assegnano un'etichetta discreta a un'istanza di input¹. 2. **Regressione:** Se l'output è un valore continuo, come il prezzo di una casa, si utilizza la regressione. Gli algoritmi di regressione prevedono un valore numerico basato sugli input¹. 3. **Ranking:** Alcuni algoritmi ordinano gli elementi in base alla probabilità di appartenenza a una certa categoria o valore¹.

Classificazione Basata sulle Tecniche Utilizzate: 1. **Alberi Decisionali:** Suddividono i dati in modo gerarchico basandosi su attributi specifici. Sono semplici da interpretare ma possono soffrire di overfitting². 2. **Random Forest:** Una collezione di alberi decisionali che riduce il rischio di overfitting e gestisce meglio le variabili non correlate². 3. **Support Vector Machine (SVM):** Trovano il miglior iperpiano che separa i dati in classi. Sono efficaci in spazi ad alta dimensionalità². 4. **K-Nearest Neighbors (K-NN):** Classificano i nuovi dati in base alla classe più comune tra i vicini più prossimi. Sono semplici da implementare ma computazionalmente costosi². 5. **Reti Neurali:** Sono modelli ispirati al funzionamento del cervello umano e possono catturare relazioni complesse nei dati¹.

Ogni algoritmo ha i suoi vantaggi e svantaggi, e la scelta dipende da vari fattori come la dimensione e la natura del dataset, la velocità richiesta, la trasparenza del modello e la capacità di gestire dati non lineari o mancanti. Ad esempio, gli alberi decisionali sono facili da interpretare ma possono soffrire di overfitting, mentre le SVM sono efficaci con dataset di piccole dimensioni ma meno efficienti con dataset molto grandi².

L'agente deve imparare a riconoscere alcune configurazioni del suo percepito sulla base di un'esperienza fatta su casi detti di training e deve essere in grado di riconoscere un nuovo percepito mai visto prima.

Il percepito dell'agente è composto da dati, caratteristiche, che possono essere di tipo continuo (es. temperatura) o categoriali (es. colore rosso). I dati categoriali possono essere ordinabili (es. scarso, insufficiente, ...) o non ordinabili (es. sesso)

All'agente può essere chiesto di predire un dato continuo, nel qual caso si tratta di predizione o regressione, oppure può essere chiesto di predire un dato categoriale, nel qual caso si tratta di classificazione.

Il processo di predizione segue il seguente flusso:

1. Definizione del Problema

- Identificare il tipo di problema (classificazione, regressione, clustering)

2. Raccolta dei Dati

- Ottenere dati rilevanti e di qualità

3. Pre-elaborazione dei Dati

- Pulizia dei dati
- Gestione dei valori mancanti
- Normalizzazione
- Riduzione della dimensionalità

4. Divisione dei Dati

- Creare set di addestramento e di test

5. Scelta dell'Algoritmo

- Selezionare l'algoritmo adatto al problema

6. Addestramento del Modello

- Imparare dai dati di addestramento

7. Valutazione del Modello

- Testare il modello con il set di test

8. Ottimizzazione

- Regolare i parametri per migliorare le prestazioni

9. Deployment

- Implementare il modello in produzione

10. Monitoraggio e Manutenzione

- Aggiornare e ottimizzare il modello nel tempo

Nei prossimi paragrafi si presenterà una descrizione di ognuno di questi passi con un esempio concreto.

2.6.1 Definizione del Problema

Questo è il primo passo nel processo di predizione nell'intelligenza artificiale. Questa fase richiede una comprensione chiara e precisa dell'obiettivo che si desidera raggiungere con l'algoritmo di predizione. Che si tratti di prevedere il comportamento del consumatore, di diagnosticare malattie o di identificare frodi, è fondamentale stabilire parametri chiari e misurabili. La definizione del problema guida tutte le fasi successive, dalla raccolta dei dati alla scelta dell'algoritmo più adatto, assicurando che l'intero processo sia allineato con l'obiettivo finale. Un'accurata definizione del problema è la base per un modello predittivo efficace e funzionale.

esempio pratico:

Un esempio pratico di definizione del problema nel contesto del processo di predizione potrebbe essere il seguente scenario nel settore della vendita al dettaglio:

Scenario: Un'azienda di e-commerce ha notato un aumento del tasso di abbandono del carrello da parte dei clienti durante il processo di checkout.

Definizione del Problema: - **Obiettivo:** Ridurre il tasso di abbandono del carrello e aumentare il tasso di conversione delle vendite. - **Dati Necessari:** Dati di navigazione del sito web, transazioni completate e abbandonate, feedback dei clienti, dati demografici e comportamentali degli utenti. - **Ipotesi:** Si ipotizza che l'abbandono del carrello possa essere dovuto a fattori quali costi di spedizione inaspettati, complessità del processo di checkout, mancanza di opzioni di pagamento, o problemi tecnici del sito. - **Metodologia:** Utilizzare algoritmi di machine learning per analizzare i pattern di abbandono e identificare i punti critici nel processo di checkout. - **Soluzione Attesa:** Implementare miglioramenti mirati nel processo di checkout basati sui risultati dell'analisi predittiva, come la semplificazione delle procedure, l'aggiunta di più opzioni di pagamento, o la trasparenza dei costi di spedizione.

In questo esempio, la definizione del problema è ben strutturata e orientata all'obiettivo di migliorare l'esperienza dell'utente e ottimizzare il processo di vendita. L'analisi predittiva aiuterà l'azienda a comprendere le cause dell'abbandono del carrello e a formulare interventi efficaci per risolvere il problema.

2.6.2 Raccolta dei Dati

Si tratta di acquisire informazioni rilevanti per il problema da risolvere. Questo processo non si limita alla mera acquisizione di dati grezzi; è una pratica strategica che trasforma questi dati in insights preziosi, capaci di guidare decisioni informate. I dati possono essere raccolti da fonti interne come database aziendali, o esterne come social media, sensori IoT (Internet of Things), o registri pubblici. La raccolta deve essere sistematica e organizzata, assicurando che i dati siano accurati, completi e aggiornati. È fondamentale anche considerare la privacy e la sicurezza dei dati durante la raccolta e l'elaborazione.

Esempio Pratico: Un ospedale vuole sviluppare un sistema di predizione per identificare i pazienti a rischio di riammissione entro 30 giorni dalla dimissione. La raccolta dei dati inizia con l'identificazione delle informazioni necessarie, che includono dati demografici dei pazienti, diagnosi, trattamenti ricevuti, durata del soggiorno ospedaliero e dati storici sulle riammissioni. Questi dati vengono poi raccolti da cartelle cliniche elettroniche, registri ospedalieri e interviste con il personale sanitario. Una volta raccolti, i dati sono puliti e preparati per l'analisi, rimuovendo eventuali errori o duplicazioni e assicurando che siano in un formato utilizzabile per l'addestramento dell'algoritmo di predizione. Questo processo permette all'ospedale di costruire un modello predittivo che può aiutare a migliorare la qualità delle cure e ridurre i costi associati alle riammissioni non necessarie.

2.6.3 Pre-elaborazione dei Dati

Si prepara il dataset per garantire che l'algoritmo di machine learning funzioni in modo ottimale. Questo passo include diverse attività chiave:

- **Pulizia dei dati:** correzione o rimozione di dati errati, corrotti, duplicati o non pertinenti. La pulizia assicura che il modello non apprenda da informazioni fuorvianti o irrilevanti.
- **Gestione dei valori mancanti:** I dati incompleti sono comuni in molti dataset. La gestione dei valori mancanti può includere tecniche come l'imputazione, dove i valori mancanti sono sostituiti con stime, o l'eliminazione delle righe o colonne con dati mancanti.
- **Normalizzazione:** scalatura dei dati in modo che attributi con ampi intervalli di valori non dominino quelli con intervalli più stretti. La normalizzazione è essenziale per algoritmi che sono sensibili alle scale dei dati, come la regressione lineare o le reti neurali.
- **Riduzione della dimensionalità:** Tecniche come l'Analisi delle Componenti Principali (PCA) sono utilizzate per ridurre il numero di variabili nel dataset, mantenendo solo quelle più informative. Questo non solo semplifica il modello, ma può anche migliorare le prestazioni riducendo il rischio di overfitting.

Esempio Pratico: Immaginiamo di avere un dataset di immagini per un sistema di riconoscimento facciale. La pulizia dei dati potrebbe comportare la rimozione di immagini sfocate o non riconoscibili. Per gestire i valori mancanti, potremmo utilizzare tecniche di imputazione per completare le caratteristiche facciali parzialmente occluse. La normalizzazione potrebbe essere applicata per assicurare che le variazioni di luminosità siano tutte uguali in modo da non influenzino il riconoscimento. Infine, si procede a uniformare il formato file e le dimensioni. Ad esempio, formato file jpg e dimensioni 128x128 pixel.

2.6.4 Divisione dei Dati

Si separa il dataset in due o più gruppi per diversi scopi: addestramento, validazione e test. Questa divisione serve per avere dati per valutare l'efficacia e la generalizzabilità del modello predittivo. Il set di addestramento è utilizzato per insegnare all'algoritmo a riconoscere i pattern nei dati. Il set di validazione, quando presente, aiuta a ottimizzare i parametri del modello e a prevenire l'overfitting. Infine, il set di test serve a valutare le prestazioni del modello su dati non visti durante l'addestramento, fornendo una stima dell'errore di generalizzazione.

La proporzione della divisione può variare, ma una suddivisione comune è 70% per l'addestramento, 15% per la validazione e 15% per il test. È importante che la divisione dei dati sia rappresentativa dell'intero dataset, quindi tecniche come il campionamento stratificato possono essere utilizzate per mantenere la stessa distribuzione delle classi in ciascun set.

Esempio Pratico: Un'azienda di telecomunicazioni vuole prevedere quali clienti potrebbero lasciare l'azienda (churn). Il dataset include variabili come l'uso dei servizi, la durata del contratto, e la soddisfazione del cliente. Dopo la pre-elaborazione, il dataset di 1000 clienti viene diviso in 700 per l'addestramento, 150 per la validazione e 150 per il test. Questa divisione permette all'azienda di addestrare il modello sul set di addestramento, ottimizzarlo sul set di validazione e infine valutare la sua capacità di prevedere il churn sul set di test.

Il **training set**, il **validation set** e il **test set** sono tre sottoinsiemi di dati utilizzati nel processo di machine learning per sviluppare e valutare modelli predittivi. Ecco a cosa servono:

- **Training Set:** È il sottoinsieme di dati utilizzato per addestrare il modello. Il modello apprende a riconoscere i pattern e le relazioni tra i dati in modo che possa fare previsioni o prendere decisioni. Tipicamente, è il più grande dei tre set e fornisce la base su cui il modello costruisce la sua comprensione del problema.
- **Validation Set:** Viene utilizzato per fornire una valutazione imparziale delle prestazioni del modello durante la fase di addestramento. Questo set è cruciale per il tuning dei parametri del modello, come la scelta del numero di strati in una rete neurale o il valore di regolarizzazione in una regressione. Aiuta a prevenire l'overfitting, che si verifica quando un modello è troppo complesso e si adatta troppo bene ai dati di addestramento, perdendo la capacità di generalizzare su nuovi dati.
- **Test Set:** Dopo che il modello è stato addestrato e validato, il test set viene utilizzato per valutare le prestazioni finali del modello. Questo set non è mai stato visto dal modello durante l'addestramento e simula dati del mondo reale su cui il modello dovrà operare. Fornisce una misura oggettiva di quanto bene il modello possa aspettarsi di esibirsi in pratica.

Esempio Pratico: Immaginiamo di voler studiare il problema della previsione dei prezzi delle case basato su caratteristiche come la posizione, la dimensione e l'anno di costruzione. Il training set potrebbe includere migliaia di esempi di case vendute negli ultimi anni. Il validation set potrebbe essere usato per regolare i parametri del modello, come la complessità

del modello stesso. Infine, il test set, che consiste in dati recenti di vendite di case, verrebbe utilizzato per valutare quanto accuratamente il modello può prevedere i prezzi delle case in condizioni attuali di mercato.

2.6.5 Scelta dell' algoritmo

La scelta dell'algoritmo determina l'approccio con cui il modello analizzerà i dati e farà previsioni. La selezione dell'algoritmo dipende da vari fattori, tra cui il tipo di problema (classificazione, regressione, clustering), la natura dei dati, la dimensione del dataset e le risorse computazionali disponibili. Ad esempio, per problemi di classificazione, algoritmi come le reti neurali, le macchine a vettori di supporto (SVM) e gli alberi decisionali sono spesso utilizzati. Per la regressione, si possono considerare algoritmi come la regressione lineare, la regressione polinomiale o le reti neurali. È importante anche considerare la complessità dell'algoritmo: algoritmi più complessi possono offrire maggiore accuratezza, ma richiedono più tempo e risorse per l'addestramento.

Esempio Pratico: Nell' esempio della predizione del prezzo delle case introdotto nel precedente paragrafo, se si dispone di un dataset con caratteristiche come la dimensione della casa, il numero di stanze, la posizione, ecc., si potrebbe scegliere un algoritmo di regressione lineare per iniziare, poiché è semplice e offre una buona interpretabilità. Tuttavia, se i dati mostrano relazioni non lineari, si potrebbe passare a un algoritmo più complesso come una rete neurale per migliorare la precisione delle previsioni.

2.6.6 Addestramento del modello

L'addestramento di un modello di machine learning è un processo iterativo che consiste nell'esporre un algoritmo a un ampio dataset di apprendimento, con l'obiettivo di insegnargli a riconoscere pattern e a fare predizioni accurate su nuovi dati. La qualità e l'efficacia di un modello dipendono fortemente dalla scelta dell'algoritmo, dalla qualità dei dati e dalle tecniche di addestramento utilizzate.

Modalità di apprendimento Esistono diverse modalità di apprendimento:

- **Apprendimento supervisionato:** Il modello viene addestrato su un dataset etichettato, dove ogni esempio è associato a una risposta corretta. L'obiettivo è insegnare al modello a mappare nuovi input alle loro rispettive etichette.
- **Apprendimento non supervisionato:** Il modello lavora con dati non etichettati, cercando di scoprire strutture nascoste nei dati, come gruppi di dati simili (clustering).
- **Apprendimento semi-supervisionato:** Combina elementi dei due approcci precedenti, utilizzando sia dati etichettati che non etichettati.

Hardware e software

L'addestramento di modelli complessi richiede risorse computazionali significative. Le GPU e le TPU sono hardware specializzati che accelerano i calcoli necessari per l'addestramento di reti neurali profonde. Inoltre, sono necessari software specifici per definire le architetture delle reti neurali e gestire il processo di addestramento.

Overfitting e underfitting Durante l'addestramento, è fondamentale evitare due problemi comuni: l'overfitting e l'underfitting.

Overfitting: Si verifica quando il modello si adatta troppo ai dati di addestramento, perdendo la capacità di generalizzare a nuovi dati. In questo caso, il modello memorizza i dettagli specifici dei dati di addestramento invece di apprendere le caratteristiche generali. **Underfitting:** Si verifica quando il modello è troppo semplice per catturare la complessità dei dati. In questo caso, il modello non è in grado di apprendere le relazioni significative tra le variabili.

Implicazioni etiche

L'addestramento di modelli di machine learning solleva importanti questioni etiche. È fondamentale utilizzare dataset rappresentativi e bilanciati per evitare bias e discriminazioni. Inoltre, è necessario considerare le potenziali conseguenze negative dell'utilizzo di modelli in contesti reali, come la privacy e la sicurezza dei dati.

2.6.7 Valutazione del modello

La valutazione del modello è una fase cruciale nel processo di sviluppo di un modello di machine learning, poiché consente di misurare la capacità del modello addestrato di generalizzare a nuovi dati, ovvero di fare predizioni accurate su esempi che non ha mai visto durante l'addestramento.

Metriche di valutazione

Le metriche di valutazione variano a seconda del tipo di problema. Nei **problemi di classificazione**, ad esempio, si utilizzano metriche come:

- **Accuratezza:** misura la percentuale di casi classificati correttamente.
- **Precisione:** misura la percentuale di veri positivi (cioè quei casi che effettivamente appartengono alla classe positiva) tra i casi classificati come positivi.
- **Richiamo:** misura la percentuale di veri positivi tra tutti i casi positivi reali.
- **F1-score:** misura la media armonica tra precisione e richiamo.
- **Curva ROC** (Receiver Operating Characteristic) con l'area sotto la curva (AUC): misura la capacità del modello di distinguere correttamente tra le classi.

Nei **problemi di regressione**, invece, si utilizzano metriche come:

- **Errore quadratico medio (MSE):** misura la differenza media quadratica tra i valori predetti dal modello e i valori reali.
- **Coefficiente di determinazione (R^2):** misura la percentuale di variazione dei valori predetti rispetto ai valori reali.

Spero che queste correzioni siano utili! Hai altre domande o c'è qualcos'altro su cui posso aiutarti? **Analisi dei risultati**

L'analisi dei risultati della valutazione permette di identificare eventuali problemi come l'overfitting o l'underfitting. L'overfitting si verifica quando il modello si adatta troppo ai dati di addestramento, perdendo la capacità di generalizzare a nuovi dati. L'underfitting si verifica quando il modello è troppo semplice e non riesce a catturare la complessità dei dati.

Esempio pratico: Consideriamo un modello di machine learning addestrato per predire la probabilità di ricidività di un reo in base ai dati raccolti su di lui. Dopo l'addestramento, il modello viene valutato su un dataset di test che contiene informazioni su nuovi reati. Utilizzando la Cross-validation, calcolando metriche come l'accuratezza e il richiamo, possiamo valutare l'affidabilità delle predizioni del modello. Un'alta accuratezza indica che il modello è generalmente corretto nelle sue previsioni, mentre un alto richiamo indica che il modello è bravo a identificare i reati che effettivamente si sono verificati.

2.6.8 Ottimizzazione degli iperparametri

L'**ottimizzazione degli iperparametri** è un processo iterativo che consiste nel regolare i parametri esterni al modello che non vengono appresi durante l'addestramento, ma che influenzano significativamente le sue prestazioni. Esempi di iperparametri includono il tasso di apprendimento, la profondità di un albero decisionale o il numero di neuroni in una rete neurale.

L'obiettivo dell'ottimizzazione è individuare la combinazione di iperparametri che massimizza le prestazioni del modello su un dataset di valutazione indipendente. Per raggiungere questo obiettivo, si utilizzano diverse tecniche, tra cui:

- **Ricerca a griglia:** Esplora sistematicamente tutte le possibili combinazioni di iperparametri all'interno di un intervallo specificato.
- **Ricerca casuale:** Seleziona casualmente combinazioni di iperparametri, potendo essere più efficiente della ricerca a griglia in spazi di ricerca ampi.
- **Ottimizzazione bayesiana:** Utilizza modelli probabilistici per guidare la ricerca verso le regioni dello spazio degli iperparametri più promettenti.

Per valutare l'efficacia delle diverse combinazioni di iperparametri, si ricorre alla **validazione incrociata**. Questa tecnica consiste nel suddividere il dataset in più parti, addestrando il modello su una parte e valutandolo sulle altre. Ripetendo questo processo multiple volte, si ottiene una stima più robusta delle prestazioni del modello.

Tecniche come l'**early stopping** possono essere utilizzate per migliorare ulteriormente il processo di ottimizzazione. L'early stopping consiste nell'interrompere l'addestramento quando le prestazioni del modello sul dataset di convalida iniziano a peggiorare, evitando così l'overfitting.

Esempio pratico: Consideriamo un modello di rete neurale convoluzionale (CNN) addestrato per classificare immagini di cani e gatti. Per ottimizzare le prestazioni del modello, potremmo variare i seguenti iperparametri: il numero di strati convoluzionali, il numero di filtri per strato, il tasso di apprendimento e la funzione di attivazione. Utilizzando la ricerca a griglia e la validazione incrociata, possiamo identificare la combinazione di iperparametri che porta alle migliori prestazioni. Inoltre, possiamo utilizzare l'early stopping per evitare di addestrare eccessivamente il modello.

2.6.9 Deployment

Il deployment è la fase finale del processo di predizione, in cui il modello addestrato e ottimizzato viene messo in produzione per essere utilizzato in applicazioni reali. Questa fase implica la preparazione del modello per l'integrazione con sistemi esistenti, garantendo che sia scalabile, affidabile e sicuro. Il deployment può avvenire su diverse piattaforme, come server locali, cloud o dispositivi edge (dispositivi periferici che elaborano i dati vicino alla fonte, riducendo la latenza e il carico sui server centrali), a seconda delle esigenze dell'applicazione. È importante notare che l'hardware necessario per il deployment è diverso da quello utilizzato per l'addestramento. Durante l'addestramento, sono necessarie risorse computazionali elevate, come GPU o TPU, per gestire i complessi calcoli e l'ottimizzazione dei parametri del modello. Tuttavia, una volta che il modello è addestrato, il deployment richiede meno potenza computazionale, poiché il modello deve solo eseguire previsioni basate sui dati in ingresso. Questo permette di utilizzare hardware meno potente e più economico per il deployment, riducendo i costi operativi.

Sfide del deployment:

- * **Scalabilità:** Il sistema di deployment deve essere in grado di gestire un aumento del carico di lavoro e di scalare in modo elastico per soddisfare le esigenze dell'applicazione.
- * **Affidabilità:** Il modello deve essere disponibile e funzionante in modo continuo, minimizzando i tempi di fermo e garantendo la qualità delle previsioni.
- * **Sicurezza:** È fondamentale proteggere il modello e i dati sensibili da accessi non autorizzati e attacchi informatici.

esempio pratico: Un esempio pratico di deployment è l'implementazione di un modello di raccomandazione per un sito di e-commerce. Supponiamo di avere un modello che suggerisce prodotti agli utenti basato sul loro comportamento di navigazione e acquisto. Dopo aver addestrato e ottimizzato il modello utilizzando hardware potente come GPU, il passo successivo è integrarlo con il sistema del sito web. Questo può comportare la creazione di API (Application Programming Interfaces, interfacce che permettono a diverse applicazioni di comunicare tra loro) che permettono al sito di inviare dati al modello e ricevere raccomandazioni in tempo

reale. Una volta implementato, il modello può analizzare i dati degli utenti e fornire suggerimenti personalizzati, migliorando l'esperienza dell'utente e potenzialmente aumentando le vendite. Per garantire prestazioni elevate, il modello può essere deployato su un server cloud scalabile, come **AWS** (Amazon Web Services) o **GCP** (Google Cloud Platform). Monitorando le prestazioni del modello, è possibile fare aggiustamenti e aggiornamenti per mantenere alta la qualità delle raccomandazioni.

2.6.10 Monitoraggio e manutenzione

Il monitoraggio e la manutenzione sono attività cruciali nel ciclo di vita di un modello di predizione, volte a garantire che il modello rimanga accurato e affidabile nel tempo. Dopo il deployment, è essenziale monitorare continuamente le prestazioni del modello per rilevare eventuali degradi dovuti a cambiamenti nei dati o nel contesto operativo. Questo può includere il monitoraggio di indicatori di degrado come:

1. **Aumento dell'errore di previsione:** Se il modello inizia a fare più errori nelle previsioni rispetto a prima, potrebbe essere un segnale di degrado. Questo può essere misurato attraverso metriche come l'errore quadratico medio (MSE) per i modelli di regressione o l'accuratezza per i modelli di classificazione.
2. **Diminuzione dell'accuratezza:** Un calo nell'accuratezza complessiva del modello indica che le previsioni non sono più affidabili come in passato.
3. **Aumento dei falsi positivi/negativi:** Per i modelli di classificazione, un aumento dei falsi positivi (previsioni errate di eventi positivi) o dei falsi negativi (previsioni errate di eventi negativi) può indicare che il modello non sta più funzionando correttamente.
4. **Cambiamenti nelle distribuzioni dei dati:** Se i dati in ingresso cambiano significativamente rispetto ai dati su cui il modello è stato addestrato, il modello potrebbe non essere più in grado di generalizzare correttamente. Questo può essere monitorato attraverso tecniche di drift detection.
5. **Aumento del tempo di risposta:** Se il modello impiega più tempo per fare previsioni, potrebbe essere un segnale che qualcosa non va, come un sovraccarico computazionale o inefficienze nel codice.

La manutenzione del modello può comportare aggiornamenti periodici, riaddestramento con nuovi dati e ottimizzazioni per adattarsi a nuove condizioni. Inoltre, è importante implementare sistemi di allerta per notificare tempestivamente eventuali problemi. La manutenzione preventiva e correttiva aiuta a mantenere il modello efficiente e a evitare errori significativi che potrebbero influenzare le decisioni basate sulle sue previsioni.

Esempio pratico: Un esempio pratico di monitoraggio e manutenzione è l'uso di un modello di rilevamento delle frodi in una banca. Dopo il deployment, il modello analizza le transazioni in tempo reale per identificare attività sospette. Il team di data science monitora costantemente

le prestazioni del modello, verificando che mantenga un alto tasso di rilevamento delle frodi e un basso tasso di falsi positivi. Se il modello inizia a mostrare segni di degrado, come un aumento dei falsi positivi, il team può riaddestrarlo utilizzando dati più recenti o aggiustare i parametri per migliorare la sua accuratezza. Questo processo continuo garantisce che il modello rimanga efficace nel proteggere la banca dalle frodi.

Esempio di algoritmo predittivo in Python:

```
from sklearn.linear_model import LogisticRegression
import numpy as np

# Dati storici dei casi (caratteristiche e risultati)
X = np.array([[25, 1], [45, 0], [35, 1], [50, 0]]) # Et  e tipo di crimine
y = np.array([1, 0, 1, 0]) # Risultato del caso (1 = colpevole, 0 = innocente)

# Addestramento del modello predittivo
model = LogisticRegression()
model.fit(X, y)

# Previsione di un nuovo caso
new_case = np.array([[30, 1]]) # Nuovo caso: 30 anni, tipo di crimine 1
prediction = model.predict(new_case)
print(f"Previsione del nuovo caso: {'colpevole' if prediction[0] == 1 else 'innocente'}.")
```

In questo capitolo, abbiamo esplorato vari algoritmi utilizzati nell'intelligenza artificiale, con esempi pratici applicati alla giurisprudenza. Questi algoritmi sono strumenti potenti che possono aiutare a migliorare l'efficienza e l'equit  nel contesto legale.

3 Machine learning

Benvenuti al Capitolo 4 del nostro “Laboratorio di Intelligenza Artificiale”, dedicato al Machine Learning. In questo capitolo, esploreremo come l’intelligenza artificiale può “imparare” dai dati, un concetto che sta rivoluzionando numerosi campi, incluso quello giuridico. Per voi, futuri professionisti del diritto, comprendere il Machine Learning non è solo un esercizio accademico, ma una necessità pratica. Le tecnologie basate sul ML stanno già influenzando il settore legale, dalla ricerca giurisprudenziale automatizzata alla previsione degli esiti dei processi. La vostra capacità di navigare tra queste innovazioni sarà cruciale per la vostra carriera.

Inizieremo il nostro viaggio esplorando l’apprendimento supervisionato, dove gli algoritmi imparano da esempi etichettati. Vedremo come questo approccio possa essere applicato per classificare documenti legali o prevedere sentenze basandosi su casi precedenti. Ci addenteremo anche nell’affascinante mondo dell’apprendimento per rinforzo, particolarmente rilevante per la modellazione di strategie decisionali in contesti giuridici complessi.

Passeremo poi all’apprendimento non supervisionato, esplorando come gli algoritmi possano scoprire pattern nascosti in grandi volumi di dati legali non etichettati. Questa tecnica potrebbe rivoluzionare il modo in cui analizziamo la giurisprudenza e identifichiamo tendenze legali emergenti.

Un tema cruciale che affronteremo è quello dei bias negli algoritmi di ML. Come futuri giuristi, dovrete essere consapevoli di come i pregiudizi possano infiltrarsi nei sistemi automatizzati e delle implicazioni etiche e legali che ne derivano.

Infine, ci immergeremo nel mondo delle reti neurali e del deep learning, con un focus speciale sull’applicazione ai linguaggi naturali. Esploreremo i Large Language Models (LLM) come GPT e BERT, discutendo il loro potenziale rivoluzionario nel campo legale, dalla redazione di documenti all’assistenza nella ricerca legale.

Attraverso questo capitolo, non solo acquisirete una solida comprensione teorica del Machine Learning, ma imparerete anche ad applicare questi concetti utilizzando Python. Ogni sezione includerà esempi pratici e esercizi mirati al contesto legale, permettendovi di sperimentare direttamente con queste tecnologie.

Il nostro obiettivo è prepararvi ad essere non solo consumatori informati di tecnologie basate sul ML, ma anche potenziali innovatori nel vostro campo. Comprendere questi strumenti vi permetterà di anticipare come l’IA potrebbe influenzare la pratica legale futura e di contribuire attivamente a plasmare questo futuro.

Prepariamoci dunque a esplorare questo affascinante campo, dove la legge incontra l'intelligenza artificiale. Le competenze che acquisirete in questo capitolo non solo arricchiranno il vostro bagaglio tecnico, ma vi doteranno di una prospettiva unica e preziosa nel panorama giuridico in rapida evoluzione. ione di contenuti.

3.1 Apprendimento Supervisionato

3.1.1 Definizione e Principi di Base

L'apprendimento supervisionato è un sottoinsieme del machine learning che si occupa di costruire modelli predittivi utilizzando un dataset etichettato, dove ogni esempio di input è associato a un output corrispondente (l'etichetta). Il processo di apprendimento supervisionato può essere visto come una forma di mappatura funzionale ($f: X \rightarrow Y$), dove (X) rappresenta lo spazio degli input (caratteristiche o feature) e (Y) rappresenta lo spazio degli output (etichette). L'obiettivo principale è imparare una funzione (f) che, dato un nuovo input, sia in grado di predire l'output corretto.

Il processo di addestramento coinvolge due fasi principali: l'**apprendimento** e la **generalizzazione**. Durante la fase di apprendimento, il modello viene addestrato su un insieme di dati di addestramento, cercando di minimizzare la funzione di perdita, che misura la discrepanza tra le previsioni del modello e le etichette effettive. Successivamente, nella fase di generalizzazione, il modello viene testato su nuovi dati non visti per valutare la sua capacità di fare previsioni accurate al di fuori del set di addestramento.

I principi fondamentali che guidano l'apprendimento supervisionato includono la **funzione di perdita**, che determina quanto una previsione è lontana dal valore vero; l'**ottimizzazione**, che è il processo attraverso il quale il modello migliora le sue previsioni iterativamente; e il **bias-variance tradeoff**, che è il bilanciamento tra un modello troppo semplice (alto bias) e uno troppo complesso (alta varianza).

3.1.2 Classificazione

La classificazione è una tecnica di apprendimento supervisionato in cui l'obiettivo è assegnare una classe o etichetta specifica a un input, in base a un insieme di dati di addestramento. Esistono vari tipi di problemi di classificazione:

- **Classificazione binaria:** Qui, l'output è limitato a due classi, come “sì” o “no”, “spam” o “non spam”. Questo tipo di problema è comune in scenari come la diagnosi medica (es. malato o non malato) e nella sicurezza informatica (es. email sicura o phishing).

- **Classificazione multiclasse:** In questo caso, l'output può appartenere a una di più classi (es. classificare un documento come “legale”, “finanziario” o “scientifico”). Le tecniche utilizzate possono includere approcci come la **regressione logistica multinomiale**, le **reti neurali** e le **macchine a supporto di vettori (SVM)**.
- **Classificazione multilabel:** Qui, un singolo input può essere associato a più classi contemporaneamente (es. un articolo di giornale che potrebbe essere classificato sia come “politico” che come “economico”). Tecniche come l'**approccio One-vs-All** e le reti neurali sono frequentemente utilizzate in questi contesti.

Un punto di interesse particolare nella classificazione è il concetto di **boundary decisionale**. Questo rappresenta il confine nello spazio delle caratteristiche che separa le diverse classi. Nei modelli lineari, questo confine è una linea retta o un iperpiano, mentre nei modelli non lineari può assumere forme molto più complesse.

3.1.3 Regressione

La regressione è un tipo di problema di apprendimento supervisionato, focalizzato sulla previsione di un valore continuo piuttosto che su una classe discreta. A differenza della classificazione, dove l'output è un'etichetta, nella regressione l'output è un valore numerico che può variare su un intervallo continuo.

- **Regressione lineare:** Il modello di regressione lineare è uno dei più semplici e intuitivi. Esso assume che esista una relazione lineare tra le caratteristiche dell'input e l'output. La formula generale per la regressione lineare semplice è

$$y = \beta_0 + \beta_1 x,$$

dove β_0 è l'intercetta con l'asse delle ordinate e β_1 è la pendenza della retta.

- **Regressione polinomiale:** Quando la relazione tra le variabili non è lineare, si può ricorrere alla regressione polinomiale, che permette di modellare relazioni più complesse includendo termini polinomiali delle variabili indipendenti.
- **Regressione multivariata:** Questo tipo di regressione viene utilizzato quando si desidera prevedere l'output in base a più variabili indipendenti. È un'estensione naturale della regressione lineare e polinomiale.

La **regolarizzazione** (es. Ridge e Lasso) è una tecnica comune utilizzata nella regressione per prevenire l'overfitting, imponendo una penalità alla complessità del modello, limitando così i valori dei coefficienti di regressione.

3.1.4 Algoritmi Principali dell'Apprendimento Supervisionato

L'apprendimento supervisionato si avvale di una vasta gamma di algoritmi che possono essere utilizzati per risolvere problemi sia di classificazione che di regressione. Ogni algoritmo ha caratteristiche specifiche che lo rendono più o meno adatto a particolari tipi di dati e problemi. Di seguito, verranno presentati alcuni dei principali algoritmi utilizzati nell'apprendimento supervisionato.

3.1.4.1 Regressione Lineare

La regressione lineare è uno degli algoritmi più semplici e ampiamente utilizzati per problemi di regressione. Assume una relazione lineare tra le variabili indipendenti e la variabile dipendente e cerca di trovare la retta (o l'iperpiano nel caso di più variabili indipendenti) che meglio approssima i dati. La semplicità della regressione lineare la rende facile da interpretare, ma la sua capacità di modellare solo relazioni lineari può limitare la sua applicabilità in scenari più complessi.

3.1.4.2 Regressione Logistica

La regressione logistica è un algoritmo di classificazione che viene utilizzato quando l'output è binario. A differenza della regressione lineare, la regressione logistica utilizza una funzione logistica (o sigmoide) per modellare la probabilità che un dato appartenga a una classe specifica. Questo approccio è ampiamente utilizzato per problemi come la classificazione di e-mail come "spam" o "non spam" o per la predizione di eventi binari (es. successo o fallimento di un'azione legale).

3.1.4.3 Alberi di Decisione

Gli alberi di decisione sono modelli non parametrici che possono essere utilizzati sia per la classificazione che per la regressione. Essi segmentano il dataset in sottogruppi omogenei attraverso una serie di decisioni basate sui valori delle caratteristiche. Ogni nodo dell'albero rappresenta una decisione basata su una caratteristica, e i rami rappresentano le possibili conseguenze di tale decisione. Gli alberi di decisione sono facili da interpretare e visualizzare, il che li rende particolarmente utili quando è necessaria una comprensione trasparente del processo decisionale. Tuttavia, gli alberi di decisione possono essere inclini all'overfitting, specialmente se non adeguatamente potati.

3.1.4.4 Random Forest

Il Random Forest è un metodo ensemble basato su alberi di decisione. Consiste in un insieme di alberi di decisione indipendenti addestrati su diverse porzioni del dataset (attraverso il bootstrapping) e utilizzando un sottoinsieme casuale di caratteristiche. Il risultato finale è ottenuto aggregando le previsioni di tutti gli alberi (es. tramite voto di maggioranza per la classificazione o media per la regressione). Questa tecnica riduce significativamente il rischio di overfitting rispetto a un singolo albero di decisione e migliora la precisione e la robustezza del modello.

3.1.4.5 Support Vector Machines (SVM)

Le Support Vector Machines (SVM) sono algoritmi molto potenti sia per la classificazione che per la regressione. Il loro obiettivo è trovare un iperpiano ottimale che separi i dati di diverse classi con il massimo margine possibile. Le SVM sono particolarmente efficaci in spazi ad alta dimensionalità e possono essere estese per gestire separazioni non lineari utilizzando il **kernel trick**, che permette di mappare i dati in uno spazio di dimensione superiore dove la separazione diventa lineare.

3.1.4.6 k-Nearest Neighbors (k-NN)

Il k-Nearest Neighbors (k-NN) è un algoritmo di classificazione e regressione basato su un'idea semplice ma efficace: per predire l'etichetta di un nuovo dato, si cercano i k punti più vicini nel dataset di addestramento e si assegna al nuovo dato la classe maggioritaria (nel caso di classificazione) o la media dei valori (nel caso di regressione). Il k-NN è molto intuitivo e non richiede una fase di addestramento, ma può diventare inefficiente con dataset molto grandi o in presenza di rumore.

3.1.4.7 Reti Neurali

Le reti neurali sono modelli ispirati al funzionamento del cervello umano e sono particolarmente potenti per la modellazione di relazioni non lineari complesse. Una rete neurale è composta da strati di nodi (neuroni) interconnessi, dove ciascun nodo applica una funzione non lineare ai dati in ingresso e trasmette il risultato ai nodi dello strato successivo. Le reti neurali possono essere utilizzate sia per la classificazione che per la regressione e sono alla base di tecniche avanzate come il **deep learning**.

- **Percettrone Multistrato (MLP)**: È una delle architetture più semplici di reti neurali, composto da uno o più strati nascosti tra l'input e l'output. Il MLP è capace di apprendere rappresentazioni complesse dei dati, ma richiede un'attenta configurazione dei parametri e una grande quantità di dati per addestramento.

- **Reti Neurali Convoluzionali (CNN):** Utilizzate principalmente per l'elaborazione di immagini, le CNN applicano convoluzioni ai dati in ingresso per estrarre automaticamente caratteristiche di alto livello. Sono particolarmente efficaci in problemi di riconoscimento di immagini e visione artificiale.
- **Reti Neurali Ricorrenti (RNN):** Progettate per gestire dati sequenziali come testi o serie temporali, le RNN hanno connessioni che permettono l'uso di informazioni provenienti da precedenti stati dell'input. Questo le rende ideali per problemi come la modellazione del linguaggio naturale o la previsione di sequenze.

3.1.4.8 Gradient Boosting Machines (GBM)

Il Gradient Boosting è una tecnica di ensemble che costruisce modelli in modo sequenziale, dove ogni nuovo modello cerca di correggere gli errori commessi dai modelli precedenti. I modelli individuali sono generalmente alberi di decisione semplici (stump), e il risultato finale è una somma ponderata di questi alberi. Algoritmi popolari come **XGBoost** e **LightGBM** sono varianti ottimizzate del Gradient Boosting, note per la loro efficacia e velocità, specialmente in competizioni di machine learning.

3.1.4.9 Naive Bayes

Il Naive Bayes è un algoritmo di classificazione basato sul teorema di Bayes, con l'assunzione "naive" che le caratteristiche siano indipendenti l'una dall'altra, una ipotesi raramente vera nel mondo reale. Nonostante questa assunzione, il Naive Bayes è sorprendentemente efficace, specialmente per problemi di classificazione testuale come la categorizzazione di documenti o l'analisi del sentiment.

3.1.4.10 Ensemble Learning

L'**ensemble learning** combina le previsioni di più modelli per ottenere un risultato finale più robusto e accurato. Oltre al Random Forest e al Gradient Boosting, altre tecniche di ensemble includono il **bagging** e lo **stacking**. Il bagging riduce la varianza addestrando lo stesso modello su diverse porzioni del dataset, mentre lo stacking combina le previsioni di diversi modelli tramite un meta-modello, che apprende a pesare le diverse previsioni.

3.1.4.11 Conclusioni

Ciascuno degli algoritmi discussi ha punti di forza e di debolezza che lo rendono più o meno adatto a particolari problemi di apprendimento supervisionato. La scelta dell'algoritmo più appropriato dipende dalla natura del problema, dalla quantità e qualità dei dati disponibili e dalle

specifiche esigenze dell'applicazione. In contesti giuridici, dove la trasparenza e l'interpretabile sono spesso fondamentali, gli algoritmi semplici e interpretabili come gli alberi di decisione o la regressione logistica potrebbero essere preferibili, mentre in applicazioni più complesse come l'analisi di grandi volumi di dati testuali, algoritmi più sofisticati come le reti neurali o le tecniche di ensemble possono offrire prestazioni superiori.

3.1.5 Apprendimento per Rinforzo

3.1.5.1 Concetti Base

L'apprendimento per rinforzo (Reinforcement Learning, RL) si distingue dagli altri tipi di apprendimento supervisionato in quanto l'agente apprende attraverso l'interazione diretta con l'ambiente, senza avere accesso diretto a una serie di etichette corrette per ogni azione. In RL, l'agente prende decisioni sequenziali e riceve ricompense (o punizioni) che riflettono l'efficacia delle sue azioni. Il compito dell'agente è quindi quello di imparare una politica, o strategia, che massimizza la ricompensa totale nel tempo.

3.1.5.2 Agenti, Ambiente e Politiche

Gli elementi chiave nell'apprendimento per rinforzo includono:

- **Agente:** L'entità che prende decisioni nell'ambiente.
- **Ambiente:** Il contesto in cui l'agente opera e da cui riceve feedback sotto forma di ricompense.
- **Politica (Policy):** La strategia che l'agente segue per determinare quali azioni intraprendere in ogni stato.
- **Funzione di valore (Value Function):** Una funzione che valuta l'utilità di essere in un certo stato, dato un insieme di azioni future possibili.
- **Funzione di ricompensa (Reward Function):** Una funzione che fornisce un feedback immediato sulle azioni dell'agente.

3.1.5.3 Algoritmi Principali (es. Q-Learning, Deep Q-Networks)

- **Q-Learning:** È uno degli algoritmi di apprendimento per rinforzo più semplici e più conosciuti. Q-Learning si basa sull'apprendimento della funzione Q, che stima la qualità (o valore) di un'azione in un dato stato. L'agente utilizza questa funzione per decidere quali azioni intraprendere al fine di massimizzare la ricompensa cumulativa. Q-Learning è un algoritmo **off-policy**, il che significa che l'agente può apprendere la politica ottimale indipendentemente dalla politica attualmente seguita.

- **Deep Q-Networks (DQN):** Estende Q-Learning utilizzando reti neurali profonde per approssimare la funzione Q, consentendo così di gestire ambienti con spazi di stato molto grandi o continui. Questo approccio è stato utilizzato con successo in diversi contesti, tra cui il superamento delle prestazioni umane in giochi complessi come Atari.

3.1.5.4 Applicazioni

L'apprendimento per rinforzo è utilizzato in un'ampia varietà di applicazioni, che vanno dai giochi (es. scacchi, Go, e videogiochi come quelli sviluppati da OpenAI e DeepMind) alla robotica (es. robot che imparano a camminare o manipolare oggetti), fino a scenari come la guida autonoma. Nell'ambito giuridico, potrebbe essere applicato per ottimizzare flussi di lavoro complessi, simulare scenari di negoziazione o migliorare i processi decisionali attraverso simulazioni avanzate.

3.1.6 Overfitting e Underfitting

L'overfitting e l'underfitting sono due delle principali problematiche che emergono nell'apprendimento supervisionato e possono influenzare significativamente la capacità di un modello di generalizzare su nuovi dati.

- **Overfitting:** Si verifica quando un modello diventa troppo complesso, catturando non solo i pattern rilevanti nei dati di addestramento ma anche il rumore. Un modello overfit avrà prestazioni eccellenti sui dati di addestramento ma scarse prestazioni su dati nuovi e non visti. Questo problema può essere mitigato attraverso tecniche come la **regolarizzazione** (es. Lasso, Ridge), l'**early stopping** (interrompere l'addestramento prima che il modello inizi a memorizzare il rumore), e l'**utilizzo di più dati** o di **modelli più semplici**.
- **Underfitting:** Si verifica quando un modello è troppo semplice per rappresentare adeguatamente i dati. Un modello underfit avrà scarse prestazioni sia sui dati di addestramento che sui dati di test, poiché non riesce a catturare i pattern sottostanti. Per evitare l'underfitting, è necessario aumentare la complessità del modello o migliorare la qualità dei dati.

L'obiettivo nella costruzione di un modello è trovare il giusto equilibrio tra bias e varianza, in modo da ottenere un modello che sia abbastanza complesso da catturare i pattern rilevanti nei dati senza diventare così complesso da catturare anche il rumore.

3.1.7 Valutazione dei Modelli

La valutazione dei modelli è un passo critico per garantire che un modello di apprendimento supervisionato sia non solo accurato ma anche robusto e generalizzabile a dati non visti. La scelta delle metriche di valutazione dipende dal tipo di problema (classificazione o regressione) e dalle specifiche esigenze dell'applicazione.

- **Valutazione nei problemi di classificazione:**

- **Accuratezza:** È la misura più semplice e rappresenta la proporzione di previsioni corrette sul totale delle previsioni. Tuttavia, in presenza di classi sbilanciate, l'accuratezza può essere ingannevole, poiché un modello che predice sempre la classe maggioritaria avrà un'accuratezza elevata anche se non è utile.
- **Precisione e Recall:** La precisione misura la proporzione di veri positivi rispetto al totale delle predizioni positive, mentre il recall misura la proporzione di veri positivi rispetto al totale dei veri positivi più i falsi negativi. Queste metriche sono particolarmente importanti quando ci sono costi associati ai falsi positivi o ai falsi negativi.
- **F1-Score:** È la media armonica tra precisione e recall e fornisce un singolo valore che rappresenta un buon compromesso tra queste due metriche. L'F1-score è utile in contesti dove è necessario bilanciare precisione e recall.
- **AUC-ROC:** La curva ROC (Receiver Operating Characteristic) e l'area sotto la curva ROC (AUC) sono strumenti grafici che rappresentano la capacità di un classificatore di distinguere tra le classi. AUC fornisce un valore che varia da 0 a 1, dove 1 rappresenta un classificatore perfetto e 0.5 rappresenta un classificatore casuale.
- **Matrice di Confusione:** Questa tabella riassume i veri positivi, falsi positivi, veri negativi e falsi negativi, offrendo una visione più dettagliata delle prestazioni del modello. È particolarmente utile per identificare se un modello ha bias specifici in determinate classi.

- **Valutazione nei problemi di regressione:**

- **Errore Quadratico Medio (MSE):** Misura la media dei quadrati degli errori, penalizzando maggiormente gli errori grandi. È una delle metriche più utilizzate per problemi di regressione, ma è sensibile ai valori anomali.
- **Errore Assoluto Medio (MAE):** Misura la media delle differenze assolute tra le previsioni e i valori reali. A differenza del MSE, il MAE è meno sensibile agli outlier, rendendolo una buona scelta quando si desidera una valutazione robusta.
- **R² (R-quadrato):** Questa metrica rappresenta la proporzione della varianza nei dati di output che è spiegata dal modello. Un valore vicino a 1 indica che il modello spiega bene la varianza dei dati, mentre un valore vicino a 0 indica il contrario.

- **Cross-Validation:**

La cross-validation è una tecnica statistica utilizzata per valutare la capacità di generalizzazione di un modello. Uno dei metodi più comuni è la **k-fold cross-validation**, dove il dataset viene diviso in k sottoinsiemi (folds), e il modello viene addestrato k volte, utilizzando ogni volta un diverso fold come set di test e gli altri k-1 come set di addestramento. La cross-validation aiuta a mitigare l'overfitting, fornendo una stima più affidabile delle prestazioni del modello.

- **Bias e Varianza:**

Il tradeoff tra bias e varianza è cruciale nella valutazione dei modelli. **Bias** alto indica che il modello è troppo rigido e non riesce a catturare la complessità dei dati (underfitting), mentre **varianza** alta indica che il modello è troppo sensibile ai dati di addestramento e non riesce a generalizzare (overfitting). Le tecniche di regolarizzazione, il tuning dei parametri e la scelta appropriata degli algoritmi possono aiutare a bilanciare bias e varianza per ottenere un modello ottimale.

Valutazione nei Problemi di Classificazione: Un Esempio di Recidiva Penale

Consideriamo un modello utilizzato per predire la recidiva penale, dove l'obiettivo è determinare se un individuo sarà recidivo (1) o non recidivo (0) entro un certo periodo di tempo. Supponiamo di avere un dataset di test composto da 100 individui, e il modello ha prodotto le seguenti previsioni:

- **Veri Positivi (VP):** 40 (il modello ha correttamente predetto che 40 individui sarebbero recidivi)
- **Falsi Positivi (FP):** 10 (il modello ha predetto che 10 individui sarebbero recidivi, ma in realtà non lo sono)
- **Veri Negativi (VN):** 30 (il modello ha correttamente predetto che 30 individui non sarebbero recidivi)
- **Falsi Negativi (FN):** 20 (il modello ha predetto che 20 individui non sarebbero recidivi, ma in realtà lo sono)

Utilizziamo queste informazioni per calcolare alcune delle metriche di valutazione più comuni:

- **Accuratezza (Accuracy):** La proporzione di tutte le previsioni corrette sul totale delle osservazioni.

$$\text{Accuratezza} = \frac{VP + VN}{VP + FP + VN + FN} = \frac{40 + 30}{40 + 10 + 30 + 20} = \frac{70}{100} = 0,7$$

L'accuratezza del modello è 0,7, cioè il 70% delle previsioni sono corrette.

- **Precisione (Precision):** La proporzione di veri positivi sul totale delle previsioni positive (veri positivi più falsi positivi).

$$\text{Precisione} = \frac{VP}{VP + FP} = \frac{40}{40 + 10} = \frac{40}{50} = 0,8$$

La precisione del modello è 0,8, ovvero l'80% delle predizioni di recidiva erano corrette.

- **Recall (Sensibilità):** La proporzione di veri positivi sul totale dei veri recidivi (veri positivi più falsi negativi).

$$\text{Recall} = \frac{VP}{VP + FN} = \frac{40}{40 + 20} = \frac{40}{60} = 0,67$$

Il recall del modello è 0,67, cioè il 67% dei recidivi è stato correttamente identificato.

- **F1-Score:** La media armonica di precisione e recall, che fornisce un compromesso tra le due metriche.

$$\text{F1-Score} = 2 \times \frac{\text{Precisione} \times \text{Recall}}{\text{Precisione} + \text{Recall}} = 2 \times \frac{0,8 \times 0,67}{0,8 + 0,67} = 2 \times \frac{0,536}{1,47} \approx 0,73$$

L'F1-Score del modello è 0,73, che indica un buon bilanciamento tra precisione e recall.

Valutazione dei Risultati: Questo modello presenta una precisione alta, il che è positivo per evitare falsi allarmi, ma il recall è relativamente basso, suggerendo che alcuni recidivi non vengono identificati. In un contesto di recidiva penale, potrebbe essere necessario considerare un compromesso tra riduzione dei falsi positivi e aumento del recall, magari esplorando altre strategie di modellazione o modificando la soglia decisionale del modello.

Valutazione nei Problemi di Regressione: Un Esempio di Previsione del valore di un Immobile

Consideriamo ora un modello di regressione utilizzato per predire il valore di un immobile. Supponiamo di avere un dataset di test con i valori reali di 5 immobili e le previsioni del modello, come mostrato nella tabella seguente:

Immobile	Valore Reale (€)	Valore Predetto (€)
1	300,000	310,000
2	450,000	430,000
3	500,000	490,000
4	400,000	420,000
5	350,000	345,000

Utilizziamo questi dati per calcolare alcune metriche di valutazione:

- **Errore Assoluto Medio (MAE):** La media delle differenze assolute tra i valori predetti e quelli reali.

$$\begin{aligned} \text{MAE} &= \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i| = \\ &= \frac{|300 - 310| + |450 - 430| + |500 - 490| + |400 - 420| + |350 - 345|}{5} \\ \text{MAE} &= \frac{10,000 + 20,000 + 10,000 + 20,000 + 5,000}{5} = \frac{65,000}{5} = 13,000 \text{ €} \end{aligned}$$

L'errore assoluto medio è 13,000 €, indicando che, in media, le previsioni del modello differiscono dal valore reale di circa 13,000 €.

- **Errore Quadratico Medio (MSE):** La media dei quadrati degli errori, che penalizza maggiormente gli errori più grandi.

$$\begin{aligned} \text{MSE} &= \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2 = \\ &= \frac{(300 - 310)^2 + (450 - 430)^2 + (500 - 490)^2 + (400 - 420)^2 + (350 - 345)^2}{5} \\ &= \frac{(10,000)^2 + (-20,000)^2 + (-10,000)^2 + (-20,000)^2 + (5,000)^2}{5} \\ &= \frac{100,000,000 + 400,000,000 + 100,000,000 + 400,000,000 + 25,000,000}{5} \\ &= \frac{1,025,000,000}{5} = 205,000,000 \text{ €}^2 \end{aligned}$$

L'errore quadratico medio è 205,000,000 €², che indica un'elevata sensibilità agli errori più grandi.

- **Errore Quadratico Medio Radice (RMSE):** La radice quadrata dell'MSE, che riporta l'errore medio alla stessa scala delle variabili predette.

$$\text{RMSE} = \sqrt{\text{MSE}} = \sqrt{205,000,000} \approx 14,318 \text{ €}$$

Il valore dell'RMSE è 14,318 €, fornendo un'indicazione dell'errore medio sulle predizioni in termini di valore dell'immobile.

- **R² (R-quadrato):** Misura la proporzione della varianza nei valori reali spiegata dal modello.

$$R^2 = 1 - \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{\sum_{i=1}^n (y_i - \bar{y})^2}$$

Dove \bar{y} è il valore medio dei valori reali. Supponiamo che $\bar{y} = 400,000$ e che la somma dei quadrati delle differenze rispetto alla media sia 200,000,000,000 €. Allora:

$$R^2 = 1 - \frac{1,025,000,000}{200,000,000,000} \approx 0,994$$

Valutazione dei Risultati: Il modello di regressione mostra una forte capacità di spiegare la varianza nei dati ($R^2 = 0,994$), il che suggerisce che è altamente predittivo per questo specifico dataset. Tuttavia, l'RMSE di 14,318 € e il MAE di 13,000 € indicano che, sebbene il modello sia generalmente accurato, ci sono ancora errori significativi nelle predizioni, che potrebbero essere rilevanti in contesti dove la precisione è cruciale, come nella determinazione del valore di un immobile per fini fiscali o di compravendita. La presenza di un MSE elevato suggerisce che il modello potrebbe essere sensibile agli outlier, e ulteriori indagini potrebbero essere necessarie per migliorare la robustezza del modello, magari esplorando tecniche di regolarizzazione o di gestione degli outlier.

In ambito giuridico, la valutazione dei modelli può assumere un ruolo particolarmente delicato, poiché la precisione delle previsioni può influenzare decisioni importanti. Ad esempio, in un sistema predittivo per la recidiva criminale, è cruciale minimizzare sia i falsi positivi (persone etichettate erroneamente come ad alto rischio) che i falsi negativi (persone etichettate erroneamente come a basso rischio), utilizzando metriche come l'AUC-ROC e l'F1-score per garantire un equilibrio tra precisione e recall. In contesti dove i falsi positivi possono avere conseguenze legali severe, come nel rilevamento di frodi, l'accuratezza da sola potrebbe non essere sufficiente, rendendo necessaria una valutazione più sfumata attraverso una combinazione di metriche.

3.2 Apprendimento Non Supervisionato

L'apprendimento non supervisionato è un ramo del machine learning in cui i modelli vengono addestrati su dati senza etichette, ossia senza output conosciuti. L'obiettivo è scoprire strutture, pattern o relazioni nascoste all'interno dei dati. A differenza dell'apprendimento supervisionato, che richiede un dataset etichettato, l'apprendimento non supervisionato si concentra su come raggruppare dati simili o ridurre la complessità dei dati mantenendo le informazioni essenziali. Di seguito esamineremo i principali approcci e tecniche utilizzati in questo campo.

3.2.1 Clustering

Il clustering è una tecnica di apprendimento non supervisionato che mira a raggruppare i dati in gruppi (cluster) in base alla somiglianza tra gli elementi. Gli elementi all'interno di un cluster sono più simili tra loro rispetto a quelli di cluster differenti. Questa tecnica è ampiamente utilizzata per esplorare la struttura sottostante dei dati e per identificare segmenti naturali all'interno di un dataset.

Esistono vari metodi di clustering, tra cui:

- **K-means:** Uno degli algoritmi di clustering più popolari, il k-means divide il dataset in k cluster, dove k è un numero predefinito. L'algoritmo funziona iterativamente, assegnando ogni punto dati al cluster il cui centroide è il più vicino e poi aggiornando i centroidi in base ai punti assegnati. Il processo continua fino a che i centroidi non cambiano più o le assegnazioni dei punti si stabilizzano. K-means è semplice ed efficace, ma può essere sensibile alla scelta di k e ai valori iniziali dei centroidi.
- **Agglomerative Hierarchical Clustering:** Questo approccio costruisce una gerarchia di cluster attraverso un processo iterativo in cui ogni punto dati inizia come un cluster separato, e a ogni passo, i due cluster più vicini vengono uniti. Questo processo continua fino a che tutti i punti dati non appartengono a un singolo cluster. Il risultato può essere rappresentato come un dendrogramma, che visualizza la struttura gerarchica dei cluster. Questo metodo è utile per esplorare la struttura dei dati a diversi livelli di granularità.
- **DBSCAN (Density-Based Spatial Clustering of Applications with Noise):** DBSCAN è un algoritmo di clustering basato sulla densità che identifica cluster di alta densità separati da aree di bassa densità. A differenza di k-means, DBSCAN non richiede di specificare il numero di cluster in anticipo e può identificare cluster di forma arbitraria, oltre a gestire outlier in modo naturale.

3.2.2 Riduzione della Dimensionalità

La riduzione della dimensionalità è una tecnica che mira a ridurre il numero di variabili (o caratteristiche) nel dataset mantenendo la maggior parte dell'informazione rilevante. Questo è particolarmente utile quando si lavora con dati ad alta dimensionalità, dove un numero elevato di variabili può complicare l'analisi e aumentare il rischio di overfitting.

Alcuni dei metodi principali per la riduzione della dimensionalità includono:

- **Principal Component Analysis (PCA):** PCA è una tecnica matematica che trasforma i dati in un nuovo spazio di coordinate ridotto, dove le nuove variabili (componenti principali) sono combinazioni lineari delle variabili originali. Le componenti principali sono ordinate in modo tale che la prima componente catturi la massima varianza nei dati, la seconda componente catturi la seconda massima varianza, e così via. Riducendo il numero di componenti principali, PCA può ridurre la dimensionalità dei dati preservando gran parte dell'informazione originale.
- **t-SNE (t-Distributed Stochastic Neighbor Embedding):** t-SNE è una tecnica di riduzione della dimensionalità non lineare che è particolarmente efficace per la visualizzazione di dati ad alta dimensionalità. Riduce i dati in uno spazio a 2 o 3 dimensioni, preservando le relazioni di vicinanza tra i punti dati, il che lo rende ideale per visualizzare cluster naturali nei dati.

- **Autoencoder:** Gli autoencoder sono reti neurali progettate per imparare una rappresentazione compressa dei dati. Sono composti da due parti: l'encoder, che riduce i dati in uno spazio a bassa dimensionalità, e il decoder, che ricostruisce i dati originali dalla rappresentazione compressa. Gli autoencoder sono particolarmente utili per la riduzione della dimensionalità in problemi complessi dove le relazioni tra le variabili non sono lineari.

3.2.3 Algoritmi Principali

Nel contesto dell'apprendimento non supervisionato, ci sono diversi algoritmi che giocano un ruolo fondamentale. Alcuni di questi sono:

- **K-means:** Come già discusso, k-means è un algoritmo di clustering ampiamente utilizzato grazie alla sua semplicità ed efficacia. Tuttavia, la scelta del numero di cluster (k) e la sensibilità ai valori iniziali possono influenzare significativamente i risultati.
- **Gaussian Mixture Models (GMM):** GMM è un metodo di clustering che assume che i dati siano generati da una combinazione di distribuzioni gaussiane. Ogni cluster è modellato come una distribuzione gaussiana, e l'algoritmo cerca di trovare i parametri delle gaussiane che meglio spiegano la distribuzione dei dati. GMM è più flessibile di k-means in quanto può modellare cluster con forme ellittiche e non richiede che i cluster siano di forma sferica.
- **Hierarchical Clustering:** Questo algoritmo costruisce una gerarchia di cluster che possono essere esplorati a diversi livelli di granularità. Può essere agglomerativo o divisivo, offrendo una rappresentazione visiva della struttura dei cluster attraverso dendrogrammi.
- **DBSCAN:** Oltre al clustering basato sulla densità, DBSCAN è noto per la sua capacità di identificare outlier, rendendolo particolarmente utile in dataset con rumore o in cui i cluster non sono facilmente distinguibili.
- **PCA:** Sebbene originariamente progettato per la riduzione della dimensionalità, PCA è spesso utilizzato anche come metodo preliminare di analisi per comprendere la struttura dei dati e preparare i dati per ulteriori analisi di clustering.
- **t-SNE:** Questo algoritmo è particolarmente apprezzato per la visualizzazione di dati ad alta dimensionalità, specialmente in contesti dove è importante capire la struttura interna dei dati, come nel clustering o nell'identificazione di pattern nascosti.

3.2.4 Applicazioni

L'apprendimento non supervisionato trova applicazione in una vasta gamma di settori e problemi, soprattutto in contesti in cui i dati non sono etichettati e l'obiettivo è scoprire strutture nascoste o ridurre la complessità dei dati. Alcune delle principali applicazioni includono:

- **Segmentazione del Mercato:** Le tecniche di clustering come k-means e GMM sono utilizzate per segmentare i clienti in gruppi omogenei in base ai loro comportamenti o caratteristiche, permettendo strategie di marketing mirate e personalizzate.
- **Analisi delle Reti Sociali:** L'apprendimento non supervisionato può essere utilizzato per identificare comunità o gruppi di interesse all'interno di reti sociali, analizzando le connessioni tra individui o entità.
- **Rilevamento delle Anomalie:** Algoritmi come DBSCAN sono utilizzati per identificare outlier o anomalie nei dati, come transazioni fraudolente, guasti nei sistemi o attività insolite.
- **Preprocessing dei Dati per Modelli Supervisionati:** La riduzione della dimensionalità attraverso PCA o autoencoder è spesso utilizzata come fase di preprocessing per migliorare le prestazioni di modelli di apprendimento supervisionato, riducendo il rumore e la complessità dei dati.
- **Visualizzazione dei Dati:** Tecniche come t-SNE sono utilizzate per ridurre la dimensionalità dei dati ad alta complessità, facilitando la visualizzazione e l'interpretazione delle strutture interne dei dati, come cluster o pattern nascosti.

In ambito giuridico, l'apprendimento non supervisionato può essere utilizzato per l'analisi di grandi volumi di documenti legali, la scoperta di pattern nei dati dei casi, o la segmentazione dei casi giudiziari per identificare tipologie ricorrenti e relazioni tra i casi. Queste tecniche forniscono strumenti potenti per esplorare e comprendere i dati in modo più profondo, senza la necessità di etichette predefinite.

3.3 Bias

Il bias nei modelli di machine learning è una questione critica, soprattutto in settori sensibili come il diritto, dove le decisioni automatizzate possono avere implicazioni significative su persone e gruppi. Il bias può portare a previsioni errate e ingiuste, perpetuando disuguaglianze sociali e legali. In questa sezione, esploreremo in dettaglio le diverse tipologie di bias, le cause e gli impatti che esse possono avere, e le tecniche per mitigare questi bias, con esempi pratici che illustrano il problema.

3.3.1 Tipologie di Bias

I modelli di machine learning possono essere affetti da vari tipi di bias, ciascuno con caratteristiche specifiche e potenziali impatti:

- **Bias di Selezione:** Si verifica quando il dataset utilizzato per addestrare il modello non rappresenta adeguatamente la popolazione o il fenomeno che si intende modellare. Ad esempio, immaginate un modello di machine learning sviluppato per predire il successo di un'azione legale basato su dati storici. Se il dataset include solo casi di successo e non quelli falliti, il modello potrebbe sovrastimare le probabilità di successo. Supponiamo che su un campione di 1.000 casi, solo 100 siano stati inclusi nel dataset e questi siano stati scelti per la loro rilevanza legale; se 90 di questi 100 casi sono stati di successo, il modello potrebbe imparare che il successo è estremamente probabile (90%), ignorando che nella realtà solo 200 dei 1.000 casi totali erano di successo, riducendo la probabilità reale al 20%.
- **Bias di Conferma:** Questo tipo di bias emerge quando i dati o le caratteristiche selezionate per il modello confermano preconetti o ipotesi preesistenti. Ad esempio, se un modello per la concessione del credito utilizza dati storici di prestiti concessi prevalentemente a individui di una determinata etnia o genere, potrebbe perpetuare lo stesso comportamento discriminatorio. Se nel dataset storico il 70% dei prestiti è stato concesso a uomini, il modello potrebbe imparare a favorire inconsciamente le richieste di prestito fatte da uomini, basando le sue decisioni su pattern storici piuttosto che su criteri obiettivi di solvibilità.
- **Bias di Sopravvivenza:** Questo bias si verifica quando l'analisi si basa solo sui dati relativi ai "sopravvissuti" a un determinato processo, ignorando i casi che non ce l'hanno fatta. Ad esempio, se un'analisi per determinare i fattori di successo per avviare uno studio legale si basa solo su studi legali che sono riusciti, si ignoreranno i casi di studi legali che hanno fallito, portando a una sovrastima delle caratteristiche di successo. Se su 1.000 studi legali, solo 100 sono rimasti attivi dopo 10 anni, ma l'analisi considera solo questi 100, il modello non riuscirà a catturare i motivi del fallimento degli altri 900.
- **Bias Sistemico:** Il bias sistemico riflette le disuguaglianze o le discriminazioni già presenti nei dati storici e nei sistemi sociali. Per esempio, un modello che predice la recidiva basato su dati storici che riflettono pratiche discriminatorie della polizia potrebbe perpetuare tali discriminazioni. Se i dati storici mostrano che un determinato gruppo etnico ha un tasso di arresto più alto a causa di pratiche di profilazione razziale, il modello potrebbe etichettare automaticamente questo gruppo come più a rischio di recidiva, senza considerare il bias nella raccolta dei dati.
- **Bias Algoritmico:** Questo tipo di bias è introdotto dall'algoritmo stesso, spesso a causa di obiettivi di ottimizzazione che non rappresentano adeguatamente il problema. Ad esempio, se un algoritmo di scoring creditizio è ottimizzato esclusivamente per ridurre i tassi di insolvenza, potrebbe penalizzare ingiustamente gruppi demografici che storicamente hanno avuto meno accesso al credito. Supponiamo che l'algoritmo tenda a ridurre il credito alle persone con un background socio-economico più basso per minimizzare il rischio; in questo caso, il bias si manifesta nel modo in cui l'algoritmo interpreta e valuta le caratteristiche socio-economiche.

3.3.2 Cause e Impatti del Bias

Le cause del bias nei modelli di machine learning sono molteplici e possono derivare da diverse fasi del ciclo di vita del modello:

- **Dataset Non Rappresentativi:** Uno dei motivi principali del bias è l'uso di dataset non rappresentativi della popolazione target. Se il modello è addestrato su dati che non riflettono tutte le possibili variabili del problema, esso sarà inevitabilmente biased. Ad esempio, se un modello di predizione della recidiva è addestrato principalmente su dati relativi a reati minori, potrebbe non essere accurato quando applicato a reati più gravi. Questo può portare a una sovrastima del rischio per alcuni individui, con conseguenti decisioni ingiuste.
- **Scelte di Modellazione:** Le decisioni prese durante la fase di modellazione, come la selezione delle caratteristiche o la scelta dell'algoritmo, possono introdurre bias. Se le caratteristiche scelte riflettono pregiudizi culturali o storici, il modello potrebbe imparare e riprodurre questi pregiudizi. Ad esempio, se si decide di includere la variabile "quartiere di residenza" in un modello per la concessione di prestiti, questo potrebbe introdurre bias se certi quartieri sono associati a gruppi etnici o socio-economici specifici.
- **Feedback Loop:** Un feedback loop si verifica quando le previsioni di un modello influenzano i dati futuri, rafforzando il bias. Ad esempio, un sistema di polizia predittiva che invia più pattuglie in quartieri già sorvegliati più intensamente potrebbe generare più arresti in quelle aree, portando a un ulteriore aumento della sorveglianza e a un rafforzamento del bias iniziale. Se un modello predice che una particolare zona ha un'alta probabilità di criminalità e quindi concentra lì le risorse di polizia, si avrà un numero maggiore di segnalazioni di crimini in quell'area, creando un ciclo che perpetua il bias.

Gli impatti del bias possono essere devastanti, soprattutto in contesti dove le decisioni automatizzate influenzano direttamente le vite delle persone:

- **Discriminazione:** Un modello biased può perpetuare o amplificare disuguaglianze esistenti, portando a decisioni discriminatorie. Ad esempio, se un modello di scoring creditizio discrimina ingiustamente contro minoranze etniche, potrebbe negare l'accesso al credito a persone altrimenti meritevoli. Supponiamo che in un dataset di 10.000 richieste di prestito, il modello respinga il 30% delle richieste da parte di una minoranza etnica specifica a causa di un bias nei dati storici; questo porterebbe a una discriminazione ingiustificata e potenzialmente illegale.
- **Perdita di Fiducia:** Se i sistemi di intelligenza artificiale sono percepiti come ingiusti o discriminatori, la fiducia del pubblico in questi sistemi può essere gravemente compromessa, limitando la loro accettazione e utilità. Ad esempio, se un sistema di giustizia predittiva è percepito come discriminatorio nei confronti di certi gruppi etnici, potrebbe sollevare preoccupazioni pubbliche e ridurre la legittimità delle decisioni automatizzate, portando a un rifiuto dell'uso di tali tecnologie.

- **Conseguenze Legali:** L'uso di modelli biased in decisioni legali può portare a contenziosi e danni reputazionali per le istituzioni che li utilizzano, oltre a violare normative e leggi sulla non discriminazione. Ad esempio, un'azienda che utilizza un modello di selezione del personale che favorisce inconsciamente candidati maschi rispetto a candidati femmine potrebbe essere esposta a cause legali per discriminazione di genere, con conseguenti danni economici e reputazionali.

3.3.3 Tecniche di Mitigazione del Bias

Esistono diverse tecniche per mitigare il bias nei modelli di machine learning, che possono essere implementate in varie fasi del ciclo di vita del modello:

- **Raccolta e Preparazione dei Dati:** Una delle tecniche più efficaci per mitigare il bias è assicurarsi che il dataset sia il più possibile rappresentativo della popolazione target. Ciò può richiedere la raccolta di dati aggiuntivi per garantire che tutti i gruppi siano equamente rappresentati. Ad esempio, se un dataset per la concessione di mutui mostra che solo il 10% dei richiedenti appartiene a una minoranza etnica, potrebbe essere utile raccogliere dati aggiuntivi per aumentare questa percentuale, riducendo così il bias nel modello.
- **Pre-processing dei Dati:** Prima dell'addestramento del modello, si possono applicare tecniche di pre-processing per ridurre il bias. Un esempio è l'eliminazione di caratteristiche correlate al bias (come genere o etnia) o la normalizzazione dei dati per garantire che nessun gruppo sia sovrarappresentato. Ad esempio, se si sta costruendo un modello di selezione del personale, si potrebbe rimuovere il campo "genere" per evitare che il modello impari a discriminare in base a esso.
- **Modellazione In-process:** Durante l'addestramento, possono essere applicate tecniche di regolarizzazione che penalizzano il modello se sfrutta caratteristiche correlate al bias. Ad esempio, si possono utilizzare obiettivi di equità, come la parità di trattamento tra diversi gruppi. Supponiamo di avere un modello di classificazione che tende a discriminare un gruppo specifico; applicando una regolarizzazione che penalizza il modello se questo gruppo ha un tasso di errore significativamente diverso dagli altri, è possibile ridurre il bias.
- **Post-processing:** Dopo l'addestramento, possono essere applicate tecniche di post-processing per correggere il bias nelle previsioni del modello. Un esempio è la calibrazione delle probabilità predette per garantire che le previsioni siano uniformi tra i diversi gruppi. Se un modello di scoring creditizio assegna punteggi più bassi a un determinato gruppo etnico, si può applicare un aggiustamento che uniformi i punteggi tra i gruppi, riducendo così il bias.
- **Monitoraggio e Aggiornamento Continuo:** Il bias può evolversi nel tempo man mano che cambiano i dati e le condizioni. È quindi essenziale monitorare continuamente

i modelli e aggiornarli periodicamente per garantire che il bias non si reintroduca o peggiori. Ad esempio, un modello utilizzato per la concessione di prestiti potrebbe essere rivalutato ogni anno per verificare che non stia emergendo nuovo bias a causa di cambiamenti nei dati demografici o economici.

- **Analisi di Impatto e Auditing:** Condurre un'analisi di impatto e auditing regolare sui modelli di machine learning è fondamentale per identificare e correggere eventuali bias. Questo processo dovrebbe coinvolgere non solo esperti tecnici, ma anche stakeholder etici e legali per garantire che i modelli siano equi e conformi alle normative. Ad esempio, un modello utilizzato per predire la recidiva potrebbe essere sottoposto a una revisione annuale da parte di un comitato etico, che esamini l'equità delle previsioni e proponga modifiche se necessario.

Conclusione: La gestione del bias nei modelli di machine learning è essenziale per garantire che le applicazioni dell'IA, soprattutto in ambiti sensibili come il diritto, siano eque e giuste. Attraverso l'adozione di tecniche appropriate di mitigazione del bias, è possibile sviluppare modelli che non solo siano accurati, ma che rispettino anche i principi di equità e non discriminazione. Questi modelli possono quindi essere utilizzati in modo responsabile, minimizzando il rischio di perpetuare disuguaglianze e promuovendo decisioni più giuste e trasparenti.

3.4 Reti Neurali

Le reti neurali rappresentano una delle aree più affascinanti e potenti dell'intelligenza artificiale, capaci di modellare e risolvere problemi complessi che spaziano dalla classificazione di immagini alla generazione di testi. Originariamente ispirate al funzionamento del cervello umano, le reti neurali sono diventate un pilastro fondamentale nel campo del machine learning e del deep learning, offrendo soluzioni avanzate per una vasta gamma di applicazioni, compresi i sistemi giuridici. In questo paragrafo, esploreremo le basi delle reti neurali, partendo dai concetti fondamentali della regressione lineare e logistica, fino ad arrivare alle architetture più avanzate utilizzate nei moderni modelli di deep learning.

Il viaggio inizia con la **Regressione Lineare e Logistica** (Sezione 4.4.1), che, pur essendo modelli semplici, costituiscono le fondamenta su cui si costruiscono concetti più complessi delle reti neurali. La regressione lineare è utilizzata per problemi di previsione continua, mentre la regressione logistica è cruciale per problemi di classificazione binaria. Questi modelli, pur essendo relativamente semplici, offrono intuizioni fondamentali sulla modellazione dei dati e sulla costruzione di funzioni di previsione.

Proseguendo, discuteremo il **Percettrone** (Sezione 4.4.2), che è la forma più semplice di rete neurale e rappresenta un singolo neurone artificiale. Il percettrone è in grado di risolvere problemi di classificazione lineare ed è il punto di partenza per la comprensione delle reti neurali più complesse. Nonostante la sua semplicità, il percettrone ha un'importanza storica significativa, poiché ha introdotto il concetto di apprendimento supervisionato nelle reti neurali.

Successivamente, ci addentreremo nelle **Reti Neurali Multistrato (MLP)** (Sezione 4.4.3), che sono estensioni del perceptrone e costituiscono il cuore delle reti neurali moderne. Un MLP è composto da più strati di neuroni, che permettono di modellare relazioni non lineari tra le variabili. Questa capacità di apprendere rappresentazioni complesse rende le MLP strumenti estremamente potenti per una vasta gamma di applicazioni, dalla predizione di valori continui alla classificazione di immagini.

Il capitolo esplorerà poi il mondo del **Deep Learning** (Sezione 4.4.4), una sottocategoria delle reti neurali che ha rivoluzionato il campo dell'intelligenza artificiale. Approfondiremo diverse **Architetture** (Sezione 4.4.4.1), come le Reti Neurali Convoluzionali (CNN), utilizzate principalmente per l'elaborazione delle immagini, le Reti Neurali Ricorrenti (RNN), ideali per l'elaborazione di dati sequenziali come il testo, e le Generative Adversarial Networks (GAN), che hanno aperto nuove frontiere nella generazione di contenuti. Inoltre, esploreremo le **Tecniche di Addestramento** (Sezione 4.4.4.2) che consentono alle reti neurali di apprendere in modo efficiente e accurato da grandi quantità di dati.

Infine, il capitolo si concentrerà sulle **Applicazioni ai Linguaggi Naturali** (Sezione 4.4.5), un'area in cui le reti neurali hanno fatto progressi straordinari. Discuteremo i **Modelli di Linguaggio Preaddestrati** (Sezione 4.4.5.1) e i **Large Language Models (LLM)** (Sezione 4.4.5.2), come GPT e BERT, che sono in grado di comprendere e generare testi in modo sorprendentemente umano. Esploreremo l'**Architettura e le Caratteristiche** (Sezione 4.4.5.2.1) di questi modelli, fornendo **Esempi di LLM** (Sezione 4.4.5.2.2) e analizzando le loro **Applicazioni e Impatti** (Sezione 4.4.5.2.3) nei campi del diritto, dell'educazione e oltre.

Questo capitolo fornirà agli studenti una comprensione profonda e completa delle reti neurali, evidenziando non solo i principi teorici, ma anche le applicazioni pratiche e le implicazioni etiche e sociali dell'uso di queste tecnologie avanzate.

3.4.1 Regressione Lineare e Logistica

Le tecniche di regressione lineare e logistica costituiscono le basi delle reti neurali e di molti altri algoritmi di machine learning. Questi modelli semplici ma potenti permettono di comprendere come le reti neurali apprendono dai dati e come vengono effettuate le previsioni.

3.4.1.1 Regressione Lineare

La **regressione lineare** è uno degli algoritmi più semplici e intuitivi per predire un valore continuo. L'idea centrale della regressione lineare è trovare una funzione lineare che meglio approssimi la relazione tra una o più variabili indipendenti (o caratteristiche) e una variabile dipendente.

Matematicamente, la regressione lineare può essere espressa come:

$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n$$

Dove: - \hat{y} è il valore predetto della variabile dipendente. - x_1, x_2, \dots, x_n sono le variabili indipendenti (le caratteristiche). - β_0 è l'intercetta, che rappresenta il valore di \hat{y} quando tutte le variabili indipendenti sono uguali a zero. - $\beta_1, \beta_2, \dots, \beta_n$ sono i coefficienti di regressione che indicano l'influenza di ciascuna variabile indipendente su \hat{y} .

L'obiettivo della regressione lineare è trovare i valori dei coefficienti $\beta_0, \beta_1, \dots, \beta_n$ che minimizzino la differenza tra i valori predetti \hat{y} e i valori osservati y nei dati. Questa differenza è spesso misurata utilizzando l'**errore quadratico medio** (Mean Squared Error, MSE), definito come:

$$MSE = \frac{1}{m} \sum_{i=1}^m (\hat{y}_i - y_i)^2$$

Dove: - m è il numero di esempi nel dataset. - \hat{y}_i è il valore predetto per l' i -esimo esempio. - y_i è il valore osservato per l' i -esimo esempio.

La regressione lineare trova i coefficienti β che minimizzano l'MSE, utilizzando tecniche di ottimizzazione come il **metodo dei minimi quadrati**.

esempio di regressione lineare: si hanno a disposizione i valori di vari immobili di cui si conosce la superficie calpestabile. Si vuole prevedere/stimare il prezzo di un immobile di 150 m². Le ipotesi sono che tutti gli immobili sono nella stessa area urbana e sono di pari pregio e stato di conservazione. Inoltre, per comodità genereremo i dati usando il generatore di numeri casuali della libreria `numpy`.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

# Generare dati di esempio
np.random.seed(42)
# Metri quadrati (variabile indipendente)
# 50 immobili con una media di 100 m² e deviazione standard di 20 m²
X = np.random.normal(100, 20, 50).reshape(-1, 1)
# Prezzo in migliaia di euro (variabile dipendente) con una certa
# correlazione e aggiunta di rumore
y = 200 + 2 * X.flatten() + np.random.normal(0, 15, X.shape[0])
```

```

# Creare il modello di regressione lineare
model = LinearRegression()
model.fit(X, y)

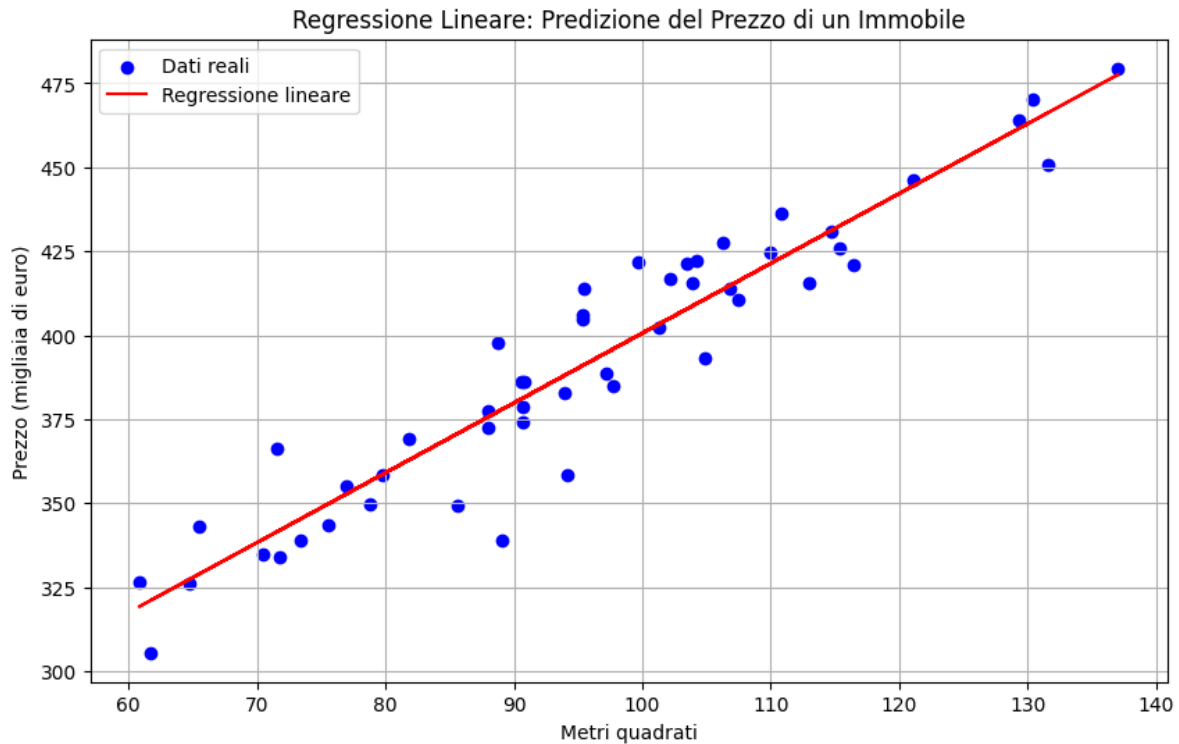
# Predire i valori per la linea di regressione
y_pred = model.predict(X)

# Grafico dei dati e del risultato della regressione
plt.figure(figsize=(10, 6))
plt.scatter(X, y, color='blue', label='Dati reali')
plt.plot(X, y_pred, color='red', label='Regressione lineare')
plt.xlabel('Metri quadrati')
plt.ylabel('Prezzo (migliaia di euro)')
plt.title('Regressione Lineare: Predizione del Prezzo di un Immobile')
plt.legend()
plt.grid(True)
plt.show()

# Applicare i coefficienti della regressione a un immobile di 150 metri quadrati
square_meters = np.array([[150]])
predicted_price = model.predict(square_meters)

model.coef_, model.intercept_, predicted_price[0]

```

```
(array([2.07730673]),
 np.float64(192.88465267224188),
 np.float64(504.48066278658195))
```

Il modello di regressione lineare costruito ha prodotto i seguenti risultati:

- **Coefficiente della regressione** (β_1): 2.08 (approssimato), che rappresenta l'aumento medio del prezzo (in migliaia di euro) per ogni metro quadrato aggiuntivo.
- **Intercetta** (β_0): 192.88 (approssimato), che rappresenta il prezzo di base in migliaia di euro quando l'immobile ha 0 metri quadrati.

Utilizzando questi coefficienti per predire il prezzo di un immobile di 150 metri quadrati:

$$\text{Prezzo predetto} = 192.88 + 2.08 \times 150 \approx 504.48 \text{ migliaia di euro}$$

Quindi, il prezzo stimato per un immobile di 150 metri quadrati è di circa 504.48 migliaia di euro, ossia 504,480 euro.

3.4.1.2 Regressione Logistica

Mentre la regressione lineare è utilizzata per la previsione di valori continui, la **regressione logistica** è un modello di classificazione utilizzato quando l'obiettivo è prevedere una variabile dipendente binaria (cioè, che può assumere solo due valori, come 0 o 1).

La regressione logistica trasforma la previsione lineare utilizzando una **funzione logistica** (o sigmoide), che mappa i valori reali in un intervallo compreso tra 0 e 1. Questo intervallo può essere interpretato come una probabilità.

La funzione logistica è definita come:

$$\text{sigmoide}(z) = \frac{1}{1 + e^{-z}}$$

Dove: - z è la combinazione lineare delle caratteristiche, simile a quella usata nella regressione lineare:

$$z = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n$$

Il valore predetto dalla regressione logistica, \hat{y} , rappresenta la probabilità che la variabile dipendente assuma il valore 1, dato un insieme di caratteristiche x_1, x_2, \dots, x_n . Formalmente:

$$\hat{y} = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n)}}$$

Il modello viene addestrato utilizzando una funzione di perdita specifica, nota come **log-loss** o **cross-entropy loss**, che misura la differenza tra la probabilità predetta e il valore osservato:

$$\text{Log-Loss} = -\frac{1}{m} \sum_{i=1}^m [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)]$$

L'obiettivo è minimizzare la log-loss durante l'addestramento, regolando i coefficienti β per migliorare la capacità del modello di distinguere tra le due classi.

esempio di regressione logistica: si immagini di conoscere i dati relativi ad un coefficiente di rischio e alla probabilità di colpevolezza di un imputato. Per comodità i dati saranno simulati usando il generatore di numeri casuali di numpy con una distribuzione gaussiana.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import ConfusionMatrixDisplay
```

```

# Simuliamo un dataset per predire la probabilità che un imputato sia dichiarato colpevole
# Basandoci su un punteggio di rischio (ipotetico) su scala da 0 a 100

# Generare dati di esempio
np.random.seed(42)
# Punteggio di rischio (variabile indipendente)
# 100 imputati con un punteggio medio di 50 e deviazione standard di 15
X = np.random.normal(50, 15, 100).reshape(-1, 1)
# Esito (colpevole = 1, non colpevole = 0), determinato da una funzione logistica con
# un po' di rumore
y = (1 / (1 + np.exp(-0.1 * (X.flatten() - 50)))) > np.random.rand(100)

# Dividere i dati in train e test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

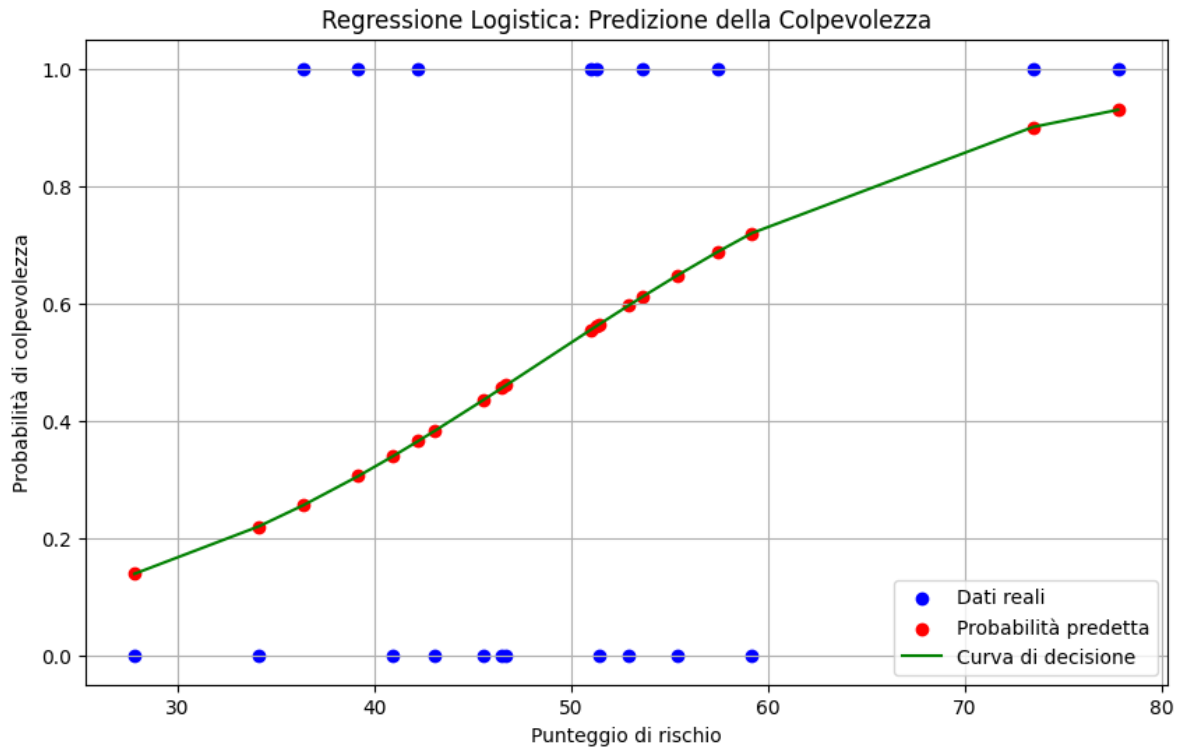
# Creare il modello di regressione logistica
model = LogisticRegression()
model.fit(X_train, y_train)

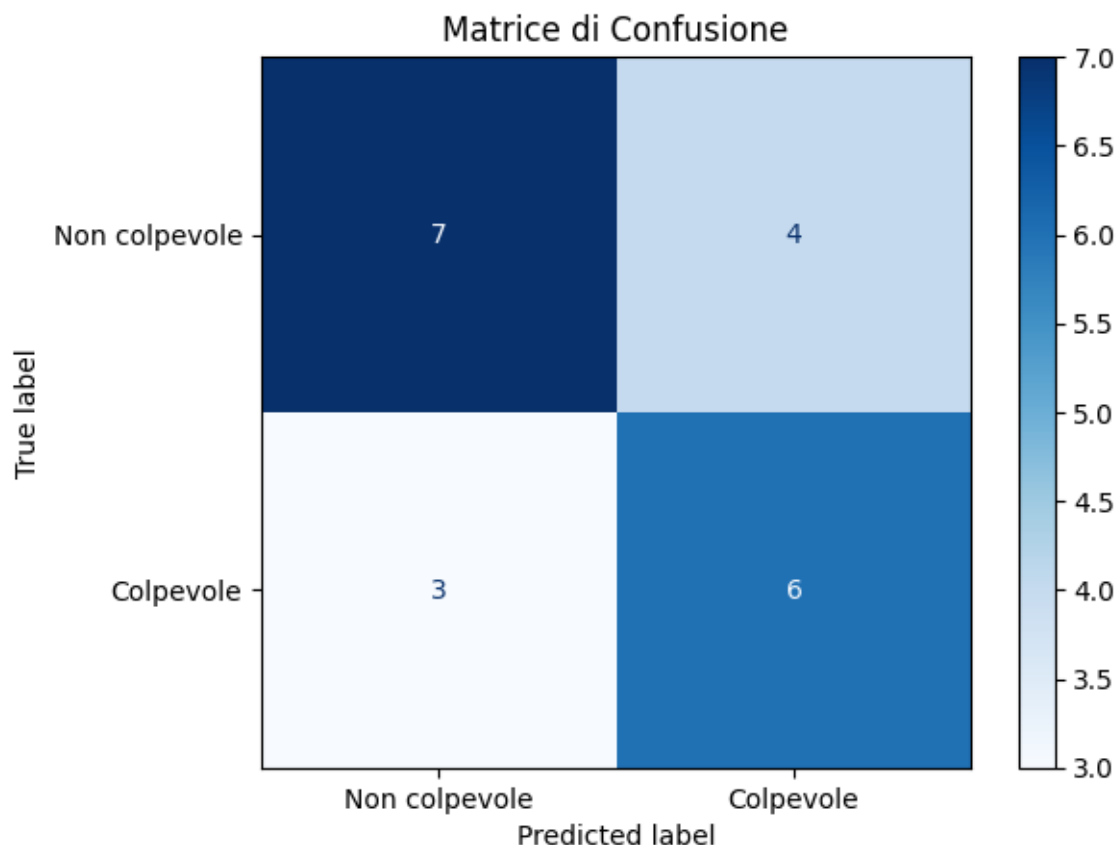
# Predire i valori sul set di test
y_pred_prob = model.predict_proba(X_test)[:, 1]
y_pred = model.predict(X_test)

# Grafico dei dati e della curva di decisione
plt.figure(figsize=(10, 6))
plt.scatter(X_test, y_test, color='blue', label='Dati reali')
plt.scatter(X_test, y_pred_prob, color='red', label='Probabilità predetta')
plt.plot(sorted(X_test.flatten()), sorted(y_pred_prob), color='green', \
         label='Curva di decisione')
plt.xlabel('Punteggio di rischio')
plt.ylabel('Probabilità di colpevolezza')
plt.title('Regressione Logistica: Predizione della Colpevolezza')
plt.legend()
plt.grid(True)
plt.show()

# Mostrare la matrice di confusione
ConfusionMatrixDisplay.from_estimator(model, X_test, y_test, display_labels=\
                                     ["Non colpevole", "Colpevole"], cmap=plt.cm.Blues)
plt.title('Matrice di Confusione')
plt.show()

```





Questo ultimo grafico mostra la distribuzione delle previsioni del modello rispetto ai valori reali, evidenziando il numero di imputati correttamente classificati come “Colpevole” o “Non colpevole” e quelli classificati erroneamente. Questo strumento è utile per valutare le prestazioni del modello e comprendere meglio i suoi errori in un contesto giuridico.

3.4.1.3 Connessione con le Reti Neurali

Le reti neurali, in particolare quelle più semplici come il percettrone, possono essere viste come estensioni dei modelli di regressione lineare e logistica. Un neurone in una rete neurale esegue una combinazione lineare delle caratteristiche (come nella regressione lineare), seguita da una funzione di attivazione (come la funzione logistica nella regressione logistica). Questo permette alle reti neurali di apprendere rappresentazioni complesse e non lineari, rendendole strumenti estremamente potenti per la modellazione dei dati.

In sintesi, la comprensione della regressione lineare e logistica fornisce le basi per comprendere come funzionano le reti neurali, ponendo le fondamenta per concetti più avanzati come il deep learning. Questi modelli non solo offrono un'introduzione alla modellazione predittiva,

ma forniscono anche intuizioni cruciali su come le reti neurali apprendono dai dati per fare previsioni.

3.4.2 Percettrone

Il percettrone è uno dei modelli più semplici e fondamentali delle reti neurali artificiali ed è stato introdotto da Frank Rosenblatt nel 1958. È considerato il progenitore delle reti neurali moderne e rappresenta un singolo neurone artificiale. Nonostante la sua semplicità, il percettrone ha avuto un impatto significativo nello sviluppo dell'intelligenza artificiale, dimostrando che le macchine potevano apprendere a classificare dati attraverso un processo di addestramento supervisionato.

3.4.2.1 Struttura e Funzionamento

Il percettrone è un modello di classificazione lineare che mira a separare i dati in due classi distinte (ad esempio, “positivo” e “negativo”). È composto da un insieme di ingressi, pesi associati a ciascun ingresso, un bias, una funzione di somma e una funzione di attivazione. Ogni ingresso rappresenta una caratteristica del dato da classificare.

Matematicamente, il funzionamento del percettrone può essere descritto come segue:

1. **Calcolo della somma ponderata:** Ogni ingresso (x_i) viene moltiplicato per un peso w_i corrispondente, e viene aggiunto un bias b :

$$z = \sum_{i=1}^n w_i \cdot x_i + b$$

Dove:

- x_i sono le caratteristiche in input.
 - w_i sono i pesi associati agli input.
 - b è il bias, un termine che permette di traslare la funzione di attivazione.
2. **Funzione di attivazione:** Il valore ottenuto dalla somma ponderata viene passato attraverso una funzione di attivazione. Nel percettrone classico, la funzione di attivazione è una funzione a soglia (o funzione di Heaviside):

$$\hat{y} = \begin{cases} 1 & \text{se } z \geq 0 \\ 0 & \text{se } z < 0 \end{cases}$$

Questa funzione determina l'output del percettrone: se la somma ponderata è maggiore o uguale a zero, l'output sarà 1 (classe positiva); altrimenti, sarà 0 (classe negativa).

3.4.2.2 Addestramento del Perceptrone

L'obiettivo dell'addestramento del perceptrone è trovare i pesi w_i e il bias b che permettono al modello di classificare correttamente i dati. Questo processo avviene attraverso l'algoritmo di aggiornamento del perceptrone, che segue questi passi:

1. **Inizializzazione:** I pesi e il bias vengono inizializzati a valori casuali o a zero.
2. **Predizione:** Per ogni esempio nel dataset di addestramento, il perceptrone calcola l'output \hat{y} utilizzando i pesi e il bias correnti.
3. **Aggiornamento:** Se l'output \hat{y} non corrisponde al valore atteso y (cioè, il modello ha commesso un errore), i pesi e il bias vengono aggiornati:

$$w_i \leftarrow w_i + \Delta w_i$$

$$b \leftarrow b + \Delta b$$

$$\Delta w_i = \eta \cdot (y - \hat{y}) \cdot x_i$$

e

$$\Delta b = \eta \cdot (y - \hat{y})$$

, con η che rappresenta il tasso di apprendimento.

4. **Iterazione:** Questo processo continua iterativamente fino a quando il modello non commette più errori o il numero massimo di iterazioni viene raggiunto.

3.4.2.3 Limitazioni e Applicazioni

Una delle limitazioni principali del perceptrone è che può risolvere solo problemi di classificazione linearmente separabili. Questo significa che se le classi non possono essere separate da una linea retta (o un iperpiano in dimensioni superiori), il perceptrone non sarà in grado di classificare correttamente i dati. Tuttavia, l'introduzione di reti neurali multistrato (MLP) ha superato questa limitazione, permettendo di risolvere problemi più complessi.

Nonostante questa limitazione, il perceptrone ha applicazioni pratiche in compiti di classificazione semplice e continua a essere un importante strumento educativo per comprendere i fondamenti delle reti neurali e del machine learning.

esempio di Perceptrone: Esempio di applicazione del perceptrone per predire l'esito di una causa legale, basandosi su due caratteristiche: la complessità del caso e l'esperienza dell'avvocato.

```

import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import Perceptron
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.metrics import ConfusionMatrixDisplay, classification_report
from sklearn.datasets import make_blobs

# Creiamo un dataset simulato per un'applicazione giuridica
# Ad esempio, predire se una causa sarà vinta o persa basandosi su due caratteristiche
# legali (ad es., complessità del caso e esperienza dell'avvocato)

# Generare dati di esempio
X, y = make_classification(n_samples=100, n_features=2, n_informative=2, n_redundant=0,\
                           n_clusters_per_class=1, random_state=42)
# X, y = make_blobs(n_samples=100, centers=[[1,3], [3,1]], random_state=1)
# Dividere i dati in train e test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Creare e addestrare il modello di Perceptrone
perc_model = Perceptron(max_iter=1000, tol=1e-3, random_state=42)
perc_model.fit(X_train, y_train)

# Predire sul set di test
y_pred = perc_model.predict(X_test)

# Grafico dei risultati
plt.figure(figsize=(10, 6))

# Dati reali
plt.scatter(X_test[y_test == 0][:, 0], X_test[y_test == 0][:, 1], color='blue',\
            marker='o', label='Perso (Reale)')
plt.scatter(X_test[y_test == 1][:, 0], X_test[y_test == 1][:, 1], color='green',\
            marker='x', label='Vinto (Reale)')

# Dati predetti
plt.scatter(X_test[y_pred == 0][:, 0], X_test[y_pred == 0][:, 1], color='red',\
            marker='o', facecolors='none', label='Perso (Predetto)')
plt.scatter(X_test[y_pred == 1][:, 0], X_test[y_pred == 1][:, 1], color='yellow',\
            marker='x', label='Vinto (Predetto)')

# Linea di decisione del perceptrone

```



```

x_values = np.linspace(X_test[:, 0].min(), X_test[:, 0].max(), 100)
decision_boundary = -(perc_model.coef_[0, 0] * x_values + perc_model.intercept_[0]) \
                    / perc_model.coef_[0, 1]
plt.plot(x_values, decision_boundary, color='black', linestyle='--', \
         label='Confine Decisionale')

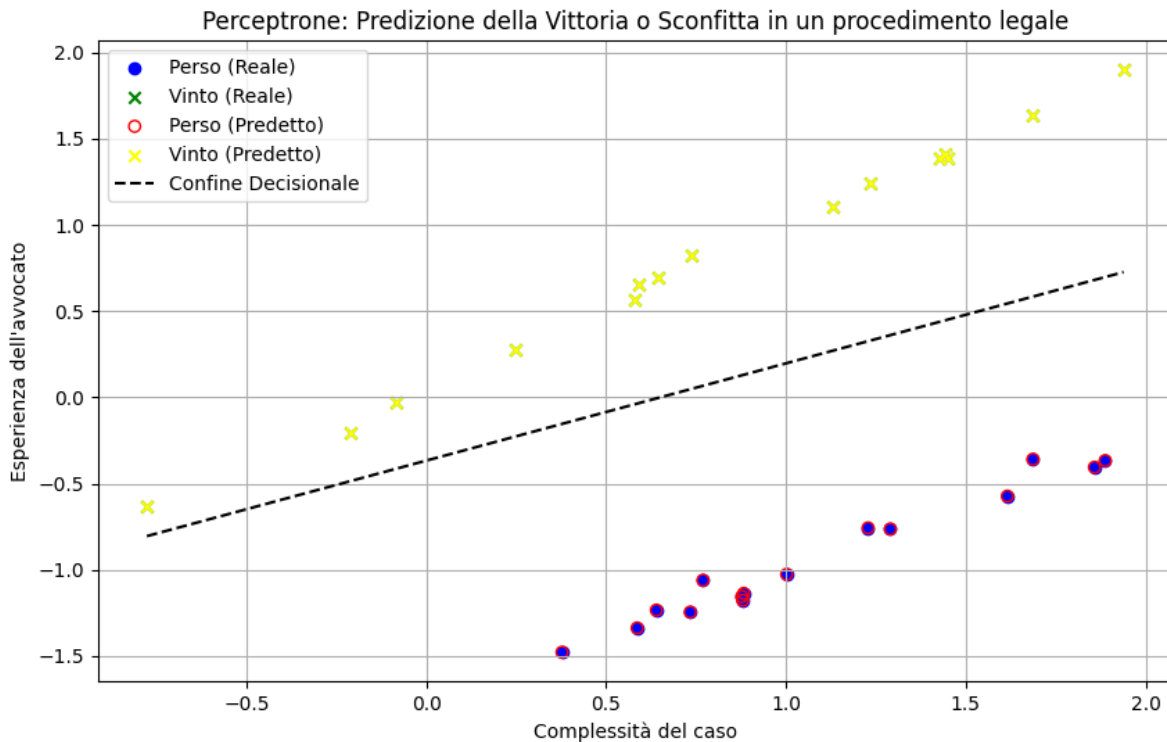
plt.xlabel('Complessità del caso')
plt.ylabel('Esperienza dell\'avvocato')
plt.title('Perceptrone: Predizione della Vittoria o Sconfitta in un procedimento legale')
plt.legend()
plt.grid(True)
plt.show()

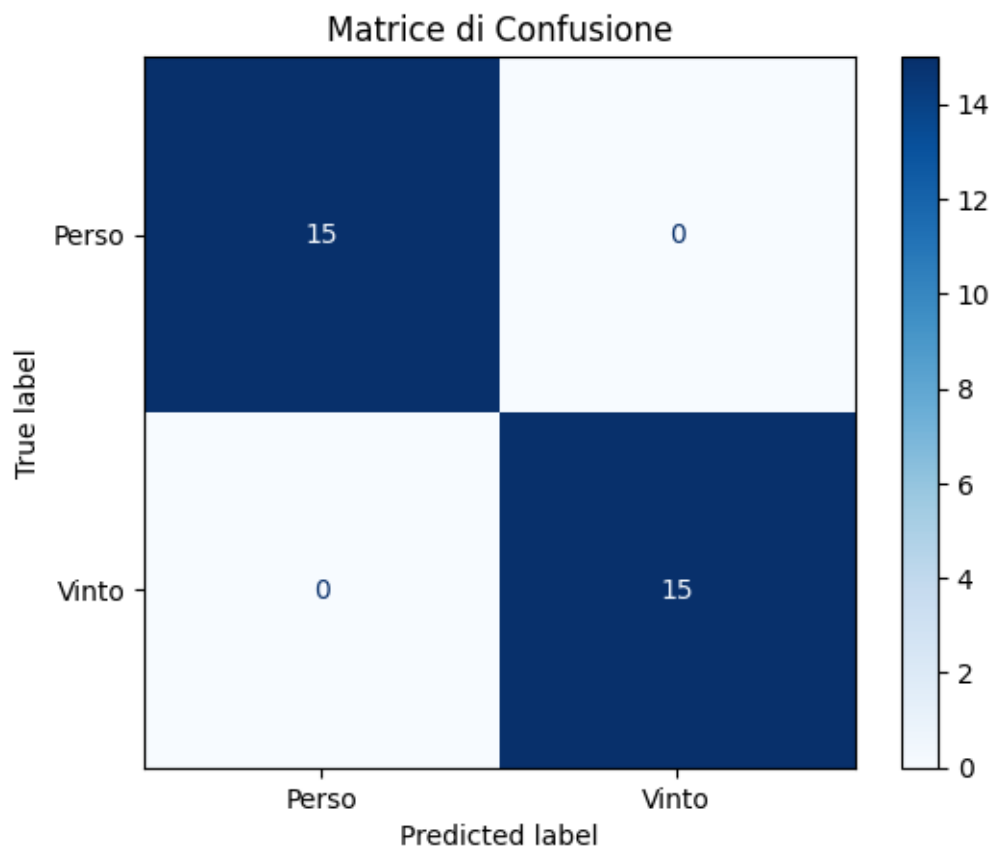
# Mostrare la matrice di confusione e il rapporto di classificazione
ConfusionMatrixDisplay.from_estimator(perc_model, X_test, y_test, display_labels=\
                                     ["Perso", "Vinto"], cmap=plt.cm.Blues)

plt.title('Matrice di Confusione')
plt.show()

print(classification_report(y_test, y_pred, target_names=["Perso", "Vinto"]))

```





	precision	recall	f1-score	support
Perso	1.00	1.00	1.00	15
Vinto	1.00	1.00	1.00	15
accuracy			1.00	30
macro avg	1.00	1.00	1.00	30
weighted avg	1.00	1.00	1.00	30

Nel grafico sopra, i punti blu rappresentano i casi persi, mentre i punti verdi rappresentano i casi vinti, basati sui dati reali. I punti rossi e gialli rappresentano rispettivamente le predizioni del modello per i casi persi e vinti. La linea nera tratteggiata indica il confine decisionale del percettrone, che separa i casi predetti come vinti o persi.

Il modello ha eseguito una classificazione quasi perfetta su questo dataset di test, come evidenziato dalla matrice di confusione e dal rapporto di classificazione. Tutti i casi sono stati

correttamente classificati, con una precisione, recall e F1-score di 1.00 per entrambe le classi (“Perso” e “Vinto”).

Questo risultato, pur essendo ideale, è tipico di dati sintetici e ben separabili linearmente, che spesso non riflettono la complessità dei casi reali. Tuttavia, dimostra come il perceptrone possa essere utilizzato per costruire modelli di classificazione in ambiti giuridici, fornendo una base per decisioni automatizzate in scenari meno complessi. Questo esempio ci aiuta a comprendere l'importanza di valutare le performance del modello in contesti realistici e di considerare l'uso di modelli più complessi, come le reti neurali multistrato, quando le classi non sono facilmente separabili linearmente.

3.4.3 Deep learning

Il deep learning è una sottocategoria avanzata del machine learning che sfrutta reti neurali profonde per affrontare problemi complessi, caratterizzati da grandi quantità di dati e dalla necessità di apprendere rappresentazioni astratte e stratificate delle informazioni. Il viaggio nel deep learning inizia spesso con le Reti Neurali Multistrato (MLP), che rappresentano il primo passo verso la comprensione delle reti neurali profonde.

3.4.3.1 Reti Neurali Multistrato (MLP)

Le Reti Neurali Multistrato (MLP = Multi Layer Perceptron) sono una forma di rete neurale feedforward, composte da uno strato di input, uno o più strati nascosti e uno strato di output. Le MLP sono in grado di apprendere rappresentazioni non lineari dei dati grazie all'uso di funzioni di attivazione non lineari nei neuroni dei loro strati nascosti. Sebbene siano efficaci per molti compiti, le MLP tradizionali hanno una capacità limitata di affrontare problemi particolarmente complessi, poiché sono generalmente composte da pochi strati nascosti.

L'addestramento delle MLP avviene attraverso l'algoritmo di backpropagation, che calcola e minimizza l'errore del modello aggiornando i pesi dei collegamenti tra i neuroni. Questa tecnica è fondamentale non solo per le MLP, ma anche per le reti neurali più complesse utilizzate nel deep learning.

3.4.3.1.1 Architettura delle MLP

Un MLP può essere configurato in diverse architetture a seconda del problema da risolvere. La configurazione più comune è quella con un singolo strato di input, uno o più strati nascosti e uno strato di output. Ogni neurone in uno strato è collegato a tutti i neuroni dello strato successivo, rendendo la rete completamente connessa.

- **Reti con un solo strato nascosto:** Questo è il tipo più semplice di MLP, in cui un singolo strato nascosto è sufficiente per risolvere problemi relativamente semplici o linearmente separabili con una funzione di attivazione non lineare.

- **Reti con più strati nascosti:** Quando i dati presentano una complessità maggiore, un MLP con più strati nascosti può catturare pattern più complessi. Ogni strato aggiuntivo consente alla rete di apprendere rappresentazioni intermedie che possono essere utilizzate per ottenere una predizione finale più accurata.
- **Deep MLP:** Quando il numero di strati nascosti aumenta significativamente, la rete viene considerata “profonda” (deep). Questi modelli, sebbene potenti, richiedono una maggiore attenzione durante l’addestramento per evitare problemi come l’overfitting o la vanishing gradient problem.

3.4.3.1.2 Funzioni di Attivazione

Le funzioni di attivazione sono cruciali per introdurre la non linearità nelle reti neurali, permettendo al modello di apprendere pattern complessi. Le funzioni di attivazione più comuni negli strati nascosti includono:

- **Sigmoide:** Questa funzione mappa qualsiasi valore reale in un intervallo compreso tra 0 e 1, ed è definita come:

$$\text{sigmoide}(z) = \frac{1}{1 + e^{-z}}$$

La funzione sigmoide è utile quando si ha bisogno di un output probabilistico, ma può soffrire del problema della vanishing gradient, che rende difficile l’addestramento di reti profonde.

- **ReLU (Rectified Linear Unit):** Una delle funzioni di attivazione più popolari, definita come:

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

ReLU è ampiamente utilizzata perché risolve in parte il problema della vanishing gradient, accelerando l’addestramento delle reti profonde. Tuttavia, può soffrire del problema della “morte dei neuroni”, dove i neuroni possono rimanere bloccati su zero.

- **Tanh:** Un’alternativa alla funzione sigmoide, mappa i valori in un intervallo tra -1 e 1, ed è definita come:

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

Tanh è spesso preferita alla sigmoide per la sua capacità di centrare i dati attorno a zero, migliorando la convergenza del modello.

3.4.3.1.3 Funzioni di Attivazione degli Strati di Uscita

La scelta della funzione di attivazione nello strato di uscita dipende dal tipo di problema che il modello deve risolvere:

- **Classificazione binaria:** Si utilizza comunemente la funzione sigmoide nello strato di uscita per ottenere una probabilità che l'output appartenga a una delle due classi.
- **Classificazione multiclasse:** La funzione softmax è preferita, poiché mappa i valori di output in un intervallo compreso tra 0 e 1 e la loro somma è 1, fornendo quindi una distribuzione di probabilità tra le diverse classi:

$$\text{softmax}(z_j) = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}}$$

Dove z_j è l'output per la j-esima classe.

- **Regressione:** Per problemi di regressione, in genere non si applica alcuna funzione di attivazione nell'ultimo strato (o si utilizza l'identità) per mantenere l'output come un valore reale continuo.

3.4.3.1.4 Funzioni di Errore

Le funzioni di errore (o funzioni di perdita) misurano la discrepanza tra l'output predetto dal modello e il valore reale, guidando così il processo di apprendimento:

- **Errore Quadratico Medio (MSE):** Utilizzato per problemi di regressione, è definito come:

$$MSE = \frac{1}{n} \sum_{i=1}^n (\hat{y}_i - y_i)^2$$

Dove y_i è il valore reale e \hat{y}_i è il valore predetto.

- **Cross-Entropy Loss:** Utilizzata per la classificazione, particolarmente con softmax o sigmoide, misura la distanza tra le distribuzioni di probabilità:

$$\text{Cross-Entropy} = - \sum_{i=1}^n y_i \log(\hat{y}_i)$$

3.4.3.1.5 Algoritmo di Backpropagation

Il backpropagation è l'algoritmo chiave che permette l'addestramento delle reti neurali multistrato. Funziona in due fasi:

1. **Feedforward:** I dati vengono propagati in avanti attraverso la rete fino a generare un output.
2. **Calcolo della perdita e propagazione all'indietro:** L'errore viene calcolato confrontando l'output predetto con il valore reale. Questo errore viene poi propagato all'indietro attraverso la rete, calcolando il gradiente della funzione di perdita rispetto ai pesi della rete. I pesi vengono aggiornati utilizzando il gradient descent, minimizzando così la funzione di perdita.

Il backpropagation è iterativo e viene eseguito per molte epoche fino a quando il modello non raggiunge un livello accettabile di precisione.

3.4.3.1.6 Esempio di Rete MLP

Applicazione di una rete neurale multistrato (MLP) per la predizione dell'esito di un caso giudiziario basandosi su tre caratteristiche: complessità del caso, esperienza dell'avvocato, e importanza mediatica. La rete è composta da:

- **Strato di input:** Tre neuroni, ciascuno corrispondente a una delle caratteristiche del dataset (complessità del caso, esperienza dell'avvocato, importanza mediatica).
- **Strati nascosti:** Due strati nascosti, il primo con 10 neuroni e il secondo con 5 neuroni, che permettono alla rete di apprendere rappresentazioni più complesse dei dati.
- **Strato di output:** Un singolo neurone di output, utilizzato per la classificazione binaria (vittoria o sconfitta del caso).

Ogni neurone in un determinato strato è connesso a tutti i neuroni dello strato successivo, consentendo il flusso delle informazioni attraverso la rete durante l'addestramento e la predizione. Una rappresentazione grafica di una rete neurale può essere ottenuta usando la libreria `networkx` in Python. Qui di seguito si propone una funzione Python che disegna il grafo di una rete MLP data la sua descrizione in termini di numero di neuroni di ingresso, numero di strati e numero di neuroni per ogni strato e numero di neuroni di uscita.

```
import matplotlib.pyplot as plt
import networkx as nx

# Funzione per disegnare una rappresentazione grafica della rete MLP
def draw_mlp(hidden_layers, input_size, output_size):
    G = nx.DiGraph()
    layer_sizes = [input_size] + list(hidden_layers) + [output_size]

    # Posizionamento dei nodi
    pos = {}
    n_layers = len(layer_sizes)
    v_spacing = 1
    h_spacing = 1 / float(max(layer_sizes))

    # Creazione dei nodi
    for i, layer_size in enumerate(layer_sizes):
        layer_top = v_spacing * (layer_size - 1) / 2
        for j in range(layer_size):
            pos[f'{i}-{j}'] = (i, layer_top - v_spacing * j)
            G.add_node(f'{i}-{j}')
```

```

# Creazione degli archi
for i, (layer_size_a, layer_size_b) in enumerate(zip(layer_sizes[:-1], layer_sizes[1:])):
    for j in range(layer_size_a):
        for k in range(layer_size_b):
            G.add_edge(f'{i}-{j}', f'{i+1}-{k}')

# Disegna il grafico
plt.figure(figsize=(12, 8))
nx.draw(G, pos=pos, with_labels=False, arrows=False, node_size=300, node_color="lightblue")

# Etichette
for i in range(input_size):
    pos[f'0-{i}'] = (pos[f'0-{i}'][0] - 0.1, pos[f'0-{i}'][1])
    plt.text(pos[f'0-{i}'][0], pos[f'0-{i}'][1], f'Input {i+1}', horizontalalignment='right')

for i in range(output_size):
    pos[f'{n_layers-1}-{i}'] = (pos[f'{n_layers-1}-{i}'][0] + 0.1, pos[f'{n_layers-1}-{i}'][1])
    plt.text(pos[f'{n_layers-1}-{i}'][0], pos[f'{n_layers-1}-{i}'][1], f'Output {i+1}', horizontalalignment='left')

plt.title("Rappresentazione Grafica della Rete MLP")
plt.show()

```

Usando la funzione appena introdotta possiamo disegnare la rete MLP usata nell'esempio come segue:

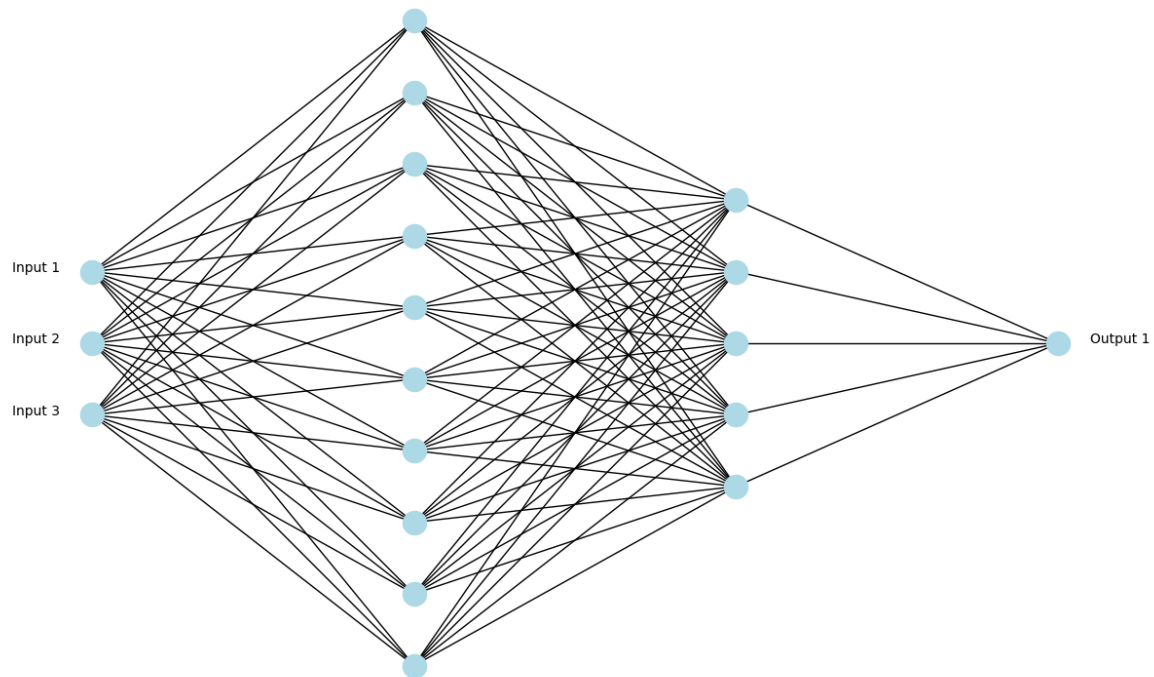
```

# Parametri della rete MLP utilizzata nell'esempio
hidden_layers = (10, 5) # Due strati nascosti con 10 e 5 neuroni rispettivamente
input_size = 3 # Tre caratteristiche in input
output_size = 1 # Un neurone di output (classificazione binaria)

# Disegnare la rappresentazione della rete MLP
draw_mlp(hidden_layers, input_size, output_size)

```

Rappresentazione Grafica della Rete MLP



L'implementazione in Python della rete MLP per il nostro problema di classificazione è la seguente:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neural_network import MLPClassifier
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.metrics import ConfusionMatrixDisplay, classification_report

# Simuliamo un dataset per predire se un caso giudiziario sarà vinto o perso basandosi su tre fattori
# Ad esempio, complessità del caso, esperienza dell'avvocato, e importanza mediatica

# Generare dati di esempio
X, y = make_classification(n_samples=200, n_features=3, n_informative=3, n_redundant=0, n_classes=2)

# Dividere i dati in train e test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```



```

# Creare e addestrare un modello MLP
mlp_model = MLPClassifier(hidden_layer_sizes=(10, 5), max_iter=1000, random_state=42)
mlp_model.fit(X_train, y_train)

# Predire sul set di test
y_pred = mlp_model.predict(X_test)

# Mostrare la matrice di confusione
ConfusionMatrixDisplay.from_estimator(mlp_model, X_test, y_test, display_labels=["Perso", "Vinto"])
plt.title('Matrice di Confusione')
plt.show()

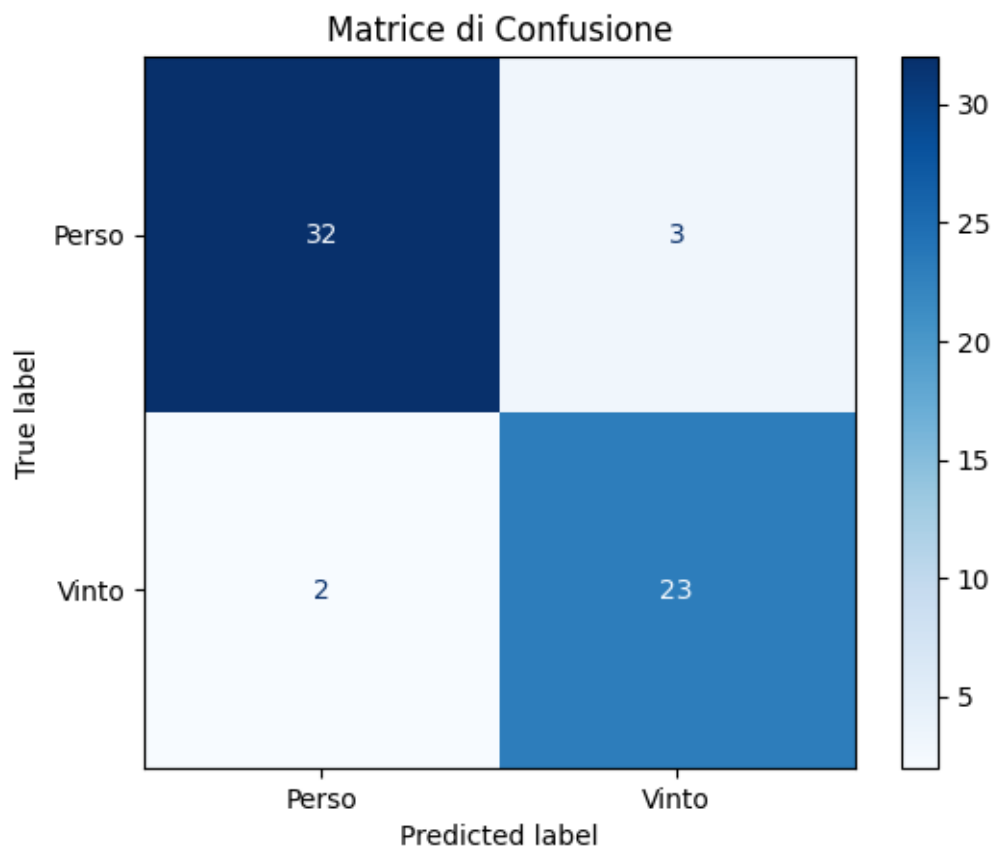
# Visualizzare il rapporto di classificazione
report = classification_report(y_test, y_pred, target_names=["Perso", "Vinto"])
print(report)

```

```

c:\Users\lcapitano\AppData\Local\Programs\Python\Python312\Lib\site-packages\sklearn\neural_networks\mlp\
warnings.warn(

```



	precision	recall	f1-score	support
Perso	0.94	0.91	0.93	35
Vinto	0.88	0.92	0.90	25
accuracy			0.92	60
macro avg	0.91	0.92	0.91	60
weighted avg	0.92	0.92	0.92	60

Analisi dei Risultati

1. **Matrice di Confusione:** La matrice di confusione mostra le prestazioni del modello nella classificazione dei casi giudiziari come “Vinto” o “Perso”. Nel set di test, il modello ha classificato correttamente la maggior parte dei casi, con solo pochi errori. La matrice di confusione indica che il modello ha identificato con una buona precisione sia i casi vinti che quelli persi.
2. **Rapporto di Classificazione:**

- **Precisione:** La precisione per i casi persi è del 94%, mentre per i casi vinti è dell'88%. Questo significa che quando il modello prevede un caso come "Perso", nel 94% dei casi ha ragione, mentre per i casi "Vinto", la precisione è leggermente inferiore.
- **Recall:** La recall per i casi persi è del 91% e per i casi vinti è del 92%. Questo indica che il modello è riuscito a identificare correttamente la maggior parte dei casi vinti e persi.
- **F1-score:** L'F1-score, che rappresenta un bilanciamento tra precisione e recall, è del 93% per i casi persi e del 90% per i casi vinti, riflettendo una buona performance complessiva del modello.

Osservazioni: - Il modello ha raggiunto un'accuratezza complessiva del 92%, che è un buon risultato considerando che i dati generati non sono perfettamente separabili. - È importante notare che il modello non ha raggiunto il valore di convergenza entro il numero massimo di iterazioni impostato (1000), come indicato dall'avviso di convergenza. Questo suggerisce che con ulteriori iterazioni o con l'ottimizzazione dei parametri del modello, le prestazioni potrebbero migliorare ulteriormente. - In sintesi, l'MLP si è dimostrato efficace nel classificare correttamente i casi giudiziari in base alle caratteristiche fornite, anche in presenza di dati non perfettamente distinti. - Questo esempio mostra il potenziale delle reti neurali multistrato per applicazioni giuridiche, come la predizione degli esiti legali, pur sottolineando l'importanza di una corretta configurazione e addestramento del modello per ottenere i migliori risultati possibili.

3.4.3.2 Architetture di Reti Neurali Profonde

Le reti neurali profonde rappresentano una specializzazione e un'estensione delle MLP. Queste reti, spesso costituite da decine o centinaia di strati nascosti, sono in grado di apprendere rappresentazioni molto più complesse e astratte rispetto alle MLP tradizionali. Ogni strato di una rete profonda elabora i dati in modo più dettagliato, consentendo al modello di catturare caratteristiche gerarchiche dei dati, come pattern semplici nei primi strati e strutture più complesse nei successivi.

Reti Neurali Convoluzionali (CNN)

Le reti neurali convoluzionali (CNN) sono una classe specializzata di reti neurali artificiali, particolarmente efficaci nell'elaborazione di dati strutturati a griglia, come le immagini. Il termine "convoluzionale" è fondamentale per comprendere il loro funzionamento unico.

Cos'è la Convoluzione?

La convoluzione è un'operazione matematica che sta alla base di queste reti. In termini semplici, consiste nell'applicare un filtro (o kernel) a una porzione dell'input, facendolo "scorrere" su tutta l'immagine. Questo processo può essere immaginato come una lente che si muove sull'immagine, focalizzandosi su piccole aree alla volta.

```
graph LR
    A[Immagine Input] --> B[Applicazione Filtro]
    B --> C[Feature Map]
    B --> D[Scorrimento]
    D --> |Ripeti| B
    C --> E[Attivazione]
    E --> F[Pooling]
    F --> G[Prossimo Strato]
```

1. Filtri e Feature Maps:

- I filtri sono matrici di pesi che vengono applicati all'input.
- Ogni filtro è progettato per rilevare specifiche caratteristiche (come bordi, curve, o texture).
- Il risultato dell'applicazione di un filtro è chiamato "feature map".

2. Processo di Scorrimento:

- Il filtro si muove sistematicamente attraverso l'immagine, pixel per pixel.
- Ad ogni posizione, esegue una moltiplicazione elemento per elemento e una somma.
- Questo crea una nuova rappresentazione dell'immagine che evidenzia certe caratteristiche.

3. Vantaggi della Convoluzione:

- **Invarianza spaziale:** La stessa caratteristica può essere rilevata ovunque nell'immagine.
- **Parametri condivisi:** I pesi del filtro sono riutilizzati, riducendo il numero totale di parametri.
- **Gerarchia di features:** Strati più profondi combinano features semplici in rappresentazioni più complesse.

Le CNN impilano multiple operazioni di convoluzione, alternate con funzioni di attivazione non lineari (come ReLU) e strati di pooling. Questa architettura permette alla rete di costruire una comprensione gerarchica dell'input, partendo da caratteristiche semplici negli strati iniziali (come bordi e texture) fino a concetti più astratti negli strati più profondi (come forme complesse e oggetti interi). Grazie a questa struttura "convoluzionale", le CNN sono eccezionalmente efficaci in compiti come il riconoscimento di immagini, la detection di oggetti, e la segmentazione semantica, superando spesso le capacità umane in questi domini.

Esempio di creazione, addestramento e test di una rete convoluzionale Applicazione di una rete convoluzionale (CNN) al dataset CIFAR-10 in Python con la libreria [Keras](#) (che fa parte di [TensorFlow](#)). Il dataset [CIFAR-10](#) contiene 60.000 immagini a colori di 32x32 pixel suddivise in 10 classi, con 50.000 immagini per il training e 10.000 immagini per il test. Descrizione del Codice: - Caricamento e Preprocessamento dei Dati: Carichiamo il dataset CIFAR-10 già disponibile in Keras e lo dividiamo in training set e test set. Normalizziamo i valori dei

pixel per far sì che siano compresi tra 0 e 1. - Visualizzazione delle Immagini: Visualizziamo alcune immagini del training set con i rispettivi nomi delle classi per avere un'idea del tipo di dati. - Creazione della Rete Convoluzionale: Utilizziamo tre blocchi di convoluzione seguiti da pooling, aumentando il numero di filtri in ogni strato. Alla fine della rete convoluzionale, utilizziamo uno strato Flatten per trasformare i dati in un formato compatibile con uno strato completamente connesso. Lo strato completamente connesso finale ha 10 unità, corrispondenti alle 10 classi del dataset CIFAR-10. - Compilazione del Modello: Utilizziamo l'ottimizzatore Adam e la funzione di perdita SparseCategoricalCrossentropy. - Allenamento del Modello: Alleniamo il modello per 10 epoche e visualizziamo la precisione e la perdita sia sul training set che sul validation set. - Valutazione: Alla fine, valutiamo il modello sul test set e stampiamo l'accuratezza. - Risultato Atteso: Il modello raggiungerà un'accuratezza intorno al 70-75% sul test set, a seconda della configurazione e dell'hardware utilizzato.

```
# Importiamo le librerie necessarie
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt

# Carichiamo il dataset CIFAR-10
(train_images, train_labels), (test_images, test_labels) = datasets.cifar10.load_data()

# Normalizziamo i valori dei pixel a un intervallo [0, 1]
train_images, test_images = train_images / 255.0, test_images / 255.0

# Mostriamo alcune immagini di esempio dal dataset CIFAR-10
class_names = ['Aereo', 'Auto', 'Uccello', 'Gatto', 'Cervo', 'Cane', 'Rana', 'Cavallo', 'Nav']

plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5, 5, i + 1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i])
    # Le etichette nel dataset sono numeri interi, dobbiamo mapparle con i nomi delle classi
    plt.xlabel(class_names[train_labels[i][0]])
plt.show()

# Creiamo il modello della rete convoluzionale
model = models.Sequential()

# Primo strato convoluzionale
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
```

```

model.add(layers.MaxPooling2D((2, 2)))

# Secondo strato convoluzionale
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))

# Terzo strato convoluzionale
model.add(layers.Conv2D(64, (3, 3), activation='relu'))

# Flatten (convertiamo i dati in un vettore 1D)
model.add(layers.Flatten())

# Strato completamente connesso
model.add(layers.Dense(64, activation='relu'))

# Strato di output con 10 unità (una per ciascuna classe del CIFAR-10)
model.add(layers.Dense(10))

# Riepilogo della rete
model.summary()

# Compiliamo il modello
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

# Alleniamo il modello
history = model.fit(train_images, train_labels, epochs=10,
                    validation_data=(test_images, test_labels))

# Visualizziamo i grafici di accuratezza e perdita
plt.figure(figsize=(10, 4))

plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Accuracy')
plt.plot(history.history['val_accuracy'], label='Val Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Loss')

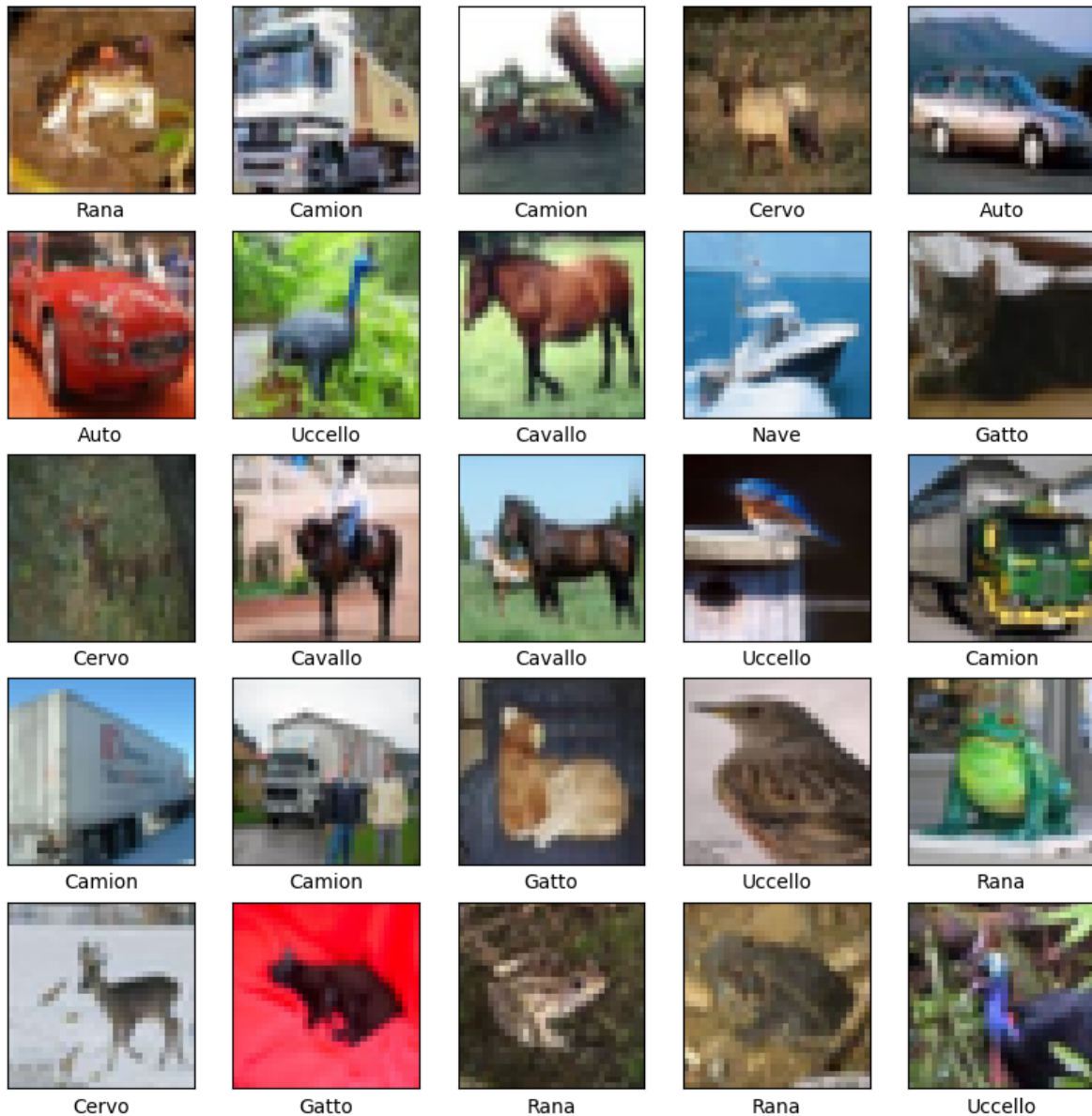
```

```
plt.plot(history.history['val_loss'], label='Val Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.show()

# Valutiamo il modello sui dati di test
test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=2)
print(f'\nAccuratezza sul test set: {test_acc}')
```

Downloading data from <https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz>
170498071/170498071 36s 0us/step



```
c:\Users\lcapitano\AppData\Local\Programs\Python\Python312\Lib\site-packages\keras\src\layer
super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential"

Layer (type)

Output Shape

Param #

conv2d (Conv2D)	(None, 30, 30, 32)	896
max_pooling2d (MaxPooling2D)	(None, 15, 15, 32)	0
conv2d_1 (Conv2D)	(None, 13, 13, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 6, 6, 64)	0
conv2d_2 (Conv2D)	(None, 4, 4, 64)	36,928
flatten (Flatten)	(None, 1024)	0
dense (Dense)	(None, 64)	65,600
dense_1 (Dense)	(None, 10)	650

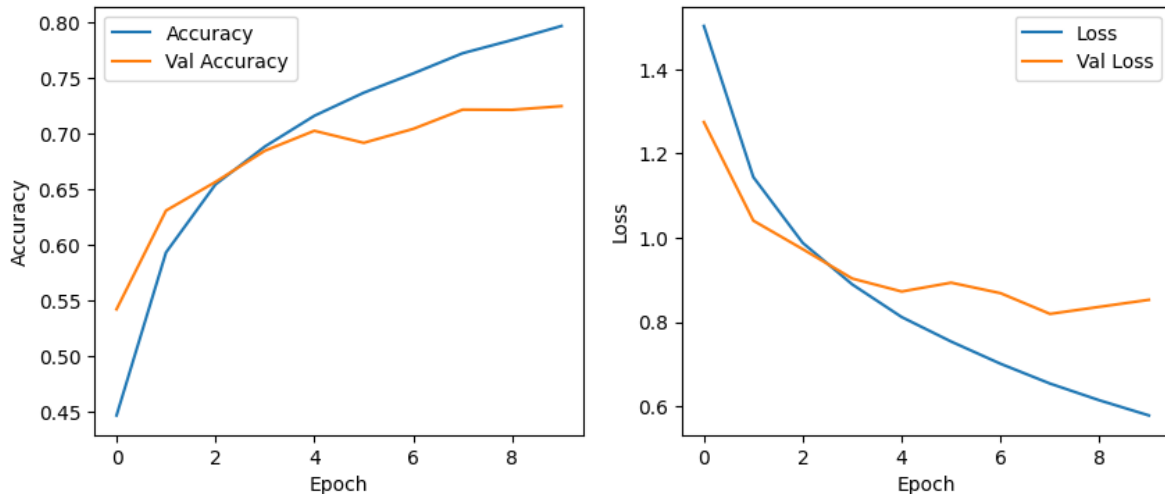
Total params: 122,570 (478.79 KB)

Trainable params: 122,570 (478.79 KB)

Non-trainable params: 0 (0.00 B)

Epoch 1/10	
1563/1563	17s 10ms/step - accuracy: 0.3583 - loss: 1.7137 - val_accuracy: 0.5424
Epoch 2/10	
1563/1563	18s 11ms/step - accuracy: 0.5744 - loss: 1.1928 - val_accuracy: 0.6309
Epoch 3/10	
1563/1563	18s 11ms/step - accuracy: 0.6488 - loss: 1.0025 - val_accuracy: 0.6566
Epoch 4/10	
1563/1563	18s 12ms/step - accuracy: 0.6862 - loss: 0.8934 - val_accuracy: 0.6846
Epoch 5/10	
1563/1563	21s 13ms/step - accuracy: 0.7160 - loss: 0.8103 - val_accuracy: 0.7025
Epoch 6/10	
1563/1563	18s 11ms/step - accuracy: 0.7379 - loss: 0.7481 - val_accuracy: 0.6917
Epoch 7/10	
1563/1563	18s 12ms/step - accuracy: 0.7555 - loss: 0.6952 - val_accuracy: 0.7042
Epoch 8/10	
1563/1563	19s 12ms/step - accuracy: 0.7762 - loss: 0.6473 - val_accuracy: 0.7215
Epoch 9/10	

1563/1563 19s 12ms/step - accuracy: 0.7879 - loss: 0.6004 - val_accuracy: 0.7213
Epoch 10/10
1563/1563 19s 12ms/step - accuracy: 0.8032 - loss: 0.5645 - val_accuracy: 0.7246



313/313 - 3s - 8ms/step - accuracy: 0.7246 - loss: 0.8529

Accuratezza sul test set: 0.7246000170707703

- **Reti Neurali Ricorrenti (RNN):** Le RNN sono un'estensione delle MLP per dati sequenziali, come testi o segnali audio. Grazie alle connessioni ricorrenti, le RNN possono mantenere una memoria delle informazioni precedenti nella sequenza, rendendole ideali per compiti che richiedono la modellazione del contesto temporale. Tuttavia, le RNN tradizionali soffrono del problema del vanishing gradient, che può ostacolare l'apprendimento di dipendenze a lungo termine nelle sequenze. Per superare questa limitazione, sono state sviluppate varianti come le LSTM (Long Short-Term Memory) e le GRU (Gated Recurrent Unit), che migliorano la capacità della rete di apprendere e mantenere informazioni su lunghe sequenze temporali.
- **Reti Generative Adversariali (GAN):** Le GAN rappresentano un'architettura avanzata che combina due reti, una generativa e una discriminativa, in un quadro competitivo. Queste reti sono in grado di generare nuovi dati simili a quelli reali, estendendo le capacità delle MLP in modi creativi e innovativi. La rete generativa tenta di produrre dati falsi che siano indistinguibili dai dati reali, mentre la rete discriminativa cerca di distinguere tra dati reali e falsi. Questo approccio ha portato a notevoli innovazioni nella generazione di immagini realistiche, video, musica e persino testo, aprendo nuove possibilità nel campo della creatività artificiale e della simulazione.

3.4.3.3 Tecniche di Addestramento per Reti Profonde

L'addestramento delle reti profonde è più complesso rispetto a quello delle MLP a causa della maggiore profondità e del numero di parametri coinvolti. Il processo di addestramento utilizza algoritmi di ottimizzazione come la discesa del gradiente, ma con alcune sfide specifiche:

- **Problema del Vanishing Gradient:** Nelle reti molto profonde, i gradienti calcolati durante la backpropagation possono diventare molto piccoli, impedendo l'aggiornamento efficace dei pesi nei primi strati della rete. Questo problema è particolarmente critico nelle RNN, dove la propagazione dei gradienti attraverso molteplici passi temporali può portare alla perdita di informazioni utili. Per mitigare questo problema, si utilizzano funzioni di attivazione come ReLU, che mantengono gradienti più ampi, e tecniche come il batch normalization, che stabilizza e accelera il processo di addestramento.
- **Batch Normalization:** Questa tecnica normalizza gli input a ciascuno strato per avere una media zero e una varianza unitaria, riducendo così il rischio di gradienti esplosivi o vanishing e migliorando la stabilità dell'addestramento. Il batch normalization è ampiamente utilizzato nelle reti profonde, poiché permette un addestramento più efficiente e riduce la sensibilità agli iperparametri, facilitando l'uso di learning rate più elevati.
- **Dropout:** Per prevenire l'overfitting, una delle tecniche più comuni è il dropout, che consiste nel disattivare casualmente alcuni neuroni durante l'addestramento, impedendo alla rete di dipendere troppo da specifiche connessioni. Questo forza la rete a generalizzare meglio, migliorando le sue prestazioni su dati mai visti. Durante la fase di inferenza, tutti i neuroni vengono utilizzati, ma i pesi sono scalati per mantenere la coerenza delle attivazioni.

3.4.3.4 Tecniche di Ottimizzazione dei Parametri delle Reti Profonde

Oltre alle tecniche di addestramento, le reti profonde richiedono l'uso di tecniche avanzate di ottimizzazione per gestire la complessità e migliorare la convergenza:

- **Algoritmi di Ottimizzazione:** Sebbene la discesa del gradiente stocastica (SGD) sia l'approccio di base, varianti più avanzate come Adam (Adaptive Moment Estimation) e RMSprop sono ampiamente utilizzate. Adam, in particolare, combina i vantaggi di AdaGrad (che adatta il learning rate per ogni parametro) e RMSprop (che mantiene un learning rate efficiente per ogni parametro), risultando in una convergenza più rapida e stabile anche in reti molto profonde.
- **Learning Rate Scheduling:** Il learning rate, ossia la velocità con cui vengono aggiornati i pesi, è un parametro critico che influisce sulla velocità e sull'efficacia dell'addestramento. Tecniche come il learning rate scheduling permettono di iniziare l'addestramento con un learning rate elevato, che viene ridotto man mano che il modello

si avvicina a una soluzione ottimale. Questo aiuta a trovare il minimo globale della funzione di perdita più rapidamente.

- **Early Stopping:** Per evitare l'overfitting durante l'addestramento, l'early stopping monitora la performance del modello su un set di validazione e interrompe l'addestramento quando le prestazioni iniziano a peggiorare. Questo evita che la rete apprenda troppo i dettagli del set di addestramento, migliorando la generalizzazione.
-

3.4.3.5 Uso di modelli pre-addestrati

L'addestramento delle reti neurali profonde richiede notevoli risorse computazionali e dataset di grandi dimensioni, rendendo i costi in termini di tempo e potenza di calcolo molto elevati. Per ridurre questi costi, l'uso di modelli pre-addestrati rappresenta una soluzione efficace, poiché consente di sfruttare reti già addestrate su ampi dataset e adattarle a specifici problemi con un processo noto come fine-tuning. Ciò permette di evitare il lungo e dispendioso processo di addestramento da zero, ottenendo comunque prestazioni eccellenti.

Le collezioni di modelli pre-addestrati disponibili su piattaforme come [TensorFlow Hub](#), [PyTorch Hub](#) e [Hugging Face Model Hub](#) offrono reti avanzate, già ottimizzate, come ResNet, EfficientNet per la visione e BERT, GPT per il linguaggio. Questi modelli, addestrati su dataset estesi, possono essere facilmente utilizzati per applicazioni specifiche con poche risorse computazionali aggiuntive, rendendo il processo più accessibile ed economico senza sacrificare la qualità delle prestazioni.

esempio di uso di rete pre-addestrata

Usiamo una rete pre-addestrata per il compito di classificazione di immagini che abbiamo visto nel paragrafo 4.4.3.2. Per utilizzare una rete preaddestrata con il dataset CIFAR-10, possiamo sfruttare un modello preaddestrato su ImageNet, come ResNet50, e adattarlo al nostro compito tramite il fine-tuning. L'idea è quella di caricare il modello preaddestrato, congelare i pesi degli strati inferiori e modificare solo gli ultimi strati per adattare il modello alle 10 classi di CIFAR-10.

Ecco un esempio di fine tuning di una rete pre-addestrata utilizzando Keras e TensorFlow.

```
import tensorflow as tf
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Dropout, GlobalAveragePooling2D
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

```

# Carica e prepara il dataset CIFAR-10
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0
y_train = tf.keras.utils.to_categorical(y_train, 10)
y_test = tf.keras.utils.to_categorical(y_test, 10)

# Data augmentation
datagen = ImageDataGenerator(
    rotation_range=15,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True,
    zoom_range=0.1
)
datagen.fit(x_train)

# Carica il modello VGG16 preaddestrato senza i layer fully connected
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(32, 32, 3))

# Fine-tuning: sblocca gli ultimi blocchi convoluzionali
for layer in base_model.layers[-4:]:
    layer.trainable = True

# Crea un nuovo modello aggiungendo layer personalizzati
model = Sequential([
    base_model,
    GlobalAveragePooling2D(),
    Dense(512, activation='relu'),
    Dropout(0.5),
    Dense(256, activation='relu'),
    Dropout(0.5),
    Dense(10, activation='softmax')
])

# Compila il modello
model.compile(optimizer=Adam(learning_rate=0.0001),
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Addestra il modello
history = model.fit(datagen.flow(x_train, y_train, batch_size=16),

```

```

        steps_per_epoch=len(x_train) // 16,
        epochs=5,
        validation_data=(x_test, y_test),
        verbose=1)

# Valuta il modello sul set di test
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=0)
print(f"Accuratezza sul set di test: {test_acc:.4f}")

import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import confusion_matrix
import seaborn as sns

# Supponiamo che 'history', 'model', 'x_test', 'y_test' siano già definiti dal codice precedente

# Nomi delle classi CIFAR-10
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']

# 1. Grafico dell'accuratezza
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

# 2. Grafico della loss
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.tight_layout()
plt.show()

# 3. Matrice di confusione
y_pred = model.predict(x_test)

```

```

y_pred_classes = np.argmax(y_pred, axis=1)
y_true_classes = np.argmax(y_test, axis=1)

cm = confusion_matrix(y_true_classes, y_pred_classes)
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=class_names, yticklabels=class_names)
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()

# 4. Visualizzazione di 20 esempi di immagini classificate
n_images = 20
indices = np.random.choice(range(len(x_test)), n_images, replace=False)

plt.figure(figsize=(20, 10))
for i, idx in enumerate(indices):
    plt.subplot(4, 5, i+1)
    img = x_test[idx]
    plt.imshow(img)
    true_label = class_names[y_true_classes[idx]]
    pred_label = class_names[y_pred_classes[idx]]
    color = 'green' if true_label == pred_label else 'red'
    plt.title(f"True: {true_label}\nPred: {pred_label}", color=color)
    plt.axis('off')
plt.tight_layout()
plt.show()

# Stampa l'accuratezza complessiva
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=0)
print(f"Accuratezza complessiva sul set di test: {test_acc:.4f}")

```

3.4.3.6 Applicazioni e Sfide

Le applicazioni del deep learning sono vaste e coprono molte aree, dalla visione artificiale all'elaborazione del linguaggio naturale. In ambito giuridico, le reti profonde possono essere utilizzate per l'analisi predittiva, la classificazione automatica di documenti legali e l'estrazione di informazioni da grandi volumi di testo. Tuttavia, l'implementazione del deep learning richiede una grande quantità di dati e risorse computazionali, oltre a una profonda comprensione delle reti neurali per evitare problemi di interpretabilità e bias. Nonostante queste sfide, il deep

learning continua a spingere i confini dell'intelligenza artificiale, offrendo soluzioni avanzate a problemi complessi che erano precedentemente irrisolvibili.

Machine learning e linguaggi naturali

A Elementi di Python

Python è un linguaggio di programmazione versatile e potente, ampiamente utilizzato in diversi ambiti, inclusi quelli legali e accademici. La sua sintassi semplice e leggibile lo rende ideale per chiunque si avvicini alla programmazione per la prima volta (Lutz 2013; Rossum and Drake 2016).

In questo capitolo, imparerai le basi di Python, con esempi pratici tratti dal mondo giuridico, come la gestione di fascicoli, il calcolo delle pene e l'elaborazione di dati legali.

A.1 Perché Python è utile in ambito giuridico?

Python è particolarmente utile per automatizzare compiti ripetitivi, analizzare grandi quantità di dati (ad esempio, sentenze giudiziarie) e sviluppare strumenti personalizzati per supportare il lavoro legale. Con poche righe di codice, è possibile:

- Analizzare documenti legali.
- Calcolare sanzioni o durate delle pene.
- Automatizzare la creazione di report (Srinath 2017).

A.2 Introduzione alla sintassi di Python

A.2.1 1. Variabili e Tipi di Dati

Le **variabili** sono contenitori per dati. Ad esempio, puoi salvare il nome di un giudice o il numero di un fascicolo:

```
giudice = "Mario Rossi" # Nome del giudice
fascicolo = 2024112345 # Numero del fascicolo
```

Python riconosce automaticamente il tipo di dato (stringhe, numeri, ecc.). Non serve dichiararlo (Rossum and Drake 2016).

A.2.2 2. Condizioni: Prendere Decisioni

Con Python, puoi verificare condizioni e decidere quale azione intraprendere. Ad esempio, puoi controllare se un fascicolo è chiuso:

```
fascicolo_chiuso = False # Il fascicolo non è ancora chiuso

if fascicolo_chiuso:
    print("Il fascicolo è chiuso.")
else:
    print("Il fascicolo è ancora aperto.")
```

Nota: Il codice è sensibile all'indentazione (spazi all'inizio di una riga). Assicurati di rispettare la struttura (Lutz 2013)!

A.2.3 3. Cicli: Ripetere Azioni

I **cicli** permettono di eseguire azioni ripetitive. Ad esempio, puoi stampare l'elenco di testimoni da interrogare:

```
testimoni = ["Testimone A", "Testimone B", "Testimone C"]

for testimone in testimoni:
    print(f"Interrogare {testimone}")
```

Output:

```
Interrogare Testimone A
Interrogare Testimone B
Interrogare Testimone C
```

A.2.4 4. Gestione degli Errori

Python permette di gestire gli errori con eleganza. Ad esempio, se provi ad aprire un file di sentenze che non esiste:

```
try:
    with open("sentenze.txt", "r") as file:
        contenuto = file.read()
except FileNotFoundError:
    print("Errore: Il file non è stato trovato.")
```

Output:

Errore: Il file non è stato trovato.

A.2.5 5. Strutture Dati Utili

A.2.5.1 Liste

Le **liste** sono utili per memorizzare più elementi. Ad esempio, un elenco di articoli di legge:

```
articoli = ["Art. 1", "Art. 2", "Art. 3"]
articoli.append("Art. 4") # Aggiungi un articolo
print(articoli)
```

A.2.5.2 Dizionari

I **dizionari** memorizzano dati in coppie chiave-valore. Ad esempio, puoi salvare i dettagli di un fascicolo:

```
fascicolo = {
    "numero": "2024/12345",
    "giudice": "Mario Rossi",
    "stato": "aperto"
}
print(fascicolo["giudice"]) # Stampa: Mario Rossi
```

A.3 Applicazioni Pratiche

Con Python, puoi creare funzioni per automatizzare calcoli giuridici. Ad esempio:

```
def calcola_pena(reato_grave):
    if reato_grave:
        return 10 # Anni di reclusione per reato grave
    else:
        return 2 # Anni per reati meno gravi

pena = calcola_pena(True)
print(f"Anni di reclusione: {pena}")
```

Output:

Anni di reclusione: 10

A.3.1 Creazione di Moduli Personalizzati

Puoi creare file Python con funzioni personalizzate, utili per compiti specifici. Ad esempio, un modulo `calcoli_legali.py` potrebbe contenere:

```
def calcola_sanzione(sanzione_base, giorni_ritardo, mora=50):  
    return sanzione_base + giorni_ritardo * mora
```

E poi importarlo nel tuo programma:

```
import calcoli_legali  
  
sanzione = calcoli_legali.calcola_sanzione(500, 10)  
print(f"La sanzione totale è di {sanzione} euro.")
```

A.4 Conclusione

Con una conoscenza di base di Python, puoi affrontare problemi complessi in modo semplice ed efficiente, rendendo il tuo lavoro legale più rapido e preciso (Foundation 2024).

Foundation, Python Software. 2024. “Python Official Documentation.” <https://docs.python.org/3/>.

Jiang, Feng, Yong Jiang, Hui Zhi, Dong Yang, Hua Li, Shanyu Ma, and Yan Wang. 2017. “Artificial Intelligence in Healthcare: Past, Present and Future.” *Stroke and Vascular Neurology*.

Lutz, Mark. 2013. *Learning Python*. 5th ed. O’Reilly Media.

McCarthy, John, Marvin Minsky, Nathaniel Rochester, and Claude E. Shannon. 1955. *A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence*.

Rossum, Guido Van, and Fred L. Drake. 2016. *The Python Language Reference Manual*. Python Software Foundation. <https://docs.python.org/3/reference/>.

Russell, Stuart, and Peter Norvig. 2016. *Artificial Intelligence: A Modern Approach*. Pearson.

Searle, John R. 1980. “Minds, Brains, and Programs.” *Behavioral and Brain Sciences*.

Srinath, R. 2017. “Applications of Python Programming in Legal Practice.” *Journal of Legal Technology* 12: 34–45.

Turing, Alan M. 1950. “Computing Machinery and Intelligence.” *Mind*.